Statistical Data Analysis
Material for Tutorials

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

[With thanks to Hans Dembinski, Greg Ashton, Asher Kaboth, and Enzo Canonero.]

Outline

Tutorial 1: Maximum Likelihood fitting

Confidence regions from log-likelihood

Tutorial 2: Bayesian parameter estimation

Markov Chain Monte Carlo

Tutorial 3: Hypothesis tests

Experimental sensitivity

Tutorial 4 (extra): Student's *t* average

Tutorial Materials

The exercises involve some paper-and-pencil calculations and running/modifying python programs.

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

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

Tutorial 1: Maximum Likelihood

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).

Gaussian signal on exponential background

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

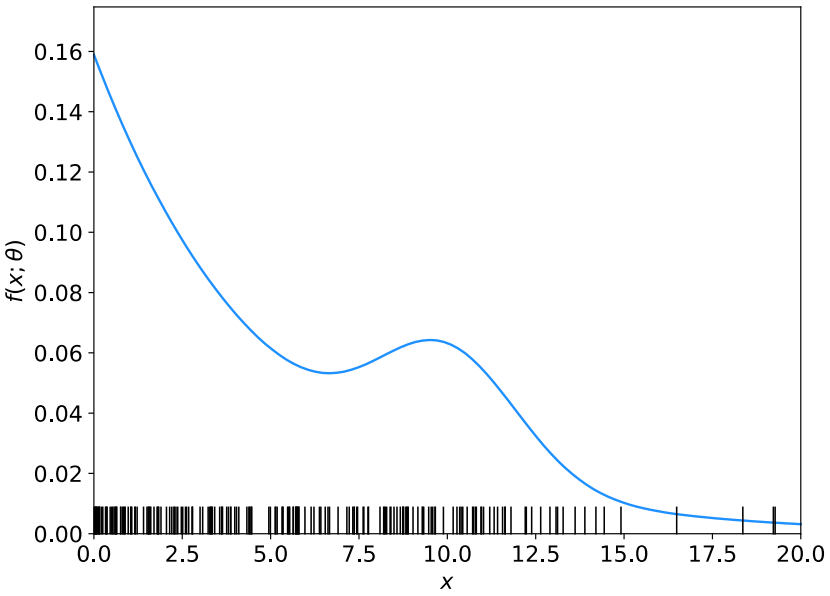
$$f(x;\theta,\xi) = \theta \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} + (1-\theta)\frac{1}{\xi} e^{-x/\xi}$$

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...

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

$$\ln L = \ln L_{\max} - \frac{1}{2} F_{\chi^2}^{-1}(1-\alpha; N) \ , \qquad N = \text{number of parameters}$$

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.,

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

Exercises on Maximum Likelihood (a)

1a) Run mlFit.py, look at the plots

1(a) By default the program `mlFit.py` fixes the parameters μ and σ, and treats only θ and ξ as free. By running the program, obtain the following plots:

- the fitted pdf with the data;
- a "scan" plot of $-\ln L$ versus θ;
- a contour of $\ln L = \ln L_{\max} - 1/2$ in the (θ, ξ) plane;
- confidence regions in the (θ, ξ) plane with confidence levels 68.3% and 95%.

From the graph of $-\ln L$ versus θ, show that the standard deviation of $\hat{\theta}$ is the same as the value printed out by the program.

From the graph of $\ln L = \ln L_{\max} - 1/2$, show that the distances from the MLEs to the tangent lines to the contour give the same standard deviations $\sigma_{\hat{\theta}}$ and $\sigma_{\hat{\xi}}$ as printed out by the program.

Exercises on Maximum Likelihood (b,c)

1b,c) show standard deviation of estimator $\sim 1/\sqrt{n}$

1(b) Recall that the inverse of the covariance matrix variance of the maximum-likelihood estimators $V_{ij} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_j]$ can be approximated in the large sample limit by

$$V_{ij}^{-1} = -E\left[\frac{\partial^2 \ln L}{\partial\theta_i \partial\theta_j}\right] = -\int \frac{\partial^2 \ln P(\mathbf{x}|\boldsymbol{\theta})}{\partial\theta_i \partial\theta_j} P(\mathbf{x}|\boldsymbol{\theta})\, d\mathbf{x}\,, \qquad (2)$$

where here $\boldsymbol{\theta}$ represents the vector of all of the parameters. Show that V_{ij}^{-1} is proportional to the sample size n and thus show that the standard deviations of the MLEs of all of the parameters decrease as $1/\sqrt{n}$. (Hint: write down the general form of the likelihood for an i.i.d. sample: $L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(x_i; \boldsymbol{\theta})$. There is no need to use the specific $f(x; \boldsymbol{\theta})$ for this problem.)

1(c) By modifying the line

```
numVal = 200
```

rerun the program for a sample size of $n = 100, 400$ and 800 events, and find in each case the standard deviation of $\hat{\theta}$. Plot (or sketch) $\sigma_{\hat{\theta}}$ versus n for $n = 100, 200, 400, 800$ and comment on how this stands in relation to what you expect.

Exercises on Maximum Likelihood (d,e)

1d,e) Investigate effect of nuisance parameters

1(d) By modifying the line

```
parfix  = [False, True, True, False]          # change these to fix/free parameters
```

find $\hat{\theta}$ and its standard deviation $\sigma_{\hat{\theta}}$ in the following four cases:

