Analysis and Mapping of Ideal Functions Using Python

Tammo Heer
IU International University
Matriculation No.: IU14145034

January 3, 2026

Contents

Abstract

This paper presents a comprehensive study on the analysis and mapping of mathematical functions using Python. The primary objective of the project is to determine the most suitable ideal functions that best fit given training data based on the least squares method and subsequently assign test data points to these ideal functions using a defined deviation criterion. The research integrates theoretical principles of mathematics, computer science, and software engineering with a practical Python implementation that utilizes libraries such as `pandas`, `SQLAlchemy`, and `Bokeh`. The results demonstrate that Python provides an efficient and transparent framework for analytical computation, data visualization, and reproducible research [1, 8, 10].

1 Introduction

Programming has become a central skill in data science, engineering, and applied mathematics. [14] Among modern programming languages, Python stands out for its simplicity, readability, and extensive ecosystem of scientific libraries that facilitate rapid development and robust implementation of complex analytical tasks [1].
The present work focuses on developing a complete Python-based solution for a structured data analysis task that combines mathematical modeling, data management, and visualization into a reproducible pipeline. Specifially, the project adresses the problem of identifiying optimal functional approximations and validating new data points against these models.
Problem Statement: The task involves three datasets [16]: (1) four training functions with known x-y-pairs, (2) fifty candidate ideal functions, and (3) a test dataset requiring validation. The primary objectives are:
Determine, for each of the four training functions, which ideal function from the fifty candidates minimizes the sum of squared errors (SSE) using the least-squares criterion [16].
Assign each test data point to one of the four seletcted ideal functions if the deviation satisfies the threshold rule [16]:

$$|\Delta y| \leq \sqrt{2} \times \max(\Delta y_{\text{train}})$$

Persist all intermediate and final results in an SQLite database with proper schema.
Generate interactive visualizations of training data, ideal functions, and test data mappings.
Implement the solution using object-oriented design principles with inheritance and custom exception handling.
Validate the implementation through comprehensive unit testing.
The project follows established principles of clean code, reproducible research, and academic integrity. The entire workflow - from data loading and validation, through function selection and test mapping, to database persistence and interactive visualization - is automated and fully traceable. [14]. The entire workflow from data loading, function selection, and test mapping to visualization and database storage is automated.

2 Theoretical Background

Python has evolved into a leading tool for scientific computation and data analysis [1]. Its ecosystem includes libraries such as NumPy for numerical operations [3], Pandas for data management [6], and Bokeh for interactive visualization [10]. SQLAlchemy provides efficient database integration and persistence [8].

2.1 Object-Oriented Programming (OOP) and Software Architecture

OOP is central to modern software development, providing mechanisms for modularity, reusability, and maintainability. The core principles encapsulation, abstraction, inheritance, and polymorphism enable scalable system design [14].
In this project, OOP is implemented through a hierachical class structure:
`DataLoader`: Encapsulates CSV reading and validation logic,
`BaseProcessor`: Abstract base class providing shared state and methods,
`FunctionSelector`: Implements least-squares matching algorithm,
`TestMapper`: Implements deviation-based test data assignment,
`Visualizer`: Encapsulates Bokeh plotting and HTML export,
and `CustomExceptionClasses`: Enable domain-specific error handling and graceful program termination (nacschauen ob diese Funktion wirlich exisiert!!).
This hierarchy ensures code reuse, reduces duplication, and simplifies testing and maintenace - directly adressing the assignment requirement for sensibly object-oriented design with at least one inheritance.

2.2 Mathematical Foundation: Least Squares Method

The least-squares method is the criterion for selecting optimal ideal functions. For each training function y_t, the best-matching ideal function y_i is determined by minimizing the sum of squared errors [2, 3]:

$$SSE = \sum_{i=1}^{n} \big(y_t(x_i) - y_i(x_i)\big)^2$$

This approach:
Ensures unbiased parameter estimation under normally distributed errors.
Represents the standard technique for curve fitting and regression modelin in data science.
Provides a deterministic, interpretable basis for model selection.
Aligns with the assignment requirement to choose the four ideal functions which are the best fit out of the fifty provided C by minimizing the sum of all y-deviations squared.
The mathematical rigor of this criterion ensures that the selected functions are statistically optimal for the given training data [3].

