1_curve_fitting

January 8, 2024

1 Curve fitting

We stop for the moment the physics related stuff and have a look at a different important topic, which is curve fitting. We demonstrate the least-square fitting of a quadratic function with three parameters to experimental data. You may of course also have more complex function or even a simple linear function. For some fitting functions you may write down explicit estimators of the parameters and you do not have to stress the fitting procedure. So before you fit, think about how to obtain a good estimate of your model parameters. For those of you, who are interested a bit more in that topic, have a look at maximum likelihood estimation for example.

[3]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
plt.rcParams.update({'font.size': 18})
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

1.1 Idea

The result of an experiment are data points from which you would like to understand the physics behind, meaning you would like to see if a mathematical model fits your data.

So the data comes as a series of points, usually pairs of points such as

x-data	y-data
x_1	y_1
x_2	y_2
x_3	y_3
….	…
x_N	y_N

Here, each of the point $\{x_1, y_1\}$ could be the result of a series of independent measurements, i.e. $y_{1,i}$ as well. The independent measurements yield mean values

$$y_1 = \frac{1}{N}\sum_i^N y_{1,i} \tag{1}$$

If these measurements have been carried out with an uncertainty σ for the individual measurements, then the sum of all measurements $\sum_i y_{1,i}$ has a variance of $N\sigma^2$ and a standard deviation of $\sqrt{N}\sigma$. The mean value is therefore connected to an error (standard deviation) of

$$\sigma_{SEOM} = \frac{\sigma}{\sqrt{N}} \tag{2}$$

This is the standard error of the mean (SEOM) and it has importance across all measurements in physics. For later, note that the variance is defined by

$$\sigma_1^2 = \frac{1}{N}\sum_{i=1}^N (y_{1,i} - y_1)^2 \tag{3}$$

1.2 Least squares

If we would now like to describe our data with a model function, which delivers a function value $f(x_i, a)$ for a set of parameters a at the position x_i, the Gaussian uncertainty dictates a probability

$$p_{y_i} = \frac{1}{\sqrt{2\pi}\sigma_i}\exp(-(y_i - f(x_i,a))^2/2\sigma_i^2) \tag{4}$$

of finding a data value y_i. Note that I generalized here the uncertainty, which is now valid for the each point individually.

If you now want to know how close a set of N data points is to a set of function values, you have to multiply the individual probabilities:

$$p(y_1, \dots, y_N) = \prod_i^N \frac{1}{\sqrt{2\pi}\sigma_i}\exp(-(y_i - f(x_i,a))^2/2\sigma_i^2) \tag{5}$$

If this joint probability is maximum, you will have the closest match of the function values to the data.

Applying the logarithm to both side of the equation results in

$$\ln(p(y_1, \dots, y_N)) = -\frac{1}{2}\sum_{i=1}^N \left(\frac{y_i - f(x_i,a)}{\sigma_i}\right)^2 - \sum_{i=1}^N \ln\left(\sigma_i\sqrt{2\pi}\right) \tag{6}$$

The first term on the right side (except the factor 1/2) is the least squared deviation

$$\chi^2 = \sum_{i=1}^N \left(\frac{y_i - f(x_i,a)}{\sigma_i}\right)^2 \tag{7}$$

The second term is just a constant value given by the uncertainties of our experimental data.

1.3 Data

Lets have a look at the meaning of this equation. Lets assume we measure the trajectory of a ball that has been thrown under and angle α with an initial velocity v_0. We have collected data point by measuring the height of the ball above ground at equally spaced distances from the throwing person. Lets load some data

[4]:
```python
# Here is some data of the height measurements including untertainties
x_data,y_data,err=np.loadtxt('data.txt',unpack=True)
```

We can plot the data and expect, of course,a parabola. Therefore we model our experimental data with a parabola like

$$y = ax^2 + bx + c \tag{8}$$

where the parameter a must be negative since the parabola is inverted.

I have created an interactive plotting with an interact widget, as this allows you to play around with the parameters. The value of $\chi^2$ is also included in the legend, that you get an impression of how good your fit of the data is.

