2018 Fall

Computational Statistics
HW#6

182STG18 이하경

Ⅰ. Description

Numerical Integration

Newton-Cotes Rule (Riemann, Trapezoidal, Simpson's Rule)

HW5 에서는 1 차원(one-dimensional)의 적분 계산에 대해 유용하게 사용할 수 있는 수치적분 근사법에 대해 소개하고, Newton-Cotes Quadrature 인 Riemann, Trapezoidal, Simpson's rule 의 세 가지 근사법을 직접 구현해보았다. Newton-Cotes Quadrature 는 $[x_i, x_{i+1}]$ 내의 subinterval 들을 동일한 간격으로 나눈다는 조건이 있다. 다음의 식에 따라 각 subinterval 내에서 m 차 다항식으로 근사하여 계산한 상수와 피적분함수를 이용하여 전체 적분 값을 계산한다.

$$p_{ij}(x) = \prod_{k=0,k\neq j}^{m} \frac{x - x_{ik}^*}{x_{ij}^* - x_{ik}^*} \tag{5.8}$$

$$\rightarrow\ p_i(x) = \sum_{j=0}^{m} f(x_{ij}^*) p_{ij}(x)$$

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \int_{x_i}^{x_{i+1}} p_i(x)dx \tag{5.9}$$

$$= \sum_{j=0}^{m} f\left(x_{ij}^*\right) \int_{x_i}^{x_{i+1}} p_{ij}(x)dx \tag{5.10}$$

$$= \sum_{j=0}^{m} A_{ij} f\left(x_{ij}^*\right) \tag{5.11}$$

만약 적분 구간이 유한하지 않거나 Singularity 가 발생하여 적분 계산이 어려운 경우 변수의 변환을 이용해 적분식을 적절히 변환(transformation)하여 안정적인 적분계산을 할 수 있다. Implementation 1 과 2 에서는 HW5 에 이어 수치적분 근사법을 예제에 적용해보고, 변환을 이용한 적분 또한 계산해본다.

Romberg Integration

1 차원의 적분 계산 시 Newton-Cotes 방법은 상당히 유용하나, 근사다항식의 차수 m 이 작을 경우(eg. m=0, Riemman Rule) 값의 수렴이 느리다는 단점이 있다. Romberg Algorithm 은 Newton-Cotes 의 Trapezoidal Rule 을 이용해 더 나아가 좀 더 효율적이고 빠르게 계산할 수 있도록 한 방법으로 다음의 계산을 따른다.

Let $\hat{T}_{(n)}$ *trapezoidal rule estimate of* $\int_a^b f(x)dx$ *using n subintervals of equal length* $h = \frac{b-a}{n}$,

Then $\hat{T}_{(2n)} = \frac{1}{2}\hat{T}_{(2)} + \frac{h}{2}\sum_{i=1}^{n} f\left(a + \left(i - \frac{1}{2}\right)h\right) \rightarrow \frac{4\hat{T}_{(2n)} - \hat{T}_{(n)}}{3} = \int_a^b f(x)dx + O(n^{-4})$

$$\text{Define} \qquad \hat{T}_{0.0} = (b-a)\left[\frac{1}{2}f(a) + \frac{1}{2}f(b)\right] \ \ \& \ \ \hat{T}_{i.0} = \hat{T}_{(2^i)}\ for\ i = 0, \dots, m$$

$$\text{Relationship} \qquad \hat{T}_{i,j} = \frac{4^j \hat{T}_{i,j-1} - \hat{T}_{i-1,j-1}}{4^j - 1}\ for\ j = 1, \dots, i\ \&\ i = 1, \dots, m$$

이 때 $\hat{T}_{m,j} = \int_a^b f(x)dx + O\left(4^{-mj}\right)$, 4^{-mj}의 상대오차로 비교적 빠른 속도로 수렴한다.
또한 $Q_{i,j} = \frac{\hat{T}_{i,j} - \hat{T}_{i-1,j}}{\hat{T}_{i+1,j} - \hat{T}_{i,j}} \cong 4^{j+1}$ 이 i 가 증가할수록 성립하는 지 확인하여 적분 계산 값을 평가할 수 있다.

Implementation 3 에서는 Romberg Algorithm 을 이용하여 실제 적분 계산을 적용해본다.

Simualation & MC Integration

Rejection Sampling

Monte Carlo Simulation 은 random variable X ~ f(x), pdf 에서 관심함수 h(X)에 대한 기댓값 E[h(X)]을 계산하고자 할 때, target 분포 f(x)로부터 i.i.d. sample 을 생성하여 mean 과 variance 를 추정하는 방법이다. 만약 X 가 따르는 f(x)가 알려진 분포가 아니거나 복잡할 경우, 우리는 random sample 을 뽑기 위해 target 분포를 찾을 필요가 있다. 이 중 효과적인 방법 중 하나인 Rejection Sampling 방법은 다음의 알고리즘에 따라 정확한 target 분포로부터 쉽게 random draw 를 얻을 수 있다.

