Homework 5

Jieqi Tu (jt3098)

11/28/2020

Question 1

(a)

The Bayesian model would be $\bar{x}|\mu \sim N(\mu, \frac{3^2}{7})$ and here our prior is $\mu \sim Cauchy(5, 2)$.

```
# Import data
x = c(6.52, 8.32, 0.31, 2.82, 9.96, 0.14, 9.64)
xbar = mean(x)

# Construct Bayesian Model
likelihood.function = function(mu) {
  1/sqrt(2*pi*9/7) * exp(-(xbar - mu)^2/(2*9/7))
}

prior.function = function(mu) {
  1/(pi*2*(1 + ((mu - 5)/2)^2))
}
```

Then we need to find the normalizating constant k. Here we need to perform a variable transformation. Let $\mu = log(\frac{y}{1-y})$, since $-\infty < \mu < \infty$, we will get $0 < y < 1$. The Jacobian matrix would be $(1/(y(1-y)))$.

```
# define function for y
y.function = function(y) {
  likelihood.function(log(y/(1-y))) * prior.function(log(y/(1-y))) * (1/(y*(1-y)))
}

# Riemann Sum
int_Riemann = function(f, a, b, n = 100000) {
  h = (b-a)/n
  x = seq(a, b, by = h)
  y = f(x)
  result = h*sum(y[1:n])
  return(result)
}

k = round(1/int_Riemann(y.function, 1e-10, 1-1e-10), 5)
k
```

```
## [1] 7.84654
```

(b)

```
# Define the function for posterior
posterior.function = function(mu) {
  k*likelihood.function(mu)*prior.function(mu)
}

# Riemann Sum
int_Riemann = function(f, a, b, n = 100000) {
  h = (b-a)/n
  x = seq(a, b, by = h)
  y = f(x)
  result = h*sum(y[1:n])
  return(result)
}

# Trapezoidal Rule
int_trapzoidal = function(f, a, b, n = 100000) {
  h = (b - a)/n
  x = seq(a, b, by = h)
  y = f(x)
  result = h * (y[1] + 2*sum(y[2:n]) + y[n+1]) / 2
  return(result)
}

# Simpson's Rule
int_Simpson = function(f, a, b, n = 100000) {
  h = (b - a)/n
  x = seq(a, b, by = h)
  y = f(x)
  if (n == 2) {
    result = (h/3) * (y[1] + 4*y[2] + y[3])
  } else {
    result = (h/3) * (y[1] + sum(2*y[seq(2, n, by = 2)]) + sum(4*y[seq(3, n-1, by = 2)]) + y[n+1])
    }
  return(result)
}

# Calculate the posterior probability that 2 <= mu <= 8
data.frame(Method = c("Riemann", "Trapezoidal", "Simpson's"),
           Result = c(int_Riemann(posterior.function, 2, 8),int_trapzoidal(posterior.function, 2, 8), i
```

```
##        Method    Result
## 1     Riemann 0.9960544
## 2 Trapezoidal 0.9960547
## 3   Simpson's 0.9960544
```

All of these three results are close to 0.99605. However, Trapezoidal method has a little bit higher value of integration than the other two methods.

(c)

We still need to perform variable transformation before integration. Let $\mu = log(\frac{u}{1-u})$. Since $3 \leq \mu \leq \infty$, we have $\frac{e^3}{1+e^3} < \mu < 1$. The Jacobian matrix would be $\frac{1}{u(1-u)}$.

```r
# define the posterior function
posterior.function.u = function(u) {
  k*likelihood.function(log(u/(1-u))) * prior.function(log(u/(1-u))) * (1/(u*(1-u)))
}

# Use Riemann sum method
int_Riemann(posterior.function.u, exp(3)/(1+exp(3)), 1-1e-10)
```

```
## [1] 0.9908596
```

(d)

Perform variable transformation: $\mu = 1/u$ and then $0 < u < 1/3$. The Jacobian matrix is $1/u^2$.

```r
# define the posterior function
posterior.function.u = function(u) {
  k*likelihood.function(1/u) * prior.function(1/u) * (1/u^2)
}

int_Riemann(posterior.function.u, 1e-10, 1/3)
```

```
## [1] 0.9908591
```

From (c) and (d), we know that, both results are very close to 0.99086. However, the transformation in (c) is more close.

