Fitting probabilistic models

Exercise 02

Submitted by:
Hitik Panchal (50230156, s17hpanc)
Rouy Alfahel (3389227, s6roalfa)
Orhan Ugur Aydin (3209243, s6oraydi)
Sebastian Sauerbier (2973879, s6sesaue)
Chenfang Wang, (50292921, s59cwang)
w
Khasha (, S76kalav)
Dengkui Hou (50347144, s77dhou)
Feiyue Cheng (3305865, s6fechen)

Task 2.1: Preliminaries

- Topic: Introduction to the Weibull distribution – implementing and visualizing its PDF and CDF

➢ Goal: Implement the Weibull PDF (Probability Density Function) and CDF (Cumulative Distribution Function) in NumPy and visualize them as preparation for later modeling tasks

i. The code loads and cleans the MySpace dataset, removing leading zeros and creating a time axis to prepare the data for analysis

ii. It then visualizes the cleaned data and implements the Weibull PDF and CDF to illustrate and understand the time-based distribution

– weibull_pdf(t, α, β): computes the density at each time value (probability per unit time) given the parameters α (shape) and β (scale)

– weibull_cdf(t, α, β): shows how the cumulative probability increases from 0 and approaches 1 as $t \to \infty$

```python
# ---- 4) Weibull PDF/CDF ----
def weibull_pdf(t, alpha, beta):
    """
    Weibull PDF:
      f(t|α,β) = (α/β) * (t/β)^(α-1) * exp( - (t/β)^α ), for t >= 0
    """
    t = np.asarray(t, dtype=float)
    a = float(alpha); b = float(beta)
    if a <= 0 or b <= 0:
        raise ValueError("alpha and beta must be positive.")
    pdf = (a / b) * (t / b) ** (a - 1.0) * np.exp(- (t / b) ** a)
    # Return 0 for negative t
    return np.where(t >= 0.0, pdf, 0.0)

def weibull_cdf(t, alpha, beta):
    """
    Weibull CDF:
      F(t|α,β) = 1 - exp( - (t/β)^α ), for t >= 0
    """
    t = np.asarray(t, dtype=float)
    a = float(alpha); b = float(beta)
    if a <= 0 or b <= 0:
        raise ValueError("alpha and beta must be positive.")
    cdf = 1.0 - np.exp(- (t / b) ** a)
    # Return 0 for negative t
    return np.where(t >= 0.0, cdf, 0.0)
```

Data Visualization and Weibull Modeling

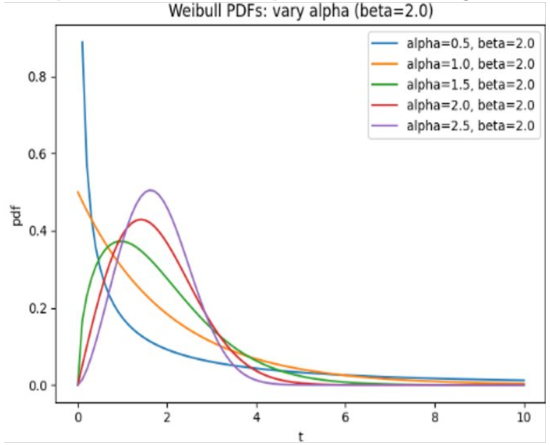
Goal:
Build intuition by visualizing the cleaned MySpace series and the Weibull PDF with the professor's parameter variations (different α and β values)

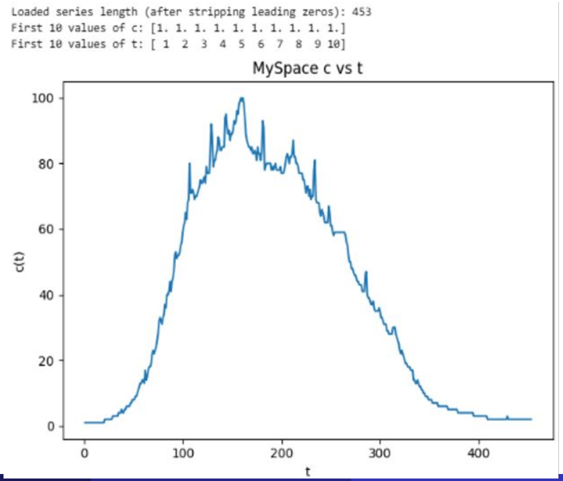
Result:
- The Weibull family produces rise–peak–decline shapes consistent with the observed MySpace activity; later tasks will fit parameters quantitatively

