Minuit

- The work-horse fitter in HEP for 30+ years
- Originally written in Fortran in the 70s
- C++ interface in the 90s, then full rewrite in C++ (Minuit2)
- Used by ROOT and ROOFIT behind the scenes
- Now also a python interface available (iMinuit) with nice support for jupyter notebooks

What does Minuit need from you.

Based on what we learnt from last time:

- A "cost function" to minimize (eg NLL or χ^2) as a function of parameters and data
- An initial guess of the parameters
- A value of Δ(cost-function) to report uncertainties (eg, for 1σ: 0.5 for NLL and 1.0 for χ^2)
- Function that gives derivative of "cost function" wrt to parameters
  - Not really necessary, can be calculated numerically internally by Minuit
- Pick minimizer to use
  - Workhorse is MIGRAD, reports "parabolic" covariance matrix at the end
  - MINOS after MIGRAD to calculate non parabolic errors

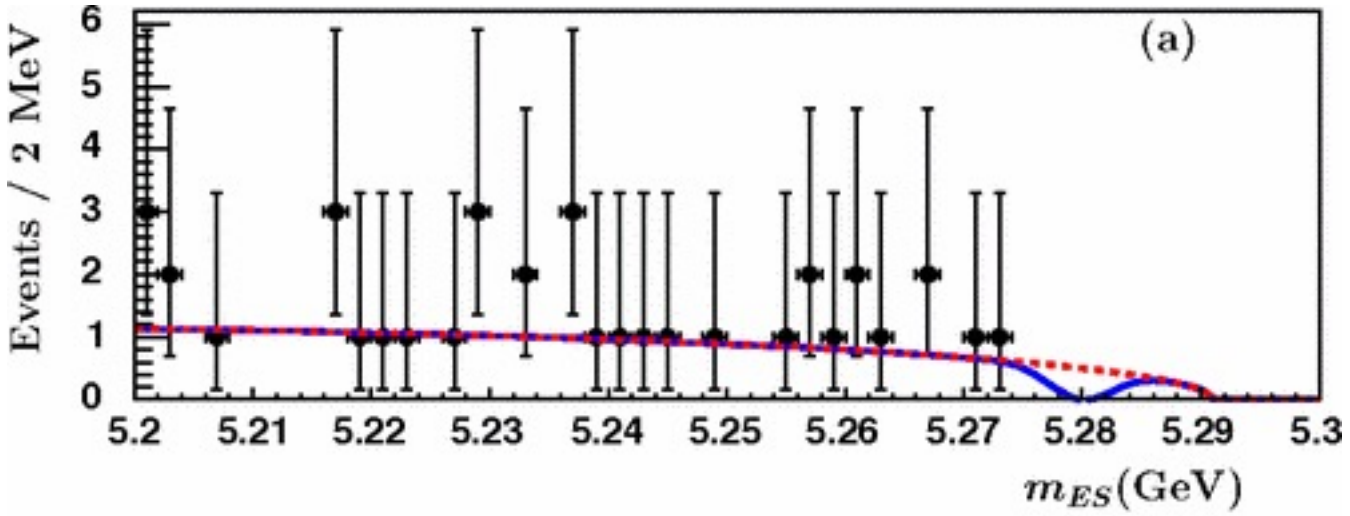
What else can you do with Minuit

- Automated plots of cost-function scans, contour plots
- Can fix some fit parameter(s), ie, turn them into constants
  - Can also be done in steps, ie, minimize once, then fix a parameter(s), then minimize again
- Can fine tune some parameters about internals of Minuit, eg “Strategy”, condition on when to call it “good enough” (“EDM”=estimated distance to minimum)
  - Better to leave it as defaults
- Impose limits on parameters, eg, $a<P<b$
  - Convenient to eg avoid unphysical regions of parameter space
    - Although I think that if your minimization is going too far into the unphysical region you probably would like to know?
  - Convenient to avoid problems with taking logs of negative quantities
    - eg: NLL includes calculations of logs of pdf, in unphysical region pdf can go negative, eg, pdf of a S+B fit

