S1: Principles of Data Science

Problem Sheet 3 Solutions

MPhil in Data Intensive Science

Matt Kenzie
mk652@cam.ac.uk

Michealmas Term 2023

Problem Sheet 3

1. (a) Ensure p.d.f. normalises to unity:

$$\int_a^b N(1+\alpha x+\beta x^2)dx = 1. \tag{1}$$

Therefore:

$$\begin{aligned}\frac{1}{N} &= \int_a^b (1+\alpha x+\beta x^2)dx \qquad (2)\\ &= \left[x+\alpha\frac{x^2}{2}+\beta\frac{x^3}{3}\right]_a^b \qquad (3)\\ &= b+\alpha\frac{b^2}{2}+\beta\frac{b^3}{3}-a-\alpha\frac{a^2}{2}-\beta\frac{a^3}{3} \qquad (4)\\ &= (b-a)+\frac{\alpha}{2}\left(b^2-a^2\right)+\frac{\beta}{3}\left(b^3-a^3\right). \qquad (5)\end{aligned}$$

On reflection choosing α and β as parameters, with a and b as limits is a poor choice that can easily lead to confusion. Alas, it is done.

(b) First two central moments are:

$$\mu_1 = x \tag{6}$$
$$\mu_2 = x^2. \tag{7}$$

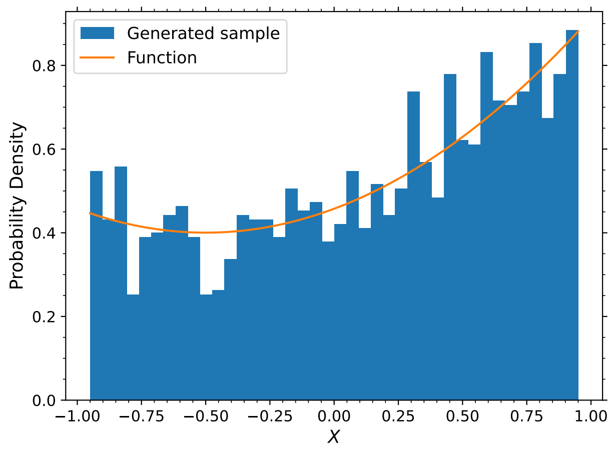
Take the expectation values of these gives

$$
\begin{aligned}
E[\mu_1] = E[x] &= \int_a^b Nx(1+\alpha x+\beta x^2)dx && (8)\\
&= N\int_a^b (x+\alpha x^2+\beta x^3)dx && (9)\\
&= N\left[\frac{x^2}{2}+\alpha\frac{x^3}{3}+\beta\frac{x^4}{4}\right]_a^b && (10)\\
&= N\left(\frac{1}{2}\left(b^2-a^2\right)+\frac{\alpha}{3}\left(b^3-a^3\right)+\frac{\beta}{4}\left(b^4-a^4\right)\right) && (11)\\
&= \frac{d_2+\alpha d_3+\beta d_4}{d_1+\alpha d_2+\beta d_3}. && (12)\\
E[\mu_2] = E[x^2] &= \int_a^b Nx^2(1+\alpha x+\beta x^2)dx && (13)\\
&= N\int_a^b (x^2+\alpha x^3+\beta x^4)dx && (14)\\
&= N\left[\frac{x^3}{3}+\alpha\frac{x^4}{4}+\beta\frac{x^5}{5}\right]_a^b && (15)\\
&= N\left(\frac{1}{3}\left(b^3-a^3\right)+\frac{\alpha}{4}\left(b^4-a^4\right)+\frac{\beta}{5}\left(b^5-a^5\right)\right) && (16)\\
&= \frac{d_3+\alpha d_4+\beta d_5}{d_1+\alpha d_2+\beta d_3}. && (17)\\
& && (18)
\end{aligned}
$$