1. Sample $\mathrm{Y} \sim \mathrm{g}$
2. Sample $\mathrm{u} \sim \mathrm{Unif}(0,1)$
3. Reject Y if $\mathrm{u} > \frac{\mathrm{f(Y)}}{\mathrm{e(Y)}}$
4. Otherwise keep Y & Set X=y

여기서 g 는 sampling 이 가능한 다른 density 함수, e 는 envelope $\mathrm{e(X)} = \frac{\mathrm{g(X)}}{\alpha} \geq \mathrm{f(X)},\ \forall \mathrm{x}, \mathrm{f(x)} > 0$ 이다. 알려진 쉬운 density 로부터 sample Y 을 생성하고, random rejection 을 통해 Accept 와 Reject 를 결정하는 방법이다. Accect 된 X sample 은 실제 분포 f 를 따른다. * $\mathrm{P(X \leq y)} = \mathrm{P}\left[\mathrm{Y} \leq \mathrm{y} \,\middle|\, \mathrm{u} \leq \frac{\mathrm{f(Y)}}{\mathrm{e(y)}}\right] = \int_{-\infty}^{y} f(z)dz$

Implementation 4 와 5 에서는 Rejection Sampling 을 이용해 각각 Gamma 분포로부터의 Sampling, Prior 와 Likelihood 를 이용한 Bayesian Posterior 로부터의 Sampling 을 직접 수행해본다.

Ⅱ. Implementation

1. Derivation for Simpson's Rule (Problems 5.2)

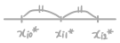
Goal 아래의 식에 따라 Simpsion's rule 에서의 $A_{ij},\ j=0,1,2$ 을 유도한다.

$$p_{ij}(x)=\prod_{k=0,k\neq j}^{m}\frac{x-x_{ik}^{*}}{x_{ij}^{*}-x_{ik}^{*}}$$

$$\int_{x_i}^{x_{i+1}}f(x)dx\approx\int_{x_i}^{x_{i+1}}p_i(x)dx=\sum_{j=0}^{m}f(x_{ij}^{*})\int_{x_i}^{x_{i+1}}p_{ij}(x)dx=\sum_{j=0}^{m}A_{ij}f(x_{ij}^{*})$$

Result

for Simpson's rule, $x_{i0}^{*}=x_i,\ x_{i1}^{*}=\frac{x_i+x_{i+1}}{2},\ x_{i2}^{*}=x_{i+1}\quad(m=2)$



(5.8) $p_{ij}(x)=\prod_{k=0,k\neq j}^{2}\left(\frac{x-x_{ik}^{*}}{x_{ij}^{*}-x_{ik}^{*}}\right),\ j=0,1,2\quad$ & $\quad p_i(x)=\sum_{j=0}^{2}f(x_{ij}^{*})\,p_{ij}(x)$

$\rightarrow p_{i0}(x)=\left(\frac{x-x_{i1}^{*}}{x_{i0}^{*}-x_{i1}^{*}}\right)\cdot\left(\frac{x-x_{i2}^{*}}{x_{i0}^{*}-x_{i2}^{*}}\right)=\frac{(2x-x_i-x_{i+1})(x-x_{i+1})}{(x_i-x_{i+1})^2}$

$p_{i1}(x)=\left(\frac{x-x_{i0}^{*}}{x_{i1}^{*}-x_{i0}^{*}}\right)\cdot\left(\frac{x-x_{i2}^{*}}{x_{i1}^{*}-x_{i2}^{*}}\right)=\frac{-4(x-x_i)(x-x_{i+1})}{(x_{i+1}-x_i)^2}$

$p_{i2}(x)=\left(\frac{x-x_{i0}^{*}}{x_{i2}^{*}-x_{i0}^{*}}\right)\cdot\left(\frac{x-x_{i1}^{*}}{x_{i2}^{*}-x_{i1}^{*}}\right)=\frac{(x-x_i)(2x-x_i-x_{i+1})}{(x_{i+1}-x_i)^2}=p_{i0}(x)$

(5.9) $\int_{x_i}^{x_{i+1}}f(x)\,dx\approx\int_{x_i}^{x_{i+1}}p_i(x)\,dx$

(5.10) $=\sum_{j=0}^{m}f(x_{ij}^{*})\int_{x_i}^{x_{i+1}}p_{ij}(x)\,dx$

(5.11) $=\sum_{j=0}^{m}A_{ij}(x)\,f(x_{ij}^{*})\quad$ & $\quad A_{ij}(x)=\int_{x_i}^{x_{i+1}}p_{ij}(x)\,dx,\ j=0,1,2$

$\rightarrow A_{i0}(x)=\int_{x_i}^{x_{i+1}}p_{i0}(x)\,dx$

$=\frac{1}{(x_{i+1}-x_i)^2}\int_{x_i}^{x_{i+1}}(2x-x_i-x_{i+1})(x-x_{i+1})\,dx=\frac{1}{(x_{i+1}-x_i)^2}\int_{x_i}^{x_{i+1}}\left[2x^2-(x_i+3x_{i+1})x+x_{i+1}(x_i+x_{i+1})\right]dx$

$=(\prime\prime)\quad\left[\frac{1}{3}x^3-\frac{1}{2}(x_i+3x_{i+1})x^2+x_{i+1}(x_i+x_{i+1})x\right]_{x_i}^{x_{i+1}}$

$=(\prime\prime)\quad\left[\frac{1}{3}(x_{i+1}^3-x_i^3)-\frac{1}{2}(x_i+3x_{i+1})(x_{i+1}^2-x_i^2)+x_{i+1}(x_i+x_{i+1})(x_{i+1}-x_i)\right]$

$=\frac{1}{6(x_{i+1}-x_i)^2}\left[2(x_{i+1}^3-x_i^3)-(3x_i+9x_{i+1}-6x_{i+1})(x_{i+1}^2-x_i^2)\right]$

$=(\prime\prime)\quad\left[2x_{i+1}^3-2x_i^3-3x_{i+1}^3+3x_{i+1}^2x_i-3x_{i+1}x_i^2+3x_i^3\right]$

