算法分析和複雜性理論

干皓丞，2101212850, 信息工程學院

2022 年 6 月 15 日

1　作業目標與章節摘要

蒙地卡羅方法求圓周率 PI。

2　作業內容概述

作業可以從 GitHub 下的 kancheng/kan-cs-report-in-2022 專案找到，作業程式碼與文件目錄為 kan-cs-report-in-2022/AATCC/lab-report/。實際執行的環境與實驗設備為 Google 的 Colab 、MacBook Pro (Retina, 15-inch, Mid 2014) 、Acer Aspire R7 與 HP Victus (Nvidia GeForce RTX 3060)。

本作業 GitHub 專案為 kancheng/kan-cs-report-in-2022 下的 AATCC‘ 的目錄。程式碼可以從 code 目錄下可以找到 *.pynb，內容包含上次課堂練習、LeetCode 範例思路整理與作業。

https://github.com/kancheng/kan-cs-report-in-2022/tree/main/AATCC

Fig. 1. 作業專案位置

1. LeetCode : https://leetcode.com/
2. LeetCode CN : https://leetcode-cn.com/
3. OnlineGDB : https://www.onlinegdb.com/

LeetCode 的平台部分，CN 的平台有針對簡體中文使用者進行處理，包含中英文切換等功能。OnlineGDB 則可線上進行簡易的環境測試，其程式碼涵蓋 C, C++, C#, Java, Python, JS, Rust, Go。

3 作業推導

Fig. 2. Java Demo

3.1 Java 範例

```
1 import javax.swing.*;
2 import javax.swing.event.*;
3 import java.awt.*;
4 import java.awt.event.*;
```

```
5  import java.util.Random;
6  public class Mondecaro extends JFrame implements ActionListener {
7      private JButton button;
8      private JPanel panel;
9      private int height = 300, width = 300;
10     public static void main (String[] argv) {
11         Mondecaro frame = new Mondecaro();
12         frame.setSize(520, 350);
13         frame.createGUI();
14         frame.setVisible(true);
15     }
16     private void createGUI() {
17         setDefaultCloseOperation(EXIT_ON_CLOSE);
18         Container window = getContentPane();
19         window.setLayout(new FlowLayout());
20         panel = new JPanel();
21         panel.setPreferredSize(new Dimension(width, height));
22         panel.setBackground(Color.white);
23         window.add(panel);
24         button = new JButton("Mondecaro");
25         window.add(button);
26         button.addActionListener(this);
27     }
28     public void actionPerformed(ActionEvent e) {
29         draw();
30     }
31     private void draw() {
32         Graphics paper = panel.getGraphics();
33         paper.setColor(Color.white);
34         paper.fillRect( 0, 0, width, height);
35         paper.setColor(Color.black);
36         int px = width, py = height / 2;
37         Random random = new Random ();
38         int n = width / 2, in = 0, N = 10000;
39         for (int i = 1; i <= N; i++) {
40             int x = random.nextInt(width);
41             int y = random.nextInt(width);
42             double dist = Math.sqrt(Math.pow(x - n, 2) + Math.pow(y - n, 2));
43             if (dist < n) {
44                 paper.setColor(Color.red);
45                 in++;
46             } else {
47                 paper.setColor(Color.blue);
48             }
49             paper.drawOval( x, y, 1, 1);
50             setTitle("Area = " + (double)in / i * 4);
51             for (int k = 0; k < 1000; k++);
```

```
        }
    }
}
```

```
import random
total = [10, 100, 1000, 10000, 100000, 1000000, 5000000]  #隨機點數
for t in total:
    in_count = 0
    for i in range(t):
        x = random.random()
        y = random.random()
        dis = (x**2 + y**2)**0.5
        if dis<=1:
            in_count += 1
    print(t,'個隨機點時，  是：', 4 * in_count/t)
```

結果如下:

```
10 個隨機點時，  是： 2.8
100 個隨機點時，  是： 3.0
1000 個隨機點時，  是： 3.044
10000 個隨機點時，  是： 3.1232
100000 個隨機點時，  是： 3.15028
1000000 個隨機點時，  是： 3.141328
5000000 個隨機點時，  是： 3.1407488
```

3.2 概述

蒙特卡羅方法是一種計算方法。原理是通過大量隨機樣本，去了解一個系統，進而得到所要計算的值。

它非常強大和靈活，又相當簡單易懂，很容易實現。對於許多問題來說，它往往是最簡單的計算方法，有時甚至是唯一可行的方法。它誕生於上個世紀 40 年代美國的" 曼哈頓計劃"，名字來源於賭城蒙特卡羅，象徵概率。

