3.6 Goodness-of-Fit Tests

Our discussion of estimation, and in particular the two most common methods of estimation (maximum likelihood and least squares), allows us to now discuss another important topic. For these types of estimation we are predominantly talking about *fitting* datasets. In other words finding the best estimate for the values of parameters of a model with regard to some dataset, and not only best estimates for the parameters themselves but also estimates for how good our estimates our (*i.e.* the uncertainty). But what we would often like to know is "how good was my fit in the first place?" The answer to this type of question is a goodness-of-fit test.

The χ^2 value gives us the perfect proxy to test this, because we know from the properties of the χ^2 distribution that on average we expect a χ^2/d.o.f ≈ 1, because the expectation of the χ^2 distribution for k degrees of freedom is k. The χ^2 in this case is an example of a *statistic*, sometimes referred to as a *test statistic*. A *test statistic* in the general sense can really be any property of the data and the model, it could be the value of the estimate itself, but for a goodness-of-fit test the *test statistic* should somehow quantify the agreement between the data and the model. If we know the probability distribution of the test-statistic then we can even compute the probability that we got the value we did, which in turn gives us information on how likely we were to get that value and hence what we think the quality of the fit was.

The question I want to answer for a goodness of fit test is how well does my data, X_0, agree with my hypothesis, H_0, where the hypothesis in this case is the model with the fitted parameters that I am making a comparison of. If I have some test statistic, T, which I know is distributed according to some probability, $P(T)$, what I then want to know is the probability that I would have got this fit or a worse one, *i.e.*

$$P_{\text{gof}} = \int_{T_0}^{\infty} P(T|H_0). \tag{3.69}$$

3.6.1 The χ^2 test

In the specific case of a χ^2 test, this is sometimes known as the χ^2-probability, and can clearly be computed from $1 - F(\chi^2)$ (*i.e.* one minus the c.d.f. of the χ^2 distribution with the appropriate degree of freedom). What it specifically means is the probability that a function that does describe the data gives a value of the χ^2 as large or larger than the one you find. This allows me to convert from my test-statistic into a probability, which is also known as a p-value. If my p-value comes out incredibly small, I should be suspicous, it means that my model and data do not agree well *i.e.* the fit is bad. If my p-value comes out incredibly high, I should be suspicious, it means that my model and data agree too well *i.e.* the fit may have too many free parameters. It's worth noting that, as we will see in the example below, the χ^2 test is much better at spotting the former type of inconsistency than the latter.

So let's see how this works for the χ^2, which is particulary useful because we know what its probability distribution will be, and then the same methods can be extended to any other test statistic. It is worth noting that computing the χ^2 to assess the goodness-of-fit can be completely independent from the fit itself. It doesn't have to have been a χ^2 fit, I am just using the χ^2 value to ascertain the *agreement* between the data I have and some model

value, how I obtained that model and its parameters is irrelevent. There is a little bit of a subtlety here that we could have a debate about. Namely, if I am making a comparison between some given straight line and ten data points then the number of degrees of freedom for my χ^2 test should be ten. However if I have fitted that line, using a χ^2 fit, I have now minimised the χ^2 thus making it smaller, so I should decrease the number of degrees of freedom to $N - m$, in this case 8.

Below, Fig. 3.10, I show some data distributed according to a 2nd order polynomial with some random noise added in. There are $N = 10$ points in this dataset. I lay over the true model and then also a leat-squares fit using a 1st, 2nd and 8th order polynomial. In Table 3.1 I show what the χ^2 values and then associated p-values are in each case. You can see that the poor quality of the straight line fit shows us as an extremely low p-value ($p = 1^{-9}$), however the overly good quality of the 8th order polynomial (which almost goes through every point) gives a high but not extrme p-value ($p = 86\%$). Of course if I increase the polynomial order such that it has as many free parameters as there are points in the dataset, 9th order, then I can still compute the χ^2 but the p-value is undefined because $k = 0$. In this case the χ^2 should be 0 (or numerically close to) because the polynomial has the freedom to go through every point.

m	$n_{\rm dof}$	χ^2	χ^2/d.o.f	p-value
2	8	58.47	7.31	9.2×10^{-10}
3	7	4.95	0.71	0.67
9	1	0.032	0.03	0.86
10	0	2.73×10^{-20}	∞	undefined

Table 3.1: The number of free parameters, m, degrees of freedom, $n_{\rm dof}$, χ^2, χ^2/d.o.f and p-values for the fits shown in Fig. 3.10.

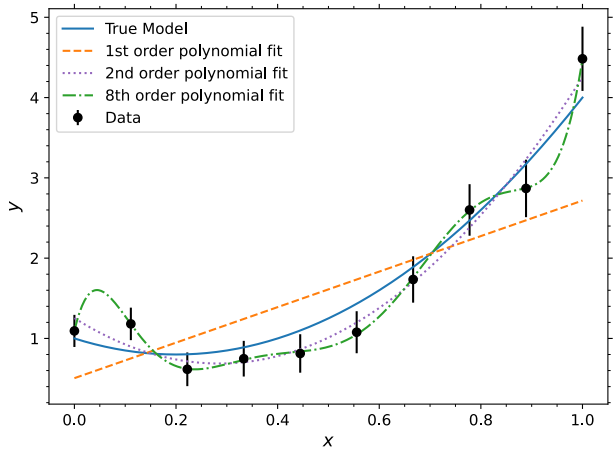

Figure 3.10: Some examples of fitting different order polynomials. The goodness-of-fit tests for these fits are shown in Tab. 3.1.

The χ^2 test is incredibly useful and you should use it when you have fitted some data to check if the fitted model and data are compatible. It is often asked for by reviewers. Something that is worth bearing in mind however, is that the χ^2 test is not necessarily very powerful, it only assesses compatibility between the data and the hypothesis. It is also very dependent on the binning that is used and it is particularly bad if you are trying to assess

the compatibility of a model which may only contain a discrepancy in one bin. A good example of this is for resonance searches in particle physics, for example the Higgs boson discovery, I show a dummy plot for this in Fig. 3.11. In this case we have a huge slowly falling background upon top of which we are searching for a tiny peaking signal. If I simply do a χ^2 goodness-of-fit test between the data and a fit of the "background only" we may still be looking for the Higgs. In my dummy example this gives a p-value of $p \approx 0.2$ so the fit isn't perfect but it by no means is strong enough evidence for us to say there has to be something other than just background here. If I use a more powerful test, a comparison between the hypothesis of "background only" and "signal plus background" then I get evidence at the level of 5σ, a p-value of $p \approx 1 \times 10^{-8}$. For much better statistical power we should compare two hypotheses. This is a topic we will discuss a bit further down in Sec. 3.8 where we will see how we can maximise the statistical power.

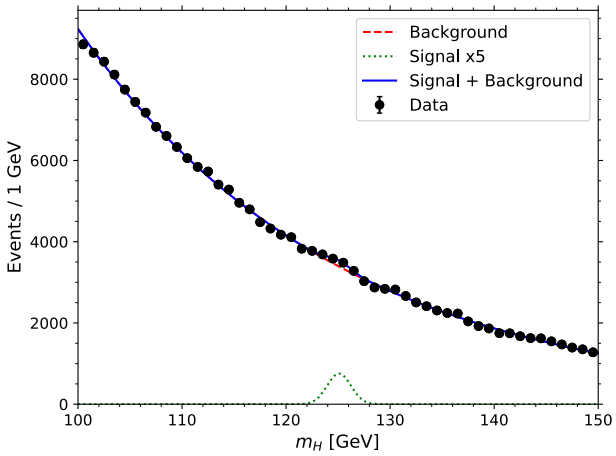

Figure 3.11: A dummy example demonstration why the χ^2-probability is not necessarily a very powerful goodness-of-fit test.

Residual and "pull" distributions underneath fit plots

A useful way to visualise the quality of a fit is to make a plot of the *residual* difference between the data points and the fitted model. When doing this you should of course keep the errors on the data points to see how well they agree with the model. A perhaps even nicer visualisation is to make the residual weighted by the error which is sometimes colloquially referred to as the *pull* and is more specifically called the *student's t-distribution*,

$$\text{pull} = \frac{n_{\text{obs}} - n_{\text{pred}}}{\sigma_{\text{obs}}}. \tag{3.70}$$

These are often displayed underneath fit plots to demonstrate the quality of the fit. I show some examples in Fig. 3.12. The error on the pull should just be unity for each point (because it is already weighted to the error). For this reason you sometimes see the pull just plotted as a histogram (because all of the errors are unity). If the fit quality is good we expect the residuals or pulls to be standard normally distributed and to that end you sometimes see a little plot to the side of the pull or residual which shows a histogram of them. What you should notice about the *pull* is that if you square each point and sum them, that is the χ^2.

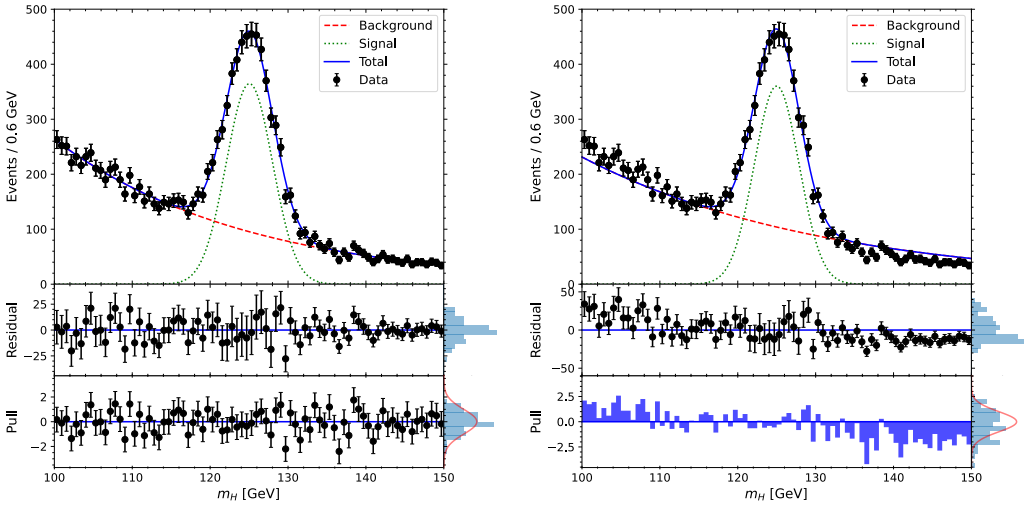

Figure 3.12: An example of plotting the residual or pull beneath a fit to demonstrate its quality. The left shows a good fit, the right shows a poor fit.