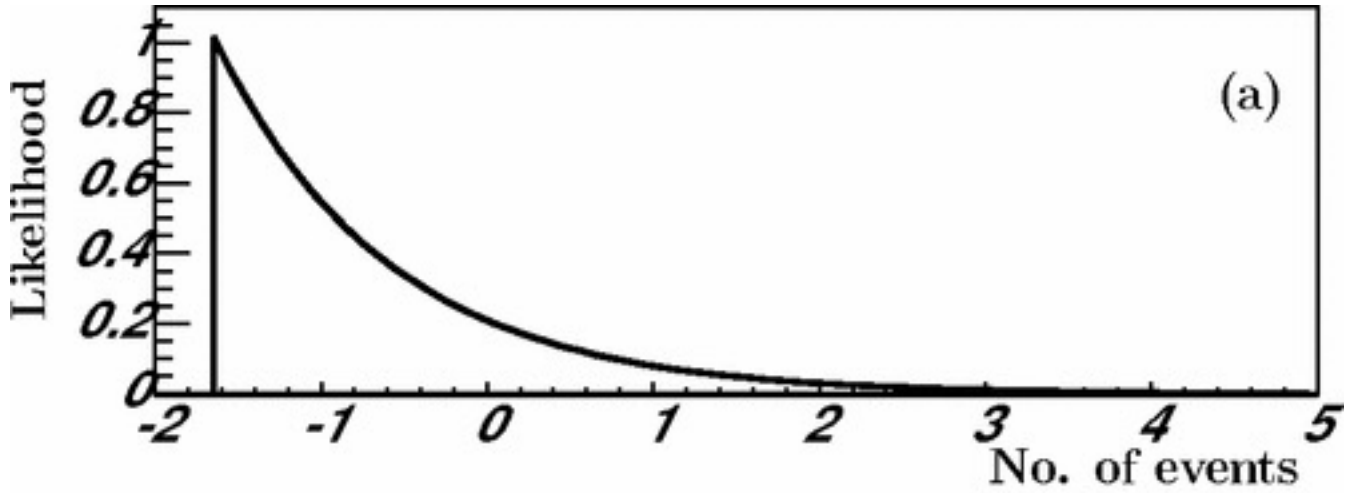
    f(mass) = { S*Gauss(mass|μ,σ) + B*f_{BG}(x) } / (S+B)

    f(mass) can go negative if S is too negative

Example, Phys. Rev. D 73, 011103(R)



Argus + Gaussian unbinned NLL fit
to mass distribution.
Deficit of signal events
(Gaussian component at m_{ES}=5.28 GeV)



Fit

$S = -1.7^{+0.7}_{-0.0}$

Actually fit wanted to go even lower but could not

Careful with limits

2.3.1 The transformation for parameters with limits

[intro:limits]

For variable parameters with double sided limits a (lower) and b (upper), M uses the following transformation:

$$P_{\text{int}} = \arcsin\left(2\,\frac{P_{\text{ext}} - a}{b-a} \quad 1\right)$$
$$P_{\text{ext}} = a + \frac{b-a}{2}\,(\sin P_{\text{int}} + 1)$$

so that the internal value P_{int} can take on any value, while the external value P_{ext} can take on values only between the lower limit a and the upper limit b. Since the transformation is necessarily non-linear, it would transform a nice linear problem into a nasty non-linear one, which is the reason why limits should be avoided if not necessary. In addition, the transformation does require some computer time, so it slows down the computation a little bit, and more importantly, it introduces additional numerical inaccuracy into the problem in addition to what is introduced in the numerical calculation of the FCN value. The effects of non-linearity and numerical roundoff both become more important as the external value gets closer to one of the limits (expressed as the distance to nearest limit divided by distance between limits). The user must therefore be aware of the fact that, for example, if they put limits of $(0, 10^{10})$ on a parameter, then the values 0.0 and 1.0 will be indistinguishable to the accuracy of most machines.

