Statistical Computing 2

숙제 1

2019년 가을학기

응용통계학과 석사과정 최석준

1. Verify normal (simulated) sample moments

```
[R]
x = rnorm(10000)
mean(x)
mean(x**2)
mean(x**3)
mean(x**4)
```

Result:

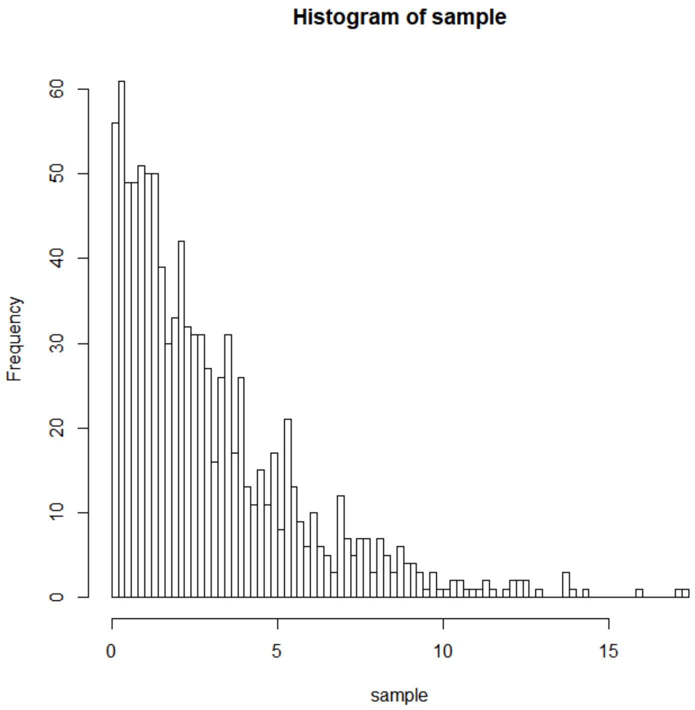
```
[R interpreter]
> x = rnorm(10000)
> mean(x)
[1] 0.01023029
> mean(x**2)
[1] 1.004741
> mean(x**3)
[1] 0.07036638
> mean(x**4)
[1] 3.004127
```

이론적 moment 인 0, 1, 0, 3 에 충분히 가까운 값이 나왔다.

2. Get exponential dist samples

```
[R]
u = runif(1000)
beta = 3
sample = -beta * log(u)
hist(sample,nclass=100)
# using nclass with more than # 100
```

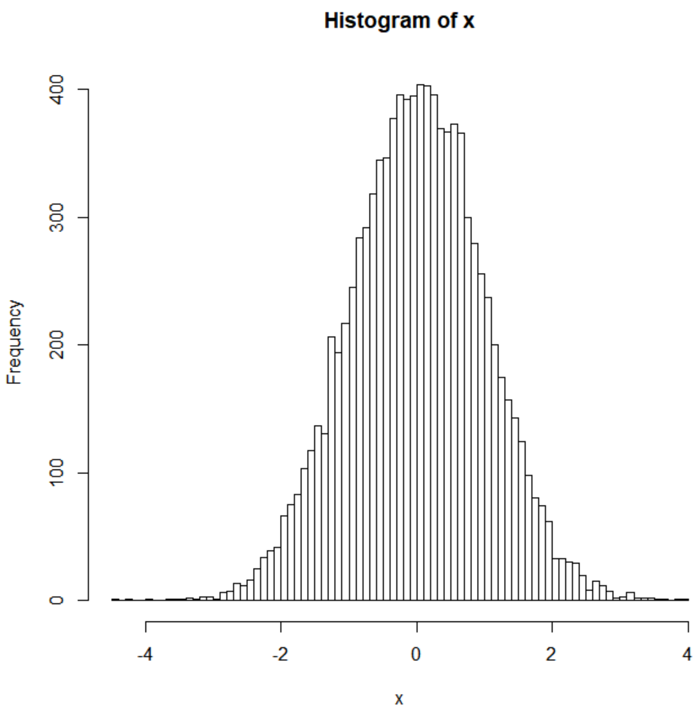
Result:



3. Generate normal dist samples using rejection sampling

```
[R]
n = 10000
x = rep(NA, n)
index = 1

while(index <=n){
    y = rexp(1)
    r = exp(-(y-1)**2 / 2)
    u = runif(1)

    if(u < r ){
      u2 = runif(1)
      if(u2 < 0.5)
        x[index] = -y
      else x[index] = y

      index = index + 1
    }
}

hist(x , nclass = 100)
```

Result:



