Lab Course SS2022:
Data Science in Astrophysics
Session 4: Linear Regression

Florian List*

March 31, 2022

*florian.list@univie.ac.at

1 Introduction

Probably you have already encountered **linear regression** at high school or during your Bachelor studies. In the simplest case in one dimension, linear regression consists in fitting a line to a set of N data points $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $(x_1, \ldots, x_N)$ are the inputs and $(y_1, \ldots, y_N)$ are the observations associated with each input. You might remember that in this case, the "best-fit" line can be determined by minimizing the **mean squared error** between the observations y_n and the line of slope m evaluated at the data points, $\tilde{y}_n = mx_n$ for $n = 1, \ldots, N$ (see Fig. 1 *left*), i.e.

$$\text{Find } m \text{ such that } \sum_{n=1}^{N} (y_n - \tilde{y}_n)^2 = \sum_{n=1}^{N} (y_n - mx_n)^2 \to \min. \tag{1}$$

While finding model parameters that minimize the error between the model and the observations as in Eq. (1) intuitively makes sense, we can ask ourselves several questions:

1. How can we generalize Eq. (1) to **multi-dimensional** inputs $\mathbf{x}$?
2. Why do we minimize the mean **squared** error and not e.g. the mean **absolute** error or any other function that compares the distance between the model values and the actual measurements?
3. Under what **assumptions** on the data-generating process is this procedure justified?
4. Assuming an **uncertainty** σ_n is associated with each measurement (x_n, y_n), how can we include this knowledge in the regression?
5. How can we generalize Eq. (1) to **general functions**, e.g. polynomials?
6. How can we ensure that our model **generalizes** to new data without **overfitting**? $\to$ *next lecture*
7. How can we include **prior knowledge** that we might already have about the parameters? $\to$ *next lecture*

In this lecture, we will consider linear regression from a more rigorous probabilistic / statistical perspective. In particular, we will show how the least-square minimization (and generalizations thereof) naturally arise once a **probabilistic model** for the **noise** underlying the observations is specified, and we will answer the questions above.

2 A probabilistic perspective on regression

Just as we did in earlier lectures in this course, we treat all data as arising from a random **data generating process**. We start by postulating that the random process that produces an observation $y \in \mathbb{R}$ from an input vector $\mathbf{x} \in \mathbb{R}^D$ can be decomposed into

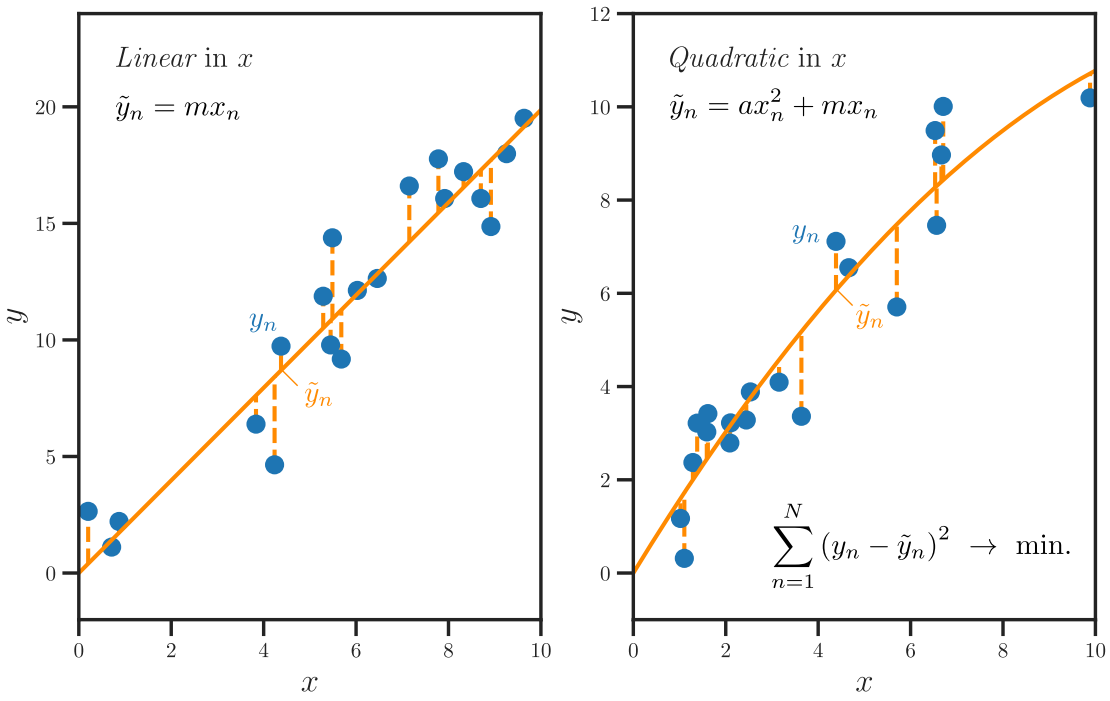

Figure 1: Linear regression in 1D. The orange best-fit line minimizes the sum of the squared deviations between the model $\tilde{y}_n$ and the observations y_n over $n = 1, \ldots, N$. In the left panel, the model function is *linear* in x, i.e. $f(x) = mx$, see Eq. (1). In the right panel, there is an additional parabolic term, so the model is given by $f(x) = ax^2 + mx$. Still, in both cases, f is linear *in the parameters* (m for the linear function on the left, $\{a, m\}$ for quadratic function on the right).

a **deterministic** part $f = f(\mathbf{x})$ that describes a functional relationship between each input $\mathbf{x}$ and an expected outcome $f(\mathbf{x})$ and a **random noise** contribution ε, i.e.

$$y = f(\mathbf{x}) + \varepsilon, \tag{2}$$

see the illustration in Fig. 2. Assuming that the noise is independent, identically distributed drawn from a **normal distribution** with a known variance σ^2 and mean zero[1] for all the data points, we can write down the probability of observing y given an input vector $\mathbf{x}$ as

$$p(y \mid \mathbf{x}) = \mathcal{N}\left(y \mid f(\mathbf{x}), \sigma^2\right). \tag{3}$$

This probability is known as the **likelihood function**. The task of regression is now to find (an approximation of) the function f underlying the data generation process. We will always consider specific classes of functions that are parameterized by a K-dimensional parameter vector $\boldsymbol{\theta} \in \mathbb{R}^K$. Finding the function f within the chosen class of functions then only requires us to determine the parameter vector $\boldsymbol{\theta}$.

