Computing_HW4_2020321163_엄상준

5.1

5.1.

$P_i(x) = \sum_{j=0}^{m} f(x_{ij}^*) P_{ij}(x)$

In Trapezoidal Rule, $m=1$. $x_{i0}^* = x_i$, $x_{i1}^* = x_{i+1}$

$$P_{i0}(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}}, \quad P_{i1}(x) = \frac{x - x_i}{x_{i+1} - x_i}$$

$$\therefore P_i(x) = f(x_{i0}^*) P_{i0}(x) + f(x_{i1}^*) P_{i1}(x)$$

$$= f(x_i) \frac{x - x_{i+1}}{x_i - x_{i+1}} + f(x_{i+1}) \frac{x - x_i}{x_{i+1} - x_i}$$

$$= f(x_i) \frac{x_{i+1} - x}{x_{i+1} - x_i} + f(x_{i+1}) \frac{x - x_i}{x_{i+1} - x_i}$$

$$= f(x_i) \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right) + f(x_{i+1}) \frac{x - x_i}{x_{i+1} - x_i}$$

$$= f(x_i) + (x - x_i) \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2 + \text{Residual}$ By Taylor Eqansion

let $x = x_{i+1}$

$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i)^2 + \text{Residual}$.

$f(x_{i+1}) - f(x_i) = f'(x_i)(x_{i+1} - x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i)^2 + \text{Residual}$.

$\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = f'(x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i) + \text{Residual}$

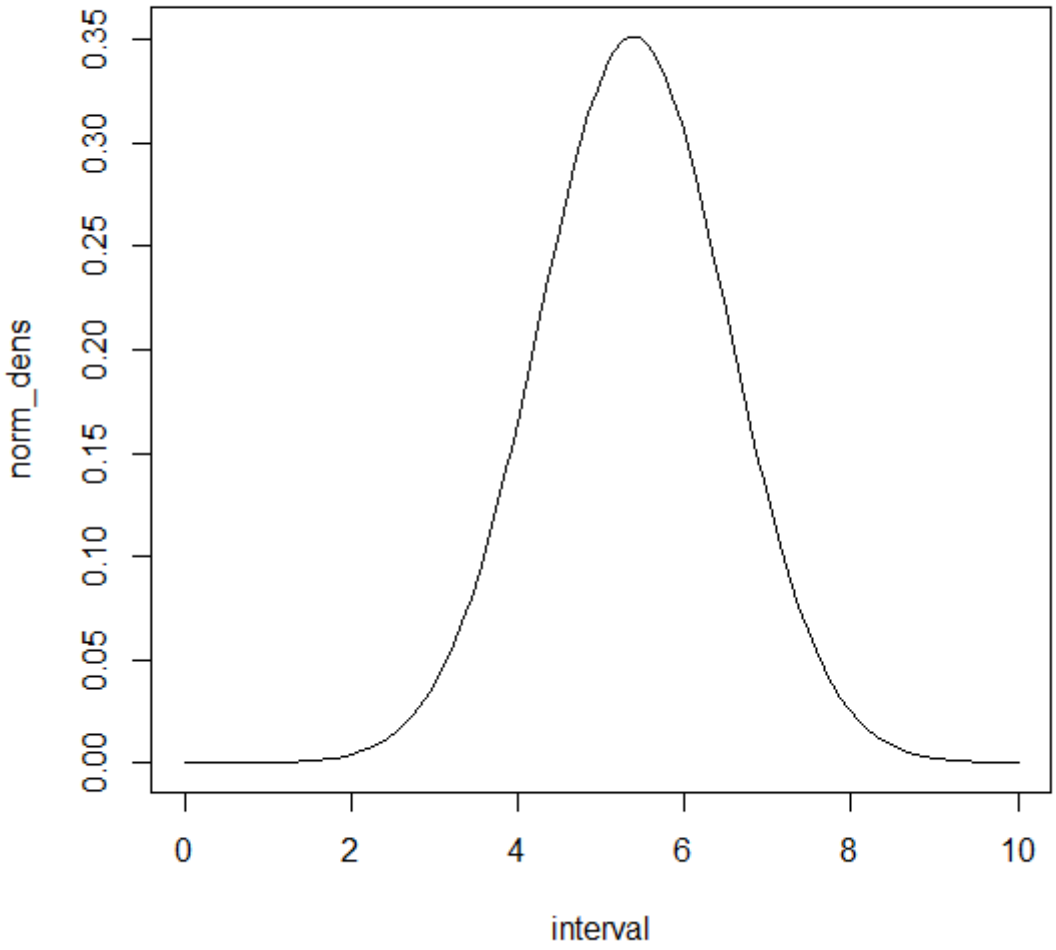
$\therefore P_i(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i)(x - x_i) + O(n^{-3})$

