

Solutions_5

October 26, 2018

1 Solution Exercise 5

This week, we are working on least squares fits and parameter estimation

```
In [1]: from __future__ import print_function
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.stats import norm, chi2, lognorm
```

1.1 Fit a polynomial

We start by fitting a polynomial to a given data set, in particular, a parabola. Compare a linear fit and a cubic fit to our parabolic fit and check the goodness of fits with chi squared distributions. Explore how the different uncertainties affect the outcome and uncertainties of the fit.

Hint: You can consider a plot similar to the lecture notes week 5 page 29.

Extra: Do you see any way to decide whether the data is better described by the parabola or the cubic?

```
In [2]: # Create some data distributed as parabola with normally distributed errors.
def parabola(x, a, b, c):
    return a*x**2 + b*x + c
def error(x, sigma):
    return norm.rvs(0.0, sigma, x.size)
a = -0.1
b = 0
c = 1
sigma_y = 0.0015

x = np.linspace(0, 1, 21)
y_true = parabola(x, a, b, c)
delta_y = error(x, sigma_y)
y = y_true + delta_y
y_error = sigma_y * np.ones(x.size)

In [3]: def fit_polynomial(x, y, degree, weight):
        """Fit polynomial of degree to data x, y with y weight = 1/sigma_y
```

Return fit, covariance matrix, residuals and chi-squared and degrees of freedom.
"""

```
dof = x.shape[0] - degree
fit, cov = np.polyfit(x, y, degree, w=weight, cov=True)
residuals = np.sum((y - np.polyval(fit, x))**2 / y_error**2)
chisq = residuals / (dof)
return fit, cov, residuals, chisq, dof
```

```
fit, cov, res, chisq, dof = fit_polynomial(x, y, 2, 1/y_error) # Fit parabola
fit_1, cov_1, res_1, chisq_1, dof_1 = fit_polynomial(x, y, 1, 1/y_error) # Fit line
fit_3, cov_3, res_3, chisq_3, dof_3 = fit_polynomial(x, y, 3, 1/y_error) # Fit cubic
```

```
def evaluate_chisq(chisq, dof):
    return chi2.sf(chisq, dof)
```

```
In [4]: print('Reduced chi^2:')
        print('parabola', chisq)
        print('line', chisq_1)
        print('cubic', chisq_3)
```

```
Reduced chi^2:
parabola 0.9459782161747974
line 32.15822425109619
cubic 0.9557527151822285
```

```
In [5]: print('Chi^2 distributions:')
        evaluate_chisq(chisq, dof), evaluate_chisq(chisq_1, dof_1), evaluate_chisq(chisq_3, dof_3)
```

```
Chi^2 distributions:
```

```
Out[5]: (0.9999999995308976, 0.04164125577461551, 0.9999999976681199)
```

```
In [6]: print('Error estimates:')
        np.sqrt(np.diag(cov)), np.sqrt(np.diag(cov_1)), np.sqrt(np.diag(cov_3))
```

```
Error estimates:
```

```
Out[6]: (array([0.00424588, 0.00439779, 0.00094897]),
        array([0.00664986, 0.00388699]),
        array([0.0162819 , 0.0247968 , 0.01051785, 0.00118487]))
```