Remark 1. We will assume in this lecture that the uncertainties in the independent variable $\mathbf{x}$ are negligible. In reality, measurements often come with error bars in x- and y-direction, in which case more advanced methods are needed (see e.g. Sec. 8.8.1 in Ref. [2]).

3 Fitting a linear function

Let us start with the simple case of fitting a linear function $f : \mathbb{R}^D \to \mathbb{R}$, $f : \mathbf{x} \mapsto y$, which can be written as

$$f(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \boldsymbol{\theta}^\top \mathbf{x}, \tag{4}$$

where the parameter vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_D)^\top \in \mathbb{R}^D$ contains one parameter for each dimension of $\mathbf{x}$ (and therefore $K = D$, see Fig. 3 for $K = D = 2$). For simplicity, we assume that the uncertainty in the *input* $\mathbf{x}$ is negligible and that the uncertainties in the dependent variable y are Gaussian and **homoscedastic** (i.e. the uncertainty variance of y has the same value σ^2 for all measurements $n = 1, \ldots, N$). This means that we can write the likelihood function as

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}\left(y \mid \boldsymbol{\theta}^\top \mathbf{x}, \sigma^2\right). \tag{5}$$

Remark 2 (Linearity). Even when the model function $f = f(\mathbf{x})$ is not linear in $\mathbf{x}$, we are dealing with a linear regression problem as long as the model is **linear in the model parameters**, see also Fig. 1 (*right*). Consider the following examples in one dimension (i.e. $D = 1$, so $\mathbf{x}$ becomes x):

[1] The assumption of zero mean for the noise does not affect the generality of the model because an offset $\mu(\mathbf{x})$ can be absorbed by the deterministic part of the model by setting $f_{\text{new}}(\mathbf{x}) := f(\mathbf{x}) + \mu(\mathbf{x})$.

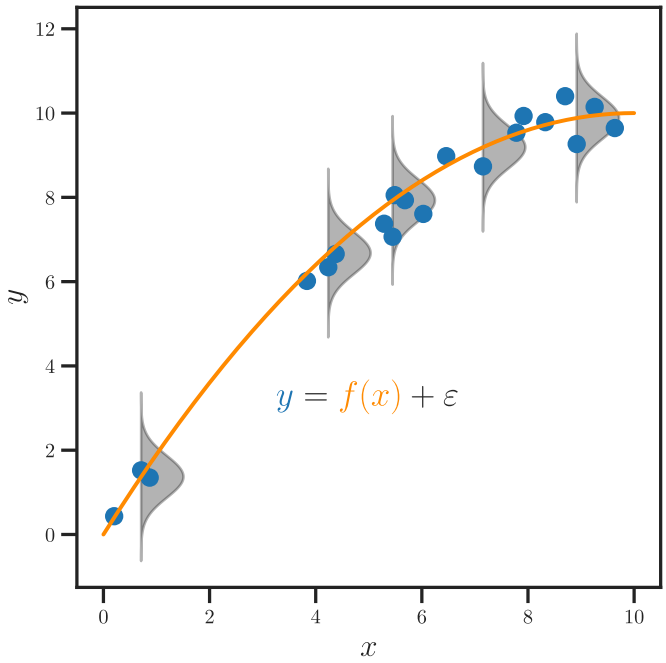

Figure 2: We interpret the measurements $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ as being generated by an underlying data-generating process, which can be split up into a deterministic part and a probabilistic part. This is illustrated here for the one-dimensional case: the measurements (blue data points) y_n can be decomposed into the sum of $f(x_n)$ (orange line, the underlying "trend") and a random noise contribution $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. The Gaussian probability distribution function characterizing the noise ε is plotted for some values of x_n, centered around $f(x_n)$ with fixed variance σ^2. For a family of functions f_K with K free parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_K)^\top \in \mathbb{R}^K$, the task of regression is to recover the function f (or equivalently the parameter vector $\boldsymbol{\theta}$, once a family of functions f_K has been chosen) that generated the data.

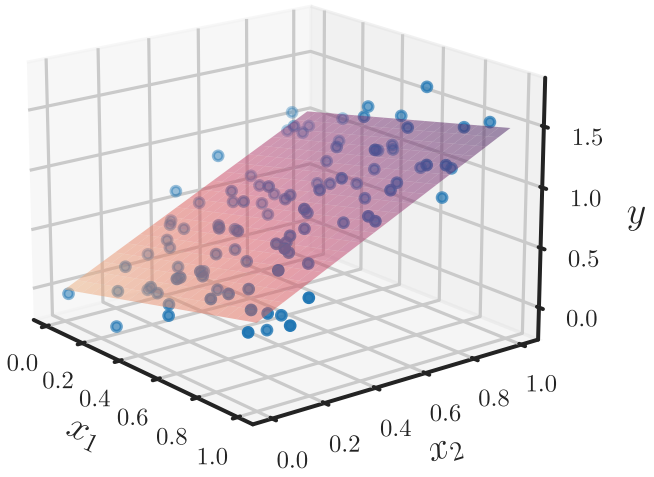

Figure 3: Fitting a linear function in $D = 2$ dimensions (see Eq. (4)). This model has $K = D = 2$ free parameters, namely θ_1 and θ_2, and the model function f is given by $f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} = \theta_1 x_1 + \theta_2 x_2$.

- For polynomial regression, we have $f(x) = \sum_{k=0}^{K-1} \theta_k x^k$, so f is linear in the coefficients $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{K-1}) \in \mathbb{R}^K$.
- The same is the case for the model $f(x) = \theta_1 \sin(x) + \theta_2 \cos(x)$.

Examples for **nonlinear** regression models are

- $f(x) = \theta_1 \sin(\theta_2 x)$ and
- $f(x) = \theta_1 \exp(-\theta_2 x^2)$.

For nonlinear regression problems, iterative methods such as the Levenberg–Marquardt algorithm are often employed. We will not consider nonlinear regression in this lecture.