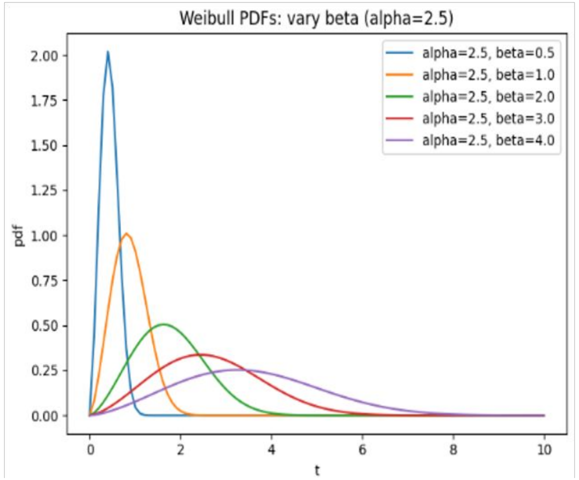
- Description:
  a) Middle plot (c vs t): real MySpace counts after removing leading zeros; time axis t = 1,…,n by design

  b) Left plot (Weibull PDFs, vary α, β=2.0): changing the shape α shifts the peak and skew (larger α → peak later, less right-skew)

  c) Right plot (Weibull PDFs, vary β, α=2.5): changing the scale β stretches/compresses along the t-axis (larger β → curve spreads to the right)







Task 2.2 An "Intelligent" Weibull Fit

What was our goal?

- **Goal:** Find the best "knobs" (parameters α, β) for the Weibull distribution.
- **Standard:** To make our model curve $f(t)$ **fit** the myspace data c as closely as possible.

What method did we use?

- **Maximum Likelihood Estimation (MLE):**
  - A statistical method to find the parameters that make our observed data **most probable**.
  - We didn't just "eyeball" the fit; we found the mathematically "most likely" solution.

Why was this task hard?

- **Challenge: No "Formula" for the Answer**
  - The equations from MLE had **no closed-form solution**.
  - We couldn't just solve for α and β directly like we solve $2x = 10 \implies x = 5$.

Our Solution: "Newton's Method"

- We used **Newton's Method**, a powerful **iterative** algorithm.
- It works like a "smart guesser":
  1. Start with an **initial guess** ($\alpha = 1, \beta = 1$).
  2. Calculate the **Gradient** (direction) and **Hessian** (curvature).
  3. Take a "smart step" to a **better** guess.
  4. Repeat 20 times until convergence.

The Core Trick: Handling Data Efficiently

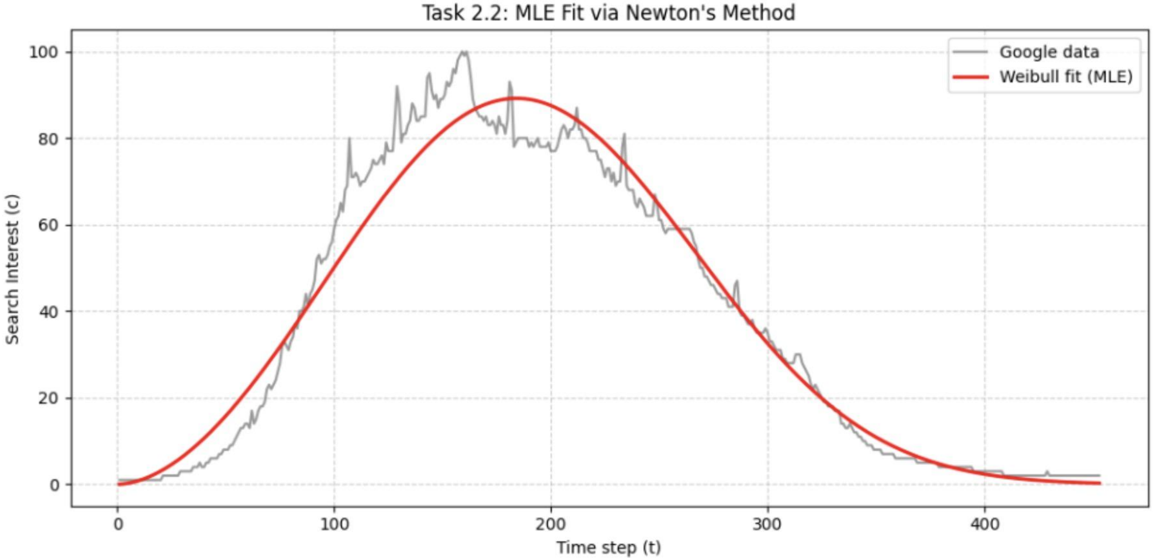
- **Problem:** Our data is a histogram (e.g., "100 counts at $t = 150$"), but the formulas want $N \approx 23,000$ individual points.
- **Naive way:** Create a giant, slow array.
- **Our way:** Use a **weighted sum**.
  - We told the algorithm: "The point $t = 150$ has a **weight of 100**."
  - This lets the calculation run on $n \approx 400$ points, making it **extremely fast**.