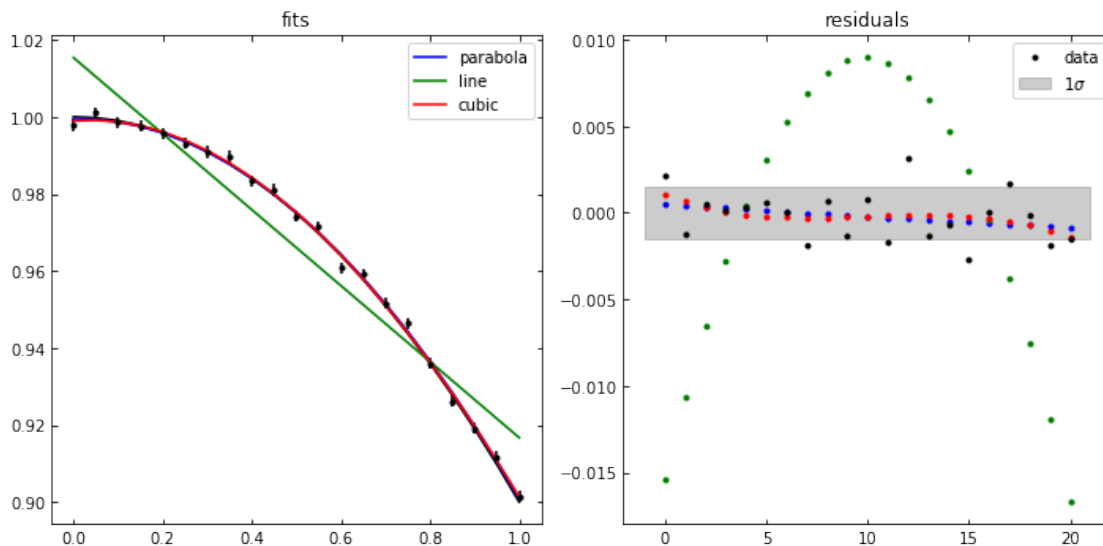
```
In [7]: f, ax = plt.subplots(1, 2, figsize=(10, 5))
        ax[0].set_title('fits')
        ax[0].errorbar(x, y, yerr=y_error, fmt='k.')
```

```

ax[0].plot(x, y_true, 'k-')
ax[0].plot(x, np.polyval(fit, x), label='parabola', color='blue')
ax[0].plot(x, np.polyval(fit_1, x), label='line', color='green')
ax[0].plot(x, np.polyval(fit_3, x), label='cubic', color='red')

ax[0].legend()
ax[1].plot(y_true - np.polyval(fit, x), '.', color='blue')
ax[1].plot(y_true - np.polyval(fit_1, x), '.', color='green')
ax[1].plot(y_true - np.polyval(fit_3, x), '.', color='red')
ax[1].plot(y_true - y, 'k.', label='data')
ax[1].set_title('residuals')
ax[1].fill_between(ax[1].get_xlim(), -sigma_y, sigma_y, color='grey', alpha=0.4, label='1σ')
ax[1].legend()
f.tight_layout()

