Open Access | https://doi.org/10.25080/xydg9873

Enhancing Curve Fitting with SciPy: Interactive Spline Modeling and Reproducibility with SplineCloud

Vadym Pasko[1,2]

[1]SplineCloud, [2]Kyiv Aviation Institute

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

Abstract

Curve fitting is a fundamental task in data science, engineering, and scientific computing, enabling researchers to extract meaningful relationships from data. However, selecting and tuning the right fitting model for complex, noisy, or multidimensional data remains a significant challenge. SciPy plays a critical role in addressing these challenges by providing robust spline fitting methods that offer flexibility and precision. Yet, fine-tuning spline parameters, ensuring stability in extrapolation, and sharing fitted models for reproducibility remain open problems.

In order to address these challenges, we developed SplineCloud - an open platform that provides interactive spline fitting capabilities and uses SciPy on the backend. SplineCloud allows the construction, analysis, and exchange of spline-based regression models using SciPy's `interpolate` module. SplineCloud's curve fitting tool extends the capabilities of SciPy spline fitting methods by enabling researchers to fine-tune spline parameters: knot vector, control points, and degree interactively, and providing the ability to quickly analyze the accuracy of the models. Models constructed on the platform obtain their unique identifiers and become reusable in code, fostering better collaboration and knowledge transfer. Reusability of spline curves and underlying datasets in code is enabled via the open-source SplineCloud client library called `splinecloud-scipy`, which is also based on SciPy. The proposed approach of interactive cloud-based fitting improves data processing workflows, allows separating data preparation and approximation routines from the main code, and brings FAIR principles to the curve fitting, enabling researchers to construct and share libraries of empirical data relations.

Keywords curve fitting, spline fitting, FAIR, empyrical relations, data analysis, regression models

Published Jul 10, 2025

Correspondence to
Vadym Pasko
vadym@splinecloud.com

1. Introduction

The adoption of FAIR data principles [1] has fundamentally transformed research practices across scientific disciplines. Initially designed to enhance research data management and increase transparency, these principles now extend to a broad range of digital scientific artifacts, including computational models, source code, 3D models, and other digital objects. Widespread adoption of FAIR principles fosters improved reproducibility and facilitates the integration and extension of scientific results. However, not all digital objects achieve the same level of FAIRness, and many platforms lack support for the diverse range of data formats and objects used in science and engineering.

This paper focuses on enhancing reusability — and thereby reproducibility — in the context of regression modeling, a technique used to mathematically describe the relationship between dependent and independent variables. This process, commonly known as curve fitting, is widely employed in data science, engineering, and scientific computing, particularly for processing experimental data and extracting interpretable models for further analysis. Of particular practical interest is spline fitting, which effectively models complex empirical relationships without explicit underlying mathematical laws. A formal overview of splines and spline fitting is provided in Some Theoretical Background Behind Splines and Spline Fitting With SciPy.

SciPy's `interpolate` module offers extensive functionality for constructing splines, from simple interpolating curves to advanced smoothing splines and parametric representations. Its spline fitting methods provide robust algorithms widely used to model complex, noisy, or multidimensional data dependencies. Nonetheless, the capabilities of splines cannot be fully leveraged through purely programmatic interfaces. In contrast, modern graphical modeling software with interactive spline manipulation affords a level of control and intuitive adjustment not achievable by traditional automatic fitting methods. This, together with a problem of reusability of obtained models, their accessibility, and interoperability, leaves a way for thorough considerations towards alternative approaches to the curve fitting process.

These considerations eventually led to the development of a dedicated platform for interactive spline fitting, which incorporates FAIR principles and adapts them to the regression models. Built upon SciPy's routines, SplineCloud enables collaborative, transparent, and reusable curve fitting[1]. Sections 4 through 8 of this paper detail the similarities and distinctions between SciPy's and SplineCloud's approaches, illustrating a novel paradigm for collaborative data fitting.

2. Some Theoretical Background Behind Splines

Splines are piecewise-defined functions used extensively in numerical analysis, computer-aided geometric design, and data fitting. The fundamental idea behind the spline fitting process is to construct a smooth function that matches a set of data points or satisfies a set of constraints, while preserving computational efficiency and numerical stability. Splines can be represented in several forms, each suitable for different applications: Piecewise Polynomial, Hermite, Bézier, B-spline, NURBS (Non-Uniform Rational B-Splines). Amongst these forms, a special place takes B-Spline representation - where the spline function (or curve) is expressed as a linear combination of basis functions.

A distinction must be made between spline functions and parametric spline curves:

- **A spline function** is a scalar-valued function $S(x)$, defined over a single independent variable x, typically used in interpolation or regression of scalar data.
- **A parametric spline curve** defines a vector-valued mapping from a scalar parameter t to a multidimensional space:

$$\boldsymbol{C}(t) = (x(t), y(t), z(t), \ldots) \tag{1}$$

where each coordinate function $x(t), y(t), \ldots$ is a spline function in t. This is the common representation in geometric modeling and computer graphics.

A particularly efficient and widely used spline representation is the B-spline (basis spline). B-splines are defined via a set of control points $\boldsymbol{P}_i$ and a knot vector t_i, and provide local control, numerical stability, and efficient evaluation. A B-spline curve of degree k is defined as:

$$\boldsymbol{C}(t) = \sum_{i=0}^{n} \boldsymbol{P}_i B_{i,k}(t) \tag{2}$$

[1]First described in Mastering Curve Fitting. How to Fit Complex Data with Parametric Splines

where $\boldsymbol{P}_i$ are control points, $B_{i,k}(t)$ are the B-spline basis functions of degree k, defined recursively by the Cox–de Boor formula [2], [3]:

$$B_{i,0}(t) = \tag{3}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t) \tag{4}$$

In (4) possible divisions by zero are resolved by the convention that 'anything divided by zero is zero' [4]. This formulation separates geometry (control points) from basis functions, allowing flexible manipulation and efficient computation of spline curves.

Parametric splines are widely used in computer graphics, computer-aided design (CAD), and more general geometric modeling, where the goal is to design and manipulate accurate and smooth curves and surfaces in two or three dimensions. Their unique properties allow for high shape flexibility, local control over curve shape and its continuity. An intuitive visual explanation of splines, different forms of their representation and unique properties is given in the video by Freya Holmér [5].

However, despite their geometric advantages, parametric splines are rarely used in statistical data fitting and regression analysis. The primary limitation is the parameterization problem: for observed data points (x_i, y_i), there is no universally optimal method to assign parameter values t_i that correspond meaningfully to the underlying data relationship. While heuristic approaches such as chord-length or centripetal parameterization are commonly used, the choice can significantly influence the quality of the resulting fit [6]. Additionally, parametric splines do not provide the direct functional relationship $y = f(x)$ that is typically required for prediction and statistical inference in data analysis contexts. This is why spline functions are often a more popular choice.

3. Spline Fitting With SciPy

SciPy provides a robust and flexible set of spline fitting tools for both **interpolation** and **approximation** through the `scipy.interpolate` module. In terms of univariate fitting, SciPy supports several spline fitting methods, including:

Interpolating splines (Figure 1 a) - exact fit to the data. This constructs a spline $S(x)$ such that:

$$S(x_i) = y_i \quad \text{for all } i \tag{5}$$

and ensures continuity of first and second derivatives (C^2 continuity for cubic splines).

Smoothing splines (Figure 1 b) - approximate fit with smoothness penalty. The smoothing spline minimizes the penalized least-squares objective:

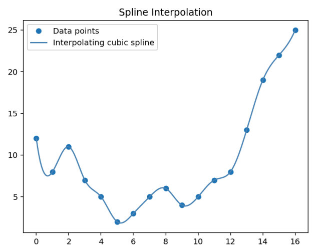
$$\min_S \left\{ \sum_{i=1}^{n} (y_i - S(x_i))^2 + \lambda \int_a^b (S''(x))^2 dx \right\} \tag{6}$$

where λ is a regularization parameter related to s.