3.6.2 Kolmogorov-Smirnoff Test

Another goodness-of-fit metric you sometimes see and I will mention briefly is the so-called *Kolmogorov-Smirnoff* test or *KS test*. It is an unbinned test so can be an alternative to the χ^2 test when the number of events is small. It also can be used to assess the compatibility between two datasets, *i.e.* the probability that these two datasets came from the same parent distribution. This is useful in the machine learning world when we may want to ascertain, for example, the compatibility between a training set and a test set. The *KS score* is determined by finding the maximum deviation between the two distributions c.d.f.s and multiplying it by the square root of the sample size

$$p_{\text{KS}} = \max(F(X_1) - F(X_2))\sqrt{N}. \tag{3.71}$$

A little example is shown in the snippet below with an illustraion in Fig. 3.13.

```
import numpy as np
from scipy.stats import norm, kstest
from tabulate import tabulate

np.random.seed(210187)
N = 200
dset1 = norm.rvs( size=N )
dset2 = norm.rvs( loc=0.5, size=N )

ks1 = kstest(dset1, dset2)
ks2 = kstest(dset1, 'norm')

print_rows = []
for name, ks in zip(['D1 vs D2', 'D1 vs Norm'], [ks1, ks2] ):
    print_rows.append( [name, ks.statistic, ks.pvalue, ks.statistic_location ]
     )

print(
    tabulate(
```

```
        print_rows,
        headers = ['Name', 'KS-score', 'p-value', 'Location']
        )
    )
```

which produces the following output:

```
Name           KS-score      p-value    Location
-----------  ----------  -----------  ----------
D1 vs D2       0.255      4.02481e-06    0.132544
D1 vs Norm     0.0500365  0.679394      -0.358556
```

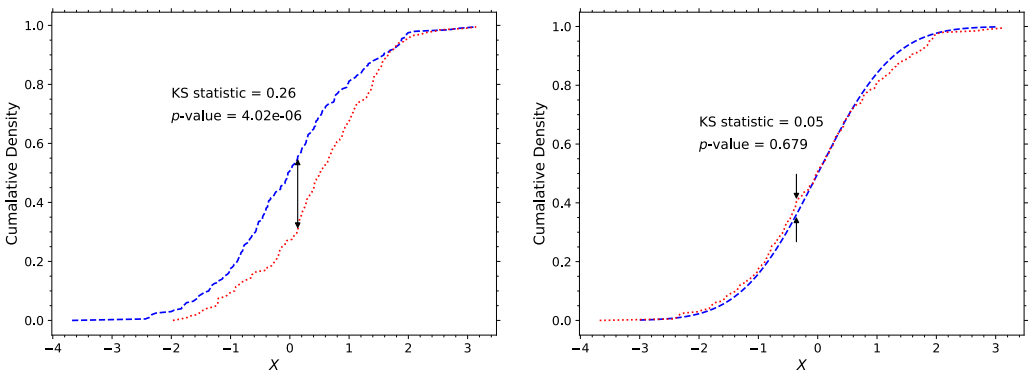

Figure 3.13: An illustration of the *KS* test which produces a *p*-value for compatibilty between two datasets which are incompatible (left) and between a dataset and distirbution which are compatible (right). This makes use of `scipy.stats.kstest`.

3.7 Confidence intervals

I want to formalise our discussion on confidence intervals a bit. I have already given quite a lot of this away in previous sections when discussing ML and least-squares estimates. We are scientists, we do not want to simply quote *point estimates i.e.* a central value for a parameter, we must also quote an interval, which is some region we believe the true value to be within, with some probability. The formal statement we are making when we quote a classical frequentist *confidence interval* is that we believe the *true* value of the parameter, θ_0, lies within some interval, $\theta_a < \theta_0 < \theta_b$, with some probability β. Typically we quote the region which contains either 68.3% or 95.4%, corresponding to 1σ and 2σ standard deviations of a normal distribution[5].

We have already seen the different ways we can end up quoting uncertiainties on fit parameters. The first was in Sec. 3.3, specifically (3.31), in which we saw that we can obtain an estimate of the variance from the inverse double differential of the log likelihood. The second was in Sec. 3.3.2 in which we saw we can obtain an estimate by computing the change in the log likelihood from the minimum. We then saw how this is related to specific values in Sec. 3.4 from the χ^2 distribution.

Below, Fig. 3.14, I put some plots and tables which attempt to demonstrate what these different probabilites mean and where they lie. I also put some plots demonstrating differnt correlations for 2D Gaussians in Fig. 3.15. Much of this is repeated information (suggesting

[5]We are of course true to choose any range we desire and different fields and have different conventions.

I already need a course restructure). The convention of quoting things in *standard deviations* is sometimes called the Z-score because any normal distribution can be translated to the standard normal distribution via $Z = (X - \mu)/\sigma$, it is also sometimes called the *significance*. One subtle point that we will return to a bit later is whether the conversion from p-value, *i.e.* the fraction contained in the tail, should be one-sided or two-sided.

Here are some points to remember regarding classical statistics and confidence intervals:

- In classical (frequentist) statistics the true value of a parameter, θ_0, is considered fixed (it does not have a probability distribution), like a physical constant *e.g.* the speed of light
- When we quote a frequentist *confidence interval* at confidence level $\beta\%$ we mean the true value will lie within in the interal $\beta\%$ of the time
- If the true value really does like within the interval $\beta\%$ of the time then the interval is said to have *good coverage* or simply the interval *covers*
- *Overcoverage* and *undercoverage* do commonly occur for confidence intervals so we may need to check or correct for it
- It is somewhat dependent on the problem but in general *undercoverage* is considered a more serious issue because it leads to Type I errors (for a defintion of Type I error see the discussion on hypothesis testing in Sec. 3.8), in other words can lead to false discoveries
- *Overcoverage* gives intervals which are bigger than they should be, so are sometimes called "conservative", which is considered a less severe problem, causing Type II errors, failure to reject current thinking, in other words a loss of statistical power.

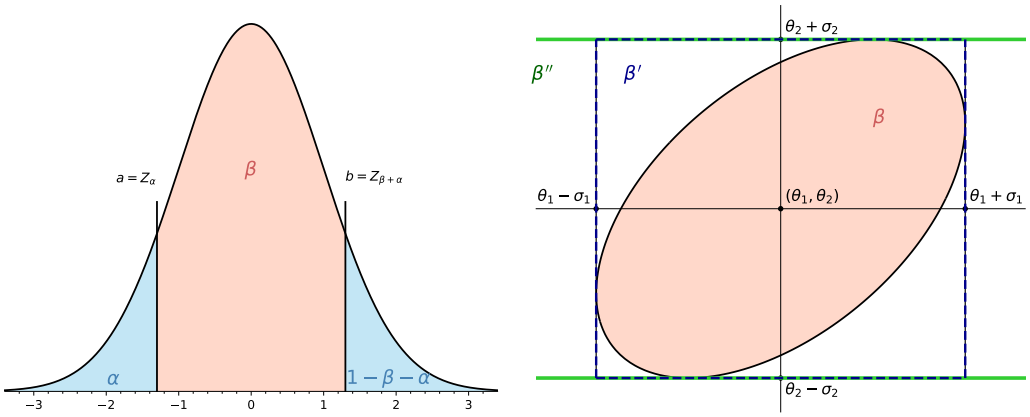

Figure 3.14: Left: demonstration of one-dimensinoal normal distribution intervals. The fraction contained within the interval is β, the fraction within each tail is $\alpha = (1 - \beta)/2$. Some typical α, β and corresponding Z-scores are given in Table 3.2. Right: demonstration of two-dimensional normal distribution intervals. The fraction contained within the ellipse is β, the fraction within the dark blue dashed rectangle is β', and the fraction in the infitite interval enclosed in green lines is β''. Some typical values for β, β', β'' are provided in Table 3.3.

Table 3.2: Table of one-dimensional normal distribution intervals and Z-scores. The areas β and α are shown in Fig. 3.14 (left).

Z	$\beta = (1-\alpha)/2$	α	Notes
1.00	0.683	0.159	Convention for quoting uncertainty
1.65	0.900	0.050	Medical convention for drug effectiveness
1.96	0.950	0.025	Common convention for *exclusion* or *disfavouring*
2.00	0.954	0.022	Some fields quote uncertainties at 2σ
2.58	0.990	5.0×10^{-3}	Sometimes used as a stricter *exclusion* bound
3.00	0.997	1.3×10^{-3}	Physics convention for *evidence*
5.00	0.9999994	2.9×10^{-7}	Physics convention for *discovery*

Table 3.3: Table of two-dimensional normal distribution intervals and Z-scores. The area β is the fraction inside the ellipse, the area β' is the fraction inside the circumscribed rectangle, the area β'' is the fraction inside the one-dimensional band. The corresponding areas are show visually in Fig. 3.14 (right).

	$Z = 1$	$Z = 2$	$Z = 3$
β	0.393	0.865	0.989
β' for $\rho = 0.00$	0.466	0.911	0.995
β' for $\rho = 0.20$	0.471	0.912	0.995
β' for $\rho = 0.50$	0.498	0.917	0.995
β' for $\rho = 0.80$	0.561	0.929	0.995
β' for $\rho = 0.90$	0.596	0.936	0.996
β' for $\rho = 0.95$	0.622	0.941	0.996
β' for $\rho = 1.00$	0.683	0.954	0.997
β''	0.683	0.954	0.997

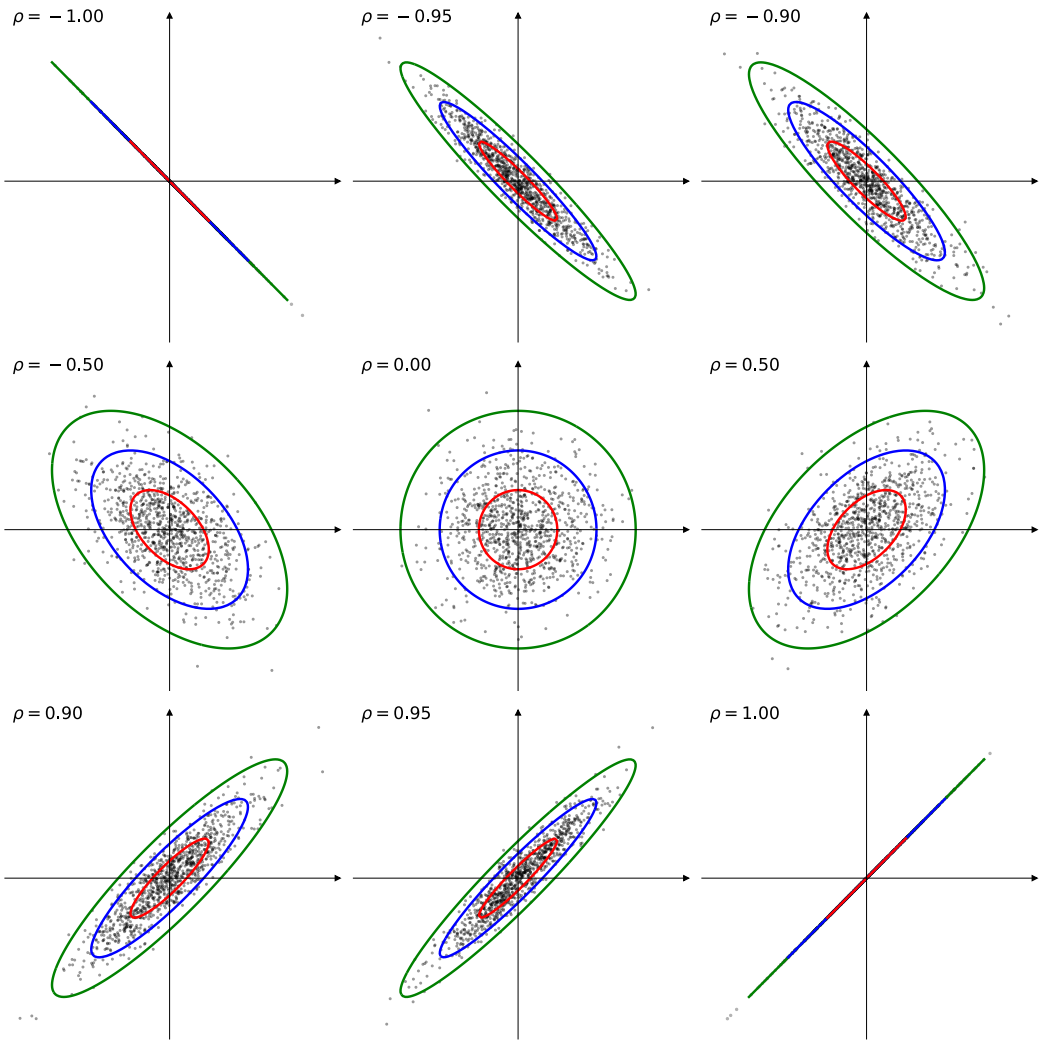

Figure 3.15: Some demonstrations of different correlation values for 2D Gaussians. Contours are shown at $Z = 1$, $Z = 2$, $Z = 3$ which contain 39.3%, 86.5%, 98.9% in blue, red and green.

3.7.1 Bayesian credible intervals

As mentioned before, this course does not cover much Bayesian statistics (it is covered more in the *Advanced Statistics* course) but is instructive for our discussion if we briefly highlight the concept of an interval in the Bayesian sense. By use of Bayes' theorem (see Sec. 2.2.8) we can obtain a posterior p.d.f. for some parameter, μ, based on observations of X, $p(\mu|X)$. A Bayesian *credible interval* of μ corresponding to some confidence β is constructed by requiring that

$$\beta = \int_{\mu_1}^{\mu_2} p(\mu|X)d\mu. \tag{3.72}$$

Once we have the posterior it is straight forward to quote a range (an interval) for the parameter at some confidence level. Normally this is done by finding the *central* interval (or narrowest interval) containing the fraction β. If we want to set only an upper limit then we just set the lower bound of the interval to the lower possible value, $\mu_1 = \mu_{\rm min}$. This is pictorially demonstrated below in Fig. 3.16.

