Chapter 3

Classical Statistics, Estimation and Uncertainties

We have now covered the basic mathematical foundations of probability and statistics. You should now understand what a probability is, what a distribution of probabilities (a p.d.f.) means and how we can manipulate and propagate errors. The last section of this course will now deploy this to perform actual statistical analysis of datasets. In particular we are interested in making *estimates* from data and also at the same providing *estimates* of those *estimates'* uncertainties. In other words we want to make measurements and quote confidence intervals for those measurements. Another important weapon in our statistical armoury will be understanding how to distinguish between different hypotheses. Sometimes we don't want to measure a particular value we just want to know if one hypothesis is significantly more likely than another (*e.g.* which of these two models is a more accurate description of nature). These are the topics we will cover in this section.

3.1 Frequentist evaluation of estimators

An *estimate* in a statistical sense can be interpreted as a measurement (or estimate) of a parameter based on a limited number of observations. In classical statistics a *point estimatation* is an estimate of the true value of a parameter, whilst an *interval estimation* is an estimate of a range of possible values within which the true value exists a certain frequency of the time. In Bayesian statistics all parameters are considered as random variables, so estimations attempt to infer the probability distribution of that parameter, which can then be used to quote a central value or interval.

In order to produce an estimate of a parameter, which we will always label with a "hat" over the parameter, we need some procedure (a function or an algorithm) which produces that estimate based on the observations we have, this is called the *estimator*. The numerical value the *estimator* produces based on the observations is called the *estimate*.

A small note on notation is that following other conventions I will normally label parameters we want to estimate with greek letters, θ, ϕ *etc.* and estimates of them with hats over them, $\hat{\theta}$, $\hat{\phi}$ *etc.*. Random variables I will continue to label with captial roman letters, X, Y, Z *etc.* although sometimes I am lazy and use lower case x, y, z. Remember that for

a Bayesian, and in Bayesian analysis, parameters are also *also* random variables, whereas for a frequentist parameters are considered fixed.

3.1.1 Consistency, bias and efficiency of estimates

An estimate can be almost anything you like. If I want to know the average height of students in this class I could decide that my *estimator* will be to add up the number of minutes each student has been alive and divide it by π. This fulfils the properties we need of estimator but it will clearly produce an absolutely dreadful estimate. Consequently we need to define, or at least consider, some properties of estimators that we desire to make sure our estimates are as accurate and precise as possible.

Consistency

An estimator is said to be *consistent* if the estimate it produces tends to the true value as the datasize increases. If we define an estimator $f(x_i)$ which produces an estimate $\hat{\theta} = f(x_i)$ of the true value of θ given the data we observe x_i, then the estimate will not necessarily be exactly θ for a finite sample, because it contains statistical flucutations. However a consistent estimator should be close and should get closer as the sample size increases,

$$\lim_{N\to\infty} \hat{\theta} = \theta. \tag{3.1}$$

Another way of thining about this is that the variance of the estimator should get smaller as the sample size increases. An example of a consistent estimator is the arithmetic mean of some randomly sampled population.

Bias

The bias of an estimator is defined as the deviation of the expectation of the estimate, $E[\hat{\theta}]$, from the true value, θ_0,

$$b_N(\hat{\theta}) = E[\hat{\theta}] - \theta_0 = E[\hat{\theta} - \theta_0], \tag{3.2}$$

where the subscript $_N$ labels the bias for a given number of observations, N. An estimator is said to be *unbiased* if for any sample size $b_N(\hat{y}) = 0$ which is equivalent to saying that the expectation value of the estimate should equal the true value, $E[\hat{\theta}] = \theta_0$.

In general it is preferable to have a consistent estimator as opposed to a biased estimator. A graphical illustration of the difference is shown in Fig. 3.1.

Efficiency

Depending on the particular data that is obtained then the value of the estimate, $\hat{\theta}$, will vary. We want the measurement to be a good estimate of the true value so we want the spread of possible values of the estimate to be as small as possible, in other words have minimal variance. This is known as an *efficient* estimator.

3.1.2 Bias-variance trade-off

In general we want our estimators to be as efficient as possible, for example why base an estimator on 100 data points of a sample if I have 200 data points available. However,

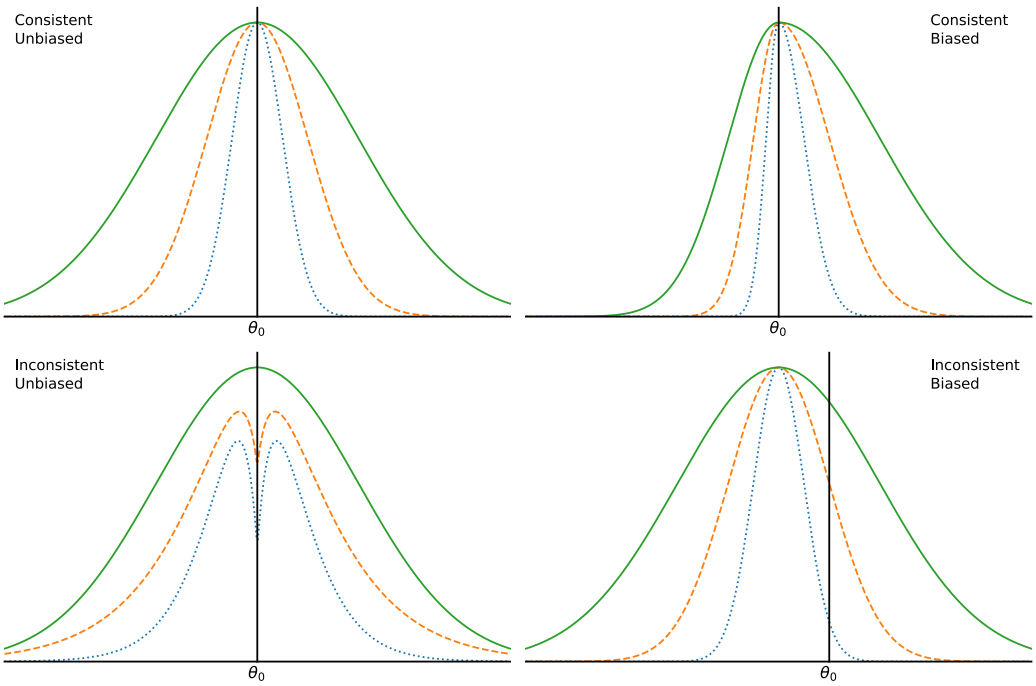


Figure 3.1: An illustration of the difference between consistency and bias. The examples show an imagined probability distribution for an estimate $\hat{\theta}$ and where it convereges as the sample size increases. An unbiased and consistent estimator should converge on the true value θ_0 with decreasing spread. The sample size is increasing from green-to-orange-to-blue / solid-to-dashed-to-dotted. Top left shows an unbiased and consistent estimator. Top right shows a biased but consistent estimator (variance decreases). Bottom left shows an unbiased and inconsistent estimator. Bottom right shows biased and inconsistent. Reproduced from Ref. [1].

there is sometimes a trade-off, because a more efficient estimator may be more biased. In the example above maybe the 100 extra data points I have are taken under some other conditions or in some different configuration to the first 100.

This is when you have to make a judgement. There is no unambiguous "best" estimator and you may find you have to find the sweet spot between bias and efficiency. Typically we can estimate the bias and efficiency using bootstrapping methods (see Sec. 3.9 below) and this will then allow you to make an at least informed and quantitative decision about the relative merits. For scientists the distinction is often made as whether you count an uncertainty as belonging to the "statistical" uncertainty (based on random flucutations within the sample) or the "systematic" uncertainty (shifting or scaling of the distirbution based on the equipement or sampling procedure used).

Relation to accuracy-precision

This question, or topic, you may have seen before but in a less mathematically rigorous way and is often taught to undergraduates as the accuracy-precision trade off. Accuracy, in this sense, is the equivalent of bias, how close are my measurements to the true value? Precision, is the equivalent of efficiency, what is the spread of my measurements? These differences are graphically illustrated in the probably familiar Fig. 3.2.