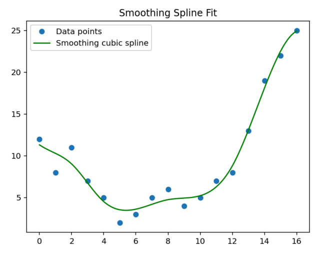
Least squares splines (Figure 1 c) - approximate fit with squared residual minimization penalty:

$$\min_S \sum_{i=1}^{n} (y_i - S(x_i))^2 \tag{7}$$

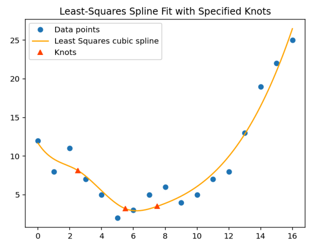
Parametric splines. Used to fit looped curve, isolines, or 3D curves.



(a) Simple interpolating cubic spline.



(b) Smoothing cubic spline.



(c) Least-Squares cubic spline.

Figure 1. *Spline fitting results with a use of SciPy's `interpolate` methods.*

3.1. *Pain Points of Pure Programmable Spline Fitting*

SciPy provides a programmatic interface to robust spline fitting methods suitable for a variety of tasks, particularly when working with complex experimental or statistical data. However, despite offering multiple fitting methods and adjustable parameters, selecting an appropriate combination can be challenging. As noted by [7], issues such as overfitting and extrapolation control lack straightforward solutions and often require multiple iterations, either with visual inspection or advanced scripting.

The mathematical formulation of B-splines—especially parametric B-splines and NURBS—enables fine control over curve shapes through manipulation of control points and knot vectors. While this functionality is widely leveraged in computer graphics and CAD applications, it is not supported in SciPy.

Introducing interactive control over curve geometry can improve fitting accuracy and reduce the time spent tuning parameters in search of acceptable results from automatic fitting routines.

Another limitation is the lack of portability and reproducibility of fitted models. SciPy spline objects are tightly coupled to the Python runtime and local environment. Although spline parameters (order, knot vector, and coefficients) can be exported, reconstructing models from these components requires an understanding of B-spline structure. Model serialization is also possible, but may not be practical in all scenarios.

These limitations hinder reproducibility and lead to duplicated effort, where researchers repeatedly perform the same fitting procedures on shared datasets, reimplementing logic and parameters independently. Even when code and data are shared, fitted models often remain environment-specific and are difficult to reuse across projects.

Overcoming these challenges requires workflows in which spline models can be edited interactively, stored independently from source code and raw data, and reused across tools and teams.

These needs motivated the development of SplineCloud, a platform that extends SciPy's spline fitting with interactivity, transparency, and full model lifecycle support.

4. Introducing SplineCloud

SplineCloud is an open platform for formalized knowledge exchange. It is designed to make technical data, like results of simulations, tests and modeling, more accessible and reusable in computer code. Its architecture and instruments help reduce the routine overhead of data collection, processing, and model reconstruction in code.

SplineCloud is free to use for everyone, publishing open data, however, its source code is closed (as of now). Repositories with limited access are a part of the platform's monetization model. All tools are free to use without limitations (except repository size, which as of now is limited to 3 GB). The platform's main features are: plot digitizer, advanced spline fitting tool, open API, and client libraries for Python and MATLAB (libraries for other languages are planned for future implementation).

The philosophy behind the platform is in representing data relations in the form of spline models, which can be reused in code. In this way, SplineCloud can be considered as a repository-based open library of functional relations. The typical user journey on the platform is given in Figure 2.

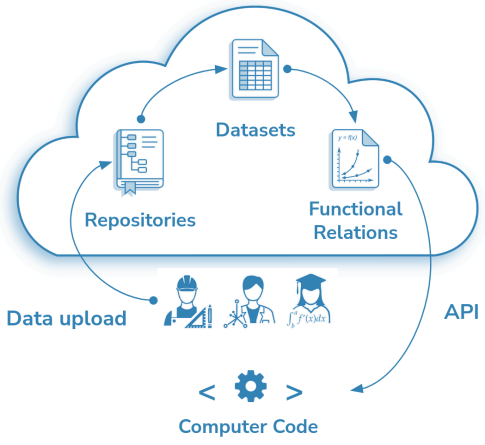


Figure 2. *SplineCloud workflow scheme: upload data to repository, define datasets, create data relations using spline fitting tool, share and access curves and datasets in code via API.*

The platform's data structure is given in Figure 3. Each repository may contain multiple data files (uploaded by authors), which serve as sources for datasets; authors and other users can define subsets (clean data) related to datasets. Subsets are used for the construction of data relations, which capture relationships between columns in subsets. Parametric spline curves are used to model these relations.

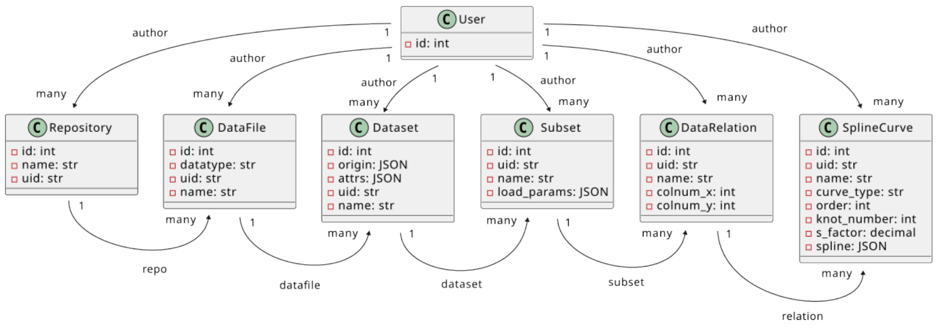


Figure 3. *SplineCloud objects relationship diagram*

Spline fitting tool, a main instrument of the platform, is built on top of SciPy's interpolate module and provides a visual interface to three main spline fitting methods described in Interactive Spline Fitting Workflow: Interpolating Splines, Smoothing Splines, Least Squares Splines.

The web interface of the spline fitting tool is powered by D3.js and JavaScript build of verb-nurbs - an open-source library for NURBS modeling. The RESTful API and an open-source Python client library (splinecloud-scipy) allow the reuse of data objects and spline curves in code. In this way, SplineCloud addresses the key limitations of traditional spline fitting workflows by offering the following core capabilities:

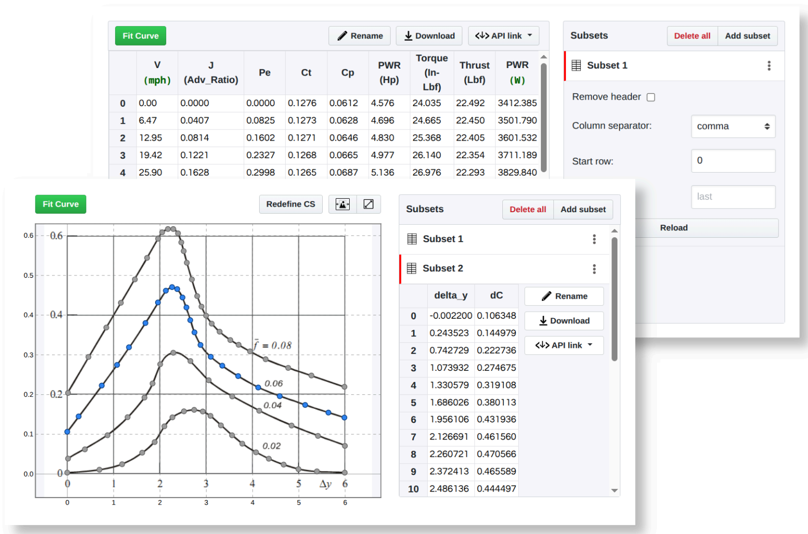
- Interactive Spline Modeling;
- Reusability and Interoperability of Models;
- Model Traceability.

