Statistics for Particle Physicists

Lecture 4: Systematic uncertainties, etc.

Summer Student Lectures
CERN
8 – 10 July 2026

https://indico.cern.ch/event/1616585/timetable/

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

Outline

Lecture 1: Introduction, probability,

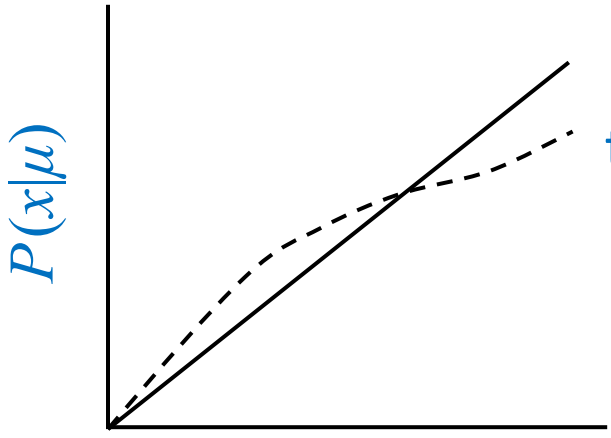
Lecture 2: Parameter estimation

Lecture 3: Hypothesis tests

→ Lecture 4: Systematic uncertainties and further examples

Systematic uncertainties and nuisance parameters

In general, our model of the data is not perfect:



model: $P(x|\mu) = \mu x$

truth: $P(x|\mu) = \mu x + \alpha x^2 + \beta x^3 + \cdots$

Can improve model by including additional adjustable parameters.

$$P(x|\mu) \to P(x|\mu, \boldsymbol{\theta})$$

Nuisance parameter ↔ systematic uncertainty. Some point in the parameter space of the enlarged model should be "true".

Presence of nuisance parameter decreases sensitivity of analysis to the parameter of interest (e.g., increases variance of estimate).

Profile Likelihood

Suppose we have a likelihood $L(\boldsymbol{\mu},\boldsymbol{\theta}) = P(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\theta})$ with N parameters of interest $\boldsymbol{\mu} = (\mu_1,..., \mu_N)$ and M nuisance parameters $\boldsymbol{\theta} = (\theta_1,..., \theta_M)$. The "profiled" (or "constrained") values of $\boldsymbol{\theta}$ are:

$$\hat{\hat{\boldsymbol{\theta}}}(\boldsymbol{\mu}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, L(\boldsymbol{\mu}, \boldsymbol{\theta})$$

and the profile likelihood is: $L_{\mathrm{p}}(\boldsymbol{\mu}) = L(\boldsymbol{\mu}, \hat{\hat{\boldsymbol{\theta}}})$

The profile likelihood depends only on the parameters of interest; the nuisance parameters are replaced by their profiled values.

The profile likelihood can be used to obtain confidence intervals/regions for the parameters of interest in the same way as one would for all of the parameters from the full likelihood.

Profile Likelihood Ratio – Wilks theorem

Goal is to test/reject regions of $\boldsymbol{\mu}$ space (param. of interest).

Rejecting a point $\boldsymbol{\mu}$ should mean $p_{\boldsymbol{\mu}} \leq \alpha$ for all possible values of the nuisance parameters $\boldsymbol{\theta}$.

Test $\boldsymbol{\mu}$ using the "profile likelihood ratio": $\lambda(\boldsymbol{\mu}) = \dfrac{L(\boldsymbol{\mu}, \hat{\hat{\boldsymbol{\theta}}})}{L(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\theta}})}$

Let $t_{\boldsymbol{\mu}} = -2\ln\lambda(\boldsymbol{\mu})$. Wilks' theorem says in large-sample limit:

$$t_{\boldsymbol{\mu}} \sim \text{chi-square}(N)$$

where the number of degrees of freedom is the number of parameters of interest (components of $\boldsymbol{\mu}$). So p-value for $\boldsymbol{\mu}$ is

$$p_{\boldsymbol{\mu}} = \int_{t_{\mu,\text{obs}}}^{\infty} f(t_{\boldsymbol{\mu}}|\boldsymbol{\mu}, \boldsymbol{\theta})\, dt_{\boldsymbol{\mu}} = 1 - F_{\chi^2_N}(t_{\boldsymbol{\mu},\text{obs}})$$

Profile Likelihood Ratio – Wilks theorem (2)

If we have a large enough data sample to justify use of the asymptotic chi-square pdf, then if $\boldsymbol{\mu}$ is rejected, it is rejected for any values of the nuisance parameters.

The recipe to get confidence regions/intervals for the parameters of interest at CL = 1 – α is thus the same as before, simply use the profile likelihood:

$$\ln L_{\rm p}(\boldsymbol{\mu}) = \ln L_{\rm max} - \frac{1}{2} F^{-1}_{\chi^2_N}(1-\alpha)$$

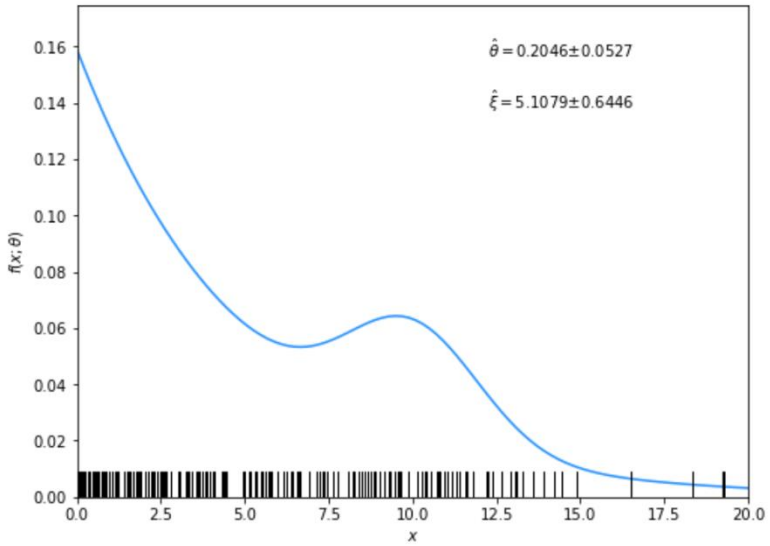
where the number of degrees of freedom N for the chi-square quantile is equal to the number of parameters of interest.

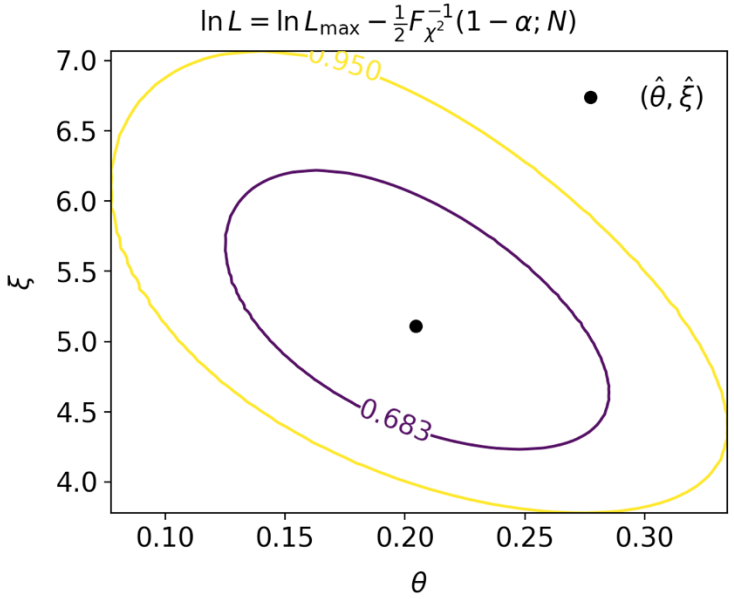
If the large-sample limit is not justified, then use e.g. Monte Carlo to get distribution of $t_{\boldsymbol{\mu}}$.