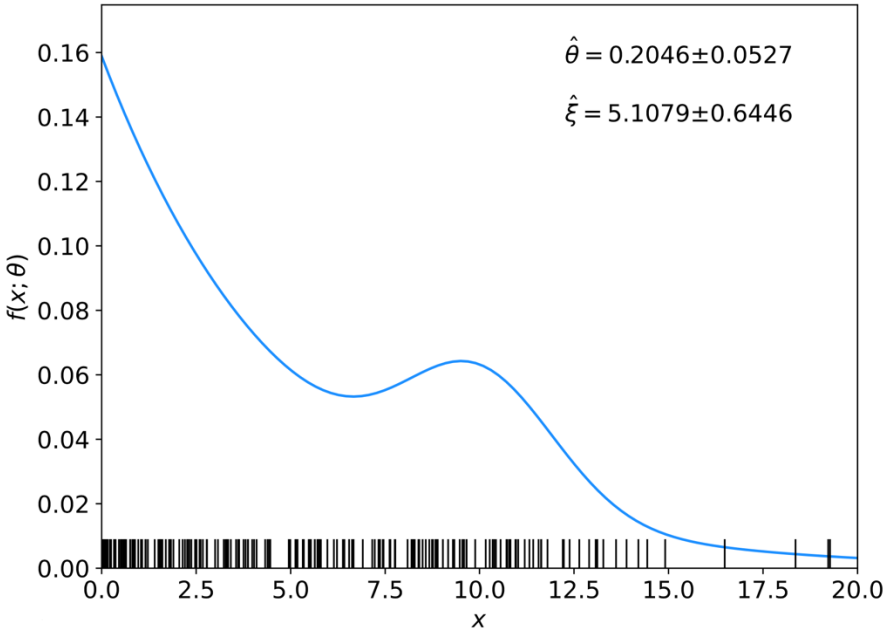
- θ free, μ, σ, ξ fixed;
- θ and ξ free, μ, σ fixed;
- θ, ξ and σ free, μ fixed;
- θ, ξ, μ and σ all free.

Comment on how the standard deviation $\sigma_{\hat{\theta}}$ depends on the number of adjustable parameters in the fit.

1(e) Consider the case where θ and ξ are adjustable and σ and μ are fixed. Suppose that one has an independent estimate u of the parameter ξ in addition to the $n = 200$ values of x. Treat u as Gaussian distributed with a mean ξ and standard deviation $\sigma_u = 0.5$ and take the observed value $u = 5$. Find the log-likelihood function that includes both the primary measurements $(x_1, \ldots, x_n)$ and the auxiliary measurement u and modify the fitting program accordingly. Investigate how the uncertainties of the MLEs for θ and ξ are affected by including u.

Comments on using iminuit

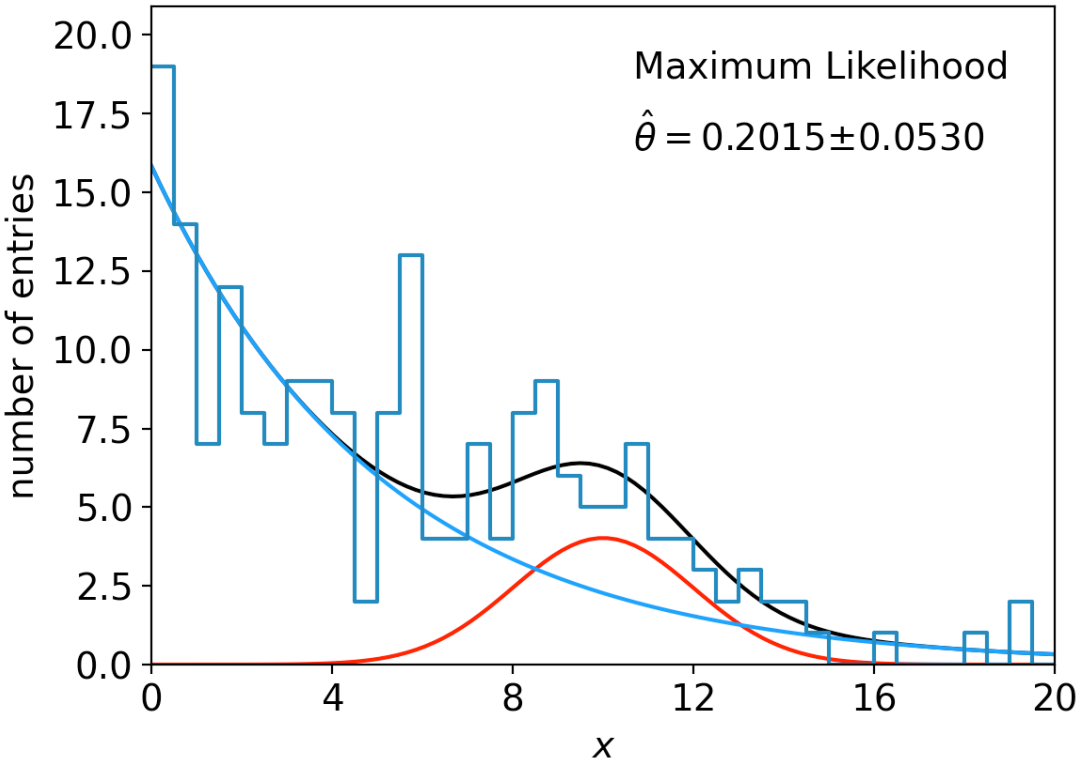
In our earlier iminuit example mlFit.py, the only argument of the log-likelihood function was the parameter array, and the data array xData entered as global (usually not a good idea):




```
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
```