5. Interactive Spline Fitting Workflow

Interactivity in the curve fitting process significantly simplifies and accelerates model construction. It enables users to identify overfitting, discontinuities, or extrapolation issues early, and make real-time adjustments. This is especially valuable in parametric spline fitting, where fine-tuning the curve shape and knot configuration often requires iterative, visual feedback. In this section, we will take a look at how these capabilities are implemented in SplineCloud and how the interactive curve fitting approach can be complementary to the programmatic data processing workflows.

5.1. Data Preparation

According to the workflow presented in Figure 2, data has to be uploaded to the existing or new repository. It can be a text file, a spreadsheet, or an image containing a plot. In a case of text data, a default dataset will be created automatically and a subset of data can be identified by adjusting data loading options. Datasets will be created automatically for each sheet in the spreadsheet source file. An interactive plot digitizer tool will be displayed for the image file to help extract data ranges from plots (Figure 4).



	V (mph)	J (Adv_Ratio)	Pe	Ct	Cp	PWR (Hp)	Torque (In-Lbf)	Thrust (Lbf)	PWR (W)
0	0.00	0.0000	0.0000	0.1276	0.0612	4.576	24.035	22.492	3412.385
1	6.47	0.0407	0.0825	0.1273	0.0628	4.696	24.665	22.450	3501.790
2	12.95	0.0814	0.1602	0.1271	0.0646	4.830	25.368	22.405	3601.532
3	19.42	0.1221	0.2327	0.1268	0.0665	4.977	26.140	22.354	3711.189
4	25.90	0.1628	0.2998	0.1265	0.0687	5.136	26.976	22.293	3829.840

Remove header ☐
Column separator: comma
Start row: 0
last

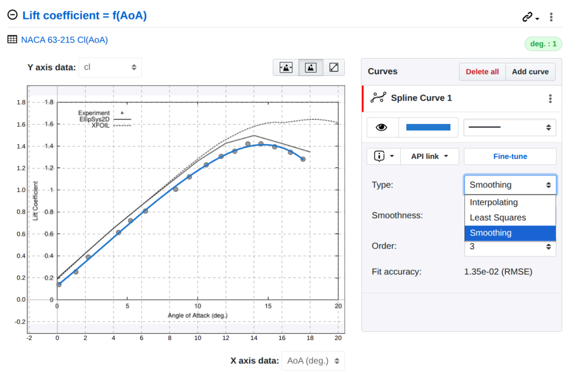
	delta_y	dC
0	-0.002200	0.106348
1	0.243523	0.144979
2	0.742729	0.222736
3	1.073932	0.274675
4	1.330579	0.319108
5	1.686026	0.380113
6	1.956106	0.431936
7	2.126691	0.461560
8	2.260721	0.470566
9	2.372413	0.465589
10	2.486136	0.444497

Figure 4. *SplineCloud datasets extracted from the tabular data file and plots*

After identifying clean subsets, a default Data Relation object will be created after entering a curve fitting mode (either by clicking the Fit Curve button or going into the Relations tab).

5.2. Automatic Spline Fitting

By adding a first curve, an initial spline fit is generated using a smoothing splines method (Figure 5).



Type: Smoothing

Smoothness:

Order: 3

Fit accuracy: 1.35e-02 (RMSE)

Figure 5. *Spline fitting tool*

SplineCloud's smoothness parameter is a relative parameter used to calculate SciPy's (FITPACK's) smoothing factor s defined as:

$$\sum_{i=1}^{n} w_i (y_i - S(x_i))^2 \leq s \tag{8}$$

The transition from the relative smoothing parameter to SciPy's s-factor is implemented in three steps:

1. building the least squares fit with the minimal possible number of knots (zero internal knots);
2. calculating the actual smoothing factor s_{max} for this fit using formula (8);
3. multiplying the relative smoothing parameter by this value: $s_{\text{scipy}} = s_{\text{max}} \cdot s_{\text{rel}}$

This approach improves user experience - instead of guessing each time the correct absolute value (which depends on the scale of data points), it is more intuitive to use relative values. By selecting several values for one data range the developed feedback instructs the more appropriate values for another curve of a different scale.

Least Squares fitting is implemented as an alternative to smoothing splines and is built on top of SciPy's `LSQUnivariateSpline` class. For simplicity, a uniform knot vector is constructed and passed to the class constructor using the number of internal knots from the user input. However, there is an option to adjust the knot vector interactively and use least squares fitting for the given non-uniform knot vector. This capability is implemented in the fine-tuning mode (see Section 4).

SplineCloud also supports interpolating splines, implemented via hardcoded $s = 0$ in `UnivariateSpline` class constructor.

5.3. *Fine-Tuning. Interactive Adjustments of Control Points and Knot Vector*

A principal enhancement over the automatic fitting approach is SplineCloud's Fine-Tuning functionality. It provides the ability to visually adjust control points and knot vectors of fitted splines. As proved in many cases, this interface enables users to achieve curve refinements that exceed the capabilities of SciPy's automatic fitting algorithms in terms of smoothness and accuracy. Moreover, the visual interface to knot vectors provides control over curve continuity in the specific regions. This helps in modelling complex data behavior with steep changes in main trends.

In SplineCloud, all spline curves are represented as parametric B-splines or NURBS. As mentioned in Some Theoretical Background Behind Splines, a parametric B-spline curve is defined as a vector-valued function (2). This function, however, can be decomposed into its scalar components:

$$S^{(x)}(t) = \sum_{i=0}^{n} c_i^{(x)} B_{i,k}(t)$$
$$S^{(y)}(t) = \sum_{i=0}^{n} c_i^{(y)} B_{i,k}(t) \tag{9}$$

In simpler words, parametric spline curves can be defined by two distinct univariate spline functions, $x(t)$ and $y(t)$, sharing a common knot vector (Figure 6). This representation allows for the modeling of complex geometries, including loops, sharp transitions, and high-curvature regions.

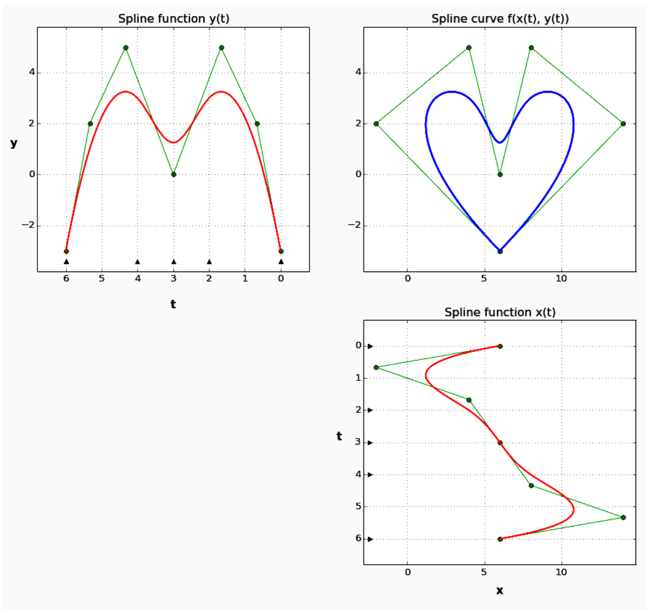


Figure 6. *Parametric spline curve as a combination of two spline functions.* Black triangles along parameter axes represent knots. Control polygons and control points are displayed in green color

5.3.1. *Interactive Editing of Control Points*

The beauty of parametric splines is in the ability to have an intuitive control over the curve shape by modifying the control polygon: moving, adding, and removing control points, increasing or decreasing their weights. This is impossible for spline functions, since they are defined through scalar coefficients, not vectors.

