Information Theory for Data Science

Group 17

Pouria Mohammadalipourahari s327015

Reihaneh Kharazmi s328816

Arezoo Parsian s327860

Armi Okshtuni s328780

Prof. Taricco

27 December 2023

1. Distortion

Pdf of random variable x:

$$f(x) = \frac{1}{2}e^{-|x|}$$

Thresholds and quantised values:

$$(2k - N - 1)h\Big|_{k=1}^{N}, \qquad (2k - N - 2)h\Big|_{k=1}^{N+1}$$

Threshold $k = (2k - N - 1)h$ for $k = 1$ to N

Quantization Point $k = (2k - N - 2)h$ for $k = 1$ to $N + 1$

○ Calculate Distortion:

$$D = E\left[(X - \hat{X})^2\right]$$

For quantization, we'll assume that $\hat{X}$ is the quantization point ($\text{Quantization Point}_k$) that is closest to the original value of X. Therefore, $\hat{X}$ will be the quantization point corresponding to the interval in which X falls.

Now, the distortion term $(X - \hat{X})^2$ for each interval can be expressed as follows:

$$D_k = \int_{\text{Interval}_k} (x - \text{Quantization Point}_k)^2 \cdot f(x)\, dx$$

The overall distortion D is the sum of the distortions for all intervals:

$$D = \frac{1}{2}\sum_{k=1}^{N} D_k$$

We substitute the expressions for the quantization points and the PDF into the distortion formula and simplify. The intervals Interval_k are determined by the thresholds.

After simplifying the expression, the distortion for the scalar case, using the given threshold and quantization point formulas, is as follows:

$$D = \frac{1}{2}\sum_{k=1}^{N}\int_{\text{Threshold}_{k-1}}^{\text{Threshold}_k} (x-(2k-N-2)h)^2 \cdot \frac{1}{2}e^{-|x|}\,dx$$

This formula depends on h and N, and it captures the distortion for the specified quantization scheme based on the provided threshold and quantization point formulas.

$$D = \frac{1}{2}\sum_{k=1}^{N}\int_{(2k-N-3)h}^{(2k-N-1)h} (x-(2k-N-2)h)^2 \cdot \frac{1}{2}e^{-|x|}\,dx$$

For $x \le (2k-N-2)h$:

$$D_{k1} = \frac{1}{2}\int_{(2k-N-3)h}^{(2k-N-1)h} (x-(2k-N-2)h)^2 \cdot \frac{1}{2}e^{x}\,dx$$

For $x > (2k-N-2)h$:

$$D_{k2} = \frac{1}{2}\int_{(2k-N-3)h}^{(2k-N-1)h} (x-(2k-N-2)h)^2 \cdot \frac{1}{2}e^{-x}\,dx$$

The overall distortion is the sum of D_{k1} and D_{k2} for each k:

$$D = \frac{1}{2}\sum_{k=1}^{N}(D_{k1} + D_{k2})$$

For the case $x \leq (2k - N - 2)h$:

$$D_{k1} = \frac{1}{2}\int_{(2k-N-3)h}^{(2k-N-1)h}(x - (2k - N - 2)h)^2 \cdot \frac{1}{2}e^x\,dx$$

To solve this integral, we expanded the square inside the integral and then perform the integration. The integral becomes:

$$D_{k1} = \frac{1}{4}\int_{(2k-N-3)h}^{(2k-N-1)h}(x^2 - 2(2k - N - 2)hx + (2k - N - 2)h)^2 e^x\,dx$$

Now, integrate each term separately:

$$D_{k1} = \frac{1}{4}\left[\frac{1}{3}x^3 - (2k - N - 2)hx^2 + (2k - N - 2)hx\right]_{(2k-N-3)h}^{(2k-N-1)h}$$

Then we evaluated the expression at the upper and lower limits of integration and subtract:

$$D_{k1} = \frac{1}{4}\left[\frac{1}{3}((2k - N - 1)h)^3 - (2k - N - 2)h((2k - N - 1)h)^2 + (2k - N - 2)h(2k - N - 3)h^2\right]$$

For the case $x > (2k - N - 2)h$:

$$D_{k2} = \frac{1}{2}\int_{(2k-N-3)h}^{(2k-N-1)h}(x - (2k - N - 2)h)^2 \cdot \frac{1}{2}e^{-x}\,dx$$

To solve this integral, we expanded the square inside the integral and then performed the integration. The integral becomes:

$$D_{k2} = \frac{1}{4} \int_{(2k-N-3)h}^{(2k-N-1)h} (x^2 - 2(2k - N - 2)hx + (2k - N - 2)h)^2 e^{-x}\, dx$$

Now, like before:

$$D_{k2} = \frac{1}{4} \left[\frac{1}{3} x^3 - (2k - N - 2)hx^2 + (2k - N - 2)hx \right]_{(2k-N-3)h}^{(2k-N-1)h}$$

$$D_{k2} = \frac{1}{4} \left[\frac{1}{3}((2k - N - 1)h)^3 - (2k - N - 2)h((2k - N - 1)h)^2 + (2k - N - 2)h(2k - N - 3)h^2 \right]$$