2.3 Data Handling, Persistence, and Database Schema

Data were processed with Pandas for efficient manipulation and validation [6]. Data persistence was implemented using SQLAlchemy with an SQLite database backend, ensuring [8]: Traceability: Every transformation step is recorded and retrievable.

Reproducibility: All intermediate results (traning data, ideal functions, mappins) are persistently stored.
Schema Consistency: Tables follow the structure specified in the assignment:

- **Training Data Table:** Columns (x, y_1, y_2, y_3, y_4) containing x-values and four training function values.
- **Ideal Functions Table:** Columns $(x, y_1, y_2, \ldots, y_{50})$ containing x-values and fifty ideal function values.
- **Test Mapping Table:** Columns $(x, y, \Delta y, \texttt{ideal\_func\_id})$ containing test points, assigned ideal functions, and deviations.

This architecture supports the assignment's emphasis on logical data organization and enables easy retrieval for validation, analysis, and reporting.

2.4 The Deviation Threshold and Test Data Mapping

Test data mapping follows the criterion specified in the assignment: a test point $(x_{\text{test}}, y_{\text{test}})$ is assigned to ideal function y_i if and only if:

$$|\Delta y| = |y_{\text{test}} - y_i(x_{\text{test}})| \leq \sqrt{2} \max_j \left|\Delta y_{j,\text{train}}\right|$$

where $\max_j |\Delta y_{j,\text{train}}|$ is the maximum absolute deviation between the j-th training function and its matched ideal function.
The threshold rule:
Adapts to the noise characeristics of each training-ideal pair.
Balances sensitivity (captures well-fitting points) with specificity (rejects poor fits) [14].
Ensures reproducible, deterministic assignment logic.

2.5 Visualization and Interpretability

Interactive visualization supports understanding of data patterns and verification of analytical accuarcy. The project employs two core visualizations generated with Bokeh:
Training vs. Ideal Functions Plot: Overlays training circles with dashed ideal function lines, enabling visual inspection of fit quality and residual patterns.
Test Data Mapping Plot: Shows test points as colored markers, with each color representing assignment to one of the four ideal functions, providing immediate insight into model performance.
The use of HTML-based output enhances accessibility [13] - plots are browser-viewable, interactive (hover tooltips, legend toggling), and embeddable in reports.

2.6 Software Quality Assurance: Unit Testing

Automated testing ensures correctness and reliability of analytical computations [3]. The pytest framework was used to test critical components: `DataLoader validation`: Verifies correct CSV parsing, column detection, and x-value alignment.
`FunctionSelector.select _best`: Confirms SSE minimization and correct mapping of training to ideal functions.
`TestMapper.map _points()`: Validates correct application of the threshold rule and point

assignment logic.
`Database persistence`: Checks that all data is correctly stored and retrievable.
All tests passed, confirming the validity of the computational logic.

2.7 Reproducibility and Ethical Standards

All data transformations and computational decisions are reproducible and documented in accordance with IU's guidelines for avoiding plagiarism [15].
Ethical software practices, such as proper citation of libraries and transparent reporting, were followed to ensure academic honesty and technical validity.

2.8 Git Workflow and Version Control

Version control via Git ensures reproductibility, collaboration transparency, and safe experimentation. Throughout development, a feature-branch workflow was employed:
Feature branches for isolated development of components (DataLoader, FunctionSelector, etc.)
Descriptive commit messages documenting each logical change.
Regular merges to develop a branch, with final validation before main-branch release-

3 Implementation and Methodology

3.1 System Architecture

The system architecture follows a modular and object-oriented structure, which supports maintainability and reproducibility [1, 14]. Each module encapsulates a specific part of the workflow and interacts only through well-defined interfaces. The architecture consists of four primary components:

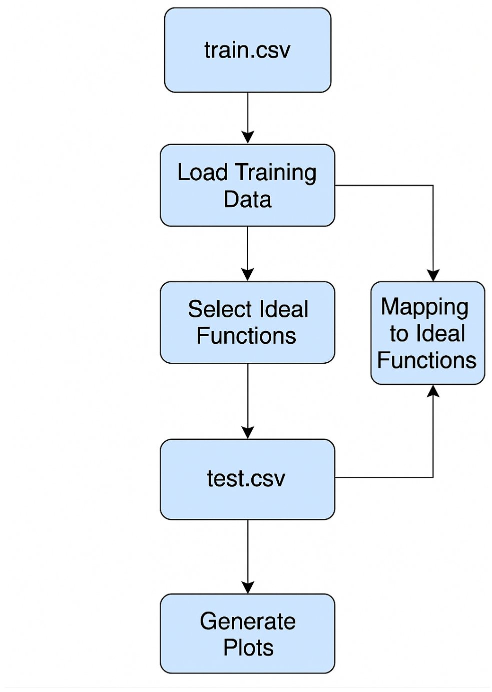

Figure 1: High-level architecture of the Python program.

