Homework 3

Computational Methods for Data Science

r11946013 簡立誠

Q1.

(a) Normal

$$p = \frac{1}{\sqrt{2\pi\sigma^2}} * exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

(b) Exponential

$$P = \lambda e^{-\lambda}$$

$$cdf = 1 - \lambda e^{-\lambda}$$

```
def Exponential_Distribution(l):
    U = get_RandomVariable()
    return np.log(1-U) / (-l)
```

(c) Poisson

$$P = \frac{\lambda^x e^{-\lambda}}{x!}$$

$$cdf = \sum_{i=0}^{n} p(x_i)$$

```
def Poisson_Distribution(l):
    U = get_RandomVariable()
    cdf = 0
    x = 0
    while cdf <= U:
        cdf += (l ** x) * np.exp(-l) / math.factorial(x)
        x += 1
    return x-1
```

(d) Chi-Square

$$\mathrm{p} = \frac{1}{2^{k/2}\,\Gamma\,(k/2)} x^{k/2-1} e^{-x/2}$$

$$cdf = \frac{1}{\Gamma\,(k/2)} \gamma\left(\frac{k}{2}, \frac{x}{2}\right)$$

(e) $F_{k,m}$

(f) Binomial

$$p = C_{\mathrm{x}}^{n} p^{\mathrm{x}} (1-p)^{n-\mathrm{x}}$$

$$cdf = \sum_{i=0}^{n} p(x_i)$$

```
def Binomial_Distribution(n, p):
    U = get_RandomVariable()
    cdf = 0
    x = 0
    while cdf <= U:
        cdf += (math.factorial(n) / (math.factorial(x) * math.factorial(n-x))) *\
            (p**x) * ((1-p)**(n-x))
        x += 1
    return x-1
```

(g) Negative Binomial

$$p = C_{\mathrm{r-1}}^{\mathrm{x+r-1}} p^{\mathrm{r}} (1-p)^{\mathrm{x}}$$

$$cdf = \sum_{i=0}^{n} p(x_i)$$

```
def N_Binomial_Distribution(r, p):
    U = get_RandomVariable()
    cdf = 0
    x = 0
    while cdf <= U:
        cdf += (math.factorial(x+r-1) / (math.factorial(r-1) * math.factorial(x))) *\
            (p**(r)) * ((1-p)**(x))
        x += 1
    return x-1
```

(h) Dirichlet

Q2.

---

p(x) is the target distribution and q(x) is the simulated distribution. I assume that the distribution is as following respectively.

```
def simulated_distribution(x):
    value = np.exp(-abs(x)**3 / 3)
    return value
def target_distribution(mean, sd):
    f = stats.norm(mean, sd)
    return f
```

(a)

```python
def Importance_sampling(n, target_mean, target_std):
    p = target_distribution(target_mean, target_std)
    x = np.zeros(n)
    weight = np.zeros(n)

    for i in range(n):
        x[i] = random.random()
        q_x = simulated_distribution(x[i])
        p_x = p.cdf(x[i])

        w_x = p_x / q_x
        weight[i] = w_x

    return np.sum((x**2) * weight) / n
```

As the problem set, target mean is 0 and target std is 1. I do sampling for 10000 times. The estimated $E(X^2)$ is 0.3058

For the distribution of estimation,

(b)

```python
def Rejection_sampling(n, target_mean, target_std):
    p = target_distribution(target_mean, target_std)
    x = np.zeros(n)

    i = 0
    while i < n:
        rv = random.random()
        U = random.uniform(0, 1)
        q_x = simulated_distribution(rv)
        p_x = p.cdf(rv)
        if U < p_x/q_x:
            x[i] = rv
            i = i + 1

    return np.sum((x**2) * p.cdf(x)) / n
```

As the problem set, target mean is 0 and target std is 1. I do sampling for 10000 times. The estimated $E(X^2)$ is 0.3135

Q3.

(a)

"h" will affect the accuracy of the estimation. Thus, if h increases, $\hat{\pi}$ will be close to 3.14. In fact, it has a lower bound.

Simulation for 100 times		
h	Mean	Std
1	3.448	0.009
2	3.319	0.011
3	3.318	0.009

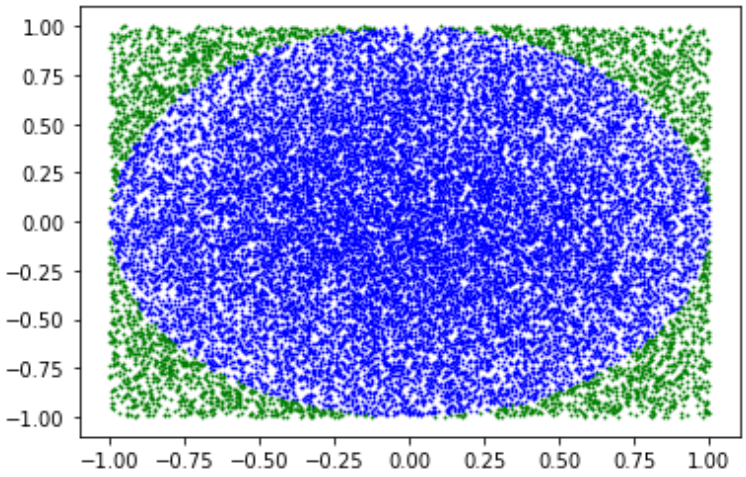
“n” will affect the convergence of the distribution. The larger the “n” is, the smaller the standard deviation is.