3.1 Maximum likelihood estimation

Let us assume we are given a set of training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ consisting of N inputs $\mathcal{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ together with associated observations $\mathcal{Y} = (y_1, \ldots, y_N)$. Note that each measurement y_n only depends on its input $\mathbf{x}_n$, and another measurement y_m given $\mathbf{x}_m$ is independent of y_n given $\mathbf{x}_n$. Therefore, we can write the likelihood as a

product over all the measurements

$$p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = p\left(y_1, \ldots, y_N \mid \mathbf{x}_1, \ldots, \mathbf{x}_N, \boldsymbol{\theta}\right) \tag{6a}$$

$$= \prod_{n=1}^{N} p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right) = \prod_{n=1}^{N} \mathcal{N}\left(y_n \mid \boldsymbol{\theta}^\top \mathbf{x}_n, \sigma^2\right), \tag{6b}$$

making use of the assumption that the measurement noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is Gaussian.

Now, we will consider **maximum likelihood estimation**. This means, we will determine the parameter vector $\boldsymbol{\theta}^\star$ that maximizes the likelihood in Eq. (6) or in formula

$$\boldsymbol{\theta}^\star = \operatorname*{argmax}_{\boldsymbol{\theta}} p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}). \tag{7}$$

Note that although the likelihood depends on the parameter vector $\boldsymbol{\theta}$, it is a probability distribution in the **observations** $\mathcal{Y}$ and not in $\boldsymbol{\theta}$. In particular, this means that in general $\int p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta} \neq 1$ and in fact, this integral does not even have to exist. Intuitively, maximizing the likelihood means maximizing the predictive probability distribution of the training data $\mathcal{D}$, given the parameter vector $\boldsymbol{\theta}$.

Remark 3 (Log-likelihood). In order to simplify the computation of the maximum likelihood estimate, we will apply the **logarithm** to the likelihood, noting that a strictly monotonically increasing function leaves the location of the maximum unchanged. Applying the logarithm often turns out to be a useful trick when dealing with probabilities because firstly, the logarithm turns the product over the different measurements into a sum that is easier to handle and secondly, the logarithm prevents numerical underflow, which can occur when multiplying many very small probabilities. For example, $\underbrace{10^{-4} \times \ldots \times 10^{-4}}_{N=10 \text{ times}} = 10^{-40}$, which is smaller than the smallest positive floating point number that can be represented by a 32-bit floating point number ($\approx 10^{-38}$). In contrast, $\underbrace{\log_{10}(10^{-4}) + \ldots + \log_{10}(10^{-4})}_{N=10 \text{ times}} = -40$, which does not pose any numerical challenge. This argument is independent of the basis of the logarithm as a change of basis from a to b simply corresponds to a multiplication with $\log_b a$.

It is customary to *minimize* the negative log-likelihood (rather than to *maximize* the log-likelihood), given by

$$-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = -\log\left(\prod_{n=1}^{N} p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right)\right) = -\sum_{n=1}^{N} \log p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right). \tag{8}$$

Recalling that the Gaussian probability distribution function with mean μ and variance σ^2 is given by

$$p(y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right), \tag{9}$$

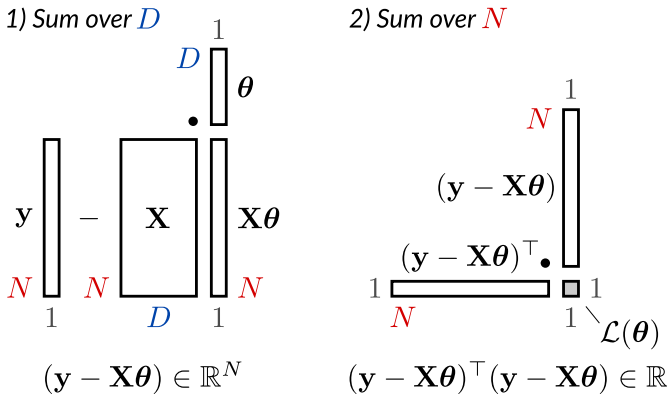

Figure 4: Dimensions of the matrices and vectors involved in Eq. (11b). *Left:* the vector $\mathbf{y} \in \mathbb{R}$ contains the N observations, the design matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ contains the D-dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ as rows, and the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^D$ contains the D parameters, one per dimension. The matrix-vector product $\mathbf{X}\boldsymbol{\theta} \in \mathbb{R}^N$ carries out the sum over the D dimensions in Eq. (4), for each measurement $n = 1, \ldots, N$. *Right:* The inner product $(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$ sums over the N observations, see Eq. (11a), from which one obtains the value of $\mathcal{L}(\boldsymbol{\theta}) \in \mathbb{R}$ (after division by $2\sigma^2$, which is not shown in this illustration).

we find the Gaussian log-likelihood for each measurement to be

$$\log p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right) = -\frac{1}{2\sigma^2}\left(y_n - \mathbf{x}_n^\top \boldsymbol{\theta}\right)^2 - \frac{\log(\sigma^2)}{2} - \frac{\log(2\pi)}{2}. \tag{10}$$

Since the last two terms are constant with respect to $\boldsymbol{\theta}$, we can ignore them for the determination of $\boldsymbol{\theta}^\star$. This yields

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{2\sigma^2}\sum_{n=1}^{N}\left(y_n - \mathbf{x}_n^\top \boldsymbol{\theta}\right)^2 \tag{11a}$$

$$= \frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2 \to \text{min.}, \tag{11b}$$

where we gather all the observations in the vector $\mathbf{y} = (y_1, \ldots, y_N)^\top \in \mathbb{R}^N$ and all the inputs in the **design matrix** $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^\top \in \mathbb{R}^{N \times D}$. The n-th row of the design matrix contains the D-dimensional input $\mathbf{x}_n$, see Fig. 4. Equation (11b) is the generalization of the 1D minimization problem in Eq. (1) to multiple dimensions, so we have already found the answer to question 1.

