PolyCurveFit: An Optimized Scipy CurveFit Wrapper for Polynomial Data Fitting

Dipl.-Ing. Karl Breuer[a]

[a] *Softwaredeveloper for engineering and industry, mail@karlbreuer.com*

Abstract

This two pager introduces PolyCurveFit, an optimized wrapper for SciPy's curvefit function, engineered to improve the fitting process of polynomial functions to data. This innovative tool not only facilitates parameter fitting but also optimizes the shape of the polynomial function, thus eliminating the traditional need for pre-selecting a fitting function. This results in a more streamlined and comprehensive approach to data fitting, especially beneficial in scientific research.

Keywords:
datafitting, python, scipy, fittingAlgorithm, regression

1. Introduction

Data fitting is a pervasive task in scientific research, often requiring investigators to preselect suitable functions to fit their data parameters, typically guided by literature. However, recent advancements in computational capabilities have paved the way for more sophisticated and comprehensive methods of data fitting.

$$\text{solve } f(x,p) = y \quad \text{for } p \text{ and } f(x,p) \tag{1}$$

The PolyCurveFit tool, designed to leverage these advancements, allows for brute force fitting of multiple functions to a single dataset, determining the best fitting function with a given number of parameters. The tool concentrates on polynomial functions, extensively used in scientific research due to their simplicity and versatility, and assigns integer exponents, both negative and positive, based on empirical evidence.

2. Method

The tool uses a standard polynomial function, where X represents the input value matrix, E denotes the exponent matrix, P refers to the fitting parameter vector, and Y signifies the calculated value matrix.

This is the general form for a polynom that calculates one single value as a result.

$$y = f(X,E,P) = \sum_{j=0}^{N-1} P_j \prod_{i=0}^{d-1} X_i^{E_{ji}} \tag{2}$$

This is the first term of the equation of the equation above.

$$y_{\text{term1}} = P_0 \cdot X_0^{E_{00}} \cdot X_1^{E_{01}} \cdots X_{d-1}^{E_{0,d-1}} \tag{3}$$

In the code a more compact equation was used that takes in a 2D array for all X values and outputs a 1D array for all outputs. This way a whole dataset can be expressed by a single equation:

$$Y = f(X,E,P) = \left[\sum_{j=0}^{N-1} P_j \prod_{i=0}^{d-1} X_{ji}^{E_{ji}}, \ldots, \sum_{j=0}^{N-1} P_j \prod_{i=0}^{d-1} X_{ni}^{E_{ji}}\right] \tag{4}$$

This is for better understanding the first term of the standard function for the first result value in Y:

$$Y_0\text{term1} = P_0 \cdot X_{00}^{E_{00}} \cdot X_{01}^{E_{01}} \cdots X_{0,d-1}^{E_{0,d-1}} \tag{5}$$

The key concept is the brute force computation of all possible E combinations, given a user-defined parameter number and a range of exponent values. Importantly, selecting a large parameter range can lead to extensive computations, which the tool can manage but may demand substantial time and computational power.

The core method capitalizes on the renowned curvefit function from Python's SciPy library, utilizing the base polynomial function and the given values to determine the optimal parameter fit. Crucially, the tool maximizes computational power by employing all available cores, facilitating large-scale computations. This parallelization is achieved by evenly distributing the computation tasks among cores.

A Python generator is employed at this stage to produce every unique E combination as required by the ongoing process, without retaining it. This step is essential to avoid excessive RAM usage and potential memory overload.

The tool records the best result, expressed by the sum of the least squares (SSR) for each worker process. It does this only when a better fit is found, saving this data to a file instead of relying on inter-process communication, thus enhancing computational speed and algorithmic simplicity. While the logging step involves slow file writing, it is retained as it only occurs when a new better solution is discovered, and it ensures the availability of the best solution for every worker, even if the process is prematurely terminated.

The final step involves comparing the best fit logs across all workers and recording the overall best fit.

The result is a Parameter Matrix, Popt, Pcov, and E for the lowest found SSR, defining a polymorphic function that fits the given data.

After this polymorphic function is found it can be used for a much faster usual parameter fit on similar structured data.

3. Results

The main result of this work is the free and open source version of the PolyCurveFit algorithm that is available on Github https://github.com/Karl1b/polyCurve_fit.

The preliminary testing of PolyCurveFit using curve data for centrifugal pumps has displayed encouraging outcomes, indicating its potential for broader applications. The algorithm's rapid setup time, coupled with its ability to operate autonomously, renders it an ideal tool for a diverse range of projects.