[7]:
```python
def parabola(x,a,b,c):
    return(a*x**2+b*x+c)

def plot(a,b,c):
    y=parabola(x,a,b,c)
    plt.figure(figsize=(8,6))
    chisq=(((y_data-parabola(x_data,a,b,c))/err)**2).sum()
    plt.plot(x,y,label='$\chi^2$={0:6.3f}'.format(chisq))
    plt.errorbar(x_data,y_data,yerr=err,marker='o',fmt="none",color='k')

    plt.scatter(x_data,y_data,marker='o',color='k')
    plt.legend()
    plt.xlabel('x- position')
    plt.ylabel('y- position')
    plt.show()
```

[8]:
```python
x=np.linspace(0,1,100)
interact(plot,a=-1.7,b=1.3,c=1.0);
```

```
interactive(children=(FloatSlider(value=-1.7, description='a', max=1.7, min=-5.
↪1), FloatSlider(value=1.3, desc…
```

We have that troubeling point at the right edge with a large uncertainty. However, since the value of $\chi^2$ divides the deviation by the uncertainty σ_i the weight for this point overall in the $\chi^2$ is smaller than for the other points.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i, a)}{\sigma_i} \right)^2 \tag{9}$$

You may simply check the effect by changing the uncertainty of the last data points in the error array.

1.4 Least square fitting

The best fit of the model to the experimental data is then obtained by minimizing the least squares, i.e.

$$\frac{d\chi^2}{da} = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \frac{df(x_i,a)}{da}[y_i - f(x_i,a)] = 0 \tag{10}$$

This kind of least squares minimization is done by a fitting software with different types of algorithms.

OK so let's do some fitting. We will use the `SciPy` module for fitting. There we have a `curve_fit` method in the `optimize` sub-module. At first we should provide a model function we would like to fit to the data. This could just be our parabola function.

[514]:
```
def parabola(x,a,b,c):
    return(a*x**2+b*x+c)
```

At second, we should not leave the fitting procedure of `SciPy` without a clue on where to look for the optimal parameters. Therefore we can provide initial parameters for the search.

[515]:
```
init_guess =[-1,1,1]
```

The fit is then obtained by calling

[516]:
```
fit=curve_fit(parabola,x_data,y_data,sigma=err,p0=init_guess,absolute_sigma=True)
```

and we obtain all the fit results in the variable `fit`. This is actually composed of various results. If we split that up, we will find

