PRINCIPLES OF MACHINE LEARNING

Exercise Sheet 2

20.11.2023

Richard Restetzki	2973740
Akmalkhon Khashimov	50178353
Valdrin Smakaj	50138041
Suraj Giri	50190564
Nijat Sadikhov	50186266
Nikita Morev	50134788
Aleksandr Semenikhin	50118777
Vibhor Sharma	50010826
Muslimbek Abduvaliev	50136555
Suyash Thapa	50205756
Mohammad Mehdi Deylamipour	50009389
Nicolás López Funes	50151598
Andrii Shevliakov	50198078
Nurmukhammad Aberkulov	50102727

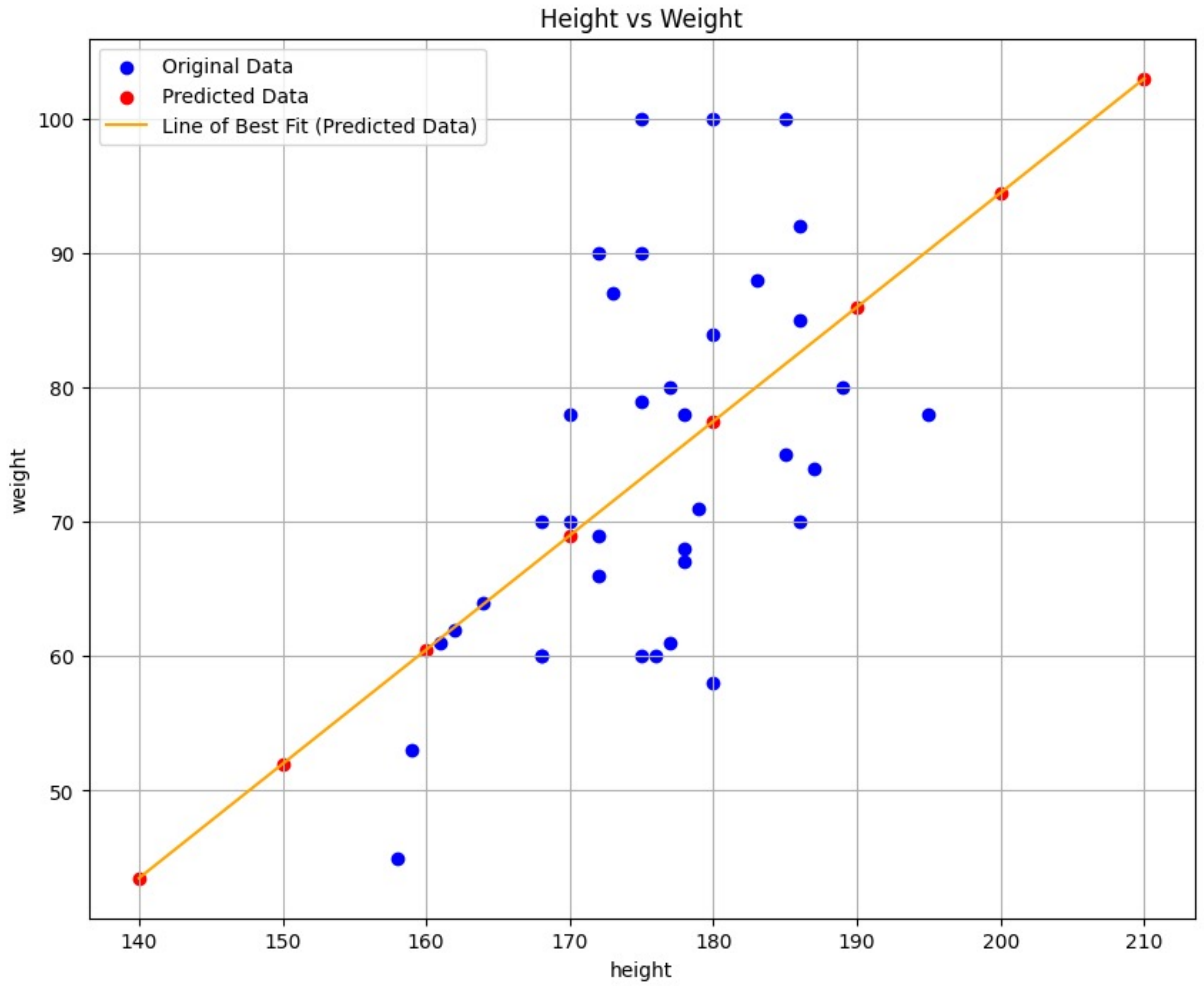
Task 2.1

bivariate Gaussian models and conditional expectations

```
  # Returns the maximum likelihood parameters for a given data matrix X
def likelihood(X):

    mean_value = np.mean(X, axis=0)
    cov_matrix = np.cov(X, rowvar=False)
    return mean_value,cov_matrix
```

```