3.3 蒙特卡羅方法的應用

通常蒙特卡羅方法可以粗略地分成兩類：

一類是所求解的問題本身俱有內在的隨機性，借助計算機的運算能力可以直接模擬這種隨機的過程。例如在核物理研究中，分析中子在反應堆中的傳輸過程。中子與原子核作用受到量子力學規律的製約，人們只能知道它們相互作用發生的概率，卻無法準確獲得中子與原子核作用時的位置以及裂變產生的新中子的行進速率和方向。科學家依據其概率進行隨機抽樣得到裂變位置、速度和方向，這樣模擬大量中子的行為後，經過統計就能獲得中子傳輸的範圍，作為反應堆設計的依據。

另一種類型是所求解問題可以轉化為某種隨機分佈的特徵數，比如隨機事件出現的概率，或者隨機變量的期望值。通過隨機抽樣的方法，以隨機事件出現的頻率估計其概率，或者以抽樣的數字特徵估算隨機變量的數字特徵，並將其作為問題的解。這種方法多用於求解複雜的多維積分問題。

3.4 的計算

第一個例子是，如何用蒙特卡羅方法計算圓周率 。正方形內部有一個相切的圓，它們的面積之比是 /4

現在，在這個正方形內部，隨機產生 10000 個點（即 10000 個坐標對 (x, y)），計算它們與中心點的距離，從而判斷是否落在圓的內部。

如果這些點均勻分佈，那麼圓內的點應該占到所有點的 /4，因此將這個比值乘以 4，就是 的值。通過 R 語言腳本隨機模擬 30000 個點， 的估算值與真實值相差 0.07%。

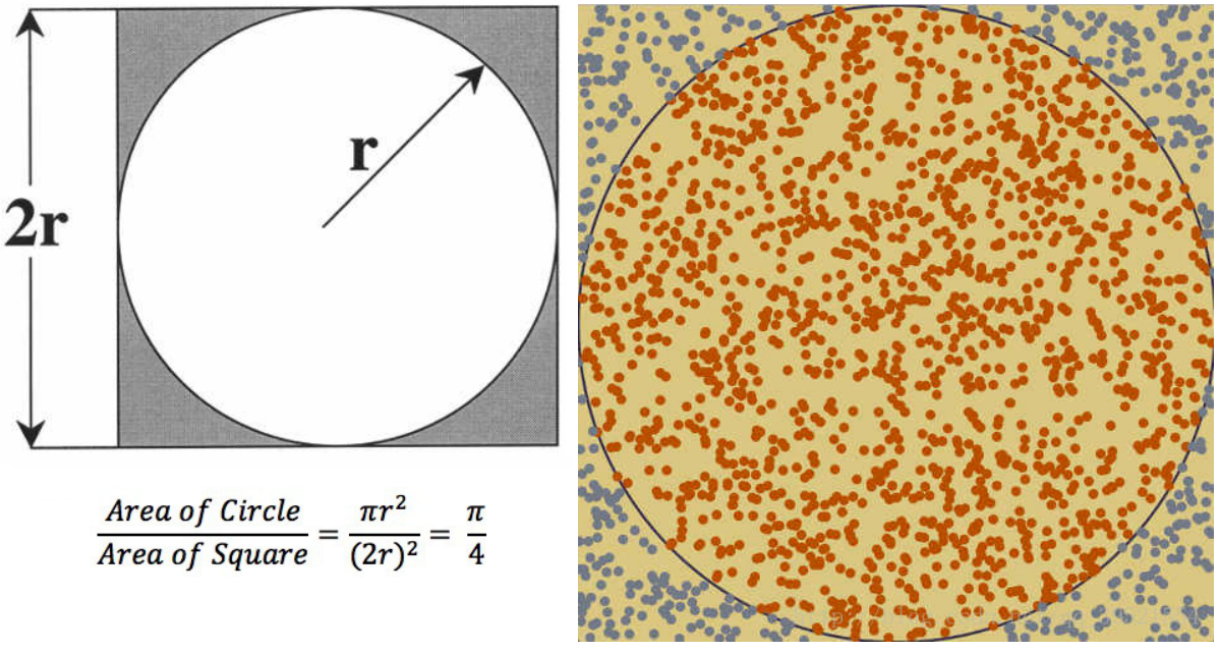

Fig. 3. PI

```python
import numpy as np
import tqdm
#random_generate = np.random.uniform(low=0.0, high=2.0, size=(1, 1))

#求解 pi
sum = 0
for i in tqdm.tqdm(range(3000000)):
        #random_generate = np.random.rand(2
        random_generate = np.random.uniform(low=0.0, high=2.0, size=(2))
        if np.sum(np.square(random_generate-np.array([1.0, 1.0]))) <=1:
                sum += 1
print(sum)
pi = 4 * (sum / 3000000)
print('pi is:{}'.format(pi))
```