Now we sum the values of D_{k1} and D_{k2} for all$k$ to obtain the overall distortion D:

$$D = \frac{1}{2} \sum_{k=1}^{N} (D_{k1} + D_{k2})$$

Then we substitute the expressions for D_{k1} and D_{k2} into the sum:

$$D = \frac{1}{2} \sum_{k=1}^{N} \left[\frac{1}{4} \left(\frac{1}{3} \left((2k - N - 1)h\right)^3 - (2k - N - 2)h \left((2k - N - 1)h\right)^2 + (2k - N - 2)h(2k - N - 3)h^2 \right) + \frac{1}{4} \left(\frac{1}{3} \left((2k - N - 1)h\right)^3 - (2k - N - 2)h \left((2k - N - 1)h\right)^2 + (2k - N - 2)h(2k - N - 3)h^2 \right) \right]$$

And simplify the expression:

$$D = \frac{1}{2} \sum_{k=1}^{N} \left[\frac{1}{2} \left(\frac{1}{3} \left((2k - N - 1)h\right)^3 - (2k - N - 2)h \left((2k - N - 1)h\right)^2 + (2k - N - 2)h(2k - N - 3)h^2 \right) \right]$$

This gave us the overall distortion D in terms of h and N. To calculate the overall distortion D for different values of N and h, we used these values $N = 3{,}5{,}7{,}\ldots{,}31$ and $h = 5/N{,}10/N{,}20/N$ into the formula:

$$D = \frac{1}{2}\sum_{k=1}^{N}\left[\frac{1}{2}\left(\frac{1}{3}\left((2k-N-1)h\right)^{3} - (2k-N-2)h\left((2k-N-1)h\right)^{2} + (2k-N-2)h(2k-N-3)h^{2}\right)\right]$$

For each combination of N and h, we calculated the sum over k and obtain the corresponding value of D.

The rate-distortion function is typically expressed as $R(D)$, where R is the rate and D is the distortion.

In this case, we'll consider the rate as a function of the distortion, $R(D)$. The rate is often given by the formula:

$$R = \frac{\log_2(M)}{N}$$

where M is the number of quantization levels. Given that $M = 2^R$, we can express R in terms of D and N. The distortion D is the term we obtained earlier.

Now, substitute N and h as specified:

$$D(N,h) = \frac{1}{2}\sum_{k=1}^{N}\left[\frac{1}{2}\left(\frac{1}{3}\left((2k-N-1)h\right)^{3} - (2k-N-2)h\left((2k-N-1)h\right)^{2} + (2k-N-2)h(2k-N-3)h^{2}\right)\right]$$

To obtain numerical values, we substitute specific values of N and h into this expression. For example, for $N = 3$ and $h = \frac{5}{3}$:

$$D(3,5/3) = \frac{1}{2}\left[\frac{1}{2}\left(\frac{1}{3}\left((2k-4)\frac{5}{3}\right)^{3} - (2k-3)\frac{5}{3}\left((2k-4)\frac{5}{3}\right)^{2} + (2k-3)\frac{5}{3}(2k-5)\left(\frac{5}{3}\right)^{2}\right)\right]$$

This expression needs to be further simplified and summed over k. Numerical methods or computational tools can be used for such evaluations.

For k = 1:

$$D(3,5/3) = \frac{1}{4} \cdot \frac{125}{27} \left(\frac{1}{3} \cdot (2(1) - 4)^3 - 3 \cdot (2(1) - 4)^2 + (2(1) - 5) \cdot \frac{25}{9} \right)$$

For k = 2:

$$D(3,5/3) = \frac{1}{4} \cdot \frac{125}{27} \left(\frac{1}{3} \cdot (2(2) - 4)^3 - 3 \cdot (2(2) - 4)^2 + (2(2) - 5) \cdot \frac{25}{9} \right)$$

For k = 3:

$$D(3,5/3) = \frac{1}{4} \cdot \frac{125}{27} \left(\frac{1}{3} \cdot (2(3) - 4)^3 - 3 \cdot (2(3) - 4)^2 + (2(3) - 5) \cdot \frac{25}{9} \right)$$

Then we should some them up and get the final result for analytical calculations.

And we do the same for all the D(N,h).

For this we implemented a python code to calculate each D(N,h) faster:

```
# Define the expression for D
D_expression = Rational(1, 2) * summation(Rational(1, 2) * (
    Rational(1, 3) * ((2 * k - N - 1) * h)**3 -
    (2 * k - N - 2) * h * ((2 * k - N - 1) * h)**2 +
    (2 * k - N - 2) * h * (2 * k - N - 3) * h**2
), (k, 1, N))

# Substitute specific values for N and h
N_values = list(range(3, 32, 2))
h_values = [Rational(5, N_val) for N_val in N_values] + [Rational(10, N_val) for
N_val in N_values] + [Rational(20, N_val) for N_val in N_values]

# Evaluate D for all combinations of N and h
```

```
D_results = [[D_expression.subs({N: N_val, h: h_val}).evalf() for h_val in
h_values] for N_val in N_values]

# Print the results
for i, N_val in enumerate(N_values):
    for j, h_val in enumerate(h_values):
        print(f"D({N_val}, {h_val}) = {D_results[i][j]}")
```

And the results are like:

```
D(3, 5/3) = 25.4629629629630
D(3, 1) = 5.50000000000000
D(3, 5/7) = 2.00437317784257
D(3, 5/9) = 0.943072702331962
...
```

And to plot the rate distortion for them we implemented a python code which computes the distortion for a given threshold and then calculates distortion values for different N and h. Finally, it plots the Rate-Distortion Function using matplotlib.

