ESC 과제 7주차.

#1. 방법: Gibbs Sampling

$X_i \overset{iid}{\sim} Exp(1)$; $pdf_X = e^{-x} I_{(0,\infty)}$

알고리즘

(1) 초기화 : $\mathbb{X}_0 = (X_1^0, \cdots, X_{10}^0)^t \sim p(\mathbb{X} \mid \prod_{i=1}^{10} X_i > 20)$.

(2) iteration : $i = 1, 2, \cdots$

$$x_1^{(i)} \sim p\left(X_1 > \frac{20}{S^{(-1)}} \,\middle|\, X_2 = x_2^{(i-1)}, \cdots, X_{10} = x_{10}^{(i-1)}\right)$$

$$x_2^{(i)} \sim p\left(X_2 > \frac{20}{S^{(-2)}} \,\middle|\, X_1 = x_1^{(i)}, X_3 = x_3^{(i-1)}, \cdots, X_{10} = x_{10}^{(i-1)}\right)$$

$$\vdots$$

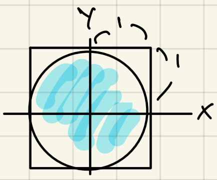
$$x_{10}^{(i)} \sim p\left(X_{10} > \frac{20}{S^{(-10)}} \,\middle|\, X_1 = x_1^{(i)}, \cdots, X_9 = x_9^{(i)}\right)$$

where $S^{(-k)} = \prod_{j<k} x_j^{(i)} \prod_{j>k}^{10} x_j^{(i-1)}$

Note $p\left(X > \frac{20}{S^{(-k)}}\right) \overset{d}{=} e^{-(x - 20/S^{(-k)})} I_{\left(\frac{20}{S^{(-k)}}, \infty\right)} \overset{d}{=} Exp\left(\underset{\text{scale}}{1}, \underset{\text{location}}{\frac{20}{S^{(-k)}}}\right)$

memoryless property

#2. Generate π



$X \sim Unif(0,1)$
$Y \sim Unif(0,1)$

$Pr(X^2 + Y^2 < 1) = \frac{\pi}{4}$ i.e. $\frac{1}{S} \sum_{i=1}^{S} I(x^2 + y^2 < 1) \xrightarrow{p} E[I(X^2 + Y^2 < 1)] = \frac{\pi}{4}$

by WLLN

In [1]:

```
import numpy as np
from scipy.stats import expon
import matplotlib.pyplot as plt
import seaborn as sns
import copy
```

Generate Pi

In [2]:

```
cnt = 0
MC_iter = 1000000
for n in range(MC_iter):
    x = np.random.rand(1)
    y = np.random.rand(1)
    if ((x ** 2 + y ** 2) < 1):
        cnt += 1

pi_MC = 4 * cnt / MC_iter
print('pi using MC: ', pi_MC)
```

```
pi using MC:  3.141692
```

HW Gibbs Sampling

In [3]:

```
def sample_initial():
    X_prod = 0
    while X_prod < 20:
        X = expon.rvs(size = 10)
        X_prod = np.product(X)
    return X
```