(c) To get estimations for $\hat{\alpha}$ and $\hat{\beta}$ we simply solve these equations for α and β and plugin the sample estimates for the first and second central moments, $\hat{\mu}_1$ and $\hat{\mu}_2$. Solving the equations:

$$
\begin{aligned}
\hat{\alpha} &= \frac{(\hat{\mu}_1 d_3-d_4)(\hat{\mu}_2 d_1-d_3)-(\hat{\mu}_1 d_1-d_2)(\hat{\mu}_2 d_3-d_5)}{(\hat{\mu}_1 d_2-d_3)(\hat{\mu}_2 d_3-d_5)-(\hat{\mu}_1 d_3-d_4)(\hat{\mu}_2 d_2-d_4)}, && (19)\\
\hat{\beta} &= \frac{(\hat{\mu}_1 d_1-d_2)(\hat{\mu}_2 d_2-d_4)-(\hat{\mu}_1 d_2-d_3)(\hat{\mu}_2 d_1-d_3)}{(\hat{\mu}_1 d_2-d_3)(\hat{\mu}_2 d_3-d_5)-(\hat{\mu}_1 d_3-d_4)(\hat{\mu}_2 d_2-d_4)}. && (20)
\end{aligned}
$$

(d) Accept-reject code we have already seen in a previous problem sheet. So just make a plot like this one



(e) Write some code similar to that which can be found in `method_of_moments.py` which computes MoM estimates and compares to MLE. I put this code below for reference

```python
## Solution to Problem Sheet 3 Question 1
## once again more optimal solutions will be available

import numpy as np
np.random.seed(210187)
from scipy.optimize import brute, minimize
from scipy.integrate import quad
from scipy.stats import moment
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')
from jacobi import propagate
from iminuit import cost, Minuit
import sys
sys.path.append('$PWD/code')
from accept_reject import accept_reject_1d, check_ok_plot
from tabulate import tabulate

if __name__=="__main__":

    xrange = [-0.95,0.95]
    alpha = 0.5
    beta = 0.5

    def dk(k):
        return (xrange[1]**k - xrange[0]**k)/k

    N = 1/( dk(1) + alpha*dk(2) + beta*dk(3) )

    func = lambda x: N*(1 + alpha*x + beta*x**2)

    # run the accept-reject
    dset = accept_reject_1d( func, xrange=xrange, size=2000 )
    check_ok_plot( func, xrange, dset, bins=40, save=f'figs/mom_gen.pdf' )
    dset = np.asarray(dset)
    np.save('../s1_principles_of_data_science/datasets/mom_data.npy', dset)

    # MoM estimate
    m1_hat = moment(dset, moment=1, center=0)
    m2_hat = moment(dset, moment=2, center=0)

    def alpha(m1, m2):
        numerator = (m1*dk(3) - dk(4))*(m2*dk(1)-dk(3)) - (m1*dk(1)-dk(2))*(m2*dk(3)-dk(5))
        denominator = (m1*dk(2) - dk(3))*(m2*dk(3)-dk(5)) - (m1*dk(3)-dk(4))*(m2*dk(2)-dk(4))
        return numerator / denominator

    def beta(m1, m2):
        numerator = (m1*dk(1) - dk(2))*(m2*dk(2)-dk(4)) - (m1*dk(2)-dk(3))*(m2*dk(1)-dk(3))
        denominator = (m1*dk(2) - dk(3))*(m2*dk(3)-dk(5)) - (m1*dk(3
)-dk(4))*(m2*dk(2)-dk(4))
        return numerator / denominator

    alpha_hat = alpha(m1_hat, m2_hat)
    beta_hat = beta(m1_hat, m2_hat)

    N = len(dset)
    cov11 = 1/(N*(N-1)) * np.sum( (dset-m1_hat) * (dset-m1_hat) )
    cov22 = 1/(N*(N-1)) * np.sum( (dset**2-m2_hat) * (dset**2-m2_hat
) )
    cov12 = 1/(N*(N-1)) * np.sum( (dset-m1_hat) * (dset**2-m2_hat) )

    cov_hat = np.array( [[cov11,cov12],[cov12,cov22]] )

    # propagate to parameter estimates

    transform = lambda p: np.array( [ alpha(*p), beta(*p) ] )

    mom, mom_cov = propagate( transform, [m1_hat, m2_hat], cov_hat )

    # Now some kind of MLE

    def density(x, alpha, beta):
        N = 1/( dk(1) + alpha*dk(2) + beta*dk(3) )
        return N*(1 + alpha*x + beta*x**2)

    nll = cost.UnbinnedNLL(dset, density)
    mi = Minuit( nll, alpha=0.5, beta=0.5 )
    mi.migrad()
    mi.hesse()

    print("---- METHOD OF MOMENTS ----")
    print(tabulate(zip(['alpha','beta'], mom, np.diag(mom_cov)**0.5)
, headers=['Parameter','Value','Error']))
    print("---- MAXIMUM LIKELIHOOD ---")
    print(tabulate(zip(['alpha','beta'], list(mi.values), list(mi.
errors)), headers=['Parameter','Value','Error']))

    plt.show()
```

2. There are 25 bins. A Gaussian has 2 shape parameters and there is one scale or "height" parameter for how much of the Gaussian there is. The background has no shape parameters (it is flat) but still has an overall height parameter. Thus the total is $25 - 3 - 1 = 21$.