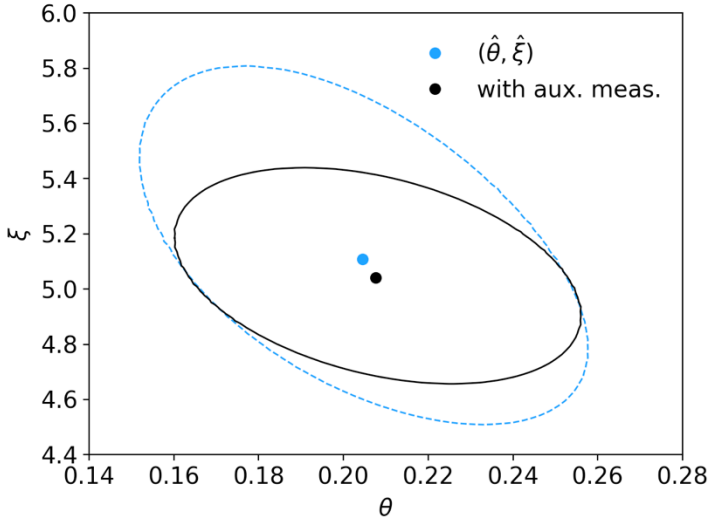
Example: mlFit.py

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

$\longrightarrow$ i.i.d. sample, n = 200







Fit results, confidence regions,... (see https://www.pp.rhul.ac.uk/~cowan/stat/exercises/cowan_stat_exercises.pdf and links therein)

Example: fitting a straight line

Data: (x_i, y_i, σ_i) , $i = 1, \ldots, n$.

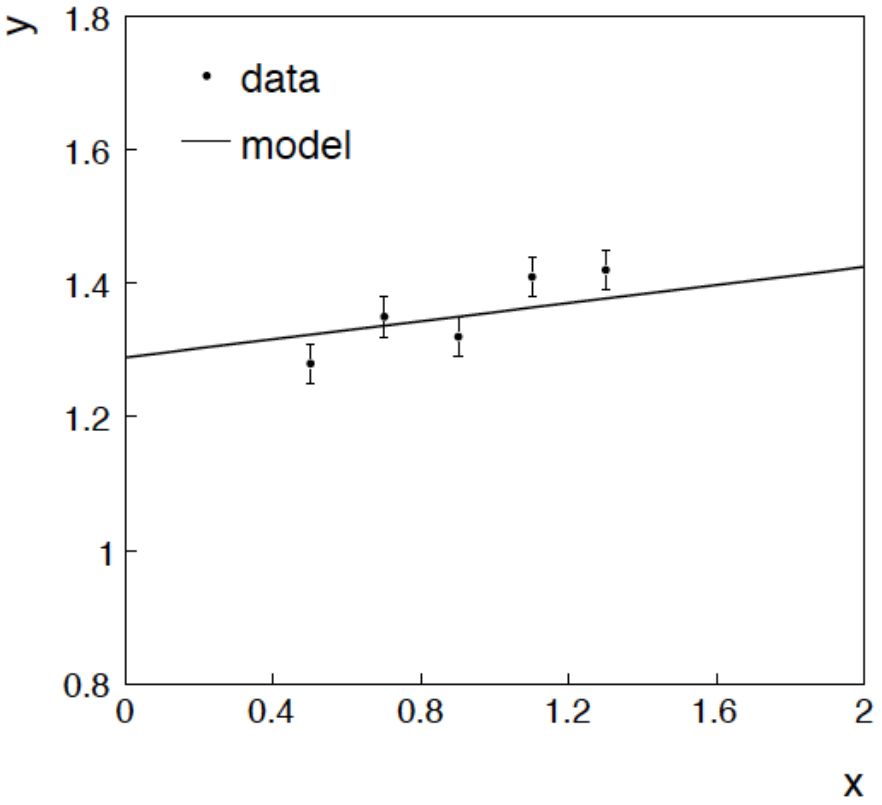
Model: y_i independent and all follow $y_i \sim \text{Gauss}(\mu(x_i), \sigma_i)$

$$\mu(x; \theta_0, \theta_1) = \theta_0 + \theta_1 x \ ,$$

assume x_i and σ_i known.

Goal: estimate θ_0

Here suppose we don't care about θ_1 (example of a "nuisance parameter")



Maximum likelihood fit with Gaussian data

In this example, the y_i are assumed independent, so the likelihood function is a product of Gaussians:

$$L(\theta_0, \theta_1) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{1}{2}\frac{(y_i - \mu(x_i; \theta_0, \theta_1))^2}{\sigma_i^2}\right] ,$$

Maximizing the likelihood is here equivalent to minimizing

$$\chi^2(\theta_0, \theta_1) = -2 \ln L(\theta_0, \theta_1) + \text{const} = \sum_{i=1}^{n} \frac{(y_i - \mu(x_i; \theta_0, \theta_1))^2}{\sigma_i^2} .$$

i.e., for Gaussian data, ML same as Method of Least Squares (LS)

θ_1 known a priori

$$L(\theta_0) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{1}{2}\frac{(y_i - \mu(x_i;\theta_0,\theta_1))^2}{\sigma_i^2}\right] .$$

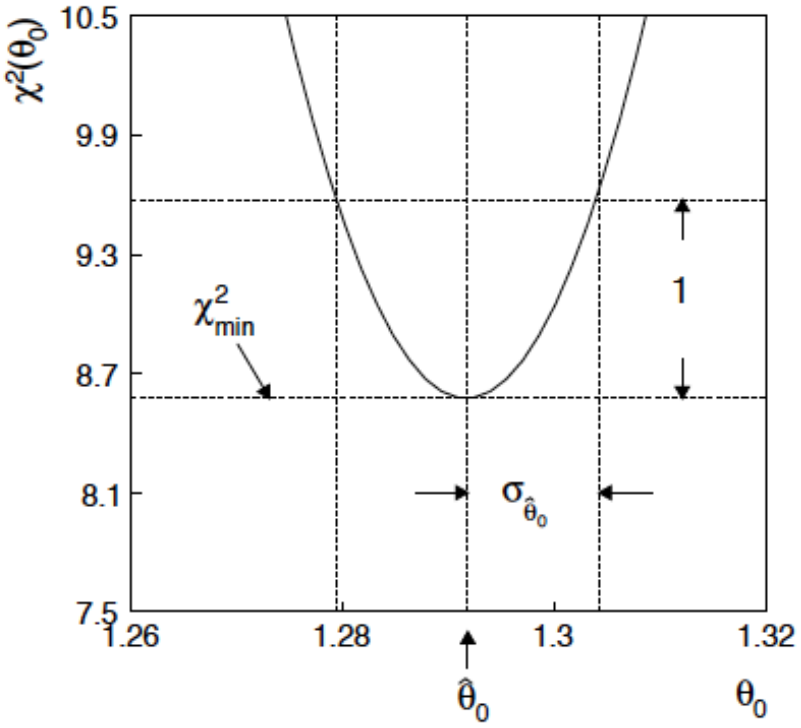
$$\chi^2(\theta_0) = -2\ln L(\theta_0) + \text{const} = \sum_{i=1}^{n} \frac{(y_i - \mu(x_i;\theta_0,\theta_1))^2}{\sigma_i^2} .$$

For Gaussian y_i, ML same as LS

Minimize $\chi^2 \rightarrow$ estimator $\hat{\theta}_0$.

Come up one unit from χ^2_{min}

to find $\sigma_{\hat{\theta}_0}$.



