Communications Project

Fitting

Homework Results

Authors: Saqib Faraz 458317
Ankita Kannan Iyer 457620
Harsh Godhani 461986
Richard Grünert 289427
Kanaiyalal Thummar 435534

25.4.2023

1 Introduction

With fitting, we try to find a function that will best describe the relationship between the measurement and some independent variable. In general, we choose a function type, such as linear, polynomial, exponential, or logarithmic, and change its parameters such that the overall differences (*residuals*) of our function to the measured data are minimal.

Generally, this can be done with the method of least squares. Assuming our measured data points look like this

$$y_j = x + \epsilon_j$$

where x is the real value and ϵ is some measurement error, we choose x (i.e. find a function that produces x) such that

$$\sum_j \epsilon_j^2 = \sum_j (x - y_j)^2$$

is minimal. [1][5]

This kind of approach can be used to find (estimate) parameters of a model from real-world measurements. For example, if you measure the step response of a simple first order system, such as the temperature of a body that is cooling off or the voltage of a discharging capacitor, then you can fit an exponential function and estimate the time constant τ of the system, since it will simply be a parameter of the exponential function.

2 Analyzing Various Problems

The following will go through the given homework problems. Solutions were implemented in Python and for each problem, a separate script was created.

2.1 Constant Acceleration

If a brick is dropped from some arbitrary height (near the surface of the earth), the gravitational acceleration can be assumed to be constant and is approximately $g = 9.81\,\mathrm{m\,s^{-2}}$. The velocity of the brick will follow a linear relationship with time (assuming zero air resistance).

$$v(t) = -g \cdot t + v_0$$

The displacement of the brick from its original position will increase with squared time (just the integral of $v(t)$).

$$h(t) = -\frac{1}{2} \cdot g \cdot t^2 + v_0 \cdot t + h_0$$

In the script in appendix A.1, a random acceleration and random inital speed in the interval $[1, 10]$ are generated and some noisy data points are created using this acceleration, assuming $v_0 = 0$ and $h_0 = 0$. Then, using a linear fit, the the parameters of the estimating function are calculated. They were calculated by two methods: The first is a simple implementation with some matrix operations [4], and the second employs the `scipy.optimize.curve_fit` function from the `scipi` module. As can be seen in the resulting Figure 1, both approaches produces the same result.

2.2 Decelerating Car

The speed of a decelerating car will be modeled by the following equation. [6]

$$v(t) = A \cdot e^{-b \cdot t} + c$$

Consequently, the integral of this will result in the distance traveled

$$d(t) = \frac{v_0}{k} \cdot \left(1 - e^{-k \cdot t}\right) + d_0$$

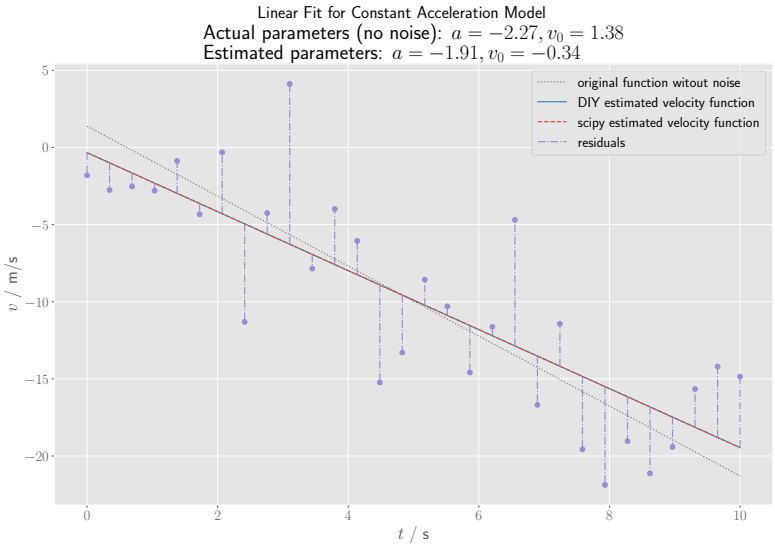

Figure 1: Result of fitting the noisy velocity data with a linear model function.

An exponential fit can actually be found using a linear model function as a helper, given some extra steps. We take the natural logarithm of $d(t)$ and substitute the parameters of the then linear equation: [1]

$$\underbrace{\ln d(t)}_{y(t)} = \underbrace{\ln a}_{x_1} + \underbrace{b}_{x_2} \cdot t$$

After supplying this to the linear model, we can perform the inverse operations (exponentiation) to get the wanted parameters. In this case, `scipy.optimize.curve_fit` was used, so that one can simply define the function to be optimized, making the substituion unnecessary. Figure 2 shows the result. This assumes that the velocity data can actually be negative.

