2019-10-30

computational Statistics

HW#5

192STG11 우나영

computational Statistics

HW#5

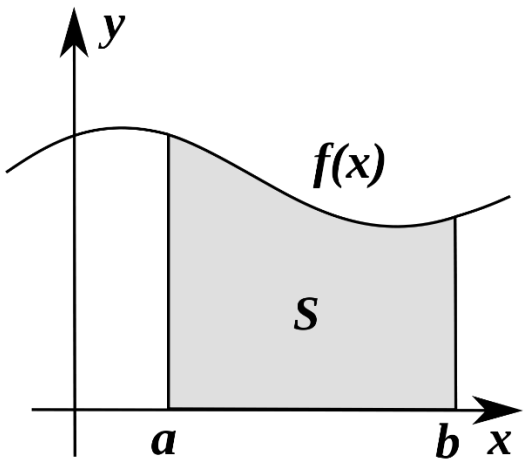
1. Problem

EM Algorithm for Multinomial

유전자 빈도(gene frequency)는 하나의 특정한 집단 안에 특정한 유전자를 가진 생물이 얼마나 존재하는지 나타내는 정도이다. 두 개 이상의 유전자라는 outcome 이 있으며 각각의 outcome 에 대응하는 확률이 있으므로 유전자 빈도는 multinomial 을 따른다. 그러나 개체가 갖는 특정 유전자의 조합인 유전자형(genotype)이 다르더라도 실제로 관찰되는 표현형(phenotype)은 같을 수 있다. 예를 들면 우성(dominant)형질이 열성(inferior)형질과 결합되면 우성 형질만 표현형으로 발현된다. 따라서 유전자형의 빈도를 알 수 없으므로 유전자 빈도 데이터를 추정하기위해 complete-incomplete 데이터 셋으로 set up 해 EM algorithm 을 이용할 수 있다.

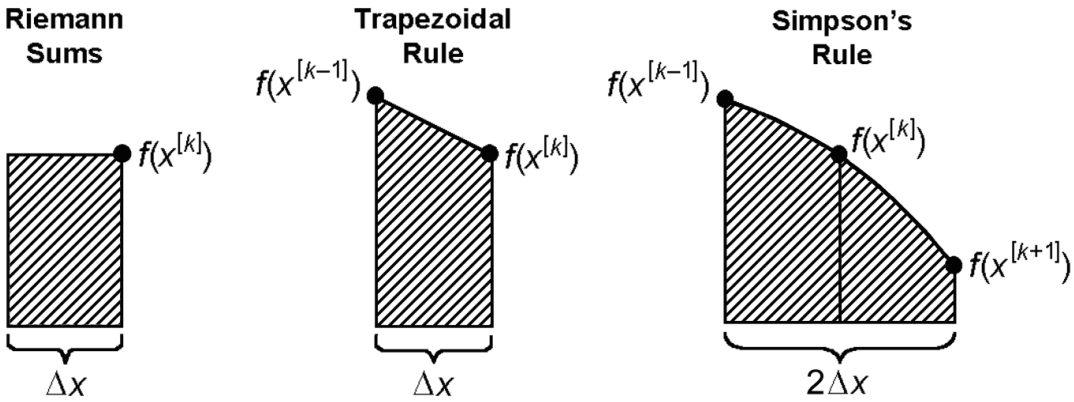
Numerical Integration: Riemann, Trapezoidal, and Simpson's

Expected value 는 모든 가능한 outcome 들을 각각의 확률로 가중평균한 값으로 예측을 위한 중요한 통계량이다. 또한 Expected value 는 integration 으로 정의되므로 Integration 을 계산하는 것은 통계학에서 중요한 문제이다. 아래의 그림에서 면적 S 를 구하는 문제를 생각해보자.



면적 S 는 $S = \int_a^b f(x)\,dx$로 정의되기 때문에 [a, b] 구간에서 integration 을 계산하면 된다. 하지만 실제로는 analytic solution 이 없는 경우가 많기 때문에 numerical approximation 이 필요하다. Numerical approximation 을 위해 일단 [a, b] 구간을 n 개의 subinterval $[x_i, x_{i+1}]$로 나눈다. 이때 $i =$

$0, \dots, n-1$이며 $x_0 = a,\ x_n = b$이다. 그 결과 전체 integration 은 subinterval 의 integration 합이 되므로 $\int_a^b f(x)\,dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx$ 로 표현할 수 있다. 각각의 subinterval 을 다시 m+1 개의 node $x_{ij}^*\ for\ j = 0, \dots, m$로 나눠 polynomial 함수로 approximation 시킨다. 이때 polynomial 함수는 항상 integration 의 analytic solution 이 존재한다. 각각의 subinterval 은 $\int_{x_i}^{x_{i+1}} f(x)\,dx \approx \sum_{j=0}^{m} A_{ij} f(x_{ij}^*)\ for\ some\ constant\ A_{ij}$로 근사한다. 노드의 개수(m+1)에 따라 numerical approximation 방법은 3 가지 Riemann, Trapezoidal, 그리고 Simpson's 로 나뉜다.



Numerical Integration by Riemann

$m = 0$

$x_{i0}^* = x_i, \qquad A_{i0} = x_{i+1} - x_{i0}^*\ approximate\ f\ to\ a\ constant\ function$

$$\int_{x_i}^{x_{i+1}} f(x)dx \cong \int_{x_i}^{x_{i+1}} f(x_i)dx = (x_{i+1} - x_i) f(x_i)$$

$Suppose\ x_i\ are\ equally\ spaced\ with\ subinverval\ length\ h = \frac{b-a}{n}$

$$\int_a^b f(x)dx \approx h \sum_{i=0}^{n-1} f(a+ih) = \widehat{R(n)}$$

Numerical Integration by Trapezoidal

$m = 1$

$x_{i0}^{*} = x_i, x_{i1}^{*} = x_{i+1},\ A_{i0} = A_{i1} = \frac{x_{i+1} - x_i}{2}$

$$\int_a^b f(x)dx \cong \sum_{i=0}^{n-1} \left(\frac{x_{i+1} + x_i}{2}\right)(f(x_i) + f(x_{i+1}))$$

