Lab 6

Ondrea Robinson

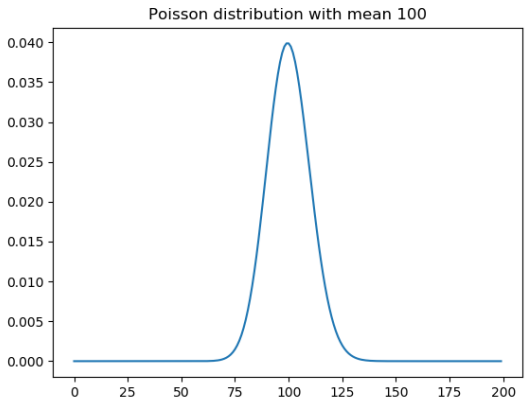
November 17, 2019

1 Problem 1

For this problem, we're looking at generated data; our data will be a Poisson distribution with $\lambda = 100$. We can easily generate this in Python. Now we want to determine with our distribution what the 5σ sensitivity threshold is. We've done this before and we can use the code below to find our threshold and plot the data:

```
prob = 1/3.5e6
mu = 100
meas = stats.poisson.ppf(1-prob, mu)
print(meas)

x = np.arange(0, 200)
figure()
plt.plot(x, stats.poisson.pmf(x, mu))
plt.show()
```

This is assuming we define signal-like as being a greater value rather than lesser. This is the plot:



We get meas = 154. So our threshold for 5σ discovery is 154 events with our current data set.

2 Problem 2

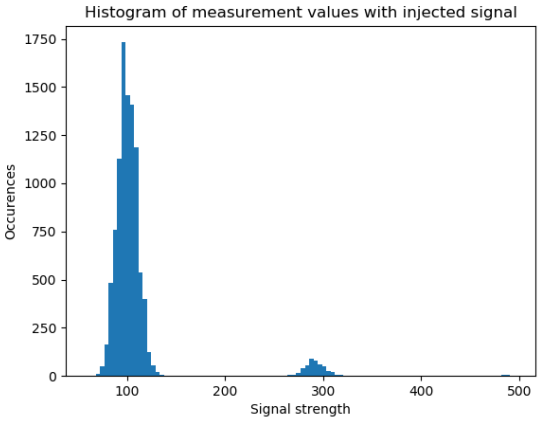
2.1 a

Now we're going to create a set of injected (simulated) signals of one specific strength then inject this signal into our data many times. Let's choose a signal with significance 8σ. First, let's find the strength of this signal in the same units as our distribution:

```
prob = 1.0 - stats.norm.cdf(8)
print(stats.poisson.ppf(1-prob, mu))
```

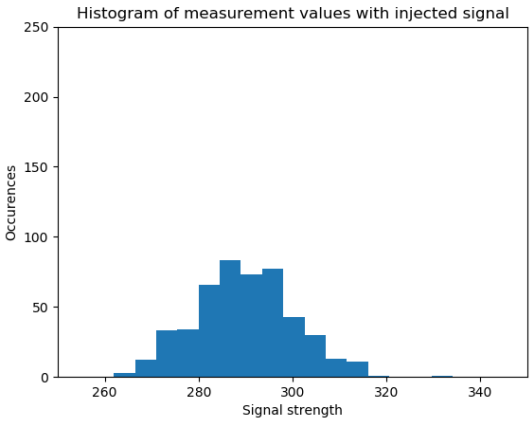
This calculation tells us we need a signal of strength 190 to have a significance of 8σ. Now let's inject this signal into our data. We're going to inject 500 instances of our signal into our regular background distribution. We want to inject our signal into this array randomly then plot a histogram of the resulting data. The code for this and the plot is shown below:

```
noise = stats.poisson.rvs(mu, size=10000)
numtimes = 500 #number of times we inject our signal into the data
signal = 190

for i in range(numtimes):
    index = np.random.randint(0, high=9999)
    noise[index] = noise[index] + signal

figure()
plt.xlabel('Signal strength')
plt.ylabel('Occurences')
plt.title('Histogram of measurement values with injected signal')
plt.hist(noise, bins=100)
plt.show()
```



Let's change our histogram so we can look closer at the region where the signal is distributed:

```
figure()
plt.xlabel('Signal strength')
plt.ylabel('Occurences')
plt.title('Histogram of measurement values with injected signal')
plt.xlim(250, 350)
plt.hist(noise, bins=100)
plt.show()
```



This shape is symmetric around the most common value of 290. We would expect this to be the most common value since the we're adding 190 to a distribution with 100 as the most common value. It also makes sense that the signals would be symmetric about this value since the signal is added to a distribution that is symmetric.

2.2 b

The observed signal is not biased. It's not more likely to be less than or greater than the most common signal so we can classify it as symmetric.

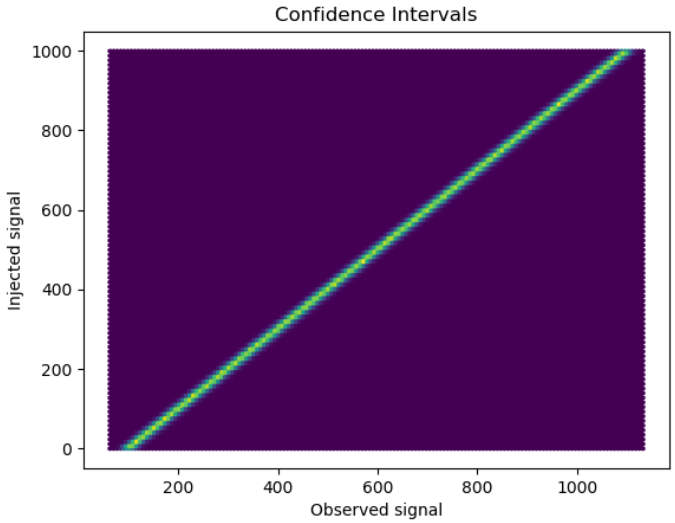
3 Problem 3

Now we're going to make a suite of injected signals. We want to have a range of injected signal strengths, starting at zero and extending well above 5σ, we want up to around 30σ or more.

3.1 a

To implement this let's set our range of signals in units pertaining to our distribution. We know the minimum value (0) and for 30 sigma we can use the value 924 (6 times our 5σ value), this isn't exact but Python has difficulties calculating the probability of signals above 8σ with precision. We'll choose 1000 as our upper bound in order to be above $30\sigma = 924$. Now we want to create a range of signals within our threshold, inject those signals into our background, then histogram. Here's the resulting code and a histogram of the results:

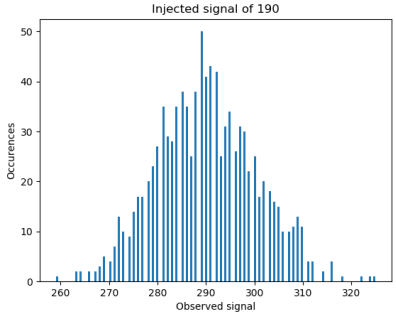
```
size = 1000
averages = np.full((1,size), 100)

signalstrength = np.linspace(0, 999, size)
signal = np.array(signalstrength)

noise = np.random.poisson(lam=(averages), size=(size, size))
backg = np.array(noise)

obssignal = backg
for i in range(size):
    sig = signal[i]
    obssignal[i] = sig + backg[i]

scratch, signalgrid = np.meshgrid(signal,signal)

figure()
plt.hexbin(obssignal, signalgrid, gridsize=150)
plt.xlabel('Observed signal')
plt.ylabel('Injected signal')
plt.title('Confidence Intervals')
plt.show()
```



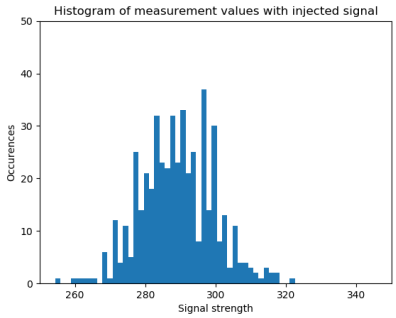
3.2 b

Now we want to show that if we chose the same injected signal power as in problem one we get the same histogram. We can think of this as taking a slice of the graph at a constant injected signal power. So if we slice at 8σ (190) we get a histogram that looks like:

```
figure()
plt.hist(obssignal[190], bins=150)
plt.title('Injected signal of 190')
plt.ylabel('Occurences')
plt.xlabel('Observed signal')
plt.show()
```



Let's compare this to our first histogram:

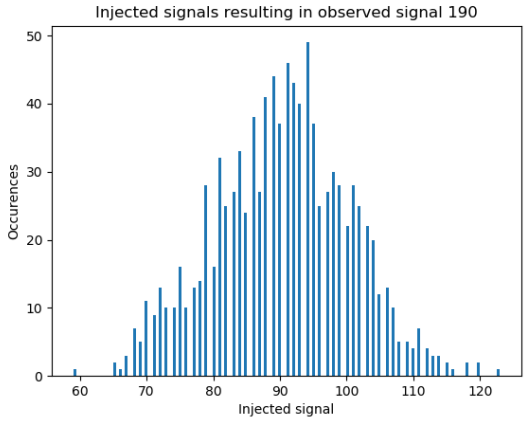


These aren't the exact same histograms because we're comparing two, different randomly generated data sets. However, the main features are still the same. The most common value for each is 190 and they are both symmetric about this value.

3.3 c

Now, rather than slicing at a constant injected signal value, we want to see what range of injected signals can give us the same observed signal. We can use very similar code to achieve this and we get something that looks like:

```
injected = np.array([])
signal = 190
for i in range(size):
    for j in range(size):
        if abs(obssignal[i][j] - signal) < 0.05:
            injected = np.append(injected, i)

figure()
plt.hist(injected, bins=150)
plt.title('Injected signals resulting in observed signal 190')
plt.ylabel('Occurences')
plt.xlabel('Injected signal')
plt.show()
```



Note that the most common value for the injected signal is 90. This makes sense since the most common background value is 100 and so 90 should be the most likely value to result in a measurement of 190.

3.4 d

Now we want to know what the one sigma uncertainty is for our true signal strength. The easiest way to do this is below:

```
sigma = np.std(injected)
print(sigma)
OUT: 10.546545418004225
```

So we can use 10.54 as our standard deviation.

3.5 e

This is symmetric about the middle value 90. This makes sense since the background distribution is symmetric as well. I don't know how to find the standard deviation analytically with Python so I used an array function. As such there are some limitations such as not being able to find a separate standard deviation on either side. I think in this case using the standard deviation function is alright because the distribution is generally symmetric so using the same sigma on either side is reasonable.

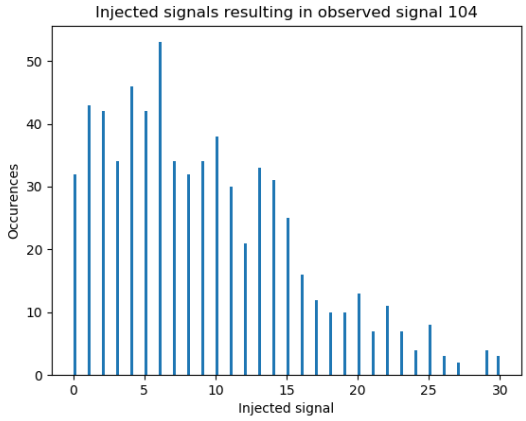
4 Problem 4

We're going to pick a weak signal (within 1σ) and see see how our analysis changes. Let our signal be 104, well within one standard deviation.

4.1 a

Let's repeat what we did in problem 3c but with our new signal. The code will be almost exactly the same except we change our injected signal value:

```
injected = np.array([])
signal = 104
for i in range(size):
    for j in range(size):
        if abs(obssignal[i][j] - signal) < 0.05:
            injected = np.append(injected, i)


figure()
plt.hist(injected, bins=150)
plt.title('Injected signals resulting in observed signal 104')
plt.ylabel('Occurences')
plt.xlabel('Injected signal')
plt.show()
```



Immediately we notice that this distribution extends to zero.

4.2 b

The range of signals extending to zero signifies that it is likely that the background entirely produced the measurement and the injected signal did not affect what was observed. In the case where the true signal equals zero and we still observed our chosen signal, we know the background produced the signal. This makes sense since we chose a value that is within one sigma and is very likely for the background to produce. This evidences why our observed measurement (104) cannot be called significant because it has a high likelihood of

being produced by the background. This is why we choose higher thresholds for significance.

4.3 c

Now we want to calculate a confidence bound. Specifically we're looking for a 95% confidence *upper* bound. What this means is we're looking for some signal X where we can say that 95% of the occurrences will be signals less than X. Because our distribution is not continuous, we need to count how many occurrences we have behind each value and divide by the total number to find the percentage. We do this until we are relatively close to 95% (within 0.001) then take that signal value. Here's the code:

```
occurences = len(injected)
sortinject = np.sort(injected)

count = 0
for i in range(occurences):
    x = sortinject[i]
    count = count + 1
    if abs(count/occurences - 0.95) < 0.001:
        print(x)
        i = occurences
```

For one trial with the observed signal being 104, the total number of occurrences was 654 with 21.0 being the 95% upper confidence bound. This means that if I observe the candidate signal 104 (and it is too weak to claim a detection), then the true signal should be less than 21.0 95% of the time.