However, it is important to mention that for parametric spline curves coordinates of the control points $P_i = [x_i, y_i]$ are mathematically equivalent to the spline coefficients for the component functions $S^{(x)}(t)$ and $S^{(y)}(t)$, so that:

$$\boldsymbol{C}(t) = \sum_{i=0}^{n} \boldsymbol{P}_i B_{i,k}(t), \quad \text{where } \boldsymbol{P}_i = \begin{bmatrix} c_i^{(x)} \\ c_i^{(y)} \end{bmatrix} \tag{10}$$

Due to the local support property of B-spline basis functions, moving a single control point influences the shape of the curve only within a limited range of the parameter domain, providing localized editing capabilities.

In the Fine-Tune mode, users can interactively drag control points to reshape the curve (Figure 7). This mirrors the behavior of spline modeling in professional CAD environments, where designers sculpt geometry directly. The influence of each control point depends not only on its position but also on the associated basis function and, optionally, its weight.

Assigning higher weights to a control point increases its pull on the curve, bringing it closer to the point's location, converting the B-Spline to NURBS.

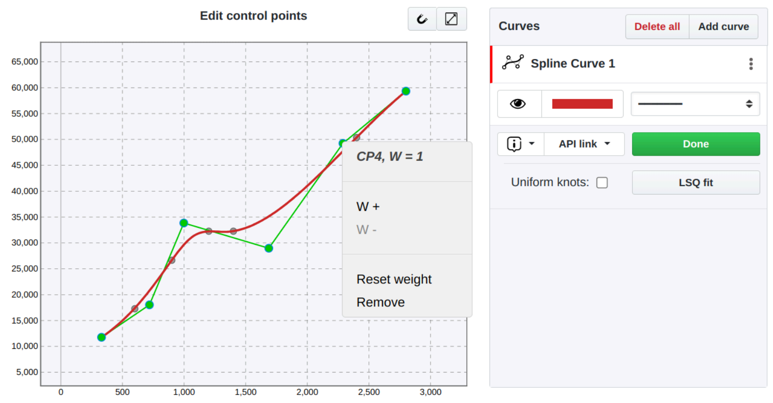


Figure 7. *Adjusting control points of the spline curve in the Fine-Tune mode*

This form of user interaction provides precise control over the spline's shape and is particularly advantageous when fitting complex datasets where automated routines produce unsatisfactory results.

5.3.2. *Dynamic Knot Vector Adjustment*

The knot vector plays a central role in determining the structure and properties of the resulting spline curve. Given a spline of degree k, the knot vector $\{t_0, t_1, ..., t_m\}$ is a non-decreasing sequence of real numbers, typically ranging from 0 to 1 (but not necessarily). Each interval $[t_i, t_{i+k+1})$ corresponds to a region over which a particular B-spline basis function $B_{i,k}(t)$ has support, meaning that each basis function is non-zero over at most $k + 1$ knot spans. The location and multiplicity of knots affect several critical properties of the resulting spline:

- **Continuity**. The number of continuous derivatives at a knot t_i is $k - m_i$, where m_i is the multiplicity of that knot. Repeated knots reduce the smoothness of the spline at that knot location. Specifically, if a knot has multiplicity m, then the continuity of the spline at that knot is reduced to C^{k-m}, where k is the degree of the spline. That is, the spline remains $(k - m)$-times continuously differentiable, and all higher derivatives are discontinuous.
- **Flexibility**. Adding more knots increases the local adaptability of the spline, allowing it to better follow variations in the data. In the B-spline formulation, the number of basis functions, and therefore the number of control points n is always $T - k - 1$, where k is the spline degree, T is the number of knots. As more interior knots are introduced, the number of basis functions increases, providing additional degrees of freedom for shaping the curve while maintaining the required continuity.

In traditional fitting methods such as `LSQUnivariateSpline` in SciPy, the knot vector must be either provided manually or generated heuristically, which typically requires some kind of iterative approach in finding an optimal (usually quasi-optimal) knot vector. SplineCloud removes this complexity by exposing the knot vector as an editable structure in the Fine-Tune mode, where users can manipulate knots directly and observe their impact on the spline in real time (Figure 8).

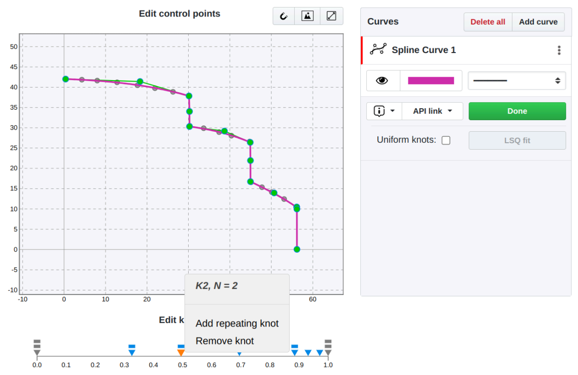


Figure 8. *Adjusting the knot vector of the spline curve in the Fine-Tune mode*

In particular, SplineCloud enables users to insert and remove knots, relocate internal knots, add and remove duplicate knots to control continuity and introduce geometric features such as cusps, kinks, or plateaus.

Uniformly spaced knot vectors typically produce curves with more predictable and symmetric behavior, which is easier to interpret visually and adjust interactively. In contrast, non-uniform vectors can help in fitting local irregularities with higher precision.

5.4. *Fitting Errors*

Comparison of the different fitted models requires a well-defined error metric that quantifies the discrepancy between the predicted and actual data. Out of many different metrics it is worth to mention the most common: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\ y_i - S(x_i)\ | \tag{11}$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - S(x_i))^2 \tag{12}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - S(x_i))^2} \tag{13}$$

where y_i are the actual data values, $S(x_i)$ are the corresponding predicted values from the curve, n is the total number of data points.

5.4.1. *Estimating Fitting Errors in SciPy*

In SciPy's spline fitting, smoothing splines support weighted data points, with fitting errors estimated using the corresponding weighted residuals. If weights w_i are provided (or implicitly set to 1), the weighted squared residuals are computed as:

$$R_i = (w_i(y_i - S(x_i)))^2 \tag{14}$$

The smoothing condition is formulated as:

$$\sum_{i=1}^{n} R_i \leq s \tag{15}$$

where s is the smoothing factor. This formulation is used internally by functions such as `UnivariateSpline` and `splrep`, where the goal is to balance curve fidelity with smoothness.

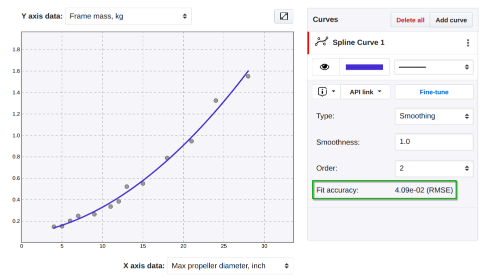
The `UnivariateSpline` class includes a method called `get_residual()`, which returns the weighted sum of squared residuals used in the spline construction process. Here is an example of estimating the fitting error using the built-in `get_residual()` and custom estimator function:

```python
import numpy as np
from scipy.interpolate import UnivariateSpline

# Generate synthetic data
x = np.linspace(0, 10, 100)
y = np.sin(x) + 0.2 * np.random.randn(100)

# Fit smoothing spline
spline = UnivariateSpline(x, y, s=5.0)

# Evaluate spline at input points
y_fit = spline(x)

# RMSE using SciPy's get_residual()
residual = spline.get_residual()
rmse_scipy = np.sqrt(residual / len(x))

# RMSE computed manually
rmse_manual = np.sqrt(np.mean((y - y_fit)**2))

print(f"RMSE (SciPy):  {rmse_scipy:.5f}")
print(f"RMSE (Manual): {rmse_manual:.5f}")
```

```
RMSE (SciPy):  0.22361
RMSE (Manual): 0.22361
```

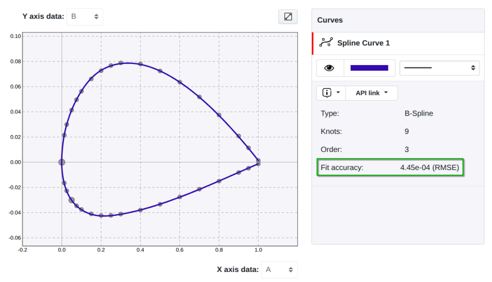
5.4.2. *Estimating Fitting Errors in SplineCloud*

In SplineCloud, the RMSE is calculated automatically for all curves. The `Fit accuracy` hint updates after each curve modification (Figure 9 a, b). This allows tracking the change of fitting error while applying different fitting parameters, comparing different models and fine-tuned curves against automatically fitted models.



Type: Smoothing
Smoothness: 1.0
Order: 2
Fit accuracy: 4.09e-02 (RMSE)

(a) **RMSE estimation in SplineCloud.**



Type: B-Spline
Knots: 9
Order: 3
Fit accuracy: 4.45e-04 (RMSE)

(b) **RMSE evaluated via shortest distance residuals.**

Figure 9. *Simple interpolaing cubic spline.*

For complex curve shapes, especially those with closed loops (Figure 9 b), the error between a data point and the curve should not be measured only along the vertical axis. Instead, the true discrepancy is the Euclidean distance from each data point to the nearest point on the curve. This approach is implemented in SplineCloud to estimate fitting errors of parametric curves that approximate data in non-ascending order via $RMSE_{sd}$ - root mean squared shortest distance error:

$$\mathrm{RMSE}_{sd} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(w_i d_i)^2} \tag{16}$$

where d_i is the shortest distance between a data point and the spline curve. Evaluation of this distance is implemented via SciPy's `optimize` module, and particularly a direct method is used as one of the most stable and fastest global optimization algorithms (based on the in-house analyses).

6. Reusability and Reproducibility with SplineCloud

Reusability and reproducibility are closely related concepts in scientific research, both essential for ensuring that data-driven results can be reliably verified, extended, and applied across different contexts and by independent researchers.

Reusability refers to the capacity of data, models, or computational results to be used beyond their original context, either by the same researcher at a later time or by others pursuing related work.

Reproducibility denotes the ability of independent researchers to replicate the results of a study using the same input data, methods, and conditions. In computational science, this typically involves the complete transparency of the modeling pipeline, including data preprocessing, parameter tuning, and evaluation metrics.

In conventional curve fitting workflows implemented via a programmable interface, fitted models are often tightly coupled with the original data and code used to generate them. While the source data and fitting scripts may be published, the resulting models themselves are rarely stored or shared as independently reusable objects. As a result, anyone wishing to replicate or build upon a previous curve fitting problem must re-execute the entire fitting process, including selecting the method, tuning parameters, and validating the fit.

This leads to the following limitations:

- **Redundant effort**. Researchers across teams or domains often repeat fitting procedures for the same datasets.
- **Poor reproducibility**. Deviations in fitting parameters and algorithms across various programming environments may cause discrepancies in reproduced models and, as a consequence, discrepancies in modelling results.
- **Poor interoperability**. There is no widely adopted convention for serializing and sharing regression models across tools and environments. As a result, transferring models typically requires manual extraction of fitting parameters or curve construction data (like spline parameters). This process is error-prone and demands significant additional effort to reuse models accurately in code.

By extending the principles of FAIR data to regression models, it becomes clear that it is not sufficient to publish raw data and code alone - intermediate or final research outputs such as fitted models (or response surface models) should also be accessible in standardized, referenceable form.

6.1. Improving Reusability and Reproducibility of Spline Models

SplineCloud addresses these challenges by decoupling the spline model from the fitting code and treating it as a first-class, shareable object. The platform is open for open data and supports persistent storage of fitted curves as structured entities that include:

- Unique IDs and web links;
- API links that return spline data and related datasets and subsets;
- A metadata layer, including authorship, tags, units, and other associated context (improvements in progress).

Once users create spline models interactively, they become instantly accessible to the broad public, including anonymous users. Reusability and reproducibility are ensured by both the open REST API and client libraries.

An official Python client, `splinecloud-scipy`, is based on SciPy and allows for fetching spline data using the spline UIDs and recreating the model in code. Reusable spline model has critical methods for evaluating spline in the form $y = f(x)$, loading underlying data, and assessing fit accuracy (using one of the methods, listed in Section 5). This enables a new level of interoperability: a spline curve fitted by one researcher can be discovered on the platform, imported and evaluated in another researcher's codebase, without downloading and importing the original data and refitting it.

Let's first take a look at the client library basic usage scenarios and then discuss the library structure and how splines are recreated and evaluated.

6.1.1. *Python client*

`splinecloud-scipy` is a library that can be installed from the Python Package Index:

```
pip install splinecloud-scipy
```

As of now, the library provides two main functions: `load_spline()` and `load_subset()`, which should be used to fetch spline models and underlying subsets:

```
from splinecloud_scipy import load_spline
spline = load_spline(<curve_uid>)
```

The curve UID can be taken from SplineCloud - an API link dropdown on the Curves toolbox (Figure 10 a). `load_spline()` returns an instance of the `ParametricUnivariateSpline` class, which allows for evaluating the spline as a function of x.

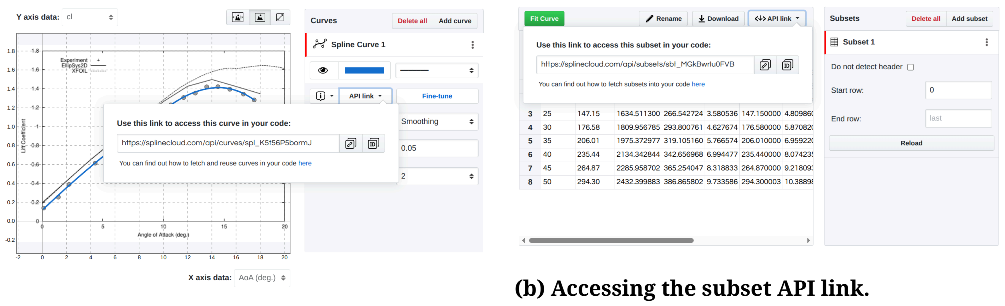
```
import numpy as np
X = np.linspace(0, 20, 100)
Y = spline.eval(X, extrapolate=True)
```

The spline object allows for loading underlying data:

```
columns, table = spline.load_data()
```

The same result can be achieved by explicitly loading a corresponding subset via its UID. Similarly to the spline curve, the subset UID can be taken from the SplineCloud API drop-downs (Figure 10 b). Both methods return a tuple with a list of column names and a NumPy array with subset data.

```
from splinecloud_scipy import load_subset
columns, table = load_subset(<subset_uid>)
```



Use this link to access this curve in your code:

https://splinecloud.com/api/curves/spl_K5t56P5bormJ

You can find out how to fetch and reuse curves in your code here

Smoothing

0.05

2

Use this link to access this subset in your code:

https://splinecloud.com/api/subsets/sbt_MGkBwrlu0FVB

You can find out how to fetch subsets into your code here

3	25	147.15	1634.511300	266.542724	3.580536	147.150000	4.80986
4	30	176.58	1809.956785	293.800761	4.627674	176.580000	5.87082
5	35	206.01	1975.372977	319.105160	5.766574	206.010000	6.95922
6	40	235.44	2134.342844	342.656968	6.994477	235.440000	8.07423
7	45	264.87	2285.958702	365.254047	8.318833	264.870000	9.21809
8	50	294.30	2432.399883	386.865802	9.733586	294.300003	10.3889

Do not detect header ☐

Start row: 0

End row: last

(a) Accessing the curve API link.