$=\frac{1}{6(x_{i+1}-x_i)^2}\cdot(x_{i+1}-x_i)^3=\frac{1}{6}(x_{i+1}-x_i)\quad\cdots①$

$A_{i1}(x)=\int_{x_i}^{x_{i+1}}p_{i1}(x)\,dx$

$=\frac{-4}{(x_{i+1}-x_i)^2}\int_{x_i}^{x_{i+1}}(x-x_{i+1})(x-x_i)\,dx=\frac{-4}{(x_{i+1}-x_i)^2}\cdot\left[-\frac{1}{6}(x_{i+1}-x_i)^3\right]=\frac{2}{3}(x_{i+1}-x_i)\quad\cdots②$

$A_{i2}(x)=A_{i0}(x)=\frac{1}{6}(x_{i+1}-x_i)\quad\cdots③$

$(\because\ p_{i0}(x)=p_{i2}(x))$

$\rightarrow\int_{x_i}^{x_{i+1}}f(x)\,dx\cong\sum_{j=0}^{2}A_{ij}f(x_{ij}^{*})=\frac{1}{6}(x_{i+1}-x_i)\left[f(x_i)+f(x_{i+1})\right]+\frac{2}{3}(x_{i+1}-x_i)\cdot f\left(\frac{x_i+x_{i+1}}{2}\right)$

$=\frac{1}{6}(x_{i+1}-x_i)\left[f(x_i)+4f\left(\frac{x_i+x_{i+1}}{2}\right)+f(x_{i+1})\right]$

2. Bayesian Estimation of μ (Problems 5.3)

Obs	(6.52, 8.32, 0.31, 2.82, 9.96, 0.14, 9.64)
Prior	$\mu \sim \text{Cauchy}(5, 2)$
Likelihood	$\bar{x} \mid \mu \sim \text{N}(\mu, 3^2/7)$
Posterior	$\pi(\mu\|\bar{x}) = \frac{\pi(\mu)L(\mu\|\bar{x})}{\int_{-\infty}^{\infty}\pi(\mu)L(\mu\|\bar{x})d\mu} = k \cdot (prior) \cdot (likelihood)$

a. Goal

수치적분법을 이용해 사후분포의 비례상수 $\text{k} = \left[\int_{-\infty}^{\infty}\pi(\mu)\cdot L(\mu|\bar{x})\,d\mu\right]^{-1}$ 을 계산하고 실제 값인 7.84654 와 비교해본다. ($\int k \times prior \times likelihood\, d\mu = 1$)

Result

method	value	subintervals	niter	comp.time
Riemann	7.84654	1024	10	0.0089
Trapezoidal	7.84654	1024	10	0.0088
Simpson	7.84654	2048	11	0.0109
Integrate	7.84654			0.0003

세가지 수치적분법을 이용하여 적분 구간은 모두 density 의 값이 0 에 수렴하는 [-100, 100] 까지로 계산하였다. Error 의 threshold E=10^-8 으로 설정하고 값을 확인하였을 때 소수 다섯째자리 내에서 세 가지 방법 모두 실제 값인 7.84654 를 정확히 계산하였고, 반복 수와 소요 시간 또한 세 가지 방법이 거의 동일하게 빨랐다.

b. Goal

(a)에서의 값 7.84654 를 이용해 사후확률 $\text{P}(2 \le \mu \le 8 \mid \bar{x}) = 7.84654 \cdot \int_2^8 \pi(\mu)\cdot L(\mu|\bar{x})\,d\mu$ 을 세 가지 수치적분법으로 계산하고 실제 값 0.99605 와 비교해본다.

Error 의 계산 시 Relative Convergence Criterion $\frac{|P^k - P^{k-1}|}{|P^{k-1}|} < \epsilon = 0.0001$ 을 사용한다.

Result

method	threshold E	value	subintervals	niter	comp.time
Riemann	10^-4	0.99596	256	8	0.0083
	10^-5	**0.99605**	4096	12	0.0189
Trapezoidal	10^-4	0.99603	64	6	0.0050
	10^-5	**0.99605**	256	8	0.0120
Simpson	10^-4	**0.99605**	16	4	0.0042
Integrate		0.99605			0.0001

먼저 threshold 를 0.0001 로 하였을 때 각각의 방법에서 계산된 결과를 확인해보면 Simpson's rule 만이 실제 값과 일치하였고 Riemann rule 의 경우 실제 값과 0.00009 만큼, Trapezoidal rule 의 경우 0.00002 만큼 오차가 발생하였다. 정확한 값을 계산해보기 위해 threshold 를 0.00001(=10^-5)으로 조정하고 다시 실행하였을 때 두 방법 모두 실제

값인 0.99605 와 일치하였다. 적분 시의 반복 수와 계산 속도는 Simpson's rule 에서 가장 빠르고 안정적이었으며 Rieman rule 이 가장 느렸다.

c. Goal

사후확률 $\mathbf{P(\mu \geq 3 \mid \bar{x})} = 7.84654 \cdot \int_3^{\infty} \pi(\mu) \cdot L(\mu|\bar{x})\, d\mu$ 을 계산하고 실제 값 0.99086 과 비교한다. 무한대의 적분 구간에 대해 transformation $\mathbf{u = exp(\mu)/(1 + exp(\mu))}$ 을 이용해 식을 변환하여 계산한다.

