Statistical Computing 2

숙제 2
2019년 가을학기
응용통계학과 석사과정 최석준

1. Generate 10000 samples from Cauchy(0,1) using inverse-CDF method

Cauchy(x_0,r)의 inverse cdf 는 다음과 같음을 이용하자.

$$F^{-1}(y) = r * \tan\left(\pi\left(y - \frac{1}{2}\right)\right) + x_0$$

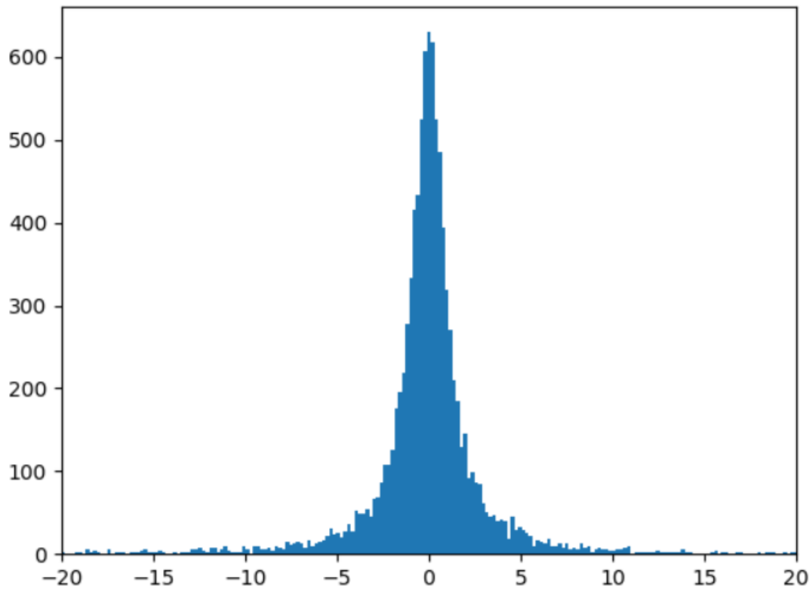
```
[Python]
#python 3 file created by Choi, Seokjun

#get cauchy-distributed random samples
#using inverse-cdf method


from math import tan, pi
from random import uniform, seed

import matplotlib.pyplot as plt

class CauchySampler:
    def __init__(self, param_loc, param_scale):
        if param_scale <= 0:
            raise ValueError("scale parameter should be >0")
        self.param_loc = param_loc
        self.param_scale = param_scale

    def sampler(self):
        unif_sample = uniform(0,1)
        return (self.param_scale * tan(pi * (unif_sample - 0.5)) + self.param_loc)

    def get_sample(self, number_of_smpl):
        result = []
        for _ in range(0, number_of_smpl):
            result.append(self.sampler())
        return result


if __name__ == "__main__":
    print('run as main')
```

```
seed(2019-311-252)
Cauchy_sampler_instance = CauchySampler(0,1)
print(Cauchy_sampler_instance.get_sample(10))

plt.xlim(-20,20)
plt.hist(Cauchy_sampler_instance.get_sample(10000), bins=50000)
plt.show()
```

Result:

(참고) Cauchy(0,1) 분포의 tail 이 매우 두꺼워서 여기저기 크게 튄 값이 많이 나오므로, histogram 그리는 작업이 느리다....

```
[Console]
run as main
[-0.11284689965119417, -1.9633021466033382, -28.380625688220956, -1.8353234847865283, -0.7508065088140116, 0.27029112673738653, 2.130170742041151, -1.023582509235834, -0.4304000163112252, 1.1886282394928387]
```



2. 다음의 분포를 따르는 sample 을 inverse CDF method 와 rejection sampling method 를 이용하여 생성하라. Rejection sampling 시 proposal distribution 으로는 g~unif(0,1)을, envelope 로는 3g 를 사용하라.