$Suppose\ x_i\ are\ equally\ spaced\ with\ subinverval\ length\ h = \frac{b-a}{n}$

$$\int_a^b f(x)dx \approx \frac{h}{2}[f(a) + f(b)] + h\sum_{i=1}^{n-1} f(a+ih) = \widehat{T(n)}$$

Numerical Integration by Simpson's

$m = 2$

$x_{i0}^{*} = x_i, x_{i1}^{*} = \frac{x_i + x_{i+1}}{2}, x_{i2}^{*} = x_{i+1},\ A_{i0} = A_{i2} = \frac{x_{i+1} - x_i}{6}, A_{i1}$
$= 2(A_{i0} + A_{i2})$

$approximate\ f\ to\ a\ quadratic\ function$

$$\int_{x_i}^{x_{i+1}} f(x)dx \cong \frac{x_{i+1} + x_i}{6}\left[f(x_i) + 4f\left(\frac{x_{i+1} + x_i}{2}\right) + f(x_{i+1})\right]$$

$Suppose\ x_i\ are\ equally\ spaced\ with\ subinverval\ length$
$= \frac{b-a}{n}\ where\ n\ is\ even$

$$\int_a^b f(x)dx \approx \frac{h}{3}\sum_{i=1}^{n/2}(f(a + (2i-2)h) + 4f(a + (2i-1)h)$$
$$+ f(a + (2i)h) = \widehat{S(\frac{n}{2})}$$

Bayesian Inference, Singularity, and Transformation

근사 적분(approximation of integrals)은 베이지안 추론에서 복잡하고 익숙하지 않은 사후분포(posterior distribution)의 적분을 구할 때 유용하게 사용된다. 베이지안 추론은 관심 모수 θ에 대한 사전 분포 π(θ)와 주어진 θ 하에 $X = x$가 관측되었을 때의 조건부 밀도 함수인 우도 함수 $f(x|\theta)$로부터 베이즈 정리를 이용하여 관심 모수 θ의 사후분포 $\pi(\theta|x)$를 구한다.

$\int_0^1 \frac{e^x}{\sqrt{x}} dx$를 풀어보자. 그러나 이는 $x = 0$에서 정의되지 않아 Singularity 가 발생한다. 따라서 이를 해결하기 위해 x를 적분에 알맞은 형태로 변수 변환(transformation)해줘야 한다. 이렇듯 변수변환은 적분의 범위 내에서 변수가 정의되지 않거나 적분 범위가 무한대(infinity)일 때 사용할 수 있다. 따라서 이 예제는 아래와 같이 풀 수 있다.

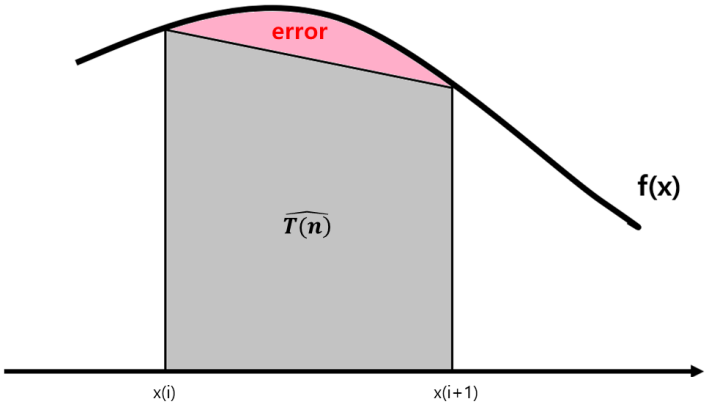
$$Let\ u = \sqrt{x}, \qquad du = \frac{1}{2\sqrt{x}} dx, \qquad 2du = \frac{1}{\sqrt{x}}$$

$$\therefore \int_0^1 \frac{e^x}{\sqrt{x}} dx = 2\int_0^1 e^{u^2} du$$

위와 같은 개념들을 응용해 예제 5.3 의 베이지안 추론을 통해 구한 사후분포의 infinity 적분 문제를 풀어보도록 하겠다.

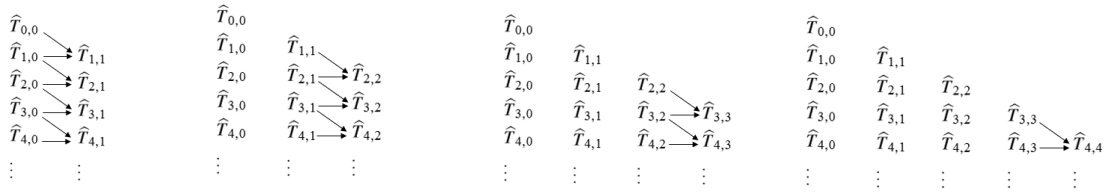
Romberg Integration

Romberg 적분은 Trapezoidal approximation 과 Richardson extrapolation 을 결합해 오차를 최소화하는 동시에 적분 값에 빠르게 수렴하는 방법이다.



$\hat{T}_{i,0}$을 subinterval 이 2^i로 나눠 trapezoidal approximation 한 적분 값이라 정의하자. trapezoidal 방법으로 적분을 하면 필연적으로 위의 그래프와 같은 오차(error)를 수반한다. 물론 더 많은 subinterval 으로 나눠 trapezoidal 근사하는 방법이 있겠지만 수렴이 느리다는 단점이 있다. 따라서 오차를 최소화하고 동시에 빠른 수렴을 하기 위해 Richardson extrapolation 으로 서로 다른 trapezoidal 적분 값($\hat{T}_{i,0}$)을 가중평균 하여 $\hat{T}_{i,j}$을 구한다. 가중 평균 식은 $\hat{T}_{i,j} = \frac{4^j \hat{T}_{i,j-1} - \hat{T}_{i-1,j-1}}{4^j - 1}, i = 1, \dots, m,\ j = 1, \dots, i$ 이다. 즉, 가중평균을 통해 trapezoidal 이 갖는 본질적인 오차를 극복하고 더 오차가 작은 적분 값을 구하는 것이다. 이와 같은 적분 값들은 아래의 triangular array 로 표현할 수 있다.

