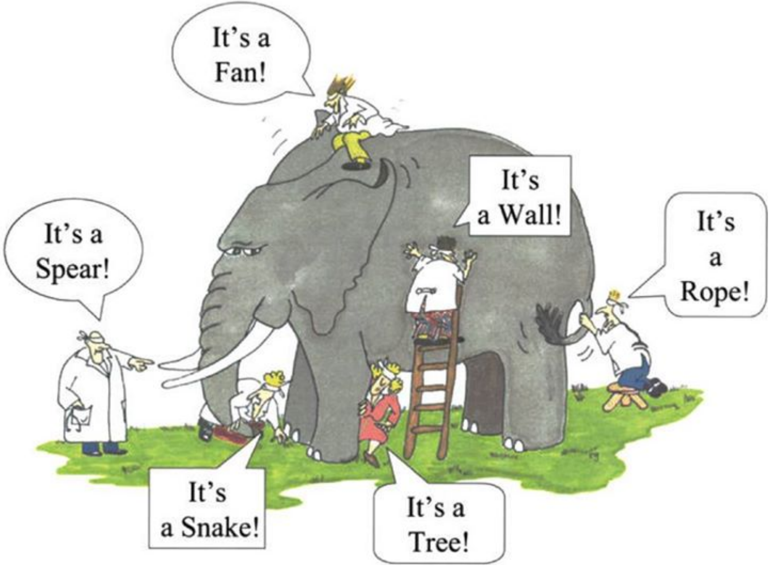
Introduction to Monte Carlo

An integration example

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.



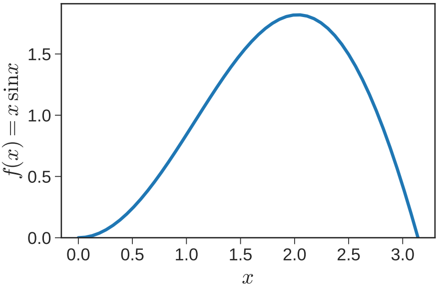
Monte Carlo is different things to different people

Contents

Example

Consider the particular 1D integral

$$I = \int_0^{\pi} x \sin x \; dx.$$



$$I = \int_0^{\pi} x \sin x \; dx = -x\cos(x) + \sin(x)\big|_0^{\pi} = \pi.$$

Average Value and Integral

The average value $\bar{f}$ is the mean "height" of the function.

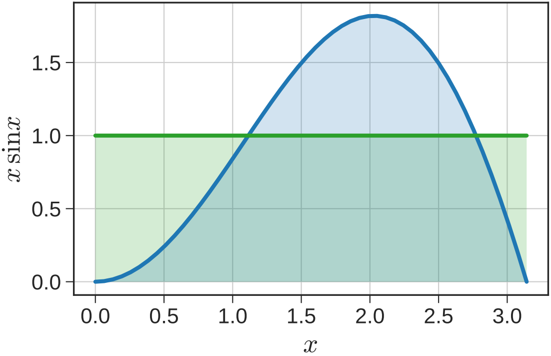
$$\begin{aligned} \bar{f} &= \frac{\int_a^b f(x)dx}{\int_a^b dx} \\ &= \frac{\int_a^b f(x)dx}{b-a} \\ &= \frac{I}{b-a} \end{aligned}$$

Therefore, integral $=$ average $\times$ domain size

$$I = \bar{f}(b-a)$$

In our example, $\bar{f} = 1$. Hence, $I = 1(\pi - 0) = \pi$.

The areas under the blue and green curves are equal.



Insight: integral $\leftrightarrow$ average value of the integrand

Average Value

Method 1:

- pick n uniformly spaced points x_i in the domain
- evaluate the average:

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

Method 2:

- select n randomly chosen points x_i in the domain
- evaluate the average:

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

Computing Average Value

```
def equispacedAverage(npts):
    xi = np.linspace(0, np.pi, npts)
    fi = xi * np.sin(xi)
    return np.mean(fi)

def randomAverage(npts):
    xi = np.random.uniform(0, np.pi, npts)
    fi = xi * np.sin(xi)
    return np.mean(fi)
```

Let us test the two subroutines.