Question 2

```r
# define the function for Romberg's algorithm
int_Romberg = function(f, a, b, m) {
  R = matrix(NA, m, m)
  h = b - a
  R[1,1] = (f(a) + f(b)) * h/2
  for (i in 2:m) {
    R[i,1] = 1/2 * (R[i-1,1] + h * sum(f(a + (1:2^(i-2) - 0.5) * h)))
    for (j in 2:i) {
      R[i,j] = R[i,j-1] + (R[i,j-1] - R[i-1,j-1]) / (4^(j-1) - 1)
    }
    h = h/2
  }
  result = R[m,m]

  return(list(R, result))
}

EY = function(a, m) {
  Romberg = int_Romberg(function(x) 1/x, 1, a, m)

  # simulated solution (MC integration)
  set.seed(1029)
```

```
  x = runif(n=10000, min = 1, max = a)
  y = (a-1)/x
  MC_integration = mean(y)

  # theoretical result
  theoretical = log(a)

  return(list(Romberg.array = Romberg[[1]],
              Romberg.solution = Romberg[[2]],
              MC.integration = MC_integration,
              theoretical = theoretical))
}

EY(a = exp(1), m = 6)
```

```
## $Romberg.array
##          [,1]     [,2]     [,3]     [,4] [,5] [,6]
## [1,] 1.175201       NA       NA       NA   NA   NA
## [2,] 1.049718 1.007890       NA       NA   NA   NA
## [3,] 1.013039 1.000813 1.000341       NA   NA   NA
## [4,] 1.003307 1.000063 1.000013 1.000008   NA   NA
## [5,] 1.000830 1.000004 1.000000 1.000000    1   NA
## [6,] 1.000208 1.000000 1.000000 1.000000    1    1
##
## $Romberg.solution
## [1] 1
##
## $MC.integration
## [1] 0.9930809
##
## $theoretical
## [1] 1
```

```
EY(a = exp(2), m = 6)
```

```
## $Romberg.array
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] 3.626860       NA       NA       NA       NA       NA
## [2,] 2.575024 2.224412       NA       NA       NA       NA
## [3,] 2.178272 2.046022 2.034129       NA       NA       NA
## [4,] 2.049460 2.006522 2.003889 2.003409       NA       NA
## [5,] 2.012847 2.000642 2.000250 2.000192 2.000180       NA
## [6,] 2.003248 2.000049 2.000009 2.000005 2.000004 2.000004
##
## $Romberg.solution
## [1] 2.000004
##
## $MC.integration
## [1] 1.969252
##
## $theoretical
## [1] 2
```

```r
EY(a = exp(9), m = 6)
```

```
## $Romberg.array
##           [,1]       [,2]      [,3]      [,4]      [,5]    [,6]
## [1,] 4051.5419         NA        NA        NA        NA      NA
## [2,] 2026.7707 1351.84697        NA        NA        NA      NA
## [3,] 1014.7181  677.36728 632.40197        NA        NA      NA
## [4,]  509.0341  340.47276 318.01312 313.02282        NA      NA
## [5,]  256.5365  172.37061 161.16380 158.67413 158.06884      NA
## [6,]  130.6316   88.66328  83.08279  81.84341  81.54211 81.4673
##
## $Romberg.solution
## [1] 81.4673
##
## $MC.integration
## [1] 7.314086
##
## $theoretical
## [1] 9
```

From the result, we could see that, when a is a small value, this method works fine. However, when we increase the value of a, the results are much more deviated from the theoretical values.