3. The code for this solution is provided in `neutrinos.py`.

   (a) To compute the expectation of our Poisson then we can do

$$\bar{\nu} = \frac{\sum w_i N_i}{\sum N_i} \tag{21}$$

   where w_i is the number of events for an internal and N_i is the number of intervals with that number of events.

   This results in a value of $\bar{\nu} = 0.777$.

I can now ask how many events I would expect to see from a Poisson distribution with this expectation and then multiply this by the total number of intervals I measured to give an extra column in my table:

No. of events	0	1	2	3	4	5	6	7	8	9
Obs no. of intervals	1042	860	307	78	15	3	0	0	0	1
Pred no. of intervals	1060.15	823.847	320.107	82.919	16.109	2.503	0.324	0.036	0.0035	0.0003

Then use this to calculate the χ^2 and p-value. Including all data gives:

$$\chi^2 = 3313 \tag{22}$$

$$p\text{-value} = 0 \tag{23}$$

Clearly this is a dreadful fit.

(b) Then omitting the data in the last bin gives

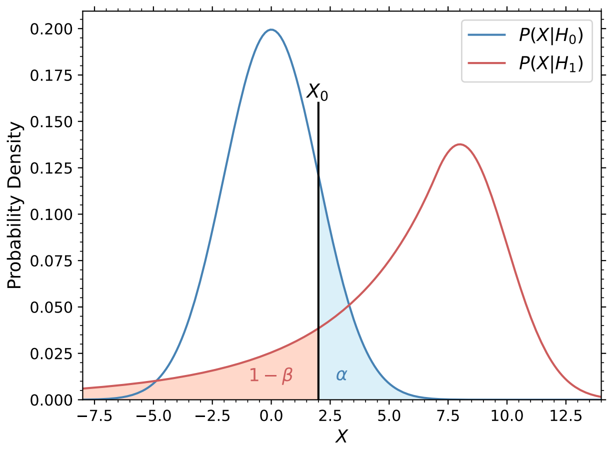
$$\chi^2 = 3.25 \tag{24}$$

$$p\text{-value} = 0.92 \tag{25}$$

Which is a very reasonable fit.

(c) The conclusion of course is that a simple Poisson only background cannot explain this occurence of an intervals countaining 9 neutrinos events. The conclusion of that is an observation has been made of some object which for a short period was emitting a high-neutrino flux. This was the supervnova S1987A.

4. I actually already do the proof of this in the lectures, but just to reproduce it here. We already saw this sort of plot in the lectures showing a test-statistic distribution under null and alternate hypotheses and then the regions α and β which lie above the observation.



The regions α and β are defined such that

$$\alpha = \int_{X_0}^{\infty} p(X;\theta_0)dX \tag{26}$$

$$\beta = \int_{X_0}^{\infty} p(X;\theta_1)dX \tag{27}$$

assuming that θ is some continuous parameter which differentiates between the null and alternate hypotheses (for example a number of signal). For a fixed α we need to find the test-statistic that will maximise β. Thus starting from the term of β above:

$$\beta = \int_{X_0}^{\infty} p(X;\theta_1)dX \tag{28}$$

$$= \int_{X_0}^{\infty} \frac{p(X;\theta_1)}{p(X;\theta_0)} p(X;\theta_0)dX \qquad = E\left[\frac{p(X;\theta_1)}{p(X;\theta_0}\right] \tag{29}$$

Thus we see that in order to maximise β we have to maximise the expectation value on the right which is done with test-statistics of the form of the likelihood ratio.