3.5 蒙特卡羅方法求定積分

比如積分 $\theta = \int_a^b f(x)dx$ ，如果 f(x) 的原函數很難求解，那麼這個積分也會很難求解。

而通過蒙特卡羅方法對其進行模擬求解的方式有二。

1. 隨機投點法

這個方法和上面的兩個例子的方法是相同的。如圖所示，有一個函數 f(x)，要求它從 a 到 b 的定積分，其實就是求曲線下方的面積：

這時可以用一個比較容易算得面積的矩型罩在函數的積分區間上（假設其面積為 Area），然後隨機地向這個矩形框裡面投點，其中落在函數 f(x) 下方的點為綠色，其它點為紅色，然後統計綠色點的數量佔所有點（紅色 + 綠色）數量的比例為 r，那麼就可以據此估算出函數 f(x) 從 a 到 b 的定積分為 Area × r。

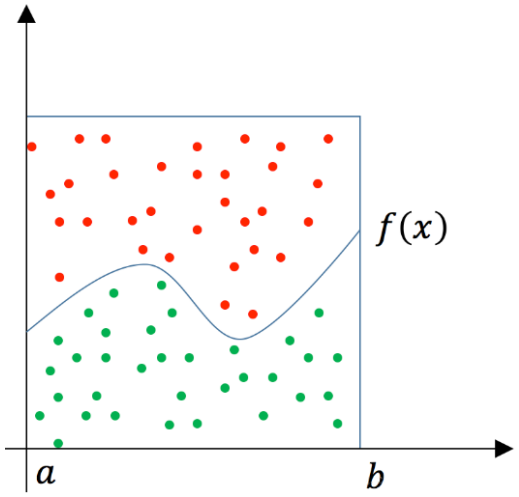

Fig. 4. 隨機投點法

```python
# 求解定積分 x^2 區間[1, 2]; 投點法
import numpy as np
import tqdm
sum = 0
for i in tqdm.tqdm(range(3000000)):

        random_generate = np.array([np.random.uniform(1, 2), np.random.uniform
            (0, 4)])
        if np.square(random_generate[0]) > random_generate[1]:
                sum += 1
print(sum)
area = 4 * sum / 3000000
print('Area is:{}'.format(area))
```

2. 平均值法 (期望法)

如下圖所示，在 [a,b] 之間隨機取一點 x 時，它對應的函數值就是 f(x)，我們要計算 $\theta = \int_a^b f(x)dx$，就是圖中陰影部分的面積。

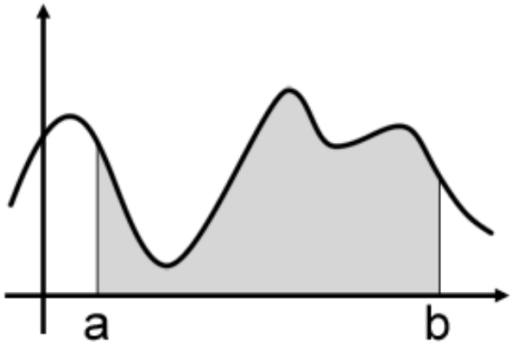

Fig. 5. 陰影部分的面積

一個簡單的近似求解方法就是用 $f(x)*(b-a)$ 來粗略估計曲線下方的面積，在 [a,b] 之間隨機取點 x，用 f(x) 代表在 [a,b] 上所有 f(x) 的值，如下圖所示：

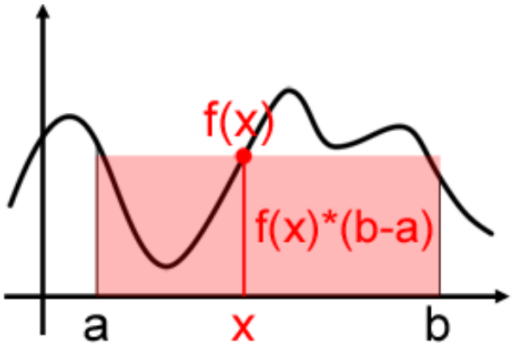

Fig. 6. 近似求解方法

用一個值代表 [a,b] 區間上所有的 ( ) 的值太粗糙了，我們可以進一步抽樣更多的點，比如下圖抽樣了四個隨機樣本 x_1, x_2, x_3, x_4 (滿足均勻分佈)，每個樣本都能求出一個近似面積值 $f(x_i)*(b-a)$ ，然後計算他們的數學期望，就是蒙特卡羅計算積分的平均值法了。