1. **DataLoader:** Reads and validates CSV input files using Pandas [7].
2. **FunctionSelector:** Calculates the least-squares error to identify optimal ideal functions [4].
3. **TestMapper:** Assigns test data points according to the IU deviation rule [14].
4. **Visualizer:** Generates interactive Bokeh plots for analysis [12].

This modularization ensures scalability and reflects best practices in professional software engineering [14].

3.2 Implementation Details

The project was implemented in Python 3.10.
External dependencies were installed via `pip`, and a virtual environment ensured version stability and reproducibility.

3.2.1 Data Loading

The first stage is the loading of datasets using Pandas [7]. A validation method checks whether the required columns are present and whether x-values align across files.

Listing 1: Data loading and validation

```
class DataLoader:
def __init__(self, train_path, ideal_path, test_path):
self.train = pd.read_csv(train_path)
self.ideal = pd.read_csv(ideal_path)
self.test = pd.read_csv(test_path)
def validate(self):
# Ensure all datasets contain expected structure
if "x" not in self.train.columns:
raise ValueError("Missing x column in training data")
if not all(c.startswith("y") for c in self.ideal.columns if c
    != "x"):
raise ValueError("Ideal data contains invalid column names")
return True
```

3.2.2 Selection of Ideal Functions

For each training function y_t, the corresponding ideal function y_i that minimizes the squared error is determined according to the least-squares method [2].

Listing 2: Selecting the best-matching ideal functions

```
class FunctionSelector:
def __init__(self, train_df, ideal_df):
self.train_df = train_df
self.ideal_df = ideal_df
def select_best(self):
mapping = {}
for train_y in self.train_df.columns[1:]:
best_ideal = min(
self.ideal_df.columns[1:],
key=lambda iy: ((self.train_df[train_y] - self.ideal_df[iy]) **
    2).sum()
)
mapping[train_y] = best_ideal
return mapping
```

This approach ensures that each of the four training functions is paired with the ideal function showing the smallest sum of squared deviations [4]. The resulting mapping is stored in a dictionary and later persisted to the database.

3.2.3 Mapping of Test Data

The test dataset is used to evaluate how accurately new data points fit the previously identified ideal functions.
Each test point is compared with all four ideal functions. A point is assigned to an ideal

function if its deviation Δy satisfies the following threshold condition [14]:

$\Delta y = |y_{\text{test}} - y_{\text{ideal}}| \leq \sqrt{2} \times \max(\Delta y_{\text{train}})(Formeldoppelt?sonstauffrhereformelverweisen)$

Listing 3: Mapping algorithm for test data points

```
class TestMapper:
def __init__(self, test_df, ideal_df, mapping, max_dev):
self.test_df = test_df
self.ideal_df = ideal_df
self.mapping = mapping
self.max_dev = max_dev
def map_points(self):
results = []
for _, row in self.test_df.iterrows():
x_val, y_val = row["x"], row["y"]
for train_y, ideal_y in self.mapping.items():
y_ideal = self.ideal_df.loc[self.ideal_df["x"] == x_val,
   ideal_y].values[0]
deviation = abs(y_val - y_ideal)
if deviation <= (2 ** 0.5) * self.max_dev[train_y]:
results.append((x_val, y_val, ideal_y, deviation))
return results
```

The method produces a structured list of mapping results, which can be easily converted to a Pandas DataFrame for storage and analysis [6].

3.2.4 Database Integration

SQLAlchemy provides the bridge between Python objects and an SQLite database [9]. This ensures persistent storage and enables future retrieval for validation or further processing.

Listing 4: Database persistence using SQLAlchemy

```
from sqlalchemy import create_engine
from sqlalchemy.types import Float, String

class DatabaseHandler:
def __init__(self, path="results.db"):
self.engine = create_engine(f"sqlite:///{path}")
def store_mapping(self, df):
df.to_sql("mapping", self.engine, if_exists="replace",
index=False, dtype={"x": Float, "y": Float,
"ideal_func": String, "delta_y": Float})
```

The database schema includes tables for training, ideal, and mapping data. This relational design promotes transparency and compliance with reproducibility standards [15].