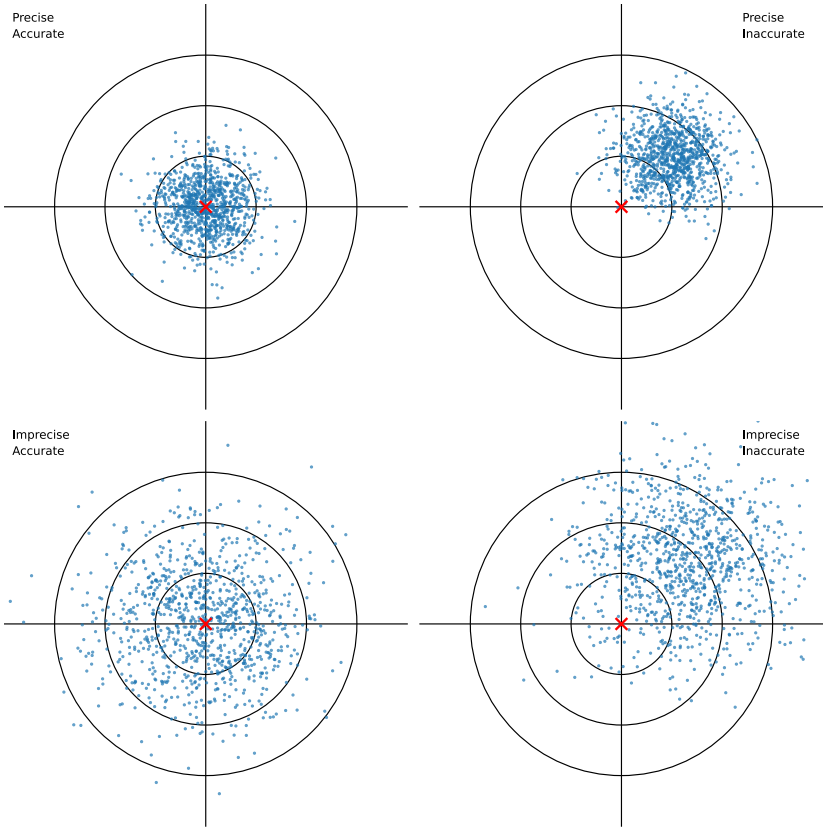


Figure 3.2: A demonstration of the differences between accuracy/bias and precision/efficiency. The distribution of data points in the top left are accurate and precise (unbiased and efficient), top right are precise but not accurate (so need an additional systematic uncertainty), bottom left are accurate but not precise (so have a large statistical uncertainty but no bias), bottom right are neither accurate nor precise (biased and inefficient).

3.1.3 Estimates of the mean, variance and standard deviation

The arithmetic mean of a sample provides a consistent and unbiased estimate of the true mean, *i.e.*

$$\hat{\mu} = E[x] = \bar{x}. \tag{3.3}$$

The central limit theorem tells us that the variance of this estimate is

$$V(\hat{\mu}) = \frac{\sigma^2}{N}, \tag{3.4}$$

where σ^2 is the variance of the parent distribution and N is the size of the sample used in the estimate. Note that the above is the variance or spread of the *estimate* itself and is ***not*** the same as the variance of the sample. In other words if I estimate the mean from the sample, $\hat{\mu}$, then the square-root of the variance in Eq. 3.4, is the *standard error on the mean*, $\sigma_{\hat{\mu}} = \sigma/\sqrt{N}$. If the parent distribution the sample originates from is Gaussian then we can prove the estimate is maximally efficient (I don't do it here but set it as one of the problems[1]). For other distributions the estimate may or may not be efficient. In practise

[1] To solve the problem you will need to read a bit further down in the notes until you get to the minimum variance bound.

we often do not know the variance of the parent distribution and so it is replaced by the sample estimate.

Now let us discuss the various different estimates for the variance. First, the case where we know the true mean, μ, for which we can then estimate the variance as

$$\hat{V} = \frac{1}{N}\sum(x_i - \mu)^2. \tag{3.5}$$

This is an unbiased and consistent estimate because its expectation value is the variance,

$$E[V] = \frac{NE[(x-\mu)^2]}{N} = E[(x-\mu)^2] = V(x). \tag{3.6}$$

However, in practise we will almost certainly not know the true mean μ. So if we decide to replace it with our estimate of the true mean, $\hat{\mu}$, then we get

$$\hat{V} = \frac{1}{N}\sum(x_i - \hat{\mu})^2 = \frac{1}{N}\sum(x_i^2 - \bar{x}^2). \tag{3.7}$$

If we now take the expectation value of this estimate (and use the fact that $E[x] = E[\bar{x}]$ and that $V(\bar{x}) = V(x)/N$ from (3.3) and (3.4)) then

$$\begin{aligned}
E[\hat{V}] &= \frac{NE[x^2 - \bar{x}^2]}{N} && (3.8)\\
&= E[x^2] - E[\bar{x}^2] && (3.9)\\
&= E[x^2] - E[\bar{x}^2] + E[\bar{x}]^2 - E[x]^2 && (3.10)\\
&= (E[x^2] - E[x]^2) - (E[\bar{x}^2] - E[\bar{x}]^2) && (3.11)\\
&= V(x) - V(\bar{x}) && (3.12)\\
&= V(x) - \frac{V(x)}{N} && (3.13)\\
&= \frac{N-1}{N}V(x) \neq V(x). && (3.14)
\end{aligned}$$

So we see that this gives us a biased estimate of the variance. However the bias gets progressively smaller with N and eventually the factor $(N-1)/N \to 1$ as $N \to \infty$. This factor of $N/(N-1)$ we have derived in (3.14) is known as *Bessel's correction* and exactly compensates for the bias, so that a consistent and unbiased estimate of the variance (providing $N > 1$) is

$$\hat{V} = s^2 = \frac{1}{N-1}\sum(x_i - \bar{x})^2. \tag{3.15}$$

It is also useful to know what the variance of this estimate is, in other words what is the *standard error on the variance*. I will not do the proof here (you can find it in Barlow Chapter 5 [2]) but the results are that for an unbiased estimate of the variance on the estimated variance, including Bessel's correction, one obtains

$$V(\hat{V}) = \frac{2\sigma^4}{N-1}. \tag{3.16}$$

In order to estimate the standard deviation we simply take the square root of the estimate of the variance,

$$\hat{\sigma} = \sqrt{\hat{V}} = s, \tag{3.17}$$

and an estimate of the standard deviation of this estimate, including Bessel's correction, is

$$\sigma_{\hat{\sigma}} = \sigma_s = \frac{\sigma}{\sqrt{2(N-1)}}. \tag{3.18}$$

From all of this there are various points to remember:

- We can always estimate the mean of a sample, (3.3), which will be an unbiased and consistent estimate of the true mean
- In order to compute the error on the mean, (3.4), we need to know the true standard deviation, σ, which in most cases we don't. Consequently we have to replace it with an estimate of the standard deviation, $\hat{\sigma}$, which means our estimate of the standard error, $\hat{\sigma}_{\hat{\mu}}$, is an approximation and is in general biased although it is consistent
- We can estimate the standard deviation of a sample using Bessel's correction, (3.17), which will be an unbiased and consistent estimate of the true standard deviation
- In order to estimate the error on the standard deviation estimate, (3.18), we need to know the true standard deviation, σ, which in most cases we don't (and if we did why would we bother trying to estimate the spread of our estimate of it). So once again we replace it with the estimate above of the standard deviation, $\hat{\sigma}$, so that out estimate of the standard error on the standard deviation, $\hat{\sigma}_{\hat{\sigma}}$, is an approximation which is biased but consistent.

In summary then, if we have a dataset we have not seen before and we do not know the parent distribution, or the properties of it, *a priori* we can make estimates of the mean and deviation and estimates of the deviations of those two estimates using:

$$\text{Estimate of the mean:} \quad \hat{\mu} = \frac{1}{N}\sum_i x_i \tag{3.19}$$

$$\text{Estimate of the standard deviation:} \quad \hat{\sigma} = \frac{1}{N-1}\sum_i (x_i - \hat{\mu})^2 \tag{3.20}$$

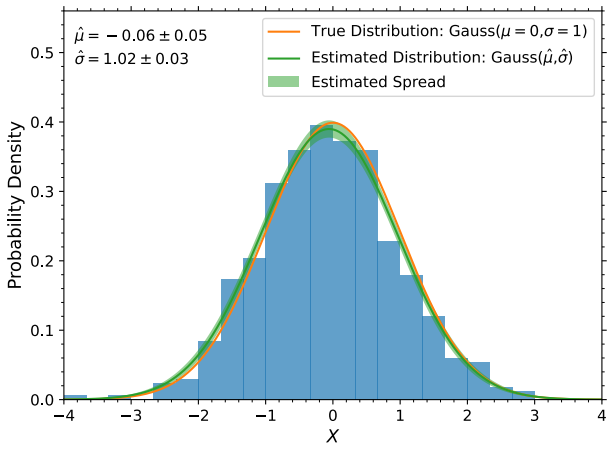
$$\text{Error on the mean estimate:} \quad \sigma_{\hat{\mu}} = \frac{\hat{\sigma}}{\sqrt{N}} \tag{3.21}$$

$$\text{Error on the standard deviation estimate:} \quad \sigma_{\hat{\sigma}} = \frac{\hat{\sigma}}{\sqrt{2(N-1)}} \tag{3.22}$$

Below is a little snippet of the code I use to make these estimates with a little plot showing an example dataset

```
import numpy as np

# generate some data
N = 500
dset = np.random.normal(size=N)

# estimate of sample mean
mu_hat = np.mean(dset)

# estimate of sample sdev
sg_hat = np.std(dset, ddof=1)

```

```
# estimate of err on mean
mu_hat_err = sg_hat/N**0.5

# estimate of err on sdev
sg_hat_err = sg_hat/(2*(N-1))**0.5
```



3.2 The likelihood function

The likelihood function is a fundamental concept which forms the basis of hypothesis testing and statistical inference, in both classical and Bayesian contexts. It is also a vital component in Information Theory, which we do not have time to discuss in this course, but for interested readers there are in-depth discussions in Chapter 5 of James [1] and Chapter 17 of Kendall [5].