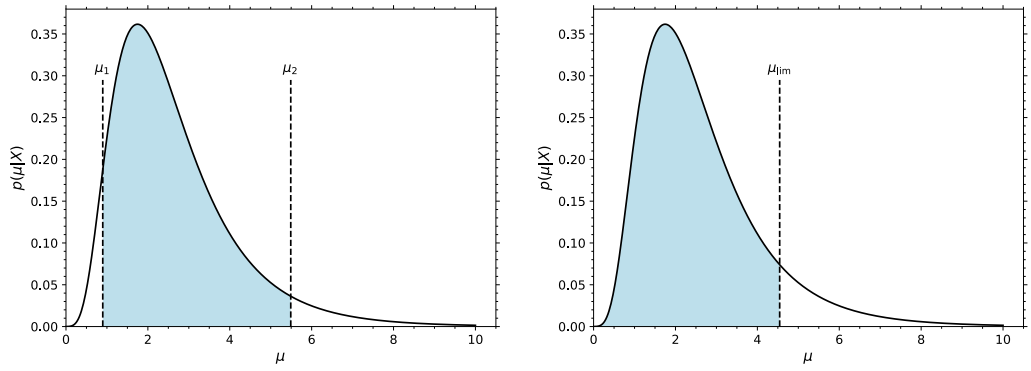

Figure 3.16: A demonstration of a Bayesian credible interval at 90% confidence (left) and a Bayesian upper limit at 90% confidence (right) based on the posterior distribution.

3.7.2 Neyman-Pearson intervals

The classical (frequentist) statement about confidence intervals has a different meaning, since we have no posterior p.d.f. in the classical case. For a frequentist the *confidence interval*, $[\mu_1, \mu_2]$, is a member of a set such that

$$p(\mu \in [\mu_1, \mu_1]) = \beta. \tag{3.73}$$

For a frequentist the value of μ is fixed and confidence interval, given by μ_1 and μ_2 will depend on the observations X and will vary over an ensemble of experiments. The frequentist statement is that the interval $[\mu_1, \mu_2]$ will contain the *true* value of μ in a fraction β of experiments.

To construct the confidence interval we begin with the p.d.f. of the observation given a fixed value of the parameter μ, $p(X|\mu)$. This is not the same as the posterior p.d.f., $p(\mu|X)$, which has a fixed observation and a varying parameter μ. Given that μ actually varies we can now construct a *confidence belt* built out of $p(X|\mu)$ for various different values of μ, and for a central interval (with equal probabilities on either side) it is built by computing values

of X_1 and X_2 for each value of μ such that

$$p(X < X_1|\mu) = p(X > X_2|\mu) = (1 - \beta)/2. \tag{3.74}$$

This is pictorially represented below in Fig. 3.17 which shows the values of X_1 and X_2 against different values of fixed μ. If we observe a value X_0 then the confidence interval $[\mu_1, \mu_2]$ is the union of all values of μ in the vertical slice through the belt.

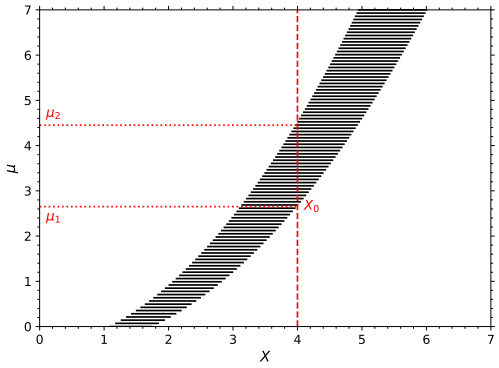

Figure 3.17: The black lines show the confidence intervals in X for a given fixed values of μ. If we observe X_0 shown by the red dashed line then the confidence interval for μ, $[\mu_1, \mu_2]$ (red dotted lines), is given by the union of all values of μ in the vertical slice.

3.7.3 Intervals at physical boundaries

The advantage of the Bayesian approach is that it allows us to remove unphysical regions using a prior. For example if I want to ensure that my parameter μ cannot be negative than I can introduce a prior in which $p(\mu) = 0$ for all $\mu < 0$. A good example for requiring a positive value is when measuring an invariant mass or an energy or an amount of something which in the real world cannot be negative. A frequentist procedure can allow us to estimate a parameter value and have a confidence interval for that parameter which goes into the unphysical region and many find it rather unsatisfactory to quote measurements of things like fractions which are less than zero or greater than unity.

Let's see a simple example with some normally distributed data. I make a sample estimate of the mean of the data and let's assume the amount of data is such that my uncertainty on that estimate is unity. If I want to construct a confidence belt for the distribution mean μ as a function of the mean I measure then this is quite straightforward. Clearly there is a one-to-one mapping between the central values, and then the width of the belt is given by the sample variance (in this case unity). If we further suppose that there is some physical restriction which requires that μ is positive, and hence the confidence belt cuts off at $\mu < 0$.

This gives me the plot on the left of Fig. 3.18, where the confidence belt is drawn at 68.3% confidence. I could also chose to provide only an upper limit, shown on the right of Fig. 3.18, where this time the belt is drawn at 90% confidence. Seeing the code to generate these plots may be instructive so I put it below.

```
1 import numpy as np
```

```
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')
from scipy.stats import norm

fig, ax = plt.subplots(1, 2, figsize=(12.8, 4.8))

mus = np.linspace(0,6,100)

# get x band for beta=0.683
beta = 0.683
alpha = (1-beta)/2
x1 = [ norm.ppf(alpha, loc=mu) for mu in mus ]
x2 = [ norm.ppf(1-alpha, loc=mu) for mu in mus ]

ax[0].fill_betweenx(mus, x1, x2, ec='k', fc='0.75')

# get upper limit band for beta=0.9
xu = [ norm.ppf(beta, loc=mu) for mu in mus ]

ax[1].fill_betweenx(mus, x1, 10, ec='k', fc='0.75')
```

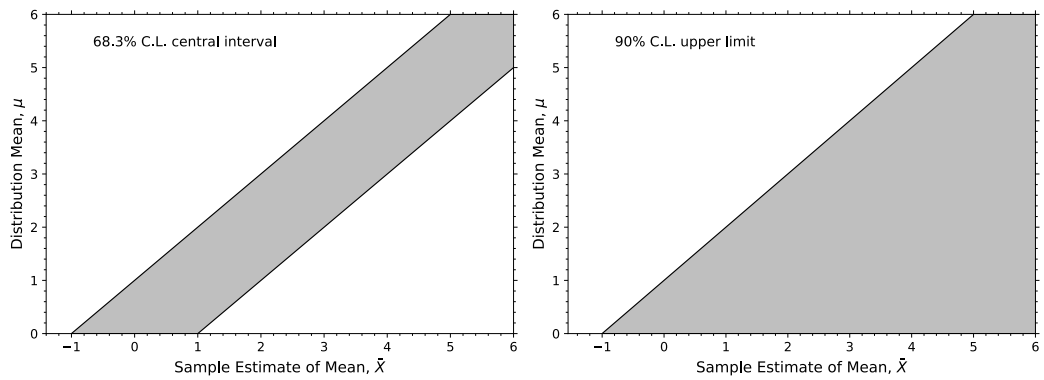

Figure 3.18: Confidence belts when estimating the positive mean of a normal distribution.

Whilst the confidence belt incorporates the fact that μ cannot be negative, it is still possible for me to measure a negative X and therefore if I draw a vertical line at *e.g.* $X = -2$ and take the union of all μ values I get an empty confidence interval. However it is not true that all possible values of μ are ruled out by such an observation. The interpretation is that we happened to observed one of the experiments that doesn't contain the true value (remember the confidence belt will only contain the true fraction β of the time). The other realisation, which we will demonstrate in a moment, is that the confidence belt actually has the wrong *coverage*, in this case it undercovers, for measurements near or beyond the physical boundary.

Let's take a look at another example (similar to one we have looked at before) where we want to fit a peaking signal on top of a smoothly falling background. I will fix (*i.e.* assume that I know) all of the parameters in the model apart from the signal yield and the background yield. I will generate a small data sample with a small signal and then fit this back (with an extended maximum likelihood fit) for the signal yield where I will restrict that signal to be positive. I can then repeat this experiment many times to see what happens over a large ensemble of experiments. The distribution of fitted values I obtain over 500

experiments is shown on the left of Fig. 3.19 below. Quite often a value of zero or near zero (*i.e.* where the boundary is gets fitted). An example profile likelihood scan of one of those fits that is near zero is shown on the right of Fig. 3.19. The issue we have is trying to extract an uncertainty on that fitted value. The likelihood abruptly stops at the physical boundary so it's impossible to work out the gradient and double differential near the minimum (which is one way to approximate the variance). The profile likelihood also abruptly stops before it crosses the approrpriate threshold for 68.3% C.L. *etc.*. This means our lower error will typically extend to the boundary, which is ok, but the upper error does not get any big to account for this. The overall effect is that in my example if I count what fraction of my 500 experiments provide a 1σ interval (using the Hessian matrix method) containing the true value, the result is 41.4%. This is considerably less than the 68.3% I should get, demonstrating that my method for extracting the confidence intervals severely undercovers.

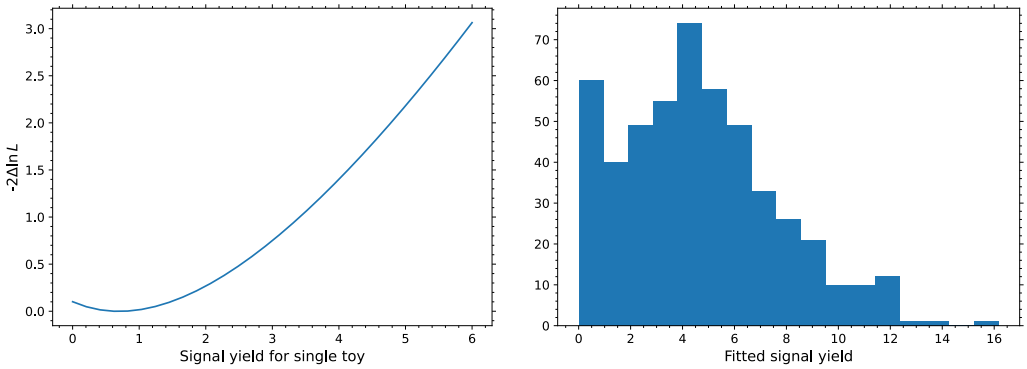

Figure 3.19: Left: fitted values of a low and restricted positive signal yield over the 500 experiments. Right: a demonstration of the profile likelihood for one single experiment which fits a value close to the physical boundary.

One approach to get around the issue of physical boundaries is to simply quote an upper limit on parameters when we are close to the physical boundary. The problem with this is firstly that we will have the incorrect coverage and the second is that we then need to define a point where we transition from quoting an upper limit to quoting a confidence interval, if we end up measuring a value sufficiently far from the boundary. This introduces a problem called "flip-flopping". Take a look at the confidence belt in Fig. 3.20. It is the same example as used above of estimating the distribution mean from the sample mean for a normal distribution, where now I deicde that

- if my measured value is less than 3σ from 0 then I will quote an upper limit, the green region)
- if my measured value if less than 0 (*i.e.* the point where I would cross the physical boundary in μ) then I quote the upper limit that is found at the boundary (*i.e.* where $\mu = 0$), the red region
- if my measured value if more than 3σ from 0 (*i.e.* I have strong evidence of a signal) then I will quote a central interval

In this plot the red region is conservative and overcovers and most of the green region actually undercovers, with coverage only correct in the blue region where we are away from

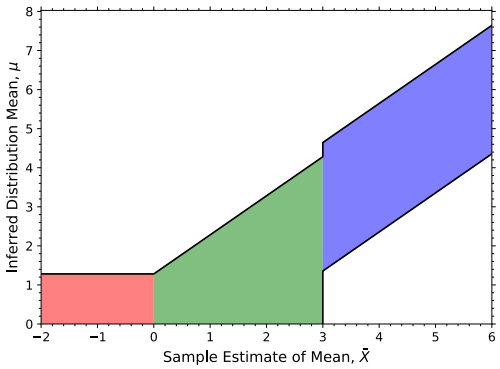

Figure 3.20: An example of flip-flopping

the boundary anyway. Furthermore we see discontinuous kinks in the confidence belt. So we really need something better for computing frequentist confidence intervals when we are at or near a physical boundary. Fortunately there is method, based exactly on long-run frequencies of outcomes, although for many parameter systems it can be computationally expensive. We will see that method now.