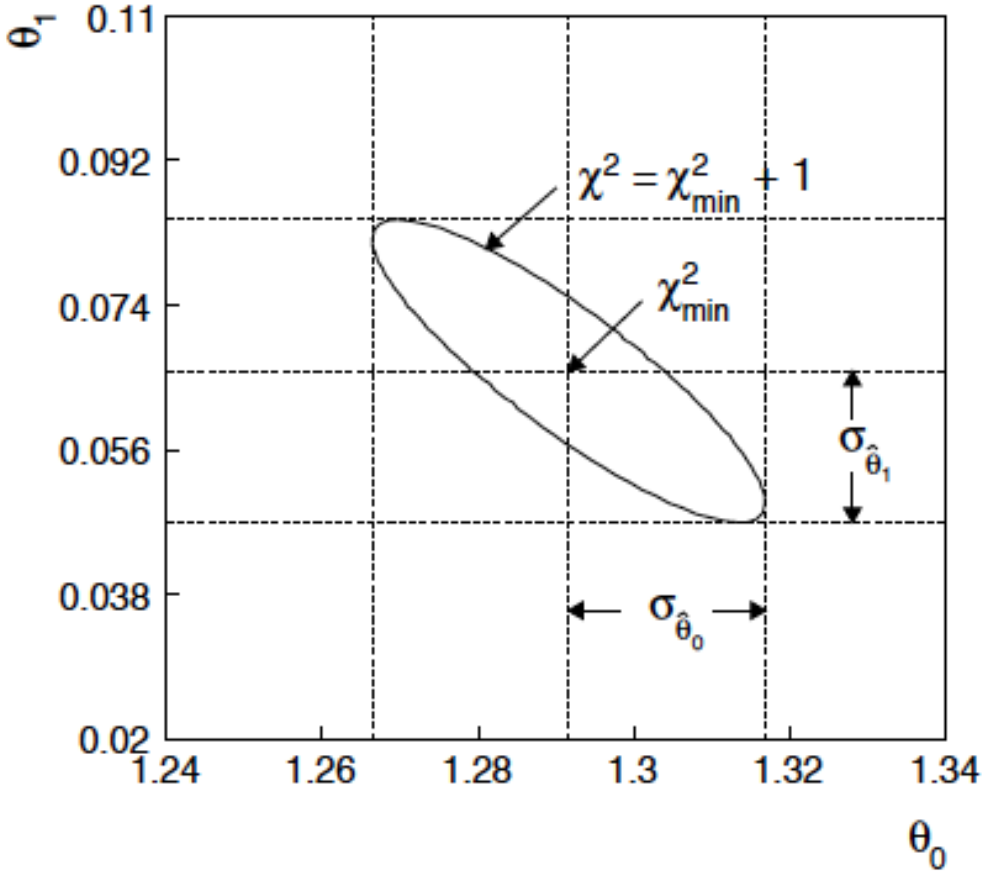
ML (or LS) fit of θ_0 and θ_1

$$\chi^2(\theta_0, \theta_1) = -2 \ln L(\theta_0, \theta_1) + \text{const} = \sum_{i=1}^{n} \frac{(y_i - \mu(x_i; \theta_0, \theta_1))^2}{\sigma_i^2} \, .$$

Standard deviations from tangent lines to contour

$$\chi^2 = \chi^2_{\text{min}} + 1 \, .$$

Correlation between $\hat{\theta}_0$, $\hat{\theta}_1$ causes errors to increase.

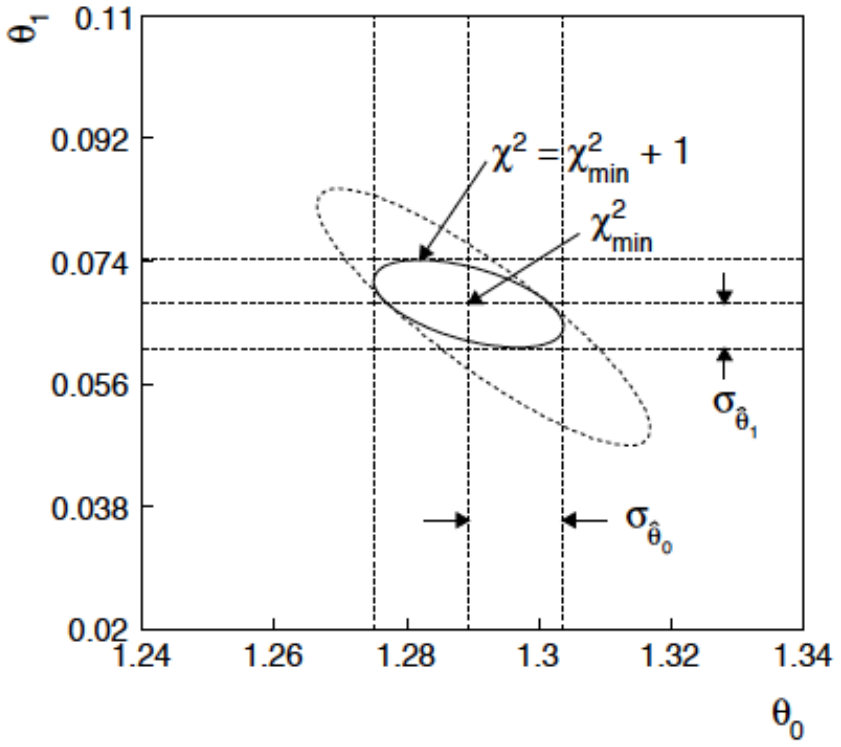


If we have a measurement $t_1 \sim \text{Gauss}\,(\theta_1, \sigma_{t1})$

$$L(\theta_0, \theta_1) = \frac{1}{\sqrt{2\pi}\sigma_t} e^{-(t_1-\theta_1)^2/2\sigma_{t_1}^2} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{1}{2}\frac{(y_i - \mu(x_i;\theta_0,\theta_1))^2}{\sigma_i^2}\right]$$

$$\chi^2(\theta_0, \theta_1) = \sum_{i=1}^{n} \frac{(y_i - \mu(x_i;\theta_0,\theta_1))^2}{\sigma_i^2} + \frac{(t_1 - \theta_1)^2}{\sigma_{t_1}^2}$$

The information on θ_1 improves accuracy of $\hat{\theta}_0$.



Reminder of Bayesian approach

In Bayesian statistics we can associate a probability with a hypothesis, e.g., a parameter value θ.

Interpret probability of θ as 'degree of belief' (subjective).

Need to start with 'prior pdf' $\pi(\theta)$, this reflects degree of belief about θ before doing the experiment.

Our experiment has data x, $\rightarrow$ likelihood $L(x|\theta)$.

Bayes' theorem tells how our beliefs should be updated in light of the data x:

$$p(\theta|x) = \frac{L(x|\theta)\pi(\theta)}{\int L(x|\theta')\pi(\theta')\,d\theta'} \propto L(x|\theta)\pi(\theta)$$

Posterior pdf $p(\theta|x)$ contains all our knowledge about θ.

Bayesian approach: $y_i \sim \text{Gauss}(\mu(x_i;\theta_0,\theta_1), \sigma_i)$

We need to associate prior probabilities with θ_0 and θ_1, e.g.,

$$\pi(\theta_0, \theta_1) = \pi_0(\theta_0)\pi_1(\theta_1)$$

← suppose knowledge of θ_0 has no influence on knowledge of θ_1

$$\pi_0(\theta_0) = \text{const.}$$

← 'non-informative', in any case much broader than $L(\theta_0)$

$$\pi_1(\theta_1) = p(\theta_1|t_1) \propto p(t_1|\theta_1)\pi_{\text{Ur}}(\theta_1) = \frac{1}{\sqrt{2\pi}\sigma_t} e^{-(t_1-\theta_1)^2/2\sigma_t^2} \times \text{const.}$$

prior after t_1, before $\boldsymbol{y}$

Ur = "primordial" prior

Likelihood for control measurement t_1

Bayesian example: $y_i \sim \text{Gauss}(\mu(x_i;\theta_0,\theta_1), \sigma_i)$

Putting the ingredients into Bayes' theorem gives:

$$p(\theta_0,\theta_1|\vec{y}) \propto \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-(y_i-\mu(x_i;\theta_0,\theta_1))^2/2\sigma_i^2} \; \pi_0 \; \frac{1}{\sqrt{2\pi}\sigma_{t_1}} e^{-(\theta_1-t_1)^2/2\sigma_{t_1}^2}$$

posterior $\propto$ likelihood $\times$ prior

Note here the likelihood only reflects the measurements $\boldsymbol{y}$.

The information from the control measurement t_1 has been put into the prior for θ_1.

We would get the same result using the likelihood $P(\boldsymbol{y},t|\theta_0,\theta_1)$ and the constant "Ur-prior" for θ_1.

Marginalizing the posterior pdf

We then integrate (marginalize) $p(\theta_0, \theta_1 | \boldsymbol{y})$ to find $p(\theta_0 | \boldsymbol{y})$:

$$p(\theta_0|\mathbf{y}) = \int p(\theta_0, \theta_1|\mathbf{y})\, d\theta_1$$

In this example we can do the integral (rare). We find

$$p(\theta_0|\mathbf{y}) = \frac{1}{\sqrt{2\pi}\sigma_{\theta_0}} e^{-(\theta_0 - \hat{\theta}_0)^2/2\sigma_{\theta_0^2}}$$

$$\hat{\theta}_0 = \text{same as MLE}$$

$$\sigma_{\theta_0} = \sigma_{\hat{\theta}_0} \qquad \text{(same as for MLE)}$$

For this example, numbers come out same as in frequentist approach, but interpretation different.