Result

method	value	subintervals	niter	comp.time
Riemann	0.99086	2097152	21	0.8432
Trapezoidal	0.99086	4096	12	0.0122
Simpson	0.99086	8192	13	0.0174

적분의 변환을 통해 구하는 사후확률의 식은 다음과 같이 표현할 수 있다.

$$\mathrm{P}(\mu \geq 3 \mid \bar{x}) = \int_{e^3/(1+e^3)}^{1} \pi\left(\log(\frac{u}{1-u})\right) \cdot L\left(\log\left(\frac{u}{1-u}\right) \middle| \bar{x}\right) \cdot \frac{1}{u(1-u)}\, du$$

위의 적분을 계산하기 위해 수치적분법을 이용할 경우 1 에서의 singularity 가 발생하므로, 이를 제외하기 위해서 적분의 상한을 1-10^-8 으로 조정하여 세 가지 수치적분 방법을 이용해 확률 값을 계산하였다. E=10^-8 하에서 계산된 확률의 값은 소수 다섯째자리 내에서 모두 실제 값과 동일하였고, 계산 속도는 Trapezoidal, Simpson 에 비해 Riemann rule 이 상당히 느렸다. 반복 수 증가에 따른 subinterval 의 수 n 이 2 의 배수의 속도로 증가함에 따라 Riemann rule 에서의 subinterval 의 수가 매우 큰 것을 확인하였다.

d. Goal

(c)의 적분 값을 transformation $\mathbf{u = 1/\mu}$ 을 이용해 계산한다.

Result

method	k	subintervals	niter	comp.time
Riemann	0.99086	4194304	22	2.5261
Trapezoidal	0.99086	4096	12	0.0211
Simpson	0.99086	256	8	0.0118

d 의 적분 변환을 통해 구하는 사후확률의 식은 다음과 같다.

$$\mathrm{P}(\mu \geq 3 \mid \bar{x}) = \int_{1/3}^{0} \pi\left(\frac{1}{\mathrm{u}}\right) \cdot L\left(\frac{1}{u} \middle| \bar{x}\right) \cdot \left(-\frac{1}{u^2}\right) du$$

이 계산 시에도 역시 0 에서의 singularity 를 해결하기 위해 적분구간을 0+10^-8 으로 조정하였다. 세 가지 적분법 모두 E=10^-8 하에서 적절한 값을 계산하였고 계산속도는 Simpson < Trapezoidal < Simpson 순으로 Simpson's rule 을 이용한 계산이 가장 효율적이었다.

3. Integration using Romberg's Algorithm (Problems 5.4)

Goal

X 가 $\mathrm{Unif}[1,a]$를 따르고 $Y=(a-1)/X$, $a>1$일 때, $E[Y]=\log(a)$를 m=6 인 Romberg's Algorithm 을 이용하여 계산한다. (임의의 a 값으로 10 을 지정하였다. 이때 실제 E(Y) 값은 $\log(10)=2.30256$이다.)

Result. Triangular array $\hat{T}_{i,j}$ $(m=6)$

$\hat{T}_{i,j}$	0	1	2	3	4	5	6
0	4.95000000						
1	3.29318182	2.74090909					
2	2.62922118	2.58495714	2.57456034				
3	2.39773710	2.39406275	2.39103268	2.38811954			
4	2.32795210	2.32767844	2.32741811	2.32716864	2.32692962		
5	2.30906066	2.30904219	2.30902397	2.30900599	2.30898824	2.30897070	
6	2.30421333	2.30421215	2.30421097	2.30420980	2.30420862	2.30420746	2.30420629

테이블의 가장 마지막 행(m=6)을 확인하였을 때 $\hat{T}_{6,6}=2.30421$로 실제 값과 약 0.00162 정도의 오차가 발생하여 적분 값이 타당하다고 판단하였다. m 의 크기를 증가시킨다면 오차의 크기가 더 줄어들 것으로 예상하였다.

Result. $Q_{i,j}$

i	subintervals	0	1	2	3	4
1	2					
2	4	2.49535604				
3	8	2.86827769	0.8169540			
4	16	3.31710413	2.875595131	2.88499426		
5	32	3.69399879	3.56210660	3.45841565	3.35583778	
6	64	**3.89729691**	3.85840634	3.82176084	3.78688681	3.75373126

j=0 일 때 i 가 증가함에 따라 $Q_{i,j}$ 가 점점 $4^{j+1}=4$ 에 가까워지는 것을 확인하였다.

4. Gamma Derivates (Example 6.1)

Goal.

Rejection Sampling Algorithm 을 따라 $\mathrm{Gamma}(\mathrm{r},1)$, $\mathrm{r} \geq 1$으로부터 n 개의 sample 에서 random variable 들을 생성해보고 r=1, ..., 4 에 따른 acceptance rate 를 확인한다. 또한 R 의 rgamma 함수를 이용해 생성한 random sample 과 Q-Q plot 을 그려 결과를 확인한다.

$$\mathrm{Y} \sim \text{target dist}^{\mathrm{n}}\ \mathrm{f(y)} = \frac{\mathrm{t(y)^{r-1}t'(y)e^{-t(y)}}}{\Gamma(\mathrm{r})}, \qquad t(y) = a(1+by)^3 > 0\ for\ -\frac{1}{b} < y < \infty,\ \ \mathrm{a} = \mathrm{r} - \frac{1}{3},\ \ b = \frac{1}{\sqrt{9a}}$$

then $\mathrm{X} = \mathrm{t(Y)} \sim Gamma(r,1)$

g(Y)를 f(Y)에 비례하는 단순화된 density, $\mathrm{e(Y)} = \mathrm{e}^{-\frac{y^2}{2}} \sim N(0,1)$ envelope 로 두었을 때 다음의 과정을 따라 random sampling 을 진행한다.