3.7.4 Feldman-Cousins intervals

The Feldman-Cousins method [18] provides a method for producing intervals with the correct coverage and which overcome the flip-flopping problem. The procedure is, for each X, to start from the maximum likelihood estimate of μ, $\hat{\mu}$, with the physical bound applied. Then for a fixed μ compute the likelihood ratio,

$$R = \frac{p(X|\mu)}{p(X|\hat{\mu})}, \tag{3.75}$$

and add values of X to the interval from higher to lower R until the desired probability content, *e.g.* 90%, is realised. Thus, this interval will have the correct coverage *by definition*. Going back to our simple example of the normal distribution then $\hat{\mu} = 0$ when $X < 0$ and $\hat{\mu} = X$ when $X \geq 0$ so that the likelihood ratio will be

$$R = \frac{p(X|\mu)}{p(X|\hat{\mu})} \begin{cases} \frac{\exp[-(X-\mu)^2/2]}{1} & \text{if } X \geq 0 \\ \frac{\exp[-(X-\mu)^2/2]}{\exp[-X^2/2]} & \text{if } X < 0 \end{cases} . \tag{3.76}$$

It's perhaps easiest to see this explicitly written in a snippet of code. Using this example I produce the plot shown in Fig. 3.21, which is a reprouction of the original plot in the Feldman-Cousins paper [18]. This is the code used:

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')
from tqdm import tqdm

def fc_interval(mu, sigma, xvals, beta):

    # this is our nominal p.d.f p(X| \mu)
```

```
10
11      dist = norm(loc=mu, scale=sigma)
12
13      bw = xvals[1]-xvals[0]
14
15      p = []
16      R = []
17
18      for x in xvals:
19
20          # get the probability content at each x
21          p.append( dist.pdf(x) * bw )
22
23          # compute the denominator
24          # i.e. probability content when mu = mu_best
25          mu_best = max(0, x)
26          p_mu_best = norm.pdf(x, loc=mu_best, scale=sigma) * bw
27
28          # compute likelihood ratio
29          R.append( p[-1] / p_mu_best )
30
31      # make sure they are arrays
32      p = np.asarray( p )
33      R = np.asarray( R )
34
35      # you will nominally have to scan in a larger range than you plot in
36      # so need to make sure that enough of the distribution is in the range
37      # used for x
38      if sum(p) < beta:
39          print(f'WARNING: X doesnt have enough prob for {beta} at mu={mu}')
40          return [np.nan, np.nan]
41
42      # now get the indices of the likelihood ratio array sorted from highest
43      # to lowest
44      index = np.argsort( -R )
45
46      # now loop through adding to the intervals based on the probability
        content
47      p_sum = 0
48      # start from the middle and work outwards
49      xmin = mu
50      xmax = mu
51      for ind in index:
52          p_sum += p[ind]
53          if xvals[ind] > xmax:
54              xmax = xvals[ind]
55          if xvals[ind] < xmin:
56              xmin = xvals[ind]
57
58          # once probability content is reached then stop
59          if p_sum > beta:
60              break
61
62      return [xmin, xmax+bw]
63
64  # the space of x values to look in (make this quite large so you can be sure
```

```
# to have enough prob content)
x = np.linspace(-2, 12, 1000)

# the space of mu values to scan
mus = np.linspace(0, 8, 100)

intervals = [ fc_interval( mu=mu, sigma=1, xvals=x, beta=0.9 ) for mu in tqdm(
    mus) ]

intervals = np.asarray(intervals)

fig, ax = plt.subplots()
ax.fill_betweenx( mus, intervals[:,1], intervals[:,0], fc='0.75', ec='k' )
```

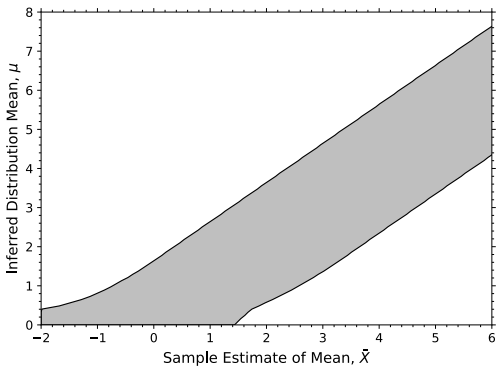

Figure 3.21: Demonstration of Feldman-Cousins intervals for a simple Gaussian example.

What we see with the Feldman-Cousins interval is some lovely features. It does not have the flip-flopping problem, the transition from upper limit to central interval is smooth and also "built-in" in the sense that has soon as my lower limit has $\mu_1 > 0$ then I no longer quote an upper limit but an interval. Every single value of X on this plot has the correct coverage, which is also "built-in" to the interval construction. One way to think of the Feldman-Cousins intervals is the pure frequentist approach in the sense that we build the interval such that on repeated experiments it will has the exact coverage.

In practise, computing Feldman-Cousins intervals using Eq. (3.75) is slow and has to be done numerically, so it is much more common to actually produce Feldman-Cousins intervals by means of Monte-Carlo simulation. This still tends to me much slower than a maximum-likelihood fit because, as we will see in a moment, we have to run many maximum-likelihood fits but it's still much quicker than attempting to compute p.d.f.s and the likelihood ratio. So let's know see how this is done in simulations if we revert back to our problem of fitting for a positive signal on a smoothly falling background which we saw in Fig. 3.19 gave us such severe coverage issues.

The Feldman-Cousins MC procedure is as follows:

- Run a maximum likelihood fit to our dataset as normal to determine $-2\ln L(\hat{\mu})$ at the best-fit point, $\hat{\mu}$.

- We then want to pick some test values of μ to find out the confidence level at that value, we can then iterate around this point until we reach the desired confidence level.

In practise this is often done by scanning a few values of μ and then interpolating to the desired C.L.

- So let's pick some test value μ_0, we then compute $-2\ln L(\mu_0)$ at this value in our dataset.
- Then define a test-statistic which is the log-likelihood ratio of Eq. (3.75), $T = -2\Delta\ln L(\mu_0)$.
- What we can now do is generate a set of pseudo-experiments (sometimes also called toys), which are a series of experiments replicating the one we have, from the fixed point μ_0.
- We then repeat the procedure above, by fitting each of those pseudoexperiments once to determine the likelihood at the best fit in toy i $-2\ln L(\hat{\mu}_i)$, and once more to determine the likelihood at the fixed point $-2\ln L(\mu_{0,i})$ to provide for each toy a value of the test statistic, $T_i = -2\Delta\ln L(mu_0)_i$.
- The value of 1-CL (*i.e.* the p-value) at the scan point, μ_0, is then given by the fraction of toys which fit worse than the data, *i.e.* the fraction for which $T_i > T$.

I have some example code for this but it is a bit long winded to put in these notes, however I will make it available via the course `gitlab` page. The result comparing three different interval methods is shown in Fig. 3.22. You can see that the interval based on the Hessian matrix (*i.e.* the inverse double differential of the log-likelihood) stretches all the way to zero and clearly undercovers at the top end. The profile likelihood method runs directly into the boundary, which in this case is ok at 1σ (68.3%) because the best fit is just about far enough from the boundary, however this would not be sufficient for 2σ. The Feldman-Cousins intervals nicely steers us away from the boundary. Notice the flucutations in the Feldman-Cousins intervals due to the number of toys being run at each point. We will discuss this is in a bit more detail in the lectures.

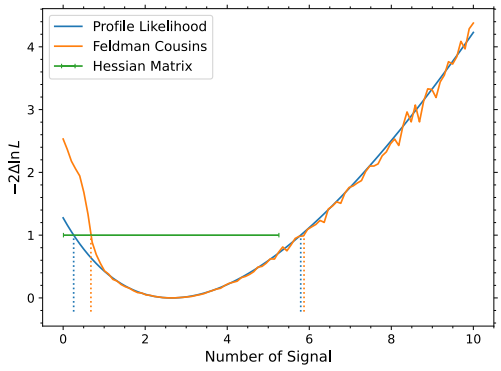

Figure 3.22: A comparison of the Hessian, profile likelihood and Feldman-Cousins methods for intervals near physical boundaries.

3.8 Hypothesis testing

We have seen in the last several sections how to make estimates of parameter values and their uncertainties from data. We will now turn our attention to the topic of *testing* the data

to see if fulfills a particular hypothesis or not. Typical examples of this include questions like, does my data suggest this drug cures people, or does this new particle exist, does my classification algorithm separate pictures of dogs and cats better than random guessing?

We have already covered one simple form of hypothesis test, the test that some hypothesis, H_0, against all other possible hypotheses, which is the goodness-of-fit tests we have discussed in Sec. 3.6. In this section we will cover tests between two different hypothesis, which are typicall labelled the *null hypothesis*, H_0, and the *alternate hypothesis*, H_1. These allow us to exploit the statistical power in knowledge of what the alternate hypothesis is (allowing for us to extract parameters of the hypothesis from the data) and therefore these types of test are typically much more performant than simple goodness-of-fit tests.