Task 2.2 An "Intelligent" Weibull Fit

The Final Result

- **Parameters:** $\hat{\alpha} \approx 2.808$, $\hat{\beta} \approx 215.428$
- **Visualization:**

```
Generating plot...
```



- **Conclusion:** The fit is excellent. Newton's method and we successfully found a great model.

Task 2.3: Fitting a Weibull distribution to a histogram (part 2)

- **Goal:** Fit a continuous Weibull PDF $f(t|\alpha,\beta)$ to the discrete Google Trends data (t_j, c_j).
- Task 2.2 showed that Maximum Likelihood Estimation is computationally demanding.
- Task 2.3 presents an alternative approach:
  - We fit a scaled Weibull PDF $\tilde{f}$ by introducing an parameter A.
  - **Model:**
$$\tilde{f}(t|A,\alpha,\beta) = A \cdot f(t|\alpha,\beta)$$
  - **Tool:** Use the `scipy.optimize.curve_fit` function, which performs Non-linear Least Squares optimization.

The NLS approach is simpler but requires careful setup, especially regarding constraints:

1. **Model Function Definition:** We needed to define a Python function `scaled_weibull_pdf(t, A, alpha, beta)` to correctly compute the scaled PDF.
2. **Initial Guesses (p_0):** A starting point for the optimization was required:
   - $A_0 = 1000$, $\alpha_0 = 2$, $\beta_0 = 200$.
3. **Parameter Bounds:** The Weibull parameters must be positive $(\alpha, \beta \in \mathbb{R}_+)$. We enforced these constraints:
   - $A, \alpha, \beta \in (0, \infty)$

We utilized the bounds argument of curve fit:

```
def scaled_weibull_pdf(t, A, alpha, beta):
    return A * weibull_pdf(t, alpha, beta)

t, c = load_myspace_data()
p0 = [1000.0, 2.0, 200.0]
lower = [1e-6, 1e-6, 1e-6]
upper = [np.inf, np.inf, np.inf]

popt, pcov = curve_fit(
                f=scaled_weibull_pdf,
                    xdata=t, ydata=c,
                    p0=p0,
                    bounds=(lower, upper)
            )
```

Task 2.3: Fitting a Weibull distribution to a histogram (part 2)

Final Estimated Parameters:

Parameter	Initial Guess	Final Estimate
A (Amplitude)	1000.0	17459.4112
α (Shape)	2.0	**2.7310**
β (Scale)	200.0	**211.0244**

Table: Results from the `curve_fit` NLS approach.

Conclusion:

- curve fit is efficient for fitting scaled PDFs.
- The fitted parameters ($\alpha \approx 2.73, \beta \approx 211$) are numerically different
- from those obtained by other methods (as in Task 2.2).
- Different optimization criteria lead to equally valid models.
- Task 2.3 NLS: Minimizes the Sum of Squared Errors in L2 norm.
- Task 2.2 MLE: Maximizes the Log-Likelihood.