Tutorial 2: 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

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

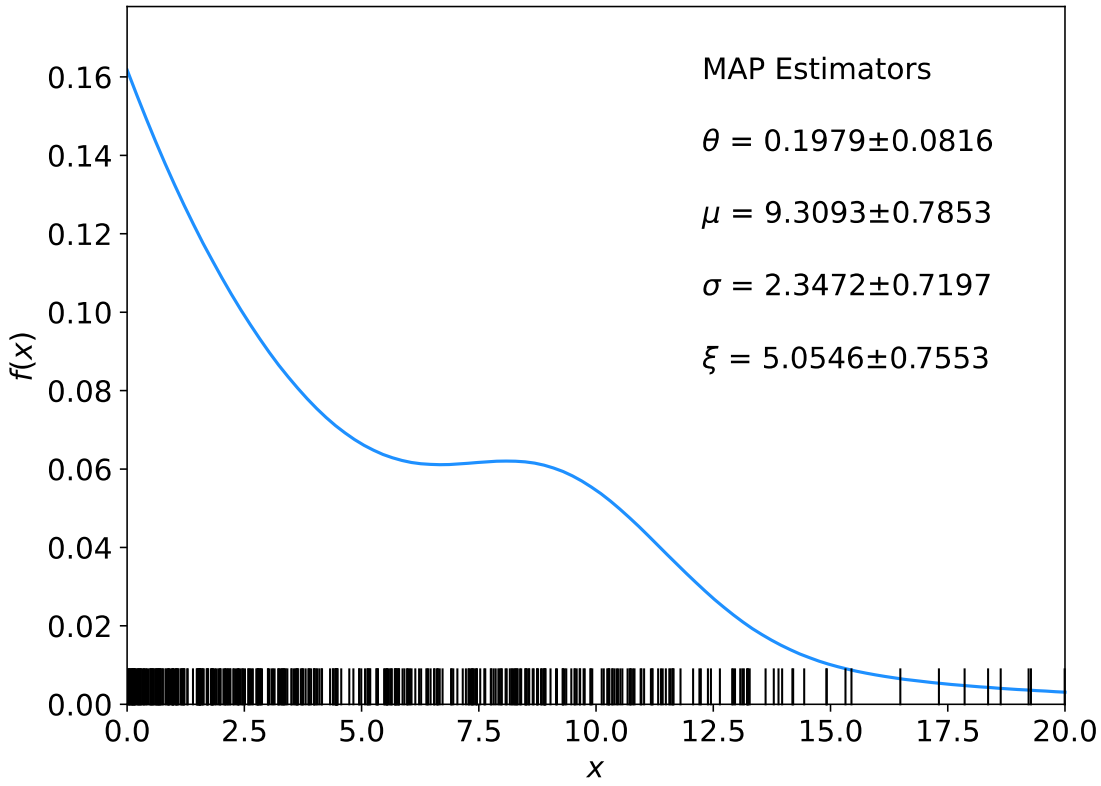
Posterior pdf for parameters $\boldsymbol{\lambda} = (\theta, \mu, \sigma, \xi)$ from Bayes theorem,

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

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).

Exercises on Bayesian parameter estimation (a)

1a) Run bayesFit.py, look at the plots

1(a) Run the program and examine the plots. These include:

1. The data values as ticks on the x axis together with the fitted curve evaluated with MAP estimators (Fig. 1 below). The uncertainties on the parameters correspond to the covariance $V_{ij} = \mathrm{cov}[\lambda_i, \lambda_j]$ that `iminuit` finds by approximating the posterior as a multivariate Gaussian near its maximum (similar to finding the covariance matrix of the MLEs).

2. Trace plots of each of the parameters (Fig. 2). In some problems it can be useful to discard a subset of the points (called "burn-in") if the starting point $\boldsymbol{\lambda}_0$ is too far from the main concentration of the target density's probability; this is indicated in the trace plots with a vertical yellow bar.

3. Marginal distributions of the individual parameters (Fig. 3). The histograms are normalized to unit area and the MAP estimates are indicated with the vertical bars.

4. The autocorrelation function for the parameters (Fig. 4).

Exercises on Bayesian parameter estimation (b,c)

1b) Investigate effect of data sample size, fixing parameters and length of MCMC chains.

1(b) Change the data sample size from $n = 400$ to 200 and 1000 and note the changes in the results.

Using again $n = 400$, fix the parameters μ and σ (by changing the corresponding elements in the array `parfix` from `False` to `True`) and note the changes in the results. When finished, go back to having all four parameters free.

Change the number of MCMC iterations from 10 000 to 100 000 and note the change in the results, particularly in the structures you see in the trace plots. (This probably takes some time to run; for the rest of the exercises it is probably best to change back to 10 000 iterations.

1c) Investigate changing the prior

1(c) Change the prior pdfs for ξ and σ to be $\pi(\xi) \propto 1/\xi$ and $\pi(\sigma) \propto 1/\sigma$ and note the change in the results. When finished, go back to constant priors.

Exercises on Bayesian parameter estimation (d)

1d) Include auxiliary measurement to constrain ξ

1(d) Suppose that one has an independent estimate u of the parameter ξ in addition to the $n = 400$ values of x. Treat u as Gaussian distributed with a mean ξ and standard deviation $\sigma_u = 0.5$ and take the observed value $u = 5$. Find the log-likelihood function that includes both the primary measurements $(x_1, \ldots, x_n)$ and the auxiliary measurement u and modify the fitting program accordingly. Investigate how the results are affected by including u.

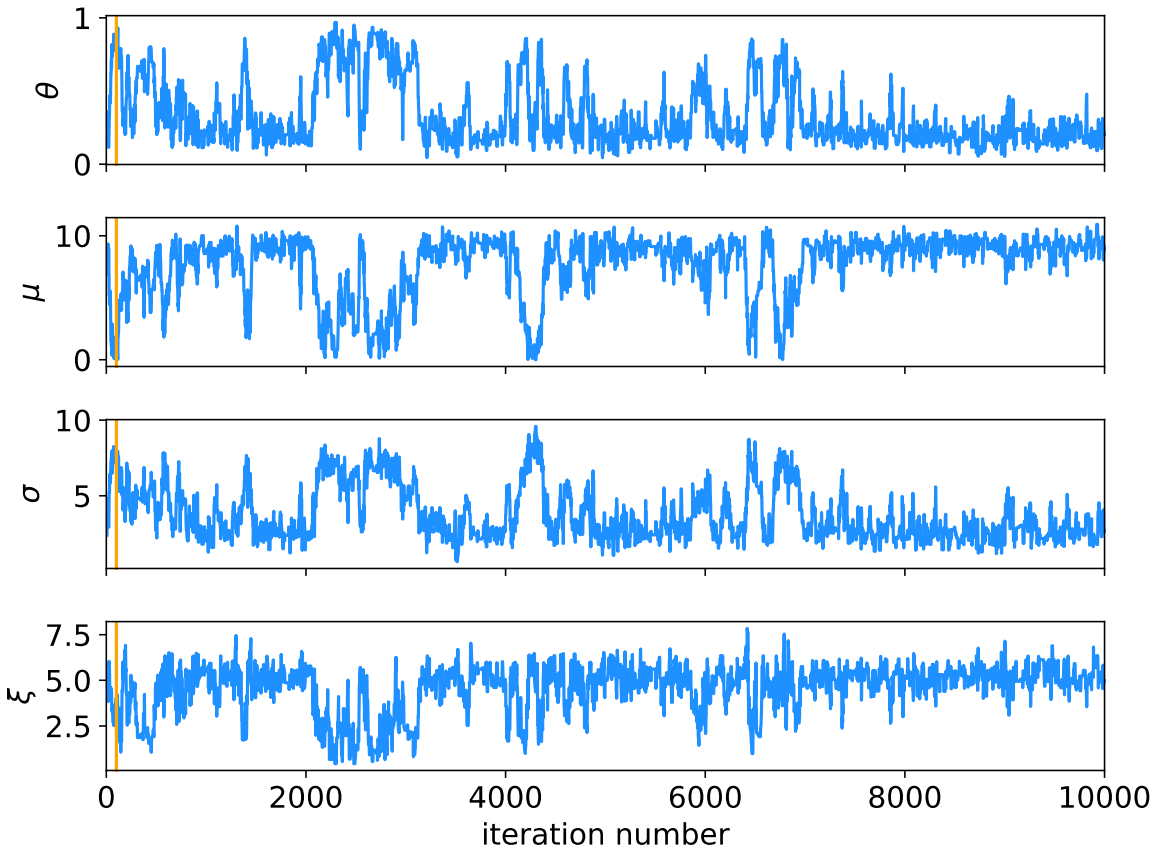
1e) Investigate point and interval estimates for θ