1. n 개의 $\mathrm{Z} \sim \mathrm{N}(0,1)$ & $\mathrm{u} \sim \mathrm{Unif}(0,1)$ sample 을 생성한다.
2. $\mathrm{u} <= \mathrm{g(Z)/e(Z)} = \exp\left[\frac{\mathrm{Z}^2}{2} + \mathrm{a}\log\frac{t(Z)}{a} - \mathrm{t(Z)} + \mathrm{a}\right]$ 일 경우 Z 를 Accept 하고, 아닐 경우 Reject 한다.

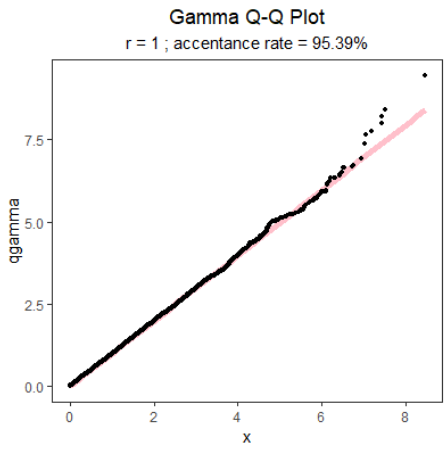
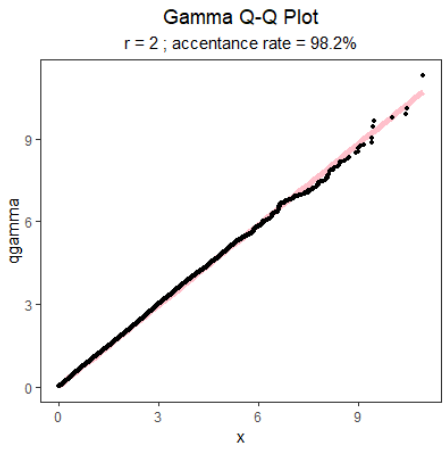
n=10000 개의 Z 와 U 의 sample 을 생성하여 이용하였다.

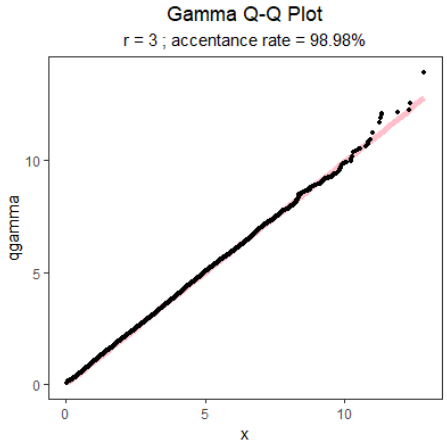
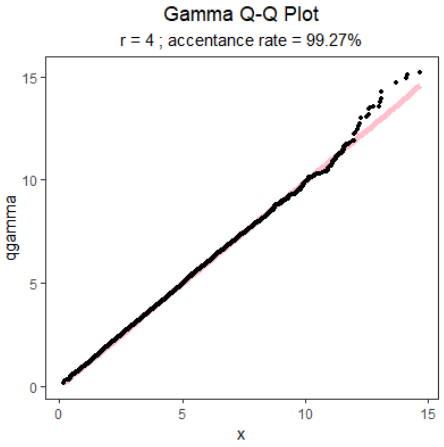
Result 1. Acceptance Rate by shape r

r	acceptance rate
1	95.39%
2	98.20%
3	98.98%
4	99.27%

r 에 따른 acceptance rate 를 확인해보면 r=4 일 때 Z 로부터 accept 된 비율이 99% 이상으로 매우 높았다. 비율이 가장 작은 r=1 에서도 95% 이상이 유지되어 전체적으로 매우 좋은 결과를 보였다.

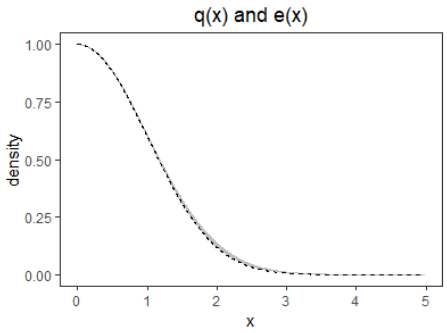
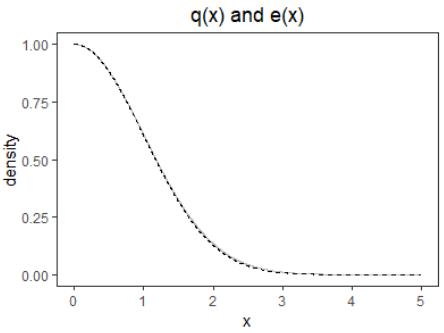
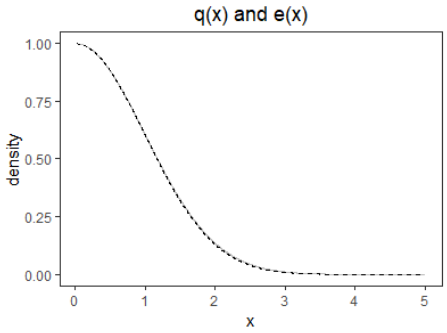
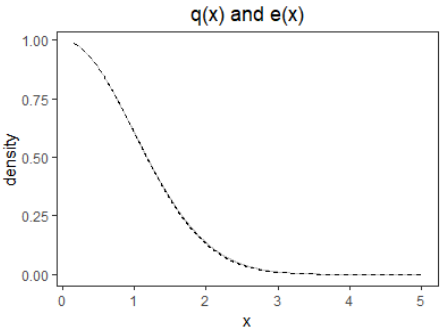
Result 2. Gamma Q-Q Plot of X=t(Z) vs random gamma sample

