exercise 2

fitting probabilistic models

solutions due

until **November 18, 2025** at **23:59** via **ecampus**

general remarks

In our very first lecture, we claimed that *machine learning is an art or a craft rather than a science*. The tasks in this exercise will illuminate one of the manifestations of the "craftfulness" of machine learning.

In particular, they will illustrate that, if we are given a fixed data set and a fixed model class and then run different model-fitting algorithms, it is likely that we end up with different fitted models.

This does not mean that some of these algorithms are "bad" or "useless". It really only means that model fitting is generally an under-specified problem and therefore tends to have several possible and plausible solutions . . .

Now, before we get started, there are two more things:

1) Many of the optimization- or model fitting methods we have studied so far are used by many of the functions provided in by `numpy` or `scipy`. The tasks in this exercise thus provide an opportunity for learing about wrappers around the techniques we already know. This may come in handy throughout the course. The downside is that many of the functions provided by `numpy` or `scipy` come with rather cumbersome APIs which take getting used to . . .

2) All the tasks in this exercise will be dealing with data provided in the file `myspace.csv`. It contains a **Google Trends** time series which shows how global interest in the query term "*myspace*" first grew and then declined over a period of several years. This data is old! We collected it in 2013. Since then, Google has unfortunately reduced the granularity of the search frequency data they provide to the public. Therefore, please do not collect your own data from Google Trends but work with the data in `myspace.csv`.

   We want to make sure that every team of students is working with the same data set! This is because, if you follow the given instructions, then the results we are asking you to produce are deterministic. In other words, it will not make sense to report "hallucinated" results. **If your results deviate significantly from your instructor's results, they are incorrect and will receive zero points.**

task 2.1 [5 points]

preliminaries

This task asks you to code basic functionalities which you will need in all subsequent tasks. Please work with `numpy` and `matplotlib`.

task 2.1.1 [2 points] data pre-processing

The file

```
myspace.csv
```

contains (historic) Google Trends data (from 2013) showing how worldwide interest in the query term *"myspace"* evolved over time.

Read the data in the second column of this file, remove all leading zeros, and store the n remaining values in a 1D array

$$\boldsymbol{c} = \big[c_1, c_2, \ldots, c_n\big] \ .$$

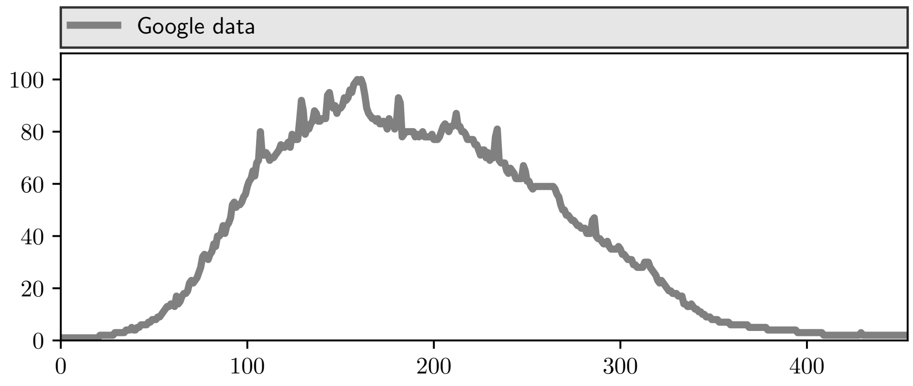
Furthermore, create a 1D array of n time steps

$$\boldsymbol{t} = \big[1, 2, 3, \ldots, n\big] \ .$$

Yes, the entries of t are deliberately chosen to range from 1 up to n !

Below, you will need to run these two preparatory steps a lot, so you may just as well wrap them in a function `load_myspace_data` ...

Next, implement a function that allows you to plot the data in $\boldsymbol{c}$ against the data in $\boldsymbol{t}$. While your plots do not have to be as fancy as the following, your results should look something like this:



task 2.1.2 [3 points] Weibull PDFs and CDFs

Note that we are dealing with histogram data, namely with a distribution of discrete search *counts* c_j over discrete *times* t_j. Also note that we can always think of discrete count data as resulting from sampling some *latent* (i.e. unknown to us), continuous probability density.

In our lectures, we frequently work with Gaussian distributions to model latent densities because they are exceptionally easy to handle. However, ease of use should never be our primary objective when modeling real life data. Instead, models must be plausible and capture crucial characteristics of the data we are working with. So, consider this:

The above figure clearly shows that searches for *"myspace"* were not Gaussian distributed. Over time, interest first grew quickly, reached some peak activity, and then steadily declined. In other words, while a Gaussian is symmetric, the distribution in the figure is positively skewed (its right tail is longer than its left tail). Modeling the temporal evolution of *"myspace"* searches in terms of a Gaussian would thus be inappropriate.