1(e) Using the functions `cc_interval` and `HPD_interval` provided in `bayesFit.py`, compute the central credible interval and HPD (highest probability density) interval for the parameter of interest θ using a credibility level of 68.3%. Compare these to the intervals one obtains from a point estimate (the MAP estimate, posterior median or posterior mean) plus or minus one standard deviation. For the standard deviation, try using both the sample standard deviation from the MCMC values and the standard deviation found by `iminuit`, which is based on a Gaussian approximation to the peak of the posterior. Find the estimates and intervales both with and without the auxiliary measurement of ξ as in (d) above and note how this effects the results.

MCMC trace plots

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

Marginalize by sampling posterior pdf with Metropolis-Hastings.

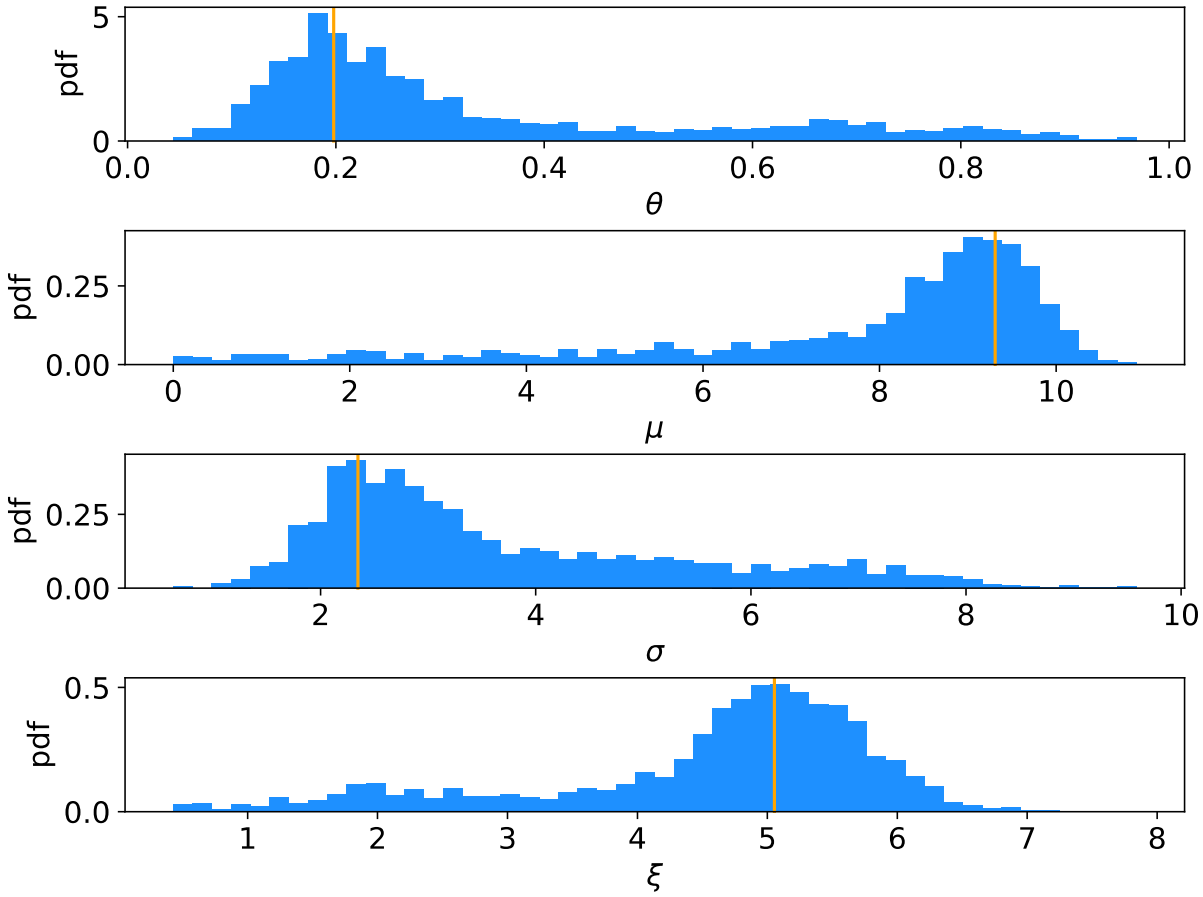


Gaussian proposal pdf, covariance $U = sV$, $s = (2.38)^2/N_{\text{par}} = 1.41$, gives acceptance probability ~ 0.24.

Here 10000 iterations (should use more).