(b) Accessing the subset API link.

Figure 10. *Accessing the curve and subset API links.*

More information on the library usage can be found in the library code repository README section.

The `splinecloud-scipy` library is built around two core classes: `ParametricUnivariateSpline` - for the construction of the parametric splines from the curve data (degree, control points, and knot vector) and `PPolyInvertible` - a helper class defined to enable solving the spline curve as a function of x values. Both classes extend SciPy's classes, namely: `interpolate.UnivariateSpline` and `interpolate.PPoly`.

`ParametricUnivariateSpline` is the main class of the `splinecloud-scipy` client library. Its instance is returned by the `load_spline()` function. It defines a 2D parametric spline curve based on two univariate spline functions — one for the $x(t)$ and one for the $y(t)$ dependencies, that share a common knot vector and spline degree.

The `ParametricUnivariateSpline` object is callable - it takes a parameter value and returns the corresponding x and y values by calling internal objects `self.spline_x()` and `self.spline_y()` - instances of SciPy's `UnivariateSpline` class. This part, however, requires refactoring, since it uses private methods to construct `UnivariateSpline` from the knot vector, coefficients, and degree. There is a plan to use the `BSpline` class instead of `UnivariateSpline` here:

```python
self.spline_x = si.UnivariateSpline._from_tck(tck_x)
self.spline_y = si.UnivariateSpline._from_tck(tck_y)
```

Evaluation of the spline in the form $y(x)$ is implemented in the `eval()` method. This method creates a piecewise polynomial representation of spline functions x(t) and y(t) using the custom `PPolyInvertible` class. The methods of this helper class allow for finding the interval containing the *x-value*, then the polynomial is solved for the *t-value*. This parameter value is then used in the spline function $y(t)$ to find the desired *y-value*.

Polynomial representations are constructed inside the private method `_build_ppolyrep()`, called on the `ParametricUnivariateSpline` initialization.

```python
def _build_ppolyrep(self):
    self.spline_x.ppoly = PPolyInvertible.from_splinefunc(self.spline_x, extrapolate=True)
    self.spline_y.ppoly = PPolyInvertible.from_splinefunc(self.spline_y, extrapolate=True)
```

The `ParametricUnivariateSpline` class also provides a method for evaluating fit accuracy using one of the three methods discussed in Section 5.

```
RMSE = spline.fit_accuracy(table, method="RMSE")
```

The client library introduces basic capabilities that allow for fetching, recreating, and evaluating spline models in code. Its structure, however, is a bit complicated: conversion to polynomial representation may be omitted in future versions, and other approaches may be used to evaluate the inverse relation $t(x)$ or solve the relation $y(x)$ directly. Contribution for this and other improvements is welcome.

Nevertheless, the implemented approach is covered with tests, supports extrapolation, and shows good performance (no reportable benchmarking was performed).

6.2. *Implications and Limitations of SplineCloud*

The interactive, collaborative and reusable curve fitting enabled by SplineCloud minimizes redundant work and enhances the reliability and continuity of model-based research outputs. However, the platform currently does not support other classes of regression models such as polynomial fits, rational functions, exponential models, or machine learning-based regressors. This limitation is not technical in nature but a deliberate, temporary design decision, rooted in both theoretical and practical considerations.

Spline models, particularly B-splines and their parametric forms, possess a standardized and well-defined mathematical representation. This makes them ideally suited for platform-independent storage, manipulation, and code-level reuse. In contrast, arbitrary analytical or statistical regression models lack the generalization inherent to splines and the support of all possible modeling techniques is an overwhelming challenge, at least for the current stage of the project.

From a technical point of view, spline-based models are especially effective for representing empirical data derived from physical experiments, simulations, and measurements — scenarios where an accurate and differentiable fit is often more valuable than an interpretable analytical expression. SplineCloud is thus particularly suited for constructing surrogate models, processing and interpolating results of numerical simulations (like wind tunnel tests), and reusing such models in downstream analysis, control, or optimization tasks.

However, users requiring analytical model validation (e.g., fitting custom functions to verify physical laws or derive closed-form expressions) will find SplineCloud insufficient for their needs. In such cases, traditional tools, including SciPy, remain more appropriate.

Despite this limitation, SplineCloud introduces the novel capability of building shareable repositories of empirical relations. The catalogs of such relations can serve as reference data and model sources in experimental domains such as fluid dynamics, structural mechanics, or thermodynamics, where precomputed curves and data-driven models are frequently reused. For example, aerodynamic design processes often rely on aggregated wind tunnel results; SplineCloud provides an infrastructure to formalize, store, and exchange such data in a reproducible and programmatically accessible way.

7. Use Cases and Applications

Among the many possible applications of SplineCloud, three main types of scenarios are especially worth highlighting:

1. situations where interactive curve fitting improves accuracy and/or reduces time spent on analysis;
2. cases where making modeling results accessible and reusable enhances research reproducibility;

3. workflows where empirical data is structured as a library of functions for use in automation.

While the examples presented here are related to engineering, the described patterns and approaches can be applied in other domains.

7.1. Modeling Step-drawdown Water Pumping From Boreholes

Step-drawdown tests are performed to determine the optimal pumping rate for water boreholes by measuring how water levels respond to progressively increasing discharge rates. Traditional analysis methods assume that relationships between pumping rates and water level drop should be predictable and follow simple mathematical models, but experimental data often reveals complex non-linear behaviors that standard equations cannot capture. When conventional models fail to fit the observed data, researchers tend to develop more sophisticated mathematical frameworks.

G. Van Tonder, J. Botha, and J. Van Bosch [8] developed a model for predicting drawdown behavior in step-drawdown tests based on experimental data from South Africa's Karoo formations. This model can be useful in determining sustainable pumping rates for boreholes. While relying on physical properties of fractured rock aquifers, flow dimensions, and elasticity parameters, the model still exhibits underfitting for some pumping rates. Spline fitting can produce significantly better accuracy and does not require a complex analytical background if practical use is the purpose.

$$s(t) = AQ + B'Q^e(\log t)^{\Gamma(n/2)} + C'Q^p\log(t)$$

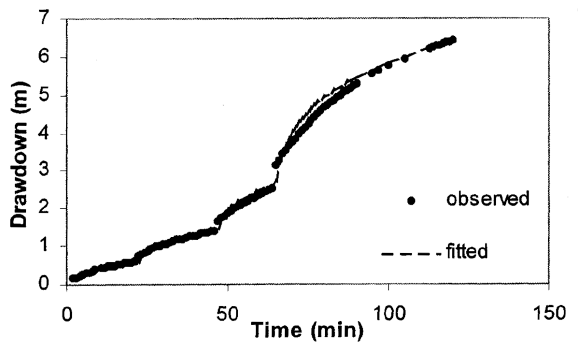
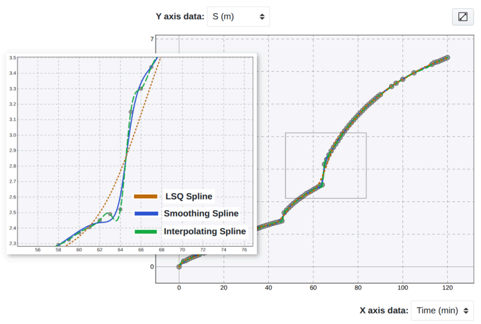


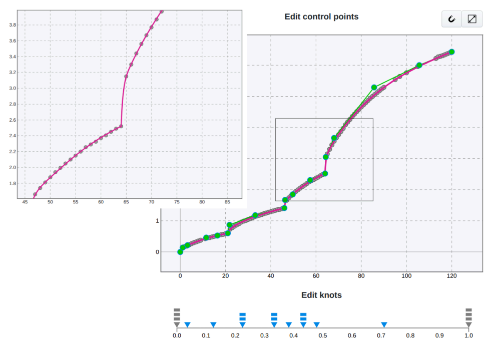
Figure 11. *empirical model fitted to experimental data on the step-drawdown test performed on Borehole UP16 [8]*

Nevertheless, in the case of relations with clearly expressed step changes, fitting becomes a challenge, even for splines. All automatic spline fitting models exhibit overfitting or underfitting (Figure 11). This is where SplineCloud's Fine-Tuning mode can be helpful. By introducing breaks in continuity in the mode change points, and manipulating control points and knot vector, it is possible to manually fit a curve[2] to the complex stepped data (Figure 12).