We defined a probability density function (PDF) for a standard normal distribution. The function takes a variable **x** and returns the PDF value at that point:

```
    def pdf(x):
        return 1 / (2 * np.pi) * np.exp(-np.abs(x) / 2)
```

integrate_trapezoidal function performs numerical integration using the trapezoidal rule. It takes a function **f**, integration bounds **a** and **b**, and an optional parameter **N** for the number of intervals (default is 1000). It returns the approximate integral value:

```
def distortion(threshold):
    integrand = lambda x: x**2 * pdf(x)
    result = integrate_trapezoidal(integrand, threshold[1], threshold[0])
    return result
```

The **distortion** function computes the distortion for a given threshold. It uses the numerical integration function with the integrand **x**2 * pdf(x)** over the specified threshold:

```
def distortion(threshold):
    integrand = lambda x: x**2 * pdf(x)
    result = integrate_trapezoidal(integrand, threshold[1], threshold[0])
    return result
```

The **compute_distortions** function calculates distortions for different **k** values given **N** and **h**. It iterates through the range of **k** values and computes distortion using the previously defined functions:

```
def compute_distortions(N_val, h_val):
    distortions = []
    for k_val in range(1, N_val + 1):
        threshold = ((2 * k_val - N_val - 1) * h_val, (2 * k_val - N_val - 2) * h_val)
        distortion_val = distortion(threshold)
        distortions.append(distortion_val)
    return distortions
```

We specify the parameters, including a range of **N** values and corresponding **h** values:

```
Ns = list(range(3, 32, 2))  # N = 3, 5, 7, ..., 31
hs = [5 / N for N in Ns]     # h = 5/N, 10/N, 20/N
```

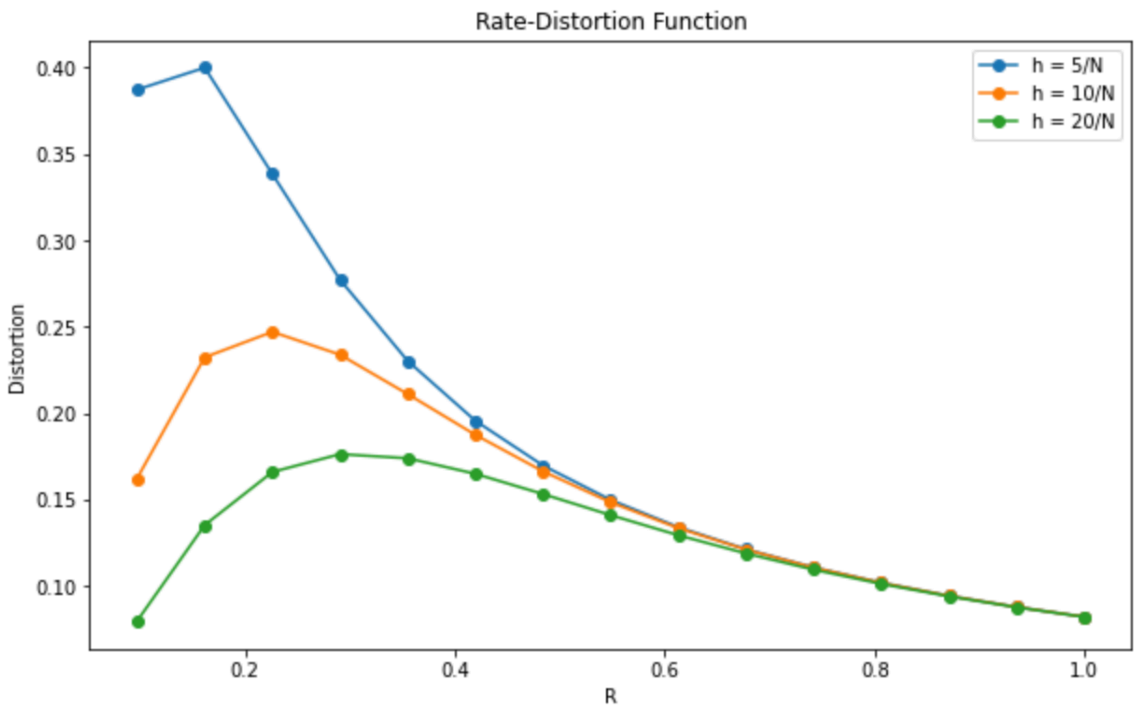
And we calculated distortion values for different **N** and **h** combinations using list comprehensions and NumPy array operations:

```
distortion_values = np.array([[np.mean(compute_distortions(N_val, h_val))
for N_val in Ns] for h_val in hs])
```

Then we transform N to R by normalising Ns:

```
R_values = np.array(Ns) / max(Ns)
```

Finally, we plot the Rate-Distortion Function using Matplotlib. It iterates over different **h** values and plots distortion against the normalized **R** values:



The plot clearly shows the inverse relationship between the rate (R) and distortion. As the rate increases, the distortion decreases, and vice versa. The plot includes three curves for different values of h, specifically h=5/N, h=10/N, and h=20/N. The different curves represent different levels of quantization or compression, and they demonstrate how varying affects the rate-distortion trade-off.