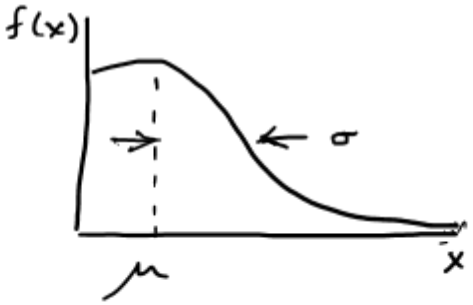
Marginal distributions

MAP estimates shown with vertical bars



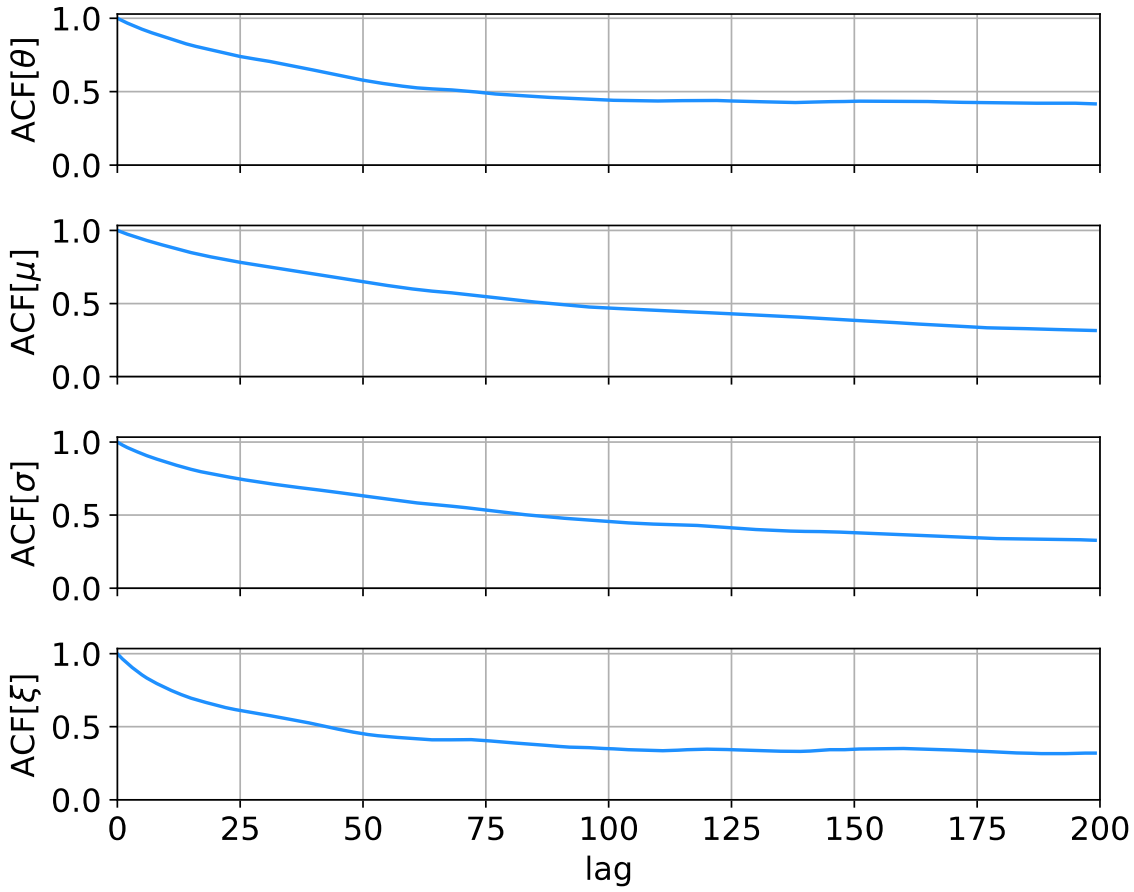
Note long tails.

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).



Autocorrelation versus lag

MCMC samples are not independent, autocorrelation function = correlation coefficient of sample x_i with x_{i+l} as a function of the lag, l, where x = any of θ, μ, σ, ξ minus its mean:



$$\mathrm{ACF} = \frac{1}{N}\sum_{i=1}^{N}\frac{x_i x_{i+\ell}}{\sigma^2}$$

Effective sample size

$$N_{\mathrm{eff}} = \frac{N}{1 + 2\sum_{\ell=1}^{\infty}\mathrm{ACF}_\ell}$$

In stat. error estimates

$$\frac{1}{\sqrt{N}} \to \frac{1}{\sqrt{N_{\mathrm{eff}}}}$$

Ways to summarize the posterior

Point estimates:

Posterior mode (MAP, coincides with MLE for constant prior).

Posterior median (invariant under monotonic transformation of parameter).

Posterior mean; coincides with above in large-sample limit.

Intervals:

Highest Probability Density (HPD) interval, shortest for a given probability content, not invariant under param. trans.

Central credible intervals, equal upper and lower tail areas, e.g., $\alpha/2$ for CL = $1 - \alpha$.

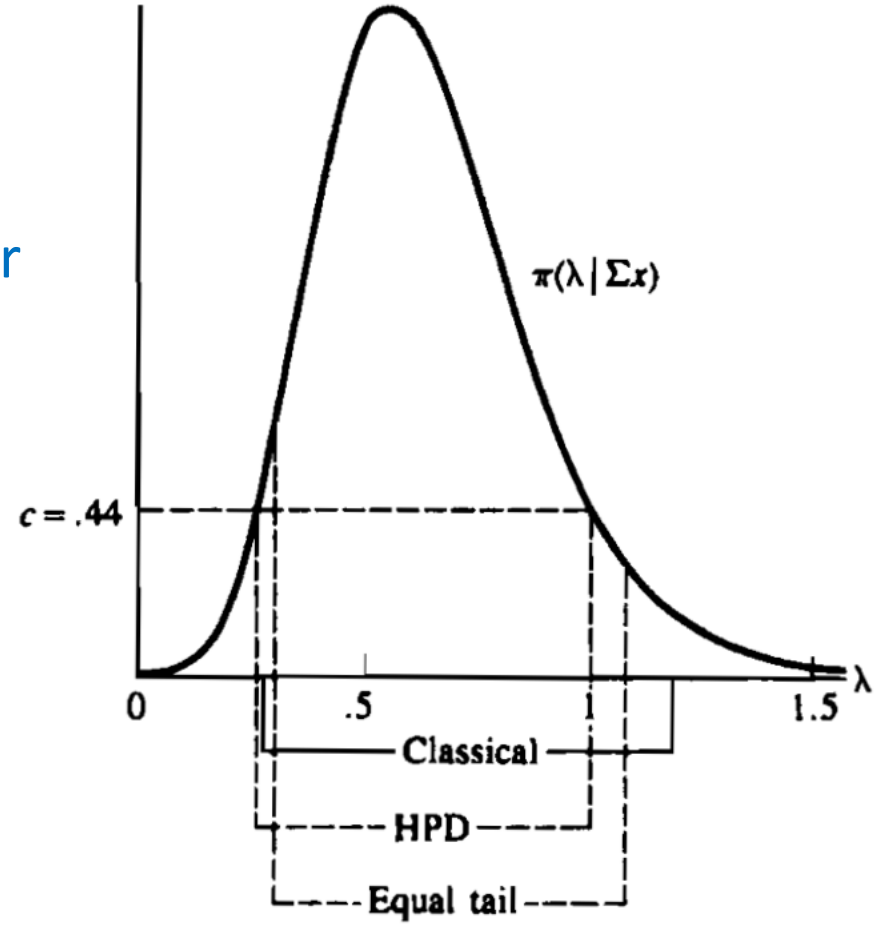
Point estimate +/- standard deviation, std. dev. from MCMC sample or by approximating core of posterior as Gaussian (from minuit); coincides with above in large-sample limit.