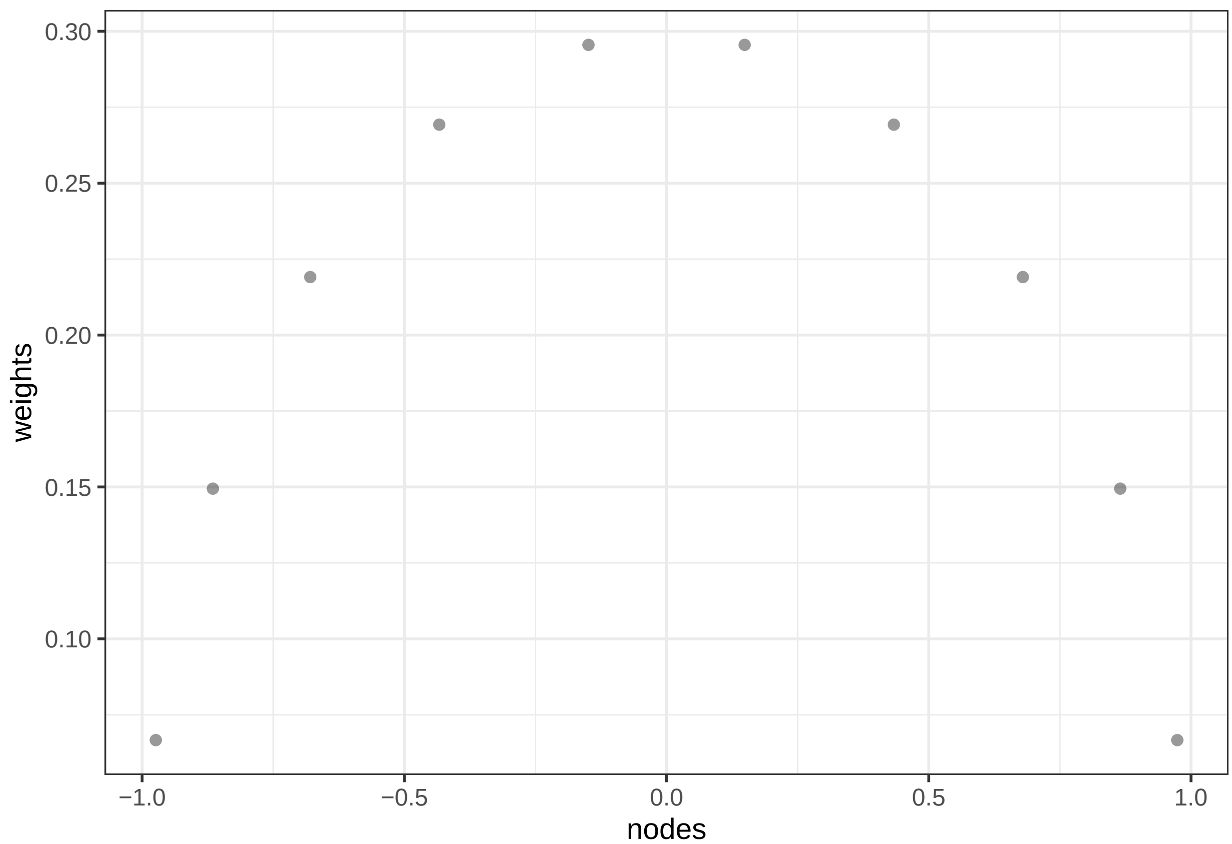
Question 3

(a)

Plot the weights versus the nodes.

```r
# 10-point Gaussian quadrature rule
nodes = c(-0.148874338981631, 0.148874338981631, -0.433395394129247, 0.433395394129247,
          -0.679409568299024, 0.679409568299024, -0.865063366688985, 0.865063366688985,
          -0.973906528517172, 0.973906528517172)
weights = c(0.295524224714753, 0.295524224714753, 0.269266719309996, 0.269266719309996,
            0.219086362515982, 0.219086362515982, 0.149451394150581, 0.149451394150581,
            0.066671344308688, 0.066671344308688)

# plot the weights versus nodes
data3 = data.frame(nodes, weights)
data3 %>%
  ggplot(aes(x = nodes, y = weights)) + geom_point(alpha = 0.4) +
  theme_bw()
```



(b)

```
# Define the function for area
area = function(nodes, weights, f, lower, upper) {
  return(list(theoretical = integrate(f, lower, upper),
              gaussian.quadrature.num = length(nodes),
              gaussian.area = sum(weights*f(nodes))))
}

# Define the function of y = x^2
f = function(x) {x^2}

area(nodes, weights, f, lower = -1, upper = 1)
```

```
## $theoretical
## 0.6666667 with absolute error < 7.4e-15
##
## $gaussian.quadrature.num
## [1] 10
##
## $gaussian.area
## [1] 0.6666667
```

From the result, we could see that, the result of 10-point Gaussian quadrature method is identical to the theoretical value.

Question 4

(a)

Q4. (a).