Example 1 (1D case). Let us come back again to the $D = 1$-dimensional case that we considered in the introduction. In that case, our model function is a line (rather than a hyperplane), and we only have a single parameter m that expresses the slope of the fitted line and plays the role of the parameter vector $\boldsymbol{\theta}$. Then, the expression in

Eq. (11a) becomes

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - mx_n)^2 \propto \sum_{n=1}^{N}(y_n - mx_n)^2\,, \tag{12}$$

which is exactly the expression in Eq. (1). Therefore, we can now answer questions 2 and 3 in the introduction: under the assumption that the data generating process underlying our observations consists of a deterministic part $f(x)$ and normally distributed noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, **minimizing the mean squared error** between our model predictions and the observations is **equivalent to maximizing the likelihood** (the probability to observe our data, given the parameter m). Although we are currently only looking at linear functions in x, we will see later that the negative log-likelihood is still proportional to $\sum_{n=1}^{N}(y_n - \tilde{y}_n)^2$ for more general functions f, where $\tilde{y}_n = f(x_n)$.

Since Eq. (11b) is quadratic in $\boldsymbol{\theta}$, we can find a unique minimizer of $\mathcal{L}(\boldsymbol{\theta})$ by computing the gradient of the log-likelihood with respect to the parameter vector and setting the gradient to zero. Explicitly writing out the product in Eq. (11b) yields

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) = \mathbf{y}^\top\mathbf{y} - 2\mathbf{y}^\top\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\theta}, \tag{13}$$

which leads to

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\left(-2\mathbf{y}^\top\mathbf{X} + 2\boldsymbol{\theta}^\top\mathbf{X}^\top\mathbf{X}\right) \tag{14}$$

Setting this to zero results in

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}^\top \iff \boldsymbol{\theta}^{\star\top}\mathbf{X}^\top\mathbf{X} = \mathbf{y}^\top\mathbf{X} \tag{15a}$$

$$\iff \boldsymbol{\theta}^{\star\top} = \mathbf{y}^\top\mathbf{X}\left(\mathbf{X}^\top\mathbf{X}\right)^{-1} \tag{15b}$$

$$\iff \boxed{\boldsymbol{\theta}^\star = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}.} \tag{15c}$$

In order for this to work, we require the matrix $\mathbf{X}^\top\mathbf{X} \in \mathbb{R}^{D\times D}$ to be invertible. This is the case if the design matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$ has rank D, in which case $\mathbf{X}^\top\mathbf{X}$ is positive definite. If we have at least as many data points as parameters (or equivalently dimensions in the present case of a linear function), i.e. $N \geq D$, assuming that all the data points in $\mathbf{X}$ are distinct.

Moore–Penrose inverse Clearly, a rectangular matrix $\mathbf{A} \in \mathbb{R}^{r\times s}$ with $r \neq s$ does not have a "classic" inverse matrix. For $r < s$, the linear mapping represented by $\mathbf{A}$ from $\mathbb{R}^s$ to $\mathbb{R}^r$ cannot be one-to-one. On the other hand, for $r > s$, $\mathbf{A}$ maps from the lower-dimensional space $\mathbb{R}^s$ to $\mathbb{R}^r$ and can therefore not be onto. Still, for *any* real (or complex) matrix, there exists a "pseudo-inverse" matrix $\mathbf{A}^+$, known as the **Moore–Penrose inverse**, which has the following properties:

1. $AA^+A = A$,
2. $A^+AA^+ = A^+$,
3. $(AA^+)^\top = AA^+$,
4. $(A^+A)^\top = A^+A$.

Notice that all these properties are trivially satisfied for the inverse matrix A^{-1} in case it exists. When the columns of $\mathbf{A}$ are linearly independent, the Moore–Penrose inverse can be computed as

$$A^+ = (A^\top A)^{-1}A^\top. \tag{16}$$

Coming back to the maximum-likelihood estimate for the parameter vector $\boldsymbol{\theta}$ in Eq. (15c), we now identify the matrix on the right-hand side as the Moore–Penrose inverse of $\mathbf{X}$, and we arrive at the compact equation

$$\boxed{\boldsymbol{\theta}^\star = \mathbf{X}^+\mathbf{y}.} \tag{17}$$

In PYTHON, this can be implemented as follows (here as an example for $D = 2$):

```
import numpy as np
np.random.seed(0)
N = 100  # number of data points
theta_0, theta_1 = 2.0, 4.0  # set the "true" slopes
X = np.random.uniform(0, 10, (N, 2))  # randomly sample N inputs X (in 2D)
f = theta_0 * X[:, 0] + theta_1 * X[:, 1] # define f(x)
y = f + np.random.normal(0, 1.5, N)  # add some measurement noise (mean 0, STD: 1.5)
theta_ml = np.linalg.pinv(X) @ y  # get max. LLH estimate for theta
print(f"Correct:  {theta_0}, {theta_1}")
print(f"Max. LLH: {theta_ml[0]}, {theta_ml[1]}")

>>> Correct:  2.0, 4.0
>>> Max. LLH: 1.9515205012466486, 4.018801984168837
```

In practice, however, computing the Moore–Penrose inverse (which is usually done via a singular value decomposition) is typically slower than solving the linear system of equations

$$\mathbf{X}^\top\mathbf{X}\boldsymbol{\theta}^\star = \mathbf{X}^\top\mathbf{y}, \tag{18}$$

or in PYTHON:

```
theta_ml = np.linalg.solve(X.T @ X, X.T @ y)
print(f"Max. LLH: {theta_ml[0]}, {theta_ml[1]}")

>>> Max. LLH: 1.9515205012466466, 4.01880198416884
```

Just try it out for yourself and see which one is faster, e.g. for $N = 10{,}000$.