r=1	r=2
	

r=3	r=4
	

Accept된 Z를 이용한 X=t(Z)와 R의 rgamma를 이용해 실제 r값에 따른 Gamma(r, 1) 분포에서 random sample을 생성한 뒤 Q-Q plot 을 통해 어느 정도 일치하는 지 확인하였다. 대부분 모두 직선을 따르며 r 이 커짐에 따라 acceptance rate가 증가하였으므로 직선과 가까운 점들이 점점 더 많아지는 것을 확인할 수 있다.

Result 3. q(Z) & envelope e(Z)

r=1	r=2
	
r=3	**r=4**
	

추가적으로 q(Z)와 e(Z)의 그래프를 그려봤을 때 거의 겹쳐 그려지는 것을 확인할 수 있었다. 이 예제의 경우 rejection region이 굉장히 작아 e(Z)가 좋은 envelope라고 할 수 있다.

5. Sampling a Bayesian Posterior (Example 6.2)

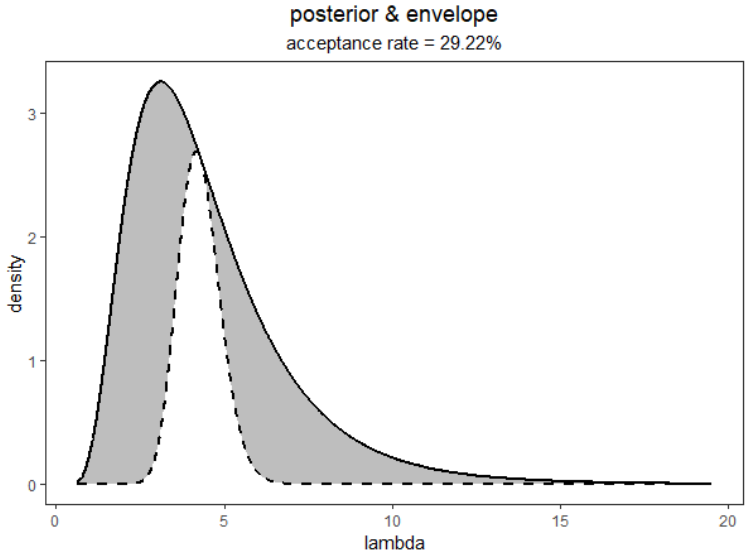
Obs	(8, 3, 4, 3, 1, 7, 2, 6, 2, 7)
Prior $f(\lambda)$	$\log\lambda \sim \mathrm{N}(\log 4, 0.5^2)$
Likelihood $L(\lambda\|X)$	$X \mid \lambda \sim Poission(\lambda)$
unnormalized Posterior	$q(\lambda\|X) \propto f(\lambda) \cdot L(\lambda\|X)$

Goal

Rejection Sampling 방법으로 Bayesian Posterior Sample 을 생성하려고 한다.
Log-Normal Prior 에서 생성된 n 개의 λ_i 와 $U_i \sim U(0,1)$으로부터 $q(\lambda|X)/e(\lambda|X) = L(\lambda|X)/L(\hat{\lambda}^{MLE} = 4.3|X)$을 이용해 λ의 posterior sample 을 생성하고 acceptance rate 를 확인해본다. 또한 λ 에 따른 $q(\lambda|X)$와 $e(\lambda|X)$의 그래프를 직접 그려본다.

초기에 n=10000 개의 λ_i 와 U_i sample 을 생성하여 알고리즘에 이용하였다.

Result



이 예제의 경우 10000 개의 λ_i 로부터 accepted 된 비율이 29.22%로 약 30%에 미치지 않았다. q(λ)와 e(λ) 의 그래프를 통해서도 accept. region 과 reject. region 의 크기를 확인할 수 있다. 그래프에서 shaded area, 즉 reject 되는 부분이 매우 커 좋은 envelope 가 될 수 없다고 판단하였다. 만약 target distribution 인 posterior 로부터 1000 개의 random sample 을 생성하고자 할 경우, 약 1000/0.3 ≅ 3422 개의 초기 sample 을 뽑아야 하므로 비효율적일 수 있다.

Ⅲ. Discussion

Newton Cotes Rule 은 1 차원의 적분 계산에서 훌륭한 퍼포먼스를 보인다. 본 과제의 Implementation 2 에서 Bayesian Posterior 함수를 이용한 적분 계산에 세 가지 근사법을 이용하였을 때 대부분의 값들이 정확히 실제 값과 일치하였음을 통해 상당히 훌륭한 방법임을 확인하였다. 대부분의 경우 근사다항식의 차수가 높아질수록, 즉 Simpson < Trapezoidal< Riemann 순으로 계산의 수렴 속도가 빨랐다. 특히 m=0 인 Riemann Rule 의 경우는 수렴 속도가 느리다는 Newton Cotes 의 단점을 깨달을 수 있었는데, Error 의 threshold 값을 매우 작게 설정하였을 때 subinterval 의 수가 기하급수적으로 늘어나 최종 값을 계산하는 데 상당히 오랜 시간이 걸렸으며 어떤 경우는 R 에서 계산을 완료하지 못하기도 하였다. 그러나 threshold 를 10^-05 정도로 하였을 때는 모두 안정적으로 적분 값을 계산하였으므로 합리적이라고 생각하였다.