The *likelihood function* kind of does what it says on the tin. It tells us the *likelihood* (in arbitary / meaningless units) that we would observe the dataset we have given a particular set of parameter values. In order to compute the likelihood we will need to use some model, *i.e.* a probability distribution, which we expect the data to follow, although this model may have parameters which we do not know.

Let's think about a set of N independent observations of a random variable X, $\{X_1, X_2, \ldots, X_n\}$, which are identically distributed according to $f(X|\theta)$, where θ represent the parameters of the model. The joint p.d.f. of the set X, given that they are independent, is then the product of the individual p.d.f.s:

$$p(X|\theta) = \prod_{i=1}^{N} f(X_i|\theta). \tag{3.23}$$

If I replace the variable X with a set of actual observations, X_0, then p is no longer a p.d.f., because it does not depend anymore on the random variable X, it is fixed by the particular set of observations X_0. This is then the product over each observation of the p.d.f. given the set of parameters θ. This is called the *likelihood function*, and is a function of the parameters θ only. It is often denoted with a L and sometimes written as $L(X_0, \theta)$ or just $L(\theta)$ to convey the fact it only depends on the parameters

$$L(\theta) = p(X_0|\theta). \tag{3.24}$$

Here are a few important points to remember about the likelihood function:

- The likelihood is ***not*** a probability distribution - do not confuse it with a p.d.f.. It is the product of p.d.f.s over the observations.
- The likelihood is a function of the parameters, θ, it is not a function of the random variable(s), X.
- What the likelihood is telling us is the probability (or likelihood) that we observe the dataset X given a particular value of the random variables θ.

To provide an illustrative example of the likelihood I have a little code block below, 3.1, which produces the plots in Fig. 3.3. This aims to show you what the difference is between a p.d.f. and the likelihood. In this case the p.d.f. is a Gaussian distribution, which depends on a random variable X, and has parameters μ and σ. I use this to generate some pretend observations, X_0. I then calculate the likelihood, which depends on the parameters μ and σ. It's worth realising that for a Gaussian probability distribution both the p.d.f. itself and the likelihood peak. For any distribution the likelihood will peak regardless of the probability. This will become important in the next section (Sec. 3.3) when we realise we can make estimates of the μ and σ parameters by maximising the likelihood, *i.e.* finding where the peak position of the likelihood is.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# define the pdf
pdf = norm(loc=0, scale=1)

# generate the data (250 events)
np.random.seed(210187)
x0 = pdf.rvs(size=250)

# now make a likelihood function (depends on mu and sg)
def likelihood(mu, sg):
    return np.exp( np.sum( norm.logpdf(x0, loc=mu, scale=sg) ) )

# this is non-optimal workaround to vectorize the above function
vec_likelihood = np.vectorize( likelihood )

# now draw the pdf
fig, ax = plt.subplots()
x = np.linspace(-4,4,200)
ax.hist( x0, range=(-4,4), bins='auto', density=True, label='Dataset $x_0$' )
ax.plot( x, pdf.pdf(x), label='p.d.f.' )
ax.legend()

# now draw the likelihood
fig, ax = plt.subplots(subplot_kw=dict(projection="3d"))
mu = np.linspace(-0.4,0.4,200)
sg = np.linspace(0.7,1.3,200)
lmu = np.array( [ likelihood(m, 1) for m in mu ] )
lsg = np.array( [ likelihood(0, s) for s in sg ] )
mu, sg = np.meshgrid(mu, sg)
L = vec_likelihood(mu,sg)
```

```
ax.plot_surface(mu, sg, L, cmap='coolwarm', lw=0)