Types of intervals

HPD = Highest Posterior Density

Equal tail (central) from posterior

Classical (frequentist)



G. Casella and R. Berger, Statistical Inference, 2002

Tutorial 3: Hypothesis Tests

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

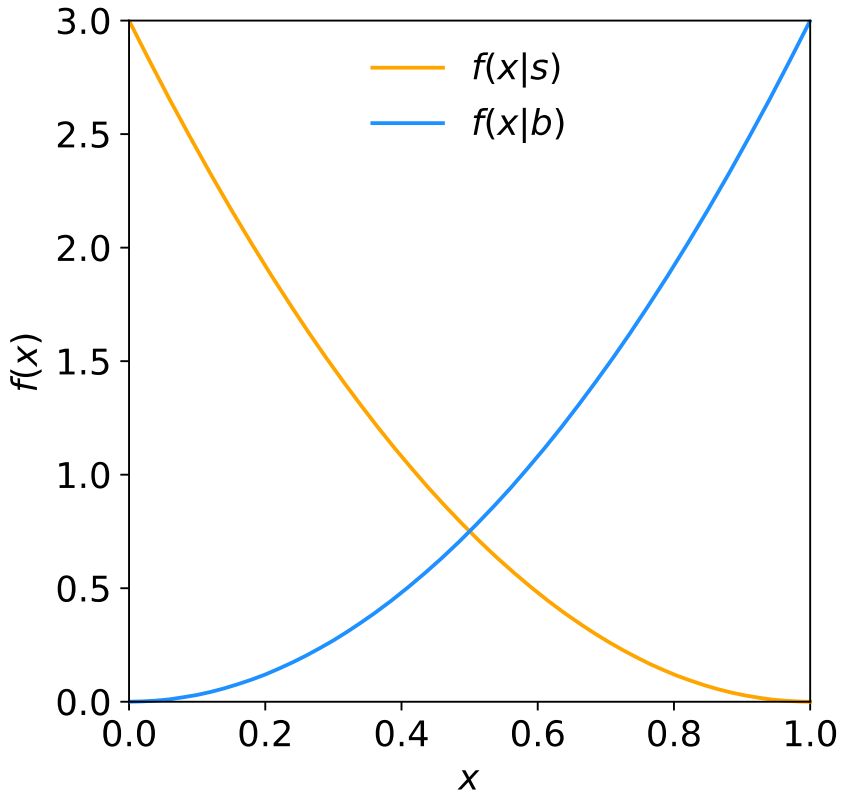
in hyp_test_exercise.pdf. Uses hypTest.py and hypTestMC.py.

Suppose we search for a signal like Dark Matter by counting events, and signal/background events are characterized by a variable x ($0 \leq x \leq 1$):

$$f(x|s) = 3(1-x)^2 ,$$

$$f(x|b) = 3x^2 .$$

As a first step, test the background hypothesis for each event: if $x < x_{\text{cut}}$, reject background hypothesis.



Exercises on hypothesis testing (a,b)

1a) Find boundary of critical region $x_{\rm cut}$ for test event is background.

1(a) Suppose for each event we test the hypothesis that it is background. We reject this hypothesis if the observed value of x is less than a specified cut value $x_{\rm cut}$. Find the value of $x_{\rm cut}$ such that the probability $P(x < x_{\rm cut}|b)$ to reject the background hypothesis (i.e., accept as signal) if it is background is $\alpha = 0.05$. (The value α is the *size* or significance level of the test used to select events.)

1b) Find power of the test with respect to event being signal.

1(b) For the value of $x_{\rm cut}$ that you find, what is the probability $P(x < x_{\rm cut}|s)$ to reject the background hypothesis (i.e., accept as a candidate signal event) with $x < x_{\rm cut}$ given that it is signal. (This is the *power* of the test of the background hypothesis with respect to the signal alternative or equivalently the signal efficiency.)

Exercises on hypothesis testing (c)

1c) In an experiment, $s_{\rm tot}$=10, $b_{\rm tot}$=100, select as s if $x < x_{\rm cut} = 0.1$, find expected numbers of signal, background.

1(c) Suppose that the expected number of background events is $b_{\rm tot} = 100$ and for a given signal model one expects $s_{\rm tot} = 10$ signal events. Find the expected numbers of events s and b of signal and background events that will satisfy $x < x_{\rm cut}$ using the value of $x_{\rm cut} = 0.1$, i.e.,

$$s = s_{\rm tot} P(x < x_{\rm cut}|s) \,, \tag{3}$$

$$b = b_{\rm tot} P(x < x_{\rm cut}|b) \,. \tag{4}$$

Exercises on hypothesis testing (d)

1d) Find signal purity

1(d) Assuming the numbers from 1(c), the prior probabilities for an event to be signal or background are

$$\pi_s = \frac{s_{\rm tot}}{s_{\rm tot} + b_{\rm tot}} = 0.09 \,, \tag{5}$$

$$\pi_b = \frac{b_{\rm tot}}{s_{\rm tot} + b_{\rm tot}} = 0.91 \,. \tag{6}$$

Using Bayes' theorem with these values, find the probability for an event to be signal given that it has $x < x_{\rm cut}$ (the signal purity of the selected sample).