# Predicting the weight for a given height using the conditional expectation:
 E[w|h] = E[w] + cov(hw) * (h − E[h]) / cov(hh)

def pred(X, mean_values, cov_matrix, h):

 mean_height = mean_values[0]
     mean_weight = mean_values[1]


      h_index = 0
     w_index = 1


    cov_hw = cov_matrix[h_index][w_index]
    cov_hh = cov_matrix[h_index][h_index]

    conditional_expectation = mean_weight + cov_hw * (h − mean_height) / cov_hh

    return conditional_expectation
```

```
# Returns the data array X without outliers

def remove_outliers():

    data = np.loadtxt('whDatadat.sec', dtype=object, comments='#', delimiter=None)
    w = data[:, 0].astype(float)
    h = data[:, 1].astype(float)
    max_dev = 2
    mean = np.mean(w)
    standard_dev = np.std(w)
    outliers_mask = np.abs((w - mean) / standard_dev) > max_dev
    X = np.column_stack((h[~outliers_mask],w[~outliers_mask]))
    return X



# Task 2.1

if __name__ == "__main__":

    data_wthout_outliers = remove_outliers()

    mean_value,cov_matrix = likelihood(data_wthout_outliers)

    h = [140, 150, 160, 170, 180, 190, 200, 210]

    for i in h:
        conditional_mean_w = pred(data_wthout_outliers, mean_value, cov_matrix, i)
```



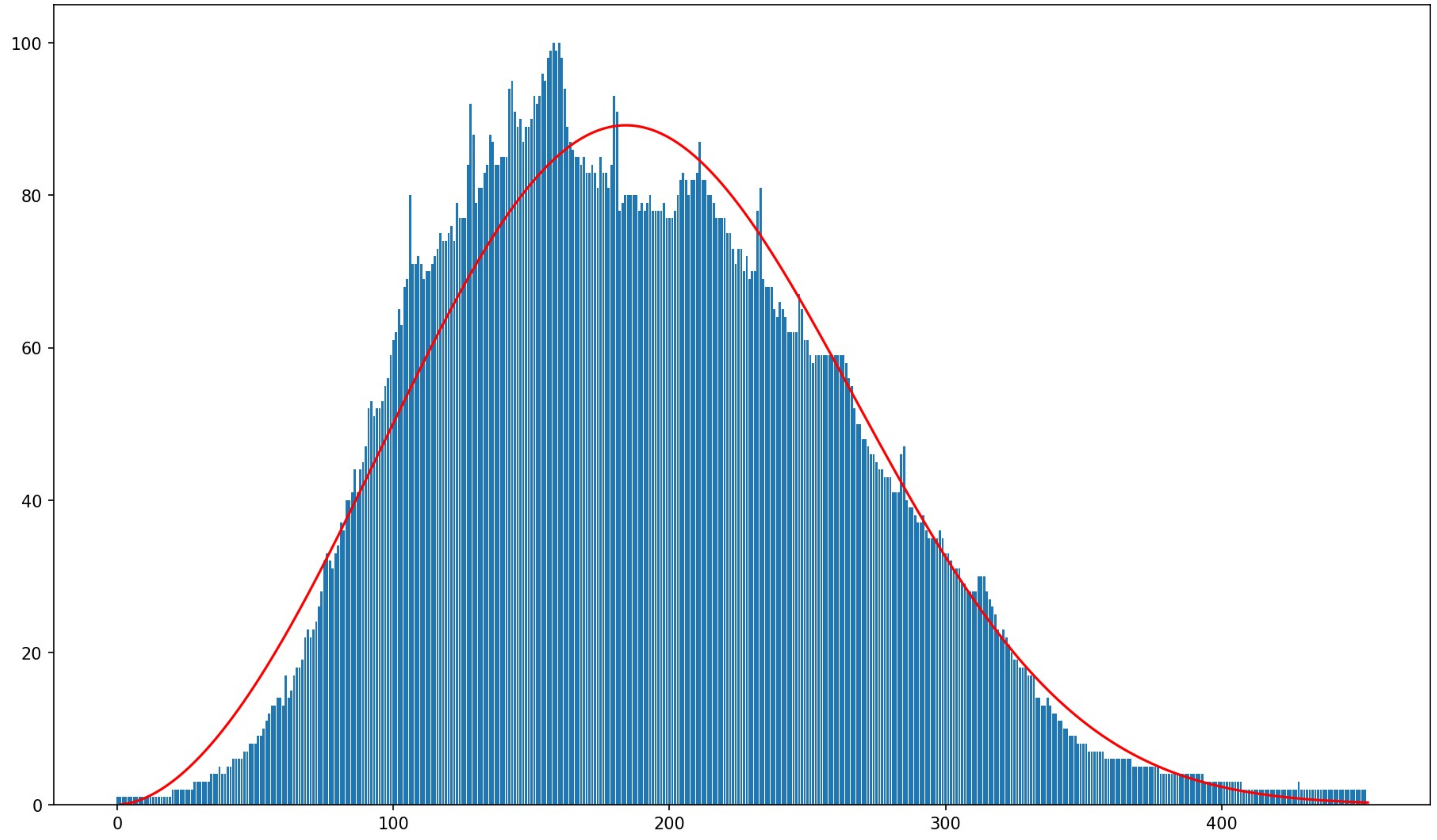
We observed that as the height increases, the weight also increases for the predicted values. Given the daily experience, we think that these results might not be plausible as in natural reality because the weight does not increase as the height increases for most of the cases.