plt.show()
```

Code Block 3.1: Code to illustrate the difference between a p.d.f. and the likelihood function. This produces the plots in Fig. 3.3.

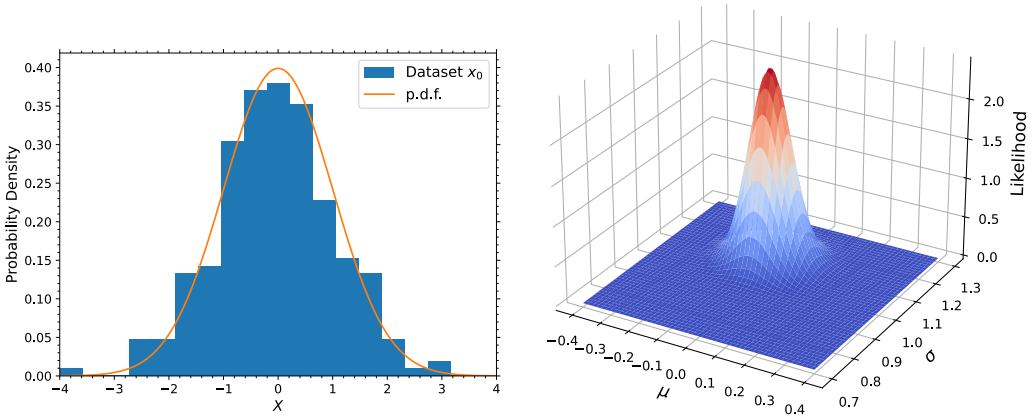


Figure 3.3: Illustration of the difference between a p.d.f. (left) and the likelihood function (right). Left: a Gaussian p.d.f. with $\mu = 0$ and $\sigma = 1$, which is a function of the random variable X, along with some data generated from that p.d.f., with values X_i. Right: the likelihood function for that generated data which is a function of μ and σ.

3.2.1 The minimum variance bound

There is another useful property of the likelihood which relates to efficiency of estimators. It can be shown (*proof in the lectures*[2]) that there is a limit to the efficiency of an estimator called the *minimum variance bound*. Providing the estimator is unbiased then the *minimum variance bound* states that

$$V(\hat{\theta}) \geq \left(E\left[\left(\frac{\partial \ln L}{\partial \theta} \right)^2 \right] \right)^{-1} \tag{3.25}$$

$$\geq -\left(E\left[\left(\frac{\partial^2 \ln L}{\partial \theta^2} \right) \right] \right)^{-1}. \tag{3.26}$$

An estimator is determined to be *efficient* if the variance of the estimator, $V(\hat{\theta})$, is the equal to the minimum variance bound. Otherwise an estimators efficiency is quantified by the ratio of the minimum variance bound to the variance.

3.3 Maximum likelihood method

The *maximum likelihood* (ML)[3] method allows us to make parameter estimates by finding the values which, suprise, surpise, maximise the likelihood. In general ML estimators are biased,

[2] see proofs also in Chapter 5 of Barlow [2] or Chapter 7 of James [1]

[3] It is rather inconvenient that the acronyms for *maximum likelihood* and *machine learning* are both the same, ML. This is unavoidable and you will have to infer the meaning from the context.

however they are consistent, and any consistent estimator will be unbiased as $N \to \infty$ so the bias vanishes as the sample size increases. We can also show the ML estimate is efficient.

Recall that the *likelihood function* is defined by the product of probabilities over the observations

$$L(\theta) = \prod_i p(X_i|\theta). \tag{3.27}$$

An estimate for the value of θ can be obtained by finding the value $\hat{\theta}$ which maximises $L(\theta)$. In other words solving the equation

$$\left.\frac{\partial L}{\partial \theta}\right|_{\theta=\hat{\theta}} = 0. \tag{3.28}$$

In practise it is *much* easier to work with the natural logarithm of the likelihood, because then the product turns into a sum, and at least for computers it is easier to sum small numbers than find their product. The $\hat{\theta}$ which maximises L will also be the value that maximise $\ln L$ so it is more common to the see ML condition written as

$$\left.\frac{\partial \ln L(\theta)}{\partial \theta}\right|_{\theta=\hat{\theta}} = \frac{\partial}{\partial \theta}\sum_{i=1}^{N} \ln p(X_i|\theta) = 0. \tag{3.29}$$

In some cases this can be solved analytically but more often than not it has to be solved numerically. There are some very good libraries available for minimisation[4] but one I would particulary recommend for this use case is `iminuit` [17] for reasons that will hopefully become more obvious later (it actually estimates derivatives of the likelihood as well thus giving us uncertainties on our estimates).

It can be shown that the variance of the estimate from the ML method is equal to the *minimum variance bound* when $N \to \infty$. This means that the ML method is asymptotically efficient. A rather stunning property of the maximum likelihood method, which is only the case when the variance is the *minimum variance bound*, is that is turns out that

$$E\left[\frac{\partial^2 \ln L}{\partial \theta^2}\right] = \left.\frac{\partial^2 \ln L}{\partial \theta^2}\right|_{\theta=\hat{\theta}}. \tag{3.30}$$

In other words we compute the expecation of the double differential of the log likelihood simply by evaluating the double differential of the log likleihood at the estimated value. Then by deploying (3.26) under the condition that the variance is the minimum variance broad we can simply compute the variance of our estimate by evaluating the inverse of the double differential

$$V(\hat{\theta}) = -\left(\left.\frac{\partial^2 \ln L}{\partial \theta^2}\right|_{\theta=\hat{\theta}}\right)^{-1}. \tag{3.31}$$

This is what makes the `iminuit` package so useful for maximum likelihood estimation, because not only will it maximise the likleihood it will also (if instructed) evaluate the double differentials at the estimated value and provide an estimate of the uncertainties.

In general if the sample size is large, and in particular if $\partial^2 \ln L/\partial\theta^2$ is slowly varying in the region of the estimate, then the ML estimate $\hat{\theta}$ will be Gaussian distributed with variance given by (3.31). As such the log likelihood will map out a parabola in the parameter space

[4] In case it wasn't obvious you can minimise $-\ln L$ to get the equivalent result of maximising $\ln L$

and the standard deviation can be read directly off the plot. We will see specifically why in a moment, and also discuss this method of interval estimation in more detail later for higher parameter spaces in which we can deploy the profile likelihood, but for now I will simply state the quantities which are that estimates of the 1σ, 2σ, 3σ, $N\sigma$ deviations can be found as the values for which the difference in $\ln L$ from the maximum are 0.5, 2, 4.5, $N^2/2$. An example is shown in Fig. 3.4 which shows an example *negative* log likelihood function.

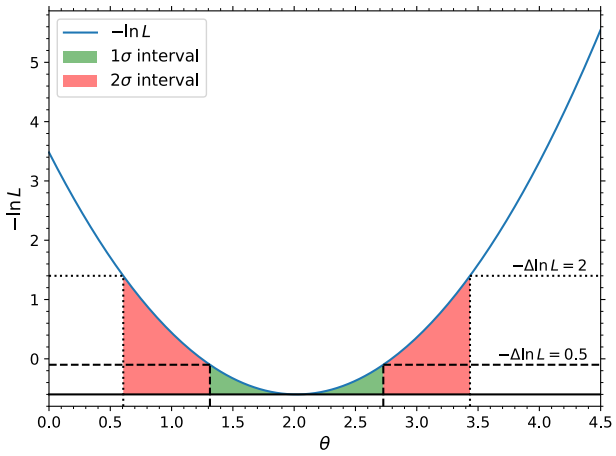


Figure 3.4: A demonstration of the negative log likelihood, showing where the 1σ and 2σ intervals can be read off.

3.3.1 The (Fisher) information

The quantity that gets inverted in (3.25) when evaluated at the maximum, *i.e.* where $\theta = \hat{\theta}$, is called the *information*, or sometimes the "Fisher" information (after Fisher).

If we have a likelihood function $L(\theta)$ then the partial derivative of the log-likelihood with respect to θ is called the *score*,

$$S(\theta) = \frac{\partial \ln L(\theta)}{\partial \theta}. \tag{3.32}$$

Thus our maximum likelihood estimation requires finding the θ for which the score is zero. We can show that the expected value of the score, if our r.v. X is really distributed according to our description in the likelihood, $X \sim f(X;\theta)$, evaluated at the true value of θ, is zero:

$$E[S(\theta)] = E\left[\frac{\partial \ln L(\theta)}{\partial \theta}\right] = E\left[\frac{\partial \ln f(X;\theta)}{\partial \theta}\right], \tag{3.33}$$

$$= \int \frac{\partial \ln f(X;\theta)}{\partial \theta} f(X;\theta) dX, \tag{3.34}$$

$$= \int \frac{\frac{\partial}{\partial \theta} f(X;\theta)}{f(X;\theta)} f(X;\theta) dX, \tag{3.35}$$

$$= \frac{\partial}{\partial \theta} \int f(X;\theta) dX = \frac{\partial}{\partial \theta} 1 = 0. \tag{3.36}$$

Notice the key steps in this proof that I know the true distribution $f(X;\theta)$ and I also evaluate at the true value of θ so can move the partial derivative outside of the integral. The result is what we would hope, *i.e.* that the score it most optimal (equal to zero) at the true value of θ.

Since the expectation of the score is zero, the variance of the score is simply,

$$V(S(\theta)) = E[S(\theta)^2] = E\left[\left(\frac{\partial \ln L(\theta)}{\partial \theta}\right)^2\right], \tag{3.37}$$

and we have recovered the minimum variance bound shown in (3.25). The variance of the score is called the *information* which can be thought of as the curvature of the log likelihood near the maximum. A low score indicates that the maximum is rather shallow / slow curving, a high score indicates it is sharply peaked, in other words there is *more information* about θ. Explicitly writing out the information in the context we have already seen above gives,

$$I(\theta) = E\left[\left(\frac{\partial \ln L(\theta)}{\partial \theta}\right)^2\right], \tag{3.38}$$

$$= -E\left[\frac{\partial^2 \ln L(\theta)}{\partial \theta^2}\right]. \tag{3.39}$$

Therefore notice that the minimum variance bound on the parameter $V(\theta)$ is the inverse of the information $I(\theta)$.

3.3.2 Many parameter likelihoods

We can use the ML method to extract estimates for many parameters at the same, as an example take a look at Fig. 3.3 which shows the two-dimensional likelihood for μ and σ. The values of $\hat{\mu}$ and $\hat{\sigma}$ which maximise the likelihood will be the estimates. In the $N \to \infty$ asymptotic limit these estimates will be distributed as multi-dimensional Gaussian distribution with the covariance generalising to the inverse of partial derivatives of the log likelihood,

$$\text{cov}^{-1}(\theta_i, \theta_j) = -E\left[\frac{\partial^2 \ln L}{\partial \theta_i \partial \theta_j}\right] = -\frac{\partial^2 \ln L}{\partial \theta_i \partial \theta_j}\bigg|_{\vec{\theta}=\hat{\vec{\theta}}}. \tag{3.40}$$

In the large $N \to \infty$ limit the ML estimate produces a Gaussian distributed estimate for $\hat{\theta}$ that is unbiased, distributed about the true value and has as variance equal to the minimum variance bound. Therefore for large samples we can safely say the ML estimate is the "best" estimator (unbiased, consistent and efficient), however for small samples or certain niche cases which are not asymptotically well behaved this might not necessarily be true. Never-the-less, the properties above make the ML estimate extremely powerful and extremely popular.

For practical purposes, do not bother writing your own minimiser, there are already good libraries available for this. In the case of `iminuit` you probably do not need to bother writing out the likleihood function either because these are already provided as part of `iminuit.cost`.

Below (Code Block 3.2) I put a simple example, making use of `iminuit`, which shows an ML fit to some generated data.

```
import numpy as np

# generate some Gaussian distributed data
dset = np.random.normal(0, 1, size=1000)

# construct a model to fit
```

```
from scipy.stats import norm
def model(x, mu, sg):
    return norm.pdf(x, mu, sg)

from iminuit import Minuit
from iminuit.cost import UnbinnedNLL

# make the cost function, in this case the
# negative log likelihood
nll = UnbinnedNLL(dset, model)

# construct the minimisation object which
# needs to be passed starting values
mi = Minuit(nll, mu=0, sg=1)

# minimise it
mi.migrad()

# run the "Hesse" algorithm to compute
# covariance matrix (from inverse second derivs)
mi.hesse()

# print the fit result
print(mi)

# plot the fit result
import matplotlib.pyplot as plt
plt.hist(dset, bins='auto', range=(-4,4), density=True, alpha=0.6, label='
    Dataset')
x = np.linspace(-4,4,200)
plt.plot( x, norm.pdf(x,0,1), label='True Model')
plt.plot( x, model(x, *mi.values), label='Fitted Model')
plt.legend()
```

Code Block 3.2: A simple ML fit

The output that `iminuit` prints look like this. It displays not only the fitted values and uncertainties, but also the correlation matrix and various useful bits of information about the minimisation procedure, including:

- `FCN` - the function value at the minimum, in this case the negative log likelihood at the minimum
- `EDM` - a quantity that uses the internal estimate of the likelihood derivative to estimate the distance from the true minimum
- `Nfcn` - the number of times it had to call the likelihood function to find the minimum
- Then various other flags detailing whether the covariance matrix is postive definite and invertible, whether it thinks the minimum is valid, whether any parameters hit their limits *etc.*.

If you print this in a jupyter notebook it will display some nice colours as well. For more information I would recommend looking at the excellent tutorials on the `iminuit` docs page. The plot which this code produces is shown in Fig. 3.5.