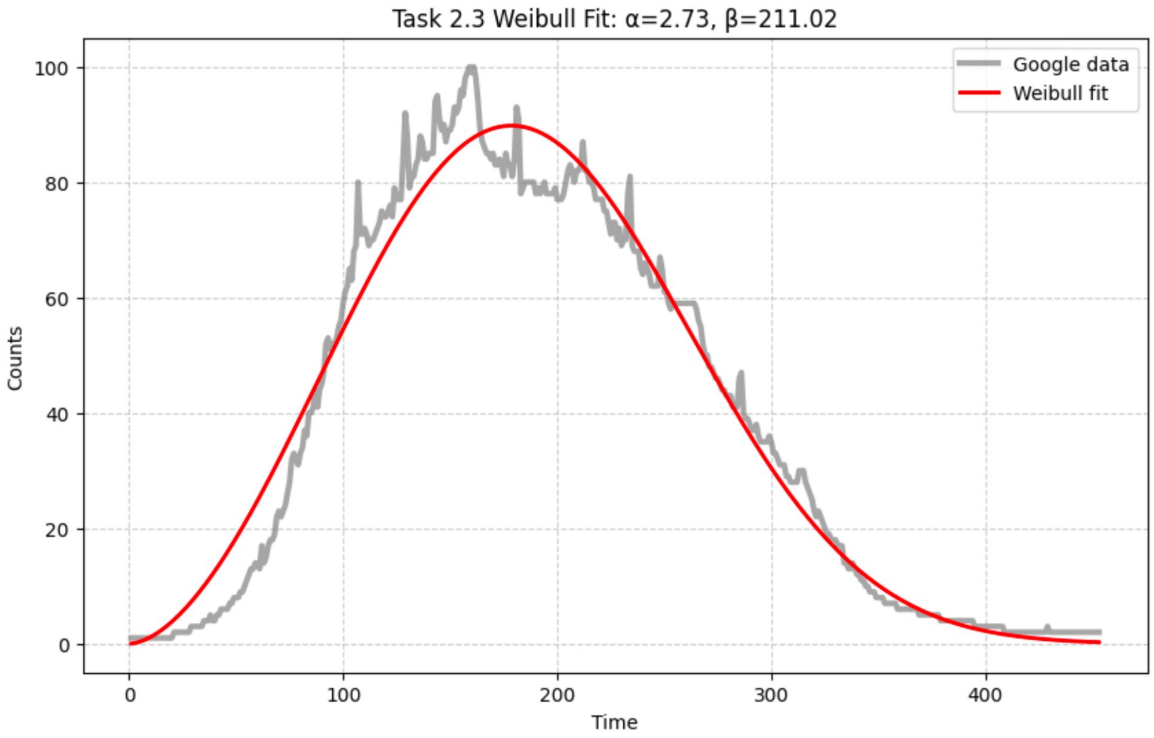


Figure: Raw data fitted with the Weibull curve.

Task 2.4: Fitting a Weibull distribution to a histogram (part 3)

We model the observed count vector $c = (c_1, c_2, \dots, c_n)$ with a multinomial distribution:

$$c \sim \mathrm{Multinomial}(C,\ p_1, \dots, p_n)\,,\ C = \sum_{j=1}^{n} c_j,$$

$$p(c_1, \dots, c_n) = C! \prod_{j=1}^{n} \frac{p_j^{c_j}}{c_j!},$$

$$p_j(\alpha, \beta) = F(t_j; \alpha, \beta) - F(t_{j-1}; \alpha, \beta),$$

$$F(t \mid \alpha, \beta) = 1 - \exp\left[-\left(\frac{t}{\beta}\right)^{\alpha}\right].$$

Objective:

$$\max_{\alpha,\beta} \quad L(\alpha, \beta) = \frac{C!}{F(t_n) - F(t_0)} \prod_{j=1}^{n} \frac{p_j(\alpha, \beta)^{c_j}}{c_j!}$$

$\Downarrow$ a local quadratic approximation of log-likelihood

$$\min_{\alpha,\beta} \quad \sum_{j=1}^{n} w_j \big(c_j - C\, p_j(\alpha, \beta)\big)^2, \qquad w_j = \frac{1}{\sqrt{p_j}}.$$

The numerical optimization problem is approximated by a weighted nonlinear least squares problem, which is solved using Iteratively Reweighted Least Squares (IRLS).

