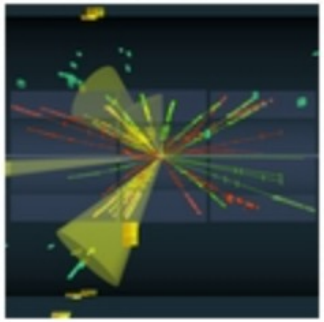
Parameter Estimation
Hands-on Session



Taller de Altas Energías
Benasque, Spain (online)
4,5 September 2023

CENTRO DE CIENCIAS
DE BENASQUE
PEDRO PASCUAL

http://benasque.org/2023tae/

ROYAL
HOLLOWAY
UNIVERSITY
OF LONDON

Glen Cowan
Physics Department
Royal Holloway, University of London
g.cowan@rhul.ac.uk
www.pp.rhul.ac.uk/~cowan

Introduction and materials

The exercises for parameter estimation are at

https://www.pp.rhul.ac.uk/~cowan/stat/exercises/fitting

The exercise and are described in the file fitting_exercises.pdf.

There are both python and ROOT/C++ versions.

mlFit.py	←	use this for today's exercises
histFit.py	←	binned version (uses python class)

For python, you need python 3 and install iminuit from https://pypi.org/project/iminuit/ with pip install iminuit

For ROOT you should have version 6 and C++ installed with a "cern-like" (e.g., lxplus) setup.

Prior to the exercises we will see how to obtain a confidence interval/region directly from contours of the log-likelihood.

Introduction to the exercises

Consider a pdf for continuous random variable x, (truncate and renormalize in $0 \leq x \leq x_{\max}$)

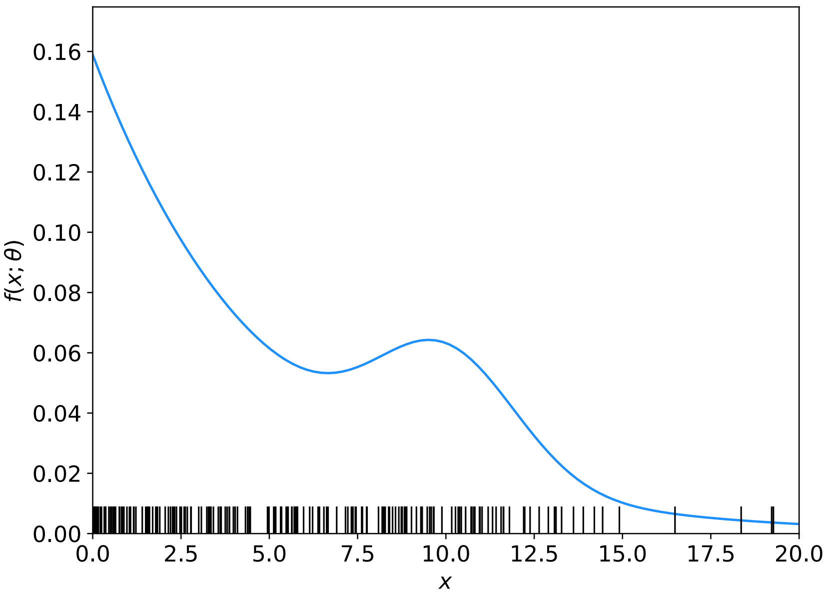
$$f(x;\theta,\xi) = \theta \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} + (1-\theta)\frac{1}{\xi} e^{-x/\xi}$$

θ = parameter of interest , gives signal rate.

Depending on context, take ξ, μ, σ as nuisance parameters or fixed.

Generate i.i.d. sample $x_1,\ldots, x_n$.

Estimate θ (and other params.)