Exercises on hypothesis testing (e)

1e) Suppose n events are found with $x < x_{\rm cut}$. Find the p-value of the background-only hypthesis ($s = 0$).

1(e) Suppose for a certain $x_{\rm cut}$ one has $b = 0.5$ and we find there $n_{\rm obs} = 3$ events in the search region $x < x_{\rm cut}$. We want to test the hypothesis that $s = 0$ (the background-only hypothesis or "b"), against the alternative that signal is present with $s \neq 0$ (the "$s + b$" hypothesis).

The actual number of events n found in the experiment with $x < x_{\rm cut}$ can be modeled as following a Poisson distribution with a mean value of $s + b$. That is, the probability to find n events is

$$P(n|s,b) = \frac{(s+b)^n}{n!} e^{-(s+b)} \ . \tag{7}$$

The p-value of the background-only hypothesis is the probability, assuming $s = 0$, to find $n \geq n_{\rm obs}$:

$$p = P(n \geq n_{\rm obs}|s = 0, b) = \sum_{n=n_{\rm obs}}^{\infty} \frac{b^n}{n!} e^{-b} = 1 - \sum_{n=0}^{n_{\rm obs}-1} \frac{b^n}{n!} e^{-b} \ . \tag{8}$$

Find the corresponding significance: $Z = \Phi^{-1}(1 - p)$

Exercises on hypothesis testing (f)

1f) Find the expected discovery significance median[$Z_b|s+b$]. Find the x_{cut} that maximizes median[$Z_b|s+b$].

1(f) The expected (median) significance assuming the $s+b$ hypothesis of the test of the $s = 0$ hypothesis is a measure of sensitivity and this is what one tries to maximize when designing an experiment. It can be approximated with a number of different formulas. For $s \ll b$ one can use $\text{med}[Z_b|s+b] = s/\sqrt{b}$. If $s \ll b$ does not hold, a better approximation is

$$\text{med}[Z_b|s+b] = \sqrt{2\left((s+b)\ln\left(1+\frac{s}{b}\right)-s\right)}\,. \qquad (10)$$

Using Eq. (10), find me median significance for $x_{\text{cut}} = 0.1$. If you have time, try to write a program to find the value of x_{cut} that maximizes the median significance.

Exercises on hypothesis testing (g)

← challenging

1g) Using the Monte Carlo program hypTestMC.py, investigate the test of s=0 by using the x values of each event (no cut).

1(g) Now suppose that for each event we do not simply count the events having x in a certain region but we design a test that exploits each measured value in the entire range $0 \leq x \leq 1$. Thus there is no cut on x and in here we use $s = 10$ and $b = 100$ to refer to the total expected numbers of signal and background events. The data consist of the number n of events, which follows a Poisson distribution with mean of $s + b$, and the n values $x_1, \ldots, x_n$.

The likelihood-ratio test statistic is

$$q = -2 \sum_{i=1}^{n} \ln \left[1 + \frac{s}{b} \frac{f(x_i|s)}{f(x_i|b)} \right]$$

(s = 10,b = 100, i.e., no cut)

Simulate 10^6 (or if possible 10^7) experiments and find the median discovery significance median[$Z_b|s+b$].

Likelihood ratio for test of $\mu = 0$

The likelihood for a signal strength μ is

$$L(\mu) = P(n, \mathbf{x}|\mu) = P(n|\mu) f(\mathbf{x}|n, \mu)$$

$$= \frac{(\mu s + b)^n}{n!} e^{-(\mu s + b)} \prod_{i=1}^{n} \left[\frac{\mu s}{\mu s + b} f(x_i|s) + \frac{b}{\mu s + b} f(x_i|b) \right]$$

We can test the hypothesis $\mu=0$ with the likelihood ratio statistic

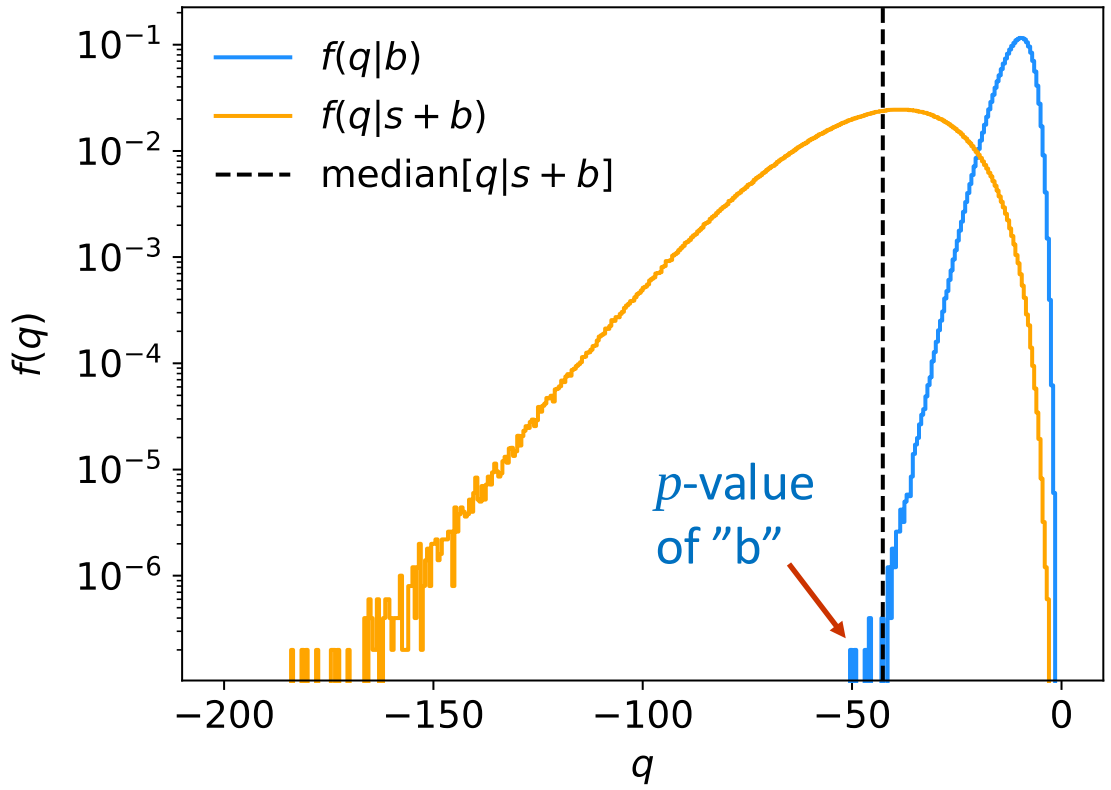
$$q = -2 \sum_{i=1}^{n} \ln \left[1 + \frac{s}{b} \frac{f(x_i|s)}{f(x_i|b)} \right] = -2 \ln \frac{L(1)}{L(0)} + C$$

constant, can drop

According to the Neyman-Pearson lemma, this gives the test of $\mu=0$ with the highest possible sensitivity (power with respect to the alternative $\mu=1$).