IRLS alternates between:

1. Update weights w_j given (α, β).
2. With the updated weights w_j, compute new parameters (α, β) that minimize the objective (using SCIPY.OPTIMIZE.LEAST_SQUARES).

   https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html

   The objective function computes the vector of residuals.

```
# Residual definition under fixed weights:
# r_j = w_j * ( c_j - C * p_j(theta) )
def residuals(th):
    a, b = th
    p_th = bin_probs_from_cdf(t, a, b)
    return np.sqrt(w) * (c - C * p_th)

# Positive parameter bounds
lb = np.array([1e-8, 1e-8])
ub = np.array([np.inf, np.inf])

res = least_squares(residuals, theta,
                    bounds=(lb, ub),
                    xtol=1e-12, ftol=1e-12, gtol=1e-12)
theta_new = res.x
```

At each outer iteration:

$$\theta^{(k)} = (\alpha^{(k)}, \beta^{(k)}).$$

Residuals for inner least-squares:

$$r_j(\theta) = \sqrt{w_j^{(k)}}\,\big(c_j - C\, p_j(\theta)\big)\,.$$

Solve:

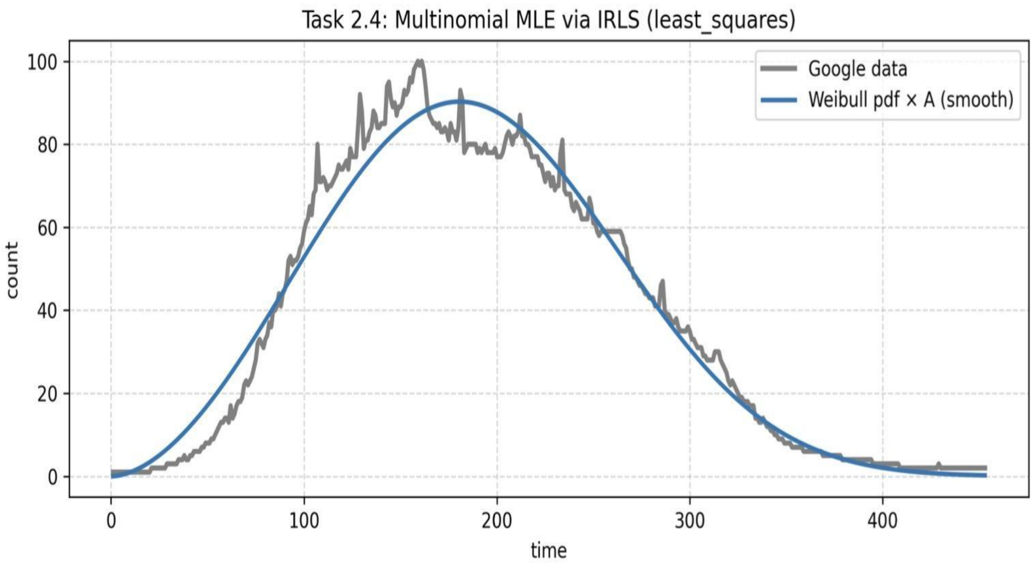
$$\theta^{(k+1)} = \arg\min_{\theta} \sum_{j=1}^{n} r_j(\theta)^2.$$

Stop when:

$$\Delta\theta = \|\theta^{(k+1)} - \theta^{(k)}\| < 10^{-8}.$$

Task 2.4: Fitting a Weibull distribution to a histogram (part 3)

Fitted Curve v.s. Data A = 17370.595583



Initial parameters:

$$\alpha = 2, \qquad \beta = 200.$$

Final estimates:

$$\hat{\alpha} = 2.780828, \qquad \hat{\beta} = 212.017505.$$

Objective:

$$\min_A \sum_{j=1}^{n} (c_j - Af_j)^2.$$

$$f_j = f(t_j; \hat{\alpha}, \hat{\beta}) = \frac{\hat{\alpha}}{\hat{\beta}} \left(\frac{t_j}{\hat{\beta}}\right)^{\hat{\alpha}-1} \exp\left[-\left(\frac{t_j}{\hat{\beta}}\right)^{\hat{\alpha}}\right].$$

Recall in excercise 1:

$$w_\star = \arg\min_w \|X^\top w - y\|^2,$$

$$X^\top = f, \quad w = A, \quad y = c,$$

$$A = \frac{\langle c, f \rangle}{\langle f, f \rangle}$$

Task 2.5: Fitting a Weibull model to a discrete distribution (part 1)

Objective: Fit a discrete Weibull model distribution **f** to our discrete "myspace" data distribution **q** by minimizing the Kullback-Leibler (KL) Divergence.