[517]:
```
ans,cov=fit
fit_a,fit_b,fit_c=ans
```

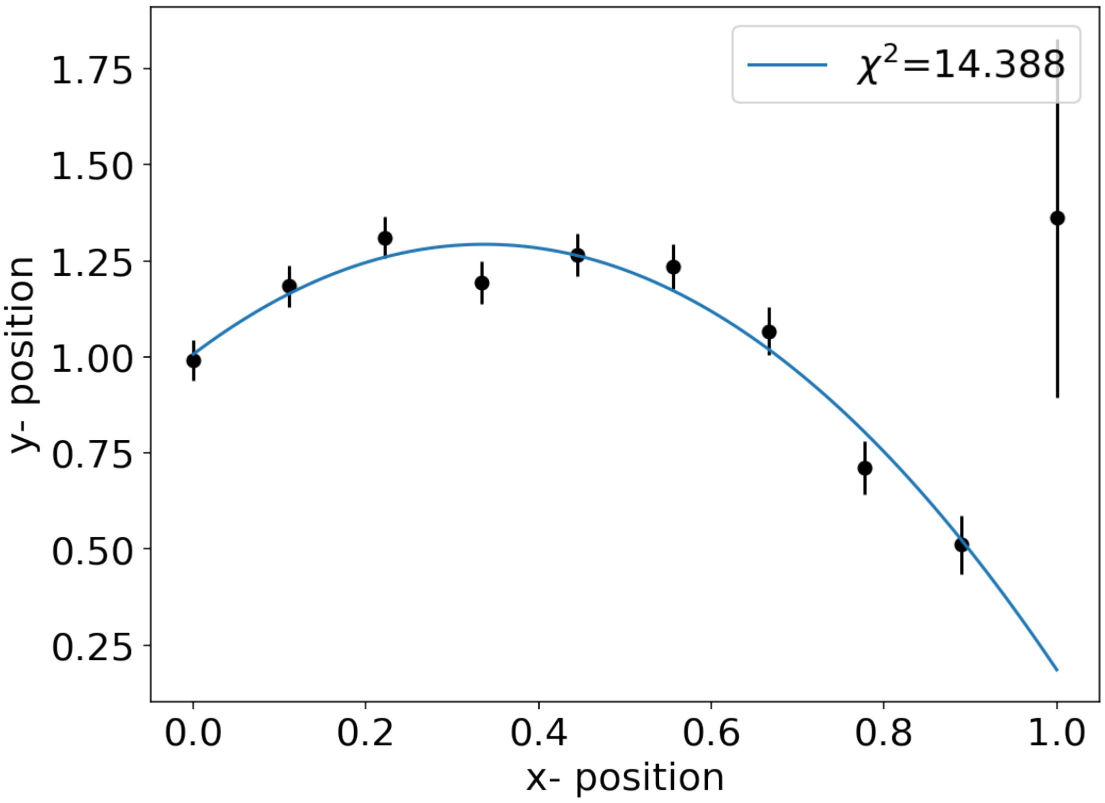
`ans` containing the fit parameters `fit_a`,`fit_b`,`fit_c` as well as the so-called covariance matrix `cov`. Lets have a look at the fit and the χ^2 value first.

[518]:
```
fit_a,fit_b,fit_c
```

[518]:
```
(-2.518360505820918, 1.6971754996789874, 1.0067886882158636)
```

[519]:
```
plt.figure(figsize=(8,6))
chisq=(((y_data-parabola(x_data,fit_a,fit_b,fit_c))/err)**2).sum()
plt.plot(x,parabola(x,fit_a,fit_b,fit_c),label='$\chi^2$={0:6.3f}'.
  ↪format(chisq))
plt.errorbar(x_data,y_data,yerr=err,marker='o',fmt="none",color='k',)
plt.scatter(x_data,y_data,marker='o',color='k')
```

```python
plt.xlabel('x- position')
plt.ylabel('y- position')
plt.legend()
plt.show()
```



1.4.1 χ-squared value

The value of χ^2 gives you a measure for the quality of the fit. We may judge the quality by calculating an expression of the expectation value of χ^2

$$\langle \chi^2 \rangle = \sum_{i=1}^{N} \frac{\langle (y_i - f(x_i, a))^2 \rangle}{\sigma_i^2} = \sum_{i=1}^{N} \frac{\sigma_i^2}{\sigma_i^2} = N \tag{11}$$

So the mean of the least squared deviation increases with the number of datapoints and thus

- $\chi^2 >> N$ means that the fit is bad
- $\chi^2 < N$ means that the uncertainties are wrong

The first may occur if you don't have a good fit to your data, for example, a wrong model. The second typically occurs if you don't have estimates of the uncertainties and you assume all uncertainties to be constant. So it is really important to have a good estimate of the uncertainties and to include it in do you fit. If you include the uncertainties in your fit it is called a `weighted` `fit` in case you don't include the uncertainties (meaning you keep them constant) it is called an `unweighted fit`.

For our fit above we obtain a χ^2 which is on the order of $N = 10$, which tells you that I have cheated well when creating the data.

1.4.2 Residuals

A similar view on the quality of the fit may be ontained from the residuals. These are defined as the deviation of the data from the model for the best fit.