2. Distortion in two dimensions

Pdf of random variable x:

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

Thresholds and quantised values:

$$(2k - N - 1)h\Big|_{k=1}^{N} , \qquad (2k - N - 2)h\Big|_{k=1}^{N+1}$$

Threshold $k = (2k - N - 1)h$ for $k = 1$ to N

Quantization Point $k = (2k - N - 2)h$ for $k = 1$ to $N + 1$

As earlier we discussed we know how to calculate distortion for every N and h. Therefore we do the same steps in the previous exercise.

We implemented a python code to calculate the distortion as a function of N and h:

```
#distortion function in terms of N and h
def distortion(N, h):
    #function to calculate quantization thresholds for given N and h
    def quantization_thresholds(N, h):
        return [(2*k - N - 1) * h for k in range(1, N + 2)]

    #function to calculate quantized values for given N and h
    def quantized_values(N, h):
        return [(2*k - N - 2) * h for k in range(1, N + 2)]

    #obtain quantization thresholds and values
```

```
        thresholds_x = quantization_thresholds(N, h)
        values_x = quantized_values(N, h)

        #initialize distortion sum
        distortion_sum = 0

        #calculate distortion using numerical integration (dblquad)
        for i in range(N):
            lower_x, upper_x = thresholds_x[i], thresholds_x[i + 1]

            mid_x = (values_x[i] + values_x[i + 1]) / 2

            #use dblquad for numerical integration
            distortion_sum += dblquad(lambda x, y: pdf_XY(x, y) * ((x - mid_x)**2),
                                  -np.inf, np.inf, lambda x: lower_x, lambda x: upper_x)
[0]

        return distortion_sum
```

It is fully commented in the file but in summary it computes the distortion function based on the number of quantization levels (**N**) and the quantization step size (**h**). The distortion is calculated by quantizing the random variable X.

There is also a Rate function which computes the rate function based on the number of quantization levels (**N**) and the quantization step size (**h**).

Quantization Functions:

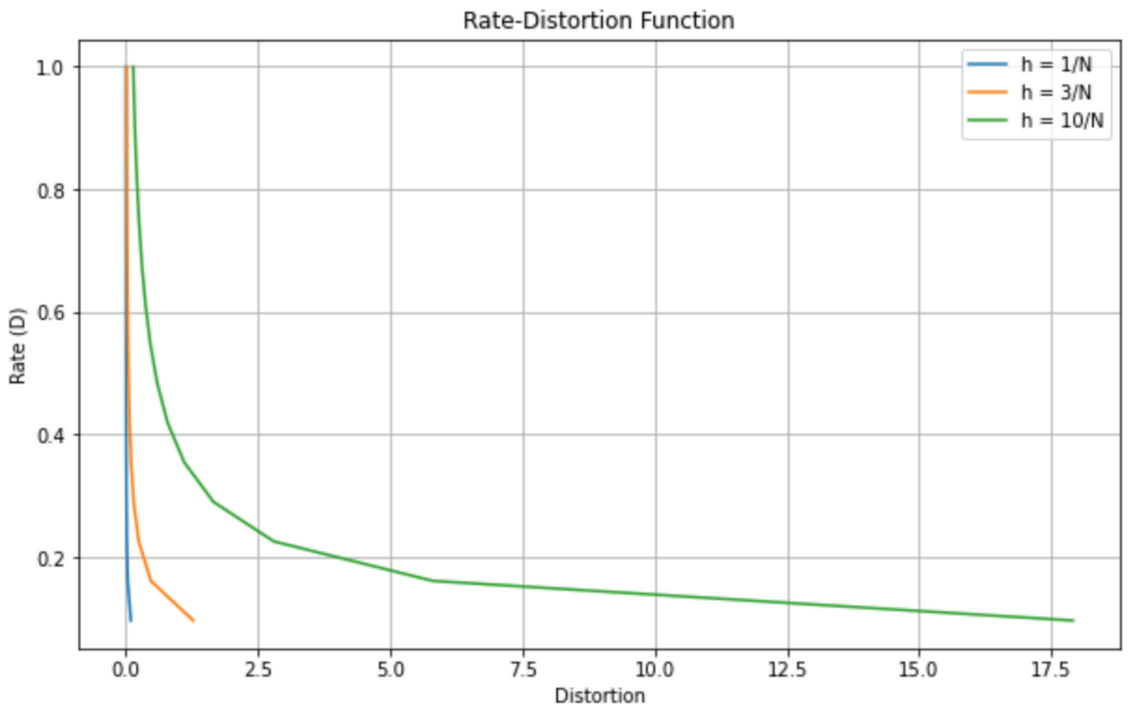
quantization_thresholds(N, h): Calculates the quantization thresholds for a given number of quantization levels (**N**) and step size (**h**).

quantized_values(N, h): Calculates the quantized values for a given number of quantization levels (**N**) and step size (**h**).