$\alpha_k = 0$, $\beta_k = 1$, $\gamma_k = k-1$, $w(x) = e^{-\frac{x^2}{2}}$

orthogonal polynomial:

$$P_k(x) = (\alpha_k + x\beta_k) P_{k-1}(x) - \gamma_k P_{k-2}(x)$$

Assume $P_0(x) = 1$

$$P_1(x) = (0+x)P_0(x) = x$$

$$P_2(x) = (0+x)P_1(x) - P_0(x) = x^2 - 1$$

$$P_3(x) = (0+x)P_2(x) - 2P_1(x) = x^3 - 3x$$

$$P_4(x) = (0+x)P_3(x) - 3P_2(x) = x^4 - 6x^2 + 3.$$

$$P_5(x) = (0+x)P_4(x) - 4P_3(x) = x^5 - 10x^3 + 15x.$$

Therefore, $H_5(x) = c(x^5 - 10x^3 + 15x)$

Since we assume $P_0(x) = 1$, here $c = 1$. However, it could be other values. But H5(x) is indeed rely on this.

(b)

Q4 (b)

If $\int_a^b f(x)^2 w(x)\,dx < \infty$, f is square-integrable with respect to w on $[a, b]$

$\langle f, g\rangle_{w,[a,b]} = \int_a^b f(x) g(x) w(x)\,dx$

If f and g are scaled, $\langle f, f\rangle_{w,[a,b]} = \langle g, g\rangle_{w,[a,b]} = 1$

$\Rightarrow$ f and g are orthogonal w.r.t w on $[a,b]$

To find c in $H_5(x)$.

$$\int_{-\infty}^{\infty} [H_5(x)][H_5(x)]\, w(x)\,dx = 1$$

$$= c^2 \int_{-\infty}^{\infty} (x^5 - 10x^3 + 15x)^2 e^{-\frac{x^2}{2}}\,dx = 1$$

$$= c^2 \int_{-\infty}^{\infty} (x^{10} - 20x^8 + 130x^6 - 300x^4 + 225x^2)\, e^{-\frac{x^2}{2}}\,dx = 1$$

Define $Y \sim \text{Gamma}(\alpha, \beta)$.

Then the pdf $f(y) = \int_0^\infty \frac{\beta^\alpha y^{\alpha-1} e^{-\beta y}}{\Gamma(\alpha)}\,dy$.

and $\int_0^\infty f(y)\,dy = 1$.

Let $\beta = \frac{1}{2}$ and $y = x^2$.

Then we will get $\frac{\Gamma(\alpha)}{(\frac{1}{2})^\alpha} = \int_0^\infty y^{\alpha-1} e^{-\frac{1}{2}y}\,dy$

$$= \int_{-\infty}^{\infty} x^{\cancel{2(\alpha-1)}\, 2\alpha-1} e^{-\frac{1}{2}x^2}\,dx$$

Let $\cancel{2(\alpha-1)}\ 2\alpha-1 = 2n$, Then we have: $\int_{-\infty}^{\infty} x^{2n} e^{-\frac{1}{2}x^2}\,dx = \frac{\Gamma(n+\frac{1}{2})}{(\frac{1}{2})^{n+\frac{1}{2}}}$

when $n = 5$, $\int_{-\infty}^{\infty} x^{10} e^{-\frac{1}{2}x^2} = \frac{\Gamma(5+\frac{1}{2})}{(\frac{1}{2})^{5+\frac{1}{2}}} = \frac{\frac{9}{2}\cdot\frac{7}{2}\,\frac{5}{2}\,\frac{3}{2}\cdot\frac{1}{2}\,\frac{1}{2}\sqrt{\pi}}{(\frac{1}{2})^{\frac{11}{2}}}$

Similarly, we can get other integrals.

Hence $c = \frac{1}{\sqrt{\int (x^{10} - 20x^8 + 130x^2 - 300x^4 + 225x^2) e^{-\frac{x^2}{2}}\,dx}} = \frac{1}{\sqrt{120\sqrt{2\pi}}}$

Here we used the probability density function of Gamma distribution to calculate the integration.