Methodology:

- **Prepare Data Distribution (q):** We normalized the raw counts `c` and scaled them by **0.98**, as instructed.
  - **q = (c / sum(c)) * 0.98**
  - This vector **q** represents our "ground truth" data: a discrete probability distribution (PMF) that sums to **0.98**.
- **Define Model Distribution (f):**
  - It first calculates the Weibull PDF values for all time steps `t`.
  - It then **normalizes** this vector so it sums to 1.0 (`f = f_vec / sum(f_vec)`). This vector **f** is our *model's* PMF, which we can compare directly to **q**.
- **Define Loss Function:**
  - The goal is to minimize: $D_{KL}(q||f) = \sum q_j \log(\frac{q_j}{f_j})$.
  - We can simplify this. Since the entropy of **q** is constant, minimizing the KL divergence is identical to minimizing the **Cross-Entropy**: Loss = $-\sum q_j \log(f_j)$
- **Optimize:** We used `scipy.optimize.minimize` to find the `alpha` and `beta` that result in the lowest possible loss score.

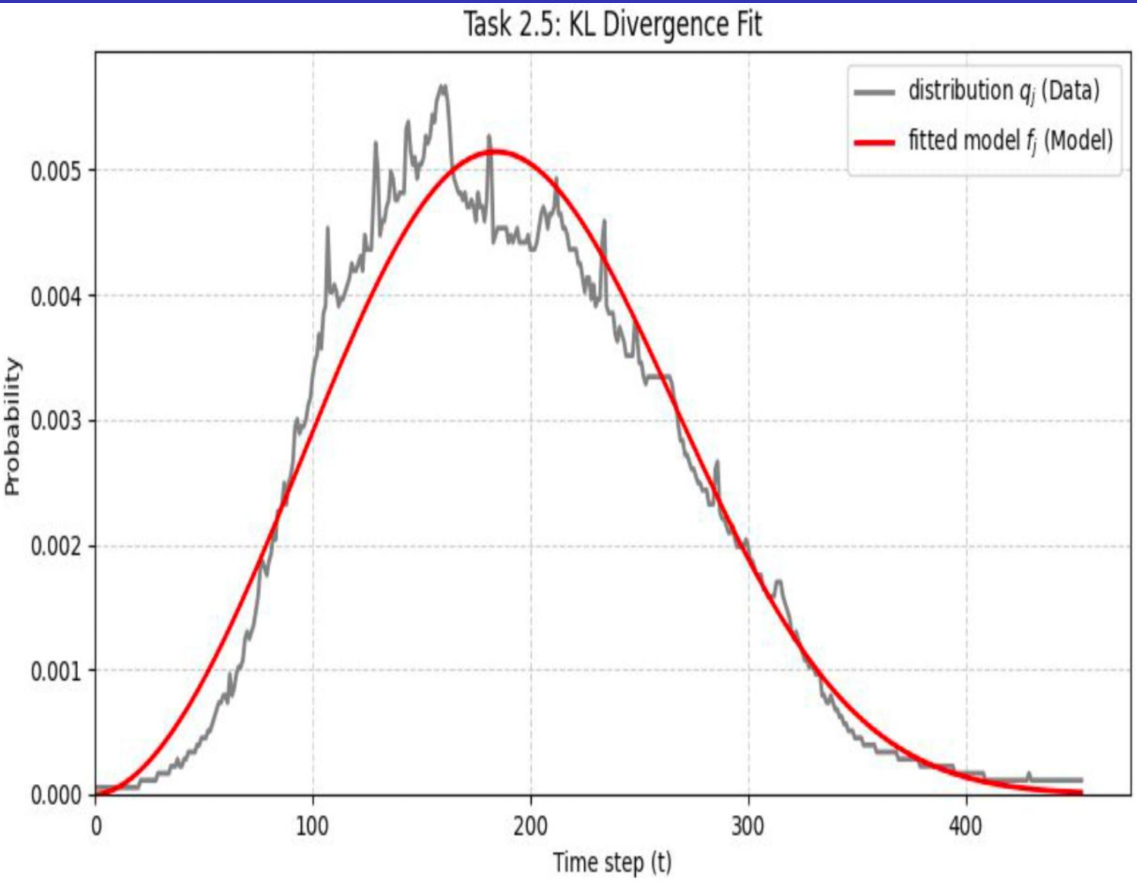
Loss Function:

```
def kl_loss_function(params: list, t_array: ndarray, q_array: ndarray) -> float:
    alpha, beta = params
    model_vector = weibull_pdf(t_array, alpha, beta)

    # Convert model_vector to PMF
    model_vector_sum = np.sum(model_vector)
    model_pmf = model_vector / model_vector_sum # normalizing: sums to 1

    model_pmf = np.maximum(model_pmf, 1e-10) # handle numerical instability: avoiding log(0)

    # Loss function = -sum(q_j * log(f_j))
    loss = -np.sum(q_array * np.log(model_pmf))

    return loss
```