One more thing: The figure also shows that Google does not disclose true search volumes. Rather, Goolge Trends data is always scaled such that the maximum search activity corresponds to a value of 100. From the point of view of probabilistic modeling, this is unfortunate but provides us with learning opportunities . . .

A much better model for our data is the Weibull distribution. It is defined for $t \in [0, \infty)$ and its *probability density function* and *cumulative density function* are given by

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

and

$$F(t \mid \alpha, \beta) = 1 - e^{-\left(\frac{t}{\beta}\right)^{\alpha}} \tag{2}$$

respectively. The two parameters $\alpha, \beta \in \mathbb{R}_+$ determine the *shape* and *scale* of the distribution.

Given these definitions, code two properly parameterized `numpy` functions `weibull_pdf` and `weibull_cdf` which implement (1) and (2).

Note: Do *not* work with any of the Weibull implementations provided in `scipy.stats`. They are lobotomized versions of the functions in (1) and (2) and nobody knows why the `scipy` developers opted to ignore the real Weibull distribution . . .

If you feel uneasy about the notions of shape and scale of a probability distribution, simply `import matplotlib.pyplot as plt` and run

```
ts = np.linspace(0, 10, 100)

for alpha in [0.5, 1.0, 1.5, 2.0, 2.5]:
    plt.plot(ts, weibull_pdf(ts, alpha, 2.0), '-')
plt.show()

for beta in [0.5, 1, 2, 3, 4]:
    plt.plot(ts, weibull_pdf(ts, 2.5, beta), '-')
plt.show()
```

background info: why the Weibull ?

To see that the Weibull distribution implicitly encodes a growth-and-decline dynamic, we note that

$$\begin{aligned} f(t) &= \tfrac{\alpha}{\beta}\big(\tfrac{t}{\beta}\big)^{\alpha-1}\big(1 - F(t)\big) \\ &= \tfrac{\alpha}{\beta}\big(\tfrac{t}{\beta}\big)^{\alpha-1} - \tfrac{\alpha}{\beta}\big(\tfrac{t}{\beta}\big)^{\alpha-1} F(t) \ . \end{aligned}$$

Hence, the Weibull sneakily involves a difference between a growth- and a decline term which are both polynomial in t. Moreover, since the decline term also depends on $F(t)$ which itself grows monotonously, we find that the longer a process is running whose dynamics are characterized by a Weibull distribution, the quicker the decline.

Below, you will see that the Weibull is not necessarily the best model for how public interest in social media sites grows and declines over time. The real contender in this arena is the generalized Gamma distribution [1,2]. Alas, since this more powerful model is slightly too cumbersome for these exercises, we focus on the Weibull for simplicity . . .

[1] C. Bauckhage and K. Kersting, “Collective Attention on The Web”, Foundations and Trends in Web Science, **5**(1–2), 2016.
[2] C. Bauckhage, K. Kersting, and F. Hadiji, “Parameterizing the Distance Distribution of Undirected Networks”, Proc. UAI, 2015.

task 2.2 [10 points]

fitting a Weibull distribution to a histogram (part 1)

Given what you did task 2.1, your next task is to fit a Weibull probability density $f(\, t \mid \alpha, \beta)$ to our (t_j, c_j) data such that you get

$$c_j \sim f(\, t_j \mid \alpha, \beta) \tag{3}$$

up to a constant scaling factor.

One would think that `scipy.stats` or `scipy.optimize` would provide ready-made functions for this task. Alas, neither one ships with tools that fit *continuous* densities to *discrete* and potentially *truncated* histograms. So, you have to implement these tools yourself . . .

We already know about maximum likelihood estimation of the parameters of Gaussian distributions. Alas, maximum likelihood estimation of the parameters of a Weibull distribution is much more involved. To see why, we next “naïvely” transfer what we learned w.r.t. the Gaussian setting to our current Weibull setting . . .

Given a data sample $D = \{x_i\}_{i=1}^N$, the log-likelihood for the parameters of the Weibull distribution is

$$\mathcal{L}(\alpha, \beta \mid D) = N\big(\log \alpha - \alpha \log \beta\big) + (\alpha - 1) \sum_i \log x_i - \sum_i \left(\frac{x_i}{\beta}\right)^\alpha .$$

Now, looking at our trusted recipe of estimating optimal $\hat{\alpha}$ and $\hat{\beta}$ by solving

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = 0$$

we quickly realize that this system of coupled partial differential equations does not admit a closed form solution.