5.3

(a)

```
#data
x = c(6.52, 8.32, 0.31, 2.82, 9.96, 0.14, 9.64)
x_mean <- mean(x)
sd <- sqrt(9/7)

#범위를 대략적으로 지정해주기 위해서 plot을 한 번 그려보자.
interval <- seq(0,10, length.out = 100)
norm_dens <- dnorm(interval, mean=x_mean, sd=sd)
plot(interval, norm_dens, type='l')
```



대략 0에서 10의 범위를 지정해주면 될 것 같다.

```
interval <- c(0,10)
```

cpp file

```
// cpp에는 normpdf를 구해주는 함수는 있지만 cauchy pdf를 구해주는 함수는 없는 듯 하여
cauchy pdf를 구해주는 함수를 먼저 만들어주었다.
double cauchy_pdf(double x, double a, double b){
  double y = (x - a) / b;
  double res = 1.0 / (datum::pi * b * (1.0 + y * y));
  return(res);
}


vec cauchy_pdf_vec(vec x, double a, double b){
  vec res;
  vec y;
  vec a_vec(x.n_rows);
  vec b_vec(x.n_rows);
  vec pi_vec(x.n_rows);
  vec ones(x.n_rows, fill::ones);
  a_vec.fill(a);
  b_vec.fill(b);
  pi_vec.fill(datum::pi);
  y = ( x - a_vec ) / b_vec;
  res = ones / ( pi_vec % b_vec % ( ones + y % y ) );
  return res;
}

// 그 다음으로는 Likeliehood * Prior 의 함수를 만들어주었다.
double f(double x, double mean, double sd){
  double res = normpdf(x, mean, sd) * cauchy_pdf(x, 5.0, 2.0);
  return res;
}

vec f_vec(vec x, double mean, double sd){
  vec res = normpdf(x, mean, sd) % cauchy_pdf_vec(x, 5.0, 2.0);
  return res;
}

// 그 다음으로 Riemann, Trapezoidal, Simpsons quadarture를 시행해주는 function을 만들
어주었다.
// [[Rcpp::export]]
double riemann_cpp(NumericVector interval, double n, double mean, double sd){
  vec x(n);
  double h = (interval[1]-interval[0])/n;
  for(int i = 0; i<n; i++){
    x[i] = interval[0] + i*h;
  }
  double res = h*accu(f_vec(x, mean, sd));
  return(res);
}


// [[Rcpp::export]]
double trapezoidal_cpp(NumericVector interval, double n, double mean, double sd)
{
  vec x(n-1);
  double a = interval[0];
  double b = interval[1];
```

```
  double h = (b-a)/n;
  for(int i = 1; i < n; i++){
    x[i-1] = a + i*h;
  }
  double low = h/2*f(a, mean, sd);
  double high = h/2*f(b, mean, sd);
  double middle = h*accu(f_vec(x, mean, sd));
  double res = low + high + middle;
  return(res);
}


// [[Rcpp::export]]
double simpsons_cpp(NumericVector interval, double n, double mean, double sd){
  vec x(n+1);
  vec out(n/2);
  double a = interval[0];
  double b = interval[1];
  double h = (b-a)/n;
  for(int i = 0; i < n+1; i++){
    x[i] = a + i*h;
  }
  for(int j = 0; j < n/2; j++){
    out[j] = h/3*f(x[2*j], mean, sd);
    out[j] = out[j] + 4*h/3*f(x[2*j+1], mean, sd);
    out[j] = out[j] + h/3*f(x[2*j+2], mean, sd);
  }
  double res = accu(out);
  return(res);
}
```