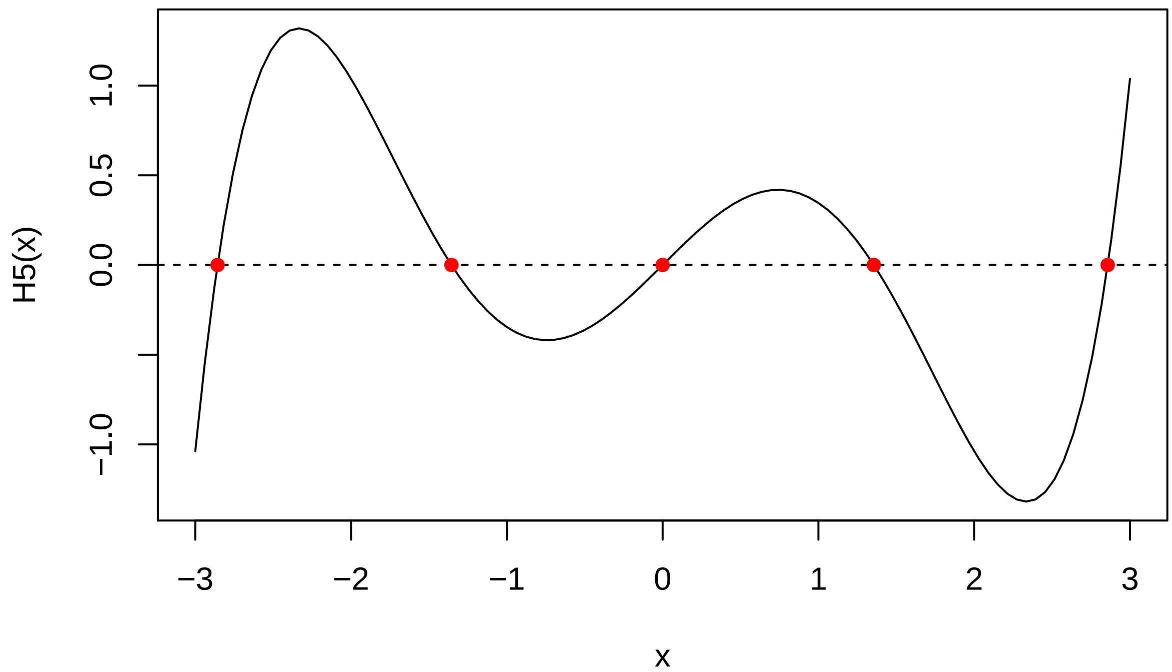
(c)

Estimate the nodes of the five-point Gauss-Hermite quadrature rule.

```r
# Use Bisection method
# Define the function for bisection method
bisection.function = function(f, a, b, n = 1000, tol = 1e-6) {
  if(f(a)*f(b)>0) {
    stop('Choose f(a) and f(b) that have different signs')
  }

  for (i in 1:n) {
    c = (a+b)/2
    if(f(c) == 0 | ((b-a)/2)<tol) {return(c)}
    if(sign(f(c)) == sign(f(a))) {a = c}
    else {b = c}
  }
}

# plot H5(x) from -3 to 3
# define H5(x)
H5 = function(x) {
  (1/sqrt(120*sqrt(2*pi))) * (x^5 - 10*x^3 + 15*x)
}
x = seq(-3, 3, length = 100)
plot(x = x, y = H5(x), type = 'l')
abline(h = 0, lty = 2)
points(x = bisection.function(f=H5, a=-3, b=-2), y = 0, pch = 16, col = 'red')
points(x = bisection.function(f=H5, a=-2, b=-1), y = 0, pch = 16, col = 'red')
points(x = bisection.function(f=H5, a=-1, b=1), y = 0, pch = 16, col = 'red')
points(x = bisection.function(f=H5, a=1, b=2), y = 0, pch = 16, col = 'red')
points(x = bisection.function(f=H5, a=2, b=3), y = 0, pch = 16, col = 'red')
```




```r
# Find the root using bisection method
data.frame(point=c(1,2,3,4,5),
           root=c(bisection.function(f=H5, a=-3, b=-2),
                  bisection.function(f=H5, a=-2, b=-1),
                  bisection.function(f=H5, a=-1, b=1),
                  bisection.function(f=H5, a=1, b=2),
                  bisection.function(f=H5, a=2, b=3)))
```

```
##   point      root
## 1     1 -2.856971
## 2     2 -1.355626
## 3     3  0.000000
## 4     4  1.355626
## 5     5  2.856971
```

(d)