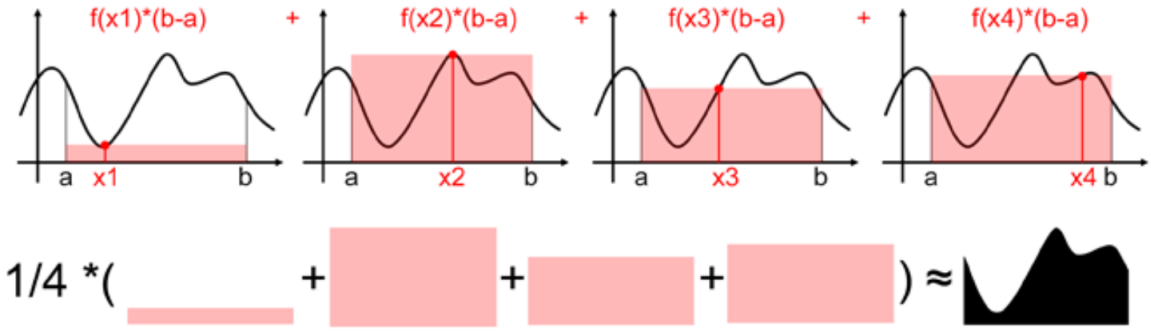

Fig. 7. 近似求解方法

用數學公式表述上述過程：

$S = \frac{1}{4}[f(x_1)(b-a)+f(x_2)(b-a)++f(x_3)(b-a)++f(x_4)(b-a)] = \frac{1}{4}(b-a)(f(x_1)+f(x_2)+f(x_3)+f(x_4)) = \frac{1}{4}(b-a)\sum_{i=1}^{4} f(x_i)$

然後進一步我們採樣 n 個隨機樣本 (滿足均勻分佈)，則有：

$S = \frac{b-a}{n}\sum_{i=1}^{n} f(x_i) \simeq \theta$

採樣點越多，估計值也就越來越接近。

上面的方法是假定 x 在 [a,b] 間是均勻分佈的，而大多時候 x 在 [a,b] 上不是均勻分佈的，因此上面方法就會存在很大的誤差。

這時我們假設 x 在 [a,b] 上的概率密度函數為 $p(x)$ ，加入到 $\theta = \int_a^b f(x)dx$ 中變換為：

$\theta = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{p(x)}p(x)dx \simeq \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{p(x_i)}$

這就是蒙特卡羅期望法計算積分的一般形式。那麼問題就換成瞭如何從 $p(x)$ 中進行採樣。

```
# 求解定積分 X^2 區間 [1， 2]; 平均法
import numpy as np
import tqdm
sum = 0
for i in tqdm.tqdm(range(3000000)):
        random_x = np.random.uniform(1, 2, size=None)
        # None 是默認的也可以不寫
```

```
        a =  np.square(random_x)
        sum += a*(2-1)
area = sum/3000000
print('calculate by mean_average:{} '.format(area))
```

3.6 蒙特卡洛法高維數值積分 Vegas

高能物理研究經常要用到高維函數的數值積分。傳統的數值積分方法，比如梯形公式，辛普森積分，Gauss Quadrature 已經統統失效。原因很簡單，那些算法每個維度需要至少 M 個離散的數據點，對於 N 維積分，需要在 M^N 個點上計算函數取值。

比如 10 維積分，每個維度用最節省的 15 個點的 Gauss Quadrature，需要計算的函數值的次數也達到了 $M^N = 15^{10} = 576650390625$ 約 5766 億次。

出現這種情況一般稱作維數災難。在此使用蒙特卡洛積分算法 Vegas 做高維數值積分，而 Python 的 Vegas 庫的安裝以及 Vegas 蒙卡⊠分的原理如下。

3.7 VEGAS 高維函數蒙特卡洛積分

安裝 vegas 庫很簡單，在命令行使用如下命令

```
pip install vegas
```

計算如下 4 維高斯函數

$f(x_0, x_1, x_2, x_3) = Nexp(-\sum_{i=0}^{3}(x_i - \frac{1}{2})^2/0.01)$

在閉區間 $x_0 \in [-1, 1], x_1 \in [0, 1], x_2 \in [0, 1], x_3 \in [0, 1]$

上的數值積分。其中 N = 1013.211 是一個歸一化因子。

```
# copy and paste to test.py
import vegas
import math

def f(x):
    dx2 = 0
    for d in range(4):
        dx2 += (x[d] - 0.5) ** 2
    return math.exp(-dx2 * 100.) * 1013.2118364296088

integ = vegas.Integrator([[-1, 1], [0, 1], [0, 1], [0, 1]])

result = integ(f, nitn=10, neval=1000)
print(result.summary())
print('result = %s     Q = %.2f' % (result, result.Q))
```

```
itn   integral        wgt average     chi2/dof        Q
-------------------------------------------------------
  1   1.04(69)        1.04(69)            0.00     1.00
  2   0.83(31)        0.87(28)            0.08     0.78
  3   1.10(25)        1.00(19)            0.24     0.79
  4   0.922(74)       0.932(69)           0.21     0.89
  5   1.052(62)       0.999(46)           0.58     0.68
```

```
  6    0.965(27)          0.974(23)          0.54      0.74
  7    0.996(21)          0.986(16)          0.53      0.78
  8    0.982(17)          0.984(12)          0.46      0.86
  9    1.002(15)          0.9910(92)         0.51      0.85
 10    0.998(12)          0.9934(74)         0.48      0.89

result = 0.9934(74)     Q = 0.89
```