$$
\begin{aligned}
&\mathrm{F(x)} \\
&= 0 \text{ if } \mathrm{x} \le 0 \\
&= 4x^2 \; if \; 0 \le x < 0.25 \\
&= \frac{8x}{3} - \frac{4x^2}{3} - \frac{1}{3} \; if \; 0.25 \le x < 1 \\
&= 1 \; if \;\; \mathrm{x} > 1
\end{aligned}
$$

Inverse cdf 를 구해보면 다음과 같다.

$$
\begin{aligned}
&\mathrm{F}^{-1}(y) \\
&= \frac{1}{2}\sqrt{y} \; if \; 0 \le y < 0.25 \\
&= -\sqrt{-\frac{3}{4}(y-1)} + 1 \; if \; 0.25 \le y \le 1
\end{aligned}
$$

또한 pdf 는 다음과 같다.

$$
\begin{aligned}
&\mathrm{f(x)} \\
&= 8\mathrm{x} \;\; if \;\; 0 < \mathrm{x} \le 0.25 \\
&= -\frac{8}{3}x + \frac{8}{3} \; if \; 0.25 < x \le 1 \\
&= 0 \text{ otherwise}
\end{aligned}
$$

이를 이용하자.

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

# sample from below distribution!
# our cdf:
# F(x)
#     = 0 if x<0
#     = 4x^2 if 0<x<0.25
#     = 8x/3 - 4x^2/3 -1/3 if 0.25<x<1
#     = 1 if x>1

# correspond pdf:
# f(x)
#     = 8x if 0<x<0.25
#     = -8x/3 + 8/3 if 0.25<=x<1
#     = 0 otherwise


from math import sqrt
from random import uniform, seed
from functools import partial
```

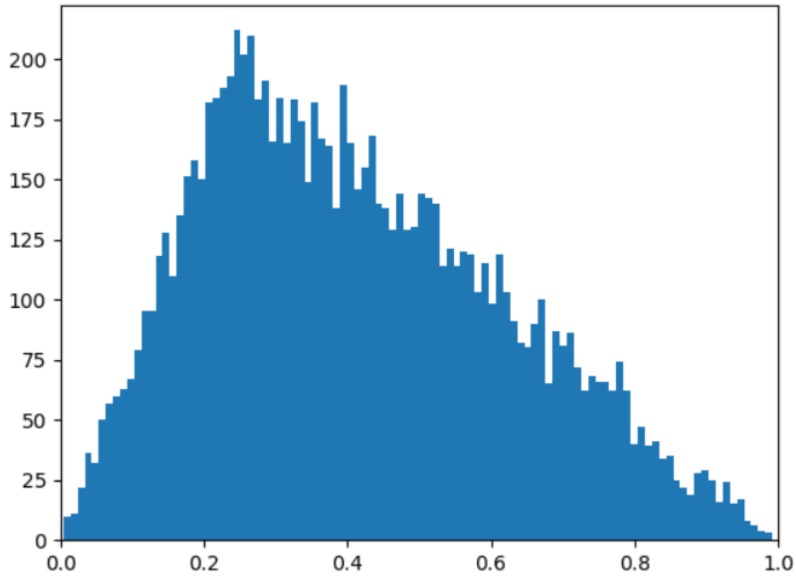
```
import matplotlib.pyplot as plt


class InvCdfSampler:
    def __init__(self, inv_cdf):
        #inv_cdf should be function
        self.inv_cdf = inv_cdf

    def sampler(self):
        unif_sample = uniform(0,1)
        return self.inv_cdf(unif_sample)

    def get_sample(self, number_of_smpl):
        result = []
        for _ in range(0, number_of_smpl):
            result.append(self.sampler())
        return result


class RejectionSampler:
    def __init__(self, target_pdf, proposal_pdf, proposal_sampler, envelope_multiplier):
        # target pdf, proposal pdf, proposal sampler must be functions.
        # <caution> "envelop_multiplier" value satisfies "envelope_multiplier * proposal_pdf > target_pdf"
        # proposal sampler should not be the function that has arguments.
        #       just from proposal_sampler() we should able to get 1 sample
        self.target_pdf = target_pdf
        self.proposal_pdf = proposal_pdf
        self.proposal_sampler = proposal_sampler
        self.envelope_multiplier = envelope_multiplier