However, since we also already know about Newton’s method for parameter optimization, we may guess a solution and refine it via the following iterative updates

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \leftarrow \begin{bmatrix} \alpha \\ \beta \end{bmatrix} - \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial \alpha^2} & \frac{\partial^2 \mathcal{L}}{\partial \alpha \partial \beta} \\ \frac{\partial^2 \mathcal{L}}{\partial \alpha \partial \beta} & \frac{\partial^2 \mathcal{L}}{\partial \beta^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \alpha} \\ \frac{\partial \mathcal{L}}{\partial \beta} \end{bmatrix} . \tag{4}$$

This already looks quite daunting, but its gets worse . . .

The entries of the gradient vector and the Hessian matrix for the Newton updates amount to

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \alpha} &= \frac{N}{\alpha} - N \log \beta + \sum_i \log x_i - \sum_i \left(\frac{x_i}{\beta}\right)^\alpha \log \frac{x_i}{\beta} \\
\frac{\partial \mathcal{L}}{\partial \beta} &= \frac{\alpha}{\beta} \left(\sum_i \left(\frac{x_i}{\beta}\right)^\alpha - N \right) \\
\frac{\partial^2 \mathcal{L}}{\partial \alpha^2} &= -\frac{N}{\alpha^2} - \sum_i \left(\frac{x_i}{\beta}\right)^\alpha \left(\log \frac{x_i}{\beta}\right)^2 \\
\frac{\partial^2 \mathcal{L}}{\partial \beta^2} &= \frac{\alpha}{\beta^2} \left(N - (\alpha + 1) \sum_i \left(\frac{x_i}{\beta}\right)^\alpha \right) \\
\frac{\partial^2 \mathcal{L}}{\partial \alpha \partial \beta} &= \frac{1}{\beta} \sum_i \left(\frac{x_i}{\beta}\right)^\alpha + \frac{\alpha}{\beta} \sum_i \left(\frac{x_i}{\beta}\right)^\alpha \log \frac{x_i}{\beta} - \frac{N}{\beta} \,.
\end{aligned}
$$

If you want to, use `sympy` or ask an AI to verify that all these derivatives are correct.

here is your task:

Implement `numpy` code that realizes the Newton updates in (4) and then use your code to fit a Weibull to the given Google Trends data.

⚠

Note: There is a subtlety here which your implementation should address!

Recall that we "naïvely" transferred our knowledge from the Gaussian setting to the Weibull time series setting.

In this latter setting, you are given histogram data (t_j, c_j) where c_j counts the number of searches at time t_j. However, the MLE procedure outlined above assumes that you are given i.i.d. observations x_i. That is, it assumes as input a number of c_1 observations of the value t_1, a number of c_2 observations observation of the value t_2, etc. In other words, $x_1 = t_1, x_2 = t_1, \ldots, x_{c_1} = t_1, x_{c_1+1} = t_2, x_{c_1+2} = t_2, \ldots, x_{c_1+c_2} = t_2, \ldots$

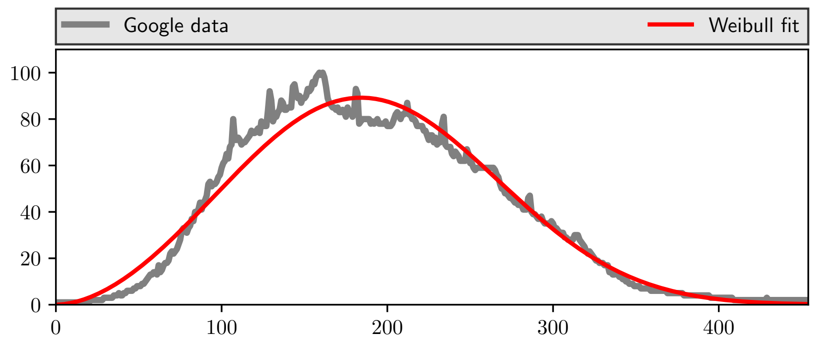
It thus seems that you would have to turn the given histogram into a (large) set of numbers for the procedure to work. But there is a much more elegant solution! Can you see and implement it?

Be ambitious! Analyze why turning the histogram into a set of numbers is unnecessarily tedious and find and then implement a faster approach.

If you initialize $\alpha = 1$ and $\beta = 1$ and run your estimation procedure for about 20 iterations, then which values do you obtain for α and β ?

Please report your estimates in your submitted solutions.

