PRINCETON COS CON

Problem 5: Abnormal Gauss [HackerRank]

By Sacheth Sathyanarayanan

1 Background: Normal (Gaussian) Distributions

The normal distribution is all around us — the blood pressure of the world's population, the height of people in America, and even our shoe sizes all seem to follow a normal distribution. All this complexity is captured by this marvelous equation:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

For the uninitiated, $P(x)$ here is called the *Probability Density Function* of the normal distribution, where μ represents the mean of the distribution and σ the standard deviation, a measure of how spread out the data is. You can think of $P(x)$ as a measure of the relative likelihood that a sample drawn from the normal distribution has value x. Note, however, that $P(x)$ does not tell you the probability of a sample drawn from the normal distribution having value x (perhaps counterintuitively, the probability that the normal distribution takes on any exact value is 0.)

2 Problem

For this problem, we are going to concern ourselves with the standard normal distribution i.e. where $\mu = 0$ and $\sigma = 1$. The Probability Density Function of the Standard Normal Distribution equals

$$P(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

The function `amirite` (provided in Python below, as well as in other languages in separate files on the website) aims to return a random number drawn from the Standard Normal Distribution. However, the code isn't complete. In particular, the functions `f` and `g` (called in lines 11 and 21 respectively) haven't been defined.

Your task is to read through the definitions of the functions `bern` and `amirite`, figure out what they do, and come with definitions for the functions `f` and `g` so that `amirite` samples a random number from the Standard Normal Distribution. You further have the constraints that `g`(0) = 0 and that `f` and `g` are deterministic functions.

```python
import random
import numpy as np

def bern():
    n = 0
    u,v = 0.5,random.uniform(0,1)
    while (v >= u):
        u = v
        v = random.uniform(0,1)
        n = n + 1
    return f(v, n)

def amirite():
    flag2 = False
    while (not flag2):
        flag1 = False
        while (not flag1):
            k = 0
            while (bern()):
                k = k + 1
            m = g(k)
            flag1 = True
            for i in range(m):
                flag1 = flag1 and bern()
        x = random.uniform(0,1)
        flag2 = random.uniform(0,1) < np.exp(-0.5*x*(2*k+x))
    y = k + x
    if (random.uniform(0,1) > 0.5):
        return -1*y
    return y
```

Once you figure out what the functions `f` and `g` should be, write a program that, takes as input some positive integer n, and prints the value of

$$\mathtt{f}(1/1,1)+\mathtt{f}(1/2,2)+\cdots+\mathtt{f}(1/(n-1),n-1)+\mathtt{f}(1/n,n)+\mathtt{g}(1)+\mathtt{g}(2)+\cdots+\mathtt{g}(n-1)+\mathtt{g}(n).$$

Remember that any variable with value "True" evaluates to 1, while any variable with value "False" evaluates to 0.

Here is some code to help you test your implementation:

```
n = 100000
data = np.zeros(n)
for i in range(n):
   data[i] = amirite()

plt.hist(data, bins=np.arange(data.min(), data.max()+1, 0.1))
plt.show()
```

This program calls the `amirite` function repeatedly and plots a histogram of the returned values. If your answer is correct, the plot here should look relatively normal. We've provided this code in C++ and Java too (found under the auxiliary files zip for this problem).

Input

The first and only line contains a positive integer n.

Output

The output should consist of the value of the expression

$$\sum_{i=1}^{n}\Big(\mathtt{f}(1/i,i)+\mathtt{g}(i)\Big)=\mathtt{f}(1/1,1)+\mathtt{f}(1/2,2)+\cdots+\mathtt{f}(1/n,n)+\mathtt{g}(1)+\mathtt{g}(2)+\cdots+\mathtt{g}(n).$$

Note

We are fairly certain that there is a unique simple choice for both `f` and `g` satisfying the constraints above. However, if you have a solution that you think works and are failing the HackerRank testcases, please email us at `ptonacm@princeton.edu` or jump on the Zoom call.