```
------------------------------------------------------------------------
|                                Migrad                                |
------------------------------------------------------------------------
| FCN = 2895                    |              Nfcn = 42               |
| EDM = 1.42e-08 (Goal: 0.0002) |                                      |
------------------------------------------------------------------------
|          Valid Minimum        |        No Parameters at limit        |
------------------------------------------------------------------------
| Below EDM threshold (goal x 10) |           Below call limit           |
------------------------------------------------------------------------
|  Covariance   |     Hesse ok      | Accurate  |  Pos. def.  | Not forced |
------------------------------------------------------------------------
-----------------------------------------------------------------
|   | Name |   Value   | Hesse Err | Limit-  | Limit+  | Fixed |
-----------------------------------------------------------------
| 0 | mu   |   0.058   |   0.033   |         |         |       |
| 1 | sg   |   1.029   |   0.023   |         |         |       |
-----------------------------------------------------------------
--------------------------
|    |       mu        sg |
--------------------------
| mu |  0.00106 9.18e-08 |
| sg | 9.18e-08 0.000529 |
--------------------------
```

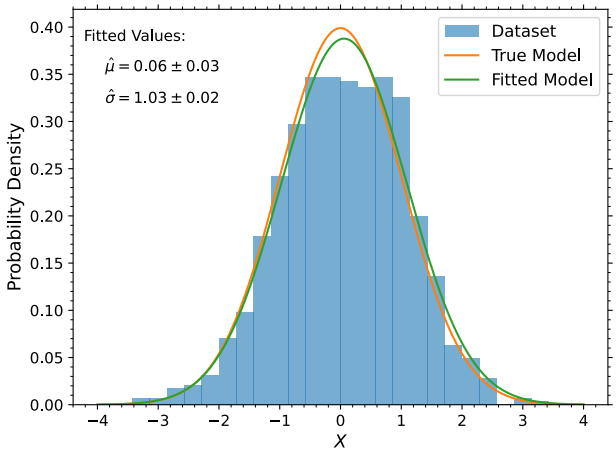


Figure 3.5: The plot produced by the simple ML fit code displayed in Code Block 3.2.

The profiled log likelihood

Quite often we will have a multi-dimensional likelihood but we are actually only really interested in a parameter estimate for one of the dimensions. Or sometimes we will have a multi-dimensional likelihood with several parameters which we would like to quote estimates and intervals for. Recall from above that the ML method will give parameter estimates which are Gaussian distributed (provided N is reasonably large). So when fitting an M dimensional likelihood I get estimates for my M parameters which are distributed according to an M-dimensional Gaussian. It is normally a bit impractical to actually provide this M-dimensional Gaussian in a scientific paper, although the best-practise nowadays is to

actually digitally publish your likelihood along with your paper. However, we still normally want to quote results in one dimension, *e.g.* I measure $\theta_1 = x \pm y$ and $\theta_2 = a \pm b$. So how do I make a plot like the one show in Fig. 3.4 for each dimension. Well, if my result is a multi-dimensional Gaussian then I can take a slice through it (conditional probability) that will be Gaussian or I can integrate over a dimension (marginal probability) that will also be Gaussian. If I want the slice that preserves the properties shown in Fig. 3.4, that I can read off 1σ, 2σ *etc.* at $-\Delta \ln L = 0.5$, 2 *etc.* then I can use the *profiled log likelihood*.

To compute this I effectively *scan* the log likelihood in the dimension of interest, at each point I fix the parameter to that value and then minimise $-\ln L$ with respect to all other parameters, then plot the curve. This is called the *profile likelihood* and you sometimes hear it referred to as *profiling* over the other dimensions. This curve then has the properties of a likelihood with just one free parameter so that the interval can be read off the plot as in Fig. 3.4. We will discuss why the values $-\Delta \ln L = 0.5$, 2, $n^2/2$ for standard deviations of 1σ, 2σ, $n\sigma$ take the values they do a bit later in Sec. 3.4.

3.3.3 Extended maximum likelihood

The ML method we have seen above makes the assumption that the total probability density is normalised. So for a random variable X distributed according to p.d.f. $f(X|\vec{\theta})$ with N observations, $\vec{X} = (X_1, X_2, \ldots, X_N)$, and M free parameters, $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_M)$, the normalised p.d.f. would ensure

$$\int f(X|\vec{\theta}) dX = 1, \tag{3.41}$$

and the likelihood would be

$$L(\vec{\theta}) = \prod_{i=1}^{N} f(X_i; \vec{\theta}). \tag{3.42}$$

However there are often times when we want to fit a dataset in which the total number of events in the dataset is also a random variable and we want to perform an ML fit to obtain not only the shape parameters of the distribution but also the overall scale (or yield). This is the case in counting experiments, *e.g.* I may expect that there are 5 buses in the next hour but due to random fluctuations I will not neccessarily see 5, it will be some random Poisson fluctuation on 5. It may be that I want to determine the average length and the average frequency of buses which pass me (i.e. the shape and the yield). Accordingly, we need to add an additional term to our likelihood to allow for this freedom, which is referred to as an "extended" term or an *extended maximum likelihood* (EML) fit.

Continuing from the logic above then N itself is a Poisson distributed random variable with some expectation, ν, so that $N \sim \text{Pois}(N|\nu)$. In this case the likelihood instead looks like

$$L(\nu; \vec{\theta}) = \frac{\nu^N e^{-\nu}}{N!} \prod_{i=1}^{N} f(X_i; \vec{\theta}), \tag{3.43}$$

$$= \frac{e^{-\nu}}{N!} \prod_{i=1}^{N} \nu f(X_i; \vec{\theta}). \tag{3.44}$$

This is known as the extended maximum likelihood. A simple example, again exploiting the cost functions in the `iminuit` pacakge, is shown below, which produces the plot in Fig. 3.6.

```python
import numpy as np

# generate some Gaussian distributed data
# with a poisson random number of events
N = np.random.poisson(1000)
dset = np.random.normal(0, 1, size=N)

# construct a model to fit
from scipy.stats import norm
def model(x, mu, sg):
    return norm.pdf(x, mu, sg)

def density(x, N, mu, sg):
    return N, N*norm.pdf(x,mu,sg)

from iminuit import Minuit
from iminuit.cost import ExtendedUnbinnedNLL

# make the cost function, in this case the
# negative log likelihood
nll = ExtendedUnbinnedNLL(dset, density)

# construct the minimisation object which
# needs to be passed starting values
mi = Minuit(nll, N=1000, mu=0, sg=1)

# minimise it
mi.migrad()

# run the "Hesse" algorithm to compute
# covariance matrix (from inverse second derivs)
mi.hesse()

# print the fit result
print(mi)

# plot the fit result
import matplotlib.pyplot as plt
nh, xe, _ = plt.hist(dset, bins=40, range=(-4,4), alpha=0.6, label='Dataset')
# need to track the bin width to get the normalisation right
bw = xe[1]-xe[0]
x = np.linspace(-4,4,200)
plt.plot( x, density(x, *mi.values)[1]*bw, label='Fitted Model')
plt.legend()
```

Code Block 3.3: A simple example of an extended maximum likelihood fit

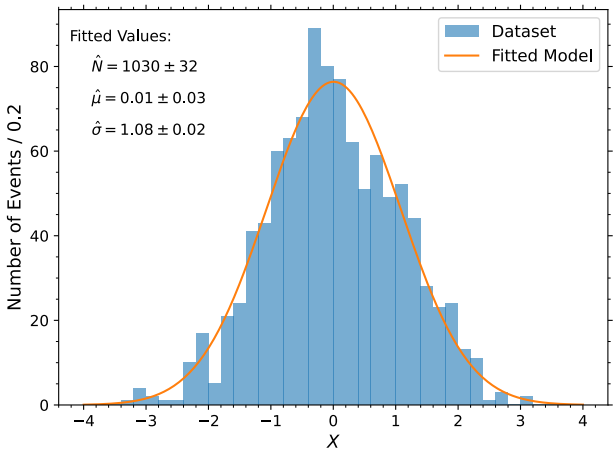


Figure 3.6: The plot produced by the simple EML fit code displayed in Code Block 3.3. Notice, in constrast to Fig. 3.5, that I am now not plotting the probability density on the y-axis but instead the actual number of events per bin.

3.3.4 Binned maximum likelihood

Maximum likelihood fits are extremely powerful and very widely used. However, when the sample size is very large and the model very complex they can be rather expensive to perform and become slow. Imagine if we have 10M data points and a model with 10 free parameters, then in order to compute the log likelihood for any given set of parameter values we have to sum 10M evaluations of the model. We then may need many hundreds or thousands of calls to find the minimum for a large number of parameters.

A way around this is to use binned fits, when the sample size is large. These are much more computationally efficient and also tend to be numerically more stable. In this case we split the dataset up into N bins and then our likelihood simply becomes the product of Poisson terms in each bin,

$$L(\vec{\theta}) = \prod_{b=1}^{N} \frac{f_b(\vec{\theta})_b^n e^{-f_b(\vec{\theta})}}{n_b!}, \tag{3.45}$$

where n_b is the number of events observed in the bin and f_b, which is the Poisson expectation, is the integral of the probability density over the bin

$$f_b(\vec{\theta}) = \int_{bl}^{bh} f(X;\theta) dX, \tag{3.46}$$

where bl and bh signifiy the lower and upper edges of the bin boundaries. Of course if we analytically know the form of the c.d.f. then this integral is very straightforward to compute because it is simply the difference between the c.d.f. evaluated at the upper and lower bin edges,

$$f_b(\vec{\theta}) = F(bh;\vec{\theta}) - F(bl;\vec{\theta}). \tag{3.47}$$

In the case of our 10M size dataset we could split it into 500 bins and then for a single call to the log likelihood we only need to compute the sum of 500 terms rather than the sum of 10M. Even if you don't know the analytical form of the c.d.f. it can often be quicker to perform a binned fit with a numerical integration of the p.d.f..

It is worth noting that if the bins are fine enough, *i.e.* the p.d.f. is not varying rapidly over the bin, then a reasonable approximation of the c.d.f. is the p.d.f. evaluated at the centre of the bin, multiplied by the bin width. This can be a considerable speed up if the c.d.f. is expensive to compute but this is only an approximation and will lead to a bias. Please be aware that some fitting packages make this assumption when asked to perform binned fits so please be careful.

In the snippet below I show an example of a binned ML fit and give a speed comparison to an unbinned one, again exploiting the built-in functionality of `iminuit`.