Marginalization with MCMC

Bayesian computations involve integrals like

$$p(\theta_0|x) = \int p(\theta_0, \theta_1|x)\, d\theta_1 \ .$$

often high dimensionality and impossible in closed form,
also impossible with 'normal' acceptance-rejection Monte Carlo.

Markov Chain Monte Carlo (MCMC) has revolutionized Bayesian computation.

MCMC (e.g., Metropolis-Hastings algorithm) generates correlated sequence of random numbers:

cannot use for many applications, e.g., detector MC;
effective stat. error greater than if all values independent .

Basic idea: sample multidimensional θ but look only at distribution of parameters of interest.

MCMC basics: Metropolis-Hastings algorithm

Goal: given an n-dimensional pdf $p(\boldsymbol{\theta})$ up to a proportionality constant, generate a sequence of points $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \ldots$

Proposal density $q(\boldsymbol{\theta}; \boldsymbol{\theta}_0)$ e.g. Gaussian centred about $\boldsymbol{\theta}_0$

1) Start at some point $\vec{\theta}_0$

2) Generate $\vec{\theta} \sim q(\vec{\theta}; \vec{\theta}_0)$

3) Form test ratio $\alpha = \min\left[1, \dfrac{p(\vec{\theta})q(\vec{\theta}_0; \vec{\theta})}{p(\vec{\theta}_0)q(\vec{\theta}; \vec{\theta}_0)}\right]$

4) Generate $u \sim \text{Uniform}[0, 1]$

5) If $u \leq \alpha$, $\vec{\theta}_1 = \vec{\theta}$, ← move to proposed point

else $\vec{\theta}_1 = \vec{\theta}_0$ ← old point repeated

6) Iterate

Metropolis-Hastings (continued)

This rule produces a *correlated* sequence of points (note how each new point depends on the previous one).

Still works if $p(\boldsymbol{\theta})$ is known only as a proportionality, which is usually what we have from Bayes' theorem: $p(\boldsymbol{\theta}|\boldsymbol{x}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})$.

The proposal density can be (almost) anything, but choose so as to minimize autocorrelation. Often take proposal density symmetric: $q(\boldsymbol{\theta}; \boldsymbol{\theta}_0) = q(\boldsymbol{\theta}_0; \boldsymbol{\theta})$

Test ratio is (*Metropolis*-Hastings): $\alpha = \min\left[1, \frac{p(\vec{\theta})}{p(\vec{\theta}_0)}\right]$

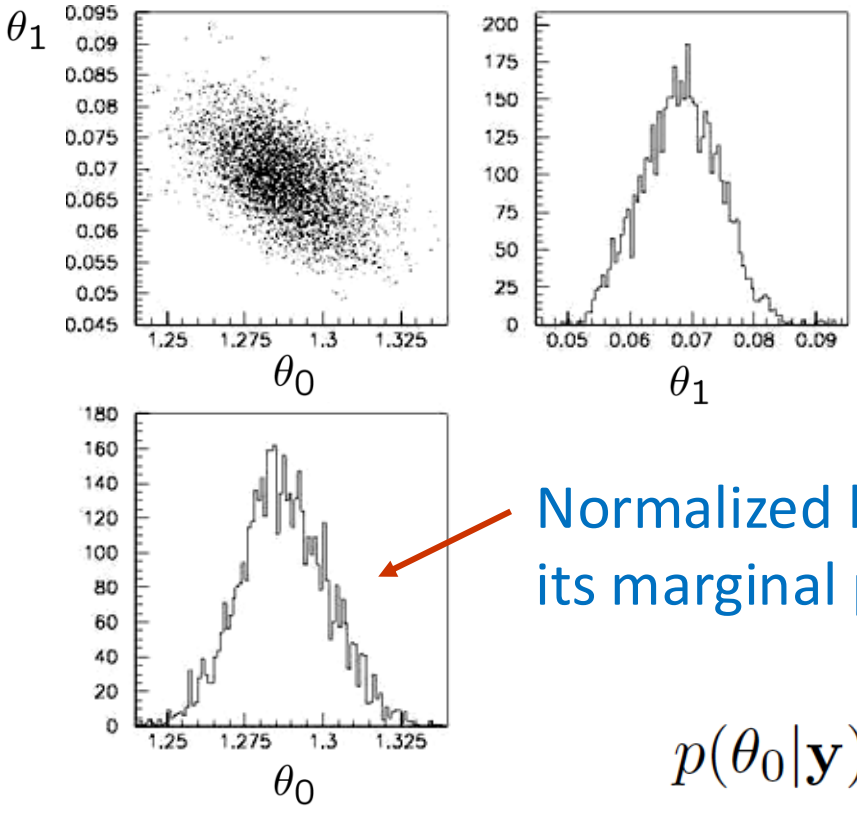
I.e. if the proposed step is to a point of higher $p(\boldsymbol{\theta})$, take it;

if not, only take the step with probability $p(\boldsymbol{\theta})/p(\boldsymbol{\theta}_0)$.

If proposed step rejected, repeat the current point.

Example: posterior pdf from MCMC

Sample the posterior pdf from previous example with MCMC:



Normalized histogram of θ_0 gives its marginal posterior pdf:

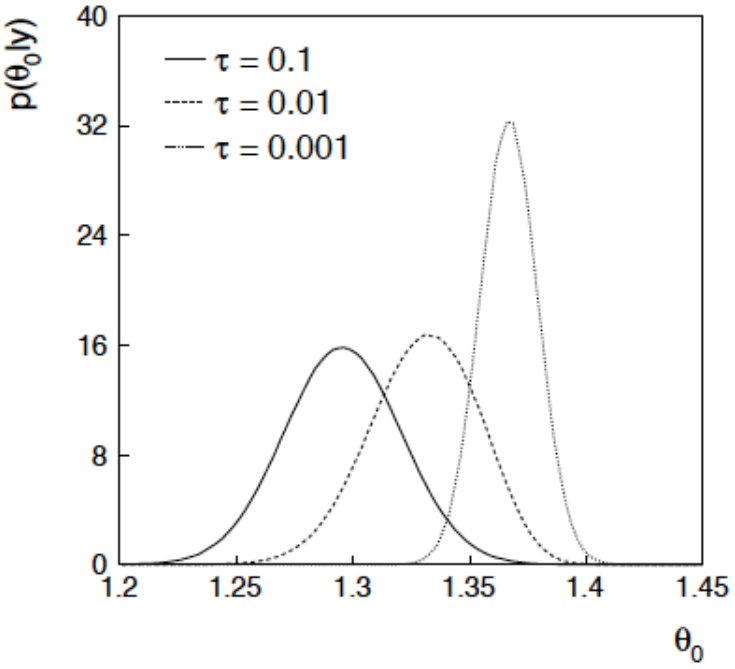
$$p(\theta_0|\mathbf{y}) = \int p(\theta_0, \theta_1|\mathbf{y})\, d\theta_1$$

Bayesian method with alternative priors

Suppose we don't have a previous measurement of θ_1 but rather, an "expert" says it should be positive and not too much greater than 0.1 or so, i.e., something like

$$\pi_1(\theta_1) = \frac{1}{\tau} e^{-\theta_1/\tau} , \quad \theta_1 \geq 0 , \quad \tau = 0.1 .$$

From this we obtain (numerically) the posterior pdf for θ_0:



This summarizes all knowledge about θ_0.

Look also at result from variety of priors.

Finally

Four lectures only enough for a brief introduction to:

Probability, frequentist & Bayesian approaches

Parameter estimation, maximum likelihood

Hypothesis tests, *p*-values, limits

Fitting with systematic uncertainties, frequentist vs. Bayesian

Many other important areas:

Statistics of Machine Learning,...

Profile likelihood ratio tests, asymptotics,...

Please take a look at the exercises in the extra slides and

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

that contain simple python programs for least squares, hypothesis tests, maximum likelihood and Bayesian fitting – enjoy!

Extra slides

Examples of maximum likelihood and confidence regions

The materials for this tutorial can be found on

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

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

The exercises for parameter estimation are done with the program mlFit.py (or with jupyter mlFit.ipynb).