3.2 Heteroscedastic uncertainties

So far, we have assumed that the *uncertainty is the same for all the measurements* $n = 1, \ldots, N$, i.e. $\sigma_n^2 = \sigma^2$. What if this is no longer the case and the uncertainties are instead **heteroscedastic**? From a probabilistic point of view, this corresponds to a model where the noise ε_n at each measurement $n = 1, \ldots, N$ is described by a Gaussian $\varepsilon_n \sim \mathcal{N}(0, \sigma_n^2)$ with n-dependent variance. Just like in Eq. (10) for the homoscedastic case, we find the likelihood for each measurement to be

$$\log p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right) = -\frac{1}{2\sigma_n^2}\left(y_n - \mathbf{x}_n^\top \boldsymbol{\theta}\right)^2 - \frac{\log(\sigma_n^2)}{2} - \frac{\log(2\pi)}{2}, \tag{19}$$

where the only difference is now the the subscript n for the noise variance σ_n^2. As before, the last two terms are independent of $\boldsymbol{\theta}$ and can be ignored when determining the maximum likelihood estimate $\boldsymbol{\theta}^\star$, implying that we should minimize

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{2}\sum_{n=1}^{N}\left(\frac{y_n - \mathbf{x}_n^\top \boldsymbol{\theta}}{\sigma_n}\right)^2 =: \sum_{n=1}^{N}\chi_n^2(\boldsymbol{\theta} \mid \mathbf{x}_n, y_n, \sigma_n^2) =: \chi^2(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}, (\sigma_1^2, \ldots, \sigma_N^2)). \tag{20}$$

This χ^2-statistic is often found in astronomical literature and expresses an error-normalized residual. Intuitively, when minimizing Eq. (20), a mismatch between the model and the observation will be penalized less when the uncertainty of the measurement is large and vice versa. Defining the uncertainty covariance matrix as

$$\mathbf{C} = \begin{pmatrix} \sigma_1^2 & 0 & \ldots & 0 \\ 0 & \sigma_2^2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & \sigma_N^2 \end{pmatrix}, \tag{21}$$

we can write Eq. (20) as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top \mathbf{C}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}), \tag{22}$$

similarly to Eq. (11b) in the homoscedastic case. The matrix $\mathbf{C}^{-1}$ is known as the **precision matrix**. Carrying out the same minimization as before (cf. Eq. (15a)), we arrive at

$$\boxed{\boldsymbol{\theta}^\star = \left(\mathbf{X}^\top \mathbf{C}^{-1} \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{C}^{-1} \mathbf{y}.} \tag{23}$$

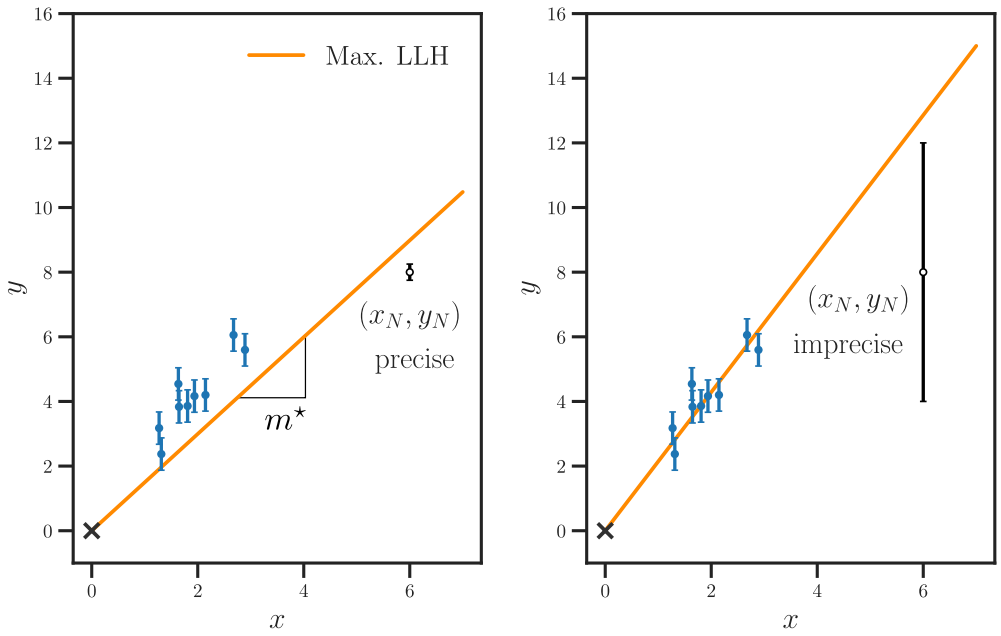

Figure 5: Heteroscedastic uncertainties in $D = 1$ dimension for a linear model $f(x) = mx$: $\sigma_n = 0.5$ for $n = 1, \ldots, N-1$ (blue points), but the last point N (black) has a different uncertainty σ_N, which is much smaller / larger in the left / right panel. The orange line shows the maximum likelihood estimate for the slope of the line $m^\star$ computed with Eq. (23). The small uncertainty σ_N causes the fit to be strongly sensitive to the measurement (x_N, y_N), and the line lies far off all the other measurements. The larger the uncertainty σ_N, the less weight is placed on measurement N, and for the case shown in the right panel, the maximum likelihood estimate is barely influenced by y_n. An interactive version of the figure is available on Moodle.

The effect of different uncertainty levels on the result of the fit is demonstrated in Fig. 5 for the $D = 1$-dimensional case, which is based on the following code:

```
from matplotlib import pyplot as plt
import numpy as np
np.random.seed(0)
N = 10  # number of data points
m = 2.0  # set the "true" slope
x = np.random.uniform(0, 3, N - 1)  # randomly sample the first N - 1 inputs x
x.sort()  # sort for simplicity
sigma_all_but_last = 0.5  # define a noise level for these points
y = m * x + np.random.normal(0.0, sigma_all_but_last, N - 1)  # define y = f(x) + eps

# Add an extra data point far away to the right that lies far off the line y = m * x
x = np.hstack([x, 6.0])
y = np.hstack([y, 8.0])

# Let's consider two different (artificial) uncertainties for the last point
sigmas_last = [0.5 * sigma_all_but_last, 8.0 * sigma_all_but_last]

x_plot = np.linspace(0, 7, 1000)

fig, axs = plt.subplots(1, 2, figsize=(10, 6))
for i, (sigma, ax) in enumerate(zip(sigmas_last, axs)):
    # Define the precision matrix (diagonal)
    C_inv = np.diag(1.0 / np.hstack([sigma_all_but_last ** 2 * np.ones(N - 1),
                                     [sigma ** 2]]))
    # Get the maximum likelihood estimate
    m_ml = np.linalg.solve(x[np.newaxis, :] @ C_inv @ x[:, np.newaxis],
                           x[np.newaxis, :] @ C_inv @ y)
    # Plot
    fit = ax.plot(x_plot, m_ml * x_plot, color="darkorange")
    ax.errorbar(x=x[:-1], y=y[:-1], yerr=sigma_all_but_last, elinewidth=2, capsize=2,
                ls="none", markersize=4, marker="o")
    ax.errorbar(x=x[-1:], y=y[-1:], yerr=sigma, elinewidth=2, capsize=2,
                ls="none", markersize=4, marker="o", ecolor="k", mfc="1.0", mec="k")
    ax.scatter(0, 0, marker="x", color="0.2")
    if i == 0:
        ax.set_title(r"$(x_N, y_N)$: precise")
        ax.legend(fit, ("Max. LLH",))
    else:
        ax.set_title(r"$(x_N, y_N)$: imprecise")
    ax.set_xlabel(r"$x$")
    ax.set_ylabel(r"$y$")
    ax.set_ylim([-1, 16])
```