For this purpose single sided limits on parameters are provided by M , with their transformation being:

Lower bound a:

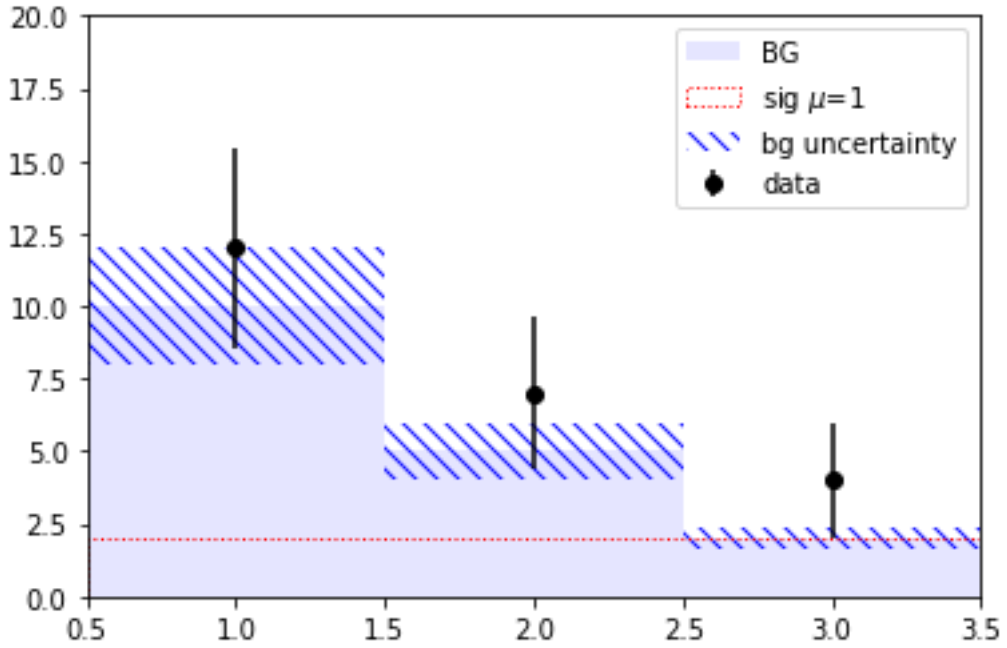
$$P_{\text{int}} = \pm\sqrt{(P_{\text{ext}} - a + 1)^2 - 1}$$
$$P_{\text{ext}} = a - 1 + \sqrt{P_{\text{int}}^2 + 1}$$

Upper bound b:

$$P_{\text{int}} = \pm\sqrt{(b - P_{\text{ext}} + 1)^2 - 1}$$
$$P_{\text{ext}} = b + 1 - \sqrt{P_{\text{int}}^2 + 1}$$

Minuit Example

- Fit a 3 bin “histogram” to μS+B
- 20% uncorrelated uncertainty on each background bin



$$\mathcal{L}(\vec{N}|\mu,\vec{B}) = \prod \frac{(\mu S_i + B_i)^{N_i} e^{-(\mu S_i + B_i)}}{N_i!} \times \mathrm{pdf}(B_i)$$

$$\mathrm{NLL} = \sum -N_i \log(\mu S_i + B_i) + \mu S_i + B_i - \log(\mathrm{pdf}(B_i))$$