The exercise does an unbinned maximum-likelihood fit and analysis of the uncertainties.

In addition there is a program histFit.py that does the same analysis but with histogram data (look at this later).

These need the iminuit package (usually “pip install iminuit”)

Gaussian signal on exponential background

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

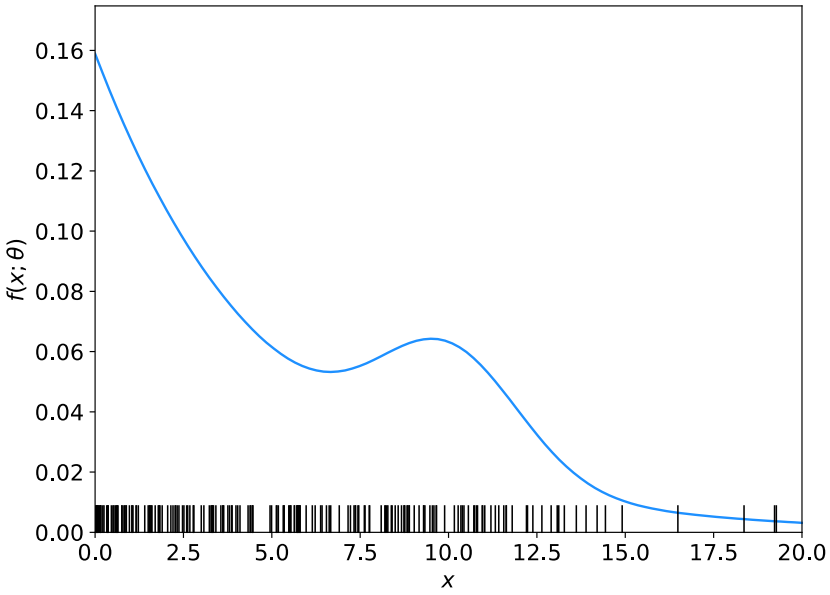
$$f(x;\theta,\xi) = \theta \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} + (1-\theta)\frac{1}{\xi} e^{-x/\xi}$$

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,..., x_n.

Estimate θ (and other params.)



A quick look at mlFit.py

```
# 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 2022


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

```
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

```
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

```
# 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']
parname_latex = [r'$\theta$', r'$\mu$', r'$\sigma$', r'$\xi$']
parstep = np.array([0.1, 1., 1., 1.])      # initial setp sizes
parfix  = [False, True, True, False]       # change these to fix/free params
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
```

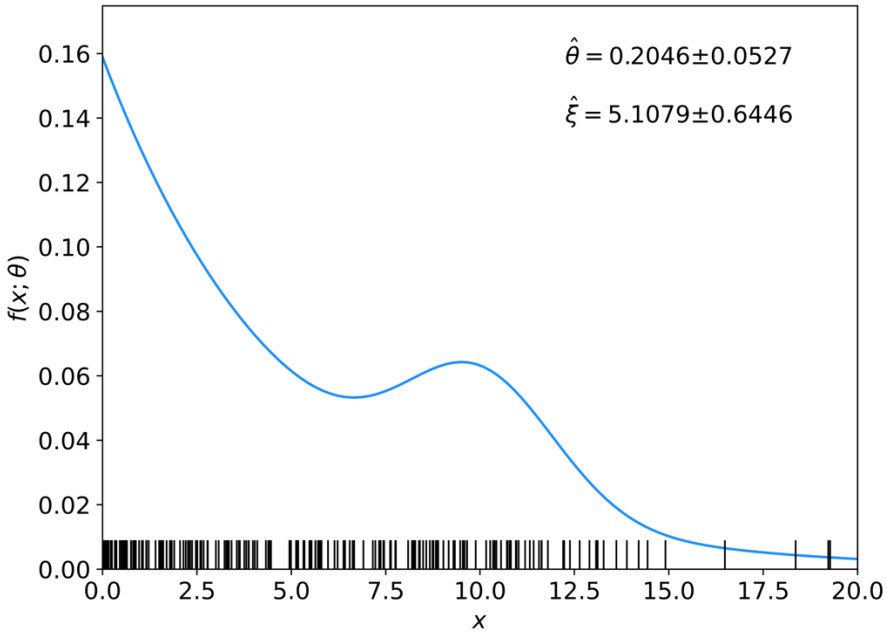
Do the fit, get errors, extract results

```
# 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...

Comments on using iminuit

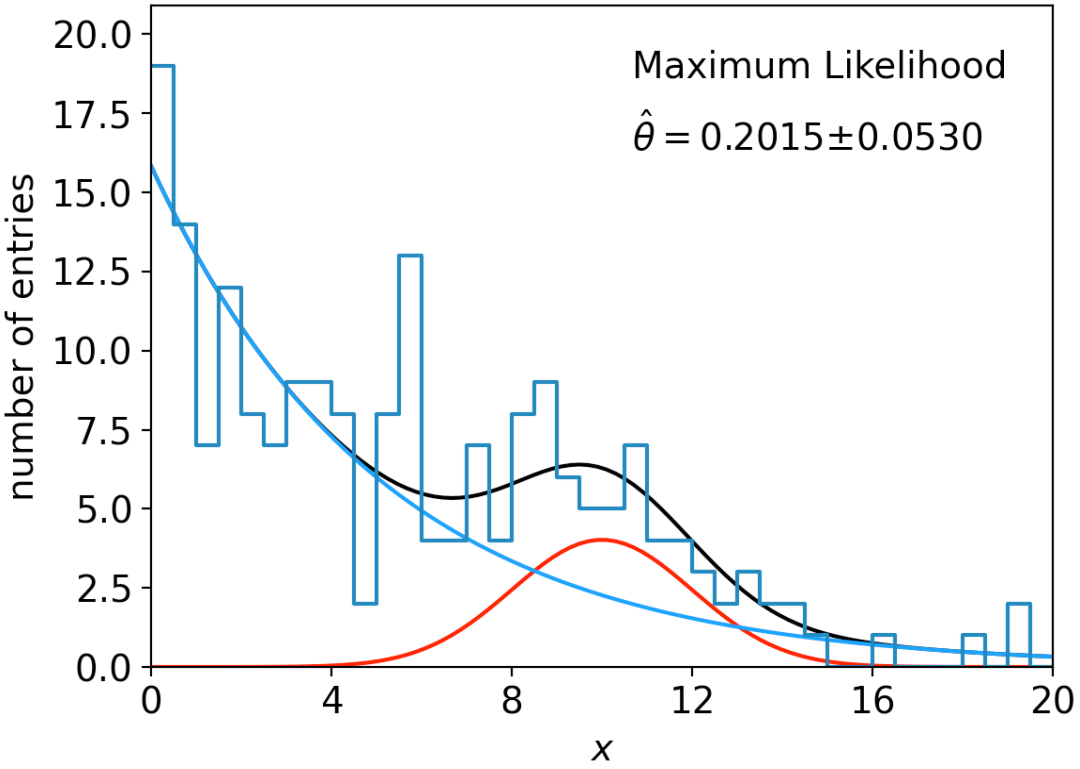
In our earlier iminuit example mlFit.py, the only argument of the log-likelihood function was the parameter array, and the data array xData entered as global (usually not a good idea):




```
def negLogL(par):
    pdf = f(xData, par)
    return -np.sum(np.log(pdf))

    ⋮

m = Minuit(negLogL, par, name=parname)
```

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, which does the same fit as in mlFit.py but with a histogram of the data:



A look at histFit.py

The global data can be avoided if we make the objective function a method of a class:

```
class ChiSquared:                    # function to be minimized

    def __init__(self, xHist, bin_edges, fitType):
        self.setData(xHist, bin_edges)
        self.fitType = fitType

    def setData(self, xHist, bin_edges):
        numVal = np.sum(xHist)
        numBins = len(xHist)
        binSize = bin_edges[1] - bin_edges[0]
        self.data = xHist, bin_edges, numVal, numBins, binSize

    def chi2LS(self, par):            # least squares
        xHist, bin_edges, numVal, numBins, binSize = self.data
        xMid = bin_edges[:numBins] + 0.5*binSize
        binProb = f(xMid, par)*binSize
        nu = numVal*binProb
        sigma = np.sqrt(nu)
        z = (xHist - nu)/sigma
        return np.sum(z**2)
```

class ChiSquared (continued)

```
def chi2M(self, par):              # multinomial maximum likelihood
    xHist, bin_edges, numVal, numBins, binSize = self.data
    xMid = bin_edges[:numBins] + 0.5*binSize
    binProb = f(xMid, par)*binSize
    nu = numVal*binProb
    lnL = 0.
    for i in range(len(xHist)):
        if xHist[i] > 0.:
            lnL += xHist[i]*np.log(nu[i]/xHist[i])
    return -2.*lnL

def __call__(self, par):
    if self.fitType == 'LS':
        return self.chi2LS(par)
    elif self.fitType == 'M':
        return self.chi2M(par)
    else:
        print("fitType not defined")
        return -1
```