```python
import numpy as np
from scipy.stats import norm
from iminuit.cost import UnbinnedNLL, BinnedNLL
from iminuit import Minuit
from time import process_time

# generate large data set of 1M events
dset = np.random.normal(size=1000000)

# bin it into 200 bins
nh, xe = np.histogram(dset, bins=200, range=(-4,4))

# set up unbinned fit
def pdf(x, mu, sg):
    return norm.pdf(x,mu,sg)

unbinned_nll = UnbinnedNLL(dset, pdf)
unbinned_mi  = Minuit(unbinned_nll, mu=0, sg=1)

# set up binned fit
def cdf(x, mu, sg):
    return norm.cdf(x,mu,sg)

binned_nll = BinnedNLL(nh, xe, cdf)
binned_mi  = Minuit(binned_nll, mu=0, sg=1)

# time the minimisations
unbinned_start = process_time()
unbinned_mi.migrad()
unbinned_mi.hesse()
unbinned_stop = process_time()

binned_start = process_time()
binned_mi.migrad()
binned_mi.hesse()
binned_stop = process_time()

print( "+++ UNBINNED +++" )
print( unbinned_mi.params )
print( f"Took {unbinned_stop-unbinned_start:5.3f} seconds")

print( "++++ BINNED ++++" )
print( binned_mi.params )
print( f"Took {binned_stop-binned_start:5.3f} seconds")
```

Code Block 3.4: Demonstration and comparison of the performance between a simple binned and unbinned fit.

The output of this snippet gives:

```
+++ UNBINNED +++
------------------------------------------------------------------
|   | Name |   Value    | Hesse Err | Limit-  | Limit+  | Fixed |
------------------------------------------------------------------
| 0 | mu   |  -0.4e-3   |  1.0e-3   |         |         |       |
| 1 | sg   | 999.2e-3   |  0.7e-3   |         |         |       |
------------------------------------------------------------------
Took 0.612 seconds
++++ BINNED ++++
------------------------------------------------------------------
|   | Name |   Value    | Hesse Err | Limit-  | Limit+  | Fixed |
------------------------------------------------------------------
| 0 | mu   |  -0.4e-3   |  1.0e-3   |         |         |       |
| 1 | sg   | 998.6e-3   |  0.7e-3   |         |         |       |
------------------------------------------------------------------
Took 0.002 seconds
```

3.4 Least-squares method

The method of least squares is applicable when we want to fit a curve to some (x, y) values of data. It is essentially a regression, or more specifically non-linear regression, which is of central important in machine learning algorithms. You will cover in more detail the concepts of linear and logistic regression in the M1: Machine Learning course so for this course I will just cover the least-squares method itself.

The principal is based on minimising the (squared) distance between the model's prediction and the observed values. For example if we have a collection of events containing (x_i, y_i) pairs and we want to determine the parameters of some curve which fits them, $f(x) = mx+c$, then we simply find the values $\hat{m}$ and $\hat{c}$ which minimise

$$\chi^2 = \sum_{i=1}^{N} (y_i^{\text{obs}} - y_i^{\text{pred}})^2 = \sum_{i=1}^{N} (y_i - mx_i - c)^2. \tag{3.48}$$

For the straight line above (linear regression) this can often be done analytically, however in the more general least squares approach is done numerically. In the above case I have assumed that the y_i values in our dataset are known with no error (even though they could include random noise). In many cases the y_i values will have associated uncertainties σ_{y_i} and thus the least-squares is modified to incoprate a weighting of the deviation proportional to the variance, such that

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i))^2}{\sigma_{y_i}^2}. \tag{3.49}$$

You may notice I have written the object to be minimised here as χ^2 which is because this quantity is called the "chi-squared" and these types of fits are sometimes referred to as *chi-squuared fits*. We have already seen how this sum will be distributed for a given number of degrees of freedom when we discussed the χ^2 distribution above in Sec. 2.4.7. I will pick up that discussion a bit more in a moment, see Sec. 3.4.

Two examples of least-squares fits to some noisily generated data, once again exploiting `iminuit`, are shown in the snippet below and the plots in Fig. 3.7.

```python
import numpy as np

# define some models
# y = mx+c
# y = mx**2 + c
def p1(x, m, c):
    return m*x + c

def p2(x, m, c):
    return m*x**2 + c

# generate some random x values
x_i = np.random.uniform(0, 1, size=200)

# calculate some y values according to
# true model of y = x**2 + 2
y_i = p2(x_i, 1, 2)

# assume no y errs so set them to 1
y_e = np.ones_like(y_i)

# now add some noise by moving the y values around
# a bit
y_i = np.random.normal(y_i, 0.2*x_i**5 + 0.05)

# now do some fits
from iminuit import Minuit
from iminuit.cost import LeastSquares
cst = [ LeastSquares(x_i, y_i, y_e, p) for p in [p1,p2] ]
mis = [ Minuit(c, m=1, c=1) for c in cst ]
_   = [ mi.migrad() for mi in mis ]