Let X be some *test statistic*, *i.e.* some function of our observations. We can find the probability distribution of X under the null and alternate hypotheses and then use this to ascertain how much of each distribution falls above or below a particular value. This fraction will tell us the frequency with which we either accept or reject the given hypothesis at some confidence value. In a two hypothesis scenario we have four possible outcomes:

(i) we correctly accept the null hypothesis when it is true

(ii) we incorrectly reject the null hypothesis when it is true

(iii) we correctly accept the alternate hypothesis when it is true

(iv) we incorrectly reject the alternate hypothesis when it is true

This gives us two types of possible of errors:

Type-I error (loss) - we reject the null hypothesis when it is true (which happens with probability α)

Type-II error (contamination) - we accept the null hypothesis when it is false (which happens with probability $1 - \beta$)

Let's write this out mathematically and then see it visually. The size of the region for which we get a Type-I error is defined by

$$\alpha = P(X > X_0|H_0), \tag{3.77}$$

where X_0 is some value of the random variable X which we can adjust such that α passes some threshold we can choose. The size of the region for which we get a Type-II error is defined by

$$\beta = P(X > X_0|H_1), \tag{3.78}$$

where we will use the convention that the Type-II probability is $1 - \beta$. The plot below, Fig. 3.23, shows an example of two distributions of a test statistic under the null and alternate hypothesis and thus what the values of α and β correspond to. Remember a test statistic can be any function of the data. You can see from this figure that there is then a trade off between how big α is relative to $1 - \beta$ and we are free to choose the value of X_0 such that we can either make α smaller or β larger.

In most cases we prefer to keep the Type-I error rate low, sacrificing a higher Type-II error rate. One example is the justice system - we would prefer to keep the rate of innocent

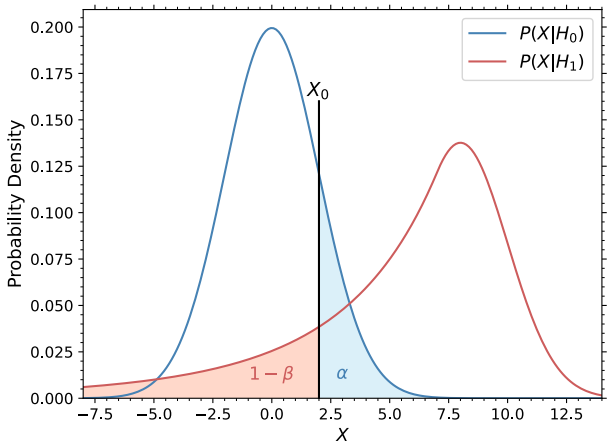

Figure 3.23: Comparison of test statistic distributions.

people going to prison low in exchange for sometimes letting guilty people go free (perhaps this is matter of taste and politics). Another example is when claiming particle physics discoveries - we would rather keep the rate of falsely claiming we have found new particles low in exchange for occasionally ascerting there aren't any when infact there are.

Once we have chosen the test statistic the distributions in Fig. 3.23 are fixed, we are then only free to chose the value which fulfils our predetermined thresholds for α and β. But what we would really like to know is what *choice* of statistic maximises the distance between the two distributions and therefore what choice gives us the smallest possible values of α and β, or perhaps phrased more carefully what choice gives us the smallest possible α for a given value of β.

3.8.1 The Neyman-Pearson Lemma

The question is what is the powerful hypothesis test I can form? The answer is provided by the Neyman-Pearson lemma which states that the most powerful test between two hypotheses is constructed from the likelihood ratio between them. We have already deployed the use of the likelihood-ratio when discussing Wilks' theorem (Sec. 3.4) but this is a slightly different use case. The beauty is that *using* Wilk's theorem we know, at least asymptotically, how the likelihood-ratio is distributed. Thus we can simply compute p-values or significances for rejecting a given hypothesis based on the likelihood ratio,

$$T = -2\ln\left(\frac{L(X|H_0)}{L(X|H_1)}\right), \tag{3.79}$$

where the test statistic, T, will be χ^2 distributed (with $k = 1$).

To demonstrate this I repeat a test done earlier which emulates searching for the Higgs boson. The repeated plot, Fig. 3.24, shows two fits to some generated data which *does* contain Higgs signal. The null hypothesis in this case is that there is no Higgs boson, so this distribution should only contain background. The alternate hypothesis is that there is a Higgs boson, which we know will give us a peak in this distribution. We do not know the position, width or height of this peak, but that doesn't matter because we can fit it from the

data. The only assumption being made is that the Higgs signal gives a Gaussian peak (in real-life this is more carefully simulated). To perform my hypothesis test, I fit the data twice. Once with the background only model and a second time with the signal-plus-background model. The resulting value of T (twice the negative difference of the log likelihoods) should be asymptotically distributed as a χ^2 with one degree of freedom. I can then compute the α value of my observation from the χ^2 c.d.f. which gives me the chance that there really is no Higgs yet I am claiming there is. It is rather arbitary then what value this has to take for us to claim an "observation" or "discovery", but the convention in particle physics is 5σ. Thus I can compute the resulting p-value and significance of my observation, which yields $\sim 5\sigma$, *much* better than I acheived from a simple χ^2 goodness-of-fit test comparing the data to the background only.

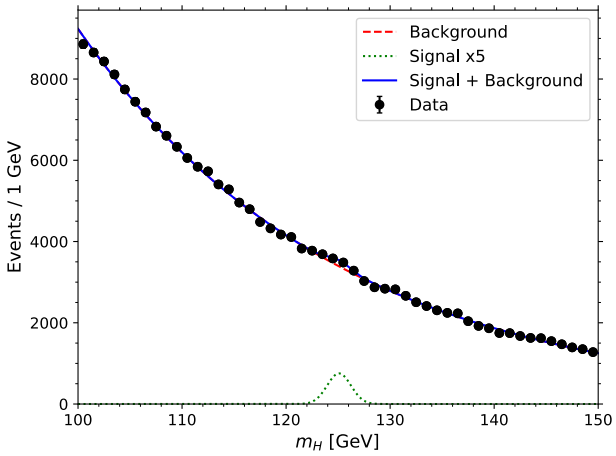

Figure 3.24: An example of a hypothesis test performed along with an estimate.

3.8.2 Quoting a significance

When we perform a goodness-of-fit test or a hypothesis test one of the things we end up quoting is a p-value. This gives us the "statistical probability we got it wrong"[6]. In other words we can say at a confidence level of $(1 - p)$ that a particular hypothesis is ruled out (possibly in favour of an alternate hypothesis). Often the p-value is found by explictly integrating the distribution under a given hypothesis, although we may also exploit Wilk's theorem that the likelihood ratio (i.e. difference of the natural logarithm of the likelihood) is asymptically distributed as a χ^2 distribution. The p-value can be considered as a rather fundamental quantity but it is often a bit hard to interpret, especially when we use it to rule out a hypothesis because we can end up with incredibly small p-values. Consequently, by convention the p-value is often converted into a Z-score or significance (corresponding to the equivalent p-value of a normal distribution). However, there can be some discrepancy, and therefore one has to be rather careful, about whether you consider your test to be one-sided or two-sided, as this will result in a different significance.

Let's see an illustrative example of a two-sided test and a one-sided test, to see what's going on. The two-sided test is perhaps the "classic" example. Let's say I make an estimate

[6]This is only a statistical measure, accounting for random flucutations, it does not account for you systematically doing something wrong which skews the results.

of some parameter θ based on some observations and obtain the result, $\hat{\theta} = \theta_1 \pm \sigma_\theta$. I then want to compare this to some prediction (null hypothesis) value, θ_0. It's clear to see that for a situation with no constraints on θ that this is then a two-sided test. I would be just as surprised if my result had fluctuated in either direction (either above or below θ_0). Consequently when converting the p-value to a Z-score I need to incorporate both sides of the normal distribution tail. This is equivalent to the conversion,

$$Z = \Phi^{-1}(1 - p/2) \tag{3.80}$$

where Φ^{-1} is the inverse c.d.f. of the normal distribution. Notice that this is equivalent to the computation we have often used from the χ^2 distribution with one degree of freedom,

$$Z = \sqrt{\Psi^{-1}(1 - p)}, \tag{3.81}$$

where Ψ^{-1} is the inverse c.d.f. of the χ^2 distribution.

Now let's consider a one-sided example. This could be when fitting for the amount of some "signal" (*i.e.* something that cannot be negative) to determine whether that signal exists or not (*i.e.* whether the amount is > 0). Perhaps another example would be a student attempting to get to a lecture for 9am. Their journey takes 15 minutes with a normally distributed uncertainty of 5 minutes (due to traffic and number of beers consumed the previous evening). If you want to compute the p-value that they will be late for the lecture then this is one-sided (because it doesn't matter if they turn up a bit early). In this the case the relevant conversion is

$$Z = \Phi^{-1}(1 - p), \tag{3.82}$$

using the inverse c.d.f. of the normal distribution and

$$Z = \sqrt{\Psi^{-1}(1 - 2p)}, \tag{3.83}$$

using the inverse c.d.f. of the χ^2 distribution. Below is a snippet of code which demonstrates this