Recall that the true $\bar{f} = 1$.

Computing Average Value

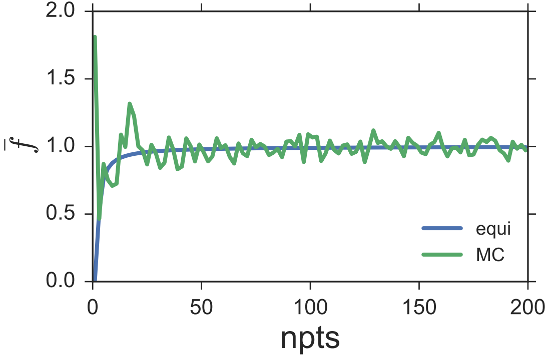
```
npts = 10; print(equispacedAverage(npts))
0.890842865047
```

```
npts = 10; print(randomAverage(npts))
0.812445848056
```

Reasonable enough.

Let us now systematically vary `npts` between 1 and 200.

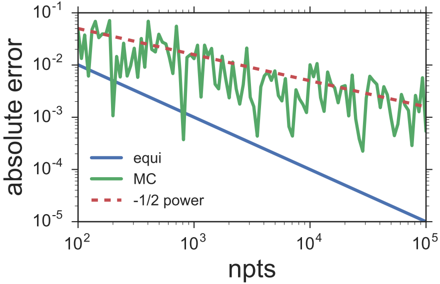
Equispaced versus Random



- As `npts` $\uparrow$, both estimates "converge" to $\bar{f} = 1$ (why?)
- Method 1 converges more "systematically" (why?)
- Random requires more work (generate random numbers), and produces an inferior answer.

Convergence in 1D

Consider the absolute error $\epsilon = |\bar{f}_{\text{true}} - \bar{f}_{\text{est}}|$,



Convergence in general,

$$\epsilon_{\text{equi}} \sim \frac{1}{n}; \epsilon_{\text{mc}} \sim \frac{1}{\sqrt{n}}$$

Insight: For 1D integrals, MC is a bad idea!

2D Integrals: "integration by darts"

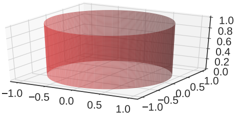
Classic problem: area of a circle

Mathematically,

$$I = \int_{-1}^{1} \int_{-1}^{1} g(x_1, x_2) dx_1 dx_2,$$

where,

$$g(x_1, x_2) = \begin{cases} 1, & \text{if } x_1^2 + x_2^2 \leq 1 \\ 0, & \text{otherwise.} \end{cases}$$



$$\text{integral} = \text{average} \times \text{domain size}$$

$$I = \bar{g} \times (2 \times 2) = 4\bar{g}$$

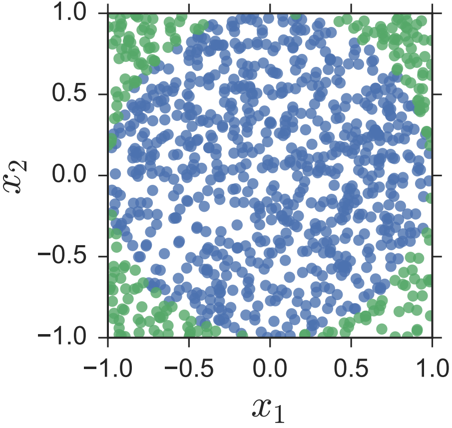
Note: True $I = \pi; \bar{g} = \pi/4$

Python

```python
def MCdarts(npts):

    """input : #darts,
       output: average value of g"""

    x1 = np.random.uniform(-1,1,size=npts)
    x2 = np.random.uniform(-1,1,size=npts)

    cond = x1**2 + x2**2 <= 1

    return np.sum(cond)/float(npts)
```