可以看到 result.summary() 返回了 10 次迭代的結果。
如果只想返回最終結果，去掉 summary(),
EX : result = 1.0101(93)
result 是一個 gvar 類型的數，括號裡的數表示誤差
- 使用 result.mean 返回均值 1.0101
- 使用 result.sdev 返回不確定部分 0.0093
如果積分變量 x[1] 的積分上限依賴積分變量 x[0], 如
$\int_0^1 dx_0 \int_0^{x_0} dx_1 \sin(x_0 x_1)$
被積函數可以這樣寫

```
def f(x):
    if x[1] > x[0]: return 0
    return sin(x[0]*x[1])
```

3.8 蒙特卡洛積分 - 重要抽樣法

這裡忽略黎曼積分的適用性以及勒貝格積分的優越性討論，來自 Mathematica 關於黎曼求和、黎曼積分的例子

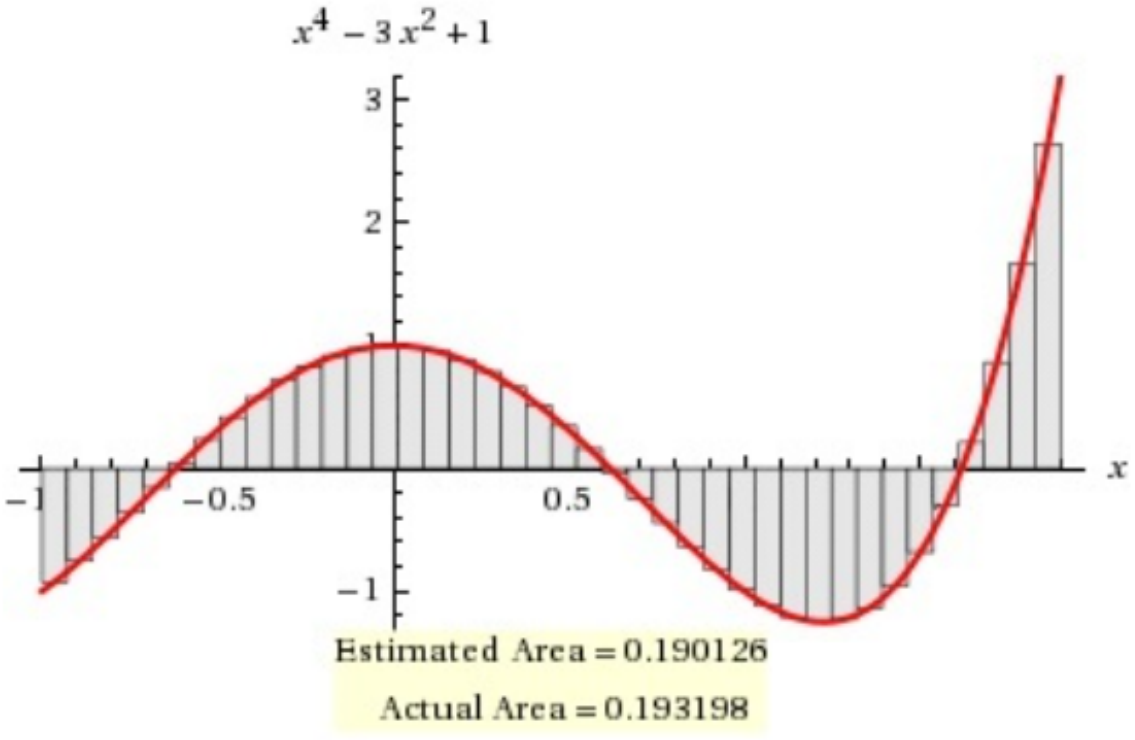

Fig. 8. Mathematica 關於黎曼求和、黎曼積分的例子

黎曼積分原理指導我們，為了求一維函數 f(x) 在閉區間 [a, b] 上的定積分，可以先把區間分成 n 份，$a < x_1 < x_2 < \cdots < x_{n-1} < b$, 其中格子大小為 $\Delta x_i = x_{i+1} - x_i$

函數的積分近似等於小格子中任一處的函數值 $f\left(x_i^*\right)$ 乘以 Δx_i ，並對所有格子求和。

$F \approx \sum_i f\left(x_i^*\right) \Delta x_i$

因此可以使用均勻分佈抽樣出的 x_i 點上函數值 $f(x_i)$ 乘以平均間距 $\frac{x_{max}-x_{min}}{N}$ 求和來近似黎曼積分。

$F = \int_{x_{\min}}^{x_{\max}} f(x)dx \approx \frac{x_{\max}-x_{\min}}{N} \sum_{i=1}^{N} f\left(x_i\right)$

如果是高維積分，只需要把右邊的 $x_{max} - x_{min}$ 換成體積 $\mathcal{V}$, 積分公式變為，

$F \approx \frac{\mathcal{V}}{N} \sum_i^N f\left(x_i\right) \approx \mathcal{V}\mathrm{E}\left[f\left(x_i\right)\right]$