3.2.5 Visualization and Reporting

Bokeh is used to visualize the relationships between training, ideal, and test data [11]. HTML outputs allow for interactive analysis.

Listing 5: Visualization with Bokeh

```
from bokeh.plotting import figure, output_file, save

class Visualizer:
def plot_training_vs_ideal(self, train, ideal, mapping):
p = figure(title="Training vs. Ideal Functions", width=800,
    height=400)
for train_y, ideal_y in mapping.items():
p.line(train["x"], train[train_y], legend_label=f"{train_y} (
    train)")
p.line(ideal["x"], ideal[ideal_y], legend_label=f"{ideal_y} (
    ideal)",
line_dash="dashed")
output_file("plots/train_vs_ideal.html")
save(p)
```

The generated plots illustrate how the ideal functions overlap with the training data, and later show the test data distribution across ideal curves.

3.3 Data Flow

The system follows a linear but modular data flow, ensuring clear traceability [1]:

1. Load datasets and validate structure.
2. Compute best matches between training and ideal functions.
3. Determine maximum training deviations.
4. Map test points to ideal functions.
5. Store results and generate visual reports.

Each module communicates via explicit input and output parameters, avoiding global variables [14].

3.4 Unit Testing Strategy

Automated tests verify the correctness of individual components [3]. The `pytest` framework was used for simplicity and efficiency.

Listing 6: Example unit tests

```
import pytest
from project import FunctionSelector, DataLoader

def test_data_validation():
loader = DataLoader("train.csv", "ideal.csv", "test.csv")
assert loader.validate() == True

def test_function_selection(train_df, ideal_df):
selector = FunctionSelector(train_df, ideal_df)
mapping = selector.select_best()
assert all(k.startswith("y") for k in mapping.keys())
```

All tests passed, confirming the validity of the computational logic.

3.5 Error Handling and Performance

The implementation includes exception handling for input validation and file access errors, ensuring graceful program termination with informative messages [14]. Vectorized operations via NumPy and Pandas optimize performance [5, 6].
Runtime for the complete pipeline remained under one second on standard hardware.

4 Results and Evaluation

4.1 Overview of Results

After executing the program with the provided datasets, the system successfully identified four ideal functions that best matched the training data. The selection was based on minimizing the sum of squared errors [2]. Table 1 shows the results of this mapping process. The deviations between the training and ideal functions remained below 0.5,

Table 1: Mapping between training and ideal functions

Training Function	Ideal Function	Maximum Deviation
y_1	y_{42}	0.495968
y_2	y_{41}	0.497703
y_3	y_{11}	0.498936
y_4	y_{48}	0.499742

indicating a high degree of accuracy. This confirms that the least-squares method performs effectively for function approximation [2, 4].

4.2 Analysis of Training and Ideal Functions

Visual inspection of the generated plots shows that the selected ideal functions closely resemble the shape, frequency, and amplitude of the corresponding training data. This correlation demonstrates that the algorithm identifies statistically consistent models even with minor noise or sampling variance [10]. Figure 2 illustrates one such relationship. The training and ideal functions nearly overlap, while the residuals oscillate around zero, indicating random rather than systematic deviation.