Main analysis parts:

It iterates over different values of **N**. For each **N**, calculates distortion for three different values of **h**: **1/N**, **3/N**, and **10/N**. Then transforms **N** into a normalized rate (**R_values**).

Plot:



The plot visually represents the rate-distortion trade-off for different values of **h** (quantization step size). The three curves represent different levels of quantization granularity (**h**). A lower **h** results in finer quantization but potentially higher distortion, while a higher **h** leads to coarser quantization with lower distortion but reduced information efficiency.

3. Lloyd Algorithm

This exercise asks to apply the **Lloyd algorithm** on a pdf (defined bellow) and write a program to find the N thresholds, the (N+1) quantization intervals, the quantization levels, and the average quantization error for three different cases N = 10, 20, 100.

$$f(x) = \frac{1}{2}e^{-|x|}$$

The Lloyd-Max algorithm is an iterative procedure that adjusts quantization intervals and levels to minimize the MSE. Given the symmetric nature of the Laplace distribution, we initialize the algorithm with symmetric intervals and iteratively calculate centroids (quantization levels) and thresholds until convergence.

Centroids (Quantization Levels):

For an interval [a , b], the centroid q is computed as the expected value of X within that interval with respect to the pdf:

$$q = \frac{\int_a^b x f_x(x)\, dx}{\int_a^b f_x(x)\, dx}$$

For x > 0 this simplifies to:

$$q^+ = \frac{\int_a^b x \frac{1}{2} e^{-|x|}\, dx}{\int_a^b \frac{1}{2} e^{-|x|}\, dx}$$

After integration by parts, we get:

$$q^{+} = a + \frac{1 - e^{(a-b)}}{e^{(a-b)} - 1}$$

Average Quantization Error:

The average quantization error for an interval [a , b] is given by:

$$D = \int_{a}^{b} (x - q)^2 f_X(x)\, dx$$

For the positive domain:

$$D^{+} = \int_{a}^{b} (x - q^{+})^2 \frac{1}{2} e^{-x}\, dx$$

The analytical expressions derived allow us to calculate quantization levels and thresholds for any given number of quantization intervals. The average quantization error requires numerical integration due to the complexity of the expression for q^{+}.

Other steps to fulfil the requirements of the question are as follows:

1. Calculate the first (N+1) quantization intervals for each N according to the following formula:

$$\left(-\infty, -\frac{N-1}{2}\right), \ldots, \left(\frac{N-1}{2}, \infty\right)$$

So, at first, we will have N+2 threshold as:

$$(t_1)^{(0)} = -\infty, \quad (t_2)^{(0)} = -\frac{N-1}{2}, \quad \ldots, \quad (t_{N+1})^{(0)} = \frac{N-1}{2}, \quad (t_{N+2})^{(0)} = \infty$$

The related part of the MATLAB code is as follows:

```
% Initialize intervals and thresholds
intervals = cell(1, N(k) + 1);
t = zeros(1,N(k)+2);
t(1) = -inf;
t(2) = -(N(k)-1)/2;
t(N(k)+2) = inf;
intervals{1} = [t(1) , -(N(k)-1)/2];
intervals{N(k) + 1} = [(N(k)-1)/2 , t(N(k)+2)];
j = 2;
  for i = N(k):-2:-N(k)+3
       intervals{j} = [-(i-1)/2, -(i-3)/2];
       t(j+1) = -(i-3)/2;
       j = j+1;
end
```

2. The first series of (N+1) quantization levels are calculated based on the following formula:

$$x_i^{(0)} = \frac{\int_{t_i^{(0)}}^{t_{i+1}^{(0)}} x f_x(x)\,dx}{\int_{t_i^{(0)}}^{t_{i+1}^{(0)}} f_x(x)\,dx} \qquad \text{i} = 1, \ldots, \text{N+1}$$

The related part of the MATLAB code is as follows:

```
% Initialize quantization levels
```

```
quantization_levels = zeros(1,N(k)+1);
for i = 1:N(k)+1 %% pdf = @(x) 0.5 * exp(-abs(x)) //
func = @(x) x .* pdf(x)

quantization_levels(i) = integral(func, t(i),
t(i+1)) / integral( ... pdf, t(i), t(i+1));

end
```

3. Then a loop starts. At this stage we define a convergence threshold.

Code snippet:

```
  % Iterative process
    convergence_threshold = 1e-10;
    new_quantization_levels = zeros(1, N(k) + 1);
    while 1
```

Inside the loop, first the thresholds are updated according to the following formula considering that t1 and tN+2 remain ±∞. Then the new x_i is calculated based on updated thresholds.

$$t_{i+1}^{(n+1)} = \frac{x_i^{(n)} + x_{i+1}^{(n)}}{2}, \quad i = 1, \ldots, N$$