Implementation 3 에서 Romberg 알고리즘을 몇 가지 step 을 이용해 빠르게 적분 값 log(a)을 계산하였고, 근사시 m 의 크기가 작아도 비교적 Newton-Cotes 에 비해 빠르게 합리적인 적분 값을 계산할 수 있다는 것을 알게 되었다.

통계학에서 많은 경우 random variable X 와 관심함수 h(X)에 대하여 기댓값을 계산하여 추정에 이용하는데, 이를 위한 접근 방법은 크게 수치적분과 MC Simulation 으로 나눌 수 있다. 위에서 확인하였듯 1 차원의 적분에서는 수치적분이 매우 정확하고 훌륭하나, 다차원의 적분에서는 Simulation 을 이용한 기댓값의 추정이 효과적일 수 있다. Simulation 을 위해서는 관심 분포로부터의 적절한 iid sample 을 뽑는 것이 매우 중요한데, Implementation 4 와 5 에서 Rejection Sampling 을 이용해 실제 simulation 으로 간단하게 random sample 을 효과적으로 생성할 수 있음을 알게 되었다. 또한 버려지는 부분을 줄이고 acceptance rate 을 높이기 위해서는 좋은 envelope 를 설정하는 것이 중요함을 알게 되었다.

[Appendix] R Code

 # functions

```
# Riemann
  myrieman <- function(f, a, b, start=1, E=10^-8)
  {
    niter = 0 ; maxiter = 100 ; error = 1

    # initial value
    R = (b-a)*f(a)
    k <- start

    while (error >= E & niter <= maxiter)
    { R_0 <- R

    n <- 2^k ; h <- (b-a)/n
    i <- 0:(n-1)
    R <- h * sum( f(a+i*h) )

    error <- abs(R - R_0) / abs(R_0 + 10^-3)
    niter <- niter + 1
    k <- k + 1

    print(paste("error = ", error, " niter = ", niter, sep = ""))
    }

    return(R)
  }


# Trapezoidal
  mytrapezoid <- function(f, a, b, start=1, E=10^-8)
  {
    niter = 0 ; maxiter = 100 ; error = 1

    # initial value
    t = (b-a)*f(a)
    k <- start

    while (error >= E & niter <= maxiter)
    { t_0 <- t

    n <- 2^k ; h <- (b-a)/n
    i <- 1:(n-1)
    t <- h/2*f(a) + h*sum( f(a+i*h) ) + h/2*f(b)

    error <- abs(t - t_0) / abs(t_0 + 10^-3)
    niter <- niter + 1
    k <- k + 1

    print(paste("error = ", error, " niter = ", niter, sep = ""))
    }

    return(t)
  }
```

```
# Simpson
  mysimpson <- function(f, a, b, start=1, E=10^-8)
  {
    niter = 0 ; maxiter = 100 ; error = 1

    # initial value
    S = (b-a)*f(a)
    k <- start

    while (error >= E & niter <= maxiter)
    { S_0 <- S

    n <- 2^k ; h <- (b-a)/n
    i <- 1:(n/2)
    S <- (h/3) * sum( f(a+(2*i-2)*h) + 4*f(a+(2*i-1)*h) + f(a+(2*i)*h) )

    error <- abs(S - S_0) / abs(S_0 + 10^-3)
    niter <- niter + 1
    k <- k + 1

    print(paste("error = ", error, " niter = ", niter, sep = ""))
    }

    return(S)
  }
```

 # 5.3 Bayesian Estimation

```
# prior : mu ~ Cauchy(5, 2)
# likelihood : xbar ~ N(mu, 3^2/7)
  data1 <- c(6.52, 8.32, 0.31, 2.82, 9.96, 0.14, 9.64)

# a) -----
  prop <- function(mu) {
    prior <- dcauchy(mu, 5, 2)
    likelihood <- dnorm(mean(data1), mu, 3/sqrt(7))
    return(prior*likelihood)
  }

# true
  k <- 1/integrate(prop, -Inf, Inf)$value ; k

# numerical int.
  k1 <- 1/myrieman(prop, -100, 100) # system.time( for (i in 1:100) myrieman(prop, -100, 100) )
  k2 <- 1/mytrapezoid(prop, -100, 100) # system.time( for (i in 1:100) mytrapezoid(prop, -100, 100) )
  k3 <- 1/mysimpson(prop, -100, 100) # system.time( for (i in 1:100) mysimpson(prop, -100, 100) )
  round(c(k1, k2, k3), 5)

# b) -----
  post <- function(mu) { prop(mu) * 7.84654 }

  b <- integrate(post, 2, 8)$value
  b1 <- myrieman(post, 2, 8, E = 0.0001) # system.time( for (i in 1:100) myrieman(post, 2, 8, E = 0.0001) )
  b2 <- mytrapezoid(post, 2, 8, E = 0.0001) # system.time( for (i in 1:100) mytrapezoid(post, 2, 8, E = 0.0001) )
  b3 <- mysimpson(post, 2, 8, E = 0.0001) # system.time( for (i in 1:100) mysimpson(post, 2, 8, E = 0.0001) )
  round(c(b1, b2, b3), 5)

  b11 <- myrieman(post, 2, 8, E = 0.00001) # system.time( for (i in 1:100) myrieman(post, 2, 8, E = 0.00001) )
```

```
b21 <- mytrapezoid(post, 2, 8, E = 0.00001) # system.time( for (i in 1:100) mytrapezoid(post, 2, 8, E = 0.00001) )
round(c(b11, b21), 5)

# c) -----
u <- function(mu) { exp(mu) / (1 + exp(mu)) }
transf <- function(u) { post(log(u/(1-u))) * 1/(u*(1-u)) }

c1 <- myrieman(transf, u(3), 1-1e-08) # system.time( for (i in 1:100) myrieman(transf, u(3), 1-1e-08) )
c2 <- mytrapezoid(transf, u(3), 1-1e-08) # system.time( for (i in 1:100) mytrapezoid(transf, u(3), 1-1e-08) )
c3 <- mysimpson(transf, u(3), 1-1e-08) # system.time( for (i in 1:100) mysimpson(transf, u(3), 1-1e-08) )
round(c(c1, c2, c3), 5)
integrate(transf, u(3), 1)

# d) -----
u2 <- function(mu) { 1/mu }
transf2 <- function(u) { post(1/u) * (-1/u^2) }

d1 <- myrieman(transf2, u2(3), 0) # system.time( for (i in 1:100) myrieman(transf2, u2(3), 0) )
d2 <- mytrapezoid(transf2, u2(3), 0+1e-08) # system.time( for (i in 1:100) mytrapezoid(transf2, u2(3), 0+1e-08) )
d3 <- mysimpson(transf2, u2(3), 0+1e-08) # system.time( for (i in 1:100) mysimpson(transf2, u2(3), 0+1e-08) )
round(c(d1, d2, d3), 5)
integrate(transf2, u2(3), 0)
```

