PH2255 Course:
Introduction to Statistical Methods
Exercise 2

Thomas Bass

22 January 2021

Abstract

Exercise Two in the PH2255 course introduces us to methods to quantify the "goodness-of-fit" of our curve fits generated last week using the method of minimising the sum of the squared residuals. By calculating the χ^2_{min} value for the parameters, we can use its ratio per the number of degrees of freedom to calculate the *p-value* for the fit, quantifying it's "goodness-of-fit".

For this exercise, we consider an historical experiment carried out by Galileo. The experiment involves, as shown in Figure 1, rolling a ball down an inclined plane, where at the end of the ramp the trajectory is completely horizontal. The ball then rolls off the edge, and falls H under projectile motion until it hits the ground, some horizontal distance d away from the edge of the ramp. Galileo used the units *"punti"* to record his data: we assume one *punto* is around 1mm.

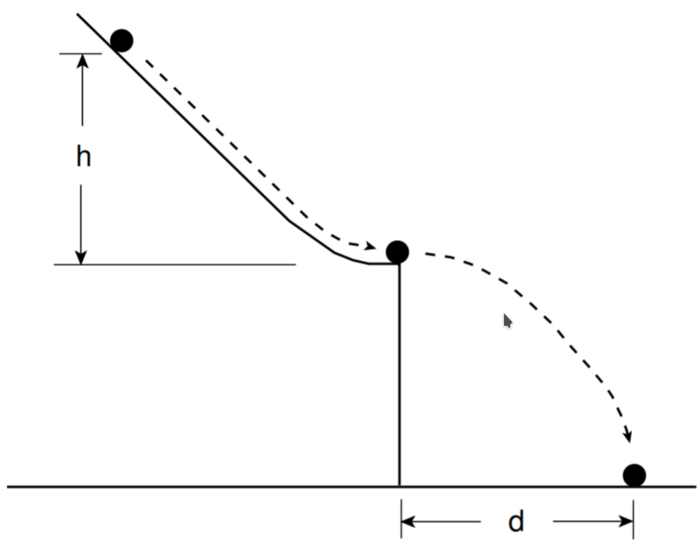

Figure 1: The set-up that Galileo used for the ball and ramp experiment.

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

Table 1: Data from Galileo's journal showing the horizontal distance d of the ball before impact for five values of h

The data, taken from Galileo's journals, can be found in Table 2. For this exercise, we assume an uncertainty of $\sigma = 15$punti for all values of d, and a negligible uncertainty for h.

We can also include the data point of $(0, 0)$, as from a basic analysis we can conclude that if the ball fell from no height, it would have no velocity, and it would fall straight down to $d = 0$.

The exercise instructs us to use two hypotheses for potential functional relationships between d and h:

$$d = \alpha h \tag{1}$$

$$d = \alpha h + \beta h^2 \tag{2}$$

$$d = \alpha h^\beta \tag{3}$$

Each of these equations, as well as the data used, was imported into Python with the following code:

```
h   = np.array([1000., 828.,  800.,  600.,  300.])
d   = np.array([1500., 1340., 1328., 1172., 800.])
sig = np.array([15.,   15.,   15.,   15.,   15.])

# First hypothesis
def f1(h, *vars):
    a, = vars
    return a*h

# Second hypothesis
def f2(h, *vars):
    a, b = vars
    return a*h + b*h**2

# Third hypothesis
def f3(h, *vars):
    a, b = vars
    return a*h**b
```

For each of these hypotheses, we then carry out the least-squares fit from the previous exercise, using SciPy's curve_fit method, to generate the fitted parameters and their covariance matrices.

```
# Curve fitting
p0_1 = np.array([1.0])
theta_hat_1, covariance_1 = curve_fit(f1, h, d, p0_1, sig,
    absolute_sigma=True)

p0_2 = np.array([1.0, 1.0])
theta_hat_2, covariance_2 = curve_fit(f2, h, d, p0_2, sig,
    absolute_sigma=True)

p0_3 = np.array([1.0, 1.0])
theta_hat_3, covariance_3 = curve_fit(f3, h, d, p0_3, sig,
    absolute_sigma=True)

# Standard deviation of parameters
theta_hat_1_std_dev = np.sqrt(np.diag(covariance_1))

theta_hat_2_std_dev = np.sqrt(np.diag(covariance_2))

theta_hat_3_std_dev = np.sqrt(np.diag(covariance_3))
```

For each of these curve fits, we can produce plots:

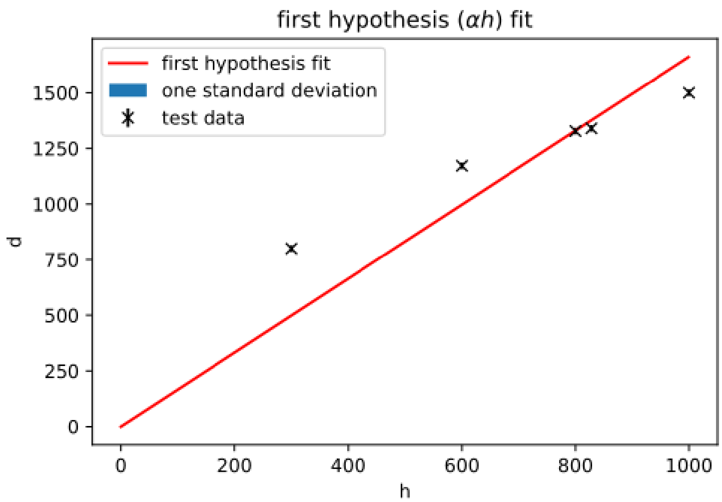

Figure 2: First hypothesis fit, showing fit curve and one standard deviation above and below

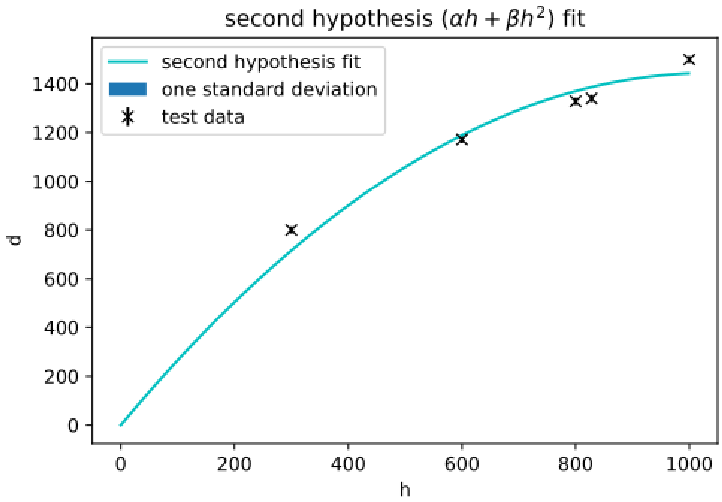

Figure 3: Second hypothesis fit, showing fit curve and one standard deviation above and below

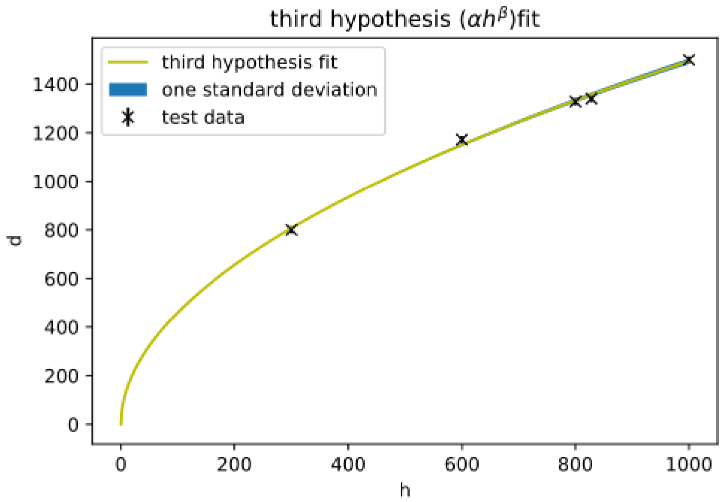

Figure 4: Third hypothesis fit, showing fit curve and one standard deviation above and below

Then, for each fit parameters, we can calculate the χ^2_{min} value. The equation, given as Equation 28 in the course handbook is:

$$\chi^2_{\text{min}} = \sum_{i=1}^{N} \frac{(y_i - f(x_i; \vec{\hat{\theta}}))^2}{\sigma_i^2} \tag{4}$$

Converted into Python, as well as the calculation for the Degrees of Freedom, we get the following functions:

Hypothesis	χ^2_{min}	*p-value*
$d = \alpha h$	165.50	5.91×10^{-142}
$d = \alpha h + \beta h^2$	21.58	5.70×10^{-14}
$d = \alpha h^{\beta}$	1.25	0.29

Table 2: The χ^2_{min} values and *p-values* obtained from each Hypothesis fit

```
chi_sq_min = sum(((d - f1(h, *theta_hat))/sig)**2)
ndof = len(h)-len(p0)
```

SciPy's statistics module then provides us with a function to calculate the *p-value* from a given χ^2_{min} and number of degrees of freedom:

```
p_val = scipy.stats.chi2.sf(chi_sq_min, df=ndof)
```

Using these functions on the calculated parameters from each fit, we get the following results:

From these values, we can clearly conclude that the third hypothesis is the most accurate - the χ^2_{min} value is the closest to 1, indicating that each value in the summation shown in Equation **4** is, on average, close to 1, and therefore close to the actual provided data.

To verify this, we can use Newton's laws of motion to derive an actual equation relating d to h, in terms of h, and the height of the edge of the ramp above the ground H. Using Conservation of Energy, we know that the total energy at the top of the ramp is equal to the total energy at the bottom of the ramp. The top of the ramp has Gravitational potential energy, and the bottom has Translational (kinetic) energy and rotational energy:

$$E_{\text{gravitational}} = E_{\text{translation}} + E_{\text{rotation}} \tag{5}$$

$$mgh = \frac{1}{2}mv^2 + \frac{1}{2}I\omega^2 \tag{6}$$

Using the moment of inertia for a sphere $I = \frac{2}{5}mr^2$:

$$mgh = \frac{1}{2}mv^2 + \frac{1}{2}kmv^2 \tag{7}$$

Then, using projectile motion from the bottom of the ramp $d = vt$

$$mgh = \frac{1}{2}m\left(\frac{x}{t}\right)^2 + \frac{1}{2}\frac{2}{5}m\left(\frac{x}{t}\right)^2 \tag{8}$$

$$\frac{x^2}{t^2} = 10gh/7 \tag{9}$$

Then, using time of flight for the vertical drop under only the acceleration of gravity $H = \frac{1}{2}gt^2$

$$d^2 = \left(\frac{10gh}{7}\right)\left(\frac{H}{\frac{1}{2}g}\right) \tag{10}$$

$$d = (10Hh/3.5)^{\frac{1}{2}} \tag{11}$$

If we set $\alpha = 10H/3.5$ we get the equation $x = \alpha h^{\frac{1}{2}}$.

If we extract the exact values of α and β from the $\vec{\hat{\theta}}$ values calculated in the curve fitting, we find $\alpha = 43.76059028$ and $\beta = 0.51105601$, giving us the empirical fitted equation:

$$d = 43.76059028 h^{0.51105601} \tag{12}$$

The calculated value of β in Equation **12** is sufficiently close to the predicted value of 0.5 in Equation **11**, so we can conclude that the derived formula fits the actual data fitted by Hypothesis 3.