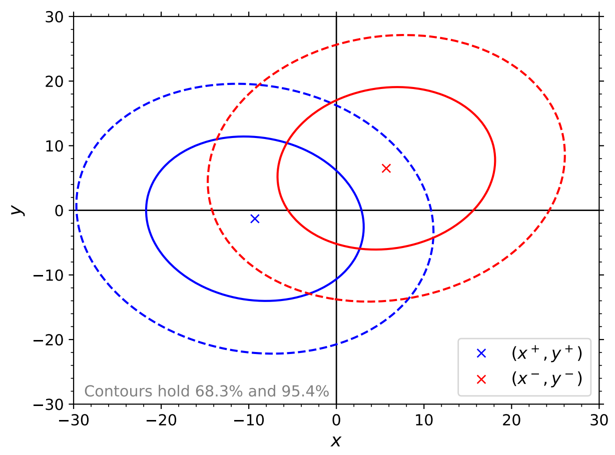
5. A question inspired by measurements of the CP-violating phase γ which appears in the CKM quark-mixing matrix

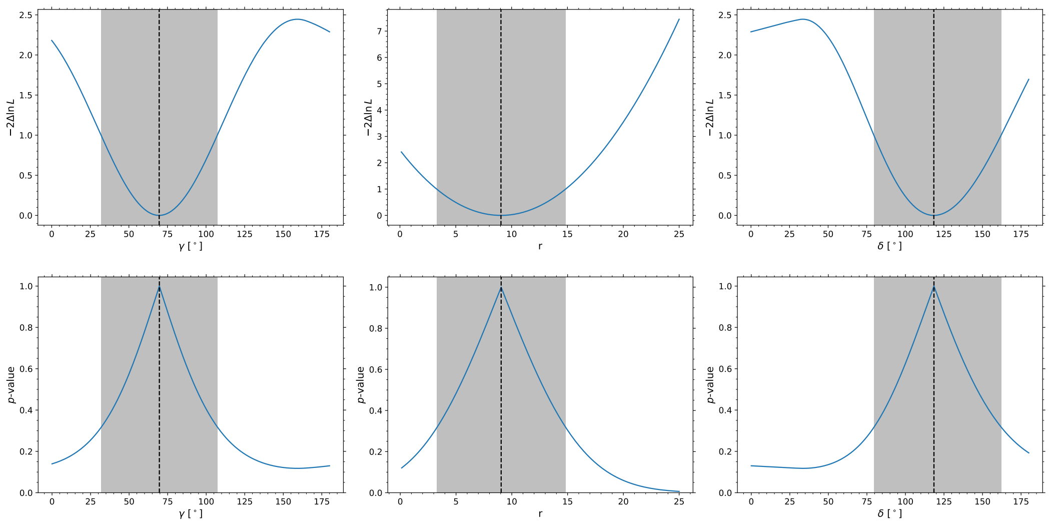
   (a) A snippet of code which will perform the relevant stuff is in `polar.py` which I put below

```python
import numpy as np
from scipy.stats import multivariate_normal as mvn, chi2
from iminuit import Minuit
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')

names = [ 'xp', 'yp', 'xm', 'ym' ]
titles = [ '$x^+$', '$y^+$', '$x^-$', '$y^-$' ]
values = np.array( [ -9.3, -1.3, 5.7, 6.5 ] )
errors = np.array( [  8.2,  8.4, 8.2, 8.3 ] )

corr = np.array([
    [  1    , -0.1  , -0.05,    0.1 ],
    [ -0.1  ,  1    ,  0.1  ,  -0.05],
    [ -0.05,  0.1  ,  1    ,    0.1 ],
    [  0.1  , -0.05,  0.1  ,    1    ] ] )

cova = np.ones_like( corr )
for i in range(4):
    for j in range(4):
        cova[i,j] = errors[i] * corr[i,j] * errors[j]

model = mvn( mean=values, cov=cova )
ll_min = model.logpdf( values )

def draw_2d(mi):
    fig, ax = plt.subplots()

    ax.axvline( 0, c='k', lw=1)
    ax.axhline( 0, c='k', lw=1)

    mvnp = mvn( mean=values[:2], cov=cova[:2,:2] )
    mvnm = mvn( mean=values[2:], cov=cova[2:,2:] )

    x = np.linspace(-30,30,100)
    y = np.linspace(-30,30,100)

    X, Y = np.meshgrid(x,y)

    pos = np.dstack([X,Y])

    n2llp = -2 * ( mvnp.logpdf( pos ) - mvnp.logpdf( values[:2] ) )
    n2llm = -2 * ( mvnm.logpdf( pos ) - mvnm.logpdf( values[2:] ) )

    sigmas = np.array( [1, 2] )
    levels = chi2.ppf( chi2.cdf( sigmas**2, 1), 2 )
    ax.contour( X, Y, n2llp, colors='b', linestyles=['-','--'],
    levels=levels )
    ax.contour( X, Y, n2llm, colors='r', linestyles=['-','--'],
    levels=levels )

    ax.plot( *values[:2], 'bx', label='$(x^+, y^+)$' )
    ax.plot( *values[2:], 'rx', label='$(x^-, y^-)$' )

    ax.text(0.02, 0.02, 'Contours hold 68.3% and 95.4%', transform=
    ax.transAxes, fontsize=11, color='0.5' )

    ax.legend(loc='lower right')

    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')

    fig.savefig('figs/polar_plot.pdf')

def draw_1d_profile( mi, par, title, ax=None, convert=False ):

    ax = ax or plt.gca()

    x, y, valid = mi.mnprofile(par, subtract_min=True, bound=mi.
    limits[par], size=200)
    if convert:
        y = chi2.sf(y,1)
    ax.plot( x[valid], y[valid] )

    ax.axvspan( mi.values[par]+mi.merrors[par].lower, mi.values[par
    ]+mi.merrors[par].upper, fc='0.5', alpha=0.5 )
    ax.axvline( mi.values[par], c='k', ls='--' )

    ax.set_xlabel(title)
    ax.set_ylabel('$-2\Delta\ln L$')
    if convert:
        ax.set_ylabel('$p$-value')

    if convert:
        ax.set_ylim(bottom=0)