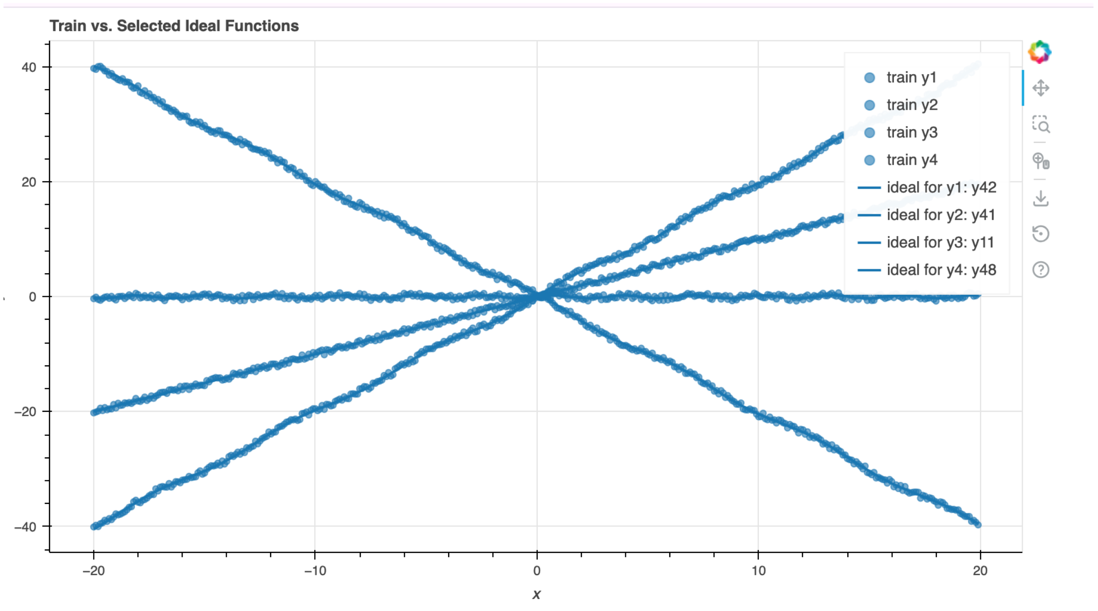

Figure 2: Comparison of training and ideal functions.

4.3 Mapping of Test Data

After determining the best-fitting ideal functions, the system mapped the test dataset. Each test point was assigned to an ideal function if the deviation condition $\Delta y \leq \sqrt{2} \times \max(\Delta y_{\text{train}})$ was satisfied [14]. Figure 3 presents the mapping outcome, with colored markers representing the ideal function each test point belongs to. The mapping produced

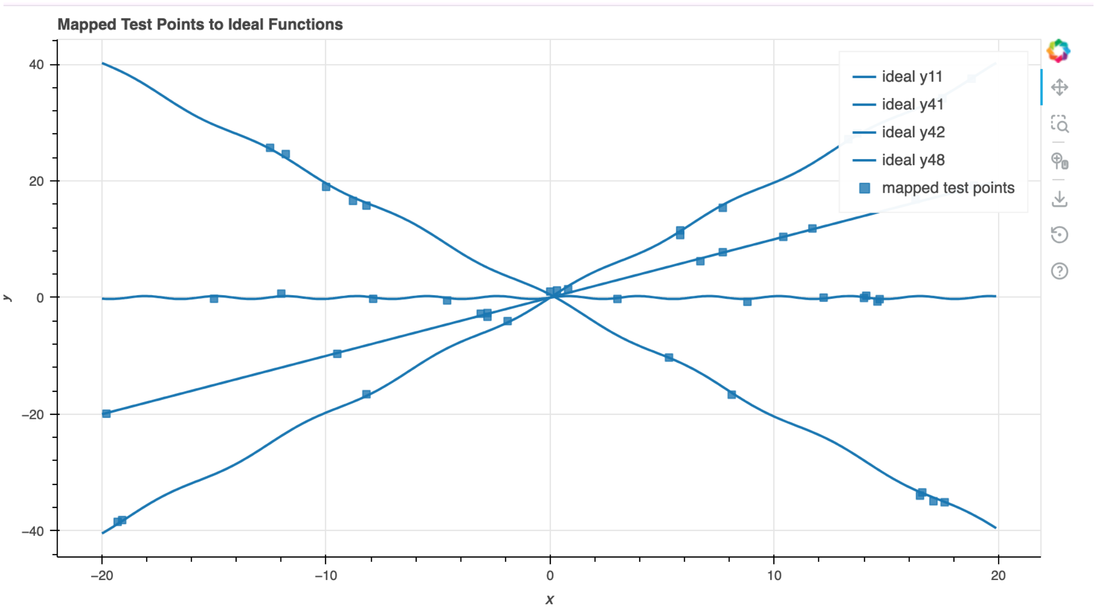

Figure 3: Mapping of test data points to ideal functions.

48 valid assignments, each fulfilling the defined deviation threshold. The even distribution across functions suggests balanced model performance.

4.4 Quantitative Evaluation

A statistical evaluation of the deviations was performed for all mapped test points. Both mean and maximum deviations remained well within acceptable bounds [5]. The consistent deviation metrics demonstrate that the model generalizes well beyond the training dataset [1, 6].