In the appendix, we provide a specific example. The author has obtained permission from KSB to utilize publicly available data from https://ksb.com/, and has chosen the Etanorm 060-050-125 as a model for demonstration. The results of this particular example are promising; the polynomial parameters (denoted as E) acquired for a standard data set could likely be applied to other comparable data sets in PolyCurveFit. This implies that the time-consuming fitting process only needs to be conducted for a specific range of measurement data. Once accomplished, the entire data set could potentially be fitted using conventional methods, necessitating far less computing power. In this specific instance, it is probable that all Etanorm sizes can be fitted with low error using the derived function.

The quality of the fitting can only be approximated, as the author had to manually transfer data points from a drawing, and the quality of the data points themselves is uncertain.

PolyCurveFit presents an efficient and streamlined solution for fitting polynomial functions to data, reducing setup time and enabling a convenient "set it and forget it" operation. Its versatility hints at potential value across a wide spectrum of scientific applications.

4. Acknowledgements

While the development of PolyCurveFit does not directly draw from specific resources, it is built upon the curvefit function from the SciPy library. The author wishes to express gratitude to the Scipy team for this foundation. https://scipy.org/

Future enhancements to the tool will continue to rely on user feedback and advancements in the broader computational field.

Appendix A. Real world data from a centrifugal pump

The KSB etanorm 060-050-125 is used as an example.

The raw dataset is fitted with

```
polyCurve_fit(filename="etanorm_h.csv", Parameters=2, lower=-3, upper=5)
polyCurve_fit(filename="etanorm_h.csv", Parameters=3, lower=-3, upper=5)
```

rpm in 1/min	diamter mm	q in m3/h	h in m
1450	112	0	4
1450	112	30	3
1450	112	44	1.7
1450	120	0	4.6
1450	120	30	3.8
1450	120	48	2.1
1450	130	0	5.4
1450	130	30	4.8
1450	130	51	2.8
1450	142	0	6.4
1450	142	30	5.8
1450	142	55	3.5
2900	112	0	15.9
2900	112	40	14.1
2900	112	85	8
2900	120	0	18.1
2900	120	40	17
2900	120	90	10
2900	130	0	21.8
2900	130	40	20.2
2900	130	100	12
2900	142	0	25.9
2900	142	40	24.5
2900	142	110	14
960	112	0	1.75
960	112	20	1.3
960	112	29	0.7
960	120	0	2
960	120	20	1.65
960	120	31	0.95
960	130	0	2.39
960	130	20	2.1
960	130	33.8	1.22
960	142	0	2.81
960	142	20	2.5
960	142	36	1.6

Table A.1: Raw input data for KSB Etanorm 060-050-125

These are the resulting best logs from the fitting process:

```
{"worker_id": 7, "SSR": 1.41711820944752, "P_opt":
[1.5251590372839772e-10, -0.13003739958313548],
"P_cov": [[3.1229332368877225e-25, -6.527254476558119e-16],
[-6.527254476558119e-16, 3.057960866749138e-06]],
"E": [[2, 2, 0], [0, -1, 2]]}
Total time: 34.98347568511963 seconds
```

This is equivalent to:

$$h = 1.5251590372839772e - 10 \times \text{rpm}^2 \times d^2 - 0.13003739958313548 \times d^{-1} \times q^2$$

```
{"worker_id": 12, "SSR": 0.3594089302432084, "P_opt":
[-13.234750765255821, 1.5159271348105557e-10,
-1.984250305787992e-06],
"P_cov": [[0.026105189916296342, -2.1753637335725905e-14,
-9.136622908175011e-09],
[-2.1753637335725905e-14, 8.049355026704036e-26,
4.263878267871656e-22],
[-9.136622908175011e-09, 4.263878267871656e-22,
6.206508126705012e-15]],
"E": [[0, -2, 2], [2, 2, 0], [-3, 3, 5]]}
Total time: 34899.62151002884 seconds
```

This is equivalent to:

$$h = -13.234750765255821 \times d^{-2} \times q^2 + 1.5159271348105557e - 10 \times \text{rpm}^2 \times d^2 - 1.984250305787992e - 06 \times \text{rpm}^{-3} d^3 \times q^5$$

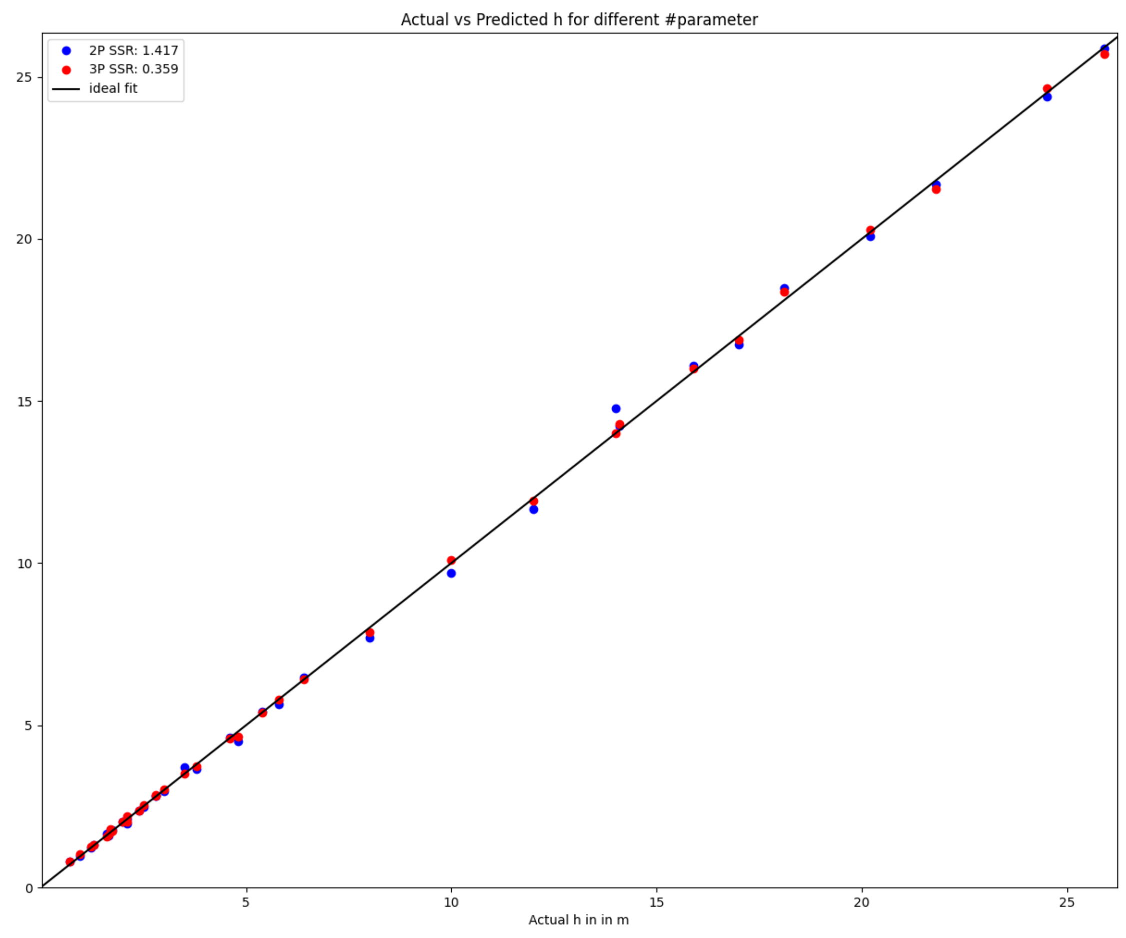

Figure A.1: Input vs Output data for 2 and 3 paramters

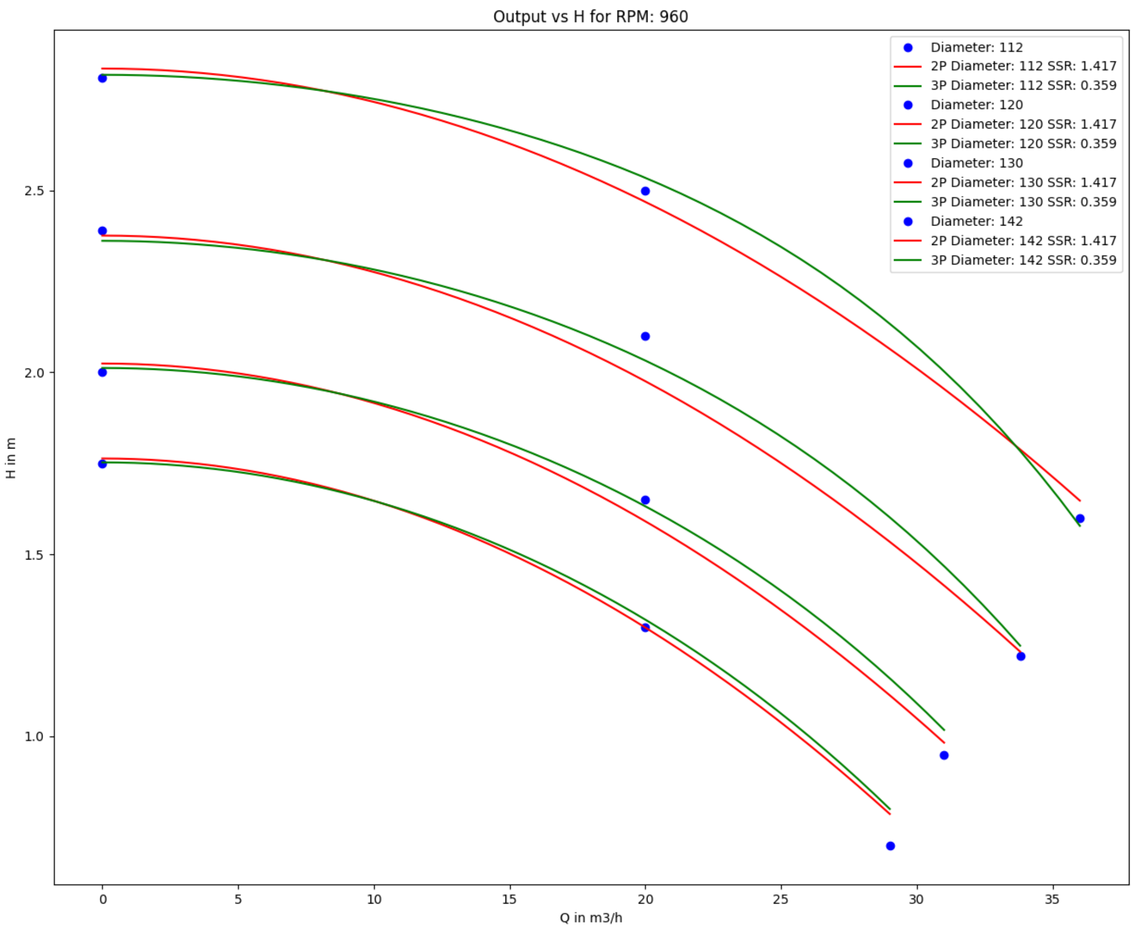

Figure A.2: Q and H for different diamters at 960 rpm

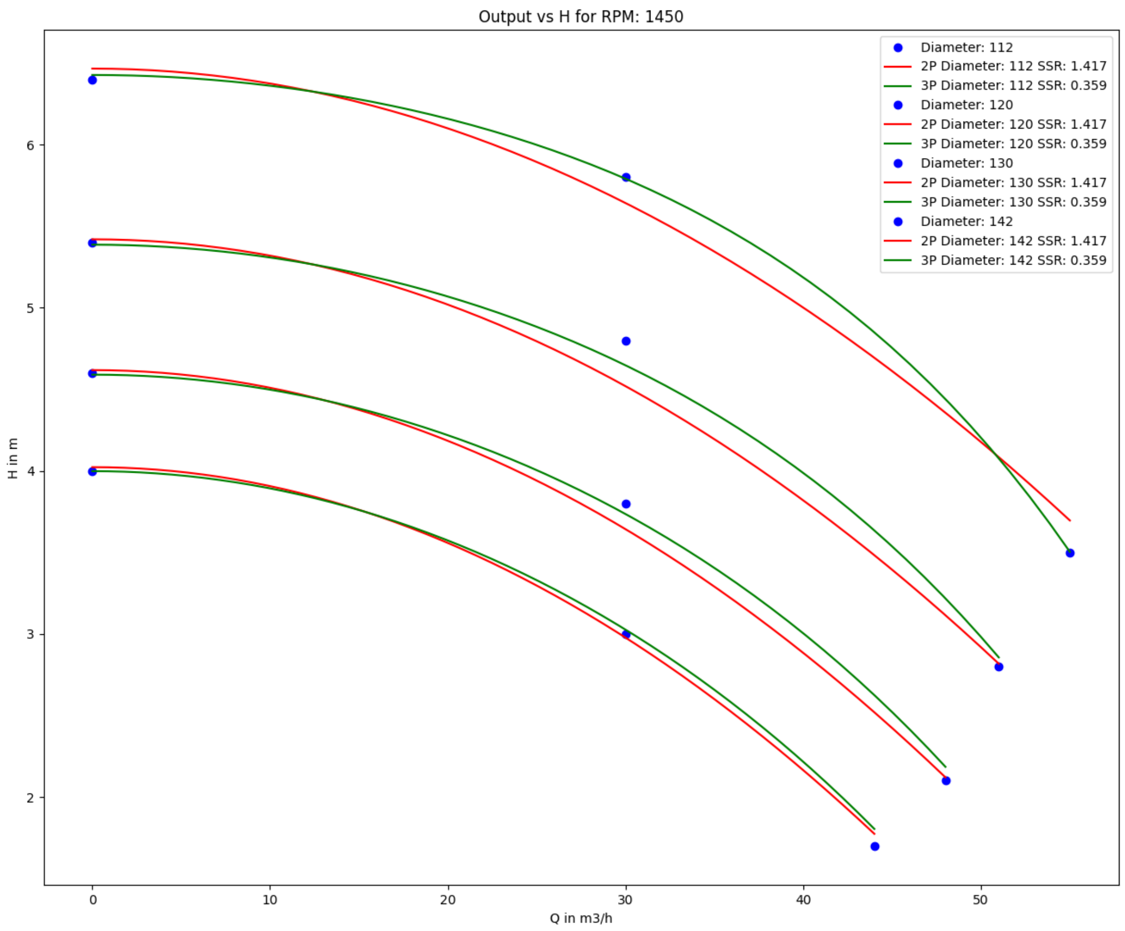

Figure A.3: Q and H for different diamters at 1450 rpm

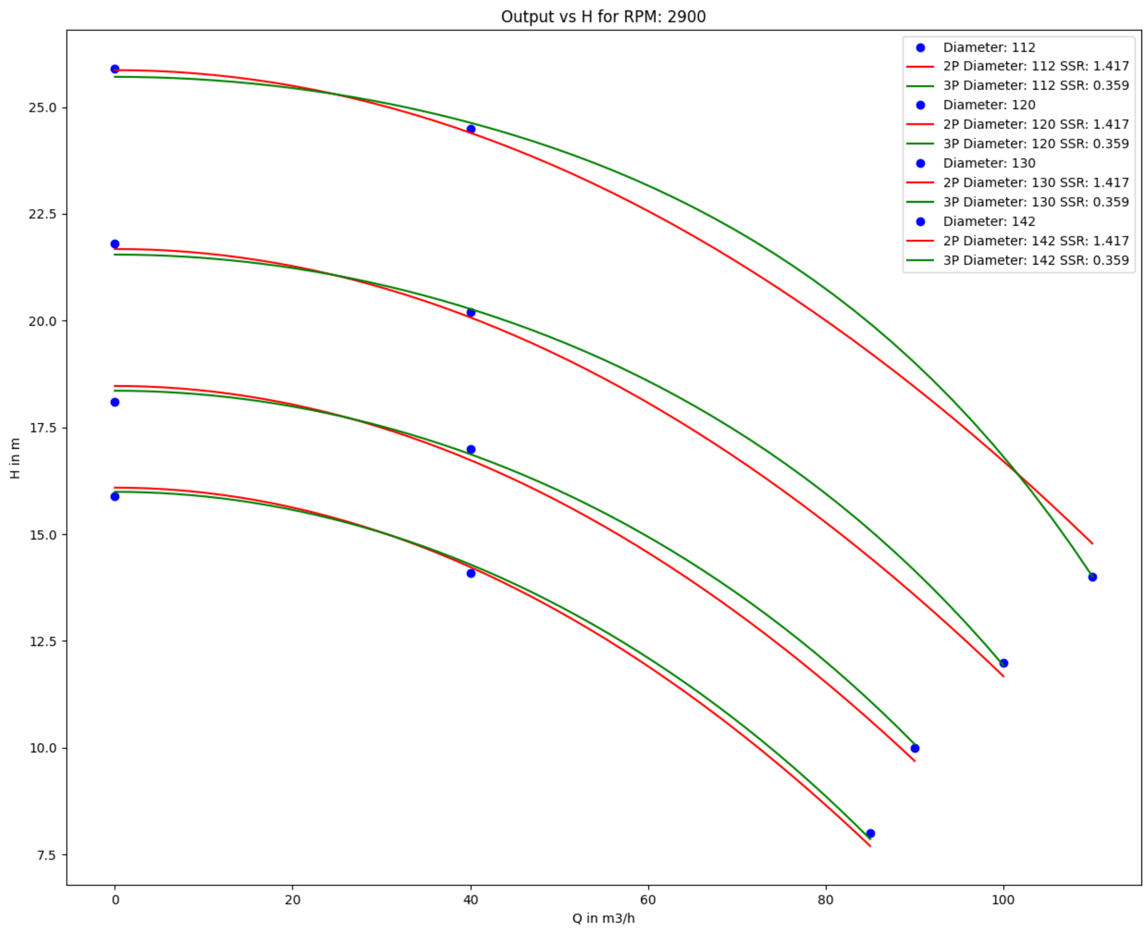

Figure A.4: Q and H for different diamters at 2900 rpm