Code snippet:

```
% Compute new thresholds
t(2:N(k)+1) = (quantization_levels(1:end-1) +
quantization_levels(2:end)) / 2;

% Compute new quantization levels
for i = 1:N(k)+1
new_quantization_levels(i) = integral(func, t(i),
t(i+1)) / integral( ...

        pdf, t(i), t(i+1));
end
```

Finally, convergence condition is checked based on the following formula:

$$\|x^{(n+1)} - x^{(n)}\|^2 < \text{convergence threshold}$$

Code snippet:

```
% Convergence check
if (max(abs(new_quantization_levels -
quantization_levels)))^2 < convergence_threshold

     break;

end
```

If the condition is satisfied, the loop stops, and the average quantization error is calculated as follows, and all question requirements are reported.

$$D = \sum_{i=1}^{N+1} \int_{t_i}^{t_{i+1}} (x - x_i)^2 f_x(x)\, dx$$

Code snippet:

```
% Compute average quantization error
avg_error = 0;
for i = 1:N(k)+1
    D = @(x) (x - new_quantization_levels(i)).^2 .*
pdf(x);
    avg_error = avg_error + integral(D, t(i), t(i+1));
end

fprintf('Results for N = %d\n', N(k));
fprintf('Thresholds: %s\n', mat2str(t(2:N(k)+1)));
fprintf('Quantization Levels: %s\n',
mat2str(quantization_levels)); fprintf('Quantization
Intervals: ');
for i = 1:length(intervals)-1
```

```
fprintf('[%g, %g] , ', intervals{i}(1), intervals{i}
(2)); end

fprintf('[%g, %g]\n', intervals{N(k)+1}(1),
intervals{N(k)+1}(2)); fprintf('Average Quantization
Error : %e\n', avg_error); fprintf('\n')
```

The output results in Command Window are as follows:

- Results for N = 10:

```
Thresholds:

[-4.23615466771985 -2.64250554814219 -1.6248952491228
-0.870827899783014 -0.270365822924113 0.270365822924113
0.870827899783013 1.6248952491228 2.64250554814219
4.23615466771985]

Quantization Levels:

[-5.23616458368313 -3.23614475175658 -2.04886634452779
-1.2009241537178 -0.540731645848227 -1.45536310615069e-16
0.540731645848227 1.2009241537178 2.04886634452779
3.23614475175658 5.23616458368313]

Quantization Intervals:

[-Inf, -4.23615] , [-4.23615, -2.64251] , [-2.64251, -1.6249] , [-1.6249,
-0.870828] , [-0.870828, -0.270366] , [-0.270366, 0.270366] ,
[0.270366, 0.870828] , [0.870828, 1.6249] , [1.6249, 2.64251] , [2.64251,
4.23615] , [4.23615, Inf]

Average Quantization Error : 6.116075e-02
```

- Results for N = 20:

Thresholds:

```
[-6.0404225679058 -4.44677237180405 -3.42915706709897
-2.67507764835525 -2.07459364335106 -1.57521451463545
-1.14759079874057 -0.773594008070498 -0.441222498504856
-0.142111819253557 0.142111819253556 0.441222498504855
0.773594008070497 1.14759079874057 1.57521451463545
2.07459364335106 2.67507764835525 3.42915706709897
4.44677237180405 6.0404225679058]
```

Quantization Levels:

```
[-7.04043242696491 -5.04041270884669 -3.8531320347614
-3.00518209943653 -2.34497319727397 -1.80421408942816
-1.34621493984274 -0.948966657638398 -0.598221358502599
-0.284223638507113 -4.09206424980716e-16 0.284223638507112
0.598221358502598 0.948966657638397 1.34621493984274
1.80421408942815 2.34497319727396 3.00518209943653 3.8531320347614
5.04041270884669 7.04043242696491]
```

Quantization Intervals:

```
[-Inf, -6.04042] , [-6.04042, -4.44677] , [-4.44677, -3.42916] , [-3.42916,
-2.67508] , [-2.67508, -2.07459] , [-2.07459, -1.57521] , [-1.57521, -1.14759] ,
[-1.14759, -0.773594] , [-0.773594, -0.441222] , [-0.441222, -0.142112] ,
[-0.142112, 0.142112] , [0.142112, 0.441222] , [0.441222, 0.773594] , [0.773594,
1.14759] , [1.14759, 1.57521] , [1.57521, 2.07459] , [2.07459, 2.67508] , [2.67508,
3.42916] , [3.42916, 4.44677] , [4.44677, 6.04042] , [6.04042, Inf]
```

Average Quantization Error : 1.837456e-02

- Results for N = 100:

Thresholds:

```
[-10.6383209126395 -9.04466984317386 -8.02705173180289
-7.27296587612163 -6.67246970924373 -6.17307032795349
-5.74541577309795 -5.37137507914171 -5.0389442870651 -4.73975692843536
-4.46774548018737 -4.21836797046792 -3.98813518031446
-3.77430813283133 -3.57469711385605 -3.38752385335708
-3.21132443064196 -3.04487925456927 -2.88716152922336
-2.73729863761147 -2.59454274013831 -2.4582480675224 -2.32785315747967
-2.20286679662143 -2.08285677670276 -1.96744081474626
-1.85627915554034 -1.74906849559515 -1.64553695488889
-1.54543988667851 -1.44855636306114 -1.35468620951549
-1.26364748856734 -1.17527435329812 -1.08941520728271 -1.00593111988073
-0.924694455473659 -0.845587682871649 -0.768502337178557
-0.693338111255321 -0.620002057825839 -0.548407886429812
-0.478475341999074 -0.410129653938141 -0.343301046319779
-0.277924301235423 -0.213938368526046 -0.151286016107118
-0.0899135159280364 -0.0297703613007344 0.0297703613007369
0.089913515928039 0.151286016107121 0.213938368526048
0.277924301235426 0.343301046319782 0.410129653938144
0.478475341999077 0.548407886429815 0.620002057825842
0.693338111255325 0.768502337178561 0.845587682871653
0.924694455473663 1.00593111988073 1.08941520728272 1.17527435329813
1.26364748856734 1.35468620951549 1.44855636306114 1.54543988667851
1.64553695488889 1.74906849559515 1.85627915554035 1.96744081474626
2.08285677670276 2.20286679662143 2.32785315747968 2.45824806752241
2.59454274013832 2.73729863761147 2.88716152922336 3.04487925456927
3.21132443064196 3.38752385335708 3.57469711385605 3.77430813283133
3.98813518031446 4.21836797046792 4.46774548018737 4.73975692843536
5.0389442870651 5.37137507914171 5.74541577309795 6.17307032795349
6.67246970924373 7.27296587612163 8.02705173180289 9.04466984317385
10.6383209126395]
```

Quantization Levels:

```
[-11.6383309086502 -9.63831091662875 -8.45102876971896
-7.60307469388683 -6.94285705835642 -6.40208236013104
-5.94405829577595 -5.54677325041995 -5.19597690786348
-4.88191166626673 -4.59760219060399 -4.33788876977075
-4.0988471711651 -3.87742318946381 -3.67119307619885 -3.47820115151325
-3.29684655520091 -3.12580230608301 -2.96395620305553
-2.81036685539118 -2.66423041983176 -2.52485506044486
-2.39164107459994 -2.26406524035941 -2.14166835288344
-2.02404520052207 -1.91083642897044 -1.80172188211025
-1.69641510908005 -1.59465880069773 -1.49622097265929
-1.40089175346299 -1.30848066556798 -1.21881431156669
-1.13173439502955 -1.04709601953588 -0.964766220225579
-0.884622690721739 -0.80655267502156 -0.730451999335554
-0.656224223175087 -0.58377989247659 -0.513035880383035
-0.443914803615113 -0.376344504261168 -0.310257588378389
-0.245591014092457 -0.182285722959634 -0.120286309254603
-0.05954072260147 1.24890472555565e-15 0.0595407226014726
0.120286309254605 0.182285722959636 0.24559101409246
0.310257588378392 0.376344504261171 0.443914803615116
0.513035880383038 0.583779892476593 0.656224223175091
0.730451999335558 0.806552675021563 0.884622690721743
0.964766220225584 1.04709601953588 1.13173439502956 1.21881431156669
1.30848066556799 1.40089175346299 1.4962209726593 1.59465880069773
1.69641510908005 1.80172188211025 1.91083642897045 2.02404520052207
2.14166835288345 2.26406524035941 2.39164107459995
2.52485506044487 2.66423041983176 2.81036685539118
2.96395620305553 3.12580230608301 3.29684655520091
3.47820115151326 3.67119307619885 3.87742318946381 4.0988471711651
4.33788876977075 4.59760219060399 4.88191166626673 5.19597690786348
5.54677325041995 5.94405829577595 6.40208236013103
6.94285705835642 7.60307469388683 8.45102876971895
9.63831091662875 11.6383309086502]
```