$\hat{T}_{i,1}$ 생성 $\hat{T}_{i,2}$ 생성 $\hat{T}_{i,3}$ 생성 $\hat{T}_{i,4}$ 생성



Romberg 적분은 위와 같이 trapezoidal 을 이용한 축차적인 계산을 통해 triangular array 로 전개되며 stopping rule 은 $\left|\frac{\hat{T}_{i-1,j}-\hat{T}_{i,j-1}}{\hat{T}_{i-1,j}}\right| < \varepsilon$ where $\varepsilon = 10^{-6}$이다. 예제 5.4 을 Romberg 적분을 적용해 풀어보겠다.

2. Result

1.1 EM Algorithm for Multinomial : Complex Cell Structure

다음은 혈액형 유전자 빈도에 대한 데이터이다. 혈액형의 대립유전자(allele)는 3 가지로 A, B 그리고 O 가 있다. 주어진 데이터의 집단에서 A, B, O 의 유전자의 빈도 어떻게 될 것인지 추정할 것이다. A 와 B 는 우성 형질인데 반해 O 는 열성이기 때문에 유전형(genotype)인 AA, AO, BB, BO 빈도가 결측치이다. 따라서 EM 알고리즘을 이용해 추정할 것이다.

Observed data

$$parameter\ \theta = (p, q)$$

$$where\ p\ is\ probability\ of\ A\ , q\ is\ probability\ of\ B\ and\ r\ is\ probability\ of\ O\ and\ r = 1 - p - q$$

Phenotype	Cell probability	Observed frequency
O	r^2	$n_O = 176$
A	$p^2 + 2pr$	$n_A = 182$
B	$q^2 + 2qr$	$n_B = 60$
AB	$2pq$	$n_{AB} = 17$

Complete data

Genotype	Cell probability	Observed frequency
OO	r^2	$n_{OO} = n_O = 176$
AA	p^2	n_{AA}
AO	$2pr$	n_{AO}
BB	q^2	n_{BB}
BO	$2qr$	n_{BO}
AB	$2pq$	$n_{AB} = 17$

$$l_c(\theta) = 2n_A^+ logp + 2n_B^+ logq + 2n_O^+ logr$$

$$where\ n_A^+ = n_{AA} + \frac{1}{2}n_{AO} + \frac{1}{2}n_{AB},\ n_B^+ = n_{BB} + \frac{1}{2}n_{BO} + \frac{1}{2}n_{AB},\ n_O^+ = n_O + \frac{1}{2}n_{AO} + \frac{1}{2}n_{BO}$$

$$\hat{\theta}^{MLE} = (\hat{p}^{MLE}, \hat{q}^{MLE})$$

$$where\ \hat{p}^{MLE} = \frac{n_A^+}{n}, \hat{q}^{MLE} = \frac{n_B^+}{n}, \hat{r}^{MLE} = 1 - \hat{p}^{MLE} - \hat{q}^{MLE}$$

EM algorithm

<k^{th} E step >

$$\begin{cases} n_{AA}^{(k)} = E_{\theta^{(k)}}(n_{AA}|n_A) = n_A \dfrac{{p^{(k)}}^2}{{p^{(k)}}^2 + 2p^{(k)}r^{(k)}} \\ n_{AO}^{(k)} = E_{\theta^{(k)}}(n_{AO}|n_A) = n_A \dfrac{2p^{(k)}r^{(k)}}{{p^{(k)}}^2 + 2p^{(k)}r^{(k)}} \\ n_{BB}^{(k)} = E_{\theta^{(k)}}(n_{BB}|n_A) = n_B \dfrac{{q^{(k)}}^2}{{q^{(k)}}^2 + 2q^{(k)}r^{(k)}} \\ n_{BB}^{(k)} = E_{\theta^{(k)}}(n_{BB}|n_A) = n_B \dfrac{2q^{(k)}r^{(k)}}{{q^{(k)}}^2 + 2q^{(k)}r^{(k)}} \end{cases}$$

k^{th} M step

Update

$$\begin{cases} p^{(k+1)} = (n_{AA}^{(k)} + \frac{1}{2}n_{AO}^{(k)} + \frac{1}{2}n_{AB})/n \\ q^{(k+1)} = (n_{BB}^{(k)} + \frac{1}{2}n_{BO}^{(k)} + \frac{1}{2}n_{AB})/n \end{cases}$$

Iterate until $\left|l_c(\theta^{(k+1)}) - l_c(\theta^{(k)})\right| < \epsilon$ or number of iterations $> N$ where $\epsilon = 10^{-6}, N = 1000$

Estimated parameters

k	$p^{(k)}$	$q^{(k)}$	$n_{AA}^{(k)}$	$n_{AO}^{(k)}$	$n_{BB}^{(k)}$	$n_{BO}^{(k)}$
0	0.3333333	0.3333333				
1	0.2857889	0.1073145	49.6363636	132.3636364	16.3636364	43.6363636
2	0.26860389	0.09410787	34.68538778	147.31461222	4.87384323	55.12615677
3	0.26514790	0.09324767	31.67867446	150.32132554	4.12547419	55.87452581
4	0.2645592	0.0931778	31.1665370	150.8334630	4.0646836	55.9353164
5	0.26446290	0.09317007	31.08272051	150.91727949	4.05796089	55.94203911
6	0.2644473	0.0931690	31.0691593	150.9308407	4.0570342	55.9429658
7	0.26444480	0.09316884	31.06697280	150.93302720	4.05689289	55.94310711
8	0.26444439	0.09316882	31.06662063	150.93337937	4.05687054	55.94312946
9	0.26444433	0.09316881	31.06656393	150.93343607	4.05686696	55.94313304
10	0.26444432	0.09316881	31.06655480	150.93344520	4.05686639	55.94313361
11	0.26444431	0.09316881	31.06655333	150.93344667	4.05686630	55.94313370
12	0.26444431	0.09316881	31.06655309	150.93344691	4.05686628	55.94313372

1.2 EM Algorithm for Multinomial : Peppered Moths

다음은 peppered moths 의 날개 색상 유전자 빈도 데이터이다. 색상의 대립유전자는 3 가지로 C, I 그리고 T 가 있다. 주어진 데이터의 집단에서 C, I, T 의 유전자의 빈도가 어떻게 될 것인지 추정할 것이다. C, I, T 순으로 우성 형질이기 때문에 유전자형(genotype)이 CC, CI, 그리고 CT 일 때는 표현형(phenotype)이 C 로 peppered moth 의 날개 색상이 Solid Black 이고 유전자형이 II 와 IT 일때는 표현형이 I 로 색상이 gray 이다. 마지막으로 가장 열성인 allele 끼리 교배해 유전자형이 TT 일 경우 색상이 light 이다. 유전자 빈도를 추정하기 위해 아래와 같은 EM 알고리즘을 사용한다.