The following figure provides a reference for what you should get when you plot the Google data together with an *appropriately scaled version* of your fitted Weibull distribution:



Of course, this is a less than perfect explanation of the data we are dealing with but a much better one than a Gaussian could provide . . .

task 2.3 [5 points]

fitting a Weibull distribution to a histogram (part 2)

It is strange that neither `scipy.stats` nor `scipy.optimize` provide functions for fitting densities to histograms but we can hack around this!

Again, let $\boldsymbol{c} = [c_1, \ldots, c_n]$, $\boldsymbol{t} = [1, \ldots, n]$, and $f(t \mid \alpha, \beta)$ be as in the tasks above. But now use the `scipy.optimize` function `curve_fit` to fit the following model

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

to the given data. Observe that this is but a scaled version of the Weibull pdf where parameter A represents an amplitude.

⚠

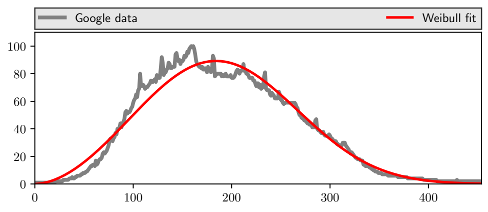
Note: When working with `curve_fit`, you need initial guesses for the parameters to be optimized. Good choices for our current setting are

$$\begin{aligned}\alpha &= 2\\ \beta &= 200\\ A &= 1000\ .\end{aligned}$$

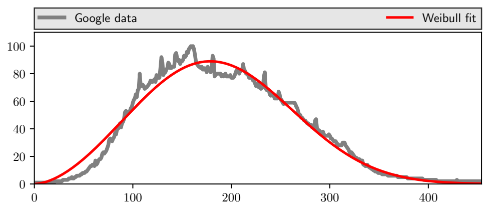
Also, and this is critical, the parameters α, β of a Weibull distribution must be positive and you have to tell `curve_fit` about this. To this end, provide appropriate bounds for the parameters when you call `curve_fit`.

If you adhere to all this, then which final estimates do you obtain for α and β ? Report them in your submission.

Note: It would be surprising if your result for this task was the same as in the previous task. For reference, here are plots of your instructor's results



typical result for task 2.2



typical result for task 2.3

task 2.4 [10 points]

fitting a Weibull distribution to a histogram (part 3)

We just saw two methods for fitting a Weibull model to count data. Both worked but led to slightly different results. Next, we look at yet another, more advanced method which will again yield slightly different results. The theory behind this method require us to consider the Weibull cumulative density function $F(\,t \mid \alpha, \beta)$ in (2) . . .

Above, you fitted a continuous density $f(t \mid \alpha, \beta)$ to a discrete series of frequency counts $c_1, \ldots, c_n$ grouped into n distinct intervals $(t_0, t_1]$, $(t_1, t_2]$, $\ldots$, $(t_{n-1}, t_n]$ where, in our setting, we have $t_0 = 0, t_1 = 1, t_2 = 2, \ldots$

To devise another, often quite robust algorithm for estimating optimal model parameters $\hat{\alpha}$ and $\hat{\beta}$, we note that the probability for observing a sample of counts can be modeled by a multinomial distribution

$$p(c_1, \ldots, c_n) = C! \prod_j \frac{p_j^{c_j}}{c_j!}$$

where

$$C = \sum_j c_j \ .$$

Now, since the cumulative density of any model distribution $f(\,t \mid \alpha, \beta)$ is

$$F(t) = F(t \mid \alpha, \beta) = \int_0^t f(\tau \mid \alpha, \beta) \; d\tau,$$

we may also think of the probabilities p_j in the multinomial representation of our histogram sample as

$$p_j(\alpha, \beta) = F(t_i) - F(t_{i-1})$$

such that $\sum_j p_j = F(t_n) - F(t_0)$. Accordingly, the likelihood for a discrete (and possibly truncated) series of counts $c_1, \ldots, c_n$ amounts to

$$L(\alpha, \beta) = \frac{C!}{F(t_n) - F(t_0)} \prod_j \frac{p_j(\alpha, \beta)^{c_j}}{c_j!}$$

and maximum likelihood estimates of the parameters $\boldsymbol{\theta} = (\alpha, \beta)$ could be determined by estimating the roots of $\nabla_{\boldsymbol{\theta}} \log L$. However, . . .