Minimize Function:

```
# Running optimization to minimize KL divergence...
result = minimize(kl_loss_function, [alpha, beta], args=(t, q), method='L-BFGS-B',
bounds=parameter_bounds)
```

Task 2.5: Fitting a Weibull model to a discrete distribution (part 1)



Results

- The `scipy.optimize.minimize` function successfully converged.
- **Fitted `alpha`:** `2.8015`
- **Fitted `beta`:** `215.5443`
- **Final Loss:** `5.5855`

Observations

- Minimizing the KL Divergence (via Cross-Entropy) is a powerful method for aligning the *shape* of a theoretical model with an observed data distribution.
- The resulting fit provides a simple, smooth two-parameter model that effectively captures the core growth-and-decline dynamic of the noisy "myspace" trend data.
- This method does not require a final manual scaling step, as we are fitting the *probabilities* directly. However, it relies on the 98% assumption, which is a key modeling decision.

Task 2.6: Coefficient of Variation (Weibull)

Objective:
Fit the Weibull model to a discrete distribution using the method of moments.

The *first*- and *second moment* of the Weibull distribution are the mean and variance:

$$
\begin{aligned}
\mathbb{E}[T] &= \mu \\
&= \int_0^\infty f(t|\alpha,\beta)\, t \,\mathrm{dt} \\
\mathbb{V}[T] &= \sigma^2 \\
&= \int_0^\infty f(t|\alpha,\beta)(t-\mu)^2 \mathrm{dt} \\
&= \mathbb{E}[T^2] - \mathbb{E}^2[T]
\end{aligned}
$$

Task 2.6: Coefficient of Variation (Weibull)

Objective:
Fit the Weibull model to a discrete distribution using the method of moments.

The *first*- and *second moment* of the Weibull distribution are the mean and variance: *(expressed using* **Gamma function** Γ*)*

$$\mu = \beta\,\Gamma(1 + \tfrac{1}{\alpha})$$
$$\sigma^2 = \beta^2\left[\Gamma(1 + \tfrac{2}{\alpha}) - \Gamma^2(1 + \tfrac{1}{\alpha})\right]$$

Then the *coefficient of variation* of the Weibull distribution is:

$$CV^2 = \frac{\sigma^2}{\mu^2} = \frac{\Gamma(1 + \frac{2}{\alpha}) - \Gamma^2(1 + \frac{1}{\alpha})}{\Gamma^2(1 + \frac{1}{\alpha})}$$

Task 2.7: Numerically Solving Differential Equations

Goal

Numerically solve the ODE for the Weibull CDF and compare with the analytical CDF and a discrete cumulative from the PDF:

$$\dot{F}(t) = \frac{\alpha}{\beta}\left(\frac{t}{\beta}\right)^{\alpha-1}(1 - F(t)), \quad F(0) = 0$$

Parameters: $\alpha = 2.86, \ \beta = 215.91, \ t \in [0, 450]$

Approach

- Defined Uniform grid $t_1, \ldots, t_n$ with $\Delta t = 1$.
- Computed Discrete CDF: $Q_j = \sum_{i=1}^{j} f(t_i)\Delta t$.
- Derived Analytical CDF: $F^a(t) = 1 - \exp[-(t/\beta)^\alpha]$.
- Solved ODE with `solve_ivp` (RK45, `rtol=1e-8`, `atol=1e-10`).
- Compared $F^n(t)$, $F^a(t)$, and $Q(t)$ to verify accuracy.