Observed data

$$parameter\ \theta = (p, q)$$

$$where\ p\ is\ probability\ of\ C\ , q\ is\ probability\ of\ I\ and\ r\ is\ probability\ of\ T\ and\ r = 1 - p - q$$

Phenotype	Cell probability	Observed frequency
C	$p^2 + 2pq + 2pr$	$n_C = 85$
I	$q^2 + 2qr$	$n_I = 196$
T	r^2	$n_T = 341$

Complete data

Genotype	Cell probability	Observed frequency
CC	p^2	n_{CC}
CT	$2pq$	n_{CT}
CI	$2pr$	n_{CI}
II	q^2	n_{II}
IT	$2qr$	n_{IT}
TT	r^2	$n_{TT} = n_T = 341$

$$l_c(\theta) = 2n_C^+ logp + 2n_I^+ logq + 2n_T^+ logr$$

$$where\ n_C^+ = n_{CC} + \frac{1}{2}n_{CI} + \frac{1}{2}n_{CT},\ n_I^+ = n_{II} + \frac{1}{2}n_{CI} + \frac{1}{2}n_{IT},\ n_T^+ = n_T + \frac{1}{2}n_{CT} + \frac{1}{2}n_{IT}$$

$$\hat{\theta}^{MLE} = (\hat{p}^{MLE}, \hat{q}^{MLE})$$

$$\hat{p}^{MLE} = \frac{n_C^+}{n}, \hat{q}^{MLE} = \frac{n_I^+}{n}, \hat{r}^{MLE} = 1 - \hat{p}^{MLE} - \hat{q}^{MLE}$$

EM algorithm

<k^{th} E step >

$$\begin{cases} n_{CC}^{(k)} = E_{\theta^{(k)}}(n_{CC}|n_C) = n_C \dfrac{p^{(k)^2}}{p^{(k)^2} + 2p^{(k)}q^{(k)} + 2p^{(k)}r^{(k)}} \\ n_{CI}^{(k)} = E_{\theta^{(k)}}(n_{CI}|n_C) = n_C \dfrac{2p^{(k)}q^{(k)}}{p^{(k)^2} + 2p^{(k)}q^{(k)} + 2p^{(k)}r^{(k)}} \\ n_{CT}^{(k)} = E_{\theta^{(k)}}(n_{CT}|n_C) = n_B \dfrac{2p^{(k)}r^{(k)}}{p^{(k)^2} + 2p^{(k)}q^{(k)} + 2p^{(k)}r^{(k)}} \\ n_{II}^{(k)} = E_{\theta^{(k)}}(n_{II}|n_I) = n_I \dfrac{q^{(k)^2}}{q^{(k)^2} + 2q^{(k)}r^{(k)}} \\ n_{IT}^{(k)} = E_{\theta^{(k)}}(n_{IT}|n_I) = n_I \dfrac{2q^{(k)}r^{(k)}}{q^{(k)^2} + 2q^{(k)}r^{(k)}} \end{cases}$$

k^{th} M step

Update

$$\begin{cases} p^{(k+1)} = (n_{CC}^{(k)} + \frac{1}{2}n_{CI}^{(k)} + \frac{1}{2}n_{CT}^{(k)})/n \\ q^{(k+1)} = (n_{II}^{(k)} + \frac{1}{2}n_{CI}^{(k)} + \frac{1}{2}n_{IT}^{(k)})/n \end{cases}$$

Iterate until $\left|\frac{(\theta^{(k+1)} - \theta^{(k)})}{\theta^{(k)}}\right| < \epsilon$ or number of iterations $> N$ where $\epsilon = 10^{-6}, N = 1000$

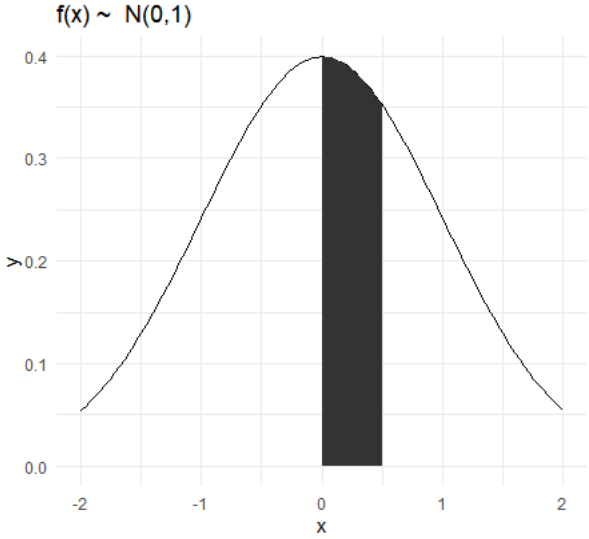
Estimated parameters

k	$p^{(k)}$	$q^{(k)}$	$n_{CC}^{(k)}$	$n_{CI}^{(k)}$	$n_{CT}^{(k)}$	$n_{II}^{(k)}$	$n_{IT}^{(k)}$
0	0.333333	0.333333					
1	0.081993	0.237406	17.000000	34.000000	34.000000	65.333333	130.666667
2	0.071248	0.197869	3.633696	21.042190	60.324112	29.107609	166.892390
3	0.070852	0.190360	3.139939	17.440215	64.419845	23.368091	172.631908
4	0.070837	0.189022	3.121804	16.774899	65.103295	22.369349	173.630650
5	0.070836	0.188786	3.121138	16.656896	65.221964	22.193972	173.806027
6	0.070836	0.188745	3.121114	16.636109	65.242775	22.163136	173.836863
7	0.070836	0.188738	3.121113	16.632452	65.246433	22.157713	173.842286
8	0.070836	0.188736	3.121113	16.631809	65.247076	22.156759	173.843240
9	0.070836	0.188736	3.121113	16.631696	65.247190	22.156592	173.843407
10	0.070836	0.188736	3.121113	16.631676	65.247209	22.156562	173.843437