    def envelope(self, x):
        return (self.proposal_pdf(x) * self.envelope_multiplier)

    def sampler(self):
        while(True):
            unif_sample = uniform(0,1)
            proposal_sample = self.proposal_sampler()

            thres = self.target_pdf(proposal_sample) / self.envelope(proposal_sample)
            if thres > unif_sample:
                return proposal_sample

    def get_sample(self, number_of_smpl):
        result = []
        for _ in range(0, number_of_smpl):
            result.append(self.sampler())
        return result
```

```python
def triangle_inv_cdf(y):
    if (0 <= y < 0.25):
        return (0.5 * sqrt(y))
    elif (0.25 <= y <= 1):
        return (-sqrt(-0.75 * (y - 1)) + 1)
    else:
        raise ValueError('input of inverse cdf should be 0<=y<=1')


def triangle_pdf(x):
    if (0 <= x < 0.25):
        return (8*x)
    elif (0.25 <= x < 1):
        return (-8*x/3 + 8/3)
    else:
        return 0

if __name__ == "__main__":
    print('run as main')

    seed(2019-311-252)

    #test for inv_cdf
    assert triangle_inv_cdf(0.25) == 0.25 #should be 0.25
    assert triangle_inv_cdf(1) == 1 #should be 1

    #test for pdf
    assert triangle_pdf(0.25) == 2 #should be 2

    #Inv CDF sampler
    TriangleInvCdfSampler = InvCdfSampler(triangle_inv_cdf)
    print(TriangleInvCdfSampler.get_sample(10))
    plt.xlim(0,1)
    plt.hist(TriangleInvCdfSampler.get_sample(10000), bins=100)
    plt.show()