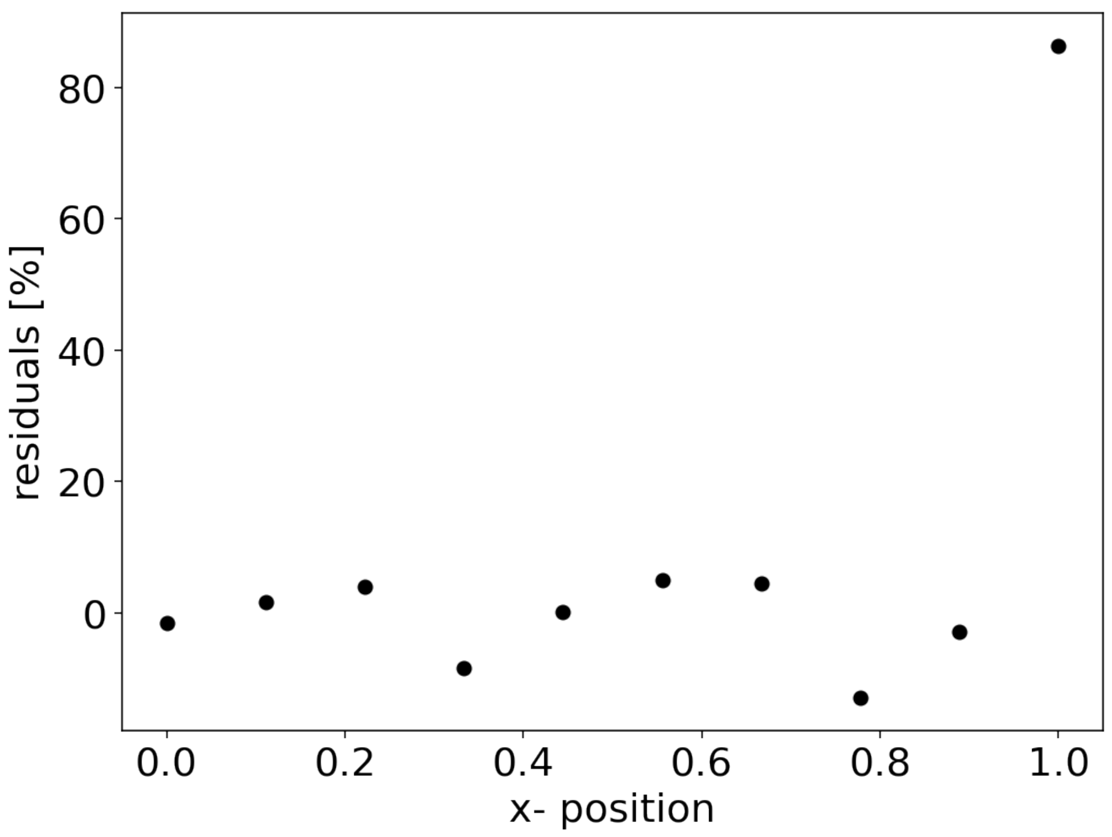
$$r_i = y_i - f(x_i, a) \tag{12}$$

The residuals may also be given as the percentage of the deviation of the data from the fit by

$$r_i = 100\left(\frac{y_i - f(x_i, a)}{y_i}\right) \tag{13}$$

If there are only statsitical fluctuations of the residuals around zero, then the fit and likely also the model is good.

[505]:
```
plt.figure(figsize=(8,6))
chisq=(((y_data-parabola(x_data,fit_a,fit_b,fit_c))/err)**2).sum()
plt.scatter(x_data,100*(y_data-parabola(x_data,fit_a,fit_b,fit_c))/
  ↪y_data,marker='o',color='k')
plt.xlabel('x- position')
plt.ylabel('residuals [%]')
plt.show()
```