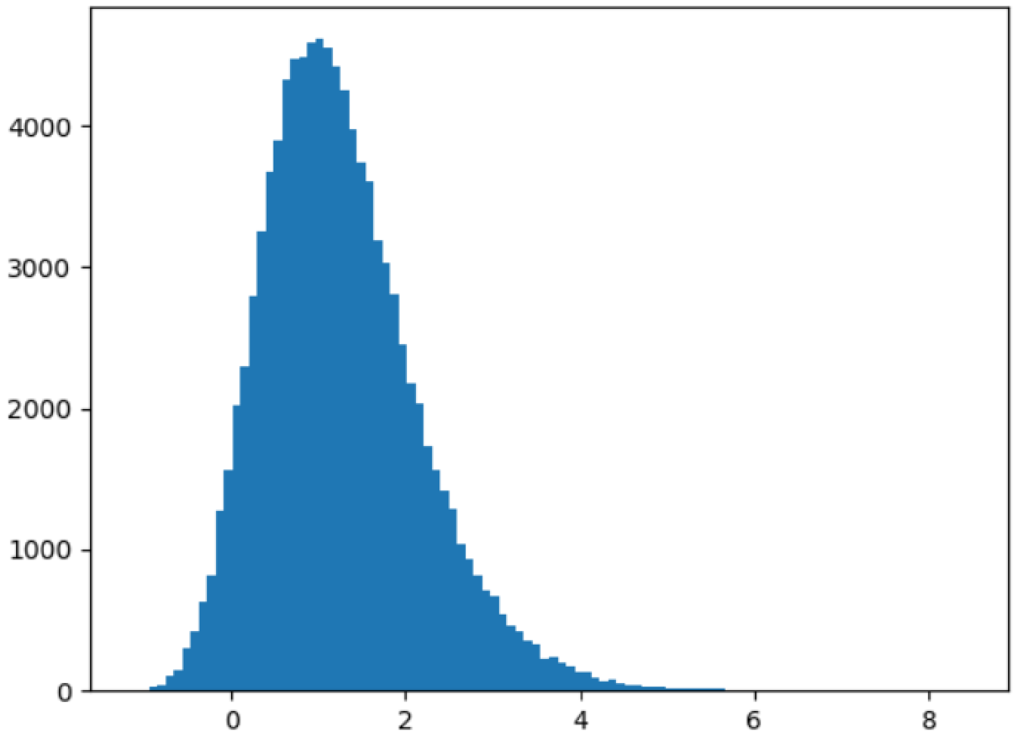
4. Get gamma dist random sample

생성할 Gamma distribution ($\mathrm{f(x)} = \frac{\mathrm{x}^{\alpha-1}e^{-\frac{x}{\beta}}}{\Gamma(\alpha)\beta^{\alpha}}$)의 parameter 는 임의로 $\alpha = 2$, $\beta = 0.5$로 선택하였다. (다른 parameter 로 바꾸고싶다면, GammaSampler 의 인스턴스 생성시 넘기는 argument 를 조정)

```
[Python]
#python 3 file created by Choi, Seokjun

#get gamma-distributed random samples
#using exponential samples achived by inverse-cdf method

from math import exp, log
from random import uniform

import matplotlib.pyplot as plt


class ExponentialSampler:
    def __init__(self, param_scale):
        self.param_scale = param_scale

    def exponential_sampler(self):
        unif_sample = uniform(0,1)
        return (-self.param_scale*log(unif_sample))

    def get_exponential_sample(self, number_of_smpl):
        result = []
        for i in range(0, number_of_smpl):
            result.append(self.exponential_sampler())
        return result

class GammaSampler(ExponentialSampler):
    def __init__(self, param_alpha, param_beta):
        if param_alpha%1 != 0:
            raise ValueError("alpha should be integer")
        self.param_alpha = param_alpha
        self.param_scale = param_beta

    def gamma_sampler(self):
        exp_samples = self.get_exponential_sample(self.param_alpha)
        product = 1
        for smpl in exp_samples:
            product = product * smpl
        return (-1 * log(product) * self.param_scale)

    def get_gamma_sample(self, number_of_smpl):
        result = []
        for i in range(0, number_of_smpl):
            result.append(self.gamma_sampler())
        return result


if __name__ == "__main__":
    print('run as main')
    # EXPsampler = ExponentialSampler(0.5)
    # print(EXPsampler.get_exponential_sample(10))

    GAMMAsampler = GammaSampler(2,0.5)
    print(GAMMAsampler.get_gamma_sample(10))

    plt.hist(GAMMAsampler.get_gamma_sample(100000), bins=100)
    plt.show()
```

Result:

```
[Console]
run as main
[0.5052901902926614, -0.36278325808861195, 0.9035354534602383, 1.4714168818996327, 1.386355241662388, 2.3548797102123236, 0.41343485059649265, 2.354007303982918, 1.3657003234263956, 1.3800411374685848]
```