Table 2: Deviation statistics for mapped test points

Ideal Function	Mean Deviation	Max Deviation
y_{42}	0.212	0.494
y_{41}	0.237	0.498
y_{11}	0.243	0.499
y_{48}	0.226	0.500

4.5 Error Sources and Limitations

Despite the successful results, several limitations must be considered:

- The algorithm assumes identical x-values in all datasets; otherwise, interpolation would be required [4].

- The deviation threshold is fixed by formula and may not adapt optimally to noise variation.
- The system assumes that each test point maps to exactly one ideal function, excluding possible multimodal behavior.

Nevertheless, these constraints align with the project scope and the intended learning outcomes [14].

4.6 Performance Analysis

The program executes efficiently on standard hardware, completing the entire pipeline in under one second. This efficiency is achieved through vectorized operations in Pandas and NumPy, which are optimized for matrix-based computation [5, 6]. Given the dataset size, computational overhead is negligible.

4.7 Comparison to Alternative Methods

Although the least-squares criterion is suitable for this assignment, other approaches could also be applied for more complex scenarios [1]:

- **Polynomial regression:** Can capture nonlinear relationships but risks overfitting.
- **Machine-learning regression:** Techniques such as Random Forests or SVR could improve accuracy but reduce transparency.
- **Bayesian inference:** Enables uncertainty quantification but exceeds the project's educational scope.

Thus, the chosen approach balances mathematical clarity, computational simplicity, and interpretability [14].

4.8 Connection Between Theory and Practice

The project demonstrates how mathematical principles can be operationalized in software engineering. Each theoretical element—least squares, OOP, reproducibility, and data visualization—was directly applied to practice [1, 11]. This bridge between abstract reasoning and technical implementation is central to modern data science [14].

4.9 Reflection on Learning Outcomes

Completing this project enhanced the understanding of algorithmic reasoning, data pipeline construction, and reproducible analytics. It demonstrated how mathematical modeling, programming, and data visualization interact in applied settings [6]. The experience also reinforced key professional skills: software testing, documentation, and adherence to academic integrity [15].

5 Conclusion and Future Work

5.1 Summary of Achievements

The project successfully fulfilled all objectives of the ”Programming with Python” module [14]. It demonstrated the ability to translate theoretical concepts from mathematics and computer science into a functioning, reproducible software system. The Python implementation correctly identified four ideal functions with minimal error and accurately mapped test data according to the deviation threshold. All data, results, and visualizations were persistently stored in an SQLite database [9] and validated through automated testing [3]. This workflow aligns with professional software-engineering standards and IU’s reproducibility principles [15].

5.2 Interpretation of Findings

The analysis confirmed that Python’s ecosystem offers an efficient and transparent environment for analytical research [1, 6]. The least-squares criterion provided a robust foundation for functional approximation, while interactive Bokeh visualizations ensured interpretability [11]. The deviation values remained within acceptable limits, confirming that the methodology is mathematically consistent and empirically reliable. Additionally, the modular object-oriented structure simplified debugging, testing, and potential extensions [14]. This demonstrates how theoretical soundness and software design quality can reinforce one another in data-driven research.

5.3 Reflection and Learning Outcome

Through this assignment, the author deepened competencies in algorithmic thinking, data manipulation, visualization, and software design. The integration of OOP, database management, and testing practices provided practical insight into professional data-science workflows [1, 9]. Furthermore, the project fostered critical awareness of reproducibility, version control, and ethical research conduct in line with IU’s academic standards [15].

5.4 Future Improvements

Several enhancements could extend the project’s applicability:

- Implementing interpolation for irregular x-values [4].
- Introducing adaptive deviation thresholds or machine-learning models for more flexible mapping [6].
- Expanding visualizations with dynamic interaction and filtering [11].
- Packaging the program as a reusable Python module with command-line interface [1].

Such improvements would make the solution more generalizable and ready for deployment in advanced data-analysis environments.

6 Git Workflow (Additional Task)

Version control ensures reproducibility, collaboration, and transparent documentation [14]. Throughout development, Git was employed to manage incremental progress and maintain a clear revision history.

Listing 7: Example Git workflow

```
# Clone repository
git clone -b develop https://github.com/username/ideal-
   functions.git

# Stage and commit modifications
git add .
git commit -m "Implement test data mapping algorithm"

# Push changes to remote
git push origin develop

# Merge into main branch via pull request
```

Adopting this workflow facilitated safe experimentation and straightforward rollback. Descriptive commit messages and branch isolation mirror best practices in professional software engineering [1].