A Python Code

```
import numpy as np
from scipy.optimize import curve_fit
from scipy.stats import chi2
import matplotlib
import matplotlib.pyplot as plt

h   = np.array([1000., 828.,  800.,  600.,  300.])
d   = np.array([1500., 1340., 1328., 1172., 800.])
sig = np.array([15.,   15.,   15.,   15.,   15.])

# First hypothesis
def f1(h, *vars):
    a, = vars
    return a*h

## Curve fit
p0_1 = np.array([1.0])
theta_hat_1, covariance_1 = curve_fit(f1, h, d, p0_1, sig,
    absolute_sigma=True)

## Standard deviation of parameters
theta_hat_1_std_dev = np.sqrt(np.diag(covariance_1))


# Second hypothesis
def f2(h, *vars):
    a, b = vars
    return a*h + b*h**2

## Curve fit
p0_2 = np.array([1.0, 1.0])
theta_hat_2, covariance_2 = curve_fit(f2, h, d, p0_2, sig,
    absolute_sigma=True)

## Standard deviation of parameters
theta_hat_2_std_dev = np.sqrt(np.diag(covariance_2))

# Third hypothesis
def f3(h, *vars):
    a, b = vars
    return a*h**b

## Curve fit
```

```
p0_3 = np.array([1.0, 1.0])
theta_hat_3, covariance_3 = curve_fit(f3, h, d, p0_3, sig,
    absolute_sigma=True)

## Standard deviation of parameters
theta_hat_3_std_dev = np.sqrt(np.diag(covariance_3))

# Plot graphs

## Setup Plots
fig1 = plt.figure()
fig2 = plt.figure()
fig3 = plt.figure()
ax1 = fig1.add_subplot()
ax1.set_title('first hypothesis ($\\alpha h$) fit')
ax2 = fig2.add_subplot()
ax2.set_title('second hypothesis ($\\alpha h+\\beta h^2$) fit')
ax3 = fig3.add_subplot()
ax3.set_title('third hypothesis ($\\alpha h^\\beta$)fit')
ax1.set_ylabel("d")
ax2.set_ylabel("d")
ax3.set_ylabel("d")
ax1.set_xlabel("h")
ax2.set_xlabel("h")
ax3.set_xlabel("h")

h_val = np.arange(0, 1000, 1)

def std_dev(h, cov): return np.sqrt(sum([(h**(i+j))*cov[i][j] for i
    in range(len(cov)) for j in range(len(cov))]))

## Given Data
ax1.errorbar(h, d, yerr=sig, fmt='kx', label="test data")
ax2.errorbar(h, d, yerr=sig, fmt='kx', label="test data")
ax3.errorbar(h, d, yerr=sig, fmt='kx', label="test data")

## Plot the three fits
fit1 = np.vectorize(lambda h, a: a*h)
ax1.plot(h_val, fit1(h_val, theta_hat_1), 'r', label="first
    hypothesis fit")
ax1.fill_between(h_val, fit1(h_val, theta_hat_1)-std_dev(h_val,
    covariance_1), fit1(h_val, theta_hat_1)+std_dev(h_val,
    covariance_1), label="one standard deviation")
ax1.legend()

fit2 = np.vectorize(lambda h, a, b: a*h + b*h**2)
ax2.plot(h_val, fit2(h_val, *theta_hat_2), 'c', label="second
    hypothesis fit")
ax2.fill_between(h_val, fit2(h_val, *theta_hat_2)-std_dev(h_val,
    covariance_2), fit2(h_val, *theta_hat_2)+std_dev(h_val,
    covariance_2), label="one standard deviation")
ax2.legend()

fit3 = np.vectorize(lambda h, a, b: a*h**b)
ax3.plot(h_val, fit3(h_val, *theta_hat_3), 'y', label="third
    hypothesis fit")
ax3.fill_between(h_val, fit3(h_val, *theta_hat_3)-std_dev(h_val,
    covariance_3), fit3(h_val, *theta_hat_3)+std_dev(h_val,
    covariance_3), label="one standard deviation")
ax3.legend()

# Chi-Square
```

```python
## Hypothesis 1
chi_sq_min_1 = sum(((d - f1(h, *theta_hat_1))/sig)**2)
ndof_1 = len(h)-len(p0_1)
print(f"---Hypothesis 1---\nChi-sq-min/NDoF:\t{chi_sq_min_1/ndof_1}\np-value:\t\t{chi2.sf(chi_sq_min_1, df=ndof_1)}")

## Hypothesis 2
chi_sq_min_2 = sum(((d - f2(h, *theta_hat_2))/sig)**2)
ndof_2 = len(h)-len(p0_2)
print(f"---Hypothesis 2---\nChi-sq-min/NDoF:\t{chi_sq_min_2/ndof_2}\np-value:\t\t{chi2.sf(chi_sq_min_2, df=ndof_2)}")

## Hypothesis 3
chi_sq_min_3 = sum(((d - f3(h, *theta_hat_3))/sig)**2)
ndof_3 = len(h)-len(p0_3)
print(f"---Hypothesis 3---\nChi-sq-min/NDoF:\t{chi_sq_min_3/ndof_3}\np-value:\t\t{chi2.sf(chi_sq_min_3, df=ndof_3)}")
```

py.py

```
---Hypothesis 1---
Chi-sq-min/NDoF: 165.49753617897267
p-value:  5.912867886616111e-142
---Hypothesis 2---
Chi-sq-min/NDoF: 21.58060512007567
p-value: 5.696198137767403e-14
---Hypothesis 3---
Chi-sq-min/NDoF: 1.2519761533914748
p-value: 0.2890541749901745
```