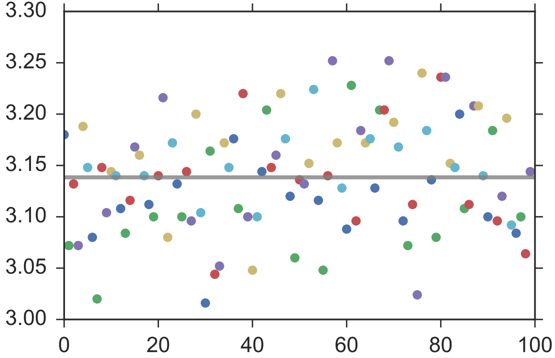
Let us throw 1000 darts:

```
print("area", 4*MCdarts(1000))
area 3.148
```



Variability

Repeat experiment of throwing 1000 darts many (=100) times:



Insight: need to account for variability when analyzing MC

Equispaced Analog

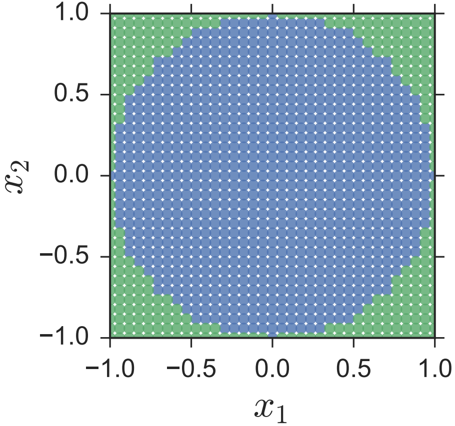
Now, let us throw darts systematically on a "grid"

```
def equiDarts(npts, isPlot=False):
    """total number of points are npts = ngrid*ngrid"""

    ngrid = int(np.sqrt(npts))

    x = np.linspace(-1, 1, ngrid)
    x1, x2 = np.meshgrid(x,x)

    cond = x1**2 + x2**2 <= 1

    y = np.sum(cond,axis=0)
    y = np.sum(y)

    return y/float(npts)
```

Darts on a Regular Grid

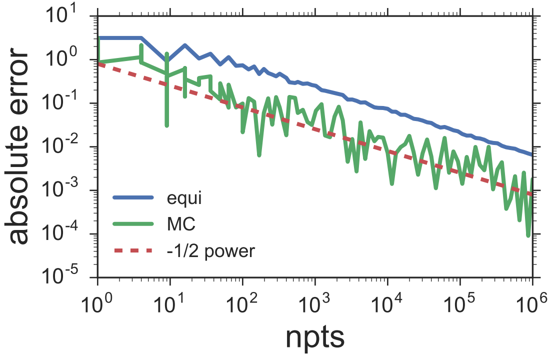
Use 1000 darts ($\sqrt{1000} \approx 35$).

```
print("area", 4*equiDarts(1000))
area 2.94204081633
```



This is worse than most of the MC answers.

Convergence



1. The error in both methods $\epsilon \sim n^{-1/2}$.
2. ϵ_{equi} no longer decays smoothly, especially initially
3. ϵ_{mc} is lower than ϵ_{equi}

Curse of Dimensionality

In fact, it can be shown that,

$$\epsilon_{\mathsf{mc}} \sim n^{-1/2}, \quad \text{independent of dimension.}$$

For *any* systematic "quadrature" method,[1]

$$\epsilon_{\mathsf{quad}} \sim n^{-p_{1D}/d},$$

where p_{1D} is the convergence of the method for 1D problems.

This is called the curse of dimensionality.

Many important computational problems are high-dimensional.

[1] not just equispaced; rectangle, trapezoidal, Simpson's have $p_{1D} = 1$, 2, and 4, respectively.

Summary

- integral = average $\times$ domain size;
- For 1D averages, MC is a bad idea;
- For higher dimensional integrals, standard quadrature methods suffer from the curse of dimensionality;
- For higher dimensional integrals, the error in MC decreases as the square root of the number of samples due to the central limit theorem;
- estimates from MC are noisy