2. Numerical integration : Riemann, Trapezoidal and Simpson's

본 과제에서는 interval 의 수를 2^{10} 에서 출발해 stopping rule $\left|\frac{S^k - S^{k+1}}{S^k}\right| < \varepsilon, where\ \varepsilon = 10^{-6}, S\ is\ integral\ value, k\ is\ the\ number\ of\ iteration$을 만족할 때까지 interval 의 수를 2 배씩 늘린다.

example 1

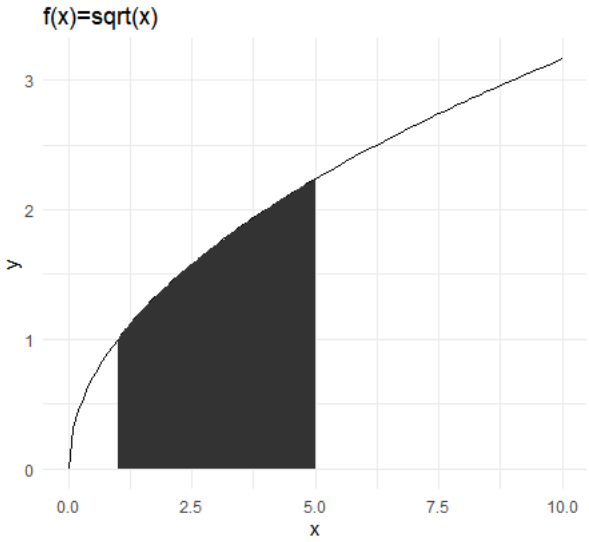


$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, x \in R$$

$$S = \int_0^{0.5} f(x)dx$$

	Riemann	Trapezoidal	Simpson's	integrate
적분 값(S)	0.1914623	0.1914625	0.1914625	0.1914625
error	9.351938e-07	1.370029e-08	7.248307e-16	error < 2.1e-15
system time	0.47	0.001	0.0009	0.0002
# of iteration	11	2	2	

example 2

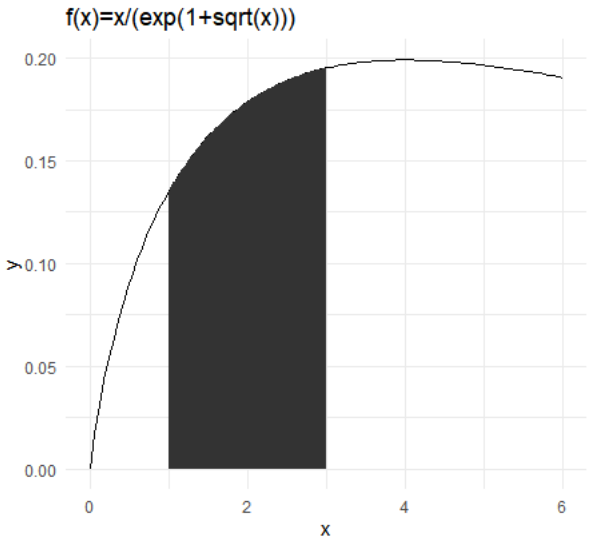


$$f(x) = \sqrt{x}, x \geq 0$$

$$S = \int_1^5 f(x)dx$$

	Riemann	Trapezoidal	Simpson's	integrate
적분 값(S)	6.786887	6.786893	6.786893	6.786893
error	9.094473e-07	3.883795e-08	6.595682e-14	error < 3.1e-12
system time	0.01	0.0001	0.00009	0.00006
# of iteration	11	2	2	

example 3



$$f(x) = \frac{x}{e^{1+\sqrt{x}}}, x \geq 0$$

$$S = \int_1^3 f(x)dx$$

	Riemann	Trapezoidal	Simpson's	integrate
적분 값(S)	0.348988	0.3489883	0.3489883	0.3489883
error	9.034056e-07	4.027142e-08	1.654256e-14	error < 3.9e-15
system time	0.2	0.0003	0.0004	0.00003
# of iteration	11	2	2	

3 개의 example 을 Riemann, Trapezoidal, Simpson's 과 R 의 베이스 함수인 integrate 방법을 이용하여 적분해보았다. 그 결과 적분 값은 Riemann 을 제외하고 나머지 방법들은 모두 소수점 6 째자리까지 동일했다. error 가 작은 순은 Simpson's≈integrate, Trapezoidal, 그리고 Riemann 이었다. 속도는 integrate, Simpson's, Trapezoidal, 그리고 Riemann 순이었다. 단 유의해야할 점은 직접 실현한 Numerical approximation 함수들과 달리 R 의 integrate 함수는 subinterval 들이 equally spaced 가 아니라 함수의 복잡성에 따라 subinterval 의 length 가 상이하다는 것이다. 따라서 R 의 integrate 함수는 좀 더 효율적인 방법으로 Numerical approximation 을 수행하는 것으로 추측된다.

3. Bayesian Inference, Singularity and Transformation

$observed\ data\ (x_1, \dots, x_7) = (6.52, 8.32, 0.31, 2.82, 9.96, 0.14, 9.64)$ 일 때 사전분포 $\pi(\mu)$ 는 Cauchy(5,2)를 따르고 우도함수인 충분통계량$(\bar{x})$의 조건부 밀도 함수 $f(\bar{x}|\mu)$는 $N(\mu, \frac{3^2}{7})$을 따른다. 사후분포를 식으로 나타내면 다음과 같다.

$$\pi(\mu|x) = \frac{\pi(\mu)f(\bar{x}|\mu)}{\int \pi(\mu)f(\bar{x}|\mu)} = k \times (prior) \times (likelihood)$$

$$where\ k\ is\ integral\ constant\ which\ makes\ posterior\ p.d.f$$

이때 적분 상수 k 는 posterior 를 전체 x 공간에서 적분했을 때 1 로 만들어주는 값이다.

a. 적분 상수 k 를 구하라.