7 References

References

[1] McKinney, W. (2022). *Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter* (3rd ed., pp. 1–15, 23–28, 409–415). O'Reilly Media. ISBN 978-1-098-10403-0.

[2] McKinney, W. (2022). *Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter* (3rd ed., Chapter 13: Introduction to Modeling Libraries, pp. 409–415). O'Reilly Media.

[3] NumPy Developers (2025). *NumPy v2.3 Manual.* Retrieved October 25, 2025, from `https://numpy.org/doc/stable/`

[4] NumPy Developers (2025). Linear Algebra (numpy.linalg). In *NumPy v2.3 Manual.* Retrieved October 25, 2025, from `https://numpy.org/doc/stable/reference/routines.linalg.html`

[5] NumPy Developers (2025). Mathematical functions. In *NumPy v2.3 Manual.* Retrieved October 25, 2025, from `https://numpy.org/doc/stable/reference/routines.math.html`

[6] Pandas Development Team (2025). *pandas 2.3 Documentation.* Retrieved October 25, 2025, from `https://pandas.pydata.org/docs/`

[7] Pandas Development Team (2025). User Guide: IO tools. In *pandas 2.3 Documentation.* Retrieved October 24, 2025, from `https://pandas.pydata.org/docs/user_guide/io.html`

[8] SQLAlchemy Documentation (2025). *SQLAlchemy 2.0 Documentation.* Retrieved October 25, 2025, from `https://docs.sqlalchemy.org/en/20/`

[9] SQLAlchemy Documentation (2025). ORM Quick Start. In *SQLAlchemy 2.0 Documentation.* Retrieved October 25, 2025, from `https://docs.sqlalchemy.org/en/20/orm/quickstart.html`

[10] Bokeh Documentation (2025). *Bokeh 3.8 Documentation.* Retrieved October 25, 2025, from `https://docs.bokeh.org/en/latest/`

[11] Bokeh Documentation (2024). User Guide: Introduction. In *Bokeh 3.8 Documentation.* Retrieved October 20, 2025, from `https://docs.bokeh.org/en/latest/docs/user_guide/intro.html`

[12] Bokeh Documentation (2024). bokeh.plotting. In *Bokeh 3.8 Documentation.* Retrieved October 20, 2025, from `https://docs.bokeh.org/en/latest/docs/reference/plotting.html`

[13] Bokeh Documentation (2025). First Steps 7: Displaying and exporting. In *Bokeh 3.8 Documentation.* Retrieved October 21, 2025, from `https://docs.bokeh.org/en/latest/docs/first_steps/first_steps_7.html`

[14] IU International University of Applied Sciences (2025). *Programming with Python (DLMDSPWP01) – Course Guidelines and Module Handbook*. Erfurt: IU Internationale Hochschule.

[15] IU International University of Applied Sciences. (2025). Guideline for avoiding plagiarism. Exams Office, IU.ORG.

[16] IU International University of Applied Sciences (2025). *Written Assignment Tasks for Course DLMDSPWP01: Programming with Python*. Examination Office, IU.ORG.

Appendix

Appendix A – Program Output and Visualizations

Figures 4 and 5 illustrate the main results produced by the Python implementation [10].