Fitting with MINUIT (python or root/C++)

To use python, you will need to install the package iminuit (should just work with "pip install iminuit"). See:

`https://pypi.org/project/iminuit/`

Then download and run the program **`mlFit.py`** or the jupyter notebook **`mlFit.ipynb`** from

`http://www.pp.rhul.ac.uk/~cowan/stat/exercises/fitting/python`

To use C++/ROOT, download the files from

`http://www.pp.rhul.ac.uk/~cowan/stat/exercises/fitting/root`

to your work directory and build the executable program by typing **`make`** and run by typing **`./mlFit`**. This uses the class **`TMinuit`**, which is described here:

`https://root.cern.ch/doc/master/classTMinuit.html`

The instructions below refer to the python version; the corresponding steps for the C++/ROOT program are similar.

mlFit.py (also jupyter notebook mlFit.ipynb)

```python
# Example of maximum-likelihood fit with iminuit version 2.
# pdf is a mixture of Gaussian (signal) and exponential (background),
# truncated in [xMin,xMax].
# G. Cowan / RHUL Physics / December 2021

import numpy as np
import scipy.stats as stats
from scipy.stats import truncexpon
from scipy.stats import truncnorm
from scipy.stats import chi2
import iminuit
from iminuit import Minuit
import matplotlib.pyplot as plt
from matplotlib import container
plt.rcParams["font.size"] = 14
print("iminuit version:", iminuit.__version__)  # need 2.x

# define pdf and generate data
np.random.seed(seed=1234567)        # fix random seed
theta = 0.2                         # fraction of signal
mu = 10.                            # mean of Gaussian
sigma = 2.                          # std. dev. of Gaussian
xi = 5.                             # mean of exponential
xMin = 0.
xMax = 20.
```

Define the fit function

```python
def f(x, par):
    theta   = par[0]
    mu      = par[1]
    sigma   = par[2]
    xi      = par[3]
    fs = stats.truncnorm.pdf(x, a=(xMin-mu)/sigma, b=(xMax-mu)/sigma, loc=mu, scale=sigma)
    fb = stats.truncexpon.pdf(x, b=(xMax-xMin)/xi, loc=xMin, scale=xi)
    return theta*fs + (1-theta)*fb
```

Generate the data

```python
numVal = 200
xData = np.empty([numVal])
for i in range (numVal):
    r = np.random.uniform();
    if r < theta:
        xData[i] = stats.truncnorm.rvs(a=(xMin-mu)/sigma, b=(xMax-mu)/sigma, loc=mu,
            scale=sigma)
    else:
        xData[i] = stats.truncexpon.rvs(b=(xMax-xMin)/xi, loc=xMin, scale=xi)
```

Set up the fit

```python
# Function to be minimized is negative log-likelihood
def negLogL(par):
    pdf = f(xData, par)
    return -np.sum(np.log(pdf))

# Initialize Minuit and set up fit:
parin   = np.array([theta, mu, sigma, xi]) # initial values (here = true values)
parname = ['theta', 'mu', 'sigma', 'xi']
parstep = np.array([0.1, 1., 1., 1.])      # initial setp sizes
parfix  = [False, True, True, False]       # change these to fix/free parameters
parlim  = [(0.,1), (None, None), (0., None), (0., None)]    # set limits
m = Minuit(negLogL, parin, name=parname)
m.errors = parstep
m.fixed = parfix
m.limits = parlim
m.errordef = 0.5                           # errors from lnL = lnLmax - 0.5
```

Do the fit, get errors, extract results

```
# Do the fit, get errors, extract results
m.migrad()                                         # minimize -logL
MLE = m.values                                     # max-likelihood estimates
sigmaMLE = m.errors                                # standard deviations
cov = m.covariance                                 # covariance matrix
rho = m.covariance.correlation()                   # correlation coeffs.

print(r"par index, name, estimate, standard deviation:")
for i in range(m.npar):
    if not m.fixed[i]:
        print("{:4d}".format(i), "{:<10s}".format(m.parameters[i]), " = ",
         "{:.6f}".format(MLE[i]), " +/- ", "{:.6f}".format(sigmaMLE[i]))

print()
print(r"free par indices, covariance, correlation coeff.:")
for i in range(m.npar):
    if not(m.fixed[i]):
        for j in range(m.npar):
            if not(m.fixed[j]):
                print(i, j, "{:.6f}".format(cov[i,j]), "{:.6f}".format(rho[i,j]))
```

Make some plots...

