pset1

September 20, 2022

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
```

1 Problem #1

See attached PDF.

2 Problem #2

```
[2]: # Start by defining our two PDFs. Note that I have converted the poisson into␣
      ↪log poisson and taken
     # the exponential of it to get the probability. I've also used Stirling's␣
      ↪approximation to remove
     # the factorial.

     def gauss(lamda, a = 3):
         x = lamda + a*np.sqrt(lamda)
         return 1/np.sqrt(2*np.pi*lamda) * np.exp(-(x - lamda)**2 / (2*lamda))


     def log_poisson(lamda, a = 3):
         x = lamda + a*np.sqrt(lamda)
         return np.exp(-0.5*np.log(2*np.pi*x) + x*np.log(lamda/x) - lamda + x)


     def _get_minimum_lambda(a):
         """Calculate the minimum value of lambda such that 1 < P/G < 2 is true at n␣
      ↪=   + a\sqrt( ).

         Parameters:
         -----------
         a: int
             Coefficient corresponding to sigma level you wish to compare the␣
      ↪poisson and gaussian distributions, e.g.
             5 sigma level ==> a = 5.
```

```
    Returns:
    lambda: int
        Minimum lambda.
    n: int
        Minimum number of repetitions.
    """

    lamda = 1
    _gauss = gauss(lamda, a = a)
    _poisson = log_poisson(lamda, a = a)

    ratio = _poisson/_gauss

    while ratio > 2 or ratio < 1:
        lamda+=1
        _gauss = gauss(lamda, a = a)
        _poisson = log_poisson(lamda, a = a)
        ratio = _poisson/_gauss

    n = lamda + a*np.sqrt(lamda)

    return lamda, int(n)
```

2.0.1 Sigma level: 3

[3]:
```
sig_level = 3
lamda, n = _get_minimum_lambda(sig_level)
print('For Poisson and Gaussian to agree at the 3  level, we would need   = {0}␣
 ↪corresponding to a minimum number of repetitions of n = {1}. '.format(lamda,␣
 ↪n))
```

```
For Poisson and Gaussian to agree at the 3  level, we would need   = 9
corresponding to a minimum number of repetitions of n = 18.
```

2.0.2 Sigma level: 5

[4]:
```
sig_level = 5
lamda, n = _get_minimum_lambda(sig_level)
print('For Poisson and Gaussian to agree at the 5  level, we would need   = {0}␣
 ↪corresponding to a minimum number of repetitions of n = {1}. '.format(lamda,␣
 ↪n))
```

```
For Poisson and Gaussian to agree at the 5  level, we would need   = 576
corresponding to a minimum number of repetitions of n = 696.
```

3 Problem #3

See attached PDF.

4 Problem #4

[5]:
```
x = np.linspace(-5,5,51)
npts = len(x)
tot_chunks = 10000

# Generate 10000 samples, each defined by a gaussian with random noise added to␣
 ↪it.
signal_arr = np.exp(-x[:,np.newaxis]**2/2) + np.random.randn(npts, tot_chunks)

# Get noise by taking variance of each chunk
noise = np.var(signal_arr, axis = 0)
```

The model parameter that we are interested in fitting for is the amplitude of the Gaussian, e.g.

$$f(x) = m \cdot \exp\left[\frac{-x^2}{2}\right], \tag{1}$$

where m is the amplitude that we expect $\langle m\rangle = m_{\text{true}} = 1$. To do this, we recall that the least-squares best fit parameter can be obtained using the equation:

$$m = (A^T N^{-1} A)^{-1} A^T N^{-1} d, \tag{2}$$

with the error on the estimate being

$$\delta m = (A^T N^{-1} A)^{-1} \tag{3}$$

[6]:
```
def unit_gauss(x):
    return np.exp(-x**2 / 2)


# Build our A matrix using our model. Will be a (51,) array
A = unit_gauss(x)

# Iterate over all the chunks
m_arr = []
m_err_arr = []
for i in range(tot_chunks):

    # Noise is uncorrelated, so matrix will be diagonal, square, with␣
 ↪dimensions npts x npts
    _Ninv = 1/noise[i] * np.eye(npts)
    _lhs = A[:, np.newaxis].T@_Ninv@A
    _rhs = A[:,np.newaxis].T@_Ninv@signal_arr[:,i]
    # Save inverse of lhs for error on best fit param. Since it's 1-D, it's␣
↪simply 1 / lhs.
    _lhs_inv = 1 / _lhs

    # Compute best fit parameters
    _m = _lhs_inv@_rhs

    m_arr.append(_m)
    m_err_arr.append(np.sqrt(_lhs_inv[0]))
```

To show that on average, $\langle m \rangle = m_{\text{true}}$, we simply average over all 10000 samples and look at the average value.

[7]:
```
print('Average value of fit amplitude over {0} samples is:  < m > = {1} ± {2}'.
↪format(tot_chunks, np.mean(m_arr), np.mean(m_err_arr)))
```

```
Average value of fit amplitude over 10000 samples is:  < m > =
0.9982162413239939 ± 0.3494662512679676
```

The expected value of $m_{\text{true}} = 1$ falls within our average value with averaged 1σ error bars.

4.0.1 Weighted Average

To get the weighted average, we simply use the following formula:

$$m = \frac{\sum_i w_i m_i}{\sum_i w_i}. \tag{4}$$

with variance (from question 2)

$$\text{Var}[m] = \frac{\sum_i w_i^2 \text{Var}[m_i]}{\sum_i w_i^2} = \frac{\sum_i w_i^2 \sigma_i^2}{\sum_i w_i^2} \tag{5}$$

[8]:
```
weights = (1/np.asarray(m_err_arr))**2
m_weighted = np.sum(np.asarray(m_arr)*weights)/np.sum(weights)
m_weighted_var = np.sum((weights*np.asarray(m_err_arr))**2)/np.sum(weights)**2
```

[9]:
```
print('Using the weighted mean, we find m = {0}  ± {1}'.format(m_weighted, np.
↪sqrt(m_weighted_var)))
```

```
Using the weighted mean, we find m = 0.9752139927205926  ± 0.0034432312094356487
```

As we can see, it has clearly biased our amplitude low by roughly 3%. As to why? We are using our model to estimate the noise statistics, so if we wanted to avoid this bias, we would want to estimate the noise statistics using the underlying physics that is causing the noise (and not taking the variance of the signal).

5 Problem #5

See attached PDF.