[2]Link to curves fitted to the discharge rates from the test data from [8]



(a) Auto fitted spline models.

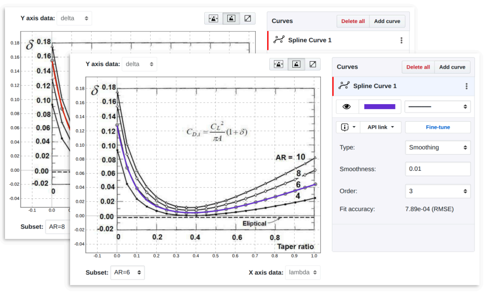


(b) Fine-tuned spline model.

Figure 12. *Spline models fitted to experimental data on the step-drawdown test data.*

7.2. Reconstructing Empirical Relations for Wing Span Efficiency Factor

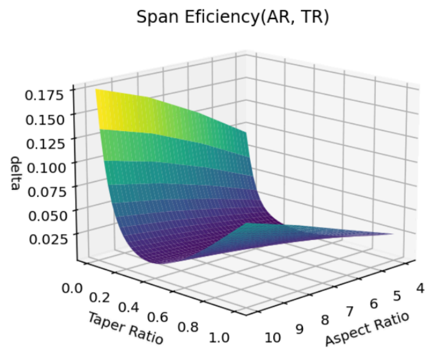
When designing an aerial vehicle, preliminary estimates of wing lift and drag forces are essential. These calculations are often based on empirical data obtained from wind tunnel experiments or numerical simulations. Such data is typically processed and generalized across different combinations of relative design parameters. One example is the estimation of the induced drag coefficient for tapered wings. Amongst various approaches [9], empirical relationships for the span efficiency parameter δ, used in induced drag calculations, were proposed by B. W. McCormick [10] in graphical form.



Type: Smoothing
Smoothness: 0.01
Order: 3
Fit accuracy: 7.89e-04 (RMSE)

(a) Auto fitted spline models.

```
delta_curves = {
    "AR=10": load_spline("spl_Nb5m075aHe42"),
    "AR=8":  load_spline("spl_YUtYmXHXXK5Q"),
    "AR=6":  load_spline("spl_VsE31rlqsGtH"),
    "AR=4":  load_spline("spl_lvdZRtwoGU6T")
}
```



(b) Fine-tuned spline model.

Figure 13. *Spline models fitted to empirical relations of the span efficiency parameter δ.*

Using SplineCloud, data from such plots can be digitized and spline models created — see Figure 13 (a). These models[3] can then be reused programmatically to construct a response surface model using linear interpolation between curves (see Figure 13 b). This enables direct evaluation of the δ parameter and ultimately the induced drag coefficient for different wing planforms. The resulting models are shareable and reusable, enabling other researchers to avoid repeating the same steps and improving reproducibility.

[3] Data relations constructed using plot for efficiency factor [10]

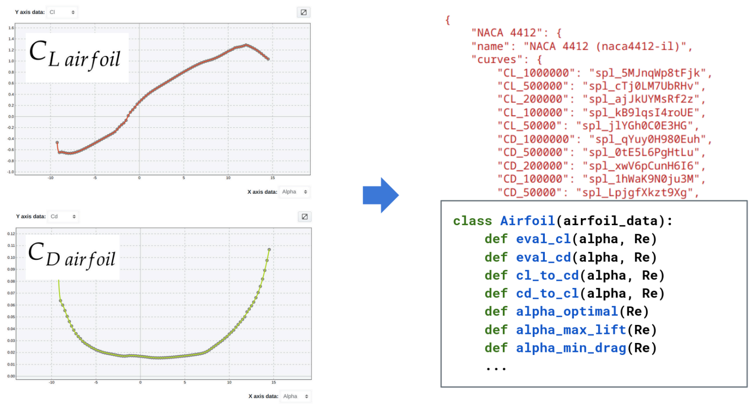
7.3. *Building Reusable Libraries of Airfoils' Performance Curves*

Airfoil performance data are fundamental for aerodynamic analyses in a wide range of aerospace applications, including wind turbines, unmanned aerial vehicles, and civil and transport aircraft. Usually, an objective in such analyses is the selection of airfoils appropriate to specific operational conditions. This task typically involves evaluating large sets of airfoils and their associated performance data, obtained either through wind tunnel experiments or numerical simulations. Resources like Airfoil Tools provide access to the results of simulations for a huge library of airfoils, and PDFs with airfoil performance data can be found in many university libraries: e.g. DTU library published Wind Turbine Airfoil Catalogue [11] that contains results of wind tunnel tests and numerical analyses for a collection of airfoils used in wind turbines.

A common limitation of these resources is their lack of programmatic accessibility and reusability. Airfoil Tools does not expose its datasets via an API, and extracting numerical values from plots in PDF documents remains a manual and time-consuming operation. Despite this, processing, fitting, and analysis of airfoil performance data are routine tasks in aerospace engineering and are frequently duplicated across research groups.

The recent approaches based on neural networks, aiming to predict aerodynamic coefficients directly from airfoil geometry, are bypassing the need for raw experimental or simulation data. For example, the AirfoilNet model [12] uses image-based representations for end-to-end prediction of lift and drag. While effective for automated prediction, such models emphasize performance over physical interpretability and reusability.

Alternatively, SplineCloud can be used to create and host airfoil performance curves fitted with splines, offering tracable, interpretabile and reusable approach. The collection of curve IDs from such repositories can be shared publicly to simplify access and reuse in custom scripts and optimization algorithms Figure 14.



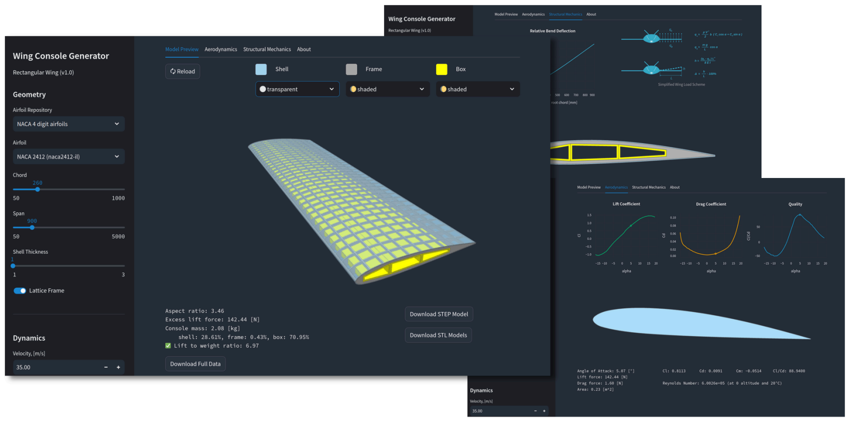
```
{
    "NACA 4412": {
    "name": "NACA 4412 (naca4412-il)",
    "curves": {
        "CL_1000000": "spl_5MJnqWp8tFjk",
        "CL_500000": "spl_cTj0LM7UbRHv",
        "CL_200000": "spl_ajJkUYMsRf2z",
        "CL_100000": "spl_kB9lqsI4roUE",
        "CL_50000": "spl_jlYGh0C0E3HG",
        "CD_1000000": "spl_qYuy0H980Euh",
        "CD_500000": "spl_0tE5L6PgHtLu",
        "CD_200000": "spl_xwV6pCunH6I6",
        "CD_100000": "spl_1hWaK9N0ju3M",
        "CD_50000": "spl_LpjgfXkzt9Xg",
```

```
class Airfoil(airfoil_data):
    def eval_cl(alpha, Re)
    def eval_cd(alpha, Re)
    def cl_to_cd(alpha, Re)
    def cd_to_cl(alpha, Re)
    def alpha_optimal(Re)
    def alpha_max_lift(Re)
    def alpha_min_drag(Re)
    ...
```

