Statistical Methods

Aleksas Girenas

November 2020

1 Abstract

We will analyse parameter fitting and model testing with the method of least squares. This is done through a series of exercises that include fitting a polynomial and using the resulting covariance matrix of estimated parameters for error propagation, as well as analysis of historical data sets from Galileo and Ptolemy. Using measures of goodness-of-fit from the least squares fits, the level of agreement between the data and competing hypotheses is assessed.

2 Introduction

The method of least squares is a powerful tool for parameter fitting and model testing in the analysis of experimental data. The goal of the following exercises is to analyse the different methods used to find a smooth curve that passes close to the data points, i.e. ”fitting” a curve to the data.

Given the available data from an experiment, there is no way to definitively figure out the curve of $f(x)$; however we can estimate the curve using statistical methods. We must therefore hypothesise a formula for $f(x)$ that contains some adjustable parameters (constants whose values are not yet determined) such as a straight line.

$$f(x;\theta) = \theta_0 + \theta_1 x \tag{1}$$

Estimating the curve is then reduced to estimating these unknown parameters. After picking values for $\theta = (\theta_0, \theta_1)$ we could find how far our predicted function strays from the data by measuring the residual $y_i - f(x_i;\theta)$ [1]. In measured data we can not have perfect results and so we must also consider the error σ_i in our measurement; measuring the residual in units of σ_i gives us the normalised residual:

$$\frac{y_i - f(x_i;\theta)}{\sigma_i} \tag{2}$$

This characterises how far the measurement y_i is from the predicted value $f(x_i;\theta)$.

Obtaining small values from (2) would suggest the curve is a good fit, but these can be both positive or negative for various data points, so we can not simply minimise their sum to check that the whole curve is a good fit. We must therefore use the method of Least Squares [1] to minimise our residuals giving the quantity chi-squared:

$$X^2(\theta) = \sum_{i=1}^{N} (\frac{y_i - f(x_i;\theta)}{\sigma_i})^2 \tag{3}$$

Parameters that give the minimum chi-squared are called the *estimators* of the parameters and are written with hats (e.g. $\hat{\theta}_0$, $\hat{\theta}_1$). This allows us to produce a curve that fits well to the data if the initial predicted function is accurate.

Finding the derivative of chi-squared to find the minimum becomes difficult and impractical for anything more than a linear function with two parameters (1). Therefore finding the *estimators* is most easily done numerically, and we can use **Python** to demonstrate this.

We will not discuss the workings of the algorithms [2][3] used to minimise chi-squared (3), however a rough idea is useful: simplified the algorithms evaluate the objective function $X^2(\theta)$ at different points in the parameter space and so figure out the minimum of chi-squared. So to use these algorithms we must start at a certain point in the parameter space θ_{start}. Sometimes default values can be used for this but in other problems a rough preliminary estimate is required.

We can also understand that the estimators $\hat{\theta}$ have statistical errors as repeating an experiment with new, statistically independent measurements y_i would give fluctuations in the data when compared to previous measurements. These uncertainties can be found with the Monte Carlo Method or, more easily, using derivatives of the function $X^2(\theta)$. These methods give a covariance matrix U and by taking the square root of the diagonal elements we can find the standard deviation $\sigma_{\hat{\theta}_i}$ and, combining with the covariance, the correlation coefficient ρ_i [1]. Also we can ”propagate” the covariance of $\hat{\theta}_i$ through to a function u of the estimators to characterise the statistical uncertainty in the function.

In our advances in finding a function that fits the data well we must also consider the pdf of the minimised value X^2_{min}. To simplify the notation $X^2_{min} = z$ and so the chi-squared pdf is given by:

$$f_{x^2}(z;n) = \frac{1}{2^{n/2}\Gamma(n/2)} z^{n/2-1} e^{-z/2} \tag{4}$$

A low value of (6) suggests that the function may be a good fit to the data; however it does not prove that the hypothesis is correct as there are random fluctuations in the data, and so X^2_{min} also exhibits these fluctuations.

We must therefore also find the probability to quantify the goodness of fit under the assumption the hypothesis is correct. This is called the p-value and can be found by integrating the chi-squared pdf (6) from the value of X^2_{min} observed to infinity:

$$p = \int_{X^2_{min}}^{\infty} f_{x^2}(z;n)dz \tag{5}$$

If we find a very low p-value, or X^2_{min} substantially greater than the number of degrees of freedom, then we may want to try another hypothesis.

3 Exercises

In the following exercises we will evaluate the use of the least squares method and the different methods that can be used to test the fit of a function in **Python**. The core structure of the code used is given in the appendix

3.1 Polynomial fit and error analysis

Considering the data points of (x, y, σ):
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
σ = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])

First we define a linear test function as seen in (1) and do a least-squares fit:

```
def func_1(x, *theta):
    theta0, theta1 = theta
    return theta0 + theta1*x

p0 = np.array([1.0, 1.0])
thetaHat, cov = curve_fit(func_1, x, y, p0, sig,
    ↪ absolute_sigma = True)
standDev = np.sqrt(np.diag(cov))
```

And we can find the error propagation from:

```
fit_standDev = np.sqrt(cov[0][0] + cov[0][1]*xfit +
    ↪ cov[1][0]*xfit + cov[1][1]*xfit**2)
```

Using this method for test functions of an Mth order polynomial $M = 1, 2, 3$ the plots and error propagation beyond the data are shown:

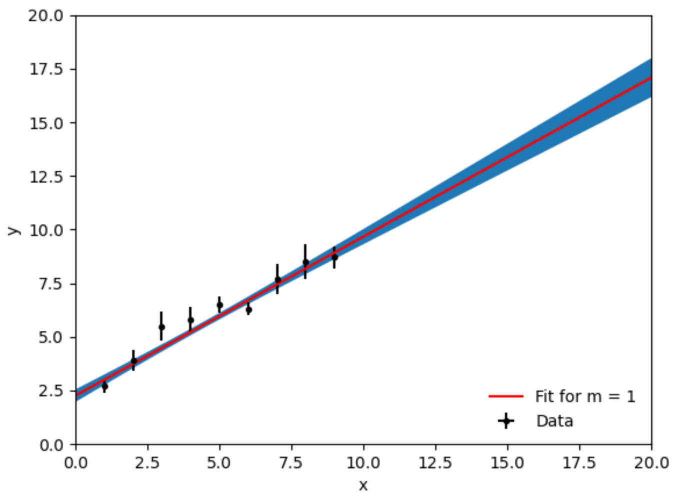

Figure 1: Test fit and error propagation for M=1

As we can see since in figure 1 the test function is linear the error propagation does not deviate significantly from the fit and the deviation increases linearly. The test function is inaccurate, however, and we can clearly see that much of the data does not fall within the deviation from the function.

The fitted values for this function were:
θ_0 = 2.2576981889195182 $\pm$ 0.29218909382046193
θ_1 = 0.7409333605720615 $\pm$ 0.05723132195270343

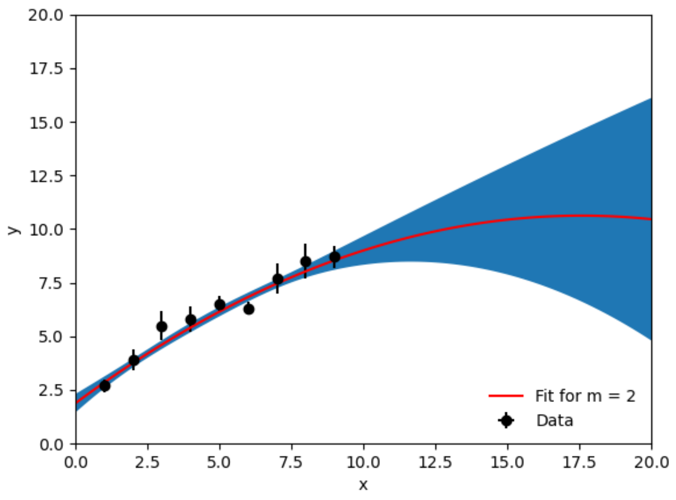

Figure 2: Test fit and error propagation for M=2

When we increase the Mth order to 2 we can see that the error propagation is very wide as soon as the curve leaves the scope of the data and continues to increase non-linearly as we move further from the data. This test function is also inaccurate within the scope of the data and much of the data does not fall within the deviation.

The fitted values for this function were:
θ_0 = 1.8818578762757632 $\pm$ 0.4308098548388211
θ_1 = 0.994562730779869 $\pm$ 0.2211713231357369
θ_2 = -0.028289746539512137 $\pm$ 0.023829155540998573

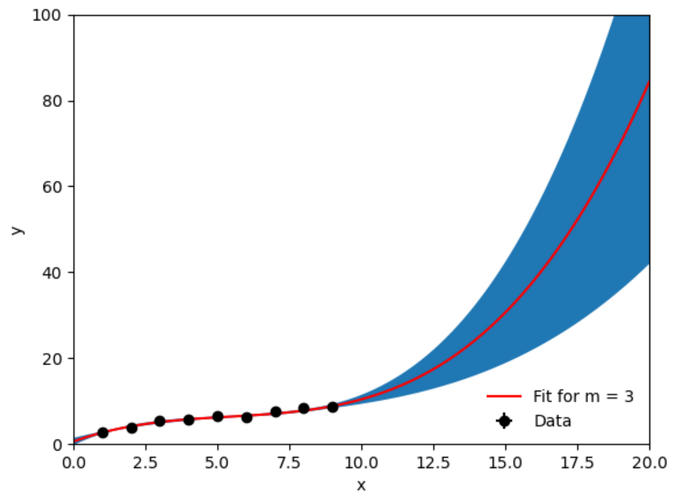

Figure 3: Test fit and error propagation for M=3

At M=3 order we find that the error propagation beyond the data follows the fit much more smoothly although it still becomes wider as me move further away from the data. Much of the data also falls within the deviation. This function of order M=3 seems to fit this set of data best.

The fitted values for this function were:
θ_0 = 0.5984830029658463 ± 0.8472740174575287
θ_1 = 2.4332610973217625 ± 0.8472481912973759
θ_2 = -0.3779205848038629 ± 0.20018139357568554
θ_3 = 0.02327545468970047 ± 0.013231624481314933

Considering the fit with M=3 and defining the difference:

$$\Delta_{ab}(\hat{\theta}) = f(a;\hat{\theta}) - f(b;\hat{\theta}) \tag{6}$$

we can compare these values with the standard deviation of the function that we plotted for this fit as a shaded band evaluated at both a and b.

The standard deviation of delta, for a = 5 and b = 6 is: 0.18069306921085257 The difference in the standard deviation of the fitted function at a = 5 and b = 6 is: 0.3262150800786794

The standard deviation of delta, for a = 5 and b = 10 is: 1.0847585761508876 The difference in the standard deviation of the fitted function at a = 5 and b = 10 is: 1.0678207462331941

The standard deviation of delta, for a = 5 and b = 20 is: 42.331410472464704 The difference in the standard deviation of the fitted function at a = 5 and b = 20 is: 42.35654499298686

As can be observed both the values obtained by using equation (6) and the deviations seen from the function plot agree.

3.2 Analysis of Galileo's ball and ramp data

Here we consider an experiment performed by Galileo using a ball and an inclined ramp as shown in figure 4. For this exercise we also do not use Newton's laws.

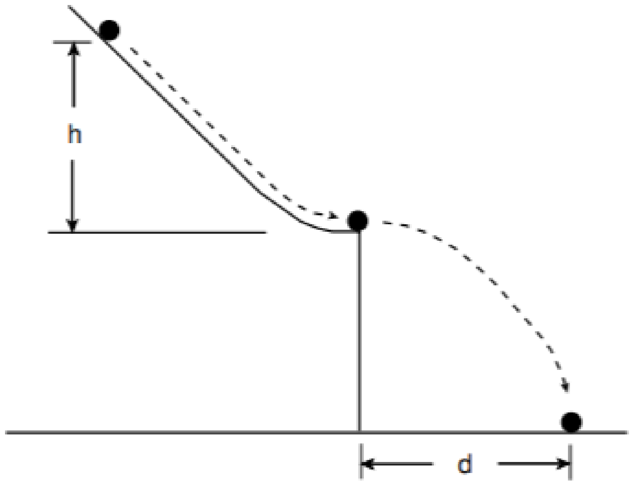

Figure 4: Galileo's ball and ramp experiment

The horizontal distance d from the edge to the point of impact is measured for different values of h. Five data points are shown in units of punti (1punto approx. 1mm):

h	d
1000	1500
828	1340
800	1328
600	1172
300	800

We also know logically that at height $h = 0$ the ball will fall straight down and so the distance d would also be zero; we can also re-use the code from the previous exercise in order to find and plot our test functions. Assuming negligible error in the heights h and that the horizontal distances d have uncertainties of $\sigma = 15$ punti we can hypothesise the relationship between d and h.

First let's test a linear relationship in the form $d = \alpha h$ where α is a single free parameter.

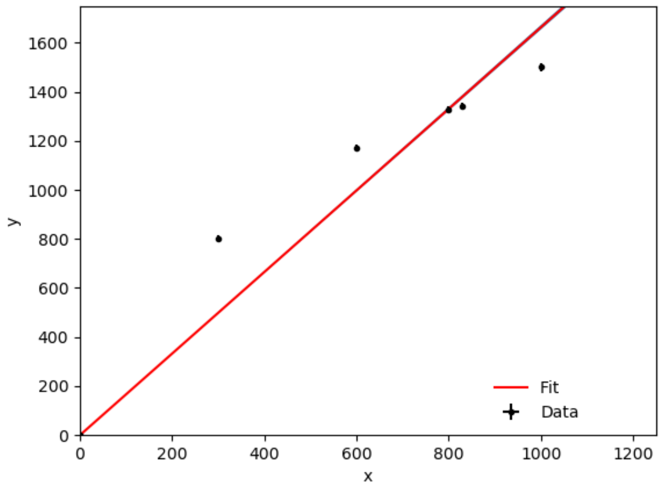

Figure 5: Linear function test in the form $d = \alpha h$ where α is a single free parameter

The fitted value of α is:
α = 1.6627562234481246 ± 0.00900356505896255

From the plotted results we can see that a linear test function does not seem to follow the data points well. This is further emphasised when we look at the chi-squared value and the p-value: the chi-squared value is: 661.9899966389389, which is huge; the p-value is: 5.9133043638181304e-142, which is small. Clearly this hypothesis does not fit well.

Testing a quadratic relationship in the form $d = \alpha h + \beta h^2$ where α and β are free parameters we find:

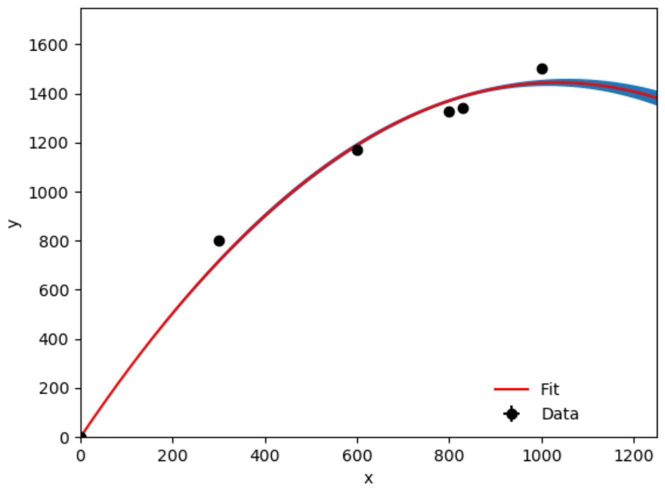

Figure 6: Quadratic function test in the form $d = \alpha h + \beta h^2$ where α and β are free parameters

The fitted values were:
α = 2.792907667005891 $\pm$ 0.04711327556259347
β = -0.0013505297485882917 $\pm$ 5.52627457602966e-05

In this case we find that the test curve seems to follow the data a lot more closely and this is supported by the chi-squared value being smaller: 64.74181304025628; and the the p-value is: 5.696204646233935e-14, which is bigger and so the curve is a better fit. This does not mean that it is a good hypothesis, however, as the p-value is still very small so the chi-squared value may not be likely.

A nonlinear function of the parameters $d = \alpha h^\beta$ gives the results:

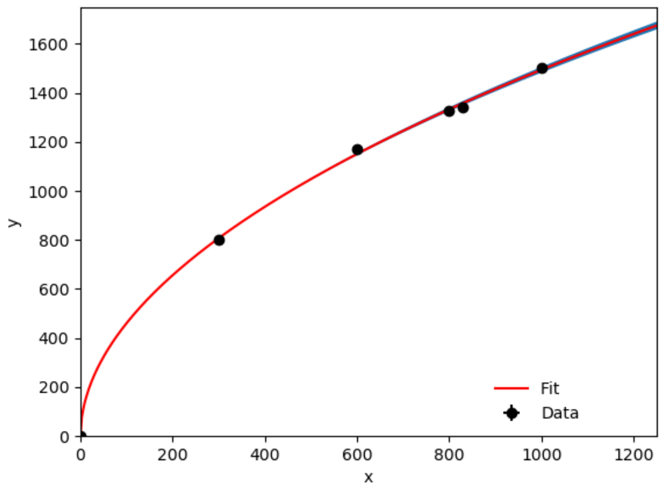

Figure 7: Nonlinear function test in the form $d = \alpha h^\beta$ where α and β are free parameters

The fitted values were:
α = 43.76058967981527 $\pm$ 4.800415663643349
β = 0.5110560146164763 $\pm$ 0.01653592654498319

The test function $d = \alpha h^\beta$ where α and β are free parameters fits the data of Galileo's ball and ramp experiment the best: the chi-squared value is: 3.755928460174434, which is smal and much closer to 0; the p-value is: 0.44004378289578483 which suggests the chi-squared value is probable and this function is a much closer fit to the data.

3.3 Analysis of refraction data from Ptolemy

Here we will use and analyse test functions for the data that Ptolemy recorded to find a relationship between the angle of incidence and the angle of refraction, assuming we do not know Snell's law.

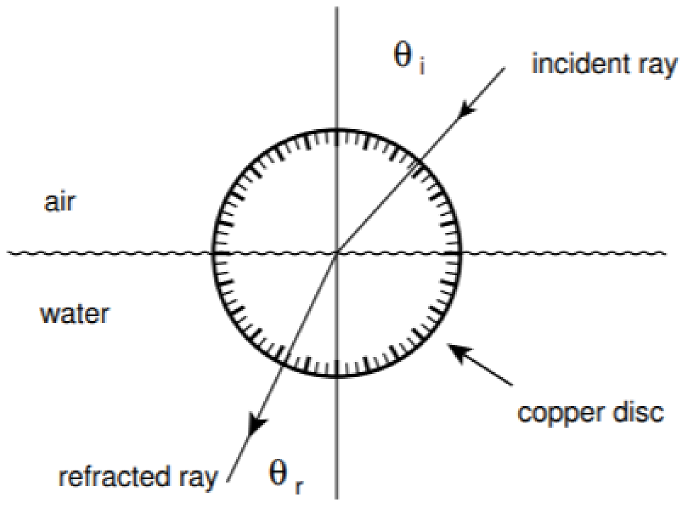

Figure 8: The apparatus used by Ptolemy to investigate the refraction of light

Angles of refraction θ_r for 8 values of the angle of incidence θ_i obtained by Ptolemy are shown in the following table:

θ_i	θ_r
10	8
20	15.5
30	22.5
40	29
50	35
60	40.5
70	45.5
80	50

For purposes of this exercise we will take the angles of incidence to be known with negligible error and treat the angles of refraction as independent Gaussian-distributed measurements with standard deviations of $\sigma = \frac{1}{2}^o$ which is reasonable given that the angles are reported to the nearest half degree.

Before the discovery of the correct law of refraction a commonly used hypothesis was $\theta_r = \alpha\theta_i$.

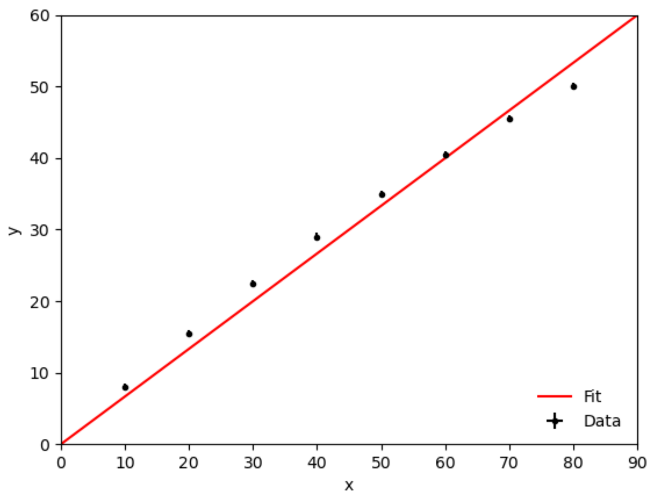

Figure 9: Test function $\theta_r = \alpha\theta_i$

The fitted value is:
$\alpha = 0.6661764705875071 \pm 0.003500700204239394$

We can observe that the function does not follow the data well and numerically: the chi-squared value is: 134.6470588235294, which is very large; the p-value is: 6.711003202955456e-26, which is very small. The curve does not fit the data well.

Ptolemy preferred to use a different function $\theta_r = \alpha\theta_i - \beta\theta_i^2$ which is plotted here:

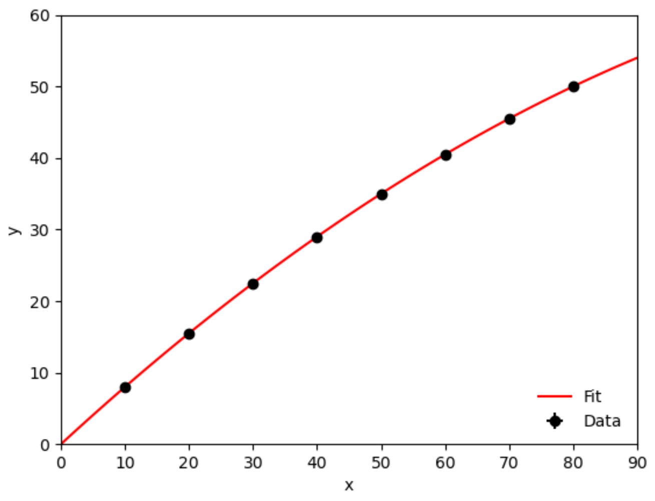

Figure 10: Test function $\theta_r = \alpha\theta_i - \beta\theta_i^2$

The fitted values are:
$\alpha = 0.825 \pm 0.014127841073706914$
$\beta = 0.0024999999999999996 \pm 0.0002154475563752497$

The hypothesis is remarkably accurate and follows the data surprisingly well. In fact numerically the chi-squared value is found to be: 0.0 and the p-value is: 1.0. The hypothesis fits the data perfectly. This, however, cannot be a true hypothesis: even if this was the true law of refraction measurement of data always brings uncertainty and the values would fluctuate so chi-squared cannot be perfectly 0.0 with absolute probability. Clearly it is not plausible that all the data values are based on actual experimental data.

The law of refraction is defined as:

$$\theta_r = \arcsin\frac{\sin\theta_i}{r} \tag{7}$$

where $r = n_r/n_i$ is the ratio of indices of refraction of the two media. We can determine the least squares estimate for r:

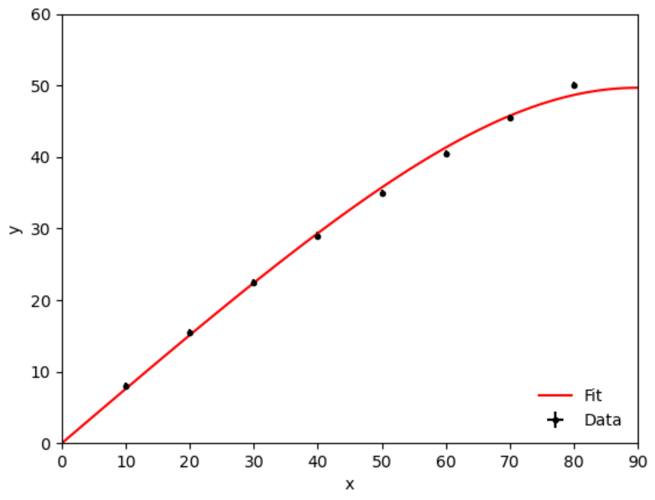

Figure 11: Test function $\theta_r = \arcsin\frac{\sin\theta_i}{r}$

The value of r is:
r = 1.3116118503277543 ± 0.005575466589663376

Plotting the actual function to fit the data we find that the chi-squared value is: 14.002173915990351, which is fairly small but the p-value is: 0.05114270171746497 which shows a low probability that the chi-squared value is true. As we can see in figure 11 initially the curve fits the data very well and then falls away.

The Gaussian assumption for θ_r with $\sigma = \frac{1}{2}^o$ is a reasonable guess given that the angles are reported to the nearest half degree. We can absorb an error in θ_i into an effective error in θ_r.

4 Conclusion

As demonstrated the method of least squares is an important tool for parameter fitting and model testing in the analysis of experimental data.

By using the method of Least Squares [1] to minimise our residuals we can obtain the quantity chi-squared and the parameters that give the minimum chi-squared are called the *estimators* of the parameters and are written with hats (e.g. $\hat{\theta}_0$, $\hat{\theta}_1$). This allows us to produce a curve that fits well to the data if the initial predicted function is accurate.

An simple understanding of the algorithms used to evaluate the objective function $X^2(\theta)$ at different points in the parameter space is paramount to the evaluation of a good test function and to figure out the minimum of chi-squared. We must start at a certain point in the parameter space θ_{start} to use these algorithms. Sometimes default values can be used for this but in other problems a rough preliminary estimate is required.

We also found that the estimators $\hat{\theta}$ have statistical errors as repeating an experiment with new, statistically independent measurements y_i would give fluctuations in the data when compared to previous measurements. These uncertainties can be found with the Monte Carlo Method or, more easily, using derivatives of the function $X^2(\theta)$. Also we can "propagate" the covariance of $\hat{\theta}_i$ through to a function u of the estimators to characterise the statistical uncertainty in the function.

In our advances in finding a function that fits the data well we also considered the pdf of the minimised value X^2_{min}. A low value of (6) suggests that the function may be a good fit to the data; however it does not prove that the hypothesis is correct as there are random fluctuations in the data, and so X^2_{min} also exhibits these fluctuations.

We therefore also found the probability to quantify the goodness of fit under the assumption the hypothesis is correct. This is called the p-value and can be found by integrating the chi-squared pdf (6) from the value of X^2_{min} observed to infinity. If we find a very low p-value, or X^2_{min} substantially greater than the number of degrees of freedom, then we should try another hypothesis.

These methods were tested in exercises and we found hypothetical functions that fit arbitrary and past data well. Clearly these methods are vital to the analysis of experimental data and the testing of models.

References

[1] G. Cowan - Introduction to Statistical Methods PH3010 Advanced Skills. Sept 2020. RHUL Physics 2020/21

[2] Vladimir Volkov - Basic numerical and statistical methods http://www.embl-hamburg.de/workshops/2001/EMBO/presentations/volkov_numer.pdf

[3] Jorge Nocedal Stephen J. Wright - Numerical Optimization http://www.apmath.spbu.ru/cnsa/pdf/monograf/Numerical_Optimization2006.pdf

Appendices

Python script used (modified for each question).

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.stats import chi2
from math import asin, sin, pi, radians


x = np.array([1000, 828, 800, 600, 300, 0])
y = np.array([1500, 1340, 1328, 1172, 800, 0])
sig = np.array([15, 15, 15, 15, 15, 0.01])
xfit = np.linspace(0, 3000, 1000)

def func_1(x, *theta):
    theta0, theta1 = theta
    return theta0 + theta1*x

p0 = np.array([1.0, 1.0])
thetaHat, cov = curve_fit(func_1, x, y, p0, sig,
    ↪ absolute_sigma = True)
standDev = np.sqrt(np.diag(cov))

print("The␣fitted␣values␣are:")
for order in range(len(thetaHat)):
    print("theta", order, "=", thetaHat[order], "+-",
        ↪ standDev[order])
print("\n")

fit_standDev = np.sqrt(cov[0][0] + cov[0][1]*xfit +
    ↪ cov[1][0]*xfit
            + cov[1][1]*xfit**2)

fig, ax = plt.subplots(1,1)
plt.errorbar(x, y, yerr = sig, xerr = 0, color='black
    ↪ ', fmt='.', label='Data')
fit1 = func_1(xfit, *thetaHat)
plt.plot(xfit, fit1, 'red', label='Fit␣for␣m␣=␣1')
plt.fill_between(xfit, fit1 + fit_standDev, fit1 -
    ↪ fit_standDev)
plt.xlim(0, 1250)
plt.xlabel("x")
plt.ylim(0, 1750)
plt.ylabel("y")

plt.legend(loc = "lower␣right", fontsize = 10,
    ↪ frameon = False)
plt.show()


def func_2(x, *theta):
    theta0, theta1, theta2 = theta
    return theta0 + theta1*x + theta2*x**2

p0 = np.array([1.0, 1.0, 1.0])
thetaHat, cov = curve_fit(func_2, x, y, p0, sig,
    ↪ absolute_sigma = True)
standDev = np.sqrt(np.diag(cov))

print("The␣fitted␣values␣are:")
for order in range(len(thetaHat)):
    print("theta", order, "=", thetaHat[order], "+-",
        ↪ standDev[order])
print("\n")

fit_standDev = np.sqrt(cov[0][0] + cov[0][1]*xfit +
    ↪ cov[1][0]*xfit
            + cov[1][1]*xfit**2
            + cov[0][2]*xfit**2 + cov[2][0]*xfit
                ↪ **2 + cov[2][2]*xfit**4
            + cov[1][2]*xfit**3 + cov[2][1]*xfit
                ↪ **3)

fig, ax = plt.subplots(1,1)
plt.errorbar(x, y, yerr = sig, xerr = 0, color = '
    ↪ black', fmt = 'o',
            label = 'Data')
fit2 = func_2(xfit, *thetaHat)
plt.plot(xfit, fit2, color = 'red', label = 'Fit␣for␣
    ↪ m␣=␣2')
plt.fill_between(xfit, fit2 + fit_standDev, fit2 -
    ↪ fit_standDev)
plt.xlim(0, 1250)
plt.xlabel("x")
plt.ylim(0, 1750)
plt.ylabel("y")

plt.legend(loc = "lower␣right", fontsize = 10,
    ↪ frameon = False)
plt.show()


def func_3(x, *theta):
    theta0, theta1 = theta
    return theta0*x**(theta1)

p0 = np.array([1.0, 1.0])
thetaHat, cov = curve_fit(func_3, x, y, p0, sig,
    ↪ absolute_sigma = True)
standDev = np.sqrt(np.diag(cov))

print("The␣fitted␣values␣are:")
for order in range(len(thetaHat)):
    print("theta", order, "=", thetaHat[order], "+-",
        ↪ standDev[order])
print("\n")

def fit_standDev(xfit, cov):

    fit_standDev = np.sqrt(cov[0][0] + cov[0][1]*xfit
        ↪ + cov[1][0]*xfit
                + cov[1][1]*xfit**2)
    return fit_standDev

fig, ax = plt.subplots(1,1)
plt.errorbar(x, y, yerr = sig, xerr = 0, color = '
    ↪ black', fmt = 'o',
            label = 'Data')
fit3 = func_3(xfit, *thetaHat)
plt.plot(xfit, fit3, color = 'red', label = 'Fit␣for␣
    ↪ m␣=␣3')
plt.fill_between(xfit, fit3 + fit_standDev(xfit, cov)
    ↪ , fit3 - fit_standDev(xfit, cov))
plt.xlim(0, 1250)
plt.xlabel("x")
plt.ylim(0, 1750)
plt.ylabel("y")

plt.legend(loc = "lower␣right", fontsize = 10,
    ↪ frameon = False)
plt.show()


x = np.array([1000, 828, 800, 600, 300, 0])
y = np.array([1500, 1340, 1328, 1172, 800, 0])
sig = np.array([15, 15, 15, 15, 15, 0.01])
xfit = np.linspace(0, 3000, 1000)
```

```
p0 = np.array([1.0, 1.0])
func = func_3
graph = 3

thetaHat, cov = curve_fit(func, x, y, p0, sig,
    ↪ absolute_sigma=True)
sigma = np.sqrt(np.diag(cov))
chisq = sum(((y-func(x, *thetaHat))/sig)**2)
n = len(x)-len(p0)
p = chi2.sf(chisq, n)

fig, ax = plt.subplots(1,1)
plt.errorbar(x, y, yerr=sig, color='black', fmt='x',
    ↪ label='Data')
plt.plot(xfit, func(xfit, *thetaHat), color='red',
    ↪ label='Fit')
plt.xlim(0, 20)
plt.xlabel('x')
plt.ylim(0, 100)
plt.ylabel('y')
plt.legend(loc='lower␣right', fontsize=10, frameon=
    ↪ False)
plt.show()

if graph == 1:
    print('alpha␣=', thetaHat[0], '+-', sigma[0])
elif graph == 2:
    print('alpha␣=', thetaHat[0], '+-', sigma[0])
    print('beta␣=', thetaHat[1], '+-', sigma[1])
else:
    print('r␣=', thetaHat[0], '+-', sigma[0])
print("\ni", '␣', 'j', '␣', 'cov[i,j]', '␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ ␣␣␣', 'rho[i,j]:')
for x in range(len(thetaHat)):
    for y in range(len(thetaHat)):
        rho = cov[x][y]/(sigma[x]*sigma[y])
        print(x, "␣", y, "␣", cov[x][y], "␣", rho)
print('\nThe␣chi-squared␣value␣is:', chisq)
print('The␣p-value␣is:', p)


standDev_delta56 = np.sqrt(cov[1][1] + cov[1][2]*11 +
    ↪  cov[2][1]*11 + cov[1][3]*91
                    + cov[3][1]*91 + cov[2][2]*121 +
                        ↪ cov[2][3]*1001
                    + cov[3][2]*1001 + cov[3][3]*8281)

standDev_delta510 = np.sqrt(cov[1][1]*25 + cov
    ↪ [1][2]*375 + cov[2][1]*375
                     + cov[1][3]*4375 + cov[3][1]*4375
                         ↪  + cov[2][2]*5625
                     + cov[2][3]*65625+ cov
                         ↪ [3][2]*65625 + cov
                         ↪ [3][3]*765625)

standDev_delta520 = np.sqrt(cov[1][1]*225 + cov
    ↪ [1][2]*5625 + cov[2][1]*5625
                     + cov[1][3]*118125 + cov
                         ↪ [3][1]*118125 + cov
                         ↪ [2][2]*140625
                     + cov[2][3]*2953125 + cov
                         ↪ [3][2]*2953125
                     + cov[3][3]*62015625)
```

```
print("The␣standard␣deviation␣of␣delta,␣for␣a␣=␣5␣and
    ↪ ␣b␣=␣6␣is:", standDev_delta56)
print("The␣difference␣in␣the␣standard␣deviation␣of␣
    ↪ the␣fitted␣function␣at␣\
a␣=␣5␣and␣b␣=␣6␣is:", np.sqrt(fit_standDev(5, cov)**2
    ↪  + fit_standDev(6, cov)**2), "\n")

print("The␣standard␣deviation␣of␣delta,␣for␣a␣=␣5␣and
    ↪ ␣b␣=␣10␣is:", standDev_delta510)
print("The␣difference␣in␣the␣standard␣deviation␣of␣
    ↪ the␣fitted␣function␣at␣\
a␣=␣5␣and␣b␣=␣10␣is:", np.sqrt(fit_standDev(5, cov)
    ↪ **2 + fit_standDev(10, cov)**2), "\n")

print("The␣standard␣deviation␣of␣delta,␣for␣a␣=␣5␣and
    ↪ ␣b␣=␣20␣is:", standDev_delta520)
print("The␣difference␣in␣the␣standard␣deviation␣of␣
    ↪ the␣fitted␣function␣at␣\
a␣=␣5␣and␣b␣=␣20␣is:", np.sqrt(fit_standDev(5, cov)
    ↪ **2 + fit_standDev(20, cov)**2), "\n")
```