Quantization Intervals:

```
[-Inf, -10.6383] , [-10.6383, -9.04467] , [-9.04467, -8.02705] , [-8.02705,
-7.27297] , [-7.27297, -6.67247] , [-6.67247, -6.17307] , [-6.17307, -5.74542] ,
[-5.74542, -5.37138] , [-5.37138, -5.03894] , [-5.03894, -4.73976] , [-4.73976,
-4.46775] , [-4.46775, -4.21837] , [-4.21837, -3.98814] , [-3.98814, -3.77431] ,
[-3.77431, -3.5747] , [-3.5747, -3.38752] , [-3.38752, -3.21132] , [-3.21132,
-3.04488] , [-3.04488, -2.88716] , [-2.88716, -2.7373] , [-2.7373, -2.59454] ,
[-2.59454, -2.45825] , [-2.45825, -2.32785] , [-2.32785, -2.20287] , [-2.20287,
-2.08286] , [-2.08286, -1.96744] , [-1.96744, -1.85628] , [-1.85628, -1.74907] ,
[-1.74907, -1.64554] , [-1.64554, -1.54544] , [-1.54544, -1.44856] , [-1.44856,
-1.35469] , [-1.35469, -1.26365] , [-1.26365, -1.17527] , [-1.17527, -1.08942] ,
[-1.08942, -1.00593] , [-1.00593, -0.924694] , [-0.924694, -0.845588] ,
[-0.845588, -0.768502] , [-0.768502, -0.693338] , [-0.693338, -0.620002] ,
[-0.620002, -0.548408] , [-0.548408, -0.478475] , [-0.478475, -0.41013] ,
[-0.41013, -0.343301] , [-0.343301, -0.277924] , [-0.277924, -0.213938] ,
[-0.213938, -0.151286] , [-0.151286, -0.0899135] , [-0.0899135, -0.0297704] ,
[-0.0297704, 0.0297704] , [0.0297704, 0.0899135] , [0.0899135, 0.151286] ,
[0.151286, 0.213938] , [0.213938, 0.277924] , [0.277924, 0.343301] ,
[0.343301, 0.41013] , [0.41013, 0.478475] , [0.478475, 0.548408] , [0.548408,
0.620002] , [0.620002, 0.693338] , [0.693338, 0.768502] , [0.768502,
0.845588] , [0.845588, 0.924694] , [0.924694, 1.00593] , [1.00593, 1.08942] ,
[1.08942, 1.17527] , [1.17527, 1.26365] , [1.26365, 1.35469] , [1.35469, 1.44856] ,
[1.44856, 1.54544] , [1.54544, 1.64554] , [1.64554, 1.74907] , [1.74907,
1.85628] , [1.85628, 1.96744] , [1.96744, 2.08286] , [2.08286, 2.20287] ,
[2.20287, 2.32785] , [2.32785, 2.45825] , [2.45825, 2.59454] , [2.59454,
2.7373] , [2.7373, 2.88716] , [2.88716, 3.04488] , [3.04488, 3.21132] , [3.21132,
3.38752] , [3.38752, 3.5747] , [3.5747, 3.77431] , [3.77431, 3.98814] , [3.98814,
4.21837] , [4.21837, 4.46775] , [4.46775, 4.73976] , [4.73976, 5.03894] ,
[5.03894, 5.37138] , [5.37138, 5.74542] , [5.74542, 6.17307] , [6.17307,
6.67247] , [6.67247, 7.27297] , [7.27297, 8.02705] , [8.02705, 9.04467] ,
[9.04467, 10.6383] , [10.6383, Inf]
```

```
Average Quantization Error : 8.628250e-04
```

In the following, the complete MATLAB code for this question is represented.

```
clc clear all

% Given pdf

pdf = @(x) 0.5 * exp(-abs(x)); func = @(x) x .*
pdf(x);
N = [10, 20, 100];

for k = 1:length(N)
    % Initialize intervals and thresholds
    intervals = cell(1, N(k) + 1);
    t = zeros(1,N(k)+2);
    t(1) = -inf;
    t(2) = -(N(k)-1)/2;
```

```
t(N(k)+2) = inf;
intervals{1} = [t(1) , -(N(k)-1)/2]; intervals{N(k) +
1} = [(N(k)-1)/2 , t(N(k)+2)]; j = 2;
for i = N(k):-2:-N(k)+3

        intervals{j} = [-(i-1)/2, -(i-3)/2];
        t(j+1) = -(i-3)/2;
        j = j+1;
end

  % Initialize quantization levels
    quantization_levels = zeros(1,N(k)+1);
    for i = 1:N(k)+1
quantization_levels(i) = integral(func, t(i),
t(i+1)) / integral( ... pdf, t(i), t(i+1));

end

    % Iterative process
    convergence_threshold = 1e-10;
    new_quantization_levels = zeros(1, N(k) + 1);
while 1
% Compute new thresholds
t(2:N(k)+1) = (quantization_levels(1:end-1) +
quantization_levels(2:end)) / 2;

        % Compute new quantization levels
for i = 1:N(k)+1
new_quantization_levels(i) = integral(func, t(i),
t(i+1)) / integral( ...

                pdf, t(i), t(i+1));
end

        % Convergence check
if (max(abs(new_quantization_levels -
quantization_levels)) ... )^2 < convergence_threshold

break; end

        quantization_levels = new_quantization_levels;
end

    % Compute quantization intervals
    for i = 1:N(k)+1
        intervals{i} = [t(i), t(i+1)];
```

```
end

    % Compute average quantization error
    avg_error = 0;
    for i = 1:N(k)+1
D = @(x) (x - new_quantization_levels(i)).^2 .*
pdf(x);

        avg_error = avg_error + integral(D, t(i),
t(i+1));
end

fprintf('Results for N = %d\n', N(k));
fprintf('Thresholds: %s\n', mat2str(t(2:N(k)+1)));
fprintf('Quantization Levels: %s\n',
mat2str(quantization_levels)); fprintf('Quantization
Intervals: ');
for i = 1:length(intervals)-1

fprintf('[%g, %g] , ', intervals{i}(1), intervals{i}
(2)); end

fprintf('[%g, %g]\n', intervals{N(k)+1}(1),
intervals{N(k)+1}(2)); fprintf('Average Quantization
Error : %e\n', avg_error); fprintf('\n')

end
```