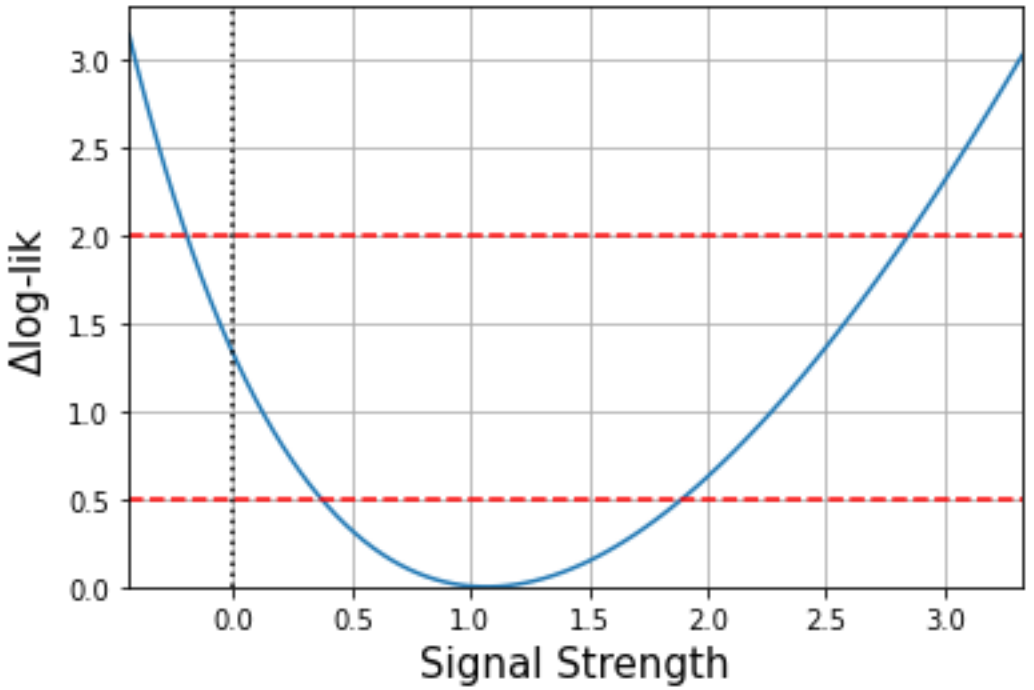
```
# This is some fake data in 3 bins
data = np.array([12, 7, 4])
sig  = np.array([ 2,  2, 2])  # signal predicted with mu=1
bg   = np.array([10,  5, 2])  # bg predicted
err  = 0.2*bg                 # bg uncertainty
```

Results

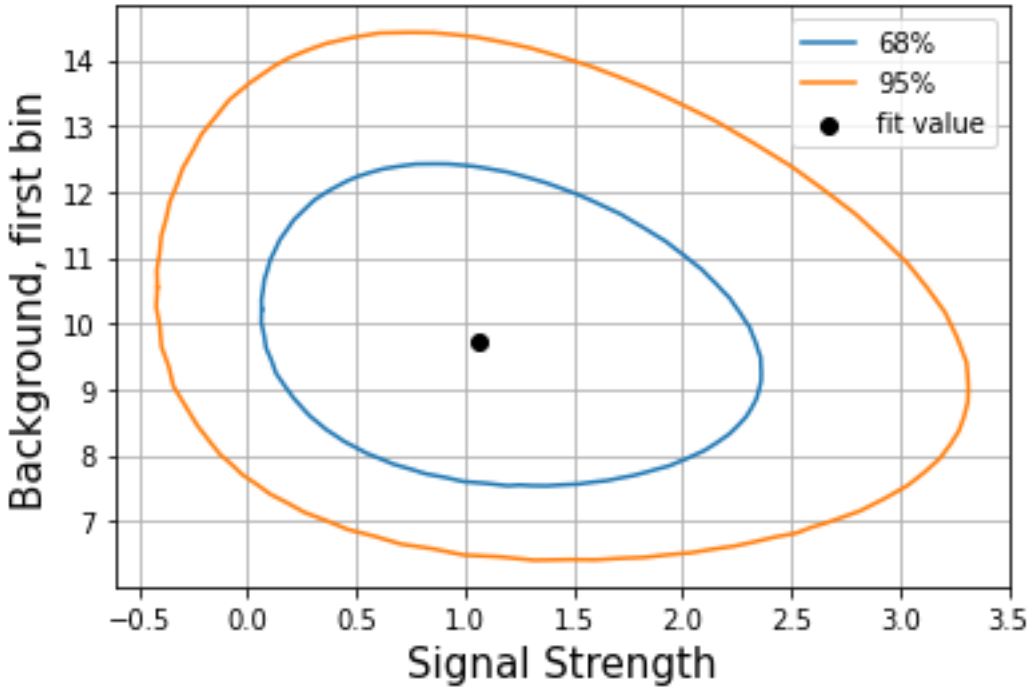
```
mu = 1.066 +/- 0.754
Minos errors:
-0.696 +0.826
```

"Parabolic error"

"Profile likelihood errors"



Just for fun….
Not an elliptical



HTML link

Jupyter link