    #Rejection sampler
    #uniform proposal, envelop = 3*uniform(0,1)
    TriangleRejectionSampler = RejectionSampler(triangle_pdf, lambda x : 1, partial(uniform,0,1), 3)
    print(TriangleRejectionSampler.get_sample(10))
    plt.hist(TriangleRejectionSampler.get_sample(10000), bins=100)
    plt.show()
```

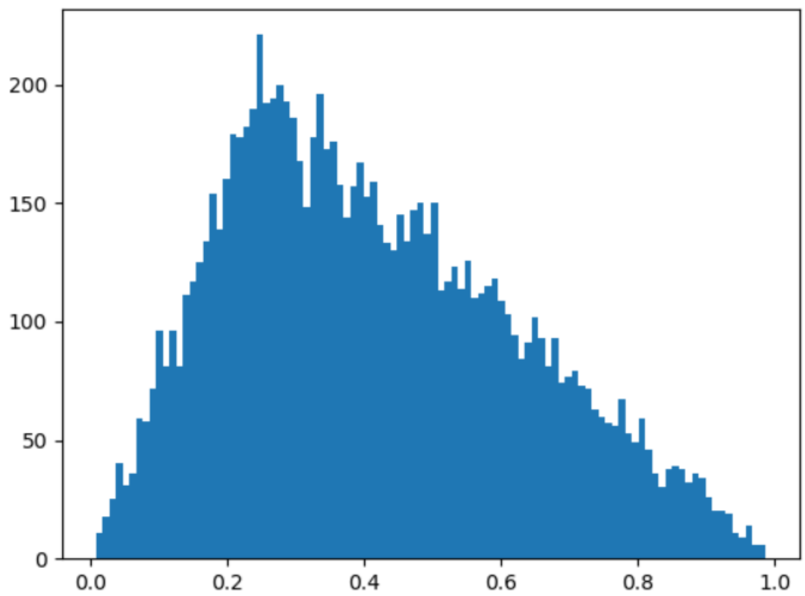
Result:

위 출력은 Inverse cdf method 를 사용한 결과이며, 아래 출력은 rejection sampling 을 사용한 결과이다.

[Console]
run as main
[0.3661019619029926, 0.19361988332535598, 0.05294125944024263, 0.19924975652352328, 0.2728495676284478, 0.44144949848959514, 0.6763005684075929, 0.24813838631423896, 0.3129560762152266, 0.5913737383162287]



[Console]
[0.5266883424875778, 0.2934369128774007, 0.4205264310994711, 0.35614461376662954, 0.4295837142002811, 0.8854860724340627, 0.2981637540961075, 0.4271526160550476, 0.8936465606324642, 0.7360811600063285]

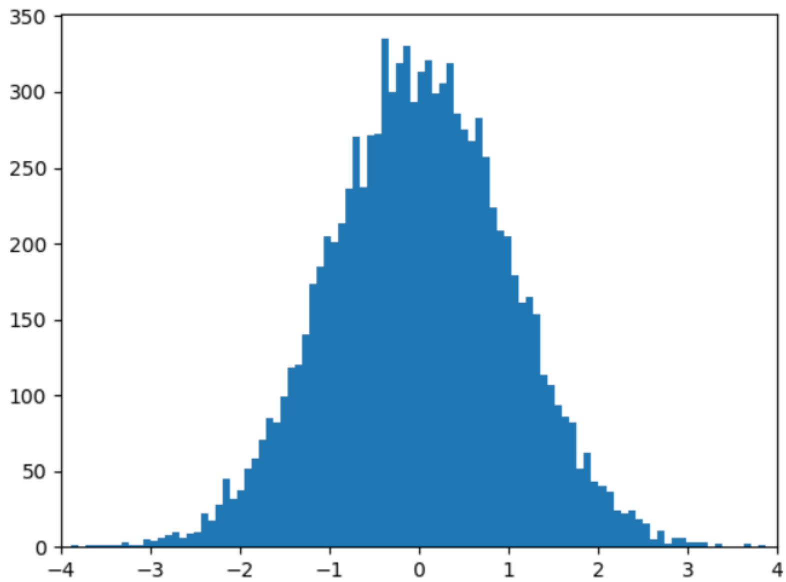


3. Polar method 로 Normal sample 을 생성하라.

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

#get normal samples by polar method

from math import sin, cos, log, pi, sqrt
from random import uniform, seed

import matplotlib.pyplot as plt


class NormalPolarSampler:
    def __init__(self, param_mean, param_std):
        self.param_mean = param_mean
        self.param_std = param_std

    def sampler(self):
        unif1 = uniform(0,1)
        unif2 = uniform(0,1)

        #polar coordinate
        R = sqrt(-2*log(unif1))
        theta = 2*pi*unif2
        return [R*sin(theta)*self.param_std + self.param_mean,
                R*cos(theta)*self.param_std + self.param_mean]

    def get_sample(self, number_of_smpl):
        result = []
        for _ in range(0, number_of_smpl//2):
            result += self.sampler()

        if(number_of_smpl%2==1):
            result.append(self.sampler()[0])

        return result

if __name__ == "__main__":
    print('run as main')

    seed(2019-311-252)

    NormalSampler = NormalPolarSampler(0,1)
    print(NormalSampler.get_sample(5))

    plt.xlim(-4,4)
    plt.hist(NormalSampler.get_sample(10000), bins=100)
    plt.show()
```

Result:

histogram 은 10000 개의 sample 을 생성하여 그렸다.

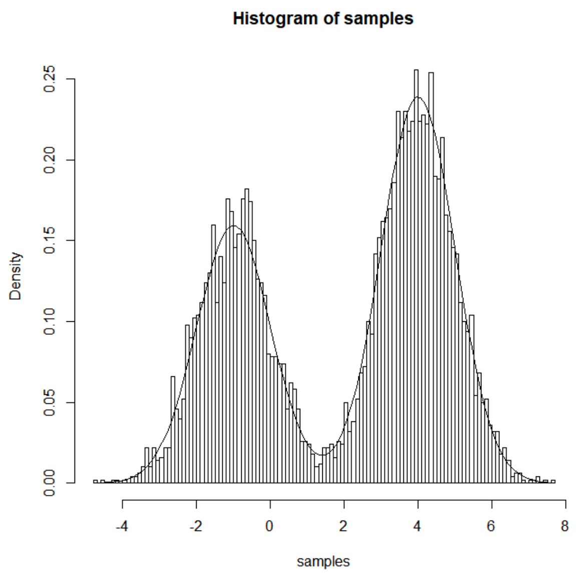
[Console] run as main [1.0020419819388395, 0.7284625736824267, 2.518246738857597, 1.624848857597545, -0.7871740210548127]