```
from scipy.stats import norm, chi2

p = 0.05

print( norm.ppf( 1 - p/2 ) )
print( chi2.ppf( 1 - p, 1 )**0.5 )

print( norm.ppf( 1 - p ) )
print( chi2.ppf( 1 - 2*p, 1 )**0.5 )
```

which produces the following output

```
1.959963984540054
1.959963984540053
1.644853626951472
1.6448536269514737
```

The tail probabilities for a one-sided and two-sided test are shown in Fig. 3.25.

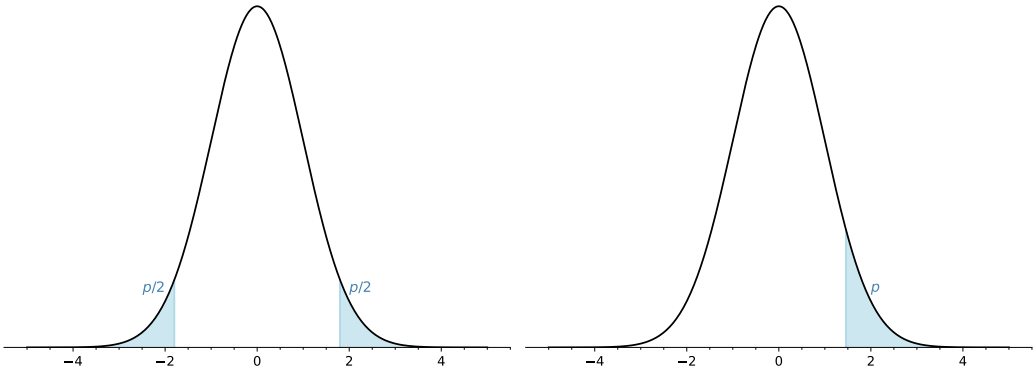

Figure 3.25: An example of the two-sided tail probability (left) and the one-sided tail probability (right). The conversion to Z-score is different for different types of test.

3.9 Limit setting

Quite often we search for something that we do not find. Consequently we want to set upper (or sometimes lower) limits on parameters. A typical example is a search for a new particle which I do not end up seeing with any statistical significance so instead I decide to set an upper limit on its production rate (or cross-section). I didn't see the particle but my data is still useful because it tells me that I didn't see it, so I can instead say (at some confidence level) that it isn't produced. This may seem rather unexciting but is quite often the "bread-and-butter" of scientific data analysis. Maybe I didn't find something new but I can still use my data to rule out (at some confidence level) a variety of different models.

We have already seen (see Sec. 3.7) that for a frequentist there can be considerable coverage issues at physical boundaries. For example if I want to measure the production rate of a particle which I don't see, then inevitably I will measure a value of that production rate consitent with the physical boundary at zero. We have already discussed one method of setting limits in such situations using the Feldman-Cousins method but I also want to mention some generic points about limit setting and limit setting methods, particularly in the context of signal searches.

So imagine we are performing some experiment which searches for a signal in the presence of background. This is a simple hypothesis test, where the null hypothesis, H_0, is the background-only, and the alternate hypothesis, H_1, is the signal plus background. I use the signal search in the presence of background as an example but the following logic can equally be applied to any hypothesis test. We already know (from Sec. 3.8) that the most powerful test-statistic we can use to distinguish hypotheses is the likelihood ratio. Consequently we should use the likelihood ratio as a test-statistic for such limit procedures.

If my experiment consists of i different counting measurements of the signal, s_i, background, b_i, with an observation in data of d_i, then the likelihood ratio is

$$X = \prod_i X_i = \prod_i \frac{e^{-(s_i+b_i)}(s_i+b_i)^{d_i}}{d_i!} \Big/ \frac{e^{-b_i} b_i^{d_i}}{d_i!}. \tag{3.84}$$

This is simply the ratio of likelihoods for H_1 and H_0 assuming a counting experiment, so a Poisson p.d.f.. The confidence level (CL) for excluding H_1, given a measurement in data of

X_0, is then

$$CL_{sb} = P_{s+b}(X < X_0). \tag{3.85}$$

In other words, this is the probability that, assuming the presence of both signal and background at their hypothesised levels, the test-statistic has a smaller value than the one found in data. This is given by the quantity β in our discussion of hypothesis tests in Sec. 3.8.

The example above is for a counting experiment but using the test statistic based on the liklelihood ratio will always be the best one to use. It's worth noting that the a log-likelihood ratio will normally be distributed as a χ^2 (although the degrees of freedom may not be obvious). A graphical representation of such a test-statistic distribution and the limit that can be set is shown below in Fig. 3.26. The distribution of the test statistic will have a dependence on the parameter(s) on which you wish to set a limit so the distribution can be moved around (by changing those parameters) until an exclusion of desired confidence level is found.

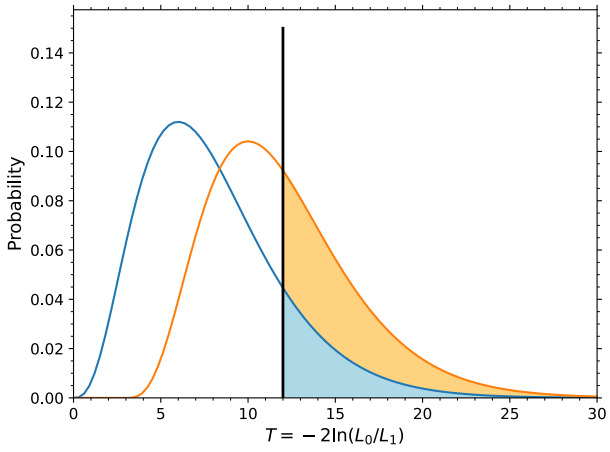

Figure 3.26: A demonstration of a possible log-likelihood ratio test-statistic distribution with an observation in data. The shaded area of $1 - CL_{sb}$ is the region excluded at $1 - CL_{s+b}$ confidence. The distribution of T is shown for different possible values of the parameter. In order to find the value exluded at a specific confidence level the parameter can be adjusted until the shaded region reaches the desired amount.

A fairly common conceptual complication is how to actually obtain the test statistic distribution itself (*i.e.* the ones shown as the blue and orange curves in Fig. 3.26). It is straightforward to find where the black point is, that is simply an evaluation of the log-likelihood ratio on the dataset you have. The question is how to determine the distribution of the test-statistic. The answer is to use a MC simulation. This is done by first fitting the data (using ML estimation) to determine the "best-fit" model. You can then generate simulated samples from this, which mimic the dataset you have, in order to build up the empirical distributions in blue and orange. There are infact asymptotic formulas for producing these if the simulation takes too long (but they are based on assumptions as $N \to \infty$) which can be found in Ref. [19].

The CL_{sb} method is a useful way of setting a limit. However, there is some critique of the method if the overall sensitivity is particularly low. One could imagine a situation in which there is not much experimental sensitivty to a signal, but the data happens to

have fluctuated very low (below even the background only hypothesis) so that you end up excluding a signal that you have little or no sensitivity to. This causes people to question whether it is justified to exclude a model that you don't have any sensitivity. To this end the so-called "CL_s method" was developed which aims to mitigate the problem by essentially down-weighting the p-value for exclusion by an amount dependent on the inherent sensitivity. In this case a test-statistic is defined as

$$CL_s = \frac{p_{sb}}{1 - p_b} \tag{3.86}$$

where p_{sb} is the p-value of the alternate hypothesis and p_b the p-value of the null hypothesis. In this case we need to determine the test-statistic (log-likelihood ratio) distribution under both the null (CL_b) and alternate hypotheses (CL_{sb}) and then we take a ratio of the two regions, this is graphically displayed in Fig. 3.27. What you can see is that if the null and alternate are perfectly distinguishable one ends up dividing the CL_{sb} by unity and thus $CL_s = CL_{sb}$. However if the null and alternate hypotheses are hard to distinguish then the p-value (confidence of exclusion) gets reduced up to the point where they are completely indistinguishable in which case the ratio in Eq. (3.86) becomes unity.

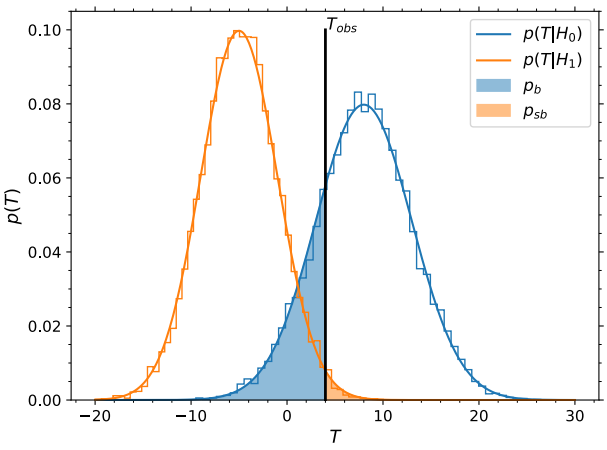

Figure 3.27: A demonstration of the CL_s method in which we want to understand the distribution of the test-statistics under both the null and alternate hypotheses (the blue and orange lines, respectively). We then quote a conservative estimate of the limit using the ratio of p-values.

As before we normally will not analytically know the test statistics distribution under both null and alternate hypotheses so will generate it using MC simulation methods. For this reason I also include a histogram in Fig. 3.27 to show how this might look in real life. The number of samples used in the MC simulation will sometimes need to be very large in order to get an accurate measure of a small p-value. To this end you can exploit the asymptotic formula provided in Ref. [19] for a faster computation.

The test-statistic distribution under the null hypothesis is fixed (so doesn't vary), however quite often it will change for the alternate hypothesis depending on the parameter value that you are setting a limit on (*e.g.* the signal strength). To this end you have to vary the parameter until it produces the test statistic distribution which gives the desired confidence level (normally 90%, 95% or 99% for quoting limits, depending on the field).

3.10 Resampling methods

Resampling methods allow us to mimic the sampling procedure. Any dataset we have is essentially a sample drawn from a parent distribution and normally our statistical procedures want us to determine what the parent distribution or at least produce equivalent samples from it. It is incredibly useful to be able to produce additional samples (*i.e.* in addition to the one we have) based only on the data in the sample we have. In this section we will discuss a few of these methods. These have very far reaching applications in machine learning but can also be used to produce estimates and in data analysis they also allow us to check our procedure over an *ensemble* of observations. The real advantage of resampling methods is that the allow us to infer the properties of a distribution from the data sample alone, they do not require any assumptions about the underlying model or distribution.

3.10.1 Permutation Test

The permutation test is a form of resampling which allows us to determine whether two samples or datasets are consistent with originating from the same parent distriubtion. In this sense it is a useful alternative to the KS-test which we discussed above in the context of goodness-of-fit.

Imagine we have two datasets A and B, with sizes N_A and N_B, containing some random variable X. We want to determine whether X_A and X_B are consistent with originating from the same parent distribution. We define a test statistic which quantifies the difference between the two datasets, the most commonly used is simply the difference of the means of the two samples

$$T = \Delta\bar{X} = \bar{X}_A = \bar{X}_B. \tag{3.87}$$

We then make random swaps, permutations, between the samples, exchanging events between A and B. We do this for all possible permutations of the data such that the sizes of the two new samples remain the same as N_A and N_B. This then builds us an *ensemble* of new datasets. We can then build a distribution of the test statistic over the *ensemble* which essentially quantifies the compatibility between the samples if I randomly swap events over. Clearly, if the events all come from the same distribution then random swaps will not in general effect the mean. We can then determine a p-value as the fraction of the *ensemble* that have a larger value of T than the one we observe in our actual data sample.

3.10.2 Non-parametric bootstrapping

Bootstrapping is procedure for generating an *ensemble* of datasets but resampling from the observered distribution with replacement. Sampling with replacement simply means that I randomly choose an event from my sample to put in my new sample but then I make sure that the event goes back into the sampling population again. For example if I have a data set $X = \{1, 2, 3, 4, 5\}$ and I want to make a new bootstrapped sample from this then I would ask for 5 random choices from X but once I pick an event I allow that event to be picked again, so one of my bootstrapped samples could be $X' = \{3, 1, 2, 2, 4\}$.

The result is that I can generate an *ensemble* of bootstrapped samples assuming that the original sample I have is a good proxy for the parent population. If I have some estimate or inference procedure I can then perform this on each one of my *ensemble* samples and I

will then build the probability distribution of my estimate. This allows me to directly read off the central value and the width from the distribution of the estimate (or I could simply publish the distribution).

Let's assume we have some random variable $X \sim f(X)$ and we have some estimating procedure for providing a point estimate of a parameter $\hat{\theta} = T(X)$. If use the data we observe to provide the estimate $\hat{\theta}$ this is often called the "plug-in" estimate, *i.e.* I plug-in my data to my inference procedure and get a result. All the estimation we have done so far uses plug-in estimation, the sample mean is a good example, $\bar{X} = \hat{\mu} = \sum_i X_i/N$, which is a plug-in estimate for the true mean μ. But as I have mentioned several times we are scientists so we need to do better than just making a point-estimate, we want to know the distribution of the estimate, $g(\hat{\theta})$. Most of our procedures above do this by assuming the distribution is Gaussian and then we simply quote a 1σ uncertainty for which the ML and least-squares estimates provide us a procedure for. The bootstrap method exploits the power of computer simulation to estimate the distribution $\hat{g}(\hat{\theta})$ by sampling directly from the data.

Let's see the example below, where I generate a Gaussian sample and estimate the mean of that sample using the bootstrap estimate, by sampling with replacement, computing the estimate (in this case the mean of the sample) then taking the mean and width of the *ensemble* of estimates. For comparison I also compute the mean and the standard error on the mean using the normal sample estimates discussed in Sec. 3.1.3. The resulting plot is shown in Fig. 3.28.

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')

# generate some random data
np.random.seed(210187)
Nobs = 1000
dset = np.random.normal(size=Nobs)

# generate some bootstrapped sample
Nboot = 5000
bsamps = [ np.random.choice( dset, size=Nobs ) for i in range(Nboot) ]

# get the mean (estimate) and error on mean of original sample
d_mean = np.mean(dset)
d_err  = np.std(dset, ddof=1)/np.sqrt(1000)

# get the mean (estimate) of each bootstrap sample
b_means = [ np.mean( bsamp ) for bsamp in bsamps ]

# now can get the mean and spread of the bootstrap estimate
b_mean = np.mean( b_means )
b_err  = np.std( b_means, ddof=1 )

# print and plot
print( 'Sample Estimate:', d_mean, '+/-', d_err )
print( 'Bootstrap Estimate:', b_mean, '+/-', b_err )

fig, ax = plt.subplots()
ax.hist( b_means, bins=25, alpha=0.5 )
ax.plot( [b_mean, b_mean], [0,650], 'k-' )
```

```
32
33 ax.text(0.05,0.94, 'Sample estimate:', transform=ax.transAxes)
34 ax.text(0.05,0.88, f'$\mu = {d_mean:5.3f} \pm {d_err:5.3f}$', transform=ax.
       transAxes)
35
36 ax.text(0.65,0.94, 'Bootstrap estimate:', transform=ax.transAxes)
37 ax.text(0.65,0.88, f'$\mu = {b_mean:5.3f} \pm {b_err:5.3f}$', transform=ax.
       transAxes)
38
39 ax.set_xlabel('$T(X)$')
40 ax.set_ylabel('Bootstrapped Samples')
41
42 plt.show()
```

Code Block 3.6: A simple example of a bootstrap estimate which produces the plot in Fig. 3.28.

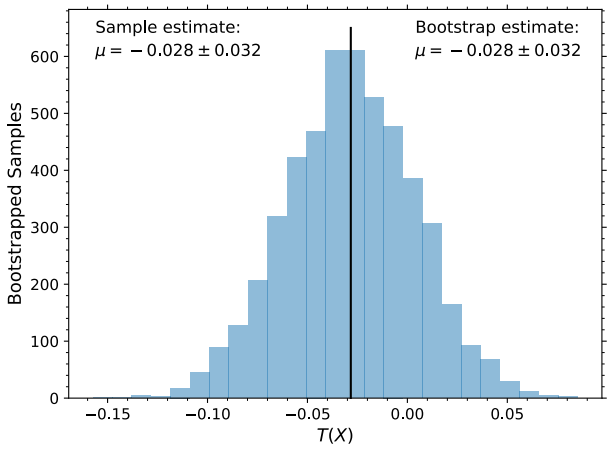

Figure 3.28: An example of a bootstrapped estimate.

Sometimes, depending on the situation, the bootstrapped distribution can be biased (offset from the central value of interest) and sometimes it can be skewed with different size tails. Fortunately we can also estimate by how much from the sample itself and so there is an algorithm which produce bias-corrected and accelerated (BCA) intervals. We will pick up the discussion on that shortly, but first I need to cover a couple of other resampling methods first.

3.10.3 Parametric boostrapping

Parametric bootstrapping is useful when we already have a model and we want to check the robustness of our procedure. In this case we actually generate our *ensemble* of samples from the model itself, based on the parameters we have fitted, hence "parametric". These *ensembles* are sometimes in the literature referred to as *pseudo-experiments* and colloquially as *toys*. It also gives us a convenient way of assessing systematic uncertainties because we can change some assumption of our model, either the model itself or some parameters within that model, generate a load of toys (*i.e.* an *ensemble* of experiments that mimic the one we actually performed) and then fit those pseduo-datasets back with our nominal model

and see what the effect is on our parameters of interest. Another use case is for performing "sensitivty studies" in which I may setup a model and then want to see how my sensitivity to some parameter improves under different assumptions or with larger datasets.

In terms of a robustness check we can use this method to generate samples and fit them back with our nominal model and check that the we get the appropriate parameters back again. If our estimation procedure is unbiased we should on average fit back the values we generated from. Furthermore if our estimation procedure includes an estimate of the uncertainty we can check that the spread of values is appropriate for the uncertainty we estimate using the "pull" distribution (we should be a standard normal distribution). The parametric bootstrapping procedure can allow us to correct for bias or under/over-coverage.

Once again this is best demonstrated with an example, so below I provide one. I am imagining we are performing a "bump-hunt" like the Higgs case we have seen above, where our dataset contains an admixture of smoothly falling background events and a peaking signal. The fit is identical to the one shown in Fig. 3.12. The parameters we care about here are the mean, width and yield of the signal, but we also fit for the slope and yield of the background. The procedure is as follows:

- First I will fit the data, in this case using an EML fit (but any procedure will do).
- I will then generate 250 pseudo-experiments from this fitted model (where I rather lazily call the parameter values the "true" values, θ_{true}) , remembering to also Poisson vary the total number of events in the sample (as we are performing an extended fit)
- I will then fit each of these 250 pseudo-experiments back again with the model (remembering that to avoid any biases I must set the same starting parameters for the fit as I use in the nominal case)
- I will then plot the distribution of the estimated values fitted for each parameter in each toy, the estimated variance for each parameter in each toy and the "pull" which is the fitted value minus the true value divided by the uncertainty, $p = (\theta_{\text{fit}} - \theta_{\text{true}})/\sigma_{\text{fit}}$
- The mean of the distribution of values should be consistent with the value I generated from, this means my estimate of the parameter is unbiased.
- The spread of the distribution of values tells me the average sensitivity I can expect
- The mean of the pull should be consistent with zero (unbiased) and the spread of the pull should be consistent with unity (correct coverage)

The code for my "toy" study is shown in Code Block 3.7 and the output plots in Fig. 3.29. You can see that in this case the fitted values, variances and pulls over the ensemble are Gaussian distributed. The fitted values are shown in the first column so we can for example read off from this that our expected sensitivity to the mean signal position is 0.07 (in some units which if this really was a Higgs sensitivty study would be GeV/c^2). The distribution of variances are in the middle column and the "pulls" in the third column. This tells us in this case that the fit is robust and the error estimate we are using has the correct coverage (all pull distributions are consistent with unit Gaussians). In more complex fits we may have a large number of parameters so making loads of these plots gets a bit unwieldy so in

Fig. 3.30 I show a summary of the pull means and widths which allows us to quickly check for robustness of the procedure[7].

If you wanted to perform such a study to compute a systematic uncertainty, based on some assumption in your model, then you could follow this same procedure but instead when generating change that assumption. For example, in this case, we may want to assess the impact of the assumption that the background follows an exponential shape. In that case we would fit the data with a different model, perhaps this time using a double exponential or a polynomial shape or power law shape, we would then generate toys from this point (an ensemble of pseudo-experiments) and fit each one back with the original model. This would show us the shift and spread of outcomes under the changed assumption.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import truncnorm, truncexpon, norm
from tqdm import tqdm
from iminuit import Minuit, cost
import uncertainties as uncert
plt.style.use('code/mphil.mplstyle')

# set the seed for reproducibility
np.random.seed(21187)

# setup the model

# model parameters
xrange = (100,150)
Ns = 4500
Nb = 9000
mu = 125
sg = 3
p  = 0.04
names = [ 'N_s', '\mu', '\sigma', 'N_b', 'p' ]

# the model functions
def bkg_model(p):
    endp = (xrange[1]-xrange[0])*p
    srtp = xrange[0]
    scl  = 1/p
    return truncexpon(b=endp, loc=srtp, scale=scl)

def sig_model(mu, sg):
    a, b = (xrange[0]-mu)/sg, (xrange[1]-mu)/sg
    return truncnorm(a=a, b=b, loc=mu, scale=sg)

def tot_model(x, Ns, mu, sg, Nb, p):
    return Ns*sig_model(mu,sg).pdf(x) + Nb*bkg_model(p).pdf(x)

def tot_density(x, Ns, mu, sg, Nb, p):
    return Ns+Nb, tot_model(x, Ns, mu, sg, Nb, p)

# a generate function
def generate(Ns, mu, sg, Nb, p):

```

[7]This plot is the brainchild of one my PhD student's, Aidan Wiederhold, so I thank him for it.

```
    # generate poisson number of events first
    Ns = np.random.poisson(Ns)
    Nb = np.random.poisson(Nb)

    # then generate from the model
    b = bkg_model(p)
    s = sig_model(mu,sg)

    bkg = b.rvs( size=Nb )
    sig = s.rvs( size=Ns )

    return np.concatenate( [bkg, sig] )

# a fit function
def fit(dset):
    n2ll = cost.ExtendedUnbinnedNLL( dset, tot_density )
    mi = Minuit( n2ll, Ns=4500, mu=125, sg=3, Nb=9000, p=0.04 )
    mi.migrad()
    mi.hesse()
    return mi

# a plot function
def plot(xvals, ax=None):
    ax = ax or plt.gca()
    # compute mean, sdev and their errors of distribution
    m = np.mean(xvals)
    s = np.std(xvals, ddof=1)
    me = s/len(xvals)**0.5
    se = s/(2*len(xvals)-1)**0.5
    # make a density histogram
    nh, xe = np.histogram( xvals, range=(m-3*s,m+3*s), bins='auto', density=
    True )
    # draw the density histogram
    ax.hist( xvals, bins=xe, density=True, alpha=0.5 )
    # also draw error bars on points
    N = np.sum(nh)/len(xvals)
    cx = 0.5*(xe[1:]+xe[:-1])
    ax.errorbar( cx, nh, N*(nh/N)**0.5, fmt='ko' )
    # draw the normal distribution with the mean and sigma
    x = np.linspace(xe[0],xe[-1],100)
    ax.plot(x, norm.pdf(x,m,s), 'r-')
    # plot the results (use uncertainties package for nice formatting)
    um = uncert.ufloat( m, me )
    us = uncert.ufloat( s, se )
    mstr = f"{um:.1u}".replace("+/-","\pm")
    sstr = f"{us:.1u}".replace("+/-","\pm")
    ax.text(0.01,0.92, f"$\hat{{\mu}} = {mstr}$", transform=ax.transAxes)
    ax.text(0.01,0.86, f"$\hat{{\sigma}} = {sstr}$", transform=ax.transAxes)

## 0. let's do a dummy generation to get ourselves the "real" data
data = generate(Ns, mu, sg, Nb, p)

## 1. now let's fit the data
# this gives us the fit values for our ensemble generation
mi = fit(data)
print(mi)
```

```
98
99  ## 2. now generate 250 toys
100 Ntoy = 250
101 toys = [ generate(*mi.values) for _ in range(Ntoy) ]
102
103 ## 3. now fit each toy and store the values and errors
104 # tqdm prints us a progress bar
105 values = []
106 errors = []
107 for toy in tqdm(toys):
108     mi_t = fit(toy)
109     values.append( list(mi_t.values) )
110     errors.append( list(mi_t.errors) )
111
112 values = np.array(values)
113 truth  = np.array(list(mi.values))
114 errors = np.array(errors)
115 pulls  = (values - truth)/errors
116
117 ## 4. now we can plot them
118 fig, axes = plt.subplots(5,3, figsize=(19.2,24))
119 for i, name in enumerate(names):
120
121     estname = f"\hat{{{name.split('_')[0]}}}"
122     if len(name.split('_'))>1: estname += '_'+name.split('_')[1]
123
124     ax = axes[i,0]
125     plot( values[:,i], ax )
126     ax.set_xlabel( f"${estname}$" )
127
128     ax = axes[i,1]
129     plot( errors[:,i]**2, ax )
130     ax.set_xlabel( f"$\hat{{V}}({name})$" )
131
132     ax = axes[i,2]
133     plot( pulls[:,i], ax )
134     ax.set_xlabel( f"$p({estname})$" )
135
136 plt.show()
```

Code Block 3.7: The code which runs a little toy study and produces the plot in Fig. 3.29.

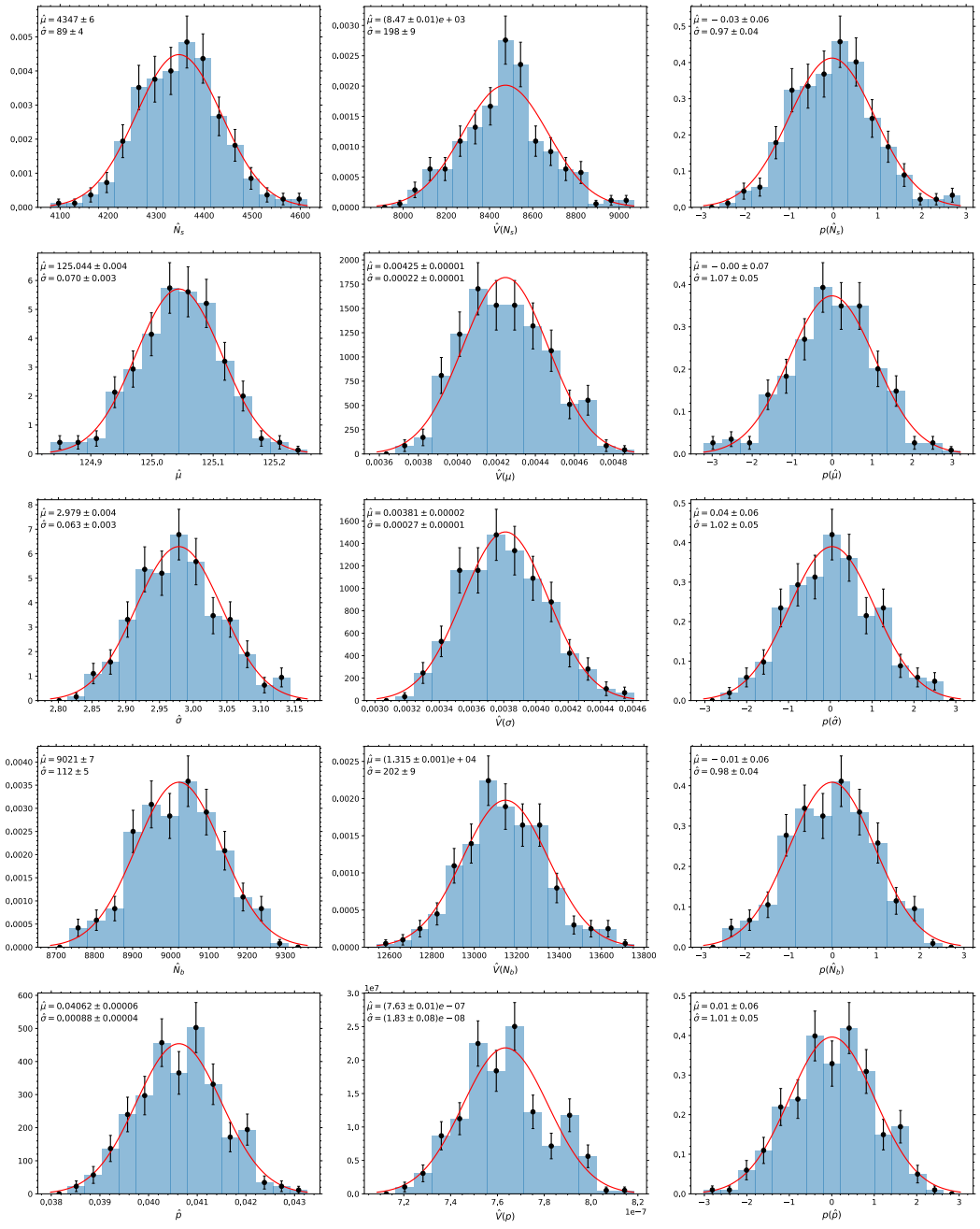

Figure 3.29: A summary of the "toy" study described above. Each row depicts a different parameter of the fit (from top to bottom: the signal yield, the signal position, the signal width, the background yield, the background slope). The first column shows the distribution of fitted values across the ensemble, the second column shows the distribution of fitted variances (error squared), the third column shows the distribution of pulls.

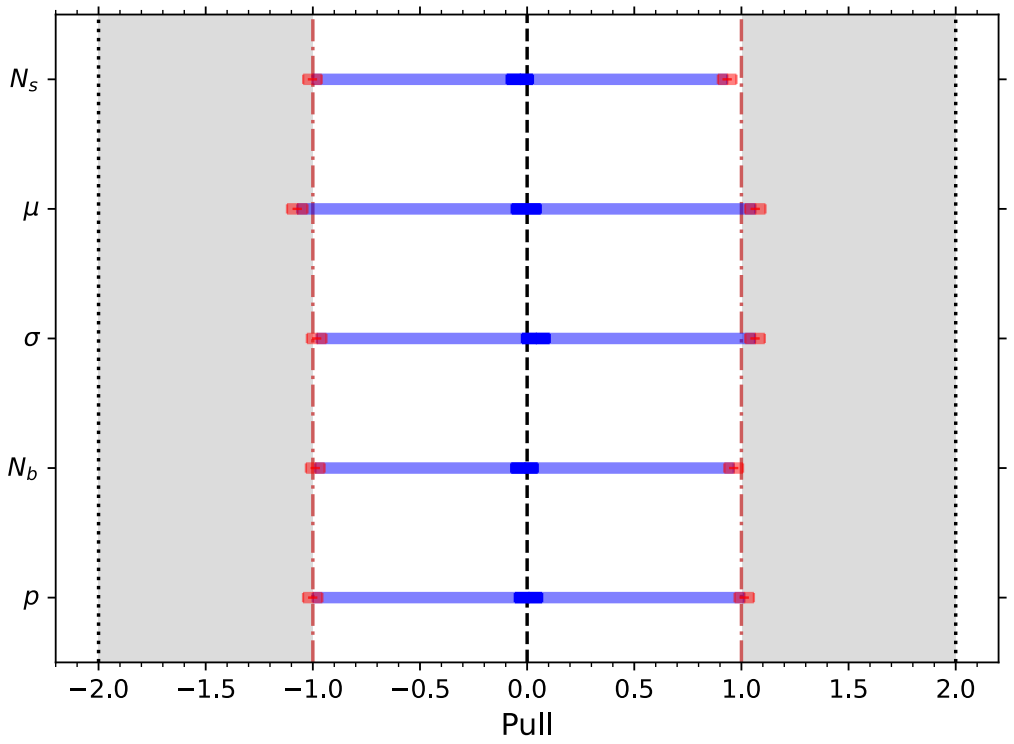

Figure 3.30: A summary of the pulls on each parameter for the "toy" study described above.

Parametric bootstrapping to plot errors on fitted curves

Sometimes it's useful to visualise the error on a fitted curve itself. This is not a trivial problem to solve analytically, one would have to propagate the uncertainty on the model parameters to the uncertainty on probability density (if you want to do this you can make use of the python https://hdembinski.github.io/jacobi/`jacobi` package). Bootstrapping makes this task absolutely trivial. If we have a description of our best fit model, with errors on the fitted parameters, we can parametrically bootstrap the parameters within their uncertainties and then get the spread of the values for the p.d.f. and plot this. I show a little example below, using the same fit setup as before (but with fewer events so you can see the band more easily). I only display the code below picking up from the code above where the `Minuit` object, called `mi`, has already been minimised. Please note that it is considered (and indeed really is) rather misleading to show the error band on the fitted curve as well as the error bars on the data points, because the sources of these two errors are the same: statistical fluctuations in the dataset. Hence, you will often see either one or the other, but not both. Sometimes when you do see both the band on the fitted curve is actually related to a systematic uncertainty on the fitted curve, in which case drawing both is valid.

```
# generate 1000 bootstrapped values
bootstrap_pars = np.random.multivariate_normal( mi.values, mi.covariance, size
    =1000)

# draw the pdf and the bootstrapped error
x = np.linspace(*xrange,200)
y = tot_model(x, *mi.values)
y_boot = [ tot_model(x, *p) for p in bootstrap_pars ]
yerr_boot = np.std( y_boot, axis=0 )

fig, ax = plt.subplots()
nh, xe = np.histogram(data, range=xrange, bins=50)
cx = 0.5*(xe[:-1]+xe[1:])
ax.errorbar( cx, nh, nh**0.5, fmt='ko', label='Data' )
ax.plot(x, y, 'b-', label='Fit')
ax.fill_between(x, y-yerr_boot, y+yerr_boot, fc='b', alpha=0.4, label='Fit
    Error')
```

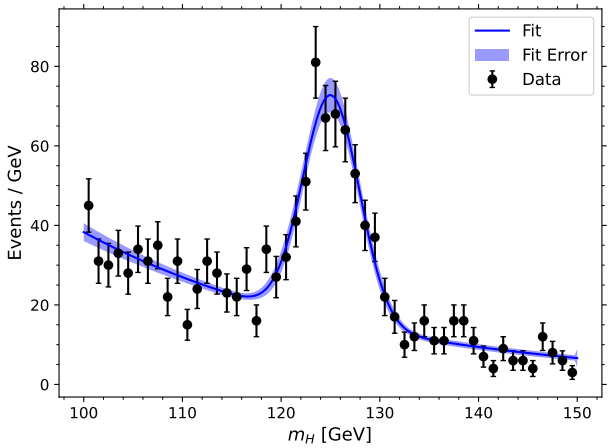

Figure 3.31: A plot showing a bootstrapped error band.

3.10.4 Jackknife resampling

Jackknife resampling actually predates bootstrapping and is a similar concept in which repeated evaluations of estimators are made on the original dataset, but this time omitting one event at a time (or indeed blocks of events). It is particularly useful for estimating (and correcting for) any bias in the estimation procedure.

Assume we have a sample of N observations $(X_1, X_2, \ldots, X_N)$ and some estimation procedure for a parameter based on these, $\hat{theta} = f(X_1, X_2, \ldots, X_N)$. We can then produce N different estimates for θ with a "leave-one-out" procedure,

$$\hat{\theta}_1 = f(X_2, X_3, \ldots, X_N) \tag{3.88}$$

$$\hat{\theta}_2 = f(X_1, X_3, \ldots, X_N) \tag{3.89}$$

$$\vdots \tag{3.90}$$

$$\hat{\theta}_N = f(X_1, X_2, \ldots, X_{N-1}). \tag{3.91}$$

We can then compute the *jackknife average* of these estimates

$$\hat{\theta}_{\text{jack}} = \frac{1}{N}\sum_{i=1}^{N}\hat{\theta}_i. \tag{3.92}$$

We can use this to estimate the bias which is given by

$$\hat{b}(\hat{\theta}) = (N-1)(\hat{\theta}_{\text{jack}} - \hat{\theta}), \tag{3.93}$$

where the factor of $(N-1)$ accounts for the fact that each of jackknifed samples only contains $N-1$ events. Thus we can correct for the bias implictly when we perform the jacknife and produce an *unbiased* jackknife estimate according to

$$\hat{\theta}'_{\text{jack}} = \hat{\theta} - \hat{b}(\hat{\theta}) = N\hat{\theta} - (N-1)\hat{\theta}_{\text{jack}}. \tag{3.94}$$

3.10.5 Bias corrected and accelerated boostrap intervals

Bias corrected and accelerated (BCA) bootstrap intervals deploy the normal sample-with-replacement bootstrapping method but then correct intervals for the bias and skewdness using jackknife estimates. In this case the interval for a given $100(1-\alpha)\%$ confidence level is computed using the emprical c.d.f. of the bootstrapped sample, it is then corrected for the bias and finally accelerated to correct for any skew *i.e.* the rate at which the standard deviation changes as a function of the estimate. This will in general will obtain coverage rates for estimates that are very close to their nominal level, corrected 1st and 2nd order biases.

Recall that in the two-sided interval case then a confidence interval has the appropriate coverage if

$$P(\theta_l < \theta < \theta_h) = 1 - \alpha, \tag{3.95}$$

where $100(1-\alpha)\%$ is the desired confidence level and θ_l and θ_h are the lower and higher edges of the quoted interval. With our standard bootstrap method we sample with replacement and produce a distribution of the estimate which allows us to estimate what is called the

BS-standard error

$$[\hat{\theta}_l, \hat{\theta}_h] = \hat{\theta} \pm Z_\alpha \hat{\sigma}_\theta, \tag{3.96}$$

where Z_α is the number of standard deviations for the required α and $\hat{\sigma}_\theta$ is the bootstrapped sample width.

This approach can also be extended to target a particular quantile instead, known as the *BS-quantile*. In other words for a given confidence level we just look up the location of the appropriate quantile in the bootstrapped sample.

The BCA method uses a simlar procedure but incorporates a correction for the bias and skew using jackknife estimates of the bias correction factor, $\hat{z}$ and acceleration parameter $\hat{a}$ according to

$$\theta_l = \hat{z} + \frac{\hat{z} + \Phi^{-1}(\alpha)}{1 - \hat{a}(\hat{z} + \Phi^{-1}(\alpha)}, \tag{3.97}$$

$$\theta_h = \hat{z} + \frac{\hat{z} + \Phi^{-1}(1-\alpha)}{1 - \hat{a}(\hat{z} + \Phi^{-1}(1-\alpha)}, \tag{3.98}$$

where Φ^{-1} is the inverse c.d.f. of the standard normal distribution. I will not cover the details of how these terms are computed from the jackknife, there is a nice discussion by Efron himself in Ref. [20].

For practical purposes there are many packages and libraries available which can perform this for you, I am a fan of the python `resmaple` package. Below I show an example of a BCA interval estimate of the median of an exponential distribution, which would nominally have a bias and a skew if not BCA corrected.

```
import numpy as np
from scipy.stats import expon
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')

from resample.bootstrap import confidence_interval, bootstrap

rng = np.random.default_rng(1)

data = rng.exponential(size=1000)

# compute BS-standard error
bs_samps = bootstrap( np.median, data, size=5000, random_state=rng )
mb = np.mean(bs_samps)
sb = np.std(bs_samps, ddof=1)


# compute BS-percentile
bs_perc = confidence_interval( np.median, data, cl=0.68, size=500, ci_method='
    percentile', random_state=rng )

# compute the BCA-interval
bs_bca  = confidence_interval( np.median, data, cl=0.68, size=500, ci_method='
    bca', random_state=rng )

# draw it
fig, ax = plt.subplots()
ax.hist( bs_samps, bins=15, alpha=0.4, label='Data' )
```

```
ax.axvline( np.log(2), lw=2, c='k', label='True Median' )
ax.errorbar( mb, 50, xerr=sb, fmt='bo', label='BS-standard error')
ax.axvline( bs_perc[0], lw=2, c='r', label='BS-percentile')
ax.axvline( bs_perc[1], lw=2, c='r')
ax.axvline( bs_bca[0], lw=2, c='g', ls='--', label='BCA-interval')
ax.axvline( bs_bca[1], lw=2, c='g', ls='--')
ax.legend()
```

Code Block 3.8: Example of different boostrap interval estimates.

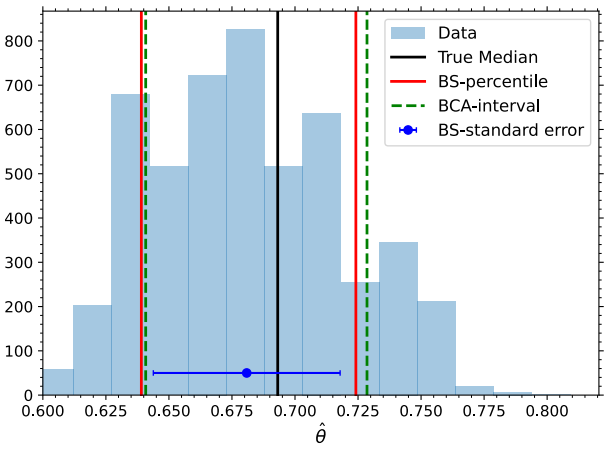

Figure 3.32: Demonstration of different bootstrap interval estimates.