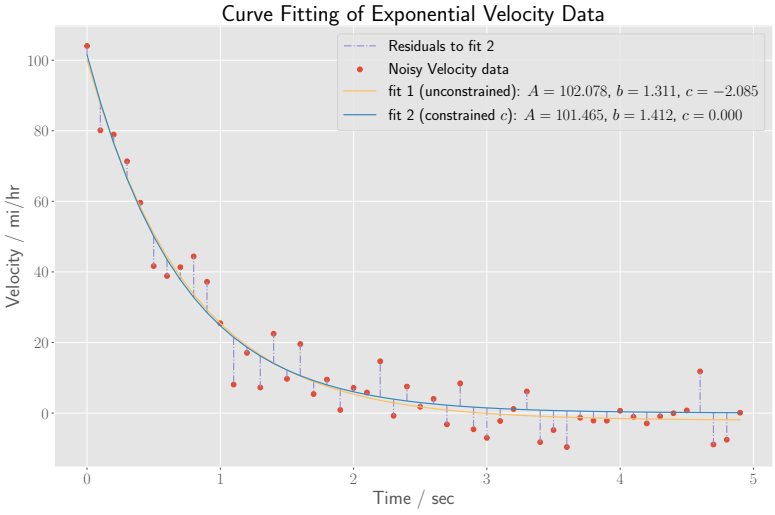

Figure 2: Result of fitting the noisy exponential velocity data.

When fitting exponential models, it might be necessary to constrain the additional y-offset parameter to zero if we know that our data actually decays to zero. However, it could also be that the time constant of a model with non zero y-offset more closely matches our system. This has to be adjusted based on the specific application.

2.3 Periodic Temperature Variation

The curve fitting of this kind of problem is again very straight-forward using `scipy.optimize.curve_fit`. The adjustable parameters for the sinusoidal function are amplitude, frequency, phase, and offset. The result can be seen in figure 3.

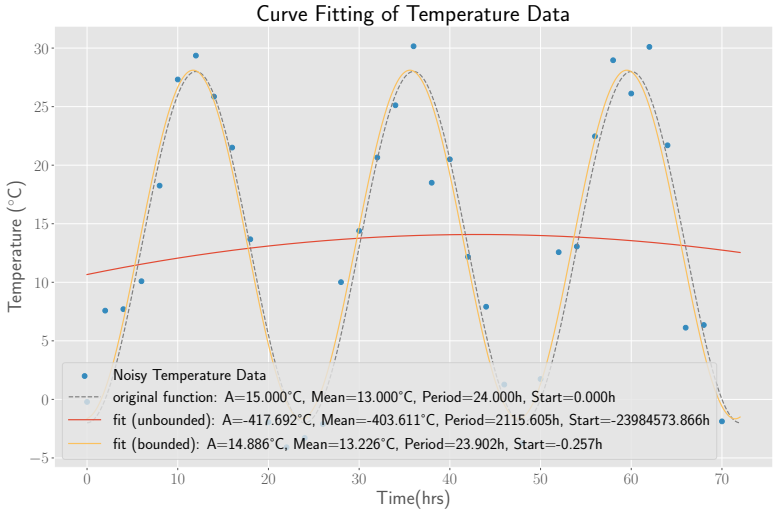

Figure 3: Result of fitting the noisy periodic temperature data.

Figure 3 also shows a problem with using the `curve_fit` function without first providing bounds on the decision variables. In this case, the period had to be relatively tightly bound to an interval of $[18\,\mathrm{h}, 24\,\mathrm{h}]$ to make the optimization produce a reasonable fit.

2.4 Randomly Distributed Variable

As required, a random discrete distribution of numbers from 0 to 10 was generated, where all values except 6, 7 and 8 are uniformly distributed. 6 and 8 appear with double the probability of the other values (12.5 %) and 7 with 4 times the probability (25 %). This was done by allocating all the numbers with their specific proportion and then picking a random entry from this allocation (see appendix A.4).

The problem with this is that trying to fit a continuous gaussian curve to the histogram data (by mean squared error) does not make much sense, mathematically. It is known that our random variable is discrete and can only take on values in the range of 0 to 10. The simple gaussian distribution fit will create a significant area outside of this interval, i.e. the area in the interval is not equal (or close) to 1. Secondly, the values (bins) of 9 and 10 close to the "mean value" 7 will have a probability that is much greater than those of the first bins $(0, 1, 2, \ldots)$, even though they should be close to equal which can be seen by a quick estimation of the area under the function (figure 4).

The solution is to use a method of dedicated "probability distribution fitting" such as the *Maximum likelihood estimation* and constrain the problem to a discrete distribution. Scipy implements this in `scipy.stats.fit`, where a certain distribution type can be passed. [2] For discrete distributions, it is a

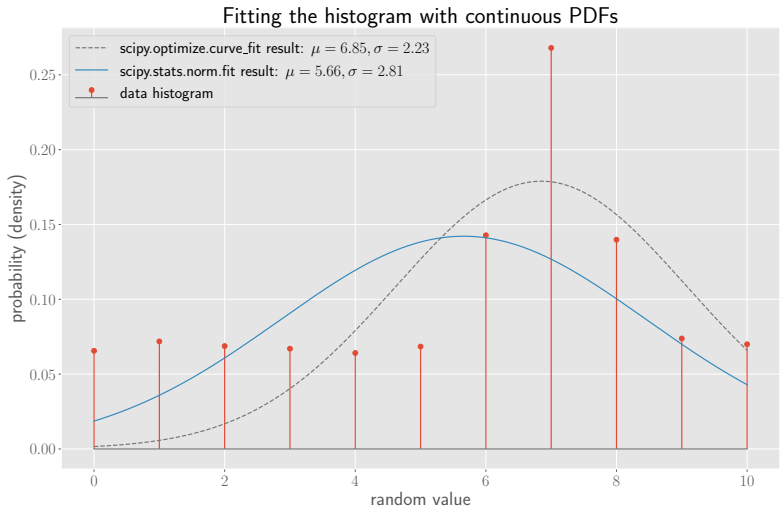

Figure 4: Fitting the discrete histogram data points with continuous PDF functions. Once with `optimize.curve_fit` (as before) and once with `stats.norm.fit`.

little bit more complicated, as a proper distribution function is not as easily found. For this kind of constructed scenario, the values follow a *multinominal distribution*, a discrete distribution where each value (or group) is associated with a distinct probability. The probability mass function of this is given by (see [3])

$$f(x) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k},$$

One could define a cost function that determines how far the specific distribution (PMF) deviates from the data. Then use that to perform a minimization optimization. However, this is not really useful in this case, as one might as well just use the values from the histogram.

3 Conclusions

A few different curve fitting problems have been studied. `scipy.optimize.curve_fit` provides a useful tool for this kind of task since it is very flexible regarding custom function definitions. Nevertheless, one has to be clear about the origin of their data and their intent when fitting to be able to make realistic predictions.

References

Literature

[1] Siegmund Brandt. *Datenanalyse für Naturwissenschaftler und Ingenieure*. 5th ed. ISBN 978-3-642-37664. Springer Spektrum, 2013.

[2] The SciPy community. *scipy.signal.windows.bartlett*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.bartlett.html. Accessed: 2023-04-17.

[3] The SciPy community. *scipy.stats.fit*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.fit.html. Accessed: 2023-04-24.

[4] The SciPy community. *scipy.stats.multinomial*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multinomial.html. Accessed: 2023-04-25.

[5] Eric W Weisstein. *Least Squares Fitting*. https://mathworld.wolfram.com/LeastSquaresFitting.html. From MathWorld–A Wolfram Web Resource, Accessed: 2023-04-23.

[6] Dirk Wentura, Benedikt Wirth, and Markus Pospeschill. *Multivariate Datenanalyse mit R*. 2nd ed. ISBN 978-3-662-65522-1. Springer, 2023.

[7] wgrenard. *Finding equation for exponential deceleration*. https://physics.stackexchange.com/questions/167111/finding-equation-for-exponential-deceleration#167122. Accessed: 2023-04-23.

Software Used

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.7. Python Software Foundation. 2019. URL: https://www.python.org/.

[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Python Code

A.1 Linear Fit

```python
# linear_fit.py

import numpy as np
import scipy
from numpy.fft import fft, fftshift
from matplotlib import pyplot as plt
import random


def fit_linear_least_squares_diy(x, y):
    # equations can be found at
    # https://mathworld.wolfram.com/LeastSquaresFitting.html
    n = len(x)
    x_sum = np.sum(x)
    x_squared_sum = np.sum(x ** 2)
    y_sum = np.sum(y)
    prod_sum = np.inner(x,y)

    A_mat = np.array([[n, x_sum], [x_sum, x_squared_sum]])
    B_mat = np.array([[y_sum], [prod_sum]])
    ab = np.matmul(np.linalg.inv(A_mat), B_mat)

    return (ab[0][0], ab[1][0])


def get_velocity_profile(a, v0):
    return lambda t: velocity(t, a, v0)


def velocity(t, a, v0):
    return -a * t + v0


def velocity_noise(length):
    return 3.8 * (np.random.normal(size=length))


def main():

    T_START = 0
    T_END = 10
    T_STEP = 1
    t_cont = np.linspace(T_START, T_END, 1000)
    t_discrete = np.linspace(T_START, T_END, 30)

    # generate random function parameters
    rand_accel = random.uniform(1.0, 10.0)
    rand_v0 = random.uniform(1.0, 10.0)

    # original values
    actual_velocity_fun = get_velocity_profile(rand_accel, rand_v0)
    actual_values = actual_velocity_fun(t_discrete)

    # add noise
    noisy_values = actual_values + velocity_noise(len(actual_values))

    # DIY approach
    est_v0, est_accel = fit_linear_least_squares_diy(t_discrete, noisy_values)
    est_accel = -est_accel
    estimated_velocity_fun = get_velocity_profile(est_accel, est_v0)

    # scipy approach
```

```
    popt, pcov = scipy.optimize.curve_fit(velocity, t_discrete, noisy_values)
    scipy_velocity_fun = get_velocity_profile(popt[0], popt[1])

    # calc residuals
    residuals = noisy_values - estimated_velocity_fun(t_discrete)

    # plotting
    # plot points and functions
    plt.figure()
    plt.scatter(t_discrete, noisy_values, color="C2")
    plt.plot(t_cont, actual_velocity_fun(t_cont),
             linestyle="dotted", color="gray",
             label="original function witout noise")
    plt.plot(t_cont, estimated_velocity_fun(t_cont),
             label="DIY estimated velocity function",
             color="C1")
    plt.plot(t_cont, scipy_velocity_fun(t_cont),
             linestyle="dashed", color="C0",
             label="scipy estimated velocity function")
    # plot residual lines
    y = estimated_velocity_fun(t_discrete)
    plt.vlines(t_discrete, y, y + residuals,
               color="C2", label="residuals", linestyle="dashdot")

    plt.legend()
    plt.suptitle("Linear Fit for Constant Acceleration Model")
    plt.title("Actual parameters (no noise): $a = "
              + str(round(-rand_accel, 2))
              + ", v_0 = " + str(round(rand_v0, 2))
              + "$\nEstimated parameters: $a = "
              + str(round(-est_accel, 2))
              + ", v_0 = " + str(round(est_v0, 2)) +
              "$")
    plt.xlabel("$t$ / s")
    plt.ylabel("$v$ / m/s")

    plt.savefig("outputs/linear_fit.pdf")

    plt.show()


if __name__ == '__main__':
    main()
```

A.2 Exponential Fit

```python
# exponential_fit.py

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

def exponential_decrease_func(x, A, b, c):
  # A is func value at 0, b is slope of the exponent, c is offset
  return A * np.exp(-b*x) + c


def main():
    # question 2b
    time_sec = np.arange(0, 5, 0.1)
    vel_data_sim = exponential_decrease_func(time_sec, 100, 1.5, 0)

    rng = np.random.default_rng()
    vel_noise = 6.2 * rng.normal(size=time_sec.size)
    vel_data = vel_data_sim + vel_noise

    # fit the data
    popt1, pcov1 = curve_fit(exponential_decrease_func, time_sec, vel_data)

    # this is a little cheated on the bounds because the
    # max bound always has to be greater than the min bound
    popt2, pcov2 = curve_fit(exponential_decrease_func, time_sec, vel_data,
                             bounds=([-np.inf, -np.inf, 0], [np.inf, np.inf, 0.000001]))

    estimated_data1 = exponential_decrease_func(time_sec, *popt2)
    residuals1 = vel_data - estimated_data1

    plt.figure()
    y = estimated_data1
    plt.vlines(time_sec, y, y + residuals1,
               color="C2", label="Residuals to fit 2", linestyle="dashdot")
    plt.scatter(time_sec, vel_data, color="C0", label='Noisy Velocity data')
    plt.plot(time_sec, exponential_decrease_func(time_sec, *popt1), color="C4",
            label='fit 1 (unconstrained): $A=%5.3f$, $b=%5.3f$, $c=%5.3f$' % tuple(popt1))
    plt.plot(time_sec, exponential_decrease_func(time_sec, *popt2), color="C1",
            label='fit 2 (constrained $c$): $A=%5.3f$, $b=%5.3f$, $c=%5.3f$' % tuple(popt2))
    plt.legend()
    plt.xlabel('Time / sec')
    plt.ylabel('Velocity / mi/hr')
    plt.title('Curve Fitting of Exponential Velocity Data')
    plt.savefig("outputs/exp_fit.pdf")
    plt.show()


if __name__ == '__main__':
    main()
```

A.3 Sinusoidal Fit

```python
# sin_fit.py

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

# sinusoidal function
def temp_func(x, amp_temp, mean_temp, period, start_hr):
  # Amp_temp is variation of temp in a day, Mean_Temp is average of the day
  return mean_temp - amp_temp * np.cos(2*np.pi*(1/period) * (x - start_hr))

def main():
    time_hr = np.arange(0, 72, 2) # 2 hours apart for 3 days
    time_hr_cont = np.linspace(0, 72, 1000)
    original_params = (15, 13, 24, 0)
    temp_data_sim = temp_func(time_hr, *original_params)

    rng = np.random.default_rng()
    temp_noise = 3.8 * rng.normal(size=time_hr.size)
    temp_data = temp_data_sim + temp_noise

    popt, pcov = curve_fit(temp_func, time_hr, temp_data)
    popt2, pcov2 = curve_fit(temp_func, time_hr, temp_data,
                             bounds = (
                                 [-np.inf, -np.inf, 18, -np.inf],
                                 [np.inf, np.inf, 24, np.inf]
                             ))

    plt.figure()
    plt.title('Curve Fitting of Temperature Data')
    plt.scatter(time_hr, temp_data, color="C1", label='Noisy Temperature Data')
    plt.plot(time_hr_cont, temp_func(time_hr_cont, *original_params),
             color="gray",
             linestyle="dashed",
             label='original function: A=%5.3f°C, Mean=%5.3f°C, Period=%5.3fh, Start=%5.3fh'
             % tuple(original_params))
    plt.plot(time_hr_cont, temp_func(time_hr_cont, *popt),
             color="C0",
             label='fit (unbounded): A=%5.3f°C, Mean=%5.3f°C, Period=%5.3fh, Start=%5.3fh'
             % tuple(popt))
    plt.plot(time_hr_cont, temp_func(time_hr_cont, *popt2),
             color="C4",
             label='fit (bounded): A=%5.3f°C, Mean=%5.3f°C, Period=%5.3fh, Start=%5.3fh'
             % tuple(popt2))
    plt.legend()
    plt.xlabel('Time(hrs)')
    plt.ylabel('Temperature ($^\circ$C)')
    plt.savefig("outputs/sin_fit.pdf")
    plt.show()


if __name__ == '__main__':
    main()
```

A.4 Probability Distribution Fit

```python
# gaussian_fit.py

import numpy as np
import scipy
from numpy.fft import fft, fftshift
from matplotlib import pyplot as plt
import random

# rigged discrete distribution
def rigged(size=None):
    # :)
    pool = [0, 1, 2, 3, 4, 5, 6, 6, 7, 7, 7, 7, 8, 8, 9, 10]
    return np.random.choice(pool, size)


def gaussian_distribution(x, mean, std):
    xn = x - mean
    xn = xn / std
    return 1 / (std * np.sqrt(2.0 * np.pi)) * np.exp(-(xn ** 2) / 2.0)


def main():
    X_START = 0
    X_END = 10
    X_STEP = 1
    x_cont = np.linspace(X_START, X_END, 1000)
    x_discrete = np.arange(X_START, X_END + 1)

    data = rigged(10000)
    hist, hist_bins = np.histogram(data, bins=11, density=True)

    mean_fit, std_fit = scipy.stats.norm.fit(data)

    popt, pcov = scipy.optimize.curve_fit(gaussian_distribution, x_discrete, hist)

    plt.figure()
    plt.stem(hist, label="data histogram")
    plt.plot(x_cont, gaussian_distribution(x_cont, popt[0], popt[1]),
             label="scipy.optimize.curve_fit result: $\mu = %.2f, \sigma=%.2f$"
             % (popt[0], popt[1]), color = "C3",
             linestyle="dashed")
    plt.plot(x_cont, scipy.stats.norm.pdf(x_cont, mean_fit, std_fit),
             label="scipy.stats.norm.fit result: $\mu = %.2f, \sigma=%.2f$"
             % (mean_fit, std_fit), color = "C1")
    plt.title("Fitting the histogram with continuous PDFs")
    plt.xlabel("random value")
    plt.ylabel("probability (density)")
    plt.legend()
    plt.savefig("outputs/gauss_cont.pdf")
    plt.show()


if __name__ == '__main__':
    main()
```