Using the ChiSquared class

```
# Put data values into a histogram
numBins=40
xHist, bin_edges = np.histogram(xData, bins=numBins, range=(xMin, xMax))
binSize = bin_edges[1] - bin_edges[0]

# Initialize Minuit and set up fit:
parin   = np.array([theta, mu, sigma, xi])      # initial values (here = true)
parname = ['theta', 'mu', 'sigma', 'xi']
parstep = np.array([0.1, 1., 1., 1.])           # initial setp sizes
parfix  = [False, True, True, False]            # change to fix/free param.
parlim  = [(0.,1), (None, None), (0., None), (0., None)]
chisq = ChiSquared(xHist, bin_edges, fitType)
m = Minuit(chisq, parin, name=parname)
m.errors = parstep
m.fixed = parfix
m.limits = parlim
m.errordef = 1.0                        # errors from chi2 = chi2min + 1
```

Example of Bayesian parameter estimation

The exercise is described

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

in the file bayes_fit_exercise.pdf.

The program is in bayesFit.py or bayesFit.ipynb.

This exercise treats the same fitting problem as seen with maximum likelihood, here using the Bayesian approach.

Bayes' theorem is used to find the posterior pdf for the parameters, and these are summarized using the posterior mode (MAP estimators).

The posterior pdf is marginalized over the nuisance parameters using Markov Chain Monte Carlo.

Gaussian signal on exponential background

Same pdf as from mlFit.py (see tutorial 1) with $n = 400$ independent values of x from

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

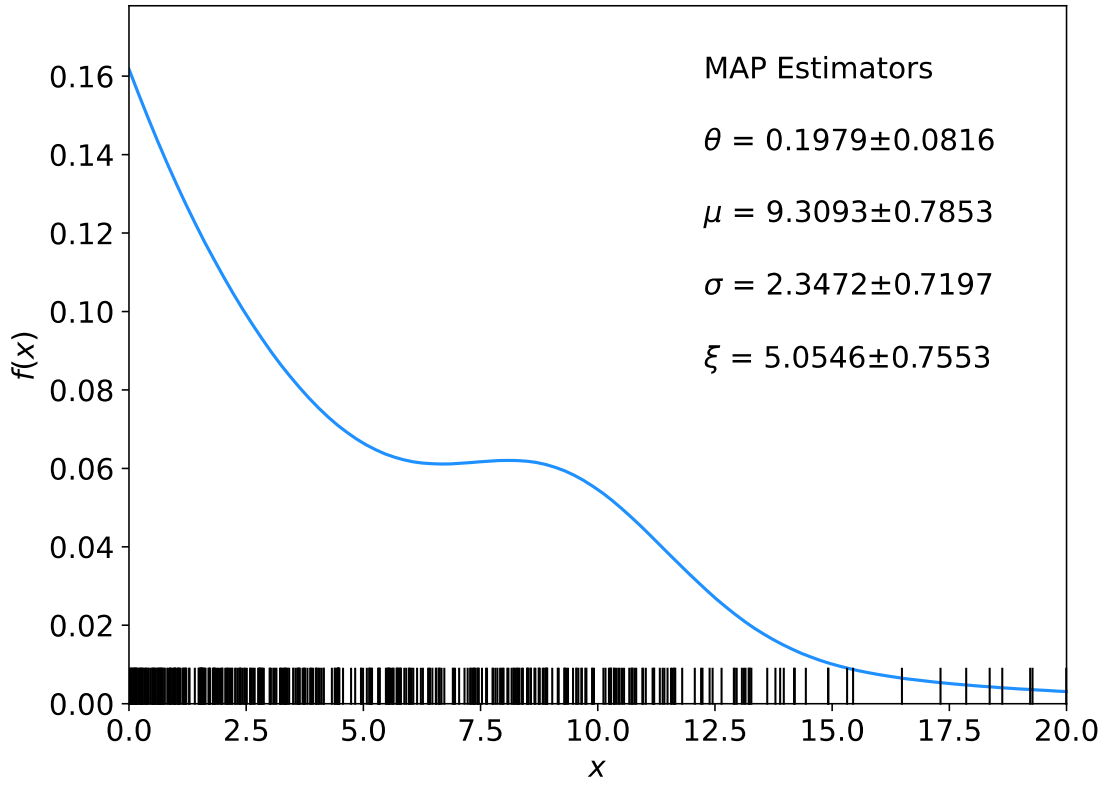
Posterior pdf for parameters $\boldsymbol{\lambda} = (\theta, \mu, \sigma, \xi)$ from Bayes theorem,

$$p(\boldsymbol{\lambda}|\mathbf{x}) \propto p(\mathbf{x}|\boldsymbol{\lambda})\pi(\boldsymbol{\lambda}) \,, \quad \text{where} \quad p(\mathbf{x}|\boldsymbol{\lambda}) = \prod_{i=1}^{n} f(x_i|\boldsymbol{\lambda})$$

At first take prior pdf constant for all parameters subject to $0 \leq \theta \leq 1$, $\sigma > 0$, $\xi > 0$ (later try different priors).

Data and MAP estimates

Maximize posterior with minuit (minimize $-\ln p(\boldsymbol{\lambda}|\boldsymbol{x})$).



Standard deviations from minuit correspond to approximating posterior as Gaussian near its peak.

Here priors constant so MAP estimates same as MLE, covariance matrix $V_{ij} = \mathrm{cov}[\theta_i, \theta_j]$ also same.

A look at bayesFit.py

Find maximum of posterior with iminuit (minimize $-\ln p(\boldsymbol{\lambda}|\boldsymbol{x})$), similar to maximum likelihood:

```
# Negative log-likelihood
def negLogL(par):
    fx = f(xData, par)
    return -np.sum(np.log(fx))

# Prior pdf
def prior(par):
    theta    = par[0]
    mu       = par[1]
    sigma    = par[2]
    xi       = par[3]
    pi_theta = 1. if theta >= 0. and theta <= 1. else 0.
    pi_mu    = 1. if mu >= 0. else 0.
    pi_sigma = 1. if sigma > 0. else 0.
    pi_xi    = 1. if xi > 0. else 0.
    piArr = np.array([pi_theta, pi_mu, pi_sigma, pi_xi])
    pi = np.product(piArr[np.array(parfix) == False])   # exclude fixed par
    return pi

# Negative log of posterior pdf
def negLogPost(par):
    return negLogL(par) - np.log(prior(par))
```