1.5 Covariance matrix

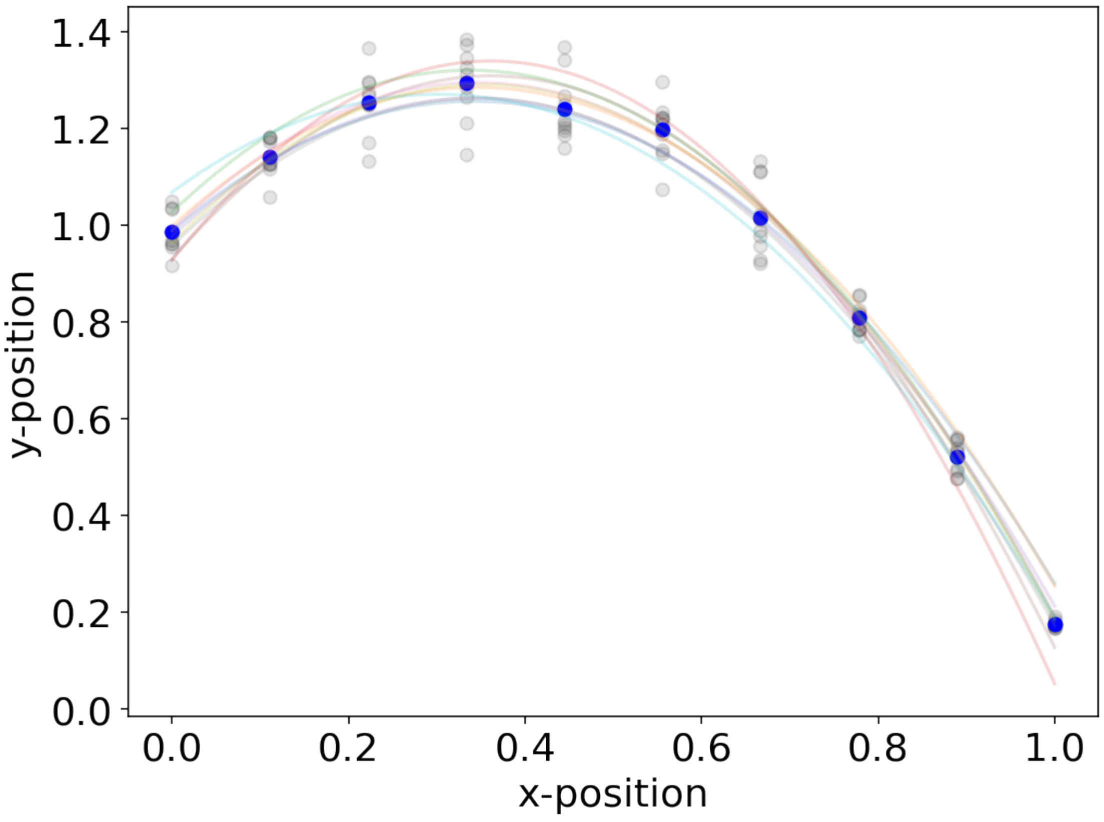
Let us now have a look at the individual measurements which have yielded the errorbars in the above plot. If I take each of those measurements and calculate a fit for each of the datasets I get a whole set of fit functions and parameters. The uncertainties in the parameters of the fit function are the result of the measurement uncertainty.

[535]:
```
def data(x,a,b,c):
    y=a*x**2+b*x+c
    err=[np.random.normal() for _ in range(len(x))]
    err=y*err*0.05
    return(y+err)
```

[522]:
```
x=np.linspace(0,1,10)
ym=np.zeros(10)
plt.figure(figsize=(8,6))

for _ in range(10):
    y=data(x,-2.52,1.6971755,1)
    ym=ym+y
    p,cov=curve_fit(parabola,x,y,sigma=err,p0=init_guess,absolute_sigma=True)
    plt.scatter(x,y,color='k',alpha=0.1)
    xf=np.linspace(0,1,100)
    plt.plot(xf,parabola(xf,p[0],p[1],p[2]),alpha=0.2)

plt.scatter(x,ym/10,color='b')
plt.xlabel('x-position')
plt.ylabel('y-position')
plt.show()
```



We may therefore want to characterize how much the individual parameters vary with each other. In other word, this means that we want to know whether the fit parameters are independent or not, which is a good quality measure of our model. For this pupose we use a generalization of the variance definition

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - < y >)^2 \tag{14}$$

which is the mean squared deviation of the individual values from its mean. This equation is a special case of a the so-called covariance

$$\mathrm{cov(x,y)} = \frac{1}{N}\sum_{i=1}^{N}(x_i - <x>)(y_i - <y>) \tag{15}$$

which measures by how much a variation from of x_i from the mean is also connected to a variation of y_i from the mean. The variance itself, is therefore just $\mathrm{cov}(x,x)$.

The `curve_fit` function delivers a covariance matrix as mentioned above. This covariance matrix is, however, not a measure of how the data varies among each other but rather a measure of how much the individual fit parameters varying with each other. If we fit our data with a model containing three parameters (a, b, c), then the covariance matrix of the parameters p_i and p_j with $i = a, b, c$ and $j = a, b, c$ is a 3×3 matrix.

$$\mathrm{cov}(p_i, p_j) = \begin{bmatrix} \sigma_{aa}^2 & \sigma_{ab}^2 & \sigma_{ac}^2 \\ \sigma_{ba}^2 & \sigma_{bb}^2 & \sigma_{bc}^2 \\ \sigma_{ca}^2 & \sigma_{cb}^2 & \sigma_{cc}^2 \end{bmatrix} \tag{16}$$

The diagonal elements thereby provide the squared errors of the fit parameters (their variances). The off diagonal elements describe by how much the individual parameters are related with each other.

[537]:
```python
print(cov)
```

```
[[ 0.07675961 -0.00252389 -0.00524745]
 [-0.00252389  0.0002834   0.0001206 ]
 [-0.00524745  0.0001206   0.00074961]]
```

The matrix above is our covariance matrix of the parameters. You see from the off diagonal elements that a number of parameters is highly related to each other. An even better view may be obtained by the so called correlation matrix R, where the matrix elements

$$R_{p_i,p_j} = \frac{\mathrm{cov}(p_i, p_j)}{\sqrt{\sigma_i^2 \sigma_j^2}} \tag{17}$$

The entries of the covariance matrix are here normalized by the variances of the parameters itself, i.e. by the diagonal elements.

[538]:
```python
s=np.diag(cov)
R=np.zeros([3,3])
for i in range(3):
    for j in range(3):
        R[i,j]=cov[i,j]/np.sqrt(s[i]*s[j])
```

[527]:
```python
print(R)
```

```
[[ 1.         -0.95608434  0.63731452]
 [-0.95608434  1.         -0.79422754]
 [ 0.63731452 -0.79422754  1.        ]]
```

The correlation matrix thus indeed reveals that the parameters are highly related to each other. `curve_fit` calculates the corresponding covariance entries using the specified uncertainties. We may access the meaning of them a bit better if we look at our synthetic data.

[539]:
```
def data(x,a,b,c):
    y=a*x**2+b*x+c
    err=[np.random.normal() for _ in range(len(x))]
    err=y*err*0.05
    return(y+err)
```