其中 $\mathrm{E}[f]$ 表示 f 在均勻分佈下的期望值。

馬文淦的《計算物理》中介紹，根據中心極限定理，因為 F 是大量隨機變量 $f(x_i)$ 的求和，它的值滿足正態分佈。

蒙卡積分的誤差 $\propto \frac{\sigma\left(f_{x_i}\right)}{\sqrt{n}}$ ，因此有兩種辦法可以提高蒙特卡洛積分的精度。

第一種是多撒點，將撒點個數 n 每增加 100 倍，蒙卡⊠分的誤差就會減小為原來的十分之一。這個結論獨立於積分的維數。

第二種是減小 x_i 點上集合 $\{f\left(x_i\right)\}$ 的漲落 $\propto \left(fx_i\right)$ 。

如果 $f(x) = c$ 是常數，則集合 $\{f\left(x_i\right)\}$ 的方差最小，為 $\sigma^2 = \left\langle (f - \langle f \rangle)^2 \right\rangle = 0$ 。

當 $f(x)$ 偏離均勻分佈，在局部有很尖的峰，則集合 $\{f\left(x_i\right)\}$ 的方差 (漲落) 就會比較大。

減小被積函數方差的方法是選擇一個與 $f(x)$ 形式相近，但比較好抽樣的函數 $g(x)$, 將積分寫為，

$F = \int_{\mathcal{V}} \frac{f(x)}{g(x)} g(x)dx = \mathcal{V}\mathbb{E}_g \left[\frac{f(x)}{g(x)}\right]$

其中期望值 $\mathbb{E}_g$ 表示按照概率密度函數 $g(x)$ 抽樣出一系列點 x_i ，並使用這些點計算 $f(x)/g(x)$ 的均值，

$\frac{1}{N} \sum_i^N \frac{f(x_i)}{g(x_i)}$

此時，因為 $f(x)\, g(x)$ ，被積函數 $f(x)/g(x)\, 1$ 接近常數，$\{f\left(x_i\right)/g\left(x_i\right)\}$ 方差更小，從理論上降低蒙卡⊠分的誤差。

與暴力增加 n 相比，$g(x)$ 函數的具體形式依賴於被積函數。

Vegas 積分就是要使用適配的方式，自動尋找 g(x)。

3.9 自適應方法做重要抽樣

VEGAS 積分有兩個版本。經典版本 VEGAS 與進化版本 VEGAS+。下面是 VEGAS+ 文章的摘要。

We describe a new algorithm, VEGAS+, for adaptive multidimensional Monte Carlo integration. The new algorithm adds a second adaptive strategy, adaptive stratified sampling, to the adaptive importance sampling that is the basis for its widely used predecessor VEGAS. Both VEGAS and VEGAS+ are effective for integrands with large peaks, but VEGAS+ can be much more effective for integrands with multiple peaks or other significant structures aligned with diagonals of the integration volume. We give examples where VEGAS+ is 2-17 times more accurate than VEGAS. We also show how to combine VEGAS+ with other integrators, such as the widely available MISER algorithm, to make new hybrid integrators. For a different kind of hybrid, we show how to use integrand samples, generated using MCMC or other methods, to optimize VEGAS+ before integrating. We give an example where preconditioned VEGAS+ is more than 100 times as efficient as VEGAS+ without preconditioning. Finally, we give examples where VEG AS+ is more than 10 times as efficient as MCMC for Bayesian integrals with D = 3 and 21 parameters. We explain why VEGAS+ will often outperform MCMC for small and moderate sized problems.

原始 VEGAS 積分將積分區域分成 N 份，然後動態調整每份的寬度 Δx_i ，使得 $f\left(x_i\right) \Delta x_i$ 的值趨於常數。即

$f\left(x_0\right) \Delta x_0 \approx f\left(x_1\right) \Delta x_1 \approx \ldots \approx f\left(x_n\right) \Delta\left(x_n\right) =$ const

這種方法使 $f(x)$ 值較大的地方，格子分的細，$f(x)$ 值較小的地方，格子分的粗。從而完成重要抽樣的目的，減小積分誤差。原始 VEGAS 積分對於峰值平行於自變量方向的函數比較有效。如果被積函數有多個峰值，則會產生很多" 假峰" 區域。

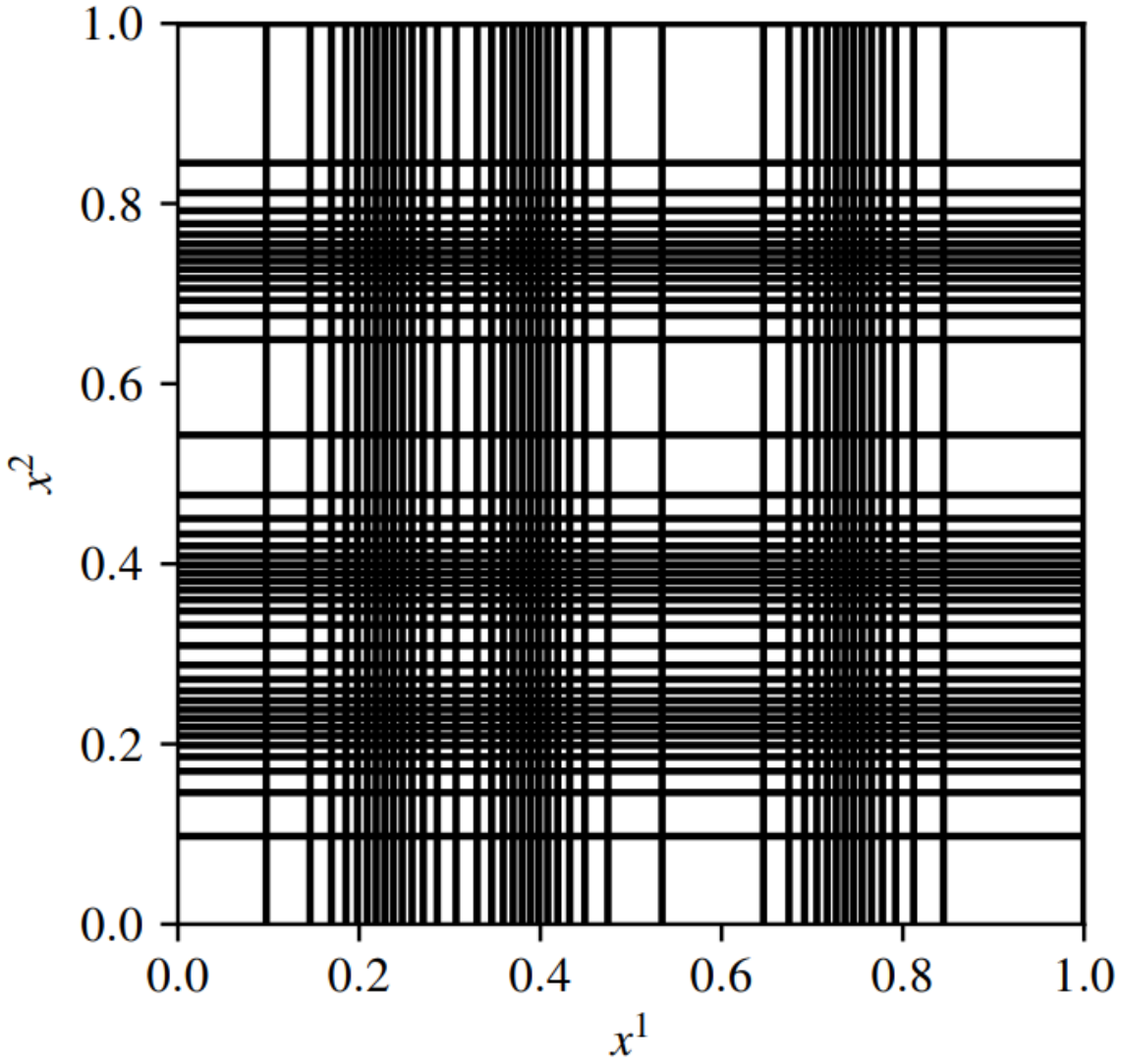

Fig. 9. 假峰

比如二維積分，有 3 個峰值，區域在紅色圓圈附近，根據原始 VEGAS 積分的自適應方法，會額外產生 6 個”假峰”，用藍色 x 標記，被積函數 $f(x^1, x^2)$ 在這些區域的值接近於 0，而自適應方法會在這 6 個區域產生很多積分點，導致原始 VEGAS 積分效率低下。

在高維積分中，如果被積函數有多個對角峰值，情況會更加嚴重。比如 10 維下的 3 對角峰值，這樣的” 假峰” 會有 $3^{10} - 3$ 個。

進化版 VEGAS+ 額外加入自適應分層抽樣 Adaptive Stratified Integration。

將積分區域分成很多小的超立方 cube，如果某個 cube 中的積分值多次迭代方差較大，就將區域細分。做多個峰值的積分更高效，比原始版本精度提高 2 到 17 倍。這種方法別的很多地方用到，比如 MISER，FOAM，ZMCIntegral 等。