In [4]:

```
# Basic  Setup & Inittialize
X0 = sample_initial()
n_iter = 1000
Xi = X0
MCMC_samples = []
burn_in = 100
accept = 0
cnt = 0

# MCMC sampling using Gibbs
for n in range(n_iter):

    for ind in range(10): # Loop for Gibbs sampling
        cnt = cnt + 1
        X_temp = copy.deepcopy(Xi)
        # Gibbs Sampling
        X_before = np.delete(Xi, ind, 0)
        S_k = np.product( X_before )
        x_i = expon.rvs(loc = 20 / S_k)

        if x_i * S_k > 20:
            check = True
            X_temp[ind] = x_i # update new value
            Xi = copy.deepcopy(X_temp)
            accept = accept + 1
        else: # for sanity check (Gibbs sampling should always accept)
            Xi = copy.deepcopy( X_temp )
            x_i = Xi[ind] # stay with current value

        if ( cnt > burn_in ) & (cnt < burn_in + 10): # Check
```

```
            print('=' * 20)
            print('n: ', cnt)
            if check == True:
                print('accept')
                check = False
            print('S:', x_i*S_k, 'x_i:', x_i)
            print('X list:', Xi)
        MCMC_samples.append(x_i)
```

```
====================
n:  101
accept
S: 92.74995673935608 x_i: 2.5612888744886506
X list: [2.56128887 4.98450013 1.50375555 4.42329594 0.19224103 4.5159913
 1.03780649 0.56671516 1.58862602 1.34650583]
====================
n:  102
accept
S: 23.321921990325595 x_i: 1.2533496223723681
X list: [2.56128887 1.25334962 1.50375555 4.42329594 0.19224103 4.5159913
 1.03780649 0.56671516 1.58862602 1.34650583]
====================
n:  103
accept
S: 22.453177958906966 x_i: 1.4477404973959602
X list: [2.56128887 1.25334962 1.4477405  4.42329594 0.19224103 4.5159913
 1.03780649 0.56671516 1.58862602 1.34650583]
====================
n:  104
accept
S: 22.977669713013903 x_i: 4.5266212749383925
X list: [2.56128887 1.25334962 1.4477405  4.52662127 0.19224103 4.5159913
 1.03780649 0.56671516 1.58862602 1.34650583]
====================
n:  105
accept
S: 104.73147616733714 x_i: 0.8762284051460016
X list: [2.56128887 1.25334962 1.4477405  4.52662127 0.87622841 4.5159913
 1.03780649 0.56671516 1.58862602 1.34650583]
====================
n:  106
accept
S: 29.11700291890049 x_i: 1.25551683943565
X list: [2.56128887 1.25334962 1.4477405  4.52662127 0.87622841 1.25551684
 1.03780649 0.56671516 1.58862602 1.34650583]
====================
n:  107
accept
S: 97.76776114609522 x_i: 3.484699900925023
X list: [2.56128887 1.25334962 1.4477405  4.52662127 0.87622841 1.25551684
 3.4846999  0.56671516 1.58862602 1.34650583]
====================
n:  108
accept
S: 59.09197587220626 x_i: 0.34252925429988856
X list: [2.56128887 1.25334962 1.4477405  4.52662127 0.87622841 1.25551684
 3.4846999  0.34252925 1.58862602 1.34650583]
====================
n:  109
accept
S: 29.388059611929396 x_i: 0.7900672703420712
X list: [2.56128887 1.25334962 1.4477405  4.52662127 0.87622841 1.25551684
 3.4846999  0.34252925 0.79006727 1.34650583]
```

In [5]:

```
MCMC_effective = MCMC_samples[burn_in:]
```

Acceptance ratio

In [6]:

```
accept / cnt
```

Out[6]:

1.0

Distribution

In [7]:

```
print('Posterior mean: ', np.mean(MCMC_effective) )
```

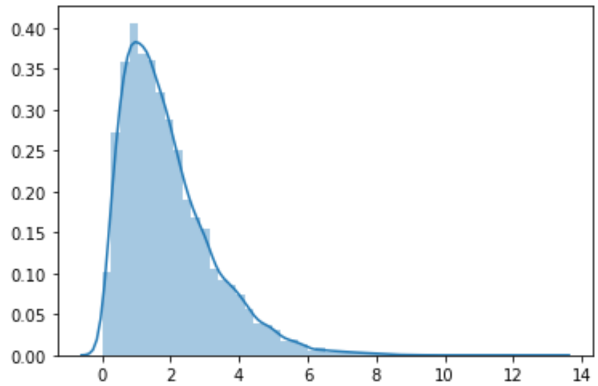
Posterior mean: 1.9085014361395742

In [8]:

```
sns.distplot(MCMC_effective)
#X_exp = expon.rvs(scale = np.mean(MCMC_effective), size = 1000)
#sns.distplot(X_exp)
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x1bfb2d36b50>



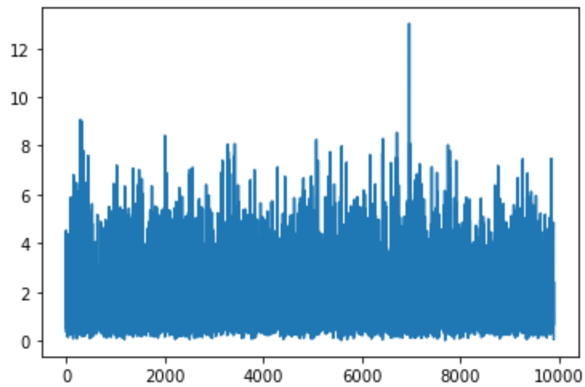
Traceplot

In [9]:

```
plt.plot(MCMC_effective)
```

Out[9]:

[<matplotlib.lines.Line2D at 0x1bfb7e250d0>]



In [ ]: