Question 1

We begin my making a few assumptions.

- Lets assume that the object we are measuring emits photons at a constant rate. Thus the density of the number of photons hitting each detector is assumed to be independent of time
- Lets assume that the number of photons hitting one detector is independent of the number of photons hitting another detector.

Part 1

Under these assumptions we have (where θ is a vector of all our parameters s, x_0, w, σ, d)

$$P(n\vec{|}\theta) = \prod_i f(n_i|\nu_i(\theta))$$

where f is the Poisson distribution and the ith index represents the ith data point. Hence

$$\ln P(n\vec{|}\theta) = \sum_i \ln\left(\frac{\nu_i^{n_i} e^{-\nu_i}}{n_i!}\right) = \sum_i (n_i \ln \nu_i - \nu_i - \ln(n_i!))$$

Part 2

We also must consider our prior probabilities of our parameters $P(\theta)$. We can express this as

$$P(\theta) = \mathcal{N}(\sigma, \sigma_0, \sigma_\sigma) \cdot \prod_{i=1}^{4} U_i$$

where $\mathcal{N}$ is a normal distribution and the U_i 's are normal distributions of our 4 other parameters (and broad enough such that they are independent of the parameters in θ). Then we have

$$\begin{aligned}\ln P(\theta) &= \ln \mathcal{N} + C \\ &= -\frac{1}{2}\frac{(\sigma - \sigma_0)^2}{\sigma_\sigma^2} + C\end{aligned}$$

where C is some constant independent of the parameters in θ (note that C will include σ_0 since this is a normalization parameter in $\mathcal{N}$).

Part 3

Noting that $P(\vec{n})$ is independent of θ, using Baye's theorem we get

$$P(\theta|\vec{n}) = \frac{P(\vec{n}|\theta)P(\theta)}{P(\vec{n})}$$

and thus

$$\begin{aligned}\ln(P(\theta|\vec{n})) &= \ln P(\vec{n}|\theta) + \ln P(\theta) - \ln P(\vec{n}) \\ &= \sum_i (n_i \ln \nu_i - \nu_i) - \frac{1}{2}\frac{(\sigma - \sigma_0)^2}{\sigma_\sigma^2} + C'\end{aligned}$$

where C' is some constant independent of our parameters.

Part 4

What if any of the parameters are less than zero? Then $P(\theta) = 0$ and hence

$$P(\theta|\vec{n}) = \frac{P(\vec{n}|\theta)P(\theta)}{P(\vec{n})} = 0$$

Question 2

Import packages.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf
from scipy.stats import uniform
import pandas as pd
from pandas.plotting import scatter_matrix
```

Load data, get left and right coordinates of pixels, and define all values.

In [2]:

```
data = np.loadtxt('datafile_A4.txt')
N = len(data)

# Left coord of pixels
a = np.arange(0, 10, 0.5)
# Right coord of pixels
b=a+0.5


# Parameters
sig0 = 0.4
sigsig = 0.2

s_NOM = 200
x0_NOM = 4.5
w_NOM = 2.5
sig_NOM = 0.4
d_NOM = 12

# Hypercube bounds
ds = 10
dx0= 0.05
dw = 0.1
dsig = 0.05
dd = 1
```

Define functions that return the count density for each pixel, and the logarithm of the probability function.

In [3]:

```
def vi(i, s, x0, w, sig, d):
    # Left and right pixel location of point i
    ai = a[i]; bi = b[i]

    # Evaluating the erf integral
    ai_p = (ai-(x0-0.5*w))/(np.sqrt(2)*sig)
    bi_p = (bi-(x0-0.5*w))/(np.sqrt(2)*sig)
    ai_pp = (ai-(x0+0.5*w))/(np.sqrt(2)*sig)
    bi_pp = (bi-(x0+0.5*w))/(np.sqrt(2)*sig)

    component_1 = s*sig/(np.sqrt(2)*w)* ( bi_p*erf(bi_p)+np.exp(
-bi_p**2)/np.sqrt(np.pi)
                                           -(ai_p*erf(ai_p)+np.exp
(-ai_p**2)/np.sqrt(np.pi))
                                           -(bi_pp*erf(bi_pp)+np.e
xp(-bi_pp**2)/np.sqrt(np.pi))
                                           +ai_pp*erf(ai_pp)+np.ex
p(-ai_pp**2)/np.sqrt(np.pi) )
    component_2 = d*(bi-ai)
    return component_1+component_2

def lnP(s, x0, w, sig, d):
    return np.sum([
        data[i]*np.log(vi(i, s, x0, w, sig, d))-vi(i, s, x0, w,
sig, d)
        for i in range(N)]) -0.5*((sig-sig0)**2 / (sigsig**2))
```

The function below is the Markov Chain Monte Carlo method. It starts at the nominal values, then makes num_points steps where each step is given by

$$\theta_{i+1} = \begin{cases} \theta_{new} & \ln\theta_{new} > \ln\theta_i \\ \theta_{new} & \text{with } e^{(\ln(P(\theta_{new} - \ln P(\theta_i)))} \text{probability if } \ln\theta_{new} < \ln\theta_i \\ \theta_i & \text{with } e^{(\ln(P(\theta_i - \ln P(\theta_{new})))} \text{probability if } \ln\theta_{new} < \ln\theta_i \end{cases}$$

θ_0 is the nominal values we have specified.

In [4]:

```
def param_arrays(num_points):

    # Get stating log likelihood
    lnP_curr = lnP(s_NOM, x0_NOM, w_NOM, sig_NOM, d_NOM)
    # Define arrays and fill with the first value
    s_array = [s_NOM]
    x0_array = [x0_NOM]
    w_array = [w_NOM]
    sig_array = [sig_NOM]
    d_array = [d_NOM]
    accept_array = [] # Counts number of acceptances

    # Start loop
    for i in range(num_points):

        # Get new values of parameters
        s_new = s_array[-1] + np.random.random()*2*ds - ds
        x0_new = x0_array[-1] + np.random.random()*2*dx0 - dx0
        w_new = w_array[-1] + np.random.random()*2*dw - dw
        sig_new = sig_array[-1] + np.random.random()*2*dsig - ds
ig
        d_new = d_array[-1] + np.random.random()*2*dd - dd

        # Check if negative. If so, keep old values and continue
to next loop.
        if(s_new<0 or x0_new<0 or w_new<0 or sig_new<0 or d_new<
0):
            accept_array.append(True)

            s_array.append(s_array[-1])
            x0_array.append(x0_array[-1])
            w_array.append(w_array[-1])
            sig_array.append(sig_array[-1])
            d_array.append(d_array[-1])
            continue

        # Compute new log likelihood
        lnP_new = lnP(s_new, x0_new, w_new, sig_new, d_new)

        # Compute ratio
        ratio = np.exp(lnP_new-lnP_curr)
        accept = ratio > np.random.random()

        # If ratio implies accept new points
        if accept:
            accept_array.append(True) # Keep track of accepted p
oints
```

```
            s_array.append(s_new)
            x0_array.append(x0_new)
            w_array.append(w_new)
            sig_array.append(sig_new)
            d_array.append(d_new)

            lnP_curr = lnP_new

        # If ratio implies reject new points
        else:
            accept_array.append(False) # Keep track of rejected 
points

            s_array.append(s_array[-1])
            x0_array.append(x0_array[-1])
            w_array.append(w_array[-1])
            sig_array.append(sig_array[-1])
            d_array.append(d_array[-1])

    return s_array, x0_array, w_array, sig_array, d_array, accep
t_array
```

Question 3

Obtain arrays for 50000 MCMC steps.

In [5]:

```
s_arr, x0_arr, w_arr, sig_arr, d_arr, acc_arr = param_arrays(500
00)
```

Check that the number of accepted steps is around 66%.

In [6]:

```
np.count_nonzero(acc_arr)/len(acc_arr)
```

Out[6]:

```
0.66228
```

Create DataFrame containing all values.

In [7]:

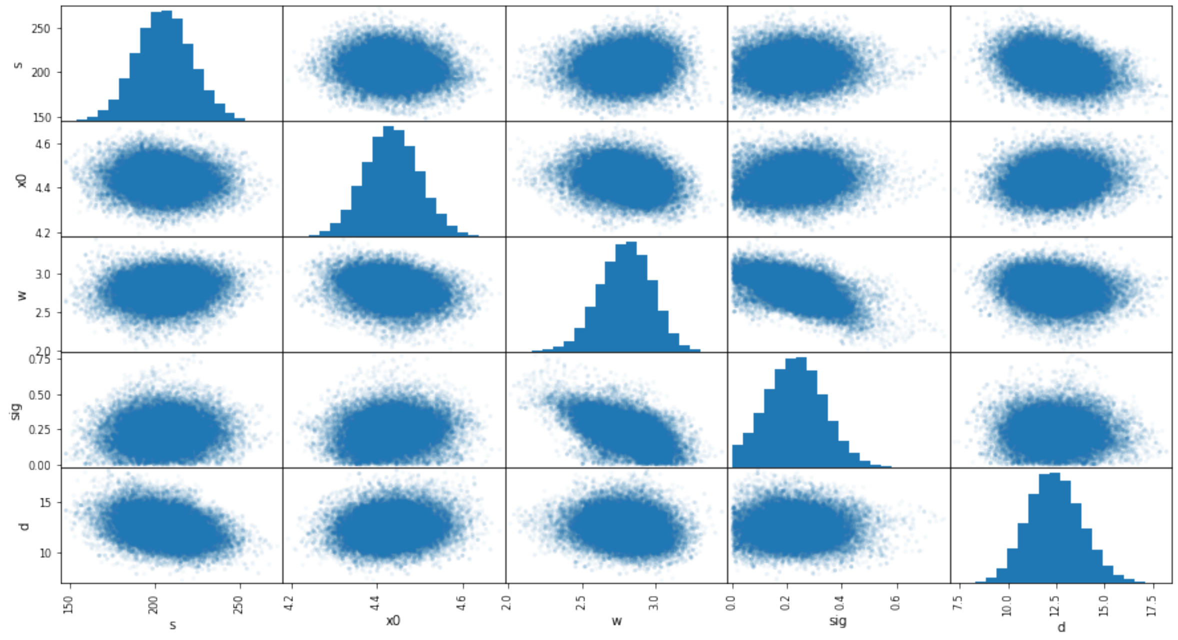
```
df = pd.DataFrame({'s':s_arr, 'x0':x0_arr, 'w':w_arr, 'sig':sig_
arr, 'd':d_arr})
```

Correlations

Make scatter matrix plot. It's important to understand what this plot actually means. We are approximating the distribution from which the data came from. The complicated distribution has 5 different parameters. As some of these parameters increase, others will decrease (given the data there will be certain combinations of parameters that are represent the most likely distribution).

In [8]:

```
scatter_matrix(df, alpha=0.05, figsize=(15,8), hist_kwds={'bins'
:20})
plt.show()
```



This is a more clear correlation matrix plot.

In [9]:

```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm', axis=None)
```

Out[9]:

	s	x0	w	sig	d
s	1	-0.0900934	0.107953	0.0636274	-0.319743
x0	-0.0900934	1	-0.253258	0.149148	0.114724
w	0.107953	-0.253258	1	-0.562313	-0.153341
sig	0.0636274	0.149148	-0.562313	1	-0.0417943
d	-0.319743	0.114724	-0.153341	-0.0417943	1

Lets discuss the meaningful correlations.

- σ shows a strong negative correlation with w. This can be explained as follows. An object measured by the telescope will have an approximate "width" (not w) in the image. This width is a combination of both the true width of the object w and the smearing parameter σ. The smearing with σ makes the width larger, so if we assume w to be larger then we must have σ smaller since the "width" in the image remains constant.
- The number of counts s and the number of dark counts d have a negative correlation. This can be explained as follows. The true data measured in each pixel comes from a combination of true counts s and dark counts d. After data is measured, the number of counts recorded in a given pixel is fixed; if we we assume we had more true counts then we must have less dark counts to keep the recorded number of counts fixed. It is not perfectly negatively correlated since the contribution from true counts f_s follows a very different ditribution than d (uniform).

These are the most important correlations observed.

Intervals and Medians

Now we get the intervals and medians.

- 5% is the lower bound of the 90% central confidence interval.
- 95% is the upper bound of the 90% central confidence interval.
- 50% is the median value.

I know you say that numpy makes this easy to do, but pandas makes this *really* easy to do.

Since our original guess for sigma was 0.4, our guess is somewhat improved, since our initial guess was $\sigma = 0.4$ with standard deviation 0.2 and normally distributed but our new estimate is 0.23 with standard deviation approximately equal to 0.085 (though it may not be normally distributed).

In [10]:

```
df.describe(percentiles=[.05, .95]).loc[['5%', '50%', '95%']]
```

Out[10]:

	s	x0	w	sig	d
5%	178.900148	4.332001	2.494832	0.060452	10.250256
50%	205.283110	4.435212	2.801045	0.229827	12.423033
95%	233.019826	4.542040	3.070348	0.405909	14.838263

Since our original guess for sigma was 0.4, our guess is somewhat improved, since our initial guess was $\sigma = 0.4$ with standard deviation 0.2 and normally distributed but our new estimate is 0.23 with standard deviation approximately equal to 0.085 (though it may not be normally distributed).