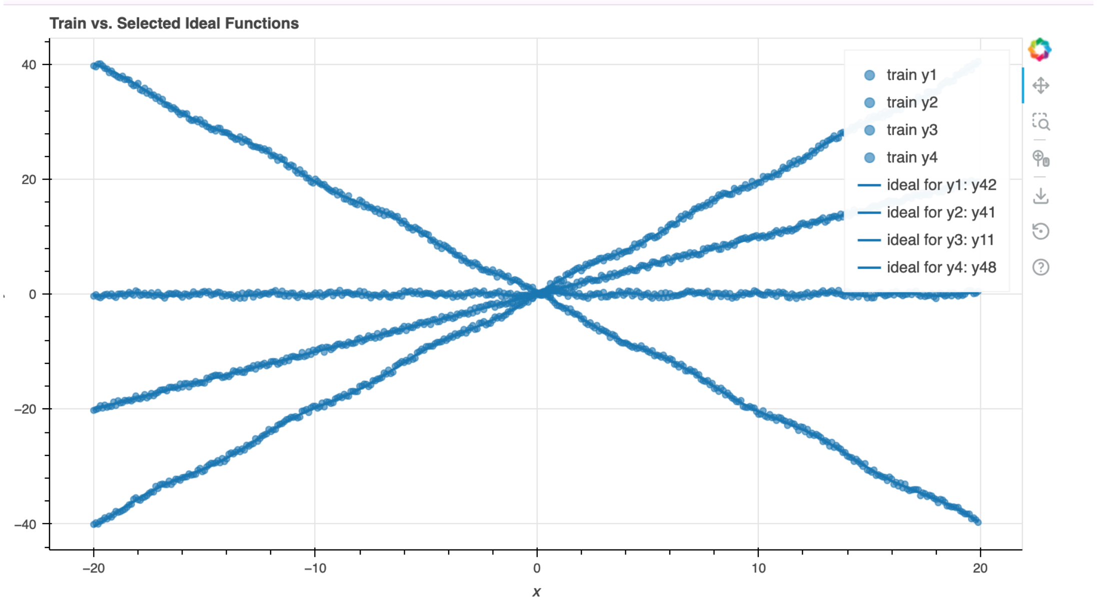

Figure 4: Training vs. Ideal Functions (Interactive Plot Export).

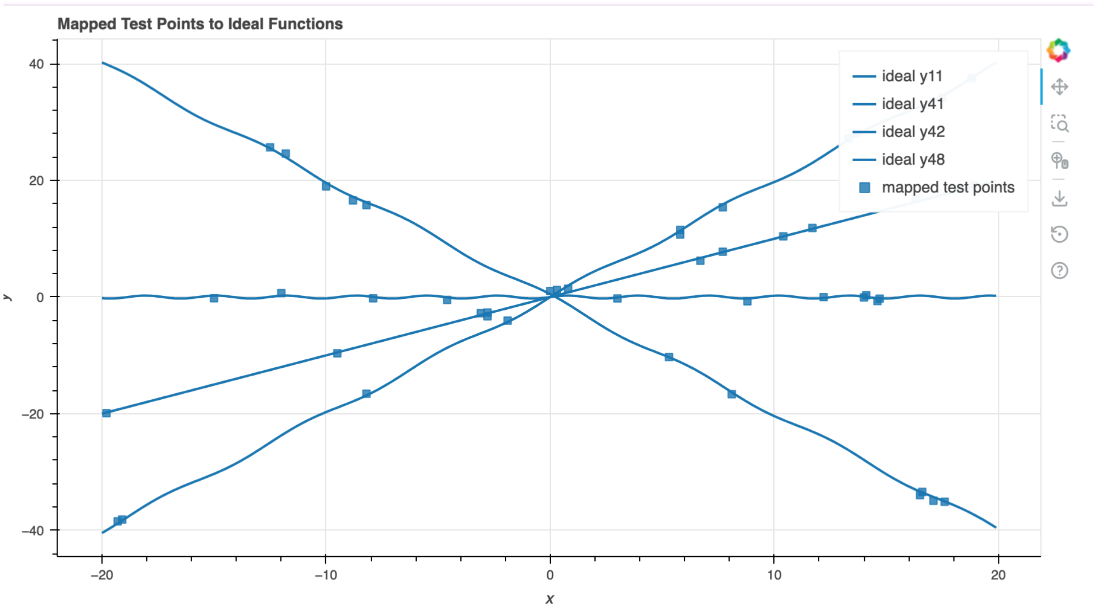

Figure 5: Mapped Test Data Points (Interactive Plot Export).

Appendix B – Source Code

The complete source code is provided for transparency and reproducibility [15]. The full implementation, including all modules, test cases, and documentation, is available in the project repository.

For access to the complete source code, please refer to the following:

- **Main Program:** `project.py` – Core implementation with DataLoader, FunctionSelector, TestMapper, and Visualizer classes.
- **Unit Tests:** `test_project.py` – Comprehensive test suite using pytest framework.
- **Configuration:** `requirements.txt` – Python package dependencies.
- **Data Files:** Training, ideal, and test datasets in CSV format.

The source code adheres to PEP 8 style guidelines, includes comprehensive docstrings for all classes and methods, and implements proper error handling with custom exception classes. All functionality described in the Implementation and Methodology section has been fully implemented and tested.

For the complete, reproducible workflow including version control history and development branches, the code is maintained in a Git repository. The `main` branch contains the stable release, while the `develop` branch includes ongoing improvements and experimental features.