 # 5.4 Romberg

```
myromberg <- function(f, a, b, k)
  { n <- 2^k ; h <- (b-a)/n
    i <- 1:(n-1)
    t <- h * ( f(a)/2 + ifelse(k==0, 0, sum(f(a+i*h)))  + f(b)/2 )
    return(t)
  }

  a = 10
  f <- function(x) { 1/x }

  m = 6
  result <- matrix(nrow = m+1, ncol = m+1)
  for (i in 0:m) { result[i+1, 1] <- myromberg(f, (a-1)/a, a-1, i) }
  for (i in 1:m) { for (j in 1:i) { result[i+1, j+1] <- (4^i*result[i+1, j] - result[i, j]) / (4^i - 1) } }
  round(result, 8)
```

 # 6.1 Gamma Derivates

```
  tfunc <- function(y, r) { a = r-1/3 ; b = 1/sqrt(9*a) ; t = a*(1+b*y)^3 ; return(t) }
  qfunc <- function(y, r) { a = r-1/3 ; q = exp( a*log(tfunc(y, r)/a) - tfunc(y, r) + a ) ; return(q) }
  efunc <- function(y) { exp(-y^2/2) }

  n = 10000 ; r = 1
  set.seed(0) ; z <- rnorm(n) ; u <- runif(n)

  x <- tfunc(z, r) ; u <- u[x > 0] ; z <- z[x > 0] ; x <- x[x > 0]

  accept <- (u <= qfunc(z, r)/efunc(z)) ; sum(accept)/n
  z.keep <- z[accept]
  x.keep <- tfunc(z.keep, r)

  set.seed(0) ; qq <- data.frame(x = sort(x.keep), qgamma = sort(rgamma(length(x.keep), r, 1)))
  ggplot(qq) + theme_test() + theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5)) +
    geom_smooth(aes(x, qgamma), method = "lm", size = 2, se = F, color = "pink") + geom_point(aes(x, qgamma), size = 1) +
```

```
  labs(title = "Gamma Q-Q Plot", subtitle = paste("r = ", r, " ; accentance rate = ", sum(accept)/n*100, "%", sep = ""))

gg <- data.frame(x, qx = qfunc(x, r), ex = efunc(x))
ggplot(filter(gg, x <= 5)) + theme_test() +
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5)) +
  geom_area(aes(x, ex), fill = "gray") + geom_area(aes(x, qx), fill = "white") +
  geom_line(aes(x, ex), color = "gray") + geom_line(aes(x, qx), linetype = "dashed") +
  labs(y = "density", title = "q(x) and e(x)")
```

 # 6.2 Bayesian Sampling

```
data2 <- c(8, 3, 4, 3, 1, 7, 2, 6, 2, 7)
Lfunc <- function(lambda)
{ lik = c()
  for (i in 1:length(lambda)) { lik[i] <- prod(dpois(data2, lambda[i])) }
  return(lik)
}

n = 10000
set.seed(0) ; l <- rlnorm(n, log(4), 0.5) ; u <- runif(n)

accept <- (u <= Lfunc(l)/Lfunc(4.3)) ; sum(accept)/n

gg <- data.frame(l, prior = dlnorm(l, log(4), 0.5)) %>% mutate(envelope = prior*Lfunc(4.3)*10^11)
for (i in 1:nrow(gg)) { gg[i, "target"] = gg[i, "prior"]*Lfunc(gg[i, "l"])*10^11 }
ggplot(filter(gg, l <= 20)) +
  theme_test() + theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5), legend.position = "") +
  geom_area(aes(l, envelope, fill = "envelope")) + geom_area(aes(l, target, fill = "target")) +
  geom_line(aes(l, envelope), size = 1) + geom_line(aes(l, target), size = 1, linetype = "dashed") +
  scale_fill_manual("", values = c(envelope = "gray", target = "white")) +
  labs(title = "posterior & envelope", x = "lambda", y = "density",
       subtitle = paste("acceptance rate = ", sum(accept)/n*100, "%", sep = ""))
```