이제 미리 설정한 interval을 이용하여 적분을 시행한 뒤 역수를 취해주면 k값을 구해줄 수 있다.

```
1/riemann_cpp(interval, 100, mean(x), sd)
1/trapezoidal_cpp(interval, 100, mean(x), sd)
1/simpsons_cpp(interval, 100, mean(x), sd)
```

결과값:

1 / Riemann : 7.846575

1 / Trapezoidal : 7.846569

1 / Simpsons : 7.846569

→ 세 가지 방법 모두 문제에서 제시한 7.84654 값과 거의 동일하다. 따라서 7.84654를 proportionality constant로 지정해줄 수 있다.

(b)

이번에는 proportionality constant를 알게되었으니 이를 cpp 파일에 반영해주자.

```
double f2(double x, double mean, double sd){
  double res = 7.84654*normpdf(x, mean, sd) * cauchy_pdf(x, 5.0, 2.0);
```

```
  return res;
}

vec f2_vec(vec x, double mean, double sd){
  vec constant(x.n_rows);
  constant.fill(7.84654);
  vec res = constant % normpdf(x, mean, sd) % cauchy_pdf_vec(x, 5.0, 2.0);
  return res;
}

double riemann_cpp2(NumericVector interval, double n, double mean, double sd){
  vec x(n);
  double h = (interval[1]-interval[0])/n;
  for(int i = 0; i<n; i++){
    x[i] = interval[0] + i*h;
  }
  double res = h*accu(f2_vec(x, mean, sd));
  return(res);
}


double trapezoidal_cpp2(NumericVector interval, double n, double mean, double
sd){
  vec x(n-1);
  double a = interval[0];
  double b = interval[1];
  double h = (b-a)/n;
  for(int i = 1; i < n; i++){
    x[i-1] = a + i*h;
  }
  double low = h/2*f2(a, mean, sd);
  double high = h/2*f2(b, mean, sd);
  double middle = h*accu(f2_vec(x, mean, sd));
  double res = low + high + middle;
  return(res);
}

double simpsons_cpp2(NumericVector interval, double n, double mean, double sd){
  vec x(n+1);
  vec out(n/2);
  double a = interval[0];
  double b = interval[1];
  double h = (b-a)/n;
  for(int i = 0; i < n+1; i++){
    x[i] = a + i*h;
  }
  for(int j = 0; j < n/2; j++){
    out[j] = h/3*f2(x[2*j], mean, sd);
    out[j] = out[j] + 4*h/3*f2(x[2*j+1], mean, sd);
    out[j] = out[j] + h/3*f2(x[2*j+2], mean, sd);
  }
  double res = accu(out);
  return(res);
}
```

그 다음 interval도 바뀌었으니, interval도 변경해준다.

```
interval <- c(2,8)
```

그 다음 세 가지 방법들에 대해 table을 만들어보자.