Expected discovery significance using q

10^7 experiments simulated simulated according to ”b” and “s+b”.

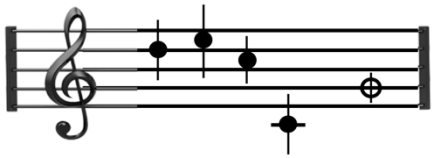


Program: hypTestMC.py

Add code to:

- find median[q|s+b]
- find median p-value of b, median[p_b|s+b]
- find median significance, median[Z_b|s+b]

Tutorial 4: Student's *t* average



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

Sample program stave.py

The program stave.py implements the Gamma Variance Model (GVM) for averaging N measurements.

For details see G. Cowan, EPJC (2019) 79:133.

In this version the model does not distinguish between statistical and systematic errors.

Confidence interval for the mean μ becomes sensitive to goodness-of-fit (increases if data internally inconsistent).

Estimated mean less sensitive to outliers.

Least Squares vs Gamma Variance Model

Quadratic terms from Least Squares replaced by logarithmic ones:

$$\frac{(y_i - \mu)^2}{\sigma_i^2} \quad \rightarrow \quad \left(1 + \frac{1}{2r_i^2}\right) \ln\left[1 + 2r_i^2 \frac{(y_i - \mu)^2}{v_i}\right]$$

where

y_i = measured value

$v_i = s_i^2$ = estimated variance

r_i = relative uncertainty on estimate of variance

Equivalent to replacing Gauss pdf for measurements by Student's t, number of degrees of freedom $= 1/2r_i^2$

A quick look at stave.py

Set measured values, estimates of std. dev., errors on errors:

```
y = np.array([17., 19., 15., 3.])       # measured values
s = np.array([1.5, 1.5, 1.5, 1.5])      # estimates of std. dev
v = s**2                                # estimates of variances
r = np.array([0.2, 0.2, 0.2, 0.2])      # relative errors on errors
```

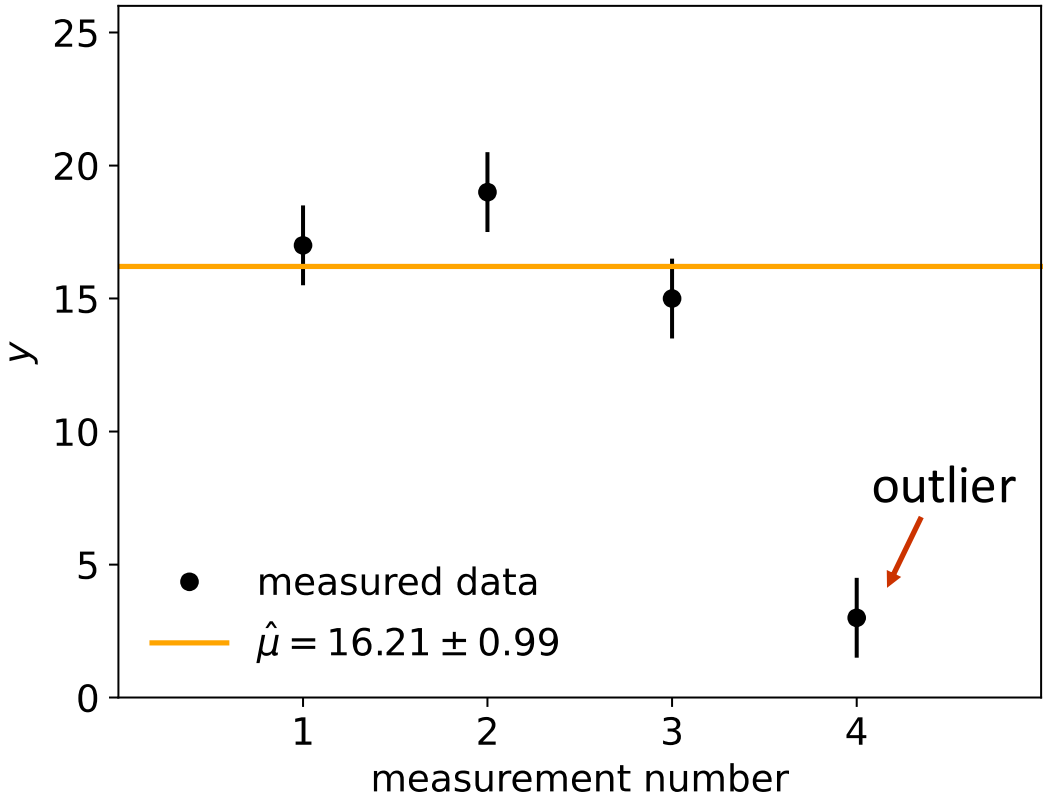
log-likelihood:

```
class NegLogL:

    def __init__(self, y, s, r):
        self.setData(y, s, r)

    def setData(self, y, s, r):
        self.data = y, s, r

    def __call__(self, mu):
        y, s, r = self.data
        v = s ** 2
        lnf = -0.5*(1. + 1./(2.*r**2))*np.log(1. + 2.*(r*(y-mu))**2/v)
        return -np.sum(lnf)
```

Example average with GVM

Suppose four measurements of the parameter μ.

Each reports an estimated standard dev. of $s = 1.5$ and a "relative error on the error" $r = 0.2$.



Suggested exercise:

Experiment with different numbers of measurements, different levels of internal consistency, different values for the std. dev. and error on error.