Report on higher order polynomial model least-square fitting

Dominik Brandstetter (1605547)
Department of Physics, Faculty of Natural Sciences, Graz University
(Dated: November 5, 2020)

As part of the exercise "Methods of Modeling and Simulation" (20W PHM.204UB), I present a report on the results of higher order polynomial fitting to a data set and model selection based on χ^2-hypothesis tests. Additionally, the code and results for a parameter distribution of a previously fitted data set will be given here. The code is generally written and does not utilise any loops to improve performance.

1. Introduction

The focus of this work lies on the comparison of different order polynomials in their capacity to fit our data and this capacity will be assessed by hypothesis tests. The capabilities of least-square fitting have been explored in this report[1]. This code is already equipped for higher order polynomials and, thus, will be utilised here again. In example 2 we will go back to an old data set from this report[1] and further explore the possibilities to analyse the errors in the model fit. This will lead us to a parameter distribution (joint probability distribution JPD), an in general multivariant distribution plotted as contour plot for different p-value boundaries.
After briefly discussing the simulation procedure in section 2 the results will be documented in section 3 and discussed in section 4. The code and the plots can be found in the appendix B and A respectively.

2. Simulation

The code for polynomial least-square fits is identical to the one used here[1]. It was extended by a function to calculate the multivariate distribution $f\left(\vec{\Delta a}\right)$. This was also entirely written in Python 3.8.5 and utilised the following packages: NumPy 1.19.1 to handle vector-shaped data and its "linalg" module to perform matrix opertions, SciPy 1.5.2 for the χ^2-distribution and MatPlotLib 3.3.1 was used for plotting.
The evaluation of $f\left(\vec{\Delta a}\right)$ was done for a whole meshgrid $\Delta p_i - \mu_i = -n_i * \sigma_i, ..., n_i * \sigma_i$ where $n_i = n = 3$ denotes the range for each parameter using a generalised matrix multiplication ("np.einsum") to improve performance by not using generally slow loops.

3. Results

3.1. Example 1

For example 1 the primary question is what order polynomial fits the data best. The actual best-fit parameters for each model are not quite as interesting as their completely agree with the results for "np.polyfit" routine. Thus, I will refrain from listing them here. Much more interesting are the p-values and the χ^2 error for each model listed here

TABLE I: Goodness-of-fit criteria for different order polynomial fits for the example 1 data set.

Polynomial Order n	χ^2 / [1]	p value / [1]
2	36.410	0.000006
3	2.850	0.827
4	1.686	0.891
5	1.663	0.797
6	0.935	0.817

4. Discussion

4.1. Example 1

As can be seen in TAB.I, the error χ^2 goes down as the order goes up. This is to be expected as higher order polynomials "contain" lower order polynomials and with more degrees of freedom a fit is bound to improve. Taking the extreme, a polynomial of order N fits N data points exactly, as it is a well-posed linear system of equations with a square matrix. However, as the degrees of freedom for the polynomial (i.e. the order) goes up, the degrees of freedom for the χ^2-distribution (i.e of data points - order of polynomial + 1) goes down. With that the χ^2_α threshold also goes down. This means that even though the fits are closer to the observed data points, it is possible for them to be rejected by the hypothesis test.
That is exactly what we see in the p-values for the example 1 data set. The p-values spike clear at an order 4th polynomial, suggesting that this model fits the data best.

It is to be said, however, that all models except the second order one, clear the $p = 0.05$ threshold easily and with the small amount of data points given it is hard to make a definitive statement about which model is to be chosen. For any regression problem also the physical nature and the laws the data is to be expected to obey should be part of the analysis.

4.2. Example 2

As can be seen in FIG.?? and FIG.?? the example 2 data set is highly elliptical and also tilted almost 45°. The latter suggested a high correlation between the two parameter in the linear order fit.

---

[1] D. Brandstetter. *Report on polynomial model least-square fitting*, 2020.

Appendix A

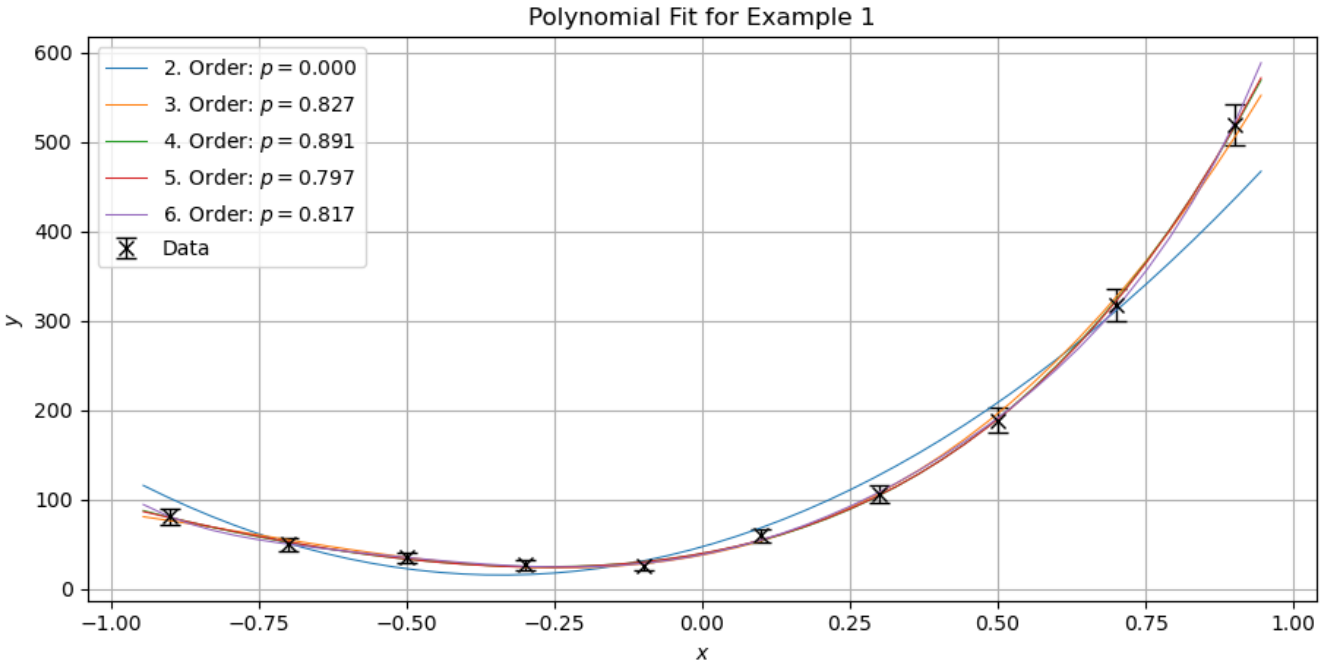

FIG. 1: Different order polynomial fits to the example 1 data set.

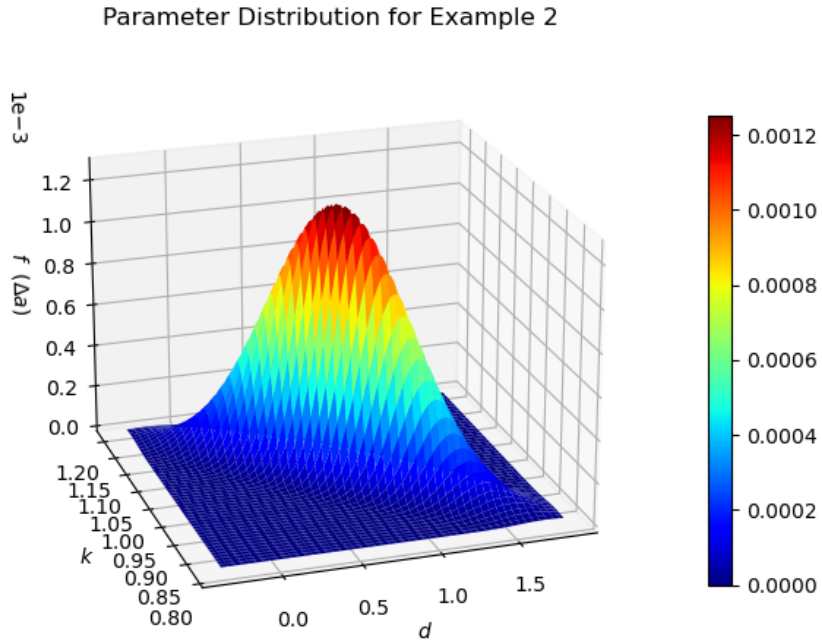

FIG. 2: Surface plot for the normalized parameter density (i.e. probability distribution of the bi-variate distribution) for the example 2 data set.

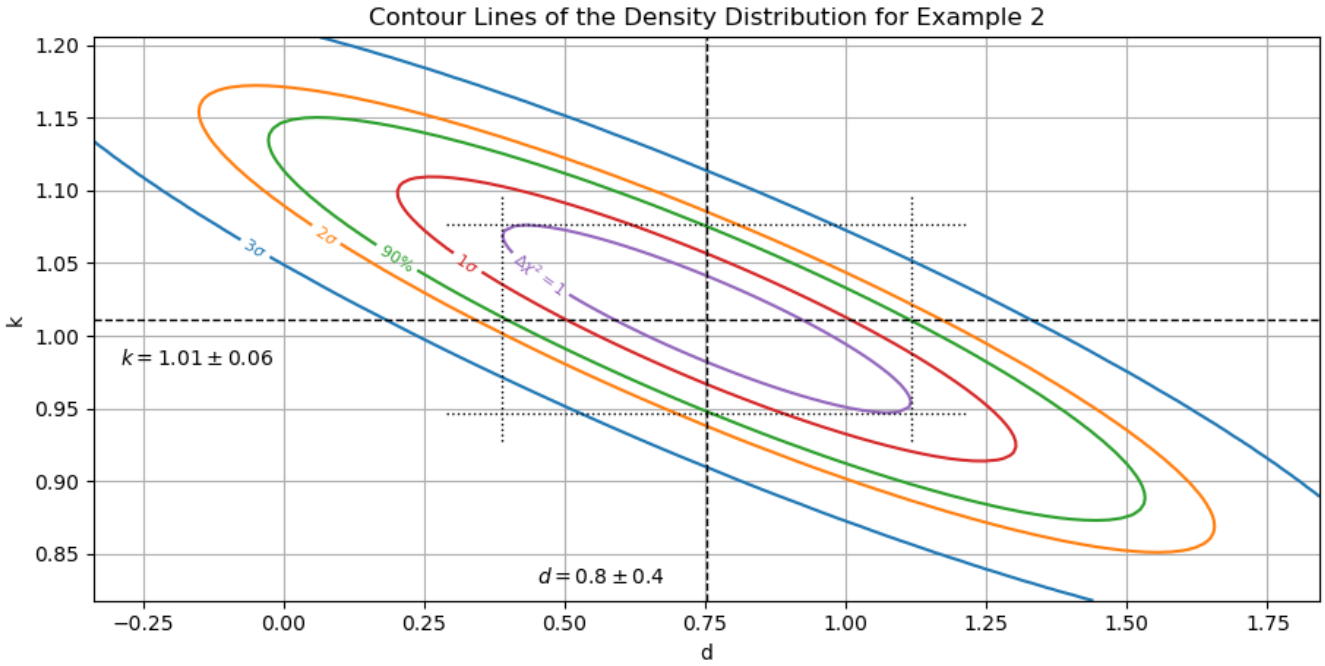

FIG. 3: Contour plot for the FIG**??** probability density with contour lines for different σ levels centred around the parameters (dashed black). The $\Delta\chi^2 = 1$ contour level is surrounded by a dotted rectangle displaying the standard deviation for each parameter.

Appendix B

```
import numpy as np
from scipy import stats

def param_distribution(C, contour_levels=[], num_sigma=3, res=50):

    # Set variables
    C_inv = np.linalg.inv(C)
    stds = np.sqrt(np.diagonal(C))
    m = len(stds)

    # Mesh grids for each parameter
    delta_p_i = [np.linspace(-num_sigma * std, num_sigma * std, res)
                 for std in stds]
    meshgrids = np.meshgrid(*delta_p_i)

    # Stack them in the last axis
    delta_a = np.stack(meshgrids, axis=m)

    # Generalised dot product in the exponent
    exponent = np.einsum('...j,...j', delta_a,
                         np.einsum('ij,...j', C_inv, delta_a))

    pre_factor = np.sqrt(np.linalg.det(C_inv) / (2 * np.pi)**m)
    f_a = lambda x: pre_factor * np.exp(-x / 2)

    # Return contour lines for certain alpha or chi^2 values
    levels = []
    for level in contour_levels:
        if level < 1:
            level = stats.chi2.ppf(q=level, df=m)

        levels.append(f_a(level))

    return meshgrids, f_a(exponent), levels
```