cpp 파일은 다음과 같다.

```
// [[Rcpp::export]]
DataFrame riemann_df(NumericVector interval,
                     double mean,
                     double sd,
                     double tol){
  vector<double> estimation;
  vector<double> rel_err;
  vector<double> sub_interval;
  int iter = 1;
  int n = 2;
  double diff = 1.0;
  double old_est;
  double new_est;
  while(diff > tol){
    if(iter==1){
      double first_est = riemann_cpp2(interval, n, mean, sd);
      estimation.push_back(first_est);
      sub_interval.push_back(n);
      rel_err.push_back(0);
      n = n+2;
      iter = iter+1;
      old_est = first_est;
    }
    else{
      new_est = riemann_cpp2(interval, n, mean, sd);
      diff = abs(new_est - old_est);
      estimation.push_back(new_est);
      sub_interval.push_back(n);
      rel_err.push_back(new_est-old_est);
      n = n+2;
      iter = iter +1;
      old_est = new_est;
    }
  }
  DataFrame res = DataFrame::create(Named("est") = estimation,
                                    Named("sub_int") = sub_interval,
                                    Named("rel_err") = rel_err);
  return(res);
}

// [[Rcpp::export]]
DataFrame trapezoidal_df(NumericVector interval,
                       double mean,
                       double sd,
                       double tol){
  vector<double> estimation;
  vector<double> rel_err;
  vector<double> sub_interval;
  int iter = 1;
```

```
  int n = 2;
  double diff = 1.0;
  double old_est;
  double new_est;
  while(diff > tol){
    if(iter==1){
      double first_est = trapezoidal_cpp2(interval, n, mean, sd);
      estimation.push_back(first_est);
      sub_interval.push_back(n);
      rel_err.push_back(0);
      n = n+2;
      iter = iter+1;
      old_est = first_est;
    }
    else{
      new_est = trapezoidal_cpp2(interval, n, mean, sd);
      diff = abs(new_est - old_est);
      estimation.push_back(new_est);
      sub_interval.push_back(n);
      rel_err.push_back(new_est-old_est);
      n = n+2;
      iter = iter +1;
      old_est = new_est;
    }
  }
  DataFrame res = DataFrame::create(Named("est") = estimation,
                                    Named("sub_int") = sub_interval,
                                    Named("rel_err") = rel_err);
  return(res);
}

// [[Rcpp::export]]
DataFrame simpsons_df(NumericVector interval,
                      double mean,
                      double sd,
                      double tol){
  vector<double> estimation;
  vector<double> rel_err;
  vector<double> sub_interval;
  int iter = 1;
  int n = 2;
  double diff = 1.0;
  double old_est;
  double new_est;
  while(diff > tol){
    if(iter==1){
      double first_est = simpsons_cpp2(interval, n, mean, sd);
      estimation.push_back(first_est);
      sub_interval.push_back(n);
      rel_err.push_back(0);
      n = n+2;
      iter = iter+1;
      old_est = first_est;
    }
    else{
      new_est = simpsons_cpp2(interval, n, mean, sd);
      diff = abs(new_est - old_est);
      estimation.push_back(new_est);
```

```
      sub_interval.push_back(n);
      rel_err.push_back(new_est-old_est);
      n = n+2;
      iter = iter +1;
      old_est = new_est;
    }
  }
  DataFrame res = DataFrame::create(Named("est") = estimation,
                                    Named("sub_int") = sub_interval,
                                    Named("rel_err") = rel_err);
  return(res);
}
```

이제 table을 확인해보자.

est = estimation

sub_int = number of sub interval

rel_err = relative error = {estimation(t+1)-estimation(t)}/estimation(t)

```
riemann_df(interval, mean(x), sd, 0.0001)
trapezoidal_df(interval, mean(x), sd, 0.0001)
simpsons_df(interval, mean(x), sd, 0.0001)
```

```
riemann_df(interval, mean(x), sd, 0.0001)
           est sub_int         rel_err
1  1.2481764       2  0.000000e+00
2  0.9902608       4 -2.066339e-01
3  0.9899524       6 -3.114651e-04
4  0.9917962       8  1.862557e-03
5  0.9928391      10  1.051442e-03
6  0.9934857      12  6.512766e-04
7  0.9939219      14  4.390556e-04
8  0.9942345      16  3.145092e-04
9  0.9944688      18  2.356758e-04
10 0.9946506      20  1.828289e-04
11 0.9947956      22  1.457744e-04
12 0.9949138      24  1.188389e-04
13 0.9950120      26  9.867175e-05
```

```
trapezoidal_df(interval, mean(x), sd, 0.0001)
  est sub_int      rel_err
1 1.2600920       2  0.0000000000
2 0.9962186       4 -0.2094080424
3 0.9939242       6 -0.0023030679
4 0.9947751       8  0.0008560803
5 0.9952222      10  0.0004493861
6 0.9954716      12  0.0002506248
7 0.9956241      14  0.0001531852
8 0.9957239      16  0.0001002583
9 0.9957927      18  0.0000691186
```

```
simpsons_df(interval, mean(x), sd, 0.0001)
  est sub_int      rel_err
1 1.6690577       2  0.000000e+00
2 0.9082608       4 -4.558242e-01
3 1.0081744       6  1.100054e-01
4 0.9942940       8 -1.376792e-02
5 0.9961586      10  1.875336e-03
6 0.9959874      12 -1.718883e-04
7 0.9960337      14  4.647313e-05
```

Riemann 방법이 가장 slow한 방법이라는 것을 알 수 있다. 또한, 참 값인 0.99605와의 차이는

0.99605-0.9950120 = 0.001038 로 거의 동일하다는 것을 알 수 있다.