```

```
def n2ll( gamma, r, delta ):
    g = np.radians(gamma)
    d = np.radians(delta)
    xp = r*np.cos(d+g)
    yp = r*np.sin(d+g)
    xm = r*np.cos(d-g)
    ym = r*np.sin(d-g)

    ll_here = model.logpdf( [xp, yp, xm, ym] )

    n2ll = -2*(ll_here - ll_min)

    return n2ll

mi = Minuit( n2ll, gamma=100, r=1, delta=150 )
mi.limits['gamma'] = (0,180)
mi.limits['delta'] = (0,180)
mi.limits['r'] = (0,25)

mi.migrad()
mi.hesse()
mi.minos()
print(mi)

draw_2d(mi)

fig, axes = plt.subplots(1,3, figsize=(19.2, 4.8))
for par, title, ax in zip(['gamma','r','delta'], ['$\gamma$ [$^\
    circ$]', 'r', '$\delta$ [$^\circ$]'], axes.flatten()):
    draw_1d_profile( mi, par, title, ax=ax )
fig.savefig('figs/polar_prof_nll.pdf')

fig, axes = plt.subplots(1,3, figsize=(19.2, 4.8))
for par, title, ax in zip(['gamma','r','delta'], ['$\gamma$ [$^\
    circ$]', 'r', '$\delta$ [$^\circ$]'], axes.flatten()):
    draw_1d_profile( mi, par, title, ax=ax, convert=True )
fig.savefig('figs/polar_prof_pval.pdf')

plt.show()
```

My code will make the following plots:





(b) The profile likelihood looks fine for r (depending on where you run the limits to). The problem is you hit a physical boundary at 0 and thus cannot quote an interval out to 95%.

(c) To compute the coverage you would through toys (pseduo-experiments) from the best-fit point. Refit those and compute what fraction of the time the X% interval contains the true value. This would tell you if you have a coverage issue. If you really did this for the 90% interval on γ you would see that it would undercover.

(d) To set the appropriate upper limit on r you should use the Feldman-Cousins method (as r is near the physical boundary).

6. Most powerful is the log-likelihood ratio and each is Poisson distributed so

$$\begin{aligned}
L_1/L_0 &= \prod_i^N \text{Poiss}(d_i; s_i + b_i) / \prod_i^N \text{Poiss}(d_i; b_i) && (30)\\
&= \prod_i^N \frac{e^{-(b_i+s_i)}(b_i+s_i)^{d_i}}{d_i!} \Big/ \prod_i^N \frac{e^{-b_i} b_i^{d_i}}{d_i!} && (31)\\
&= \prod_i^N \frac{e^{-s_i}(s_i+b_i)^{d_i}}{b_i^{d_i}} && (32)\\
\ln(L_1/L_0) &= \sum_i^N d_i \ln(s_i+b_i) - s_i - d_i \ln(b_i) && (33)
\end{aligned}$$