Again, this will not lead to closed form solutions but requires numerical optimization. To this end, "one can show that" we can work with an iterative and *weighted* least squares scheme where we consider

$$\sum_j w_j \Big(c_j - C p_j(\alpha, \beta) \Big)^2$$

which regresses the c_j onto their expectations $C p_j$ and requires to update the weights $w_j = \frac{1}{\sqrt{p_j}}$ in each iteration.

here is your task:

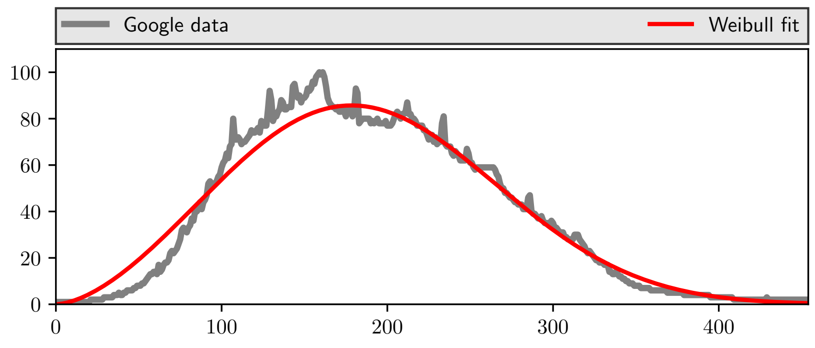
Use the `scipy.optimize` function `least_squares` to implement the idea of *multinomial maximum likelihood via weighted least squares*. and run your code to fit a Weibull to the Google data.

When Working with `least_squares`, you must again provide appropriate parameter bounds and guesses, say

$$\alpha = 2 \qquad \text{and} \qquad \beta = 200 \; .$$

If you use these, then which final estimates do you obtain for the sought after α and β ? As always, report your estimates in your submission.

To visually inspect your result, visualize your (properly scaled) fitted model. Your result should look similar to what your instructor got namely:



typical result for task 2.4

task 2.5 [10 points]

fitting a Weibull model to a discrete distribution (part 1)

Above, you fitted a continuous Weibull distribution $f(\, t \mid \alpha, \beta)$ to counts c_j which were gathered at discrete times t_j. The model fitting goal was

$$\forall\, j : c_j \sim f_j = f(\, t_j \mid \alpha, \beta)$$

up to a constant scaling factor. Now, consider this ...

We may normalize the given vector $\boldsymbol{c} = [c_1, \ldots, c_n]$ of counts into a vector $\boldsymbol{q} = [q_1, q_2, \ldots, q_n]$ whose entries are given by

$$q_j = \frac{c_j}{\sum_i c_i} \, .$$

Since our counts are non-negative $(c_j \geq 0)$ we thus obtain a vector $\boldsymbol{q}$ with

$$\begin{aligned} q_j &\geq 0 \\ \sum_j q_j &= 1 \, . \end{aligned}$$

Hence, we obtain a stochastic vector $\boldsymbol{q}$ or a *probability mass function* or *discrete distribution* $q(\, t_j)$ over discrete times $t_1, t_2, \ldots, t_n$.

By the same token, we may understand the above $f_j = f(\, t_j \mid \alpha, \beta)$ as the components of another stochastic vector $\boldsymbol{f} = [f_1, f_2, \ldots, f_n]$ which, in turn, represents a discrete Weibull distribution over discrete times $t_1, t_2, \ldots, t_n$.

Using these ideas, we may thus cast our model fitting problem as the problem of estimating Weibull parameters α, β such that $\boldsymbol{f}$ and $\boldsymbol{q}$ are well aligned. A common measure for the alignment of two discrete distributions is the Kullback-Leibler divergence

$$D_{KL}(\boldsymbol{f} \parallel \boldsymbol{q}) = \sum_j f_j \, \ln \frac{f_j}{q_j} \, .$$

Since it is minimal if $\boldsymbol{f} = \boldsymbol{q}$, we can consider the KL-divergence as a loss function for model fitting. However, ...

Note: In our particular setting, there is a catch. Our original data (t_j, c_j) represents a times series or a process that evolves over time. Alas, due to the way this data was collected, we have no way of knowing whether or not the process has already ended (there may still be people searching for *"myspace"* today). **We therefor need another modeling assumption** . . .

Our data suggest that interest in the search term *"myspace"* has more or less died out towards the end of the observation period. We therefore *assume* that the data we are given reflects 98% of the overall process.

here is your task:

Use the following to turn the given counts $\boldsymbol{c}$ into an "open-ended" discrete probability distribution $\boldsymbol{q}$

$$q_j = \frac{c_j}{\sum_i c_i} \cdot 0.98$$

and then fit $\boldsymbol{f}$ with entries $f_j(\alpha, \beta)$ by minimizing $D_{KL}(\boldsymbol{f} \parallel \boldsymbol{q})$.