	Riemann	Trapezoidal	Simpson's	integrate
k=1/적분 값(S)	7.846538	7.846538	7.846538	7.846538
error	3.52812e-14	3.52812e-14	3.413565e-07	error < 0.00011
system time	0.14	0.20	0.33	0.001
# of iteration	11	11	11	

k 는 적분 값의 역수이며 이 값은 모두 동일하다. Result2 의 결론과 다르게 속도는 integrate, Riemann, Trapezoidal 그리고 Simpson's 순으로 빠르며 error 는 Riemann = Trapezoidal, Simpson's 그리고 integrate 순으로 작다.

b. $\int_2^8 posterior(\mu)d\mu = 0.99605$인가?

	Riemann	Trapezoidal	Simpson's	integrate
k*적분 값(S)	0.9960539	0.9960544	0.9960544	0.9960544
error	5.085796e-07	6.146781e-08	8.135887e-13	error < 1e-04
system time	0.03	0.008	0.0014	0.001
# of iteration	7	2	2	

k*적분 값은 Riemann 값을 제외하고 모두 동일하다. Result2 의 결론과 같이 속도는 integrate, Simpson's, Trapezoidal 그리고 Riemann 순으로 빠르며 error 는 Simpson's, Trapezoidal, Riemann 그리고 integrate 순으로 작다.

c. $\mathrm{u} = \frac{e^{\mu}}{1+e^{\mu}}$로 변수 변환하여 $\int_3^{\infty} posterior(\mu)d\mu$을 구하라.

적분의 범위가 무한대(infinity)이므로 $\mathrm{u} = \frac{e^{\mu}}{1+e^{\mu}}$로 변수 변환한다. 아래와 같이 적분할 수 있다. 이때 적분 구간 upper 는 Singularity 를 해결하기 위해 1 에서 아주 작은 수 10^{-10}을 뺀 값을 대입했다.

$$\text{Let } \mathrm{u} = \frac{e^{\mu}}{1+e^{\mu}} \text{ then } \mu = \log\frac{u}{1-u} \quad d\mu = \frac{du}{u(1-u)}$$

$$\therefore \int_3^{\infty} posterior(\mu)d\mu = \int_{\frac{\exp(3)}{1+\exp(3)}}^{1-10^{-10}} posterior(log\frac{u}{1-u})\frac{du}{u(1-u)}$$

	Riemann	Trapezoidal	Simpson's	integrate
적분 값(S)	0.9908584	0.9908592	0.9908591	0.9908592
error	7.745883e-07	2.139252e-07	9.871994e-07	error < 9.7e-05
system time	0.0128	0.0012	0.0019	0.002
# of iteration	5	2	2	

적분 값은 4 가지 방법 모두 소수점 5 째자리까지 동일하다. 속도는 integrate, Trapezoidal, Simpson's 그리고 Riemann 순으로 빠르며 error 는 Trapezoidal, Riemann, Simpson's 그리고 integrate 순으로 작다.

d. $\mathrm{u} = \frac{1}{\mu}$로 변수 변환하여 $\int_3^{\infty} posterior(\mu)d\mu$을 구하라.

적분의 범위가 무한대(infinity)이므로 $\mathrm{u} = \frac{1}{\mu}$로 변수 변환한다. 아래와 같이 적분할 수 있다. 이때 적분 구간 upper 는 Singularity 를 해결하기 위해 0 에서 아주 작은 수 10^{-10}을 더해 값을 대입했다.

$$\text{Let u} = \frac{1}{\mu} \text{ then } \mu = \frac{1}{u} \quad d\mu = -\frac{du}{u^2}$$

$$\int_3^{\infty} posterior(\mu)d\mu = \int_{\frac{1}{3}}^{10^{-10}} posterior\left(\frac{1}{u}\right)(-\frac{du}{u^2})$$

	Riemann	Trapezoidal	Simpson's	integrate
적분 값(S)	0.9908586	0.9908592	0.9908592	0.9908592
error	5.533647e-07	3.920824e-08	3.419659e-13	error < 5.5e-08
system time	0.04	0.0008	0.0011	0.0002
# of iteration	7	2	2	

적분 값은 Riemann 을 제외하고 모두 동일하다. 속도는 integrate, Trapezoidal, Simpson's 그리고 Riemann 순으로 빠르며 error 는 Simpson's, Trapezoidal, integrate 그리고 Riemann 순으로 작다.

4. Romberg Integration

5.4 $X \sim unif(1, a)$ $Compute\ E\left(\frac{a-1}{X}\right)$ $for\ a > 1$ $using\ Romberg's\ integration\ with\ m = 6$

$$\int_1^a \frac{a-1}{x} \times \frac{1}{a-1} dx = \int_1^a \frac{1}{x} dx = logx]_1^a = loga$$

Triangular Matrix with m=6 and a=3

0	1.333333						
1	1.166667	1.111111					
2	1.116667	1.100000	1.113333				
3	1.103211	1.098725	1.102314	1.102997			
4	1.099768	1.098620	1.099538	1.099713	1.099754		
5	1.098902	1.098613	1.098844	1.098888	1.098898	1.098901	
6	1.098685	1.098612	1.098670	1.098681	1.098684	1.098684	**1.098685**

Romberg 적분을 6 스텝 실시한 결과 Romberg 적분 값이 1.098685 로 log(3)=1.098612 과 소수점 4 째자리까지 동일하게 나타난 것을 알 수 있다.