# this is a second case where I will now assume the x to be uniformly
    distributed
# and have error bars
x_j = np.linspace(0, 1, 20)
y_j = p2(x_j, 1, 2)
y_e = 0.2*x_j**5 + 0.1
y_j = np.random.normal(y_j, y_e)
cst2 = [ LeastSquares(x_j, y_j, y_e, p) for p in [p1,p2] ]
mis2 = [ Minuit(c, m=1, c=1) for c in cst2 ]
_    = [ mi.migrad() for mi in mis2 ]
```

Binned chi-squared fits

A rather nice use case of least-squares fits is for binned fits of data. Imagine I want to fit a distribution of events $X_i = (X_1, X_2, \ldots, X_n)$. I can bin these into a histogram where I know that, providing there are a large number of events in each bin, then the y-value of the bin will simply be the number of events in the bin, and the y-error of the bin will be the square root of that number (because it follows a Poisson distribution). We will see in a moment that the least-squares method is actually assuming the distribution in each bin to be Gaussian but if the number of events is large then the Poisson tends to a Gaussian so this is a reasonable approximation. The other thing we are going to see in a moment is that the least-squares method is infact a specific case of the likelihood method with Gaussian errors.

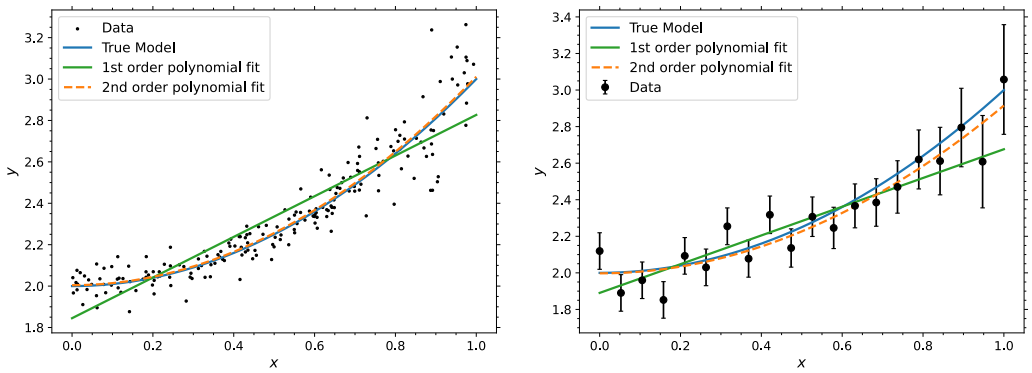


Figure 3.7: Left: A demonstration of a χ^2 fit to some randomly generated x and y values without uncertainties. Right: A demonstration of a χ^2 fit to some uniformly distributed data in x with errors on the y-values.

Relationship to the log likelihood

Following on from the examples above, imagine that I have a collection of x_i values with corresponding y_i values in data which each come with an uncertainy σ_i. If I assume these $y_i \pm \sigma_i$ values are Gaussian distributed then I can write down the likliehood as a product of Gaussian terms,

$$L(\vec{\theta}) = \prod_{i=1}^{N} \text{Gaus}(y_i - f(x_i), \sigma_i), \tag{3.50}$$

$$= \prod_{i=1} \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left[-\frac{\left(y_i - f(x_i; \vec{\theta})\right)^2}{2\sigma_i^2} \right]. \tag{3.51}$$

Now let's investigate the logarithm of the likelihood which is

$$\ln L(\vec{\theta}) = \sum_{i=1}^{N} -\frac{1}{2} \frac{(y_i - f(x_i))^2}{\sigma_i^2} + \sum_{i=1}^{N} \ln\left(\frac{1}{\sigma_i \sqrt{2\pi}} \right), \quad = \sum_{i=1}^{N} -\frac{1}{2} \frac{(y_i - f(x_i))^2}{\sigma_i^2} + C, \tag{3.52}$$

where the second term has no dependence on the parameters we are estiamting, θ, and therefore is a just a constant term which offsets the likelihood. What we see is that the first term is proportional to the χ^2 so that we can write

$$\chi^2 = -2 \ln L + C. \tag{3.53}$$

When we make parameter estimates we don't actually care what value the minimum of the negative log likelihood or chi-squared actually take, we just want to know the parameter estimates at the minimum. This relationship between the difference from the minimum in the χ^2 and the difference from the minimum in negative log likelihood is what allows us to read of uncertainties from this difference, as we will now see. Just to finish this section, we can drop the constant term if we only focus on the difference from the minimum so that,

$$\Delta\chi^2 = -2\Delta \ln L. \tag{3.54}$$

The above equation is important (it shows how the least-squares-fit is a special case of a maximum likelihood fit) but you have to remember that this relationship *only holds* for independent Gaussian measurements. When the sample size is high then we know that a binned distribution of any kind will have Gaussian uncertinaties in each bin (providing the binning is wide enough to have a significant number of events in each bin).

Wilks' theorem

The reason that the log-likelihood difference asymptotically approaches the χ^2 difference, (3.54), is known as Wilk's theorem. It states that as $N \to \infty$ then the test-statistic which is twice the negative log likelihood ratio (*i.e.* the log likelihood difference) approaches the χ^2 distribution. This is an example of a hypothesis test (which we cover in more detail later in Sec. 3.8) where the null hypothesis is the value of the parameter at the best fit and the alternate hypothesis is the value of the parameter where we read off the log likelihood difference.

Relationship to chi-squared distribution

We have seen that the χ^2, given in (3.49), is the sum of squared residuals weighted by the variance. The distribution that the χ^2 itself follows is, *quelle surprise*, given by the χ^2 distribution where the number of degrees of freedom k is equal to the number of observations in the χ^2 sum minus the number of free parameters,

$$\chi^2 \sim p(\chi^2; k) = p(\chi^2; n^{\text{obs}} - n^{\text{pars}}). \tag{3.55}$$

Recall, from Sec. 2.4.7, that the expectation value (mean) for a χ^2 distribution is given by the number of degrees of freedom k. Therefore we expect, on average that the χ^2 per degree of freedom, sometimes written χ^2/d.o.f, should be 1. If a least-squares fit (or indeed a maximum likelihood fit) returns a value significantly far from this you should be suspicious that the fit is not very good, so either the model being fitted is not consistent with the data, or seems to fit the data "too-well" (*e.g.* fitting an 8^{th} order polynomail to data generated from a straight line). Of course this relies on the assumption that the χ^2 follows a Gaussian distribution, which is only the case when k is very large. For lower values of k a better approximation is to compute $\sqrt{2\chi^2}$ which converges to a Gaussian with mean $\sqrt{2k-1}$.

Something else to recall from Sec. 2.4.7 is that one of the nice properties of the c.d.f. of the χ^2 distribution is that it allows us to read off the fraction contained within a given standard deviation of a Gaussian distribution. It is this property which allows us to read the uncertainties from the negative log likelihood curve (and also the χ^2 curve) shown in Fig. 3.4, as the $\Delta\chi^2$ and $-2\Delta \ln L$ will follow a χ^2 distribution with number of degrees of freedom k equal to the number of free parameters in the fit.

Let's see an example in the snippet below (Code Block 3.5) where I will fit a Gaussian distribution to some sample using both a least-squares fit and a log likelihood fit. This has two free parameters, the mean and width of the Gaussian, where I will assume I don't care about the overall normalisation. I will exploit `iminuit`'s inbuilt ability to produce likelihood profiles which will give me the distribution across the parameter space of $\hat{\mu}$ and $\hat{\sigma}$ for the $\Delta\chi^2$ (in the least-squares fit case) and the $-\Delta \ln L$ (in the maximum likelihood case). If we

want to find where the 1σ, 2σ, $N\sigma$ error band is when we have done a profiled likelihood in 1D then we can simply read off where $\Delta\chi^2 = -2\Delta \ln L = 1,\ 4,\ N^2$. Finding the equivalent contour in 2D requires us to find the contour where $\Delta\chi^2 = -2\Delta \ln L = 2.3,\ 6.2,\ etc.$. First let me show some code showing how these numbers are computed:

```
from scipy.stats import chi2
from tabulate import tabulate

print_rows = []
for ndof in [1,2,3]:
    for nsigma in [1,2,3]:
        # fraction of distribution inside nsigma
        # is given by the cdf of the chisq distribution
        # with d.o.f at value nsigma**2
        frac_inside = chi2.cdf(nsigma**2, ndof)

        # equivalent value expected for twice the delta
        # log likelihood (or the delta chisq) is the ppf
        # of the chisq distribution with d.o.f after
        # evaluating the fraction for 1 d.o.f
        n2dll_val   = chi2.ppf( chi2.cdf(nsigma**2,1), ndof)

        print_rows.append( [nsigma, ndof, frac_inside, n2dll_val] )

print(tabulate(print_rows, headers=['nSigma','nPar', 'fInside','n2DLL'],
    floatfmt=['.0f','.0f','5.3f','5.2f']))
```

which produces the following table:

σ	n_{dof}	Fraction	$-2\Delta \ln L$
1	1	0.683	1.00
2	1	0.954	4.00
3	1	0.997	9.00
1	2	0.393	2.30
2	2	0.865	6.18
3	2	0.989	11.83
1	3	0.199	3.53
2	3	0.739	8.02
3	3	0.971	14.16

Now the code to show an example least-squares and maximum likelihood fit with profiles

```
import numpy as np
from scipy.stats import norm

# generate data
N = np.random.poisson(5000)
dset = np.random.normal(0, 1, size=N)

# model
def density(x, N, mu, sg):
    return N, N*norm.pdf(x, mu, sg)

from iminuit import Minuit
from iminuit import cost

# EML fit
```

```
nll = cost.ExtendedUnbinnedNLL(dset, density)
mi_nll = Minuit(nll, N=1000, mu=0, sg=1)
mi_nll.migrad()
mi_nll.hesse()
print(mi_nll)

# chisq fit
nh, xe = np.histogram(dset, bins=30, range=(-3,3))
cx = 0.5*(xe[:-1]+xe[1:])
bw = xe[1:] - xe[:-1]
def model(x, N, mu, sg):
    return N*bw[0]*norm.pdf(x,mu,sg)
ne = nh**0.5
ne[ne==0] = 1
chisq = cost.LeastSquares( cx, nh, ne, model )
mi_chisq = Minuit( chisq, N=1000, mu=0, sg=1)
mi_chisq.migrad()
mi_chisq.hesse()
print(mi_chisq)