Figure 14. *Accessing and reusing airfoil performance curves in Python*

Several airfoil repositories are already available on the platform[4] and are actively used in applications such as the Wing generator web application developed for demonstration purposes Figure 15 . Upon selecting an airfoil, the application loads lift and drag curves, along with profile geometry from SplineCloud. These data are then used to compute aerodynamic loads, wing bending, and generate a 3D model of the wing using the CQ-UAV library.

[4]SplineCloud repositories containing tag 'airfoils'



Airfoil Repository: NACA 4 digit airfoils
Airfoil: NACA 2412 (naca2412-il)
Chord: 260 (50–1000)
Span: 900 (50–5000)
Shell Thickness: 1 (1–3)
Lattice Frame: on
Shell: transparent; Frame: shaded; Box: shaded
Velocity, [m/s]: 35.00

```
Aspect ratio: 3.46
Excess lift force: 142.44 [N]
Console mass: 2.08 [kg]
    shell: 28.61%, frame: 0.43%, box: 70.95%
✓ Lift to weight ratio: 6.97
```

```
Angle of Attack: 5.07 [°]          Cl: 0.8113     Cd: 0.0091     Cm: -0.0514     Cl/Cd: 88.9400
Lift force: 142.44 [N]
Drag force: 1.60 [N]               Reynolds Number: 6.0026e+05 (at 0 altitude and 20°C)
Area: 0.23 [m^2]
```

Figure 15. *Wing console generator web application that reuses airfoil performance curves from SplineCloud repositories*

8. Summary and Future Directions

This paper presented a collaborative and transparent approach to curve fitting, enabled by SplineCloud — an open platform that extends SciPy's interpolate module with interactive spline modeling and reproducibility features.

The paper contains an overview of the spline fitting approaches implemented in SciPy and SplineCloud, illustrating the practical utility of spline-based regression for addressing complex real-world problems across diverse scientific and engineering domains, especially in problems where flexible and accurate curve fitting is essential for extracting meaningful relationships from experimental and simulation data.

As described in Interactive Spline Fitting Workflow of this paper, SplineCloud extends the capabilities of SciPy and addresses key limitations of traditional programmatic spline fitting by providing visual control over spline parameters, enabling fine-tuning through interactive manipulation of control points and knot vectors. The application of FAIR principles to regression modeling enhances model reusability and promotes reproducibility in data analysis workflows. By decoupling spline models from their original fitting code and providing programmatic access through the client libraries, the platform enables researchers to build shareable repositories of empirical relations, reducing redundant effort for data processing and enhancing the reliability of model-based research outputs.

This is proved by the use case studies, given in Use Cases and Applications, that illustrate the platform's effectiveness in addressing real-world challenges, including: complex data fitting from hydrogeological experiments; improving reusability of models based on historical data in aerodynamics; and the creation of digital libraries of empirical relations to support engineering analysis and design automation in aerospace applications.

Future development directions include expanding the platform's capabilities to support multivariate splines, API and client libraries improvements, and features required for collaboration, metadata structuring, provenance, traceability and transparency. SplineCloud's mission and its approach align with the growing emphasis on open science and reproducible research practices in computational science and engineering. As the platform matures, it has the potential to become widely adopted in collaborative data-driven modeling, enabling researchers to discover, validate, and build upon existing results, saving time and resources for creativity and innovation.

Acknowledgments

This work was supported by funding from the European Union under the Seeds of Bravery initiative, as part of the Horizon Europe program for supporting Ukrainian innovation. Prior support was provided by CRDF Global, whose contribution played an essential role in the early development of the platform.

The development of SplineCloud would not have been possible without the dedication and expertise of the team. The author extends sincere gratitude to Olena Onyshchenko, Maksym Valyn, Daria Prokaza, Taras Talimonchuk, and Alina Nehoda for their valuable contributions to engineering, design, and research efforts.

References

[1] M. D. Wilkinson *et al.*, “The FAIR Guiding Principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, 2016, doi: 10.1038/sdata.2016.18.

[2] M. G. COX, “The Numerical Evaluation of B-Splines,” *IMA Journal of Applied Mathematics*, vol. 10, no. 2, pp. 134–149, 1972, doi: 10.1093/imamat/10.2.134.

[3] C. de Boor, “On calculating with B-splines,” *Journal of Approximation Theory*, vol. 6, no. 1, pp. 50–62, 1972, doi: https://doi.org/10.1016/0021-9045(72)90080-9.

[4] T. Lyche and K. Mrken, *Spline Methods. Draft*. Department of Informatics, Centre of Mathematics for Applications, University of Oslo, [Online]. (last visited on 02/15/2018), 2008, p. 230. [Online]. Available: http://www.uio.no/studier/emner/matnat/ifi/INF-MAT5340/v07/undervisningsmateriale/

[5] F. Holmér, “The Continuity of Splines.” [Online]. Available: https://youtu.be/jvPPXbo87ds?si=NbPi7mdrAJJQCUvG

[6] E. Lee, “Choosing nodes in parametric curve interpolation,” *Computer-Aided Design*, vol. 21, no. 6, pp. 363–370, 1989, doi: https://doi.org/10.1016/0010-4485(89)90003-1.

[7] V. Pasko, “Approximation by Spline Functions and Parametric Spline Curves with SciPy.” [Online]. Available: http://vadym-pasko.com/blog/2015/03/06/spline-approx-scipy.html

[8] G. Van Tonder, J. Botha, and J. Van Bosch, “A generalised solution for step-drawdown tests including flow dimension and elasticity,” *Water SA*, vol. 27, no. 3, 2001, doi: 10.4314/wsa.v27i3.4978.

[9] D. S. M. Ni\textcommabelow ťa, “Estimating the Oswald Factor from Basic Aircraft Geometrical Parameters,” *Deutscher Luft- und Raumfahrtkongress 2012, Berlin*, p. 19, 2012, [Online]. Available: https://www.dglr.de/publikationen/2012/281424.pdf

[10] B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics*. John Wiley & Sons, 1979, p. 652. [Online]. Available: https://rexresearch1.com/FlightMechanicsLibrary/AerodynamicAeronauticsFlightMechanics.pdf

[11] F. Bertagnolio, *Wind turbine airfoil catalogue*. Denmark. Forskningscenter Risoe. Risoe-R No. 1280(EN): DTU Library, 2001, p. 151. [Online]. Available: https://backend.orbit.dtu.dk/ws/portalfiles/portal/7728949/ris_r_1280.pdf

[12] Z. Cai, Z. Fan, and T. Liu, “Efficient aerodynamic coefficients prediction with a long sequence neural network.” [Online]. Available: https://arxiv.org/abs/2403.14979