```

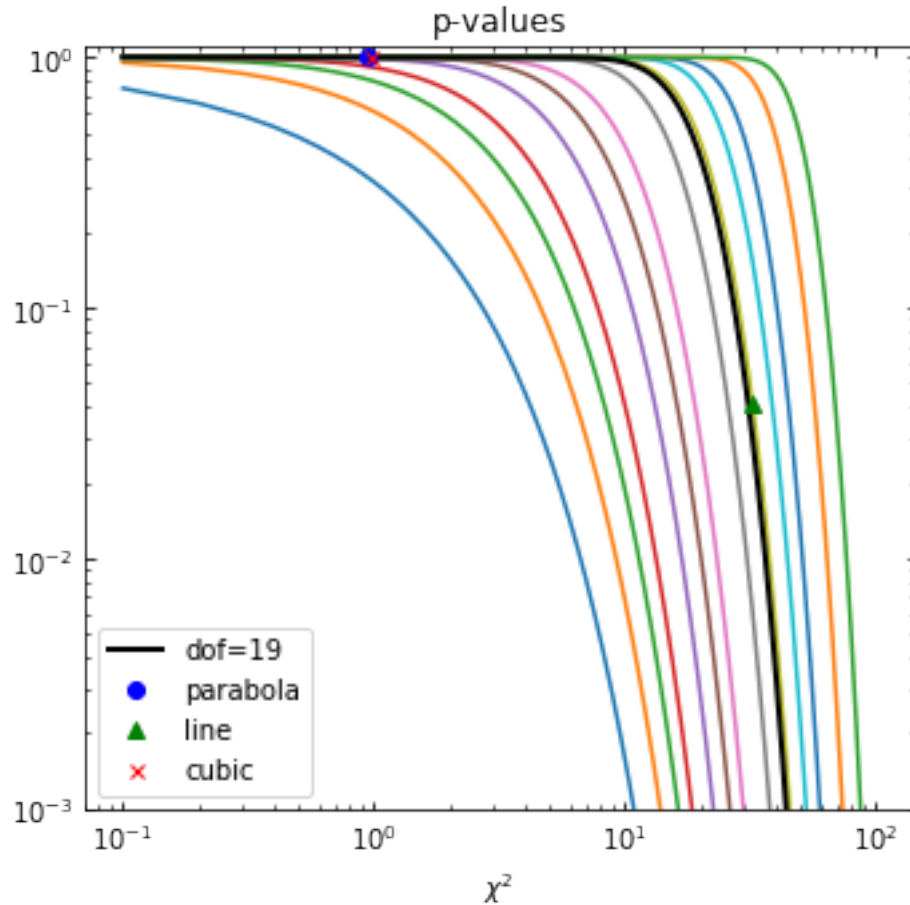


In [8]: *# Draw the plot from lecture notes week 5, page 29*

```

plt.figure(figsize=(5, 5))
chisq_arr = np.linspace(0, 100, 1001)
plt.title('p-values')
plt.xlabel(r'$\chi^2$')
for n in [1, 2, 3, 4, 6, 8, 10, 15, 20, 25, 30, 40, 50]:
    plt.loglog(chisq_arr, chi2.sf(chisq_arr, n))
plt.loglog(chisq_arr, chi2.sf(chisq_arr, dof), 'k-', lw=2, label='dof={0}'.format(dof))
plt.ylim(1e-3, 1.1)
plt.plot(chisq, chi2.sf(chisq, dof), 'o', color='blue', label='parabola')
plt.plot(chisq_1, chi2.sf(chisq_1, dof_1), '^', color='green', label='line')
plt.plot(chisq_3, chi2.sf(chisq_3, dof_3), 'x', color='red', label='cubic', ms=5)
plt.legend()
plt.tight_layout()

```



We observe that the residuals are not uniformly distributed around zero with the expected variance σ_y in the case of the line fit. This reflected in the χ^2 -distribution. Only in 7% of the cases we would expect to draw data that give a worse fit. Note that overfitting with a cubic is not easily spotted in the residuals. However we do observe higher errors on the parameter estimates in the cubic case and we could try a different approach: let's fit only a subset of our data and then compare the fit results on the complement.

```
In [9]: fit, cov, res, chisq, dof = fit_polynomial(x, y, 2, 1/y_error) # Fit parabola
        fit_1, cov_1, res_1, chisq_1, dof_1 = fit_polynomial(x, y, 1, 1/y_error) # Fit line
        fit_3, cov_3, res_3, chisq_3, dof_3 = fit_polynomial(x, y, 3, 1/y_error) # Fit cubic

x_new = np.linspace(1, 2, 21)
y_new = parabola(x_new, a, b, c) + error(x_new, sigma_y)

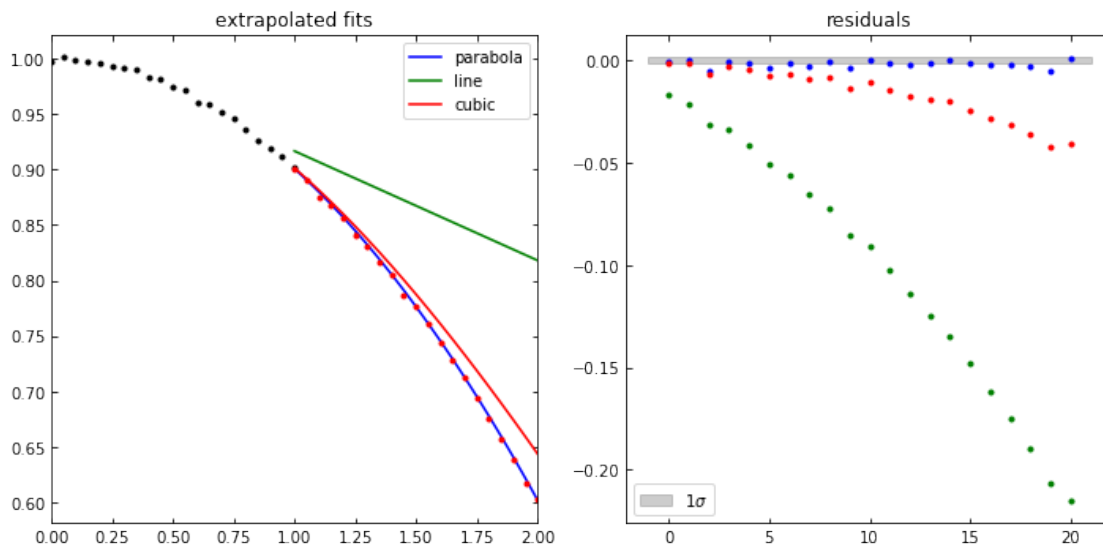
f, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].set_title('extrapolated fits')
ax[0].plot(x, y, 'k.')
ax[0].errorbar(x_new, y_new, yerr=y_error, fmt='r.')
```

```

ax[0].plot(x_new, np.polyval(fit, x_new), label='parabola', color='blue')
ax[0].plot(x_new, np.polyval(fit_1, x_new), label='line', color='green')
ax[0].plot(x_new, np.polyval(fit_3, x_new), label='cubic', color='red')

ax[0].legend()
ax[0].set_xlim(0, 2)
ax[1].plot(y_new - np.polyval(fit, x_new), '.', color='blue')
ax[1].plot(y_new - np.polyval(fit_1, x_new), '.', color='green')
ax[1].plot(y_new - np.polyval(fit_3, x_new), '.', color='red')
ax[1].set_title('residuals')
ax[1].fill_between(ax[1].get_xlim(), -sigma_y, sigma_y, color='grey', alpha=0.4, label='1σ')
ax[1].legend()
f.tight_layout()

```



Here it becomes very obvious that the hypothesis of a parabola holds against a cubic. We cheated a bit by adding data points instead of working with the initial set, but this illustrates the point of this method. An overfitted model usually does not generalize well when presented with additional data.

1.2 Fit a nonlinear function

Next, we consider a Gaussian as an example of a nonlinear function. We are measuring some feature which has a Gaussian distribution in x . This could be an inhomogeneous spectral line for $x = E$ the energy of emitted photons. We are interested in the resonance frequency and the linewidth, i. e. we want to estimate them from our observations.

```

In [10]: def gaussian_parent(x, mu, sigma):
          return norm.pdf(x, mu, sigma)

          def gaussian_sample(mu, sigma, sample_size):
              return norm.rvs(mu, sigma, sample_size)

```

```

In [11]: # Create sample
        ## SAMPLE SIZE
        sample_size = 200
        #####

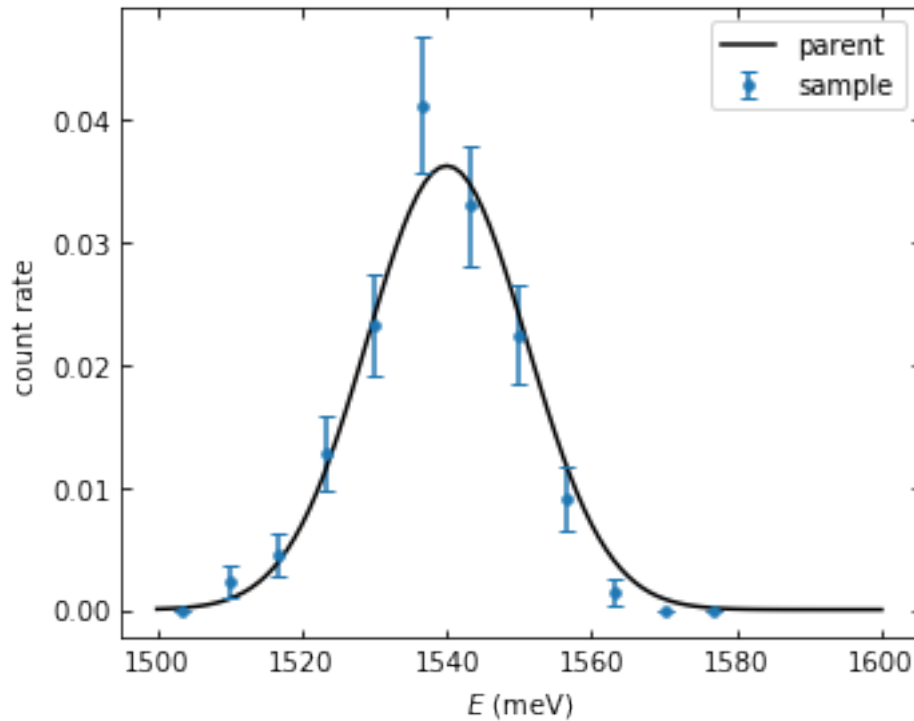
        # Prepare fake data
        mu = 1540 # True values that we will try to estimate
        sigma = 11 # using a least-squares fit

        x_arr = np.linspace(1500, 1600, 101)
        bins = 12
        sample = gaussian_sample(mu, sigma, sample_size)
        hist = np.histogram(sample, bins=bins, range=(1500, 1580))
        bin_width = np.diff(hist[1])[0]
        normalization = bin_width * sample_size
        x = hist[1][:-1] + bin_width/2
        y_error_const = 0
        y = hist[0]/normalization + gaussian_sample(0, y_error_const, bins)
        y_errors = np.sqrt((np.sqrt(hist[0]) / normalization)**2 + y_error_const**2)

        # Plot our fake measurement results
        plt.figure(figsize=(5, 4))
        plt.xlabel(r'$E$ (meV)')
        plt.ylabel('count rate')
        plt.plot(x_arr, gaussian_parent(x_arr, mu, sigma), '-', color='black', label='parent')
        plt.errorbar(x, y, yerr=y_errors, fmt='.', ms=7, capsize=3, label='sample')
        plt.legend()
        plt.tight_layout()

        # Save data
        data = np.vstack((x, y, y_errors))
        np.savetxt('data', data)
        np.savetxt('sample', sample)

```



In [12]: *# Load data from disk. Format (3,12): (x, y, y_error) x N*

```
data = np.loadtxt('data')
x = data[0, :]
y = data[1, :]
y_error = data[2, :]
# The sample used to generate
sample = np.loadtxt('sample')
```

In [13]: *# Function we want to fit to our data set*

```
def model_function(x, *args):
    mu, sigma = args[0:2]
    return norm.pdf(x, mu, sigma)
```

In [14]: *# Perform the fit minimizing least squares*

```
initial_guess = [1545, 9]
p_opt, p_cov = curve_fit(model_function, x, y, p0=initial_guess, sigma=None, absolute_sigma=True)
p_err = np.sqrt(np.diag(p_cov))
# pcov(absolute_sigma=False) = pcov(absolute_sigma=True) * chisq(popt)/(M-N)
print('Fit Results:')
print('mu = {:.1f} +- {:.1f}'.format(p_opt[0], p_err[0]))
print('sigma = {:.1f} +- {:.1f}'.format(p_opt[1], p_err[1]))

print('mu estimator {:.1f} +- {:.1f}'.format(np.mean(sample), np.std(sample, ddof=1)))
print('sigma estimator {:.1f}'.format(np.std(sample, ddof=1)))
```

Fit Results:

$\mu = 1539.0 \pm 0.4$

$\sigma = 10.3 \pm 0.3$

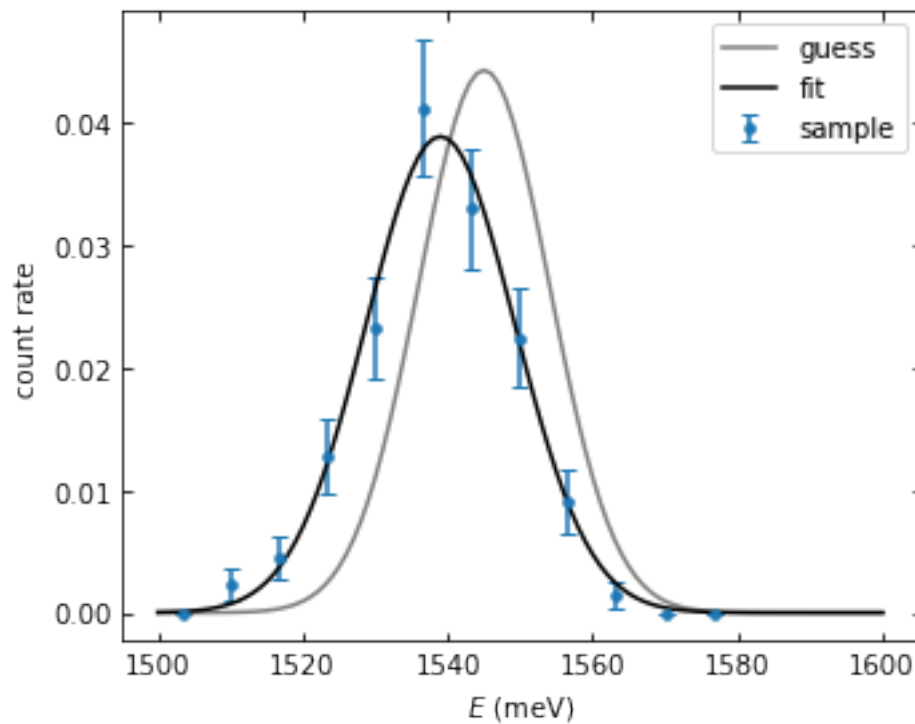
mu estimator 1538.4 \pm 0.7

sigma estimator 10.4

1.2.1 Plot the result

```
In [15]: x_arr = np.linspace(1500, 1600, 101)
```

```
plt.figure(figsize=(5, 4))
plt.xlabel(r'$E$ (meV)')
plt.ylabel('count rate')
plt.errorbar(x, y, yerr=y_errors, fmt='.', ms=7, capsize=3, label='sample')
plt.plot(x_arr, model_function(x_arr, *initial_guess), '-', color='grey', label='guess')
plt.plot(x_arr, model_function(x_arr, *p_opt), '-', color='black', label='fit')
plt.legend()
plt.tight_layout()
```