3. Discussion

본 과제는 4 가지 파트로 구성 되어있다. 첫 번째는 EM algorithm for Multinomial 으로 유전자 빈도를 추정하기 위해 complete-incomplete 데이터 셋으로 set up 후 EM algorithm 을 사용하였다. E

step 에서 잠재 변수인 유전형(genotype)의 빈도를 conditional expected value 로 추정 후 M step 에서 대립 유전자(allele) 확률의 MLE 를 구했다. 두 번째는 Numerical integration 이다. EM 알고리즘을 끝으로 첫 번째 파트인 최적화를 마무리하고 두 번째 파트인 적분을 다룰 것이다. 예측의 통계량인 기대값은 적분으로 정의되기 때문에 통계학에서 적분을 구하는 것은 아주 중요한 문제 중 하나이다. 대부분의 복잡한 분포의 적분들은 analytic solution 이 존재하지 않기 때문에 1 차원에서는 적분이 가능한 함수로 numerical approximation 을 하고 다차원에서는 sampling 을 이용해 적분의 근사값을 구한다. 본 과제에서는 대표적인 numerical approximation 방법 3 가지(Riemann, Trapezoidal 그리고 Simpson's)를 구현해 R 의 base 함수인 integrate 와 값을 비교해 보았다. 적분 함수에 따라 결과가 상이하지만 대체적으로 속도는 integrate, Simpson's = Trapezoidal, 그리고 Riemann 순으로 빨랐다. 반복 횟수(the number of iteration)은 Riemann, Simpson's=Trapezoidal 순으로 컸다. 오차(error)는 Simpson's=Trapezoidal, 그리고 Riemann 순으로 작았다. 세 번째는 복잡한 베이지안 사후 분포를 적분하기 위해 numerical approximation 을 적용하는 예제를 풀어보았다. 또한 적분 범위가 infinity 일 때 기존의 변수를 적분에 적합한 형태로 변수변환 해보았다. 네 번째는 trapezoidal approximation 을 발전시킨 Romberg's integration 이다. Trapezoidal approximation 이 근본적으로 갖는 오차를 최소화하는 동시에 빠른 수렴을 할 수 있다는 장점이 있다. 예제 5.4 에 이를 적용한 결과 실제 적분 값에 근사한 값을 얻을 수 있었다.