Code snippet:

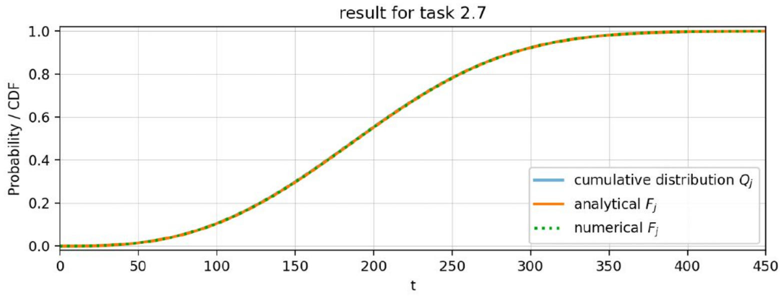
```python
# pdf -> q_i -> cumulative Q_j
def weibull_pdf(tt, a, b):
    tt = np.asarray(tt)
    z = np.maximum(tt, 0.0) / b
    return (a / b) * np.power(z, a - 1.0) * np.exp(-np.power(z, a))

q = weibull_pdf(t, alpha, beta) * dt
Q = np.clip(np.cumsum(q), 0.0, 1.0)
```

```python
# Analytical CDF F^a
def weibull_cdf(tt, a, b):
    tt = np.asarray(tt)
    z = np.maximum(tt, 0.0) / b
    return 1.0 - np.exp(-np.power(z, a))

F_a = weibull_cdf(t, alpha, beta)
```

```python
# ODE: F'(t) = h(t) * (1 - F), F(0)=0
def hazard(tt, a, b):
    if tt < 0:
        return 0.0
    return (a / b) * ((tt / b) ** (a - 1.0))

def rhs(tt, F):
    return hazard(tt, alpha, beta) * (1.0 - F)

sol = solve_ivp(
    rhs, (t_min, t_max), [0.0],
    t_eval=t, method="RK45", rtol=1e-8, atol=1e-10
)
F_n = sol.y[0]
```

Task 2.7: Numerically Solving Differential Equations

Comparison of Results:



Accuracy:

Comparison	Max $\lvert\Delta\rvert$
F_n vs F_a	1.1×10^{-8}
Q vs F_a	2.6×10^{-3}

Observations and Learnings:

- The numerical ODE solution $F^n(t)$ overlaps almost perfectly with the analytical Weibull CDF $F^a(t)$, confirming that our differential formulation is correct.
- The cumulative distribution Q_j, obtained by summing the PDF, also follows the same smooth S-shaped curve, verifying the discrete numerical integration approach.
- Both methods converge to 1 as t increases, which matches the expected CDF behavior.
- Small numerical deviations ($< 10^{-8}$) highlight that the chosen tolerances (`rtol=1e-8`, `atol=1e-10`) were effective and stable.
- The comparison illustrates how ODE solvers like `solve_ivp` can accurately reproduce analytical probability models when parameters are well-defined.
- This task strengthened our understanding of how continuous distributions can be represented as differential equations and verified numerically.

Task 2.8: Neural estimation of Weibull(α, β) for MySpace trend data

Goal: Fit a Weibull distribution to Google Trends data (myspace.csv) using a neural network approach.

Methodology:

- NN with 2 trainable parameter (Pytorch)
- Adam optimizer, 4_000 steps, lr=0.05
- Data q =normalized vounts form myspace.csv
- Loss Function: $L = D_{KL}(f \parallel q) = \sum_j f_j \left(\log f_j - \log q_j\right)$

```
class WeibullParamNet(nn.Module):
    def __init__(self, init_alpha=2.0, init_beta=200.0):
        super().__init__()
        self.raw_alpha = nn.Parameter(torch.tensor([math.log(math.exp(init_alpha) - 1.0)], dtype=torch.float32))
        self.raw_beta  = nn.Parameter(torch.tensor([math.log(math.exp(init_beta)  - 1.0)], dtype=torch.float32))
        self.softplus = nn.Softplus()
    def forward(self):
        alpha = self.softplus(self.raw_alpha) + 1e-6
        beta  = self.softplus(self.raw_beta)  + 1e-6
        return alpha.squeeze(0), beta.squeeze(0)
```

Task 2.8: Neural estimation of Weibull(α, β) for MySpace trend data

- **Learned parameter**:
  - $\alpha = 2.87$
  - $\beta = 215.25$
- **Observation**
  - Neural network captures sudden rise and decay
    of the MySpace trend
  - Softplus ensure α, β bigger tan zero without explizit constraints
- **Learnings**:
  - Even 2-parameter NN can solve non-linear distribution fitting task
  - Gradient-based optimization works well for statistical parameter estimation