Notice that using `np.linalg.solve` is actually a bit of an overkill because our "system" of equations is 1D, and we could simply write instead

```
sigmas = np.hstack([sigma_all_but_last ** 2 * np.ones(N - 1), [sigma ** 2]])
m_ml = ((x * y) / sigmas ** 2).sum() / ((x / sigmas) ** 2).sum()
```

The code above can however easily be adapted to the multidimensional case. In this section, we have found the answer to question 4 in the introduction: we can include knowledge about different uncertainties in different points by minimizing the sum of the χ^2-statistics, which represent **error-normalized** residuals.

4 Fitting general functions

In the last section, we considered fitting a linear D-dimensional function to N measurements. Now, we are ready to consider the general case of *arbitrary* model functions $f = f(\mathbf{x})$ as shown in Fig. 2. The only requirement that we keep is that f should still be linear in its parameters, which means we can write it as

$$f(\mathbf{x}) = \sum_{k=1}^{K} \theta_k \phi_k(\mathbf{x}). \tag{24}$$

with K arbitrary functions $\phi_k = \phi_k(\mathbf{x})$ and parameters θ_k. See Remark 2 for examples of model functions f that can be written in this form (and others that cannot). To give another example, in the case of a linear function that we considered in the previous section, we had $\phi_k(\mathbf{x}) = x_k$ and $K = D$.

4.1 Maximum likelihood estimation

We now consider the more general probabilistic model

$$y = \sum_{k=1}^{K} \theta_k \phi_k(\mathbf{x}) + \varepsilon = \boldsymbol{\phi}^\top(\mathbf{x})\boldsymbol{\theta} + \varepsilon \tag{25a}$$

$$\iff \quad p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y \mid \boldsymbol{\phi}^\top(\mathbf{x})\boldsymbol{\theta}, \sigma^2). \tag{25b}$$

Here, $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^K$ is called the **feature vector**, which represents a (potentially non-linear) mapping that gathers the K transformations $\phi_k : \mathbb{R}^D \to \mathbb{R}$ of the D-dimensional inputs $\mathbf{x}$, for $k = 1, \ldots, K$. Note that we consider homoscedastic uncertainties again and assume a fixed noise variance σ^2 for all the points, but the extension to heteroscedastic uncertainties is analogous to Sec. 3.2.

Example 2 (Polynomial regression)**.** A common scenario is that the model function f is a polynomial of degree $K - 1$. Let us consider this example in the one-dimensional case $D = 1$. The feature vector is given by

$$\boldsymbol{\phi}(x) = (\phi_0(x), \phi_1(x), \ldots, \phi_{K-1}(x))^\top = (1, x, x^2, \ldots, x^{K-1})^\top \in \mathbb{R}^K. \tag{26}$$

This means that the one-dimensional inputs x are mapped to all the monomials x^k for $k = 0, \ldots, K - 1$. Plugging this feature vector into Eq. (25a) yields the model

$$y = \boldsymbol{\phi}(x)^\top \boldsymbol{\theta} + \varepsilon = \sum_{k=0}^{K-1} \theta_k x^k + \varepsilon, \tag{27}$$

where each of the K parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{K-1})^\top$ is the coefficient belonging to one monomial.

Just as before we have

$$-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right), \tag{28}$$

and we can write down the likelihood function for a single measurement n

$$\log p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right) = -\frac{1}{2\sigma^2}\left(y_n - \boldsymbol{\phi}^\top(\mathbf{x}_n)\boldsymbol{\theta}\right)^2 - \frac{\log(\sigma^2)}{2} - \frac{\log(2\pi)}{2}. \tag{29}$$

The only difference in comparison with Eq. (10) is that we now replaced $\mathbf{x}_n \in \mathbb{R}^D$ by $\boldsymbol{\phi}(\mathbf{x}_n) \in \mathbb{R}^K$. Recall that before, we collected all the inputs in the design matrix $X \in \mathbb{R}^{N \times D}$. Now, we can apply the same idea and collect the *transformed* inputs in the **feature matrix**

$$\boldsymbol{\Phi} := \begin{pmatrix} \boldsymbol{\phi}^\top(\mathbf{x}_1) \\ \vdots \\ \boldsymbol{\phi}^\top(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_K(\mathbf{x}_1) \\ \phi_1(\boldsymbol{x}_2) & \cdots & \phi_K(\boldsymbol{x}_2) \\ \vdots & & \vdots \\ \phi_1(\boldsymbol{x}_N) & \cdots & \phi_K(\boldsymbol{x}_N) \end{pmatrix} \in \mathbb{R}^{N \times K}, \tag{30}$$

where $\boldsymbol{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$ and $\phi_k : \mathbb{R}^D \to \mathbb{R}$.