(c)

우선 변수 변환을 해보자.

$$\int_{\frac{\exp(3)}{1+\exp(3)}}^{1} \frac{1}{u(1-u)} f\left(\log\left(\frac{u}{1-u}\right)\right) du$$

여기서 f는 위에서 사용했던 f(mu) function이다.

이제 f(log(u/(1-u))/u(1-u) = g(u) 라고 두고 새로운 function을 세워보자.

```
double g(double x, double mean, double sd){
  double res = f2(log(x/(1-x)), mean, sd) / ( x*(1-x) );
  return res;
}
```

그런데 위의 함수는 1에서 singularity를 가진다. 따라서 1를 무시해 줄 필요가 있는데, trapezoidal rule을 이용해서 설명하면,

Trapezoidal Rule의 공식은 다음과 같다.

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

이 문제에서는 b=1에 해당하며, h*f(b)/2 값이 Nan값으로 나오지 않기 때문에 문제가 발생한다. 따라서 이 값을 그냥 무시해버리기로 하자.

그렇다면,

원래의 cpp function에서 high에 해당하는 값을 제외해주어야 한다.

```
// [[Rcpp::export]]
double trapezoidal_cpp3(NumericVector interval, double n, double mean, double
sd){
  vec x(n-1);
  double a = interval[0];
  double b = interval[1];
  double h = (b-a)/n;
  double middle = 0;
  for(int i = 1; i < n; i++){
    x[i-1] = a + i*h;
  }
  double low = h/2*g(a, mean, sd);
  for(int j = 0; j < n-1; j++){
    middle = middle + g(x[j], mean, sd);
  }
  middle = h*middle;
  double res = low + middle;
  return(res);
}
```

새로운 cpp function을 이용해서 근사값을 구해보면,

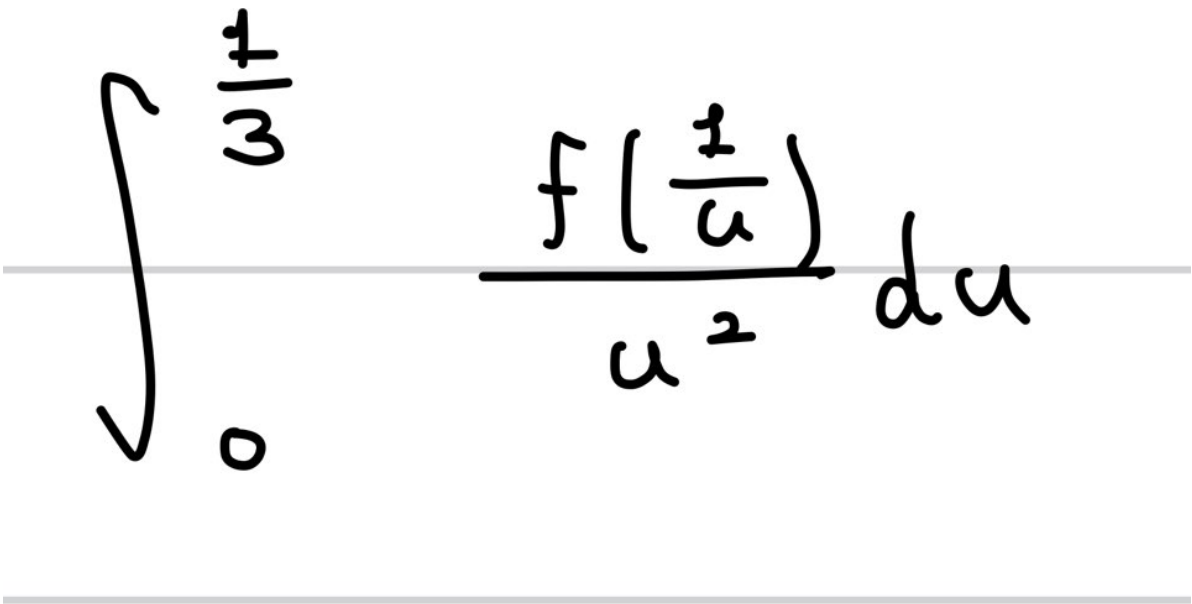
```
#ignoring
interval <- c(exp(3)/(1+exp(3)), 1)
trapezoidal_cpp3(interval, 100, x_mean, sd)
```

> trapezoidal_cpp3(interval, 100, x_mean, sd)
> [1] 0.9907884

0.99086과 거의 차이가 없다는 것을 알 수 있다.