minimize with iminuit

Metropolis-Hastings algorithm in bayesFit.py

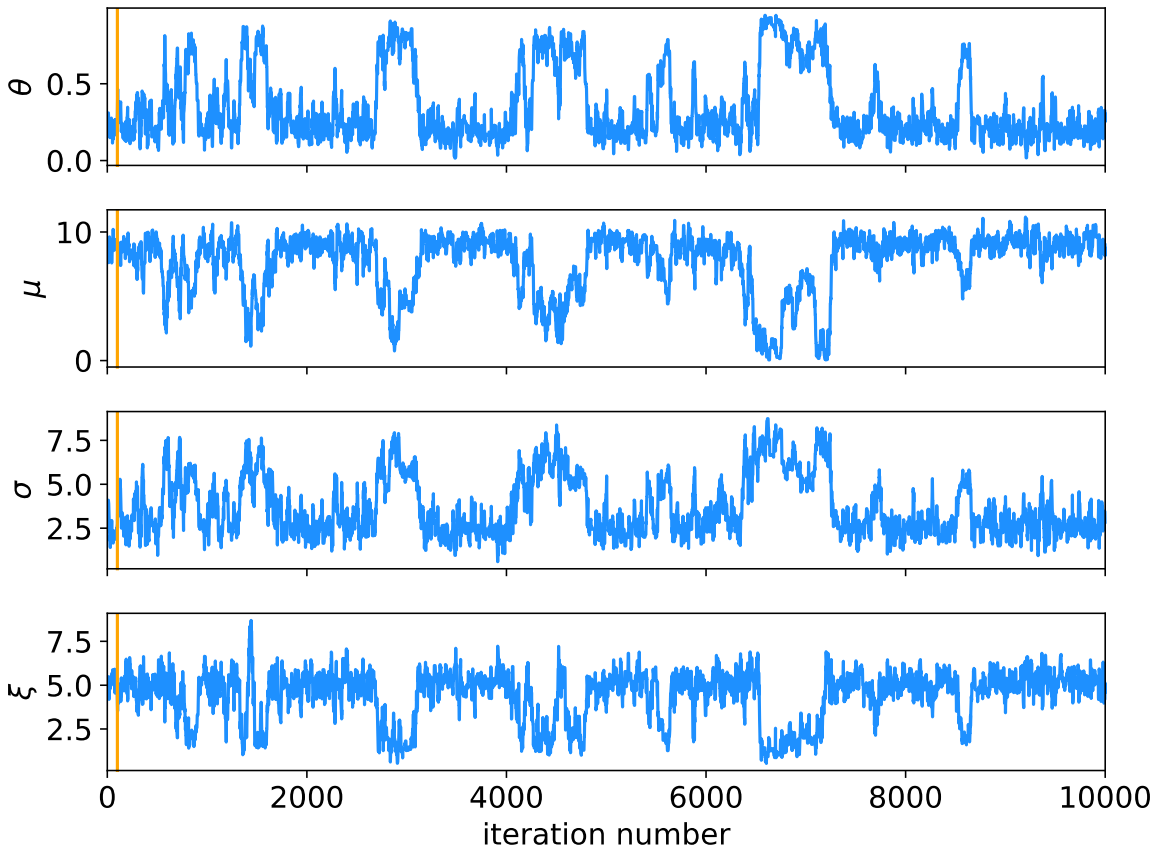
```
# Iterate with Metropolis-Hastings algorithm
chain = [np.array(MAP)]       # start point is MAP estimate
numIterate = 10000
numBurn = 100
numAccept = 0
print("Start MCMC iterations: ", end="")
while len(chain) < numIterate:
    par = chain[-1]
    log_post = -negLogL(par) + np.log(prior(par))
    par_prop = np.random.multivariate_normal(par, cov_prop)
    if prior(par_prop) <= 0:
        chain.append(chain[-1])    # never accept if prob<=0.
    else:
        log_post_prop = -negLogL(par_prop) + np.log(prior(par_prop))
        alpha = np.exp(log_post_prop - log_post)
        u = np.random.uniform(0, 1)
        if u <= alpha:
            chain.append(par_prop)
            numAccept += 1
        else:
            chain.append(chain[-1])
        if len(chain)%(numIterate/100) == 0:
            print(".", end="", flush=True)
chain = np.array(chain)
```

Try increasing number of iterations (10k runs in about 20 s).

MCMC trace plots

Take θ as parameter of interest, rest are nuisance parameters.

Marginalize by sampling posterior pdf with Metropolis-Hastings.

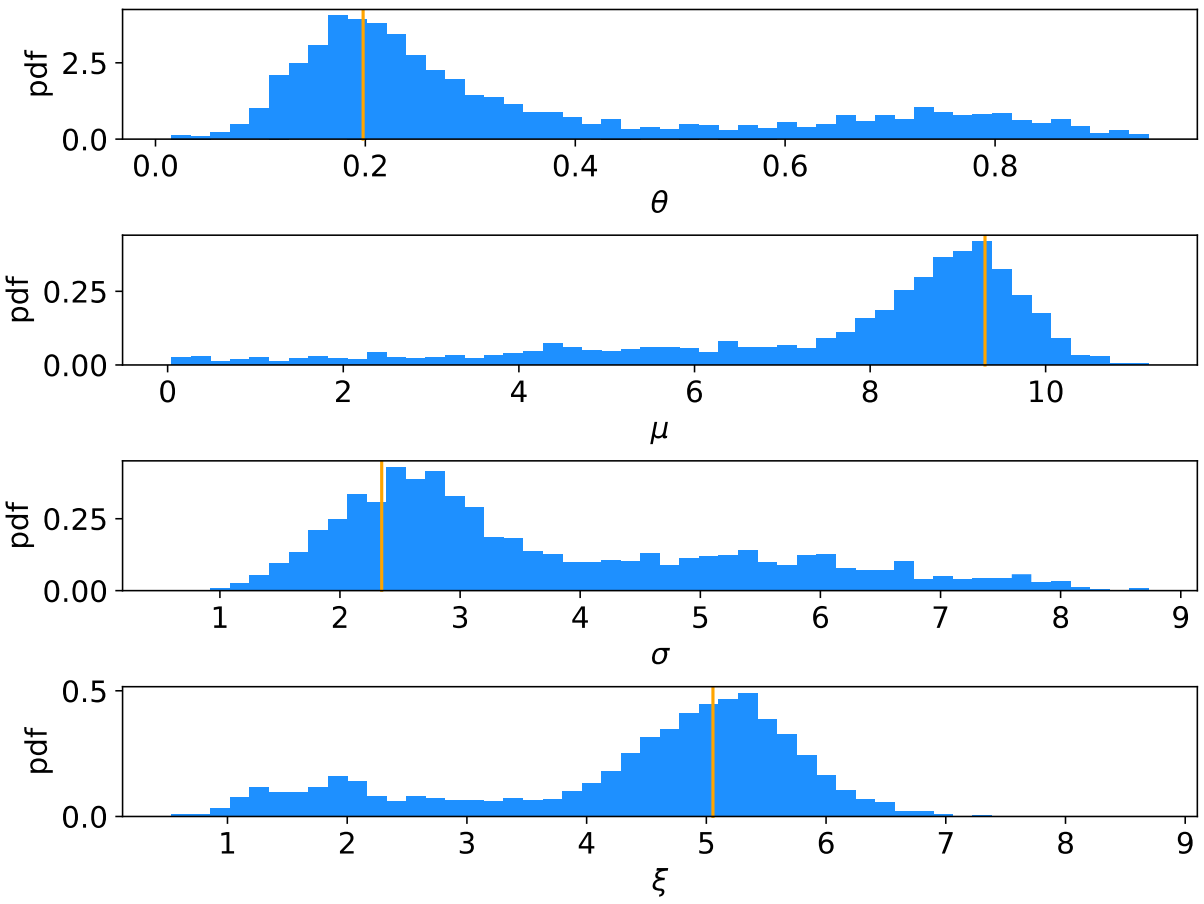


Gaussian proposal pdf, covariance $U = sV$, $s = (2.38)^2/N_{\text{par}} = 1.41$, gives acceptance probability ~ 0.24.

Here 10000 iterations (should use more).