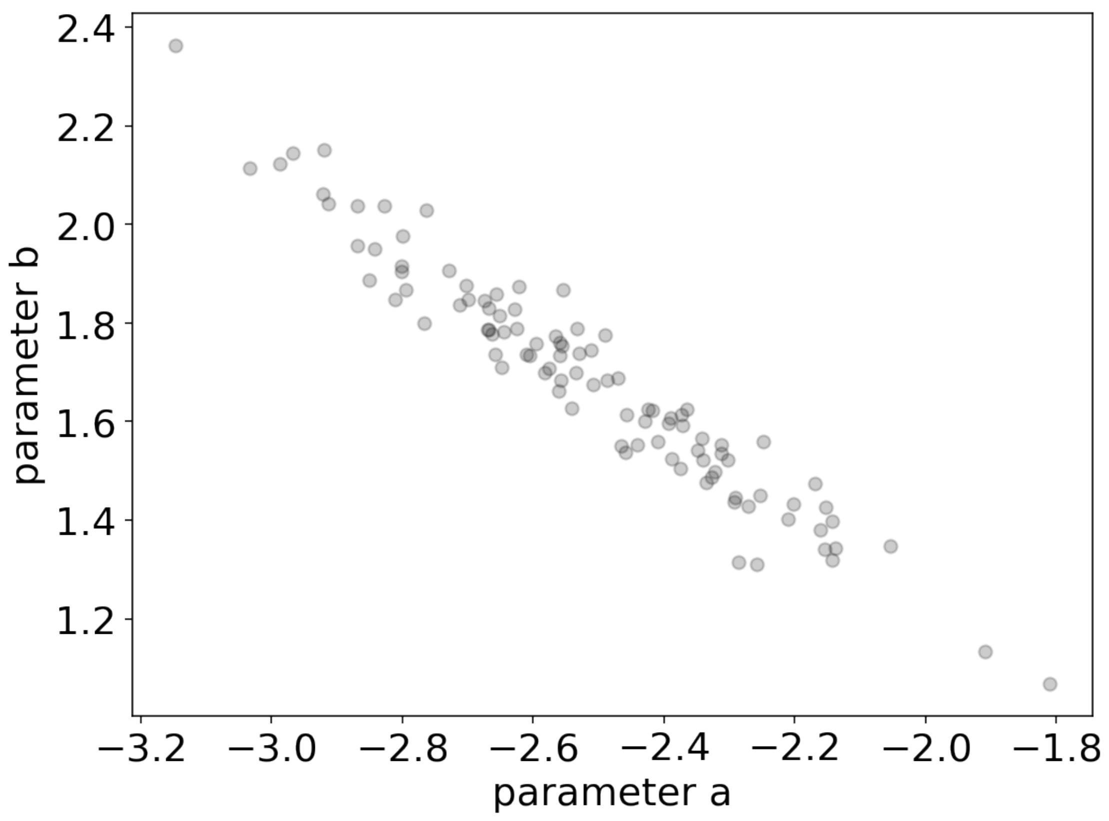
If we generate 100 different measurements and fit the accordingly, we can access a possible correlation of the parameters as sugested by the correlation matrix.

[540]:
```
a=[]
b=[]
c=[]
x=np.linspace(0,1,10)
for _ in range(100):
    y=data(x,-2.52,1.6971755,1)
    p,cov=curve_fit(parabola,x,y,sigma=err,p0=init_guess,absolute_sigma=True)
    a.append(p[0])
    b.append(p[1])
    c.append(p[2])
```

If we now plot the parameter a over the parameter b, we indeed obtain a very strong correlation, which has a negative slope as suggested by the correlation matrix.

[529]:
```
plt.figure(figsize=(8,6))
plt.scatter(a,b,color=' ',alpha=0.2)
plt.xlabel('parameter a')
plt.ylabel('parameter b')
```

[529]:
```
Text(0, 0.5, 'parameter b')
```



This correlation of parameters means, that the parameter `b` is not independent from `a` but rather strongly linearly dependent on it. We might want to find a better model containing more independent parameters. We may write down a different model

$$y = a(x-b)^2 + c \tag{18}$$

which also contains three parameters, but the parameter `b` directly refers to trhe maximum of out parabola, while the parameter `a` denotes its curvature.

[541]:
```
def newmodel(x,a,b,c):
    return(a*(x-b)**2+c)
```

[542]:
```
fit=curve_fit(newmodel,x_data,y_data,sigma=err,p0=init_guess,absolute_sigma=True)
```

[543]:
```
ans, cov=fit
```

[544]:
```
s=np.diag(cov)
R=np.zeros([3,3])
for i in range(3):
    for j in range(3):
        R[i,j]=cov[i,j]/np.sqrt(s[i]*s[j])
```

We see from the covariance matrix that the new model has a smaller correlation of the parameters on each other.

```
[546]: print(cov)
```

```
[[ 0.07675961 -0.00252389 -0.00524745]
 [-0.00252389  0.0002834   0.0001206 ]
 [-0.00524745  0.0001206   0.00074961]]
```

This is also expressed by our correlation matrix.

```
[495]: print(R)
```

```
[[ 1.         -0.54113427 -0.69177302]
 [-0.54113427  1.          0.26166299]
 [-0.69177302  0.26166299  1.        ]]
```

```
[ ]:
```