Example 3 (Feature matrix for second order polynomials in 2D)**.** Let us consider the case of fitting a second order polynomial in $D = 2$ variables $\mathbf{x} = (\xi, \eta)^\top$. The feature matrix takes the form

$$\Phi = \begin{pmatrix} 1 & \xi_1 & \eta_1 & \xi_1^2 & \eta_1^2 & \xi_1\eta_1 \\ 1 & \xi_2 & \eta_2 & \xi_2^2 & \eta_2^2 & \xi_2\eta_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \xi_N & \eta_N & \xi_N^2 & \eta_N^2 & \xi_N\eta_N \end{pmatrix}. \tag{31}$$

In general, for a dataset with D-dimensional inputs, there are

$$K = \frac{(D+r)!}{D!\, r!} \tag{32}$$

coefficients when fitting an r-th order polynomial, including the intercept. In our case, we have $D = 2$, $r = 2$ and therefore $K = 4!/(2!\, 2!) = 6$ model parameters.

Using the feature matrix, we write the negative log-likelihood as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2\sigma^2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^\top(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) \tag{33}$$

where we omitted again the terms that are independent of $\boldsymbol{\theta}$ (cf. Eq. (11b) for the linear model). Similar to the linear model, we can compute the gradient of the negative log-likelihood with respect to the parameter vector $\boldsymbol{\theta}$ and set it to zero to find the maximum

likelihood estimate for the general case

$$\boxed{\boldsymbol{\theta}^\star = \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \mathbf{y} \quad \Longleftrightarrow \quad \boldsymbol{\theta}^\star = \boldsymbol{\Phi}^+ \mathbf{y}.} \tag{34}$$

Here, the feature matrix $\boldsymbol{\Phi} \in \mathbb{R}^{N \times K}$ now takes the role of the design matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ for the linear model. In order for $\left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\right)^{-1}$ to be invertible, we now require that $\boldsymbol{\Phi}$ has rank K, which implies that the columns of $\boldsymbol{\Phi}$ have to be linearly independent (note that we assume $N \geq K$).

For heteroscedastic noise σ_n^2, $n = 1, \ldots, N$, we obtain similarly

$$\boxed{\boldsymbol{\theta}^\star = \left(\boldsymbol{\Phi}^\top \mathbf{C}^{-1} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \mathbf{C}^{-1} \mathbf{y},} \tag{35}$$

where $\mathbf{C}$ is defined in Eq. (21).

4.2 Unknown noise

Let us consider one more aspect of linear regression related to the uncertainties. So far, we have assumed that the noise – homoscedastic or heteroscedastic – is *known*. What if this is not the case? We will see in a minute that the framework of maximum likelihood estimation also allows us to obtain a maximum likelihood estimate for the unknown noise level in the data. In order to find the value $\sigma^2 > 0$ that maximizes the likelihood, let us compute the partial derivative of the negative log-likelihood with respect to σ^2 and set it to zero, just as we did for the parameter vector $\boldsymbol{\theta}$. Going back again to Eq. (29) and keeping the second term with the log-variance (that we now want to optimize), we find that

$$-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}, \sigma^2) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(y_n - \boldsymbol{\phi}^\top(\mathbf{x}_n)\boldsymbol{\theta}\right)^2 + \frac{N}{2} \log \sigma^2 + \frac{N}{2} \log(2\pi). \tag{36}$$

This yields

$$-\frac{\partial \log p\left(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}, \sigma^2\right)}{\partial \sigma^2} = -\frac{1}{2\sigma^4} \sum_{n=1}^{N} \left(y_n - \boldsymbol{\phi}^\top(\mathbf{x}_n)\boldsymbol{\theta}\right)^2 + \frac{N}{2\sigma^2}. \tag{37}$$

By setting this to zero, we obtain the maximum likelihood estimate $(\sigma^\star)^2$

$$(\sigma^\star)^2 = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \boldsymbol{\phi}^\top(\mathbf{x}_n)\boldsymbol{\theta}\right)^2. \tag{38}$$

Intuitively, the maximum likelihood estimate $(\sigma_\star)^2$ for the variance is given by the empirical mean of the squared distances between the model values $\boldsymbol{\phi}^\top(\mathbf{x}_n)\,\boldsymbol{\theta}$ and the data points y_n over all the measurements $n = 1, \ldots, N$.

4.3 Laplace-distributed noise

Throughout this lecture, we have assumed that the measurement noise is Gaussian, i.e. $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ for some constant or $\mathbf{x}$-dependent σ^2. For completeness, let us briefly consider a case of non-Gaussian noise, drawn instead from a **Laplace distribution** (shown in Fig. 6). The probability distribution function of the Laplace distribution is given by

$$p(y \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|y - \mu|}{b}\right), \tag{39}$$

where μ and b are a location and scale parameter, respectively. Consider the model

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \text{Laplace}(f(\mathbf{x}), b) = \text{Laplace}(\boldsymbol{\phi}^\top(\mathbf{x})\boldsymbol{\theta}, b), \tag{40}$$

or equivalently

$$y = f(\mathbf{x}) + \varepsilon, \tag{41}$$

where $\varepsilon \sim \text{Laplace}(0, b)$ with a fixed scale parameter $b > 0$. Now, the log-likelihood is given by

$$-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p\left(y_n \mid \mathbf{x}_n, \boldsymbol{\theta}\right) = \frac{1}{b} \sum_{n=1}^{N} \left| y_n - \boldsymbol{\phi}^\top(\mathbf{x}_n)\boldsymbol{\theta} \right| + N \log(2b). \tag{42}$$

Thus, in order to find the maximum likelihood estimate $\boldsymbol{\theta}^\star$ for this model with Laplace-distributed noise, one needs to minimize the sum of **absolute errors** (also known as the l^1 error) between the model and the observations. Unfortunately, this cannot be done analytically unlike in the case of square errors arising from the assumption of Gaussianity for the noise. The minimization of Eq. (42) is typically done iteratively, for example using linear programming. Minimizing the l^1 error is considered to be more robust than minimizing the l^2 error (that is, the mean squared error) because the l^1 error gives equal weight to all the observations, while the l^2 error grows quadratically as the residual grows. Unlike in the l^2 case where a unique maximum likelihood estimate exists for $N \geq K$ provided the columns of $\boldsymbol{\Phi}$ are linearly independent, the minimizer of Eq. (42) does not need to be unique (see Fig. 7 for an intuitive argument).