4. R 의 armspp library 를 사용하여 다음의 분포에서 adaptive rejection metropolis sampling 을 이용하여 sample 을 생성하라.

```
0.4 * N(-1, 1) + 0.6 * N(4, 1)
```

```
[R]
#(~HW2~)
#ARMS(adaptive rejection metropolis sampling) example
# Mixture of normals: 0.4 N(-1, 1) + 0.6 N(4, 1). Not log concave.

library(armspp)


dnormmixture <- function(x) {
    parts <- log(c(0.4, 0.6)) + dnorm(x, mean = c(-1, 4), log = TRUE)
    log(sum(exp(parts - max(parts)))) + max(parts)
    #overflow protect
}

#pdf 그려보자 (진짜 bimodal인지)
curve(exp(Vectorize(dnormmixture)(x)), xlim=c(-4,7))

#sample 뽑자
#이건 log-concave case가 아니므로, metropolis option을 true로 놓고 ARMS를 돌려야 함
samples <- arms(5000, dnormmixture, -1000, 1000)
hist(samples, freq = FALSE, nclass=100)
curve(exp(Vectorize(dnormmixture)(x)), add=TRUE)
```

Result:



5. Let X~Laplace(0,1). Find $E(X^2)$ using Importance Sampling.

10000 개의 sample 을 이용하겠다.

```
#python 3 file created by Choi, Seokjun

# when X~Laplace(0,1), calculate E(X^2)
# using Importance sampling with proposal pdf N(0,2^2)

from math import exp, pi, sqrt
from random import normalvariate, seed

class ImportanceSampler:
    def __init__(self, target_pdf, proposal_pdf, proposal_sampler):
        self.target_pdf = target_pdf
        self.proposal_pdf = proposal_pdf
        self.proposal_sampler = proposal_sampler

        self.sample = []
        self.weight = []

    def generate(self, num_samples_for_sim):
        unstandardized_weight_sum = 0
        for _ in range(num_samples_for_sim):
            proposed_sample = self.proposal_sampler()
            unstandardized_weight = self.target_pdf(proposed_sample) / self.proposal_pdf(proposed_sample)
            self.sample.append(proposed_sample)
            self.weight.append(unstandardized_weight)
            unstandardized_weight_sum += unstandardized_weight

        self.weight = [x/unstandardized_weight_sum for x in self.weight]

    def expectation(self, inner_func):
        expectation = 0
        for val, weight in zip(self.sample, self.weight):
            expectation += inner_func(val)*weight

        return expectation


if __name__ == "__main__":
    print('run as main')

    seed(2019-311-252)

    def Laplace01_pdf(x):
        return 0.5*exp(-abs(x))

    def squarefunc(x):
```

```
    return x**2

def normal02_sampler():
    return normalvariate(0,2)

def normal_pdf(x, mu, sigma):
    return (exp(-0.5*(x-mu)**2/(sigma**2))/(sqrt(2*pi)*sigma))

def normal02_pdf(x):
    return normal_pdf(x,0,2)



LaplaceExpectation = ImportanceSampler(Laplace01_pdf, normal02_pdf, normal02_sampler)
LaplaceExpectation.generate(10000)
result = LaplaceExpectation.expectation(squarefunc)
print(result) #should be near 2
```

Result:

(참고) analytic 한 답은 2 이다.

```
[Console]
run as main
2.004389046013196
```