Find the quadrature weights. Plot the weights versus the nodes.

```r
# set the nodes
nodes=c(bisection.function(f=H5, a=-3, b=-2),
        bisection.function(f=H5, a=-2, b=-1),
        bisection.function(f=H5, a=-1, b=1),
        bisection.function(f=H5, a=1, b=2),
        bisection.function(f=H5, a=2, b=3))
```

```r
fun_qw = function(x) {
  c5 = 1/sqrt(120*sqrt(2*pi))
  c6 = 1/sqrt(720*sqrt(2*pi))
  qw = - c6 / (c5 * c6 * (x^6 - 15*x^4 + 45*x^2 - 15) * c5 * (5*x^4 - 30*x^2 + 15))
  return(qw)
}

data.frame(points=c(1,2,3,4,5),
           nodes=nodes,
           weights=fun_qw(nodes)/sum(fun_qw(nodes)))
```

```
##   points     nodes    weights
## 1      1 -2.856971 0.01125739
## 2      2 -1.355626 0.22207595
## 3      3  0.000000 0.53333334
## 4      4  1.355626 0.22207595
## 5      5  2.856971 0.01125739
```

(e)

Q4 (e).

We know that $\bar{x} = 47$, $\bar{x}|\mu \sim N(\mu, \frac{50}{10})$

So $f(\bar{x}|\mu) = \frac{1}{\sqrt{2\pi 5}} \cdot e^{-\frac{(\bar{x}-\mu)^2}{2\cdot 5}}$

The prior: $y = \frac{\mu-50}{8} \sim t_{(df=1)} = \text{Cauchy}(0,1)$

$f(y) = \frac{1}{\pi(1+y^2)}$

Then we need to perform variable transformation

$y = \frac{\mu-50}{8} \Rightarrow \mu = 8y+50, \quad d\mu = 8\,dy$

$f(\mu) = \frac{1}{\pi[1+(\frac{\mu-50}{8})^2]} \cdot \frac{1}{8} = \text{Cauchy}(50, 8)$

$c = \int_{-\infty}^{\infty} \frac{1}{\sqrt{10\pi}} e^{-(\bar{x}-\mu)^2/10} \cdot \frac{1}{8}\frac{1}{\pi} \cdot \frac{1}{1+(\frac{\mu-50}{8})^2}\, dy$

$= \frac{1}{\sqrt{128\pi^3}} \left[\sum_{q=1}^{5} \frac{1}{1+(\frac{\sqrt{5}\,node-3}{8})^2} \cdot weight(node)\right] = (\text{normalizing constant})^{-1}$

$Var(\mu|\bar{x}) = E(\mu^2|\bar{x}) - [E(\mu|\bar{x})]^2$

$E(\mu|\bar{x}) = \frac{1}{c}\int_{-\infty}^{\infty} \mu f(\bar{x}|\mu) f(\mu)\, d\mu$

$= \frac{1}{c}\frac{1}{\sqrt{128\pi^3}} \left[\sum_{q=1}^{5} \frac{(\sqrt{5}\,node+47))}{1+(\frac{\sqrt{5}\,node-3}{8})^2} \cdot weight(node)\right]$

Also,

$E(\mu^2|\bar{x}) = \frac{1}{c}\left[\sum_{q=1}^{5} \frac{(\sqrt{5}\,node+47)^2}{1+(\frac{\sqrt{5}\,node-3}{8})^2}\, weight(node)\right]$

We will use these formula to estimate the variance.

```
constant.function = function(x) {
  1/(1 + ((sqrt(5)*x - 3)/8)^2)
}
weights = fun_qw(nodes)/sum(fun_qw(nodes))
# Calculate c
c = 1/sqrt(128*pi^3)*sum(weights*constant.function(nodes))

mu.function = function(x){
  (sqrt(5)*x + 47)/(1 + ((sqrt(5)*x - 3)/8)^2)
```

```
}
mu2.function = function(x) {
  (sqrt(5)*x + 47)^2/(1 + ((sqrt(5)*x - 3)/8)^2)
}
mu.expectation = (1/c) * (1/sqrt(128*pi^3)) * sum(weights * mu.function(nodes))
mu2.expectation = (1/c) * (1/sqrt(128*pi^3)) * sum(weights * mu2.function(nodes))
var_mu = mu2.expectation - (mu.expectation)^2
var_mu
```

```
## [1] 4.535849
```