1.3 Biased estimator example

```
In [16]: log_sample = lognorm.rvs(s=0.5, loc=3, scale=1, size=1000)
plt.figure(figsize=(5, 4))
```



```

h = plt.hist(log_sample, bins=32, rwidth=0.85, normed=True)

def model_function(x, *args):
    A, mu, sigma = args[0:3]
    return A * norm.pdf(x, mu, sigma)

x = h[1][:-1]+np.diff(h[1])[0]/2
y = h[0]
initial_guess = [0.95, 3.3, 0.3]
gauss_fit_large = curve_fit(model_function, x, y, p0=initial_guess)
#local = np.where(np.abs(x-np.mean(log_sample))<0.5)
#gauss_fit_local = curve_fit(model_function, x[local], y[local], p0=initial_guess)

x_arr = np.linspace(2, 5, 51)
plt.plot(x_arr, model_function(x_arr, *gauss_fit_large[0]), '-', color='red', label='Gauss fit')
#plt.plot(x[local], model_function(x[local], *gauss_fit_large[0]), '-', color='orange', label='Local fit')
plt.legend()
plt.tight_layout()

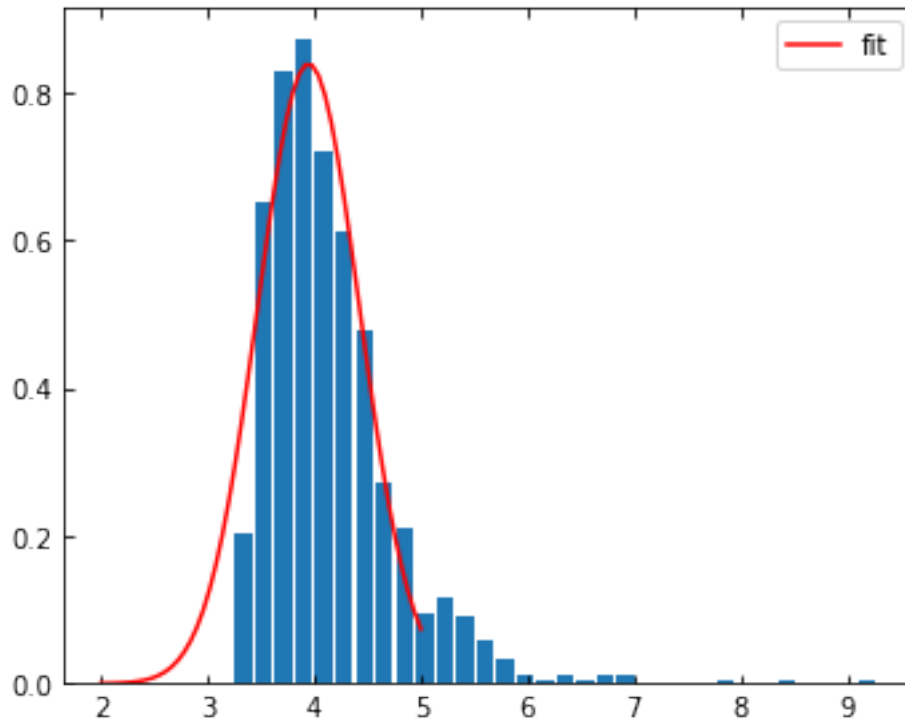
print('Expectation value:', lognorm.mean(s=0.5, loc=3, scale=1))
print('Sample mean:', np.mean(log_sample))
print('Gauss fit:', gauss_fit_large[0][1])

```

```

Expectation value: 4.133148453066826
Sample mean: 4.144495254000121
Gauss fit: 3.946186931764501

```



Fitting a Gaussian and gives a biased estimate of μ

1.4 Bonus: Errors in x and y

So far, we considered uncertainties only on y . Consider the following data set with errors in both x and y . Try to fit a line to the data below, taking into account both errors. Compare with fits neglecting the x errors or both.

Hint: A detailed solution is already on the moodle. You may choose if you want to try to write your own solution, implement a known solution (see references in solution notebook) or just try it with the scipy package ODR (orthogonal distance regression). <https://docs.scipy.org/doc/scipy/reference/odr.html>

The solution contains a python implementation of York's equation, comparison with ODR and MC tests.

Week 3: "Linear Regression errors x and y "

```
In [17]: # Test data
X = np.array([0.0, 0.9, 1.8, 2.6, 3.3, 4.4, 5.2, 6.1, 6.5, 7.4])
Y = np.array([5.9, 5.4, 4.4, 4.6, 3.5, 3.7, 2.8, 2.8, 2.4, 1.5])
wX = np.array([1000, 1000, 500, 800, 200, 80, 60, 20, 1.8, 1])
wY = np.array([1, 1.8, 4, 8, 20, 20, 70, 70, 100, 500])
sigma_x = 1.0/np.sqrt(wX)
sigma_y = 1.0/np.sqrt(wY)

plt.figure(figsize=(4, 3))
```

```
plt.errorbar(X, Y, xerr=sigma_x, yerr=sigma_y, fmt='o', label='observations')  
plt.legend()  
plt.tight_layout()
```