import matplotlib.pyplot as plt
# make some plots with minuit
for mi in [mi_chisq, mi_nll]:
    mi.draw_mnprofile('mu')
    mi.draw_mnprofile('sg')
    mi.draw_mncontour('mu','sg', size=100, interpolated=True)
    mi.draw_mnmatrix(cl=[1,2,3])
```

Code Block 3.5: Demonstration of a simple chi-squared and maximum likelihood fit with the profiles produced. Plots shown in Fig. 3.8 which I have adjusted the style of slightly.

`iminuit` can also make super nice "matrix" style plots which look the one shown below, Fig. 3.9

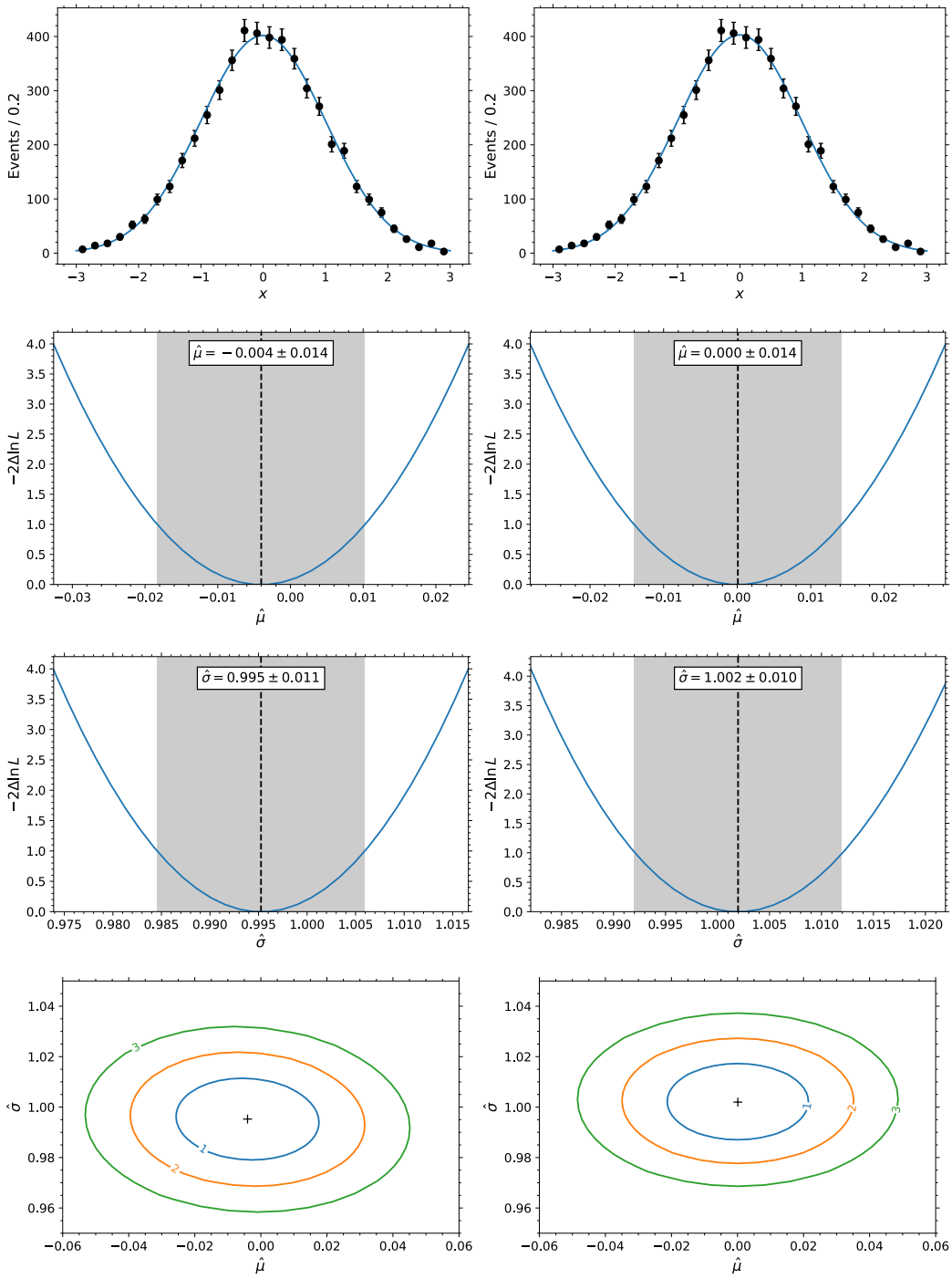


Figure 3.8: Plots of χ^2 and ML fit profiles for the simple example given in Code Block 3.5. The χ^2 fit is on the left, the ML fit on the right. Top row shows the fit reults. Second and third rows show the 1D profiled scans for $\hat{\mu}$ and $\hat{\sigma}$, respectively. Fourth row shows the 2D profiled scans.

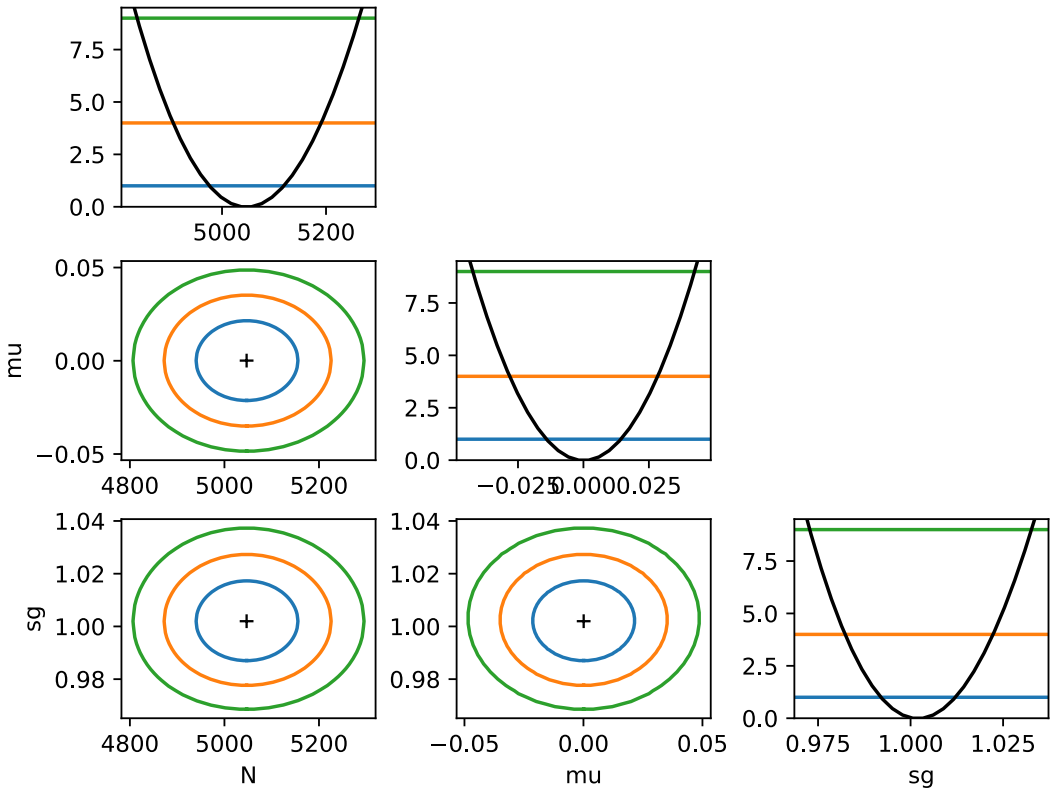


Figure 3.9: A demonstration of the "matrix" style plot that `iminuit` makes, for the example given above in Code Block 3.5.

3.5 Method of Moments

We have seen that both the maximum likelihood and least squares methods provide estimators with close to optimal properties. However, in some cases they can be very hard to implement. This is particularly relevant when they might be very slow or very complex (which is often tantamount to the same thing). An alternative approach is the method of moments (MOM), which is incredibly straight forward to implement and, very fast and normally fairly accurate, although variances of estimates from MOM tend to be larger than ML.

Method of moments estimation is based on the law of large numbers,

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \to \mu \quad \text{as} \quad n \to \infty. \tag{3.56}$$

Let's first consider the case of attempting to determine one free parameter of a model $f(X;\theta)$ given a bunch of observations $X_i = \{X_1, X_2, \ldots, X_n\}$. The true mean of the X_i, μ, is some function, $g(\theta)$, of θ (in the case where θ is the mean then this function is simply $\mu = \theta$).

If we write out μ as the expectation value then we can express $g(\theta)$ as

$$g(\theta) = \mu = \int_{-\infty}^{\infty} X f(X;\theta) dX. \tag{3.57}$$

We have seen that the law of large numbers states that the sample mean

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \to \mu \quad \text{as} \quad n \to \infty, \tag{3.58}$$

and that the sample mean $\bar{X}$ provides a good estimate of the true mean for large N. This means it also provides a good estimate of $g(\theta)$:

$$\bar{X} \approx \mu = g(\theta). \tag{3.59}$$

If we can find the inverse of the function then we can use it to provide an estimate for θ itself:

$$\hat{\theta} = g^{-1}(\bar{X}). \tag{3.60}$$

If we now extend this to m unknown parameters, $\vec{\theta} = \{\theta_1, \theta_2, \ldots, \theta_m\}$, for a model $f(X;\vec{\theta})$. We now have m unknowns but can express the first m moments with functions $\vec{g}(\vec{\theta}) = \{g_1(\vec{\theta}), g_2(\vec{\theta}), \ldots, g_m(\vec{\theta})\}$. Once again, the law of large numbers tell us that for large N we can approximate the m^{th} moment with the m^{th} sample moment:

$$\hat{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} X_i^j \to \mu_1 \quad \text{as} \quad N \to \infty. \tag{3.61}$$

The method of moments then gives us m simultaneous equations to solve which provide

estimates of $\vec{\theta}$:

$$\hat{\mu}_1 = g_1(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_m), \tag{3.62}$$
$$\hat{\mu}_2 = g_2(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_m), \tag{3.63}$$
$$\vdots \tag{3.64}$$
$$\hat{\mu}_m = g_m(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_m). \tag{3.65}$$
$$\tag{3.66}$$

Procedure for method of moments:

1. For m unknowns of probability model $f(X; \vec{\theta}$ find the first m moments of the distribution, as in (2.50), to provide the m functions $\vec{g}(\vec{\theta})$,
2. Solve these equations such that each θ is now expressed as a function, $\vec{h}(\vec{\mu})$, of the moments, *i.e.* $\theta_1 = h_1(\mu_1, \mu_2, \ldots, \mu_m)$, $\theta_2 = h_2(\mu_1, \mu_2, \ldots, \mu_m)$, *etc.*
3. Plugin the sample estimates for the moments $\vec{\mu}$ in order to produce estimates of $\vec{\theta}$.

We'll go through a couple of examples in the lectures.

3.5.1 Uncertainties from method of moments estimates

Of course just obtaining point-estimates is ok for the MoM but we really want to also be able to provide estimates of the uncertainties on those estimates. An estimate of the covariance of the moment estimates is given by

$$\widehat{\mathrm{cov}}(\hat{\mu}_\alpha, \hat{\mu}_\beta) = \frac{1}{n(n-1)} \sum_{i=1}^{N} (X_i^\alpha - \bar{X}_i)(X_i^\beta - \bar{X}_i), \tag{3.67}$$

which we can the propagate to the covariance on the parameters, θ, using normal error propagation:

$$\mathrm{cov}(\theta_i, \theta_j) = \sum_{k,l} \frac{\partial \theta_i}{\partial \mu_k} \frac{\partial \theta_j}{\partial \mu_l} \mathrm{cov}(\mu_k, \mu_l). \tag{3.68}$$