Simulation for 100 times		
n	Mean	Std
200	3.466	0.093
1000	3.449	0.038
2000	3.447	0.026
20000	3.448	0.01

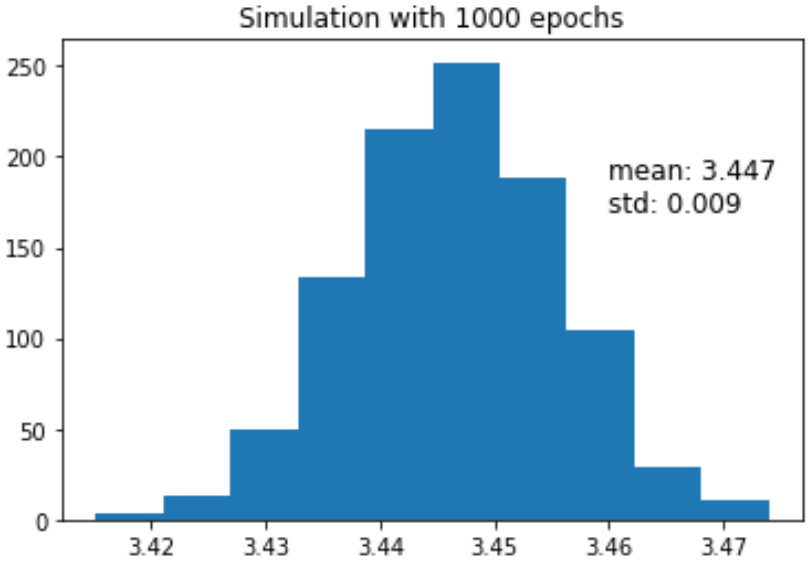
(b)

The method finds a new point based on the previous simulated point, which may cause bias of the estimation.

For example, the following figure is the simulation of the given method, and $\hat{\pi}$ is 3.4619 with generated 23109 points.

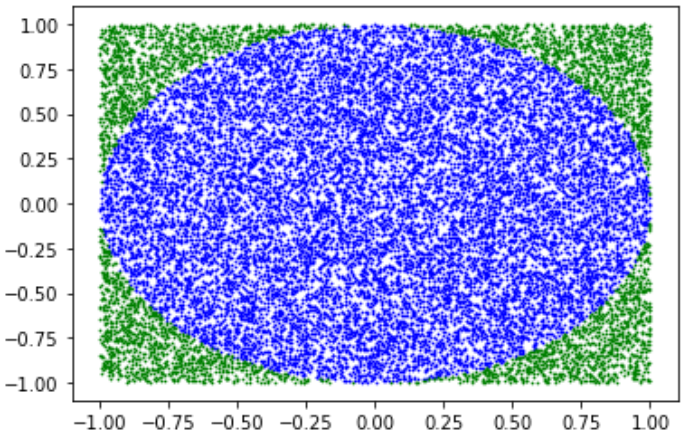


When we simulate for 1000 times to find its distribution, it shows that mean has bias to 3.44.

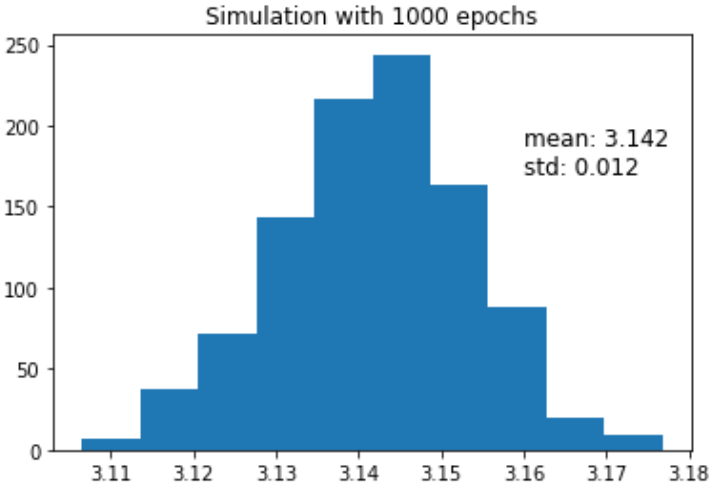


(c)

the following figure is the simulation of my method, which $\hat{\pi}$ is 3.136. And there is



the histogram of π estimation for 1000 epochs simulation. It shows that my method can accurately estimate π which is close to 3.14.



code:

```
def pi_estimation(n, h):
    X = np.zeros((2, n))
    t = np.zeros((1, n))

    for epoch in range(n):
        X[0, epoch] = 2 * uniform(0, h) - 1
        X[1, epoch] = 2 * uniform(0, h) - 1

        if X[0, epoch]**2 + X[1, epoch]**2 <= 1:
            t[0, epoch] = 1

    circle = X[:, t[0,:]==1]
    square = X[:, t[0,:]==0]
    plt.scatter(circle[0,:], circle[1,:], s=0.5, color="blue")
    plt.scatter(square[0,:], square[1,:], s=0.5, color="green")

    return t.sum() / n * 4
```

Q4.

Q5.

Q6.