Approximate confidence intervals/regions from the likelihood function

Suppose we test parameter value(s) $\boldsymbol{\theta} = (\theta_1, ..., \theta_n)$ using the ratio

$$\lambda(\boldsymbol{\theta}) = \frac{L(\boldsymbol{\theta})}{L(\hat{\boldsymbol{\theta}})} \qquad 0 \leq \lambda(\boldsymbol{\theta}) \leq 1$$

Lower $\lambda(\boldsymbol{\theta})$ means worse agreement between data and hypothesized $\boldsymbol{\theta}$. Equivalently, usually define

$$t_{\boldsymbol{\theta}} = -2 \ln \lambda(\boldsymbol{\theta})$$

so higher t_θ means worse agreement between $\boldsymbol{\theta}$ and the data.

p-value of $\boldsymbol{\theta}$ therefore $$p_{\boldsymbol{\theta}} = \int_{t_{\boldsymbol{\theta},\mathrm{obs}}}^{\infty} f(t_{\boldsymbol{\theta}}|\boldsymbol{\theta})\, dt_{\boldsymbol{\theta}}$$

need pdf

Confidence region from Wilks' theorem

Wilks' theorem says (in large-sample limit and provided certain conditions hold...)

$$f(t_{\boldsymbol{\theta}}|\boldsymbol{\theta}) \sim \chi^2_n$$

chi-square dist. with # d.o.f. = # of components in $\boldsymbol{\theta} = (\theta_1, ..., \theta_n)$.

Assuming this holds, the p-value is

$$p_{\boldsymbol{\theta}} = 1 - F_{\chi^2_n}(t_{\boldsymbol{\theta}})$$

← set equal to α

To find boundary of confidence region set $p_\theta = \alpha$ and solve for t_θ:

$$t_\theta = F^{-1}_{\chi^2_n}(1-\alpha)$$

Recall also

$$t_\theta = -2 \ln \frac{L(\theta)}{L(\hat{\theta})}$$

Confidence region from Wilks' theorem (cont.)

i.e., boundary of confidence region in $\boldsymbol{\theta}$ space is where

$$\ln L(\boldsymbol{\theta}) = \ln L(\hat{\boldsymbol{\theta}}) - \tfrac{1}{2} F^{-1}_{\chi^2_n}(1-\alpha)$$

For example, for $1-\alpha = 68.3\%$ and $n = 1$ parameter,

$$F^{-1}_{\chi^2_1}(0.683) = 1$$

and so the 68.3% confidence level interval is determined by

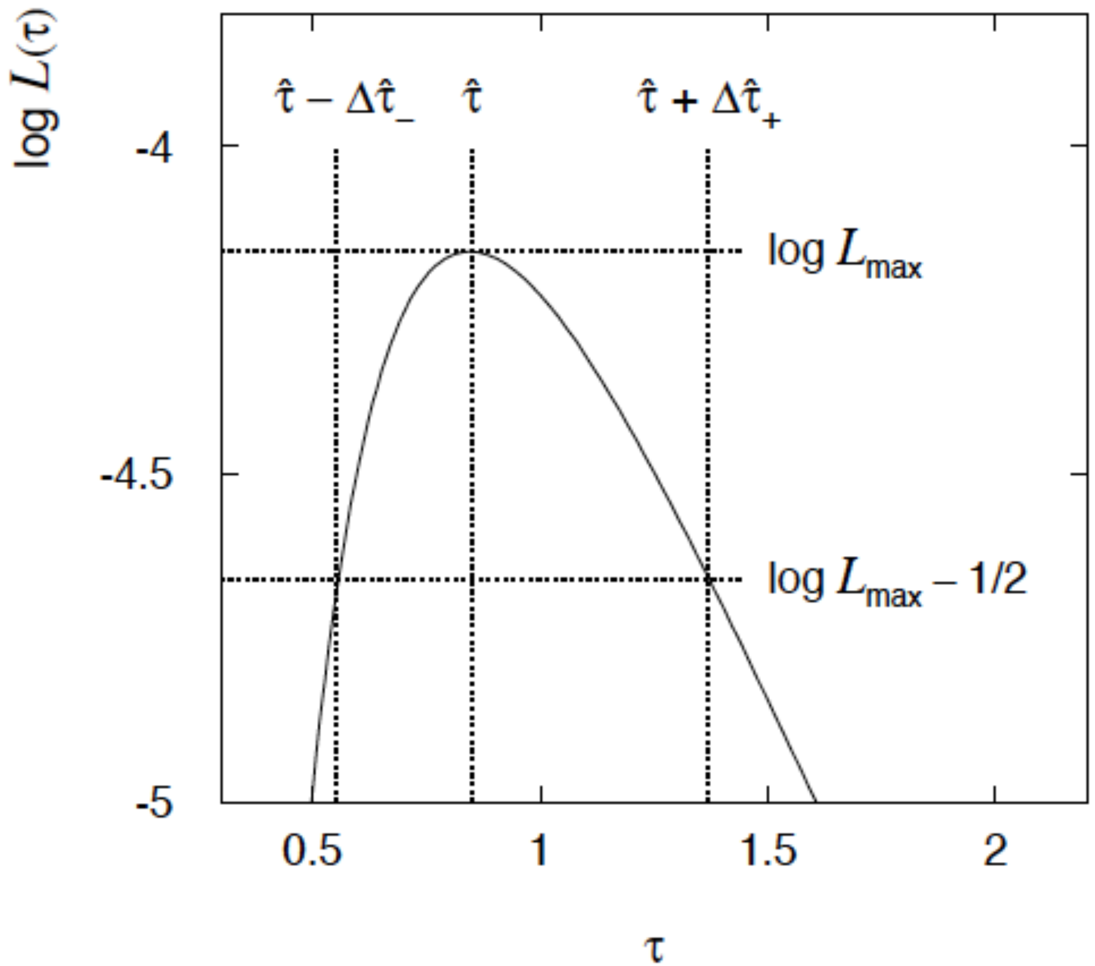
$$\ln L(\theta) = \ln L(\hat{\theta}) - \frac{1}{2}$$