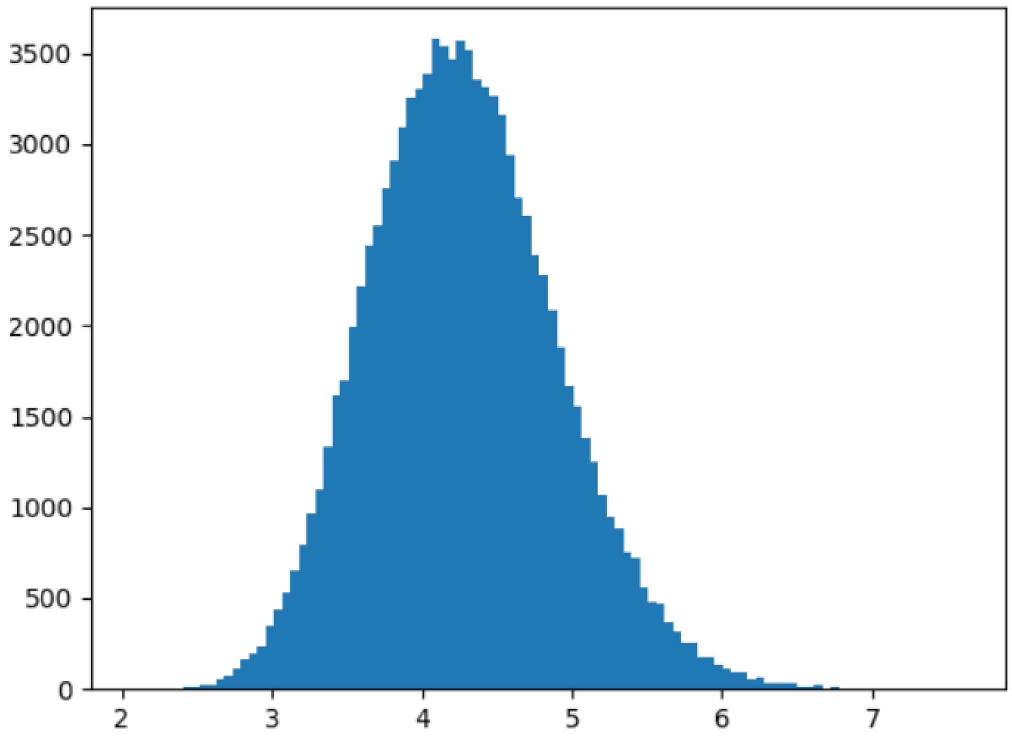


5. Get Poisson parameter’s samples using rejection sampling

```
[Python]
#python 3 file created by Choi, Seokjun

#using rejection sampling method,
#sampling poisson's parameter value
#with lognormal density

from math import log, exp, factorial
from statistics import mean
import random

import matplotlib.pyplot as plt



class Lognormal_Poisson_RejectionSampler:
    def __init__(self, data):
        self.data = data
        self.data_mean = mean(data)

    def pois_pmf(self, x, param_lambda):
        if not isinstance(x, int):
            raise ValueError("x should be integer.")
        return ((param_lambda**x)*exp(-param_lambda)/factorial(x))

    def thres_p_calculator(self, lognorm_sample):
        thres_p_upper = (self.pois_pmf(x, lognorm_sample) for x in self.data)
        thres_p_lower = (self.pois_pmf(x, self.data_mean) for x in self.data)
        thres_p = 1
        for up, low in zip(thres_p_upper, thres_p_lower):
            thres_p = thres_p * up/low
        return thres_p

    def sampler(self):
        #get one sample
        while(1):
            unif_sample = random.uniform(0,1)
            lognorm_sample = exp(random.normalvariate(log(4), 0.5))

            thres_p = self.thres_p_calculator(lognorm_sample)

            # print('p: ', thres_p ," and now uniform r.s : ", unif_sample)
            if unif_sample < thres_p:
                # print('accepted: ', lognorm_sample)
                yield lognorm_sample
            else:
                # print('rejected')
                pass
    def get_sample(self, number_of_smpl):
        result = []
        for i in range(0, number_of_smpl):
            result.append(next(self.sampler()))
        return result



if __name__ == "__main__" :
    print('run as main')
    random.seed(2019-311-252)
    given_data = (8,3,4,3,1,7,2,6,2,7)
    LPsampler = Lognormal_Poisson_RejectionSampler(given_data)
    print(LPsampler.get_sample(10))
    plt.hist(LPsampler.get_sample(100000), bins=100)
    plt.show()
```

Result:

```
[Console]
run as main
[3.9497928775588886,    4.917777219059587,    3.854358211869205,    3.9719816335092757,    4.482831620647739,
3.9060658551095644, 4.584039558823791, 4.802052259589951, 3.691898778304127, 3.6294331768172485]
```


3500
3000
2500
2000
1500
1000
500
0
2
3
4
5
6
7