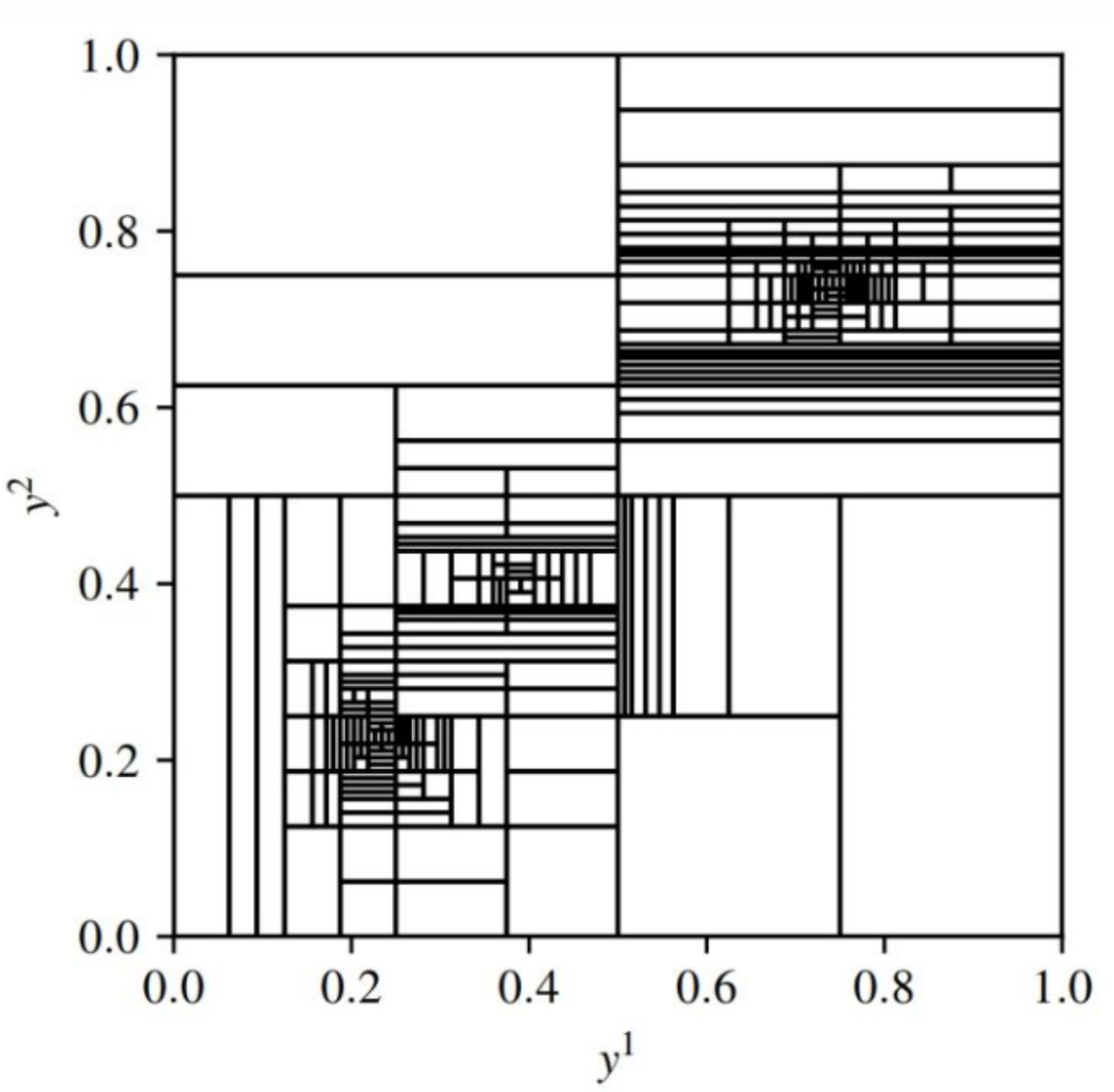

Fig. 10. MISER 自適應分層抽樣法

自適應重要抽樣加分層抽樣法加成的 VEGAS+ 在多個高維積分任務上都表現不錯。另外，2009 年的文章還提到一種新方法，使用馬爾科夫鏈蒙特卡洛 MCMC 先尋找被積函數的峰值位置，預產生一些積分位置，然後送入 VEGAS+ 中，可以比從零開始的 VEGAS+ 加速 100 倍。

4 Reference

1. MIT 6.0002 Introduction to Computational Thinking and Data - Monte Carlo Simulation :
https://www.youtube.com/watch?v=OgO1gpXSUzU
2. 蒙特卡洛原理代碼 monte carlo :
https://blog.csdn.net/yjinyyzyq/article/details/86600393
3. 蒙特卡洛法高維數值積分 Vegas :
https://zhuanlan.zhihu.com/p/264315872
4. 一文詳解蒙特卡洛（Monte Carlo）法及其應用:
https://blog.csdn.net/qq_39521554/article/details/79046646
5. https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/monte-carlo-integration

6. **蒙特卡羅方法詳解**: https://zhuanlan.zhihu.com/p/369099011
7. https://en.wikipedia.org/wiki/Monte_Carlo_method
8. https://github.com/gplepage/vegas
9. https://vegas.readthedocs.io/en/latest/
10. https://arxiv.org/abs/2009.05112
11. vegas **原始文獻**：G. P. Lepage, J. Comput. Phys. 27(1978) 192.