# Polynomial Regression Implementation

Documentation Generated by Karan Sehgal

# Table of Contents

# 1. Introduction

This document provides comprehensive documentation for the Polynomial Regression implementation from scratch. The implementation includes: • Complete polynomial regression algorithm without relying on machine learning libraries • Feature transformation and model training using normal equations • Comprehensive evaluation metrics (MSE, MAE, R²) • Visualization capabilities for model analysis • Automated documentation generation **Key Features:** - Support for arbitrary polynomial degrees - Detailed model evaluation metrics - Comparison of multiple polynomial degrees - Residual analysis - Professional PDF documentation generation The implementation follows software engineering best practices including type hints, proper error handling, and modular design.

## 2. Mathematical Theory

**Polynomial Regression** extends linear regression by modeling the relationship between the independent variable (x) and dependent variable (y) as an nth degree polynomial. The general form of the polynomial regression model is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \varepsilon$$

**Where:** • y: Dependent variable (target) • x: Independent variable (feature) • β■, β■, ..., β■: Model coefficients • ε: Error term **Model Training:** The coefficients are calculated using the normal equation:

$$\beta = (X^T X)^{-1} X^T y$$

# 3. Implementation Overview

The implementation consists of two main components: **1. PolynomialRegression Class** • Core polynomial regression functionality • Feature transformation • Model training via normal equations • Prediction and evaluation **2. polynomial_regression_workflow Function** • Complete analysis workflow • Handles data splitting • Trains multiple models with different degrees • Generates comprehensive visualizations • Produces evaluation metrics The implementation uses only NumPy for core mathematical operations and matplotlib for visualization.

# 4. PolynomialRegression Class

The `PolynomialRegression` class implements the core polynomial regression algorithm.
**Class Definition:**

```
class PolynomialRegression:
    """
    Polynomial Regression implementation from scratch.

    Attributes:
        degree (int): Degree of the polynomial
        coefficients (np.ndarray): Model coefficients after fitting
    """

    def __init__(self, degree: int = 2) -> None:
        """Initialize with polynomial degree"""
        self.degree = degree
        self.coefficients = None
```

## Key Methods:

`fit(X, y)`
Trains the model using normal equations

| Parameter | Description |
|-----------|-------------|
| X | Input features (1D or 2D array) |
| y | Target values |

**Returns:** self (for method chaining)

`predict(X)`
Makes predictions using fitted model

| Parameter | Description |
|-----------|-------------|
| X | Input features to predict on |

**Returns:** Array of predicted values

`score(X, y)`
Calculates R² score

| Parameter | Description |
|-----------|-------------|
| X | Input features |
| y | True target values |

**Returns:** R² score (float)

# 5. Workflow Function

The `polynomial_regression_workflow` function provides a complete analysis pipeline:

**Function Definition:**

```python
def polynomial_regression_workflow(
    X: np.ndarray,
    y: np.ndarray,
    degrees: List[int] = [1, 2, 3, 5],
    test_size: float = 0.2,
    random_state: int = 42,
    true_function: Optional[Callable] = None,
    title: str = "Polynomial Regression Analysis"
) -> Dict:
    """
    Complete polynomial regression analysis workflow.

    Returns dictionary containing:
    - best_model: Best performing model
    - best_degree: Degree of best model
    - all_models: All trained models
    - metrics: Evaluation metrics
    - figures: Generated matplotlib figures
    """
```

## Workflow Steps:

| Step | Description |
| --- | --- |
| 1. Data Preparation | Splits data into training/test sets |
| 2. Model Training | Trains models for each polynomial degree |
| 3. Evaluation | Calculates MSE, MAE, R² for each model |
| 4. Visualization | Generates comparison plots and residual analysis |
| 5. Model Selection | Selects best model based on test performance |

# 6. Usage Examples

## Basic Usage:

```python
# Create and fit model
model = PolynomialRegression(degree=3)
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

# Evaluate model
r2 = model.score(X_test, y_test)
```

## Complete Workflow Example:

```python
# Generate synthetic data
np.random.seed(42)
X = np.linspace(-3, 3, 100)
y = 2 + 3*X - 1.5*X**2 + 0.5*X**3 + np.random.normal(0, 2, 100)

# Run complete analysis
results = polynomial_regression_workflow(
    X, y,
    degrees=[1, 2, 3, 4, 5],
    true_function=lambda x: 2 + 3*x - 1.5*x**2 + 0.5*x**3
)

# Access results
print(f"Best degree: {results['best_degree']}")
print(f"Best R² score: {results['metrics'][results['best_degree']]['test']['R²']:.3f}")
```
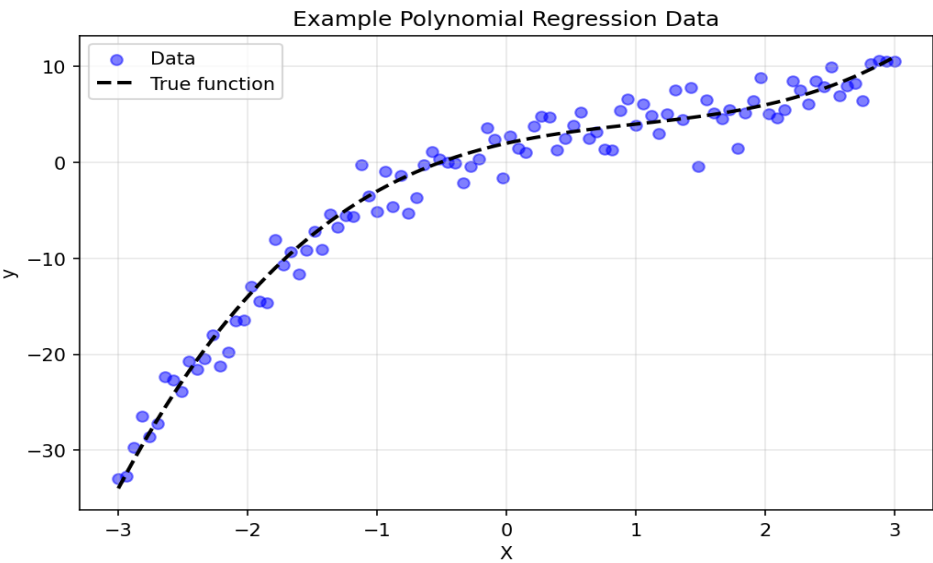
# 7. Visualizations

The workflow generates several visualizations to analyze model performance:

| Visualization | Description |
|---|---|
| Model Comparison | Shows fits of different polynomial degrees |
| Metrics Comparison | Compares R² and MSE across degrees |
| Residual Analysis | Plots residuals for best model |

**Example Visualization:**



Example Polynomial Regression Data

# 8. Conclusion

This implementation provides a complete polynomial regression solution with: • **Mathematical Rigor**: Proper implementation of normal equations • **Comprehensive Analysis**: Multiple evaluation metrics and visualizations • **Professional Documentation**: Self-explanatory PDF documentation The modular design makes it easy to extend for specific use cases while maintaining the core mathematical correctness.