4. Appendix

```
rm(list=ls())
library(tidyverse)

# HW 1.1
# Multinomial with Complex cell structure

mycell <- function(no=176, na=182, nb=60, nab=17){

  # initial value
  p <- q <- 0.3; r <- 1 - p - q
  n <- no + na + nb + nab
  err <- 1; threshold <- 10^(-6)
  niter <- 0; maxiter <- 1000
  old.likli <- 0

  while(err >= threshold && niter <= maxiter){

    # E step
    naa <- na*(p^2/(p^2+2*p*r))
    nao <- na*(2*p*r/(p^2+2*p*r))
    nbb <- nb*(q^2/(q^2+2*q*r))
    nbo <- nb*(2*q*r/(q^2+2*q*r))

    # M step
    p <- (naa + 0.5*nao + 0.5*nab)/n
    q <- (nbb + 0.5*nbo + 0.5*nab)/n
```

```
    r <- 1 - p - q
    new.likli <- 2*(naa+ 0.5*nao + 0.5*nab)*log(p) + 2*(nbb+ 0.5*nbo + 0.5*nab)*log(q) +
2*(no+ 0.5*nao + 0.5*nbo)*log(r)

    # update niter and err
    niter <- niter + 1
    err <- abs(new.likli - old.likli)
    old.likli <- new.likli
    print(c(niter, p, q, naa, nao, nbb, nbo))
  }

  return(list(prob=data.frame(p = p, q = q, r = r),
              n = data.frame(no = no, naa = naa, nao = nao, nbb = nbb, nbo = nbo, nab = nab),
              iteration=niter))

}
mycell()


# HW 1.2
# Pepper Moths


mypep <- function(nc=85, ni=196, nt=341){

  # initial value
  p <- q <- 1/3; r <- 1- p - q
  n <- nc + ni + nt
  niter <- 0; maxiter <- 1000
  err <- 1; threshold <- 10^(-6)
  oldpp <- c(p, q)

  while(niter <= maxiter && err >= threshold){

    # E step
    ncc <- nc*p^2/(p^2 + 2*p*q + 2*p*r)
    nci <- 2*nc*p*q/(p^2 + 2*p*q + 2*p*r)
    nct <- 2*nc*p*r/(p^2 + 2*p*q + 2*p*r)
    nii <- ni*q^2/(q^2 + 2*q*r)
    nit <- 2*ni*q*r/(q^2 + 2*q*r)

    # M step
    p <- (ncc + 0.5*nci + 0.5*nct)/n
    q <- (nii + 0.5*nit + 0.5*nci)/n
    r <- 1 - p - q
    pp <- c(p, q)

    # update niter and err
    niter <- niter + 1
    err <- abs(sqrt(t(oldpp-pp)%*%(oldpp-pp))/sqrt(t(oldpp)%*%(oldpp)))
    oldpp <- pp

    # print
    print(c(niter, p, q, ncc, nci, nct, nii, nit))
  }

  return(list(p = data.frame(p = p, q=q, r=r),
              n = data.frame(ncc = ncc, nci = nci, nct = nct, nii = nii, nit = nit, nt = nt),
              iteration = niter))

}
```

```
mypep()


# HW 2
# Numerical integration : Riemann, Trapezoidal, Simpson's


myint <- function(f, a, b, method="Riemann"){

  # initial
  n <- 2^10
  niter <- 0; maxiter <- 1000
  err <- 1; threshold <- 10^(-6)
  old.int <- 0

  while(niter <= maxiter && err >= threshold){
    h <- (b-a)/n
    if(method=="Riemann"){int <- h*sum(f( a + (1:(n-1))*h ))
    }else if(method=="Trapezoidal"){int <- 0.5*h*(f(a)+f(b))+h*sum(f( a + (1:(n-1))*h ))
    }else{ i <- (1:(n/2))
          int <- (h/3)*sum(f(a+(2*i-2)*h)+4*f(a+(2*i-1)*h)+f(a+(2*i)*h))}

    # update err and n and niter
    niter <- niter + 1
    n <- n*2
    err <- abs((old.int-int)/old.int)
    old.int <- int
  }

  return(list(integral=int, iteration=niter, error=err, iterval=n/2))

}

# example1
ggplot(data.frame(x=c(-
2,2)),aes(x))+stat_function(fun=dnorm)+stat_function(fun=dnorm,xlim=c(0,0.5),geom="area")+them
e_minimal()+ggtitle("f(x) ~  N(0,1)")
myint(dnorm, 0, 0.5)
myint(dnorm, 0, 0.5, method="Trapezoidal")
myint(dnorm, 0, 0.5, method="Simpson's")
integrate(dnorm, 0, 0.5)
system.time(myint(dnorm, 0, 0.5))
system.time(for(i in 1:100)myint(dnorm, 0, 0.5, method="Trapezoidal"))
system.time(for(i in 1:100)myint(dnorm, 0, 0.5, method="Simpson's"))
system.time(for(i in 1:100) integrate(dnorm, 0, 0.5))

# example2
ftn2 <- function(x){sqrt(x)}
ggplot(data.frame(x=c(0,10)),aes(x))+stat_function(fun=ftn2)+stat_function(fun=ftn2,xlim=c(1,5
),geom="area")+theme_minimal()+ggtitle("f(x)=sqrt(x)")
myint(ftn2, 1, 5)
myint(ftn2, 1, 5, method="Trapezoidal")
myint(ftn2, 1, 5, method="Simpson's")
integrate(ftn2, 1, 5)
system.time(myint(ftn2, 1, 5))
system.time(for (i in 1:1000)myint(ftn2, 1, 5, method="Trapezoidal"))
system.time(for (i in 1:1000)myint(ftn2, 1, 5, method="Simpson's"))
system.time(for (i in 1:1000) integrate(ftn2, 1, 5))

# example3
ftn3 <- function(x){x/(exp(1+sqrt(x)))}
```

```
ggplot(data.frame(x=c(0,6)),aes(x))+stat_function(fun=ftn3)+stat_function(fun=ftn3,xlim=c(1,3)
,geom="area")+theme_minimal()+ggtitle("f(x)=x/(exp(1+sqrt(x)))")
myint(ftn3, 1, 3)
myint(ftn3, 1, 3, method="Trapezoidal")
myint(ftn3, 1, 3, method="Simpson's")
integrate(ftn3, 1, 3)
system.time(myint(ftn3, 1, 3))
system.time(for(i in 1:100)myint(ftn3, 1, 3, method="Trapezoidal"))
system.time(for(i in 1:100)myint(ftn3, 1, 3, method="Simpson's"))
system.time(for(i in 1:1000) integrate(ftn3, 1, 3))


# HW 3
# ex. 5.3


# a.
data <- c(6.52, 8.32, 0.31, 2.82, 9.96, 0.14, 9.64)
posterior <- function(mu){
  likelihood <- dnorm(mean(data), mu, sqrt(9/7))
  prior <- dcauchy(mu, 5, 2)
  post <- likelihood*prior
  return(post)
}
1/integrate(posterior, -Inf, Inf)$value
1/myint(posterior, -10^5, 10^5)$integral
1/myint(posterior, -10^5, 10^5, method="Trapezoidal")$integral
1/myint(posterior, -10^5, 10^5, method="Simpson's")$integral
k <- 1/integrate(posterior, -Inf, Inf)$value
system.time(for(i in 1:100)integrate(posterior, -Inf, Inf))
system.time(myint(posterior, -10^5, 10^5))
system.time(myint(posterior, -10^5, 10^5, method="Trapezoidal"))
system.time(myint(posterior, -10^5, 10^5, method="Simpson's"))

# b.
k*integrate(posterior, 2, 8)$value
k*myint(posterior, 2, 8)$integral
k*myint(posterior, 2, 8, method="Trapezoidal")$integral
k*myint(posterior, 2, 8, method="Simpson's")$integral
system.time(for(i in 1:100)integrate(posterior, 2, 8))
system.time(myint(posterior, 2, 8))
system.time(for(i in 1:100)myint(posterior, 2, 8, method="Trapezoidal"))
system.time(for(i in 1:100)myint(posterior, 2, 8, method="Simpson's"))

# c.
u <- function(mu) {exp(mu)/(1+exp(mu))}
posterior1 <- function(u){
  likelihood <- dnorm(mean(data), log(u/(1-u)), sqrt(9/7))
  prior <- dcauchy(log(u/(1-u)), 5, 2)
  post <- k*likelihood*prior*(1/(u*(1-u)))
  return(post)
}
lower1 <- exp(3)/(1+exp(3))
integrate(posterior1, lower1, 1-10^(-10))
myint(posterior1, lower1, 1-10^(-10))
myint(posterior1, lower1, 1-10^(-10), method="Trapezoidal")
myint(posterior1, lower1, 1-10^(-10), method="Simpson's")
system.time(for(i in 1:100)integrate(posterior1, lower1, 1-10^(-10)))
system.time(for(i in 1:100)myint(posterior1, lower1, 1-10^(-10)))
system.time(for(i in 1:100)myint(posterior1, lower1, 1-10^(-10), method="Trapezoidal"))
system.time(for(i in 1:100)myint(posterior1, lower1, 1-10^(-10), method="Simpson's"))
```

```
# d.
u <- function(mu) {1/mu}
posterior2 <- function(u){
  likelihood <- dnorm(mean(data), 1/u, sqrt(9/7))
  prior <- dcauchy(1/u, 5, 2)
  post <- k*likelihood*prior*(-1/u^2)
  return(post)
}
lower2 <- 1/3
integrate(posterior2, lower2, 10^(-10))
myint(posterior2, lower2, 10^(-10))
myint(posterior2, lower2, 10^(-10), method="Trapezoidal")
myint(posterior2, lower2, 10^(-10), method="Simpson's")
system.time(for(i in 1:100)integrate(posterior2, lower2, 10^(-10)))
system.time(myint(posterior2, lower2, 10^(-10)))
system.time(for(i in 1:100)myint(posterior2, lower2, 10^(-10), method="Trapezoidal"))
system.time(for(i in 1:100)myint(posterior2, lower2, 10^(-10), method="Simpson's"))

# HW 4
# ex. 5.4
mytrap <- function(f, a, b, n){
  h <- (b-a)/n
  result <- 0.5*h*(f(a)+f(b))+h*sum(f( a + (1:(n-1))*h ))
  return(result)
}

m0 <- 6
a <- 10
f <- function(x) 1/x
tb <- matrix(0, ncol=(m0+1), nrow=(m0+1))
tb[1,1] <- (a-1)/2*(f(1)+f(a))

for (m in 1:m0){
    for(j in 1:m){
      tb[m+1,1] <- mytrap(f, 1, a, n=2^m)
      tb[m+1,j+1] <- (4^j*tb[m+1,1]-tb[m,1] )/(4^j-1)
    }
}

tb
log(a)
```