The l^1 and l^2 fits for a linear model function $f(x) = mx$ in $D = 1$ dimension are compared in Fig. 8. In the left panel, all the data points follow the linear model and are only perturbed by Gaussian noise. In this case, both fits agree quite well. In the right panel, two data points have been artificially moved, representing outliers in the data. The l^2 fit is sensitive to the outliers, and the best-fit line no longer follows the trend of the other data points. In contrast, the l^1 fit is robust to the outliers and barely changes. In the next lecture, we will look at other ways to make regression **robust**. The code used to generate this figure is provided below. For the minimization of the loss functions, we use `scipy.optimize.fmin` here.

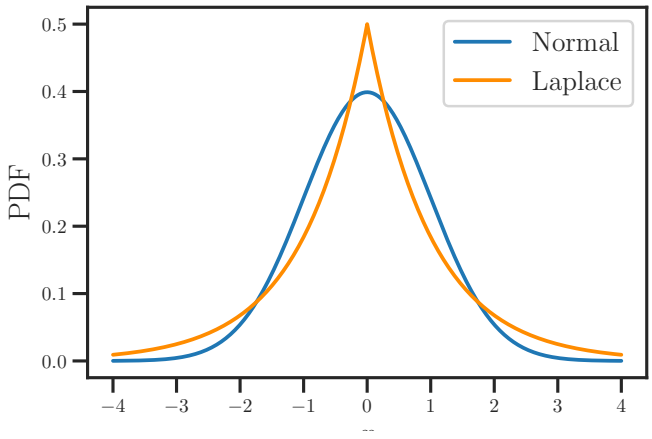

Figure 6: Probability distribution functions of the Gaussian normal distribution (blue) and the Laplace distribution (orange). The Laplace distribution is sharply peaked at 0, and its tails fall off much more slowly than for a Gaussian.

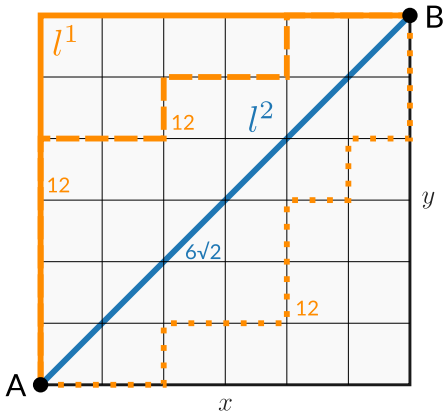

Figure 7: Non-uniqueness of the l^1 minimizer in two dimensions. All the orange paths from A to B have the same (minimal) length as measured by the l^1-distance $d_1(A, B) = |x_A - x_B| + |y_A - y_B| = 6 + 6 = 12$. In contrast, the blue line shows the unique shortest path from A to B with respect to the l^2-distance $d_2(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} = 6\sqrt{2}$.

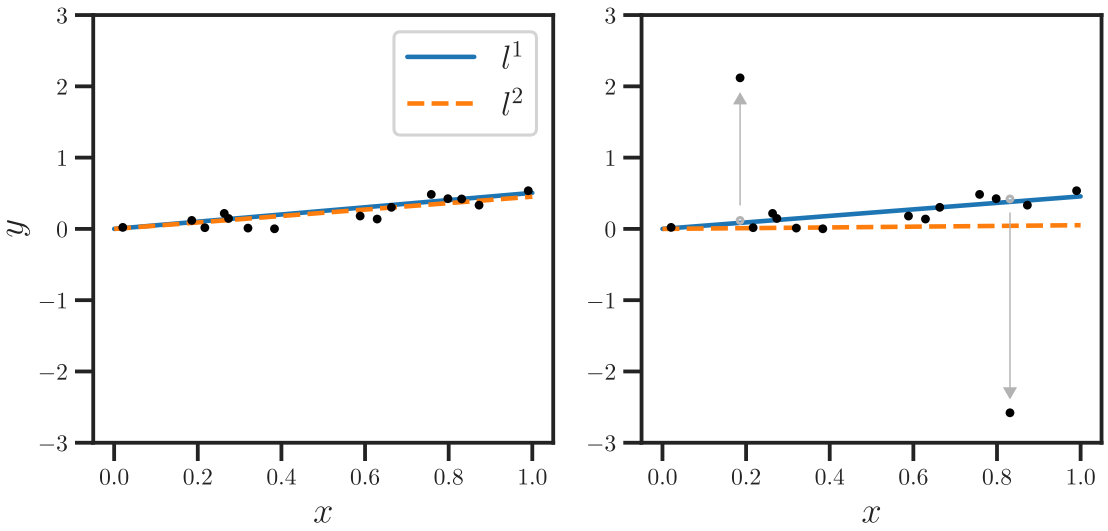

Figure 8: Best-fit lines $f(x) = mx$ found by minimizing the l^1 error and the l^2 error. In the left panel, the data points have been generated as $y_n = 0.5x_n + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, 0.01)$. In the right panel, two data points have been artificially moved away from the line, representing outliers in the data. The robustness of the l^1 fit is clearly visible. An interactive version of this figure is available on Moodle.

```
import numpy as np
from scipy import optimize
# Generate some data with noise
x = np.sort(np.random.uniform(0, 1, 15))
y = 0.5 * x + 0.1 * np.random.randn(len(x))
x_plot = np.linspace(0, 1, 1001, endpoint=True)

# Define the loss functions (l1 and l2)
l1_loss = lambda m, x, y: np.sum(np.abs(m * x - y))
l2_loss = lambda m, x, y: np.sum((m * x - y) ** 2)

# Minimizize
m_l1 = optimize.fmin(func=l1_loss, x0=1, args=(x, y))
m_l2 = optimize.fmin(func=l2_loss, x0=1, args=(x, y))

# Now, generate two outliers
y[1] += 2.0
y[12] -= 3.0

# Fit again
m_l1_outliers = optimize.fmin(func=l1_loss, x0=1, args=(x, y))
m_l2_outliers = optimize.fmin(func=l2_loss, x0=1, args=(x, y))
```

References

[1] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning.* Cambridge University Press, 2020.

[2] Ž. Ivezić, A. J. Connolly, J. T. van der Plas, and A. Gray. *Statistics, Data Mining, and Machine Learning in Astronomy: A Practical Python Guide for the Analysis of Survey Data, Updated Edition.* Princeton University Press, 2019.