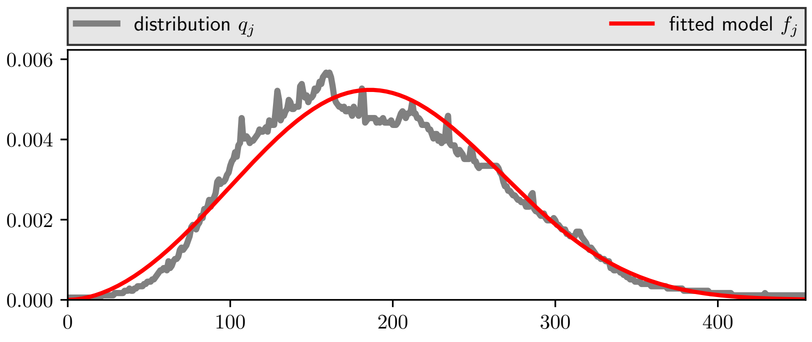
This is not a trivial task. However, if you figure out how to work with the `scipy.optimiz` function `minimize`, it should be rather straightforward.

When using `minimize`, you again need to provide parameter bounds and initial parameters guesses, say

$$\alpha = 2 \qquad \text{and} \qquad \beta = 200 \; .$$

If you use these, then which final estimates do you get for α and β ? Please report them.

It would surprising if your result would be as in the previous tasks. So, here is a plot of your instructor's result. Observe that the semantics of the y-axis has changed!!!



typical result for task 2.5

Remark: Of all the methods we considered so far, this is the first which does not require us to scale the estimated probabilities f_j to match the given counts c_j. It does, however, require us to make an evidenceless 98% assumption for which we cannot know if it holds true (it seems plausible but could be any other percentage, we just don't know) . . .

task 2.6 [5 points]

fitting a Weibull model to a discrete distribution (part 2)

In the previous task, you were given a sample distribution $\boldsymbol{q} = [q_1, \ldots, q_n]$ indicating the probability of observing events (searches for "*myspace*") at times $\boldsymbol{t} = [t_1, \ldots, t_n]$. Using this information, you had to find the parameters of a Weibull distribution such that

$$q_j \approx f(\, t_j \mid \alpha, \beta) \; .$$

Next, you will have to solve this fitting task via the method of moments.

background info: moment matching

Recall that, if T is a Weibull distributed random variable $T \sim f(\, t \mid \alpha, \beta)$, then its *expected value* or *mean* and *variance* are defined to be

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

Note: These two quantities are also called *first-* and *second moment* (of the Weibull distribution).

Tedious calculus reveals that the above integrals have the following closed form solutions

$$\mu = \beta \, \Gamma\big(1 + \tfrac{1}{\alpha}\big) \tag{5}$$

$$\sigma^2 = \beta^2 \Big[\Gamma\big(1 + \tfrac{2}{\alpha}\big) - \Gamma^2\big(1 + \tfrac{1}{\alpha}\big)\Big] \; . \tag{6}$$

The *coefficient of variation* $CV^2 = \frac{\sigma^2}{\mu^2}$ of a Weibull distribution is therefore

$$\frac{\sigma^2}{\mu^2} = \frac{\Gamma\big(1 + \frac{2}{\alpha}\big) - \Gamma^2\big(1 + \frac{1}{\alpha}\big)}{\Gamma^2\big(1 + \frac{1}{\alpha}\big)} \tag{7}$$

Note: $\Gamma(\cdot)$ is the Gamma function. In practice, we can evaluate this transcendental function using the `scipy.special` function `gamma`.

Now, consider this: Given a sample distribution $\boldsymbol{q}$ over $\boldsymbol{t}$, we may compute the *sample mean* and *sample variance*

$$m = \sum_{j=1}^{n} p_j \, t_j \tag{8}$$

$$s^2 = \sum_{j=1}^{n} p_j \, t_j^2 - m^2 \, . \tag{9}$$

Given these empirical moments, we may match them with their theoretical counterparts in (5) and (6). In other words, we may substitute $\frac{s^2}{m^2}$ for $\frac{\sigma^2}{\mu^2}$ on the left of (7) which then allows for model fitting via the following two step procedure

a) solve

$$\frac{s^2}{m^2} = \frac{\Gamma\bigl(1 + \frac{2}{\alpha}\bigr) - \Gamma^2\bigl(1 + \frac{1}{\alpha}\bigr)}{\Gamma^2\bigl(1 + \frac{1}{\alpha}\bigr)}$$

for the Weibull shape parameter α

b) given α, solve

$$\beta = \frac{m}{\Gamma\bigl(1 + \frac{1}{\alpha}\bigr)}$$

for the Weibull scale parameter β.

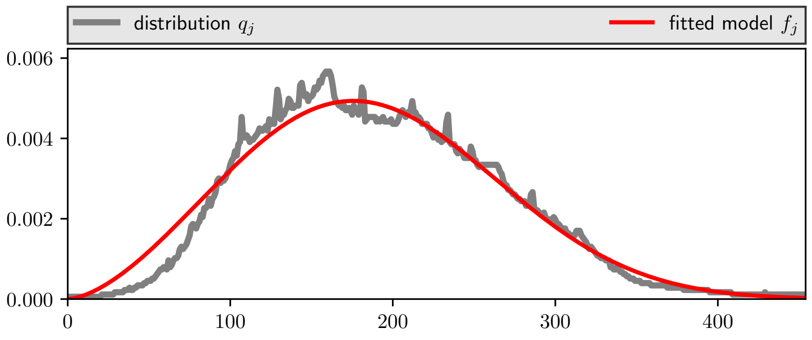
here is your task:

Compute $\boldsymbol{q}$ as in the previous task and code the above fitting procedure. Use your code to fit a Weibull to the given search frequencies q_j and report the estimates you obtain for α and β.

⚠

Note: While the computations required for b) are straightforward, there is no closed form solution for the problem in a). We therefore suggest using the `scipy.optimize` function `root_scalar` to estimate α. This requires careful thinking about how to cast the problem at hand but it can be done!

For reference, here is a plot of your instructor's result:



typical result for task 2.6

task 2.7 [5 points]

numerically solving differential equations

Throughout this course, we shall frequently consider differential equation solving as this is currently an interesting use case for machine learning.

Luckily, our setting in this exercise provides an opportunity for getting used to this ridiculously important domain. First of all, we already know that

$$f(\,t \mid \alpha, \beta) = \tfrac{\alpha}{\beta}\big(\tfrac{t}{\beta}\big)^{\alpha-1}\big(1 - F(\,t \mid \alpha, \beta)\big)\ .$$

Second of all, we should be able to convince ourselves (by using pen-and-paper calculations, by running `sympy` computations, or by asking an AI) that

$$f(\,t \mid \alpha, \beta) = \tfrac{d}{dt}\,F(\,t \mid \alpha, \beta) \equiv \dot{F}(\,t \mid \alpha, \beta)\ .$$

Putting the above together and reducing notational clutter, we thus obtain the following ordinary differential equation

$$\dot{F}(t) = \tfrac{\alpha}{\beta}\big(\tfrac{t}{\beta}\big)^{\alpha-1}\big(1 - F(t)\big)\ . \tag{10}$$

Of course we know that this equation has an analytical solution, namely the Weibull cumulative density in (2). However, in the vast majority of use cases for differential equation solving, there are no analytical- or closed form solutions. Physicists, chemists, or engineers and therefore usually have to rely on numerical solvers . . .

here is your task:

Let $\boldsymbol{t} = [t_1, \ldots, t_n]$ as in the previous tasks and consider

$$\begin{aligned}\alpha &= 2.86\\ \beta &= 215.91\ .\end{aligned}$$

Working with these parameters, apply the `scipy.integrate` method `solve_ivp` to numerically solve (10).

This should produce an array of n numbers; let's call it $\boldsymbol{F}^n = [F^n_1, \ldots, F^n_n]$.

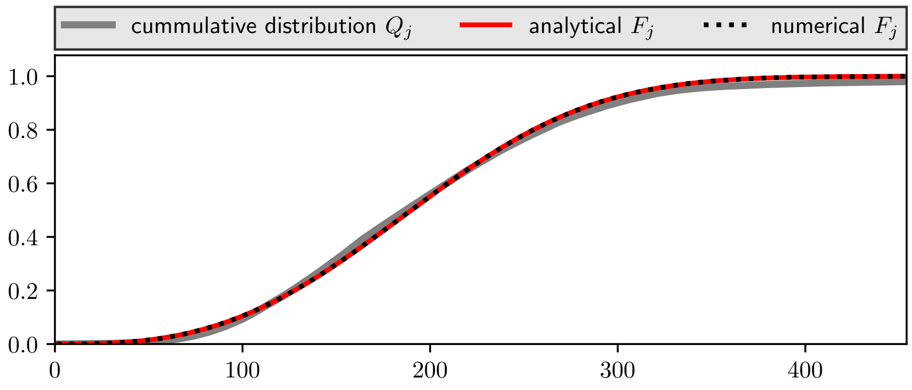
Next, to visually verify your result, proceed as follows: Let $\boldsymbol{q} = [q_1, \ldots, q_n]$ as in the previous tasks and compute $\boldsymbol{Q} = [Q_1, \ldots, Q_n]$ where

$$Q_j = \sum_{i=1}^{j} q_i \; .$$

Also, apply (2) to analytically compute an array $\boldsymbol{F}^a = [F_1^a, \ldots, F_n^a]$ where

$$F_j^a = F(\, t_j \mid \alpha = 2.86, \beta = 215.91\,) \; .$$

Finally, plot $\boldsymbol{Q}$ against $\boldsymbol{t}$, $\boldsymbol{F}^a$ against $\boldsymbol{t}$, and $\boldsymbol{F}^n$ against $\boldsymbol{t}$. Your result should again resemble your instructor's result which looks like this:



result for task 2.7

task 2.8 [25 bonus points]

what about neural networks ?

⚠

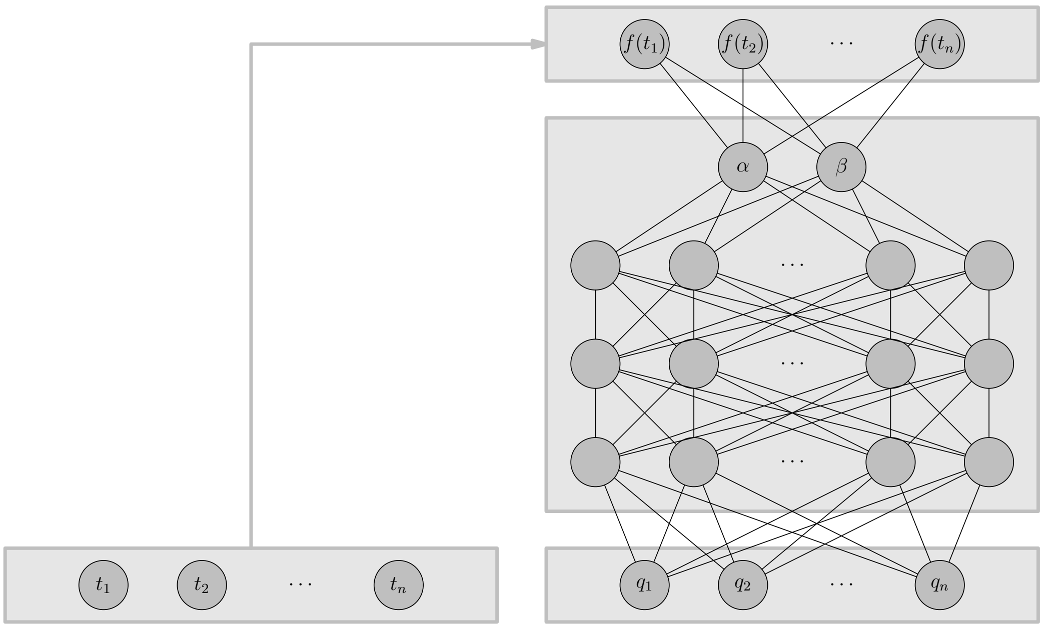
Note: This task is *not* mandatory but a bonus task for ambitious (?!?) teams of students!

Our goal here is to check backgrounds and ambitions of participants of this course: What can we expect you to already know about modern machine learning? Or, how much effort are you willing to spend to get bonus points? We therefore neither provide suggestions for detailed model design and training nor for best coding practices.

Note: This bonus task is hit or miss! Either your submitted code makes perfect sense and delivers flawless results or it doesn't. If everything checks out, you'll receive the bonus; if not, then not.

here is your bonus task:

Let $\boldsymbol{t} = [t_1, \ldots, t_n]$, $\boldsymbol{q} = [q_1, \ldots, q_n]$, and $f(\, t_j \mid \alpha, \beta)$ as in the previous tasks. Use `jax` (or `pytorch` or `tensorflow` if you have experience with those) to implement, train, and run the following model:



task 2.9

submission of presentation and code

As always, prepare a set of slides on your solutions and results. Task 2.1 does not need to be discussed!

W.r.t. to formalities, please make sure that

- your presentation contains a title slide with names and matriculation numbers of everybody in your team who contributed to the solutions.

W.r.t. content, please make sure that

- your presentation contains about 12 to 15 content slides but not more
- your presentation is concise and clearly structured
- your presentation answers questions such as
  - "what was the task / problem we considered?"
  - "what kind of difficulties (if any) did we encounter?"
  - "how did we solve them?"
  - "what were our results?"
  - "what did we learn?"

Save / export your slides as a PDF file and upload it to eCampus.

As always, please name all your code files in a manner that indicates which task they solve and put them in an archive or a ZIP file.

Upload this archive / ZIP file to eCampus.