(d)

이번에도 변수 변환을 해보자.



f(1/u)/u^2 을 새로운 function으로 지정해주자.

```
double g2(double x, double mean, double sd){
  double res = f2(1/x, mean, sd)/pow(x,2);
  return res;
}
```

이번에는 구간에서 문제가 되는 값이 0이다. 따라서 high가 아니라 low를 무시해주자.

```
// [[Rcpp::export]]
double trapezoidal_cpp4(NumericVector interval, double n, double mean, double
sd){
  vec x(n-1);
  double a = interval[0];
  double b = interval[1];
  double h = (b-a)/n;
  double middle = 0;
  for(int i = 1; i < n; i++){
    x[i-1] = a + i*h;
  }
  double high = h/2*g2(b, mean, sd);
  for(int j = 0; j < n-1; j++){
    middle = middle + g2(x[j], mean, sd);
  }
  middle = h*middle;
  double res = high + middle;
  return(res);
}
```

결과값은

```
interval <- c(0, 1/3)
trapezoidal_cpp4(interval, 100, x_mean, sd)
```

```
trapezoidal_cpp4(interval, 100, x_mean, sd)
[1] 0.990854
```

마찬가지로 거의 근접한 값이 나온다는 것을 확인할 수 있다.

5.4

우선 f function은 1/x 꼴이 나오므로 f function부터 cpp에서 설정해준다.

```
double f3(double x){
  return 1/x;
}

double f3_sum(double a, int n){
  double s = 0;
  double h = (a-1)/n;
  for(int i = 1; i < n+1; i++){
    s = s + f3(1 + (i-0.5)*h);
  }
  return s;
}
```

그 다음 trianglular array를 만들어주는 function을 만들어주자.

```
// [[Rcpp::export]]
mat triangle(double a, int m){
  mat T(m+1, m+1);

  T(0,0) = 0.5*(f3(1)+f3(a));
  for(int i = 1; i < m+1; i++){
    T(i,0) = 0.5 * T(i-1,0) + f3_sum(a, pow(2, i-1))*(a-1)/pow(2.0,i);
    for(int j = 1; j < i+1; j++){
      T(i,j) = ( pow(4,j)*T(i,j-1) - T(i-1,j-1) ) / ( pow(4,j) -1 );
    }
  }
  return T;
}
```

a=5라고 했을 때 triangular array는 다음과 같다.

```
round(triangle(5, 6),7)
```

```
round(triangle(5, 6),7)
          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
[1,] 0.6000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
[2,] 0.9666667 1.088889 0.000000 0.000000 0.000000 0.000000 0.000000
```

```
[3,] 1.2333333 1.322222 1.337778 0.000000 0.000000 0.000000 0.000000
[4,] 1.4039683 1.460847 1.470088 1.472188 0.000000 0.000000 0.000000
[5,] 1.5019063 1.534552 1.539466 1.540567 1.540836 0.000000 0.000000
[6,] 1.5544359 1.571946 1.574439 1.574994 1.575129 1.575162 0.000000
[7,] 1.5816253 1.590688 1.591938 1.592216 1.592283 1.592300 1.592304
```

맨 끝의 값을 log(5)와 비교해보자.

```
log(5)
[1] 1.609438
```

거의 동일하다는 것을 알 수 있다.

첫 열을 보면 밑으로 갈 수록 원래의 값인 log5에 가까워진다는 것을 알 수 있다. 또한 각 행에서 인접한 값들끼리의 값 차이 또한 밑으로 갈수록 그리고 오른쪽으로 갈수록 더 작아진다는 것을 알 수 있다.