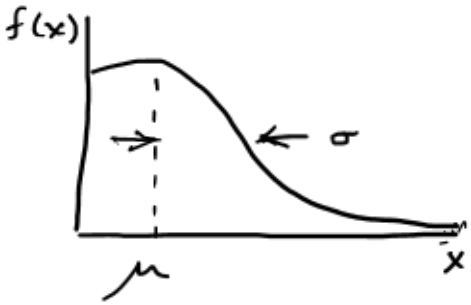
Marginal distributions

MAP estimates shown with vertical bars

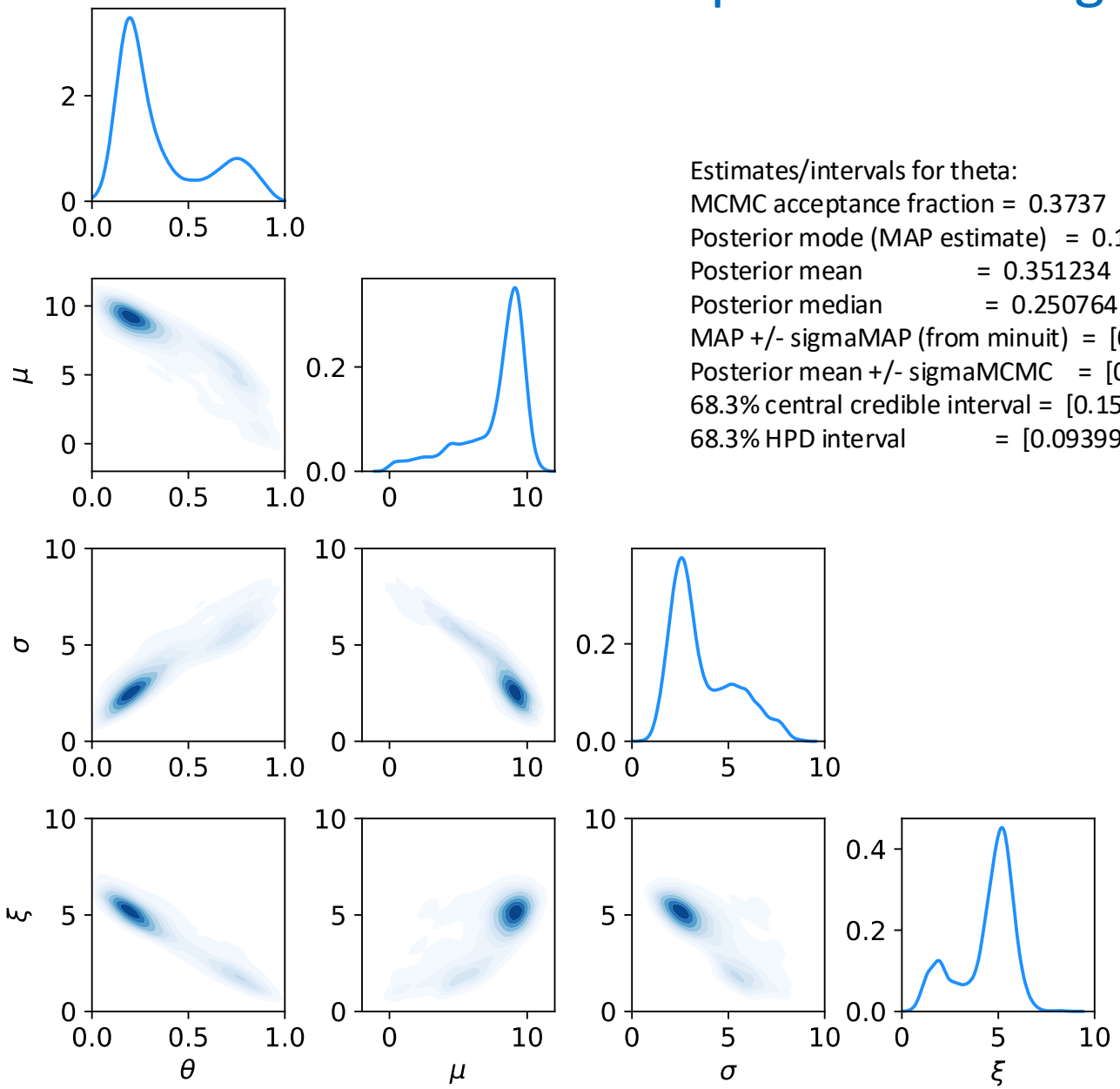


Note long tails.

Interpretation: data distribution can be approximated by Gaussian term only, (θ large, μ small) with large width ($\sigma \sim$ 4-8) and a narrow exponential ($\xi \sim$ 1-3).

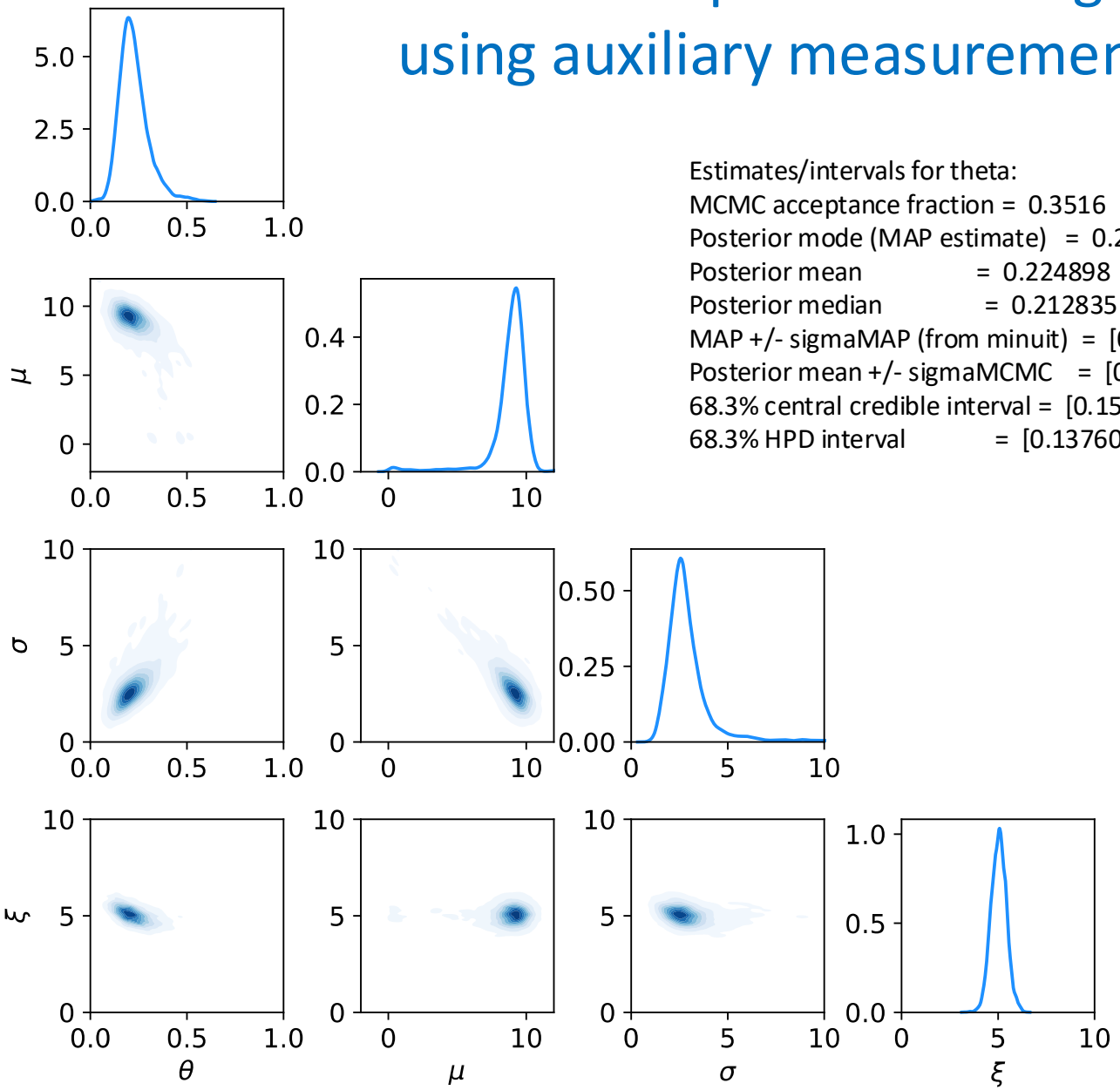


Correlation plots and marginal distributions



Estimates/intervals for theta:
MCMC acceptance fraction = 0.3737
Posterior mode (MAP estimate) = 0.197936
Posterior mean = 0.351234
Posterior median = 0.250764
MAP +/- sigmaMAP (from minuit) = [0.140056, 0.255816]
Posterior mean +/- sigmaMCMC = [0.118503, 0.583964]
68.3% central credible interval = [0.158557, 0.685847]
68.3% HPD interval = [0.093997, 0.392642]

Correlation plots and marginal distributions using auxiliary measurement for ξ



Estimates/intervals for theta:
MCMC acceptance fraction = 0.3516
Posterior mode (MAP estimate) = 0.200077
Posterior mean = 0.224898
Posterior median = 0.212835
MAP +/- sigmaMAP (from minuit) = [0.149282, 0.250873]
Posterior mean +/- sigmaMCMC = [0.148481, 0.301314]
68.3% central credible interval = [0.156736, 0.291246]
68.3% HPD interval = [0.137607, 0.267077]