Same as recipe for finding the estimator's standard deviation, i.e.,

$[\hat{\theta} - \sigma_{\hat{\theta}}, \hat{\theta} + \sigma_{\hat{\theta}}]$ is a 68.3% CL confidence interval.

Example of interval from $\ln L(\theta)$

For n=1 parameter, CL = 0.683, $Q_\alpha = 1$.



Our exponential example, now with only $n = 5$ events.

Can report ML estimate with approx. confidence interval from $\ln L_{\max} - 1/2$ as "asymmetric error bar":

$$\hat{\tau} = 0.85^{+0.52}_{-0.30}$$

Multiparameter case

For increasing number of parameters, CL $= 1 - \alpha$ decreases for confidence region determined by a given

$$Q_\alpha = F^{-1}_{\chi^2_n}(1-\alpha)$$

Q_α	$1-\alpha$				
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
1.0	0.683	0.393	0.199	0.090	0.037
2.0	0.843	0.632	0.428	0.264	0.151
4.0	0.954	0.865	0.739	0.594	0.451
9.0	0.997	0.989	0.971	0.939	0.891

Multiparameter case (cont.)

Equivalently, Q_α increases with n for a given CL $= 1 - \alpha$.

$1-\alpha$	Q_α				
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
0.683	1.00	2.30	3.53	4.72	5.89
0.90	2.71	4.61	6.25	7.78	9.24
0.95	3.84	5.99	7.82	9.49	11.1
0.99	6.63	9.21	11.3	13.3	15.1

Comment on the $\ln L = \ln L_{\max} - ½$ contour

In the lectures, we saw that the standard deviations of fitted parameters are found from the tangent lines (planes) to the contour

$$\ln L = \ln L_{\max} - \frac{1}{2}$$

A similar procedure can be used to find a "confidence region" in the parameter space that will cover the true parameter with probability CL = 1 – α (the "confidence level). This uses the contour

$$\ln L = \ln L_{\max} - \frac{1}{2} F^{-1}_{\chi^2}(1 - \alpha; N) \; ,$$

N = number of parameters

If you want the contour $\ln L = \ln L_{\max} - ½$ in iminuit, you need to choose CL (= 1 – α) such that $F_{\chi 2}{}^{-1}(1 - \alpha, N) = 1$, i.e.,

$$\mathrm{CL} = F_{\chi^2}(1; N) = \texttt{stats.chi2.cdf(1.,N)}$$

ln$L$ in a class, binned data,...

Sometimes it is convenient to have the function being minimized as a method of a class. An example of this is shown in the program histFit.py (in same directory), which does the same fit as in mlFit but with a histogram of the data: