Exercise No. 11

David Bubeck, Pascal Becht, Patrick Nisblè

July 7, 2017

2 - Importance Sampling

We should integrate the given integral with the Monte Carlo method.

$$I = \frac{1}{\pi} \int_{-\infty}^{\infty} exp(-y_1^2 - y_2^2)\, dy_1\, dy_2 \tag{1}$$

Now, due to important sampling, we can make an approximation of our integral as a sum and make a Monte Carlo estimation of the expected values. We obtain for our integral:

$$I = \frac{1}{\pi} \frac{1}{n} \sum_{i=0}^{n} f(y_1, y_2)$$

Where n is the sample random number.
Also we define a function $g(y_1, y_2) = \frac{f(y_1,y_2)}{p(y_1,y_2)}$ for the importance sampling, where $p(y_1, y_2)$ is a probability distribution function. Now we can check if our random values are in the given intervall. If it is the case the function $g(y_1, y_2)$ will return 1 otherwise -1 to make a faster calculation.
With the code we evaluated the integral for a value $I = 7367.86152808$ for a sampling number of $n = 1000$ in the given interval $[-5, 5]$.

```python
import numpy as np
import math

def MonteCarlo(f, g, y1_begin = -5, y1_end = 5, y2_begin = -5, y2_end = 5, n = 1000):
    #draw n**2 random points in the given interval
    y1 = np.random.uniform(y1_begin, y1_end, n)
    y2 = np.random.uniform(y2_begin, y2_end, n)
    #compute sum of f values inside the integration interval
    f_sum = 0
    num_inside = 0   #number of y1, y2 points inside domain (g >= 0)
    for i in range(len(y1)):
        for j in range(len(y2)):
            if g(y1[i], y2[j]) >= 0:
                num_inside += 1
                f_sum += f(y1[i], y2[j]) #calculate sum approximation of integral
    f_sum = f_sum / num_inside
    area = num_inside / n**2 * (y1_end - y1_begin)*(y2_end - y2_begin)
    return area * f_sum

#define function g for sum approximation for integration from -infinity to infinity
def g(y1, y2, y1_begin = -math.inf, y1_end = math.inf, y2_begin = -math.inf, y2_end = math.inf):
    return (1 if (y1_begin <= y1 <= y1_end and y2_begin <= y2 <= y2_end) else -1)

#define function f(y1, y2)
def f(y1, y2):
    return 1 / np.pi * np.exp(-y1 -y2)

#print calculated value of integration
print(MonteCarlo(f, g))
```