To press on the question of whether the 'plausibility' is something that can be easily quantified, we can say that it is not easily quantified. Depending upon the graph and the data, it might seem like the model can't be plausible in real life scenario.

Task 2.2

fitting a Weibull distribution to a histogram (part 1)

The data from myspace.csv was used to fit it to the Weibull distribution model. We iteratively maximized log_likelihood function provided in the task to find shape and scale parameters for Weibull probability density function.



The “subtlety” we had to address was that the observations used in the provided formulas were always summed, therefore to adapt our data, we only had to multiply each d_i with its corresponding value from h_i – thus it would still sum to what we need without creating the whole data set of observations.

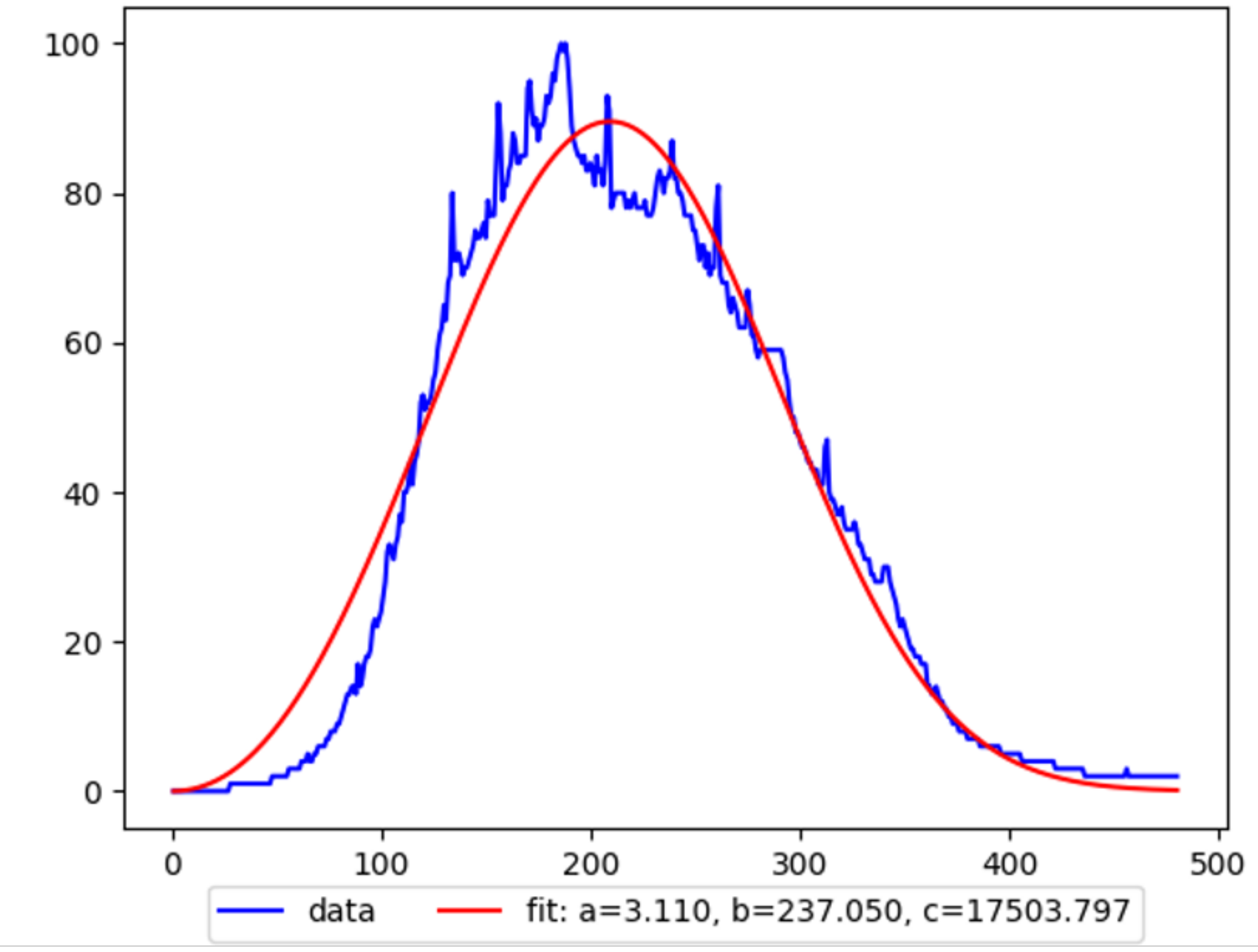
Task 2.3

Fitting a Weibull distribution to a histogram (part 2)

$$\tilde{f}(t \mid A, \alpha, \beta) = A \cdot f(t \mid \alpha, \beta)$$

$$f(t \mid \alpha, \beta) = \frac{\alpha}{\beta}\left(\frac{t}{\beta}\right)^{\alpha-1} e^{-\left(\frac{t}{\beta}\right)^{\alpha}}$$

```
def weibull_pdf(x, shape, scale, amplitude):
    """
    Compute the Weibull distribution probability density function.

    Parameters:
    x : point(s) to evaluate the pdf.
    shape : α
    scale : β
    amplitude: A

    """
    if shape <= 0 or scale <= 0:
        raise ValueError("Shape and scale parameters must be positive.")
    return amplitude * (shape / scale) * ((x / scale) ** (shape - 1)) * np.exp(-(x / scale) ** shape)
.
.
.

h = np.array(value_array)

t = np.arange(1,len(value_array)+1)

popt, pcov = curve_fit(weibull_pdf, t, h, p0=[1, 1, 1000], bounds=([0, 0, 100], [1000., 1000., np.inf]))
for i in range(20):
    popt, pcov = curve_fit(weibull_pdf, t, h, p0=popt, bounds=([0, 0, 100], [1000., 1000., np.inf]))
```

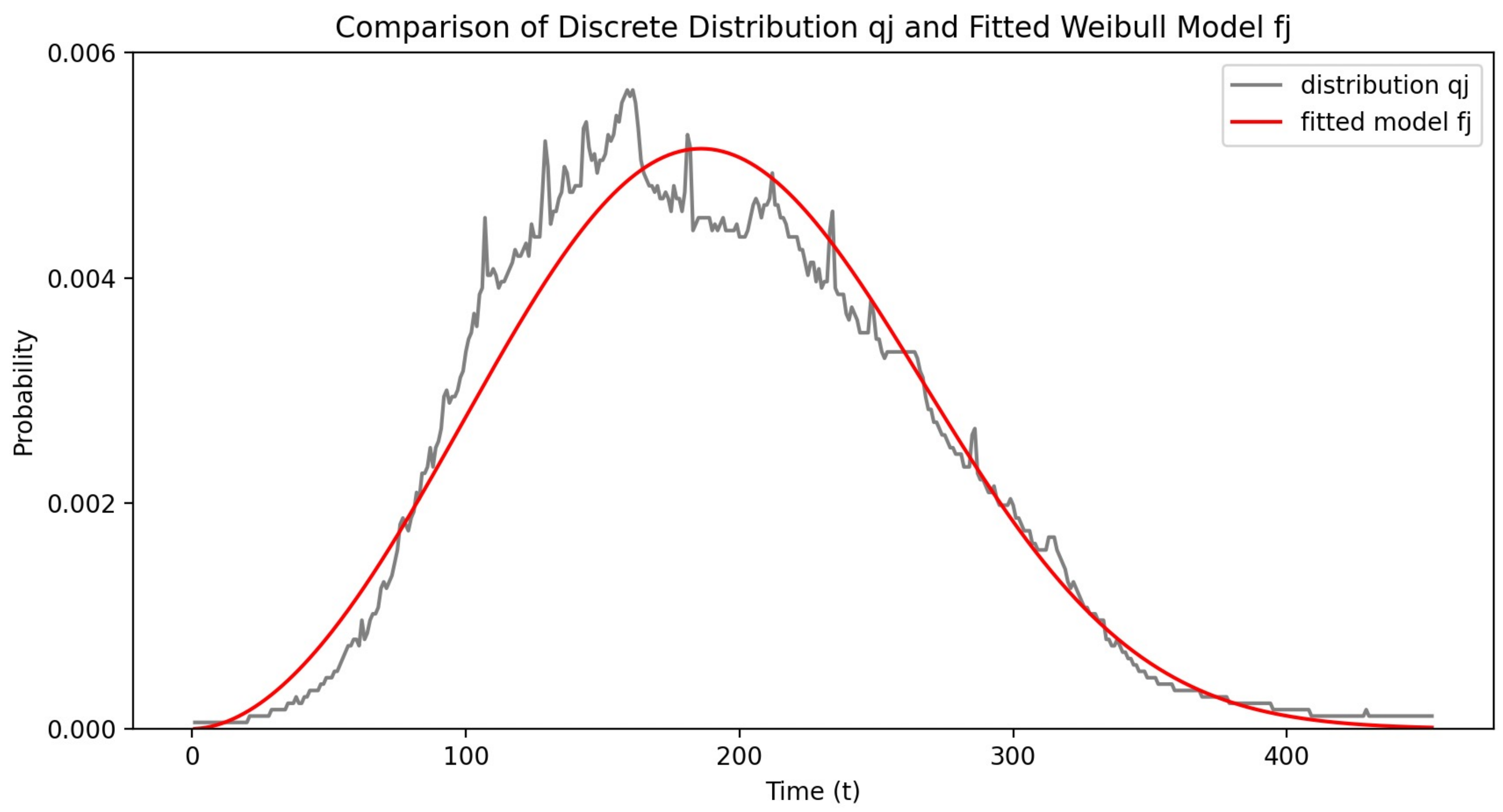


At each iteration of the loop, the curve_fit function refines the parameter estimates (α, β, A) based on the current values of popt and updates popt and pcov with the new estimates and covariance matrix, respectively. This process continues for 20 iterations, potentially improving the parameter estimates with each iteration.

As a result for fitting curve, we have got $\alpha = 3.11$ and $\beta = 237$. Result of plotting resulted curve to the given histogram is a little bit different from the result of Task 2.2

Task 2.5

Fitting a Weibull model to a discrete distribution




```
q = h * 0.98 / np.sum(h)

def kullback_leibler_divergence(params, q, t):
    alpha, beta = params
    f = weibull_distribution(t, alpha, beta)
    # Normalize the Weibull distribution so it sums to 0.98, like q
    f /= np.sum(f)
    f *= 0.98
    # avoid case for div/0 and log(0) by adding a small epsilon
    epsilon = 1e-10
    kl_divergence = np.sum(f * np.log((f + epsilon) / (q + epsilon)))
    return kl_divergence


result = minimize(kullback_leibler_divergence, [1.5, 100], args=(q, t), bounds=[(1e-10, None), (1e-10, None)])


# Calculate the fitted Weibull distribution with the parameters
alpha_est, beta_est = result.x
fitted_weibull = weibull_distribution(t, alpha_est, beta_est)
fitted_weibull /= np.sum(fitted_weibull)
fitted_weibull *= 0.98
```

After turning the histogram into the probability distribution q and initial Weibull values of $\alpha = 1.5$ and $\beta = 100$, the result of minimizing Kullback-Leibler divergence, gives us $\alpha = 2.87$ and $\beta = 215$ which is slightly different from our previous results.