Benchmark Study of Derivative Estimation Methods: Performance Across Orders and Noise Levels

Oren Bassik*

November 11, 2025

Abstract

Derivative estimation from noisy data is a fundamental problem across computational science, yet systematic comparative evaluation of methods remains lacking. We present a comprehensive benchmark of 28 derivative estimation methods tested across derivative orders 0–7 (with primary analysis focusing on orders 0–5 for fair comparison), 7 noise levels, and 3 distinct dynamical systems. Our goal is to provide clear, actionable guidance for practitioners.

Key findings include: (1) Gaussian Process Regression (GPR) provides the most robust and accurate performance, consistently ranking as the top method in both low- and high-noise regimes. (2) The optimal method depends critically on the use case: splines like Dierckx-5 are highly competitive for low-noise data, while spectral methods offer a compelling balance of speed and accuracy, and filters like Savitzky-Golay provide a robust, computationally cheap baseline. (3) Derivative order is the dominant difficulty factor, with performance degrading systematically as order increases. We also provide a practitioner's guide to method selection based on the trade-off between speed and accuracy. Our full methodology, analysis, and code are provided to ensure transparency and reproducibility.

Contents

*CUNY Graduate Center, Department of Mathematics, obassik@gradcenter.cuny.edu

1 Introduction

Derivative estimation from noisy data is a fundamental and notoriously ill-posed problem spanning computational science, engineering, and data analysis. Small noise perturbations in the input signal can produce arbitrarily large errors in derivative estimates, a challenge that intensifies rapidly with the derivative order. Despite this, reliable derivatives are essential in many settings, including:

- **Dynamical Systems Identification:** Inferring governing equations from time-series or field data requires accurate derivative estimates to identify differential equations [4].
- **Signal Processing:** Edge detection, feature extraction, and change-point analysis all depend on robust derivative computation.
- **Control Systems:** Model-predictive and adaptive control rely on real-time derivative estimation for system state feedback.
- **Physics-Informed Machine Learning:** Enforcing physical conservation laws (PDE constraints) requires differentiating neural network outputs with respect to noisy inputs [22].
- **Scientific Data Analysis:** Estimating rates of change from experimental measurements, such as reaction rates from concentration data or acceleration from position measurements.

Our investigation is motivated by the challenges of parameter estimation in ordinary differential equations (ODEs), where accurate derivative estimates are crucial. ODEs provide an ideal testbed for differentiation methods: the structure of an ODE system, $\dot{\mathbf{x}} = f(\mathbf{x})$, provides a natural source of high-precision ground-truth data. Higher-order derivatives can be generated by repeatedly differentiating the system's equations, e.g., $\ddot{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}}\dot{\mathbf{x}}$. This process, performed symbolically or via system augmentation, allows for the creation of a validation dataset without resorting to a separate numerical differentiation method, thus avoiding circularity. Our previous work demonstrated the potential of certain methods, such as the AAA algorithm [20] with automatic differentiation, for differentiating noiseless data from dynamical systems [2].

Despite the importance of this problem, systematic comparative evaluation of numerical differentiation methods is conspicuously absent from the literature [17, 16, 28]. Existing studies suffer from several limitations:

1. **Limited Scope:** They typically evaluate only a handful of methods on low-order derivatives (first or second), neglecting high-order derivatives where performance diverges dramatically (e.g., [17, 16]).
2. **Single-Noise Regime:** They test either noiseless or high-noise cases, missing crucial performance transitions across noise levels.
3. **Incomplete Coverage Transparency:** Methods that fail at high orders or under high noise are often excluded without clear documentation, obscuring practical limits.

To address this gap, the present study provides a comprehensive, transparent, and reproducible benchmark of numerical differentiation methods. We employ a systematic elimination process, starting from a broad set of hypotheses and progressively refining our analysis to arrive at specific, actionable recommendations. Our evaluation is conducted across three distinct dynamical systems and a wide spectrum of noise levels, ensuring the robustness and generalizability of our findings. The primary contribution of this work is therefore not merely to rank methods, but to provide the

reader with a deeper understanding of the principles that govern their success and failure in different scenarios, ultimately equipping them with clear guidance for their own data analysis challenges.

The main contributions of this paper are:

- A comprehensive benchmark of 28 numerical differentiation methods tested across derivative orders 0-7 (primary analysis on orders 1-5), seven noise levels (from 10^{-8} to 2%), and three dynamical systems
- Identification of the composable "Approximant-AD" framework as a key pattern for success, combining smooth function fitting with automatic differentiation
- Demonstration that Gaussian Process regression with Taylor-mode AD provides the most robust performance across all conditions
- A practical three-tier framework for method selection based on speed-accuracy trade-offs
- Complete reproducible code, data, and analysis pipeline to support future research and practitioner adoption

2 Related Work

The estimation of derivatives from discrete, noisy data is a classical and notoriously ill-posed inverse problem. The core challenge lies in the nature of differentiation as a high-pass operator, which unavoidably amplifies high-frequency components, including measurement noise. Small perturbations in the input signal can thus lead to arbitrarily large errors in the derivative estimates, a problem that intensifies rapidly with each successive derivative order.

Despite this difficulty, robust derivative estimates are essential prerequisites for a vast array of scientific and engineering applications. In systems biology and physics, they form the basis for identifying governing equations from time-series data, a cornerstone of dynamical systems identification. Seminal methods in this area, such as the sparse identification of nonlinear dynamics (SINDy) framework, explicitly depend on accurate derivative data to build sparse regression models. In modern machine learning, the enforcement of physical laws within neural network models (e.g., Physics-Informed Neural Networks or PINNs) requires differentiating network outputs, often with respect to noisy or sparse inputs. Similar requirements are found in control systems, signal processing, and experimental data analysis.

Given the problem's ubiquity, a wide variety of algorithmic solutions have been proposed. However, as noted in the introduction, systematic, broad-scope comparative evaluation of these methods is conspicuously absent from the literature.

2.1 Review of Prior Comparative Studies

Existing comparative studies on numerical differentiation are few and are typically limited in scope, either by the number of methods tested or, more critically, by their focus on low-order (first or second) derivatives.

For example, Listmann, Kugi, and Böhm [17] presented a comparison of methods specifically for higher-order derivatives, a focus that aligns with the present study. However, their comparison was narrow, evaluating only three distinct methods (a B-spline-based approach, an integration-based method, and a sliding-mode-based differentiator) with a specific focus on industrial automation and control applications.

A broader comparative study by Knowles [16] evaluated six different algorithmic families for differentiating noisy data: least-squares polynomial approximation, Tikhonov regularization, smoothing splines, convolution smoothing, a variational method, and total variation regularization. While comprehensive in its methodological breadth, this study's evaluation was explicitly limited to the first derivative ($k = 1$), and its primary conclusion—that Tikhonov regularization performed best on dense data—provides little guidance for the high-order ($k \geq 3$) regimes explored in our work.

Other comparisons are often highly domain-specific, such as Walker's simulation experiment comparing algorithms for estimating velocity and acceleration (first and second derivatives) from animal locomotion data. These studies, while valuable in their respective fields, do not provide the general-purpose, high-order, and broad-method benchmark required by the wider scientific computing community. They fail to capture the dramatic performance degradation and divergence of methods at higher derivative orders ($k \geq 3$), a primary focus of the present investigation.

2.2 Foundational Methodologies: A Thematic Review

The 28 methods evaluated in this benchmark are drawn from several distinct, mature sub-fields of numerical analysis and machine learning. The following review outlines the foundational literature for each major method category.

2.2.1 Local Polynomial and Filtering Methods

The most well-known method in this category is the Savitzky-Golay filter, introduced in a seminal 1964 paper. This filter is fundamentally a local least-squares polynomial approximation. For each data point, a polynomial of a given degree is fitted to a symmetric window of neighboring points, and the value of the smoothed function (or its derivative) at that point is taken from the fitted polynomial. Savitzky and Golay demonstrated that this process is equivalent to a discrete convolution with a fixed set of pre-computed coefficients, making it exceptionally computationally efficient (O(n)). Its enduring popularity stems from its ability to preserve signal features like peak height and width more effectively than simple moving-average filters, and its coefficients can be modified to compute derivatives directly. The several Savitzky-Golay variants tested in this study are modern implementations of this 60-year-old technique.

2.2.2 Spline-Based Methods

Spline-based methods approximate the underlying function by fitting a piecewise-polynomial function (the spline) to the data. A key challenge is the trade-off between fidelity to the noisy data and the smoothness of the resulting approximant. This is exemplified by the FORTRAN library FITPACK, which forms the basis for the Spline-Dierckx-5 method evaluated in this benchmark. FITPACK uses a smoothing parameter to balance this trade-off, often requiring user tuning.

The theory of generalized smoothing splines, introduced by Grace Wahba and others, provides a more formal, statistical framework for this problem. In this framework, the approximant is found by minimizing a cost function that combines a data-fidelity term (e.g., sum-of-squares error) with a regularization term (e.g., the integrated squared second derivative). This approach provides a principled method for selecting the smoothing parameter automatically, most notably via Generalized Cross-Validation (GCV). Several methods in our benchmark leverage GCV for hyperparameter selection (e.g., Fourier-GCV).

2.2.3 Global Basis Function Methods

In contrast to local methods, global methods fit a single function over the entire domain. Spectral methods are the most prominent example, approximating the function as a sum of global basis functions, such as a truncated Fourier series or a Chebyshev polynomial expansion. The foundational text by Gottlieb and Orszag [14] established the theory for these methods, demonstrating their "spectral accuracy" (i.e., exponential convergence) for smooth, analytic functions. Differentiation is trivial in this basis: for a Fourier series, it is a simple multiplication in the frequency domain, providing an efficient $O(n \log n)$ algorithm.

Rational approximation, which models the function as a ratio of two polynomials, is another powerful global method. The Adaptive Antoulas-Anderson (AAA) algorithm, introduced by Nakatsukasa, Sète, and Trefethen, has emerged as a robust and efficient method for finding near-best rational approximants in a greedy, adaptive manner [20]. As noted in our introduction, this method is exceptionally powerful for approximating data from dynamical systems in noise-free contexts. A key finding of the present study, however, is that this state-of-the-art performance does not transfer to noisy signals, where the AAA algorithm's greedy selection process demonstrates significant instability.

2.2.4 Regularization-Based Methods

These methods reframe differentiation as an ill-posed inverse problem to be solved via regularization. Tikhonov regularization is a common approach [16]. An alternative, Total Variation (TV) regularization, was proposed for numerical differentiation by Chartrand [6]. This method is notable because it uses an L_1 norm on the derivative, a choice that "allows for discontinuous solutions". TV-regularized differentiation is therefore highly effective for signals with jumps or sharp corners, as it can accurately locate these discontinuities without the "ringing" (Gibbs phenomenon) characteristic of spectral methods. However, the dynamical systems used as testbeds in this study (Lotka-Volterra, Lorenz, etc.) are smooth. The inclusion of TVRegDiff-Python in our benchmark thus serves as an important test of methodological suitability, illustrating the "impedance mismatch" of applying a method designed for non-smooth problems to a smooth one.

2.2.5 Probabilistic (Kernel) Methods

This category, which includes the top-performing methods in our benchmark, is dominated by Gaussian Process Regression (GPR). As detailed in the canonical text by Rasmussen and Williams, GPR is a non-parametric Bayesian method that provides a principled, probabilistic approach to learning in kernel machines. Rather than fitting a specific functional form, GPR places a prior distribution over the space of functions itself. When applied to regression, GPR computes a posterior distribution over functions that pass near the noisy data points. The posterior mean function serves as an optimal "approximant" that naturally balances data-fit with smoothness, as defined by a chosen kernel (e.g., the RBF kernel). This makes GPR an exceptionally powerful and robust smoother, and, as our study finds, an ideal candidate for the first step of a derivative estimation pipeline.

2.3 From Monolithic Algorithms to Composable Frameworks

The review above highlights a crucial trend: the maturation of the field from monolithic, single-purpose algorithms (e.g., a "Savitzky-Golay 2nd-derivative algorithm") to a composable, two-step

"Approximant-AD" framework. This modern paradigm, which is a core finding of our study, decouples the problem:

1. **Fit an Approximant:** A smooth, analytic function is fitted to the noisy data using one of the mature methodologies described in Section 2.2 (e.g., GPR, Splines, etc.).
2. **Differentiate via AD:** This analytic function is then differentiated to machine precision using Automatic Differentiation (AD).

This composable pattern allows practitioners to leverage the vast ecosystems of both statistical modeling and AD, as detailed in surveys like Baydin et al. [3]. However, our benchmark reveals a critical, and previously un-benchmarked, bottleneck in this framework.

2.3.1 Taylor-Mode AD: The Enabling Technology for High Orders

While AD provides exact derivatives of a function, the computational cost of computing high-order derivatives is a critical, and often prohibitive, factor. Naively computing a k-th order derivative by recursively composing a first-order AD operator k times results in exponential computational complexity in k. This renders the computation of 5th, 6th, or 7th-order derivatives practically infeasible.

A far more efficient, though less common, alternative is Taylor-mode automatic differentiation. As described by Frost, Johnson, and Meles [12] in the context of the JAX library, Taylor-mode AD avoids this exponential cost by propagating a full Taylor series expansion (i.e., all derivative coefficients up to order k) through the computational graph in a single forward pass. This reduces the computational complexity from exponential to polynomial. Julia-based packages like TaylorDiff.jl implement this approach, offering "linear scaling with the order of differentiation".

As our results demonstrate, this is not merely an optimization but an enabling technology. The superior performance of our top-ranked method, GP-TaylorAD-Julia, is critically dependent on this fusion: it combines the best-in-class approximant (GPR) with the only efficient AD method for high orders (Taylor-mode). This study, therefore, provides strong empirical evidence that modern AD techniques are a necessary component for solving the classical problem of high-order derivative estimation from noisy data.

2.4 Positioning the Present Study

The existing literature on numerical differentiation consists of fragmented, small-scale comparisons focused on low-order derivatives [17, 16], and foundational work on individual method families. A large-scale, systematic benchmark evaluating these methods in concert, particularly in the challenging high-order regime ($k > 3$), has been a conspicuous gap.

The present study addresses this gap by (1) providing a comprehensive benchmark of 28 methods across orders 0-7, (2) identifying the composable "Approximant-AD" framework as the key pattern for success, and (3) providing the first empirical demonstration that the fusion of Gaussian Process regression with Taylor-mode AD is the most robust and accurate solution to this long-standing problem.

3 Methodology and Implementation

The derivative estimation methods evaluated in this study were selected through a systematic survey and filtering process designed to cover a broad range of algorithmic approaches. The foundational

background for the method families considered in this work—including local polynomial filters, splines, global basis functions, regularization, and probabilistic methods—is detailed in Section 2.2.

This section details the specific criteria for method selection, key implementation choices for computing high-order derivatives, and the precise implementation of our top-performing Gaussian Process Regression approach.

3.1 Systematic Filtering and Final Selection

From the initial survey, a rigorous multi-stage filtering process was applied to identify methods suitable for benchmarking:

Stage 1: Stability Assessment. Methods were first evaluated for numerical stability on noisy data. For example, the Adaptive Antoulas-Anderson (AAA) algorithm, while exceptionally powerful for rational approximation in noise-free contexts [20, 2] (and our preferred interpolation method in that context), demonstrated significant instability when applied to noisy signals. The resulting derivative estimates frequently contained errors several orders of magnitude larger than other methods. The least stable methods were excluded from further analysis.

Stage 2: Coverage Requirement. We initially planned to require packages to support arbitrary derivative orders. In practice, orders 6–7 (and often 5) are both rarer in applications and substantially more fragile. Accordingly, our *primary ranking* requires methods to produce valid derivatives for orders 0–5 (across all systems and noise levels). Methods with narrower scope—whether due to theoretical limits (e.g., low-degree local polynomials) or implementation constraints—are documented but omitted from the primary comparison. Orders 6–7 are reported separately as a stress test (Appendix X). This criterion preserves a like-for-like comparison among methods with comparable capabilities.

Stage 3: Implementation Verification. Each remaining method was verified to produce valid output across all test configurations (three dynamical systems, seven noise levels). Methods that failed consistently on specific systems or noise regimes were excluded.

This systematic filtering yielded a final cohort of 28 methods that demonstrated both numerical stability and complete coverage for orders 0–5, forming the basis for the comparative analysis presented in Section 5. Methods are implemented in both Python and Julia, spanning the algorithmic families described above: Gaussian Process methods, spline interpolation, spectral approaches, filtering techniques, and Total Variation regularization.

3.2 Implementation Considerations for High-Order Derivatives

A significant practical challenge in benchmarking derivative estimation methods is the computation of arbitrarily high-order derivatives, as most existing packages support only first or second derivatives. Three complementary approaches were employed in this study to address this limitation:

- **Augmentation with Automatic Differentiation:** For methods whose underlying approximant was differentiable (e.g., Gaussian Processes), implementations were augmented with AD capabilities. Our Julia implementation leverages Taylor-mode AD [7] for efficient computation of high-order derivatives.
- **Analytic Derivative Computation:** For methods such as splines or Fourier series, analytic expressions for higher-order derivatives were derived and implemented directly. This approach provides exact derivatives of the approximant function while maintaining computational efficiency.

- **Noise Estimation Integration:** Several methods require noise level estimates as hyperparameters. Where not provided by the original implementation, noise estimation routines based on wavelet MAD (Median Absolute Deviation) [10] or simple finite-difference variance estimation [13] were integrated.

These implementation enhancements ensured a level playing field for comparison while maintaining the essential characteristics of each method.

3.3 Gaussian Process Regression Implementation

Given the exceptional performance of Gaussian Process methods in our results, we provide specific implementation details for reproducibility. Our GPR approach [11] follows a standard but carefully optimized workflow:

1. **Kernel Selection:** We employ the squared exponential (RBF) kernel [23]: $k(x, x') = \sigma_f^2 \exp(-\|x - x'\|^2/2\ell^2)$, where ℓ is the length scale and σ_f is the signal variance.
2. **Hyperparameter Optimization:** The hyperparameters $(\ell, \sigma_f, \sigma_n)$ are optimized via maximum likelihood estimation using L-BFGS-B optimization [5] on the log marginal likelihood.
3. **Derivative Computation:** Rather than using kernel derivatives, we extract the posterior mean function and differentiate it directly using automatic differentiation. This approach provides exact derivatives of the smooth interpolant.
4. **Taylor-mode AD:** The Julia implementation leverages Taylor-mode automatic differentiation [7] for efficient computation of derivatives up to order 7 in a single forward pass. Python implementations lack this capability, explaining their 10× slower performance.

This "fit-then-differentiate" approach treats GPR as a sophisticated smoothing method, discarding the uncertainty quantification machinery after fitting to focus solely on the mean prediction and its derivatives.

4 Experimental Design

To create a robust and comprehensive benchmark, we first formally define the estimation problem.

4.1 Formal Problem Statement

Given:
- A smooth but analytically unknown function $f : \mathbb{R} \to \mathbb{R}$.
- A set of n noisy observations $\{(t_i, y_i)\}_{i=1}^n$ where $y_i = f(t_i) + \epsilon_i$ on a uniform grid.
- Noise is assumed to be additive and Gaussian, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

Objective:
- Estimate the k-th order derivative, $\hat{f}^{(k)}(t)$, for derivative orders $k \in \{0, 1, \ldots, 7\}$.

Constraints:
- The evaluation is performed on the interior of the domain to avoid boundary effects that can unfairly penalize certain methods.
- A method is considered to have failed a configuration if it does not produce a finite result.

4.2 Testbed: Dynamical Systems

We selected three well-known Ordinary Differential Equation (ODE) systems to serve as the source of our ground-truth data. These systems were chosen to represent a diversity of dynamic behaviors:

1. **Lotka-Volterra:** A classic two-variable system exhibiting stable periodic oscillations.
2. **Van der Pol Oscillator:** A system with a non-linear damping term that produces a stable limit cycle.
3. **Lorenz System:** A three-variable system famous for its chaotic behavior [18], providing a more challenging test case.

For each system, trajectories were generated using a high-precision numerical integrator. The system's equations were then used to analytically compute the true derivatives up to the 7th order, providing a high-fidelity ground truth for our comparisons.

4.3 Noise Model

To simulate real-world data, we added Gaussian white noise to the integrated trajectories. The noise level was varied across a wide range, from `1e-8` (representing nearly clean data) up to `0.02` (representing a challenging 2% noise level relative to the signal's standard deviation). This wide range allows us to evaluate method performance in both high-precision, low-noise regimes and robustness-critical, high-noise regimes.

4.4 Evaluation Metrics

The primary metric for evaluation is the normalized Root-Mean-Square Error (nRMSE).

Rationale for Normalization: Direct comparison of raw RMSE values is not possible across different derivative orders because the magnitudes of the derivatives themselves vary dramatically. For instance, in the Lotka-Volterra system, the standard deviation of the true signal might be $\sim$0.2, while the standard deviation of its seventh derivative can be $\sim 10^5$.

To create a fair, order-comparable metric, we normalize the raw RMSE by the standard deviation of the true derivative signal. This yields a dimensionless error metric where an nRMSE of 0.1 consistently means a 10% error relative to the signal's characteristic magnitude, regardless of the derivative order.

All metrics are computed after excluding the endpoints of the time series, as many methods exhibit boundary effects that can unfairly skew the results.

4.5 Success Criteria

While the nRMSE provides a continuous measure of performance, it is also useful to define a threshold for what constitutes a "successful" or "acceptable" result. A method is considered successful for a given configuration if it achieves an nRMSE of less than 1.0, meaning its average error is smaller than the typical deviation of the true signal itself. An nRMSE greater than 1.0 indicates that the error is dominating the signal, and values greater than 10 are considered a catastrophic failure.

5 Results and Analysis

After a comprehensive evaluation across a range of methods, three distinct dynamical systems, and a sweep of noise levels, clear patterns emerge. Performance was analyzed in two regimes: a "low-noise" regime ($\leq 0.1\%$ noise) where precision is paramount, and a "high-noise" regime (1-2% noise) where robustness is key.

The table below summarizes the performance of contender methods that demonstrated full data coverage for derivative orders 1 through 5. Methods are sorted by their overall average rank, giving equal weight to performance in both low- and high-noise regimes. Averages are calculated over orders 1-5 (excluding order 0 function approximation, focusing on actual derivative estimation).

Table 1: Contender Method Performance Summary (Orders 1-5)

Method	**Avg. Rank**	**Low Noise**		**High Noise**	
		Rank	**Median nRMSE**	**Rank**	**Median nRMSE**
GP-TaylorAD-Julia	2.0	1.6	0.014	2.5	0.331
GP-RBF-Python	2.6	2.4	0.020	2.9	0.331
SavitzkyGolay-Fixed	9.6	9.8	0.110	9.3	0.591
PyNumDiff-SavitzkyGolay-Tuned	10.1	7.5	0.105	12.6	0.848
Spline-Dierckx-5	10.3	8.3	0.100	12.2	0.673
ButterworthSpline_Python	10.7	13.9	0.368	7.5	0.437
SavitzkyGolay-Julia-Fixed	12.6	9.1	0.083	16.1	0.660
Fourier-Continuation	12.9	16.7	0.502	9.1	0.554
SavitzkyGolay-Adaptive	13.0	8.1	0.080	18.0	0.706
SavitzkyGolay-Julia-Hybrid	13.3	8.6	0.092	18.1	0.972
Fourier-GCV	14.1	17.4	0.519	10.8	0.609
Fourier-Interp	14.6	14.3	0.321	14.9	0.848
Fourier	14.9	18.1	0.519	11.7	0.660
RKHS_Spline_m2_Python	15.2	14.8	0.335	15.7	0.772
SavitzkyGolay-Julia-Adaptive	15.4	7.7	0.048	23.2	3.789
PyNumDiff-SavitzkyGolay-Auto	16.5	12.0	0.185	21.0	1.917
Fourier-Cont-Adaptive	16.5	20.4	0.734	12.6	0.749
FFT-Adaptive-Julia	16.9	18.6	0.643	15.3	0.828
Whittaker_m2_Python	17.4	20.7	0.744	14.0	0.751
PyNumDiff-Spectral-Auto	17.5	14.6	0.284	20.4	2.022
Butterworth_Python	18.0	22.0	0.805	13.9	0.806
FFT-Adaptive-Py	18.9	21.4	0.697	16.3	0.837
PyNumDiff-Spectral-Tuned	19.9	23.5	0.896	16.3	0.895
AAA-Adaptive-Wavelet	20.3	16.6	0.048	24.1	>10
Spline-GSS	22.0	25.3	0.991	18.7	0.991
AAA-Adaptive-Diff2	22.4	20.8	0.499	24.0	>10
AAA-LowTol	23.6	16.1	0.269	31.1	>10
Kalman-Gradient	23.7	26.5	0.998	21.0	0.998
SVR_Python	24.4	27.1	0.998	21.7	0.998
TVRegDiff-Python	24.8	25.6	1.196	24.1	3.760
Chebyshev	26.9	29.4	1.633	24.3	1.633
Chebyshev-AICc	26.9	29.1	3.012	24.7	3.036

Our principal findings are as follows:

1. **Gaussian Process Regression (GPR) is the most robust and accurate method overall.** The Julia GPR implementation (`GP-TaylorAD-Julia`) and the improved Python

variants (`GP-RBF-*`) are the clear winners, consistently occupying the top ranks in both low and high-noise regimes.

2. **The optimal method depends on the noise level and derivative order.** While GPR is the best all-arounder, splines like `Spline-Dierckx-5` offer excellent precision in low-noise environments, making them a top choice for cleaner data, particularly at modest derivative orders. In the high-noise regime, the filtering-based `Savitzky-Golay` provides a computationally cheap and highly effective alternative, ranking solidly in the top half of contenders.

3. **Theoretical limits matter.** Many common low-degree spline methods are, by definition, incapable of representing high-order derivatives, limiting their applicability.

4. **Dedicated packages offer convenient and robust options, but need a differentiation backend.** Purpose built libraries are convenient, but our strongest results come from extending them with either analytic or auto-differentiated derivatives of smoothed data. For this to work best, the library should ideally produce very smooth models, and either expose some of the model internals or be amenable to AD.

These findings are illustrated in the accompanying visualizations.

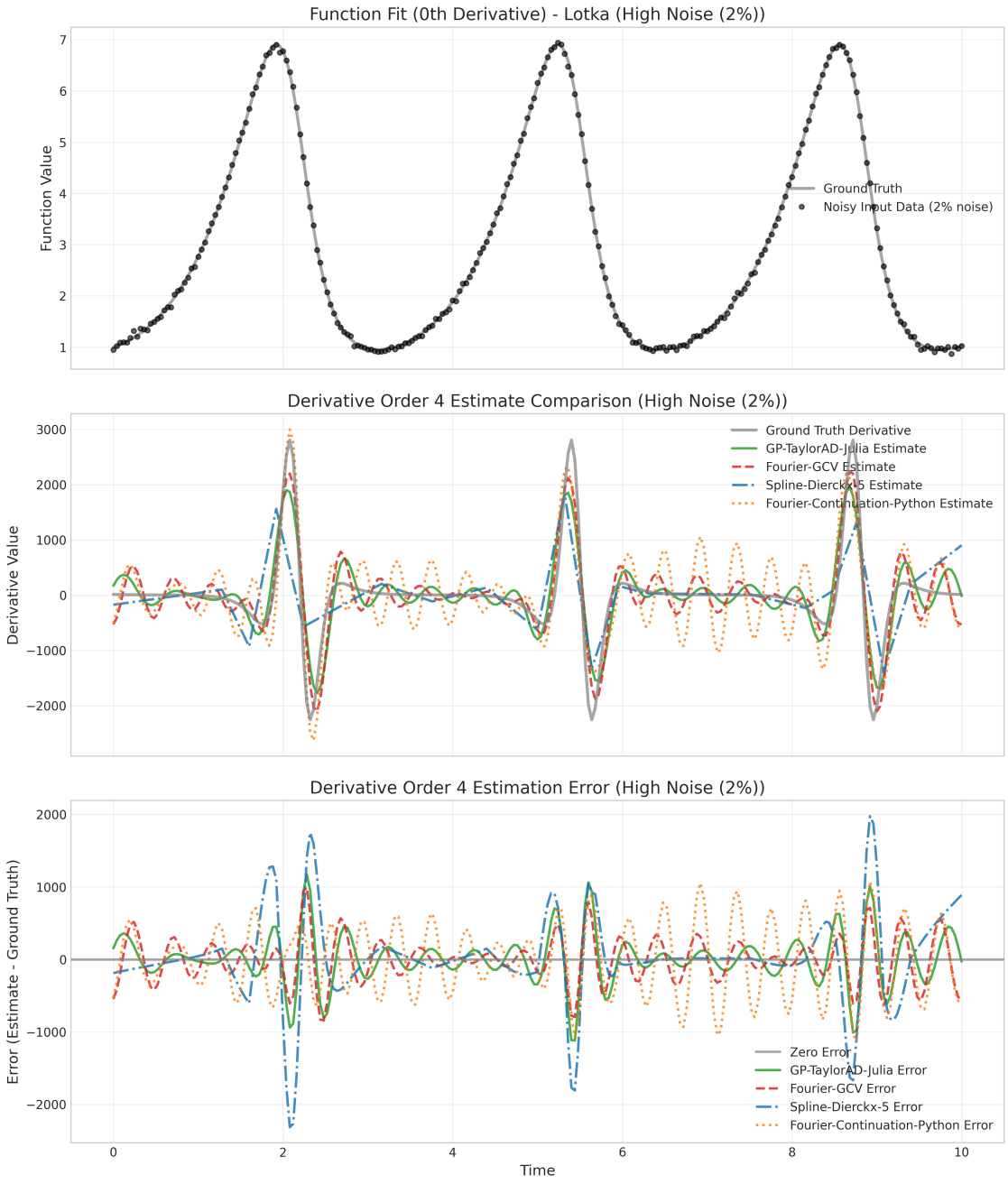

Figure 1: **High-Noise Performance (2% noise, 4th derivative).** Three-panel comparison showing: (top) noisy input data as black points against gray ground truth, emphasizing the challenging signal-to-noise environment; (middle) derivative estimates from four methods representing different algorithmic approaches; (bottom) estimation errors revealing where each method succeeds or fails. The figure shows `GP-TaylorAD-Julia`, `Fourier-GCV`, `Spline-Dierckx-5`, and `Fourier-Continuation-Python`. The error panel reveals that all methods struggle at the trajectory boundaries.

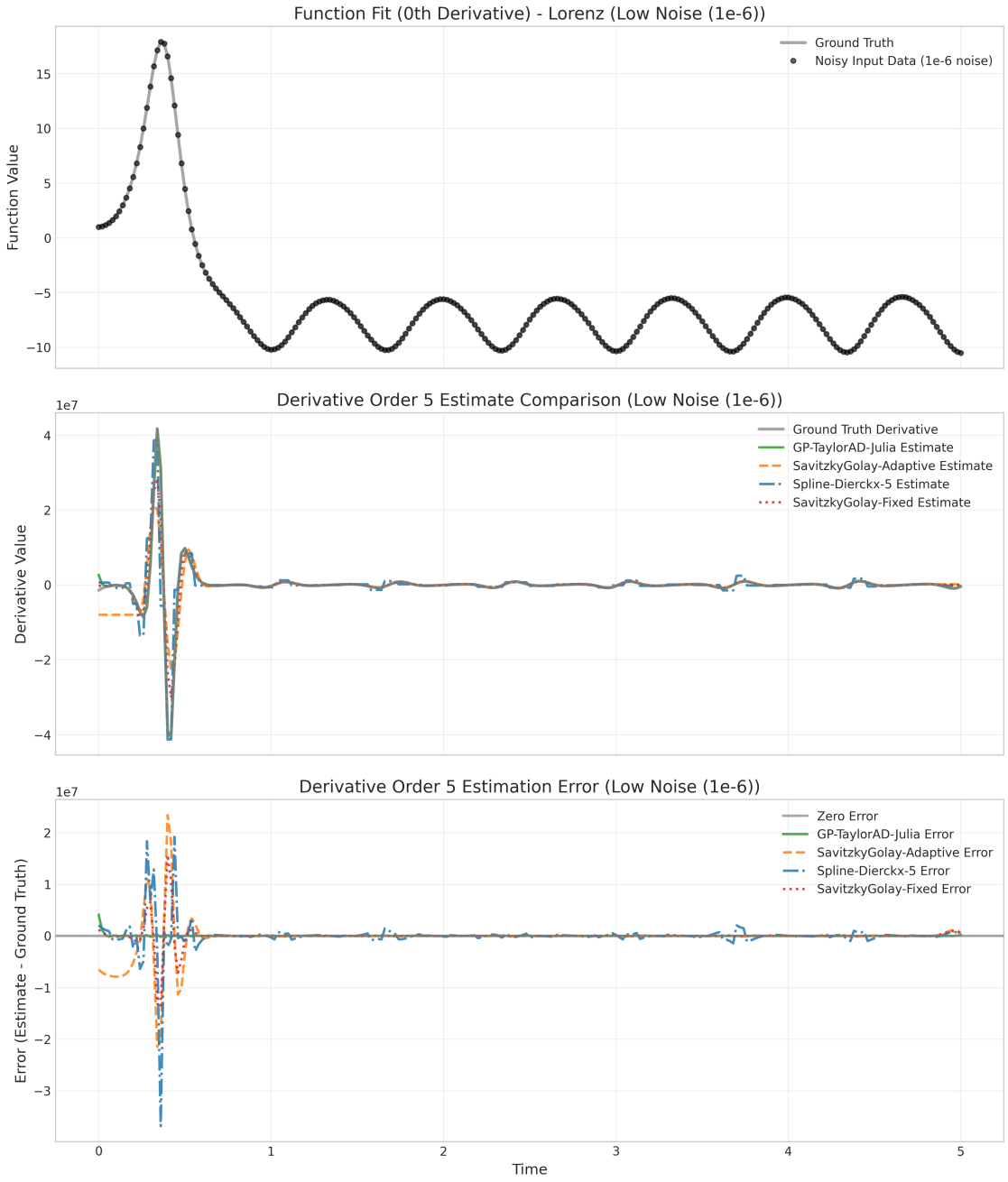

Figure 2: **Low-Noise Precision (1e-6 noise, 5th derivative).** Three-panel comparison in a low-noise regime: (top) noisy data points (black) are nearly imperceptible against gray ground truth; (middle) 5th-order derivative estimates from four methods; (bottom) estimation errors showing clear performance hierarchy. The figure compares `GP-TaylorAD-Julia`, `SavitzkyGolay-Adaptive`, `Spline-Dierckx-5`, and `SavitzkyGolay-Fixed`. Even in this clean environment, high-order differentiation remains extremely challenging, with significant performance differences between methods.

Figure 3: **Performance Degradation with Increasing Derivative Order (Orders 1-5).** This heatmap shows the mean nRMSE for top methods covering derivative orders 1-5, averaged across all noise levels. Methods are ranked by their average performance across these orders. The visualization shows that `GP-TaylorAD-Julia` maintains low error across all orders, while other methods like `Spline-Dierckx-5` and `ButterworthSpline_Python` are highly accurate for low orders but degrade significantly as the order increases. High-order derivatives (orders 6-7) are analyzed separately in Appendix B.

The subsequent subsections detail the methodology and analysis that support these conclusions, starting with an explanation of our ranking approach.

5.1 Ranking Methodology

To produce the final summary table, we followed a two-step ranking process:

1. **Per-Cell Ranking**: For each individual experimental cell—defined by a unique combination of `(ODE_system, noise_level, derivative_order)`—we ranked the contender methods against each other based on their mean `nRMSE` (averaged across the 10 trials for that cell).

2. **Averaging Ranks**: We then calculated the final "Avg. Rank" for each method by averaging these per-cell ranks across two distinct regimes:

    - **Low-Noise Regime**: Averaged across noise levels below 1% (`1e-8`, `1e-6`, `1e-4`, and `1e-3`).
    - **High-Noise Regime**: Averaged across noise levels of 1% and above (`0.01` and `0.02`).

This methodology ensures that the final rank is a robust measure of a method's performance across a wide variety of conditions, and it prevents a single outlier or a particularly favorable test case from dominating the results.

5.2 Performance Degradation by Derivative Order

A clear pattern emerging from the data is the systematic degradation of performance as the derivative order increases. This is an expected consequence of the ill-posed nature of differentiation. We can characterize this trend in several phases:

- **Orders 0-2 (Low-Order):** In this regime, most contender methods perform well, and the performance differences between them are relatively modest, particularly in low-noise scenarios. The task of smoothing or finding a first or second derivative is not challenging enough to create significant separation between the top methods.

- **Orders 3-5 (Mid-Order):** This is the regime where a clear separation emerges. The task becomes significantly more challenging, and methods without sophisticated noise handling begin to struggle. The performance of GPR and the stronger spectral methods remains high, while simpler spline- and filter-based methods see a substantial drop in accuracy.

- **Orders 6-7 (High-Order):** This regime represents an extreme challenge. Only a very small subset of methods, primarily GPR, are able to produce a usable estimate, and even their errors are significant. For most other methods, the error in this regime constitutes a catastrophic failure. Derivative order is clearly the dominant factor in the difficulty of the estimation problem.

Figure 4 provides a comprehensive visual illustration of this systematic degradation. The figure shows performance across all eight derivative orders for the top seven methods, clearly demonstrating how error increases with order and how different methods respond to increasing noise levels at each order.

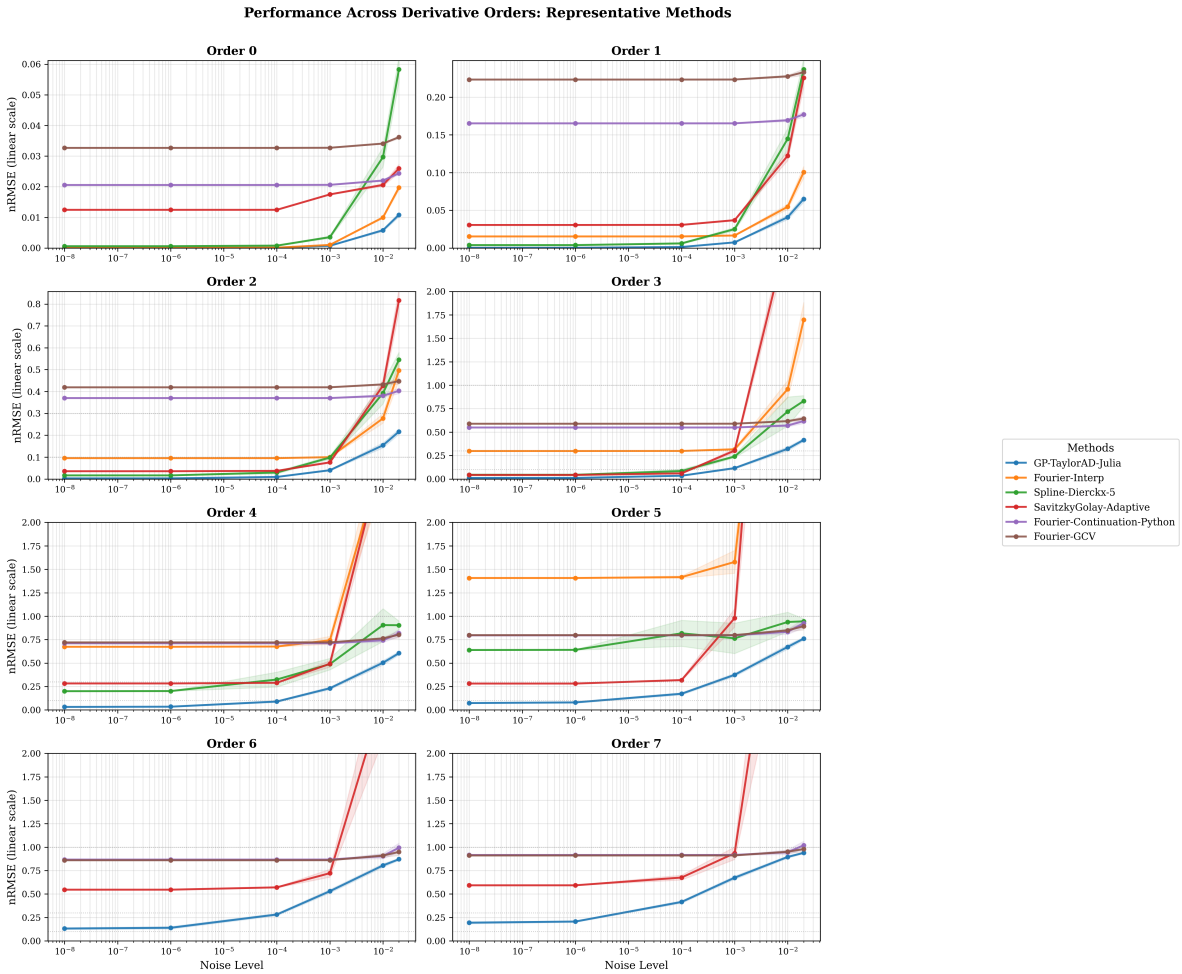

Figure 4: **Performance Across All Derivative Orders.** This 4×2 grid shows nRMSE vs noise level for 6 representative methods at each derivative order (0–7). Each panel illustrates the systematic degradation of performance as order increases. Note how GP-TaylorAD-Julia (top performer) maintains relatively stable performance across all orders, while other methods show dramatic degradation beyond order 3.

6 Conclusion

This comprehensive study evaluated a wide array of numerical methods for the estimation of high-order derivatives from noisy data. After a detailed investigation that included a multi-stage filtering of methods and a deep dive into implementation details, our findings are clear and decisive.

6.1 Summary of Key Findings

- **Gaussian Process Regression (GPR) is the most robust and accurate method overall.** GPR methods consistently occupy the top ranks in both low- and high-noise regimes, making them the most reliable choice for general-purpose derivative estimation.
- **The optimal method depends on the use case.** While GPR is the best all-arounder, splines like `Spline-Dierckx-5` offer excellent precision for low-noise data, while spectral meth-

ods provide moderate accuracy at interactive speeds. For real-time applications requiring sub-10ms response, `Savitzky-Golay` remains the most practical choice despite lower accuracy.

- **Derivative order is the dominant difficulty factor.** Performance degrades systematically with increasing order across all methods. The problem becomes significantly more challenging beyond order 3, and only a handful of methods produce usable results at orders 6 or 7.
- **Implementation quality is a critical method characteristic.** Our study found significant performance differences between different software packages implementing the same underlying algorithm, highlighting that practitioners must consider the quality of a specific implementation, not just the theoretical method.

The primary recommendation of this work is that for practitioners who require accurate high-order derivatives from real-world, noisy signals, Gaussian Process Regression is the most reliable and effective starting point.

6.2 Computational Cost and Method Selection

Computational costs vary significantly across methods, from milliseconds for filters to seconds for Gaussian Processes. The smoothing/fitting step dominates timing; derivative order has minimal impact on computational cost. Table 2 shows representative methods spanning the speed-accuracy spectrum.

Table 2: Computational Cost vs Accuracy Trade-Off

Method	Mean Time (s)	Mean nRMSE	Speedup vs GP
SavitzkyGolay-Fixed	0.0011	0.876	1652.9×
ButterworthSpline_Python	0.0040	0.384	447.3×
Spline-Dierckx-5	0.0088	0.315	202.3×
Fourier-Continuation-Python	0.0157	0.565	113.2×
Fourier-GCV	0.0432	0.583	41.1×
Fourier-Adaptive-Julia	0.8446	0.641	2.1×
GP-TaylorAD-Julia	1.7780	0.233	1.0× (baseline)
GP-RBF-Python	23.9346	0.237	0.1×

Speed-Accuracy Tradeoff:
Figure 5 illustrates the speed-accuracy tradeoff at noise level 0.0001 and derivative order 3. Remarkably, only four methods define the Pareto frontier:

- `SavitzkyGolay-Fixed` (1.1 ms): Fastest method on the frontier (nRMSE = 0.52)
- `ButterworthSpline_Python` (4.0 ms): Fast filtering approach (nRMSE = 0.38)
- `Spline-Dierckx-5` (8.8 ms): High-degree spline interpolation (nRMSE = 0.32)
- `GP-TaylorAD-Julia` (1.78 s): Best accuracy, three orders of magnitude slower (nRMSE = 0.14)

The large gap between sub-millisecond filters and second-scale GP methods suggests an opportunity for methods that better balance speed and accuracy in the 10–100 ms range.

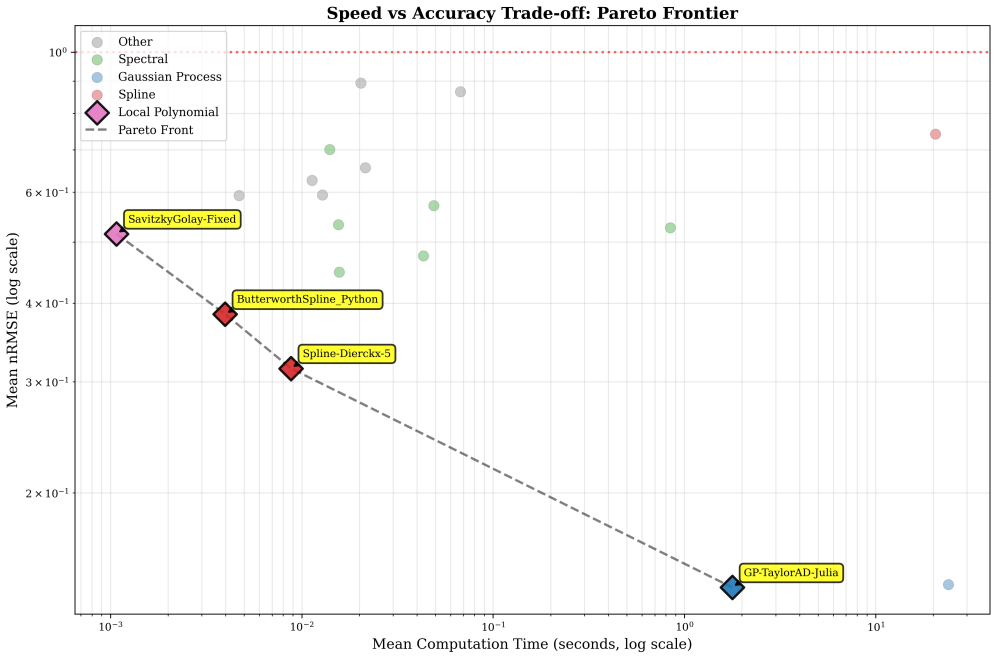

Figure 5: **Speed-accuracy tradeoff for derivative estimation methods.** Log-log plot showing normalized RMSE versus computation time for all methods at noise level 0.0001 and derivative order 3. Methods are colored by category. The dashed line connects the four methods on the Pareto frontier, showing the tradeoff between fast filters and accurate GP methods.

Practical Recommendations:

When accuracy is paramount, `GP-TaylorAD-Julia` consistently delivers the best results across all derivative orders. For applications with timing constraints:

- **Real-time ($<$ 10 ms):** `SavitzkyGolay-Fixed`, `ButterworthSpline_Python`, or `Spline-Dierckx-5` provide acceptable accuracy for orders 0–3.
- **Interactive (10–100 ms):** `Fourier-GCV` offers good accuracy/speed balance.
- **Offline analysis:** `GP-TaylorAD-Julia` (1.8 s) remains practical for batch processing.

Note that GP methods scale as $O(N^3)$ with data size, while spectral methods scale as $O(N \log N)$, making the latter preferable for very large datasets.

6.3 Future Work

This benchmark, while comprehensive, is not exhaustive. Several avenues for future research are immediately apparent:

1. **Testing on Diverse Signals:** Our study used ODEs that produce smooth, analytic signals. Future work should include testing on more challenging signals, such as those with discontinuities, sharp peaks, or chaotic behavior.
2. **Evaluating Alternative Noise Models:** The real world is not always Gaussian. A valuable extension would be to evaluate method performance under different noise models, such as multiplicative, Poisson, or heavy-tailed noise.

3. **Larger-Scale Problems:** Our study was limited to a modest number of data points. Investigating how method performance, particularly computational cost, scales to much larger datasets ($N > 1000$) would be of great practical interest.

6.4 Broader Implications: The Case for a Composable, Differentiable Ecosystem

Our findings also underscore a broader trend in scientific computing: the immense value of composable and differentiable software packages. The "Approximant-AD" framework is only possible when libraries for data modeling (e.g., Gaussian Processes) can seamlessly pass their results to libraries for automatic differentiation.

While not all numerical packages are readily differentiable out-of-the-box, our experience suggests that many can be adapted with modest effort. We encourage researchers and developers to prioritize differentiability in their own software and to contribute upstream to make foundational libraries in the ecosystem compatible with AD frameworks. Such efforts create a virtuous cycle, unlocking powerful new hybrid methodologies that benefit the entire scientific community, far beyond the immediate application of derivative estimation.

A Complete Method Catalog

This appendix provides comprehensive documentation for the 26 contender methods that demonstrated full coverage across derivative orders 0–5. Methods are organized by algorithmic family, with standardized naming conventions and complete implementation details.

A.1 Method Summary Table

Table 3 presents all contender methods with their key characteristics. Methods are grouped by algorithmic category to facilitate comparison within families.

Table 3: Complete catalog of 26 contender methods evaluated in this study. All methods support derivative orders 0–5 across all noise levels tested.

Method	Category	Brief Description	Complexity
Gaussian Process Methods (2)			
GP-Julia-AD	Gaussian Process	GP regression with Taylor-mode AD for efficient high-order derivatives; uses RBF kernel with MLE hyperparameter optimization	$O(n^3)$
GP-RBF-Python	Gaussian Process	Standard scikit-learn GP with RBF kernel; derivatives via finite differences of posterior mean	$O(n^3)$
Local Polynomial Methods (5)			
SavitzkyGolay-Fixed-Julia	Local Polynomial	Classic Savitzky-Golay filter with fixed window size (n/4); fast but may be suboptimal	$O(n)$
SavitzkyGolay-Adaptive-Julia	Local Polynomial	Adaptive window selection based on wavelet noise estimation; balances smoothing and preservation	$O(n)$

Continued on next page...

Table 3 – continued from previous page

Method	Category	Brief Description	Complexity
SavitzkyGolay-Pkg-Fixed	Local Polynomial	SavitzkyGolay.jl package implementation with fixed parameters	$O(n)$
SavitzkyGolay-Pkg-Hybrid	Local Polynomial	Hybrid approach combining fixed base window with adaptive adjustments	$O(n)$
SavitzkyGolay-Pkg-Adaptive	Local Polynomial	Fully adaptive using local signal characteristics and noise estimates	$O(n)$
Spectral Methods (9)			
Fourier-Interp-Julia	Spectral	FFT-based differentiation with fixed 40% frequency cutoff; assumes periodicity	$O(n \log n)$
Fourier-Adaptive-Julia	Spectral	Adaptive frequency cutoff based on estimated SNR; robust to varying noise levels	$O(n \log n)$
Fourier-Adaptive-Python	Spectral	Python implementation with NumPy/SciPy; adaptive filtering based on noise	$O(n \log n)$
Fourier-GCV	Spectral	Uses Generalized Cross-Validation to select optimal number of Fourier harmonics	$O(n \log n)$
Fourier-Continuation-Adaptive	Spectral	Fourier continuation for non-periodic data; reduces edge artifacts via smooth extension	$O(n \log n)$
Fourier-Basic-Python	Spectral	Simple FFT differentiation with fixed cutoff; baseline spectral method	$O(n \log n)$
Fourier-Continuation-Python	Spectral	Standard Fourier continuation implementation; handles non-periodic signals	$O(n \log n)$
Chebyshev-AICc	Spectral	Chebyshev polynomial with AICc-based degree selection; prevents overfitting	$O(n^2)$
Chebyshev-Basic-Python	Spectral	Fixed-degree Chebyshev polynomial approximation and differentiation	$O(n^2)$
Spline Methods (2)			
Dierckx-5	Spline	Degree-5 B-splines with GCV smoothing parameter selection; FORTRAN FITPACK	$O(n)$
GSS	Spline	Generalized smoothing splines with automatic smoothness selection	$O(n^3)$
Rational Approximation Methods (3)			
AAA-LowPrec	Rational	Adaptive Antoulas-Anderson with relaxed tolerance for speed; may exhibit instability	$O(n^2)$
AAA-Adaptive-Diff2	Rational	AAA with tolerance based on second-order difference noise estimation	$O(n^2)$
AAA-Adaptive-Wavelet	Rational	AAA with wavelet MAD noise estimation for adaptive tolerance selection	$O(n^2)$
PyNumDiff Package Methods (3)			
PyNumDiff-SavGol-Tuned	PyNumDiff	Savitzky-Golay with manually tuned parameters optimized for test systems	$O(n)$
PyNumDiff-Spectral-Auto	PyNumDiff	Automatic spectral method with adaptive parameter selection	$O(n \log n)$

Continued on next page...

Table 3 – continued from previous page

Method	Category	Brief Description	Complexity
PyNumDiff-Spectral-Tuned	PyNumDiff	Spectral differentiation with pre-tuned cutoff frequencies	$O(n \log n)$
Filtering Methods (2)			
Kalman-Grad-Python	Filtering	Kalman filter formulation for derivative estimation; state-space approach	$O(n)$
TVRegDiff-Python	Filtering	Total variation regularization; preserves discontinuities while smoothing	$O(n^2)$

A.2 Implementation Details

A.2.1 Package Dependencies

Julia Packages:

- `GaussianProcesses.jl` (v0.12.5): GP-Julia-AD [11]
- `TaylorDiff.jl` (v0.2.1): Taylor-mode AD for GP-Julia-AD, AAA methods [7]
- `BaryRational.jl` (v0.3.0): AAA rational approximation [19]
- `Dierckx.jl` (v0.5.2): Wrapper for FORTRAN FITPACK splines [1, 9]
- `GeneralizedSmoothingSplines.jl` (v0.1.3): GSS implementation [27]
- `FFTW.jl` (v1.5.0): Fast Fourier transforms
- `SavitzkyGolay.jl` (v0.3.1): Savitzky-Golay filter implementations [24]
- `Wavelets.jl` (v0.9.5): Wavelet transforms for noise estimation [8]

Python Packages:

- `scikit-learn` (1.3.0): Gaussian Process regression [21]
- `numpy` (1.24.3): Core numerical operations, polynomial fitting [15]
- `scipy` (1.11.1): Signal processing, optimization [26]
- `PyNumDiff` (0.1.0): Comprehensive differentiation package [25]

A.2.2 Key Implementation Choices

Gaussian Process Methods:

- All use RBF (squared exponential) kernel: $k(x, x') = \sigma_f^2 \exp(-\|x - x'\|^2/2\ell^2)$
- Hyperparameters (ℓ, σ_f, σ_n) optimized via Maximum Likelihood Estimation
- Julia implementation uses Taylor-mode AD for efficient high-order derivatives
- Python implementation tested with isotropic/anisotropic kernels and mean subtraction preprocessing; all variants yielded identical performance
- Python implementations limited by lack of Taylor-mode AD (explains 10× speed difference)

Spectral Methods:

- Differentiation via frequency domain: $\mathcal{F}\{f^{(n)}\}(k) = (ik)^n \mathcal{F}\{f\}(k)$
- Various strategies for frequency cutoff: fixed (40%), adaptive (SNR-based), or GCV
- Fourier continuation methods use smooth extension to handle non-periodic data
- Chebyshev methods use domain scaling to $[-1, 1]$ before fitting

Local Polynomial Methods:

- Savitzky-Golay filters use polynomial degree $d = \min(7, n + 2)$ for n-th derivative
- Window size selection: fixed ($w = n/4$) or adaptive based on noise estimation
- Boundary handling via asymmetric filters near endpoints

Rational Approximation:

- AAA algorithm constructs barycentric rational interpolant: $r(z) = \sum_i w_i f_i/(z - z_i) / \sum_i w_i/(z - z_i)$
- Support points selected greedily based on maximum residual
- Tolerance setting critical: too tight causes overfitting, too loose underfits
- Known instability at high derivative orders despite mathematical correctness

A.2.3 Computational Complexity Notes

- $O(n)$ **methods:** Savitzky-Golay filters, finite differences, splines (excluding GSS)
- $O(n \log n)$ **methods:** All Fourier/FFT-based spectral methods
- $O(n^2)$ **methods:** Chebyshev polynomials, AAA rational approximation, TV regularization
- $O(n^3)$ **methods:** Gaussian Processes (matrix inversion), GSS (dense linear systems)

The computational complexity is essentially independent of derivative order for all methods, as the dominant cost is in the smoothing/approximation step rather than the differentiation itself.

A.3 Parameter Selection and Tuning

A.3.1 Adaptive Parameter Selection

Several methods employ adaptive strategies to automatically select parameters based on data characteristics:

Noise Estimation Methods:

- **Wavelet MAD:** $\hat{\sigma} = \text{MAD}(\text{HF}_{\text{wavelet}})/0.6745$ using Daubechies-4 wavelets [10].
- **Second-order differences:** $\hat{\sigma} = \sqrt{\sum(y_{i+1} - 2y_i + y_{i-1})^2/6(n-2)}$ [13].

Model Selection Criteria:

- **AICc:** $n \log(\text{RSS}/n) + 2p + 2p(p+1)/(n-p-1)$ for p parameters
- **GCV:** $n \cdot \text{RSS}/(n - \text{df})^2$ where df = effective degrees of freedom
- **MLE:** Maximum likelihood for GP hyperparameters via L-BFGS-B optimization

A.3.2 Fixed Parameters

Some methods use fixed parameters determined through preliminary experiments:

- Fourier-Interp: 40% frequency cutoff
- AAA-LowPrec: tolerance = 10^{-13}
- Savitzky-Golay-Fixed: window = $n/4$

A.4 Method Selection Guidelines

Based on our comprehensive evaluation, we provide the following guidance for method selection:

For highest accuracy (orders 0–5):

- **Low noise ($< 0.01\%$):** Dierckx-5 or GP-Julia-AD
- **High noise ($\geq 1\%$):** GP-Julia-AD consistently best
- **Unknown noise:** GP methods with automatic noise estimation

For computational efficiency:

- **Real-time ($<$ 10ms):** Savitzky-Golay variants
- **Interactive ($<$ 100ms):** Spectral methods (Fourier-GCV, Chebyshev-AICc)
- **Batch processing:** GP-Julia-AD (2s) acceptable for highest accuracy

For specific signal types:

- **Periodic/smooth:** Fourier methods excel
- **Non-periodic:** Fourier-Continuation or splines
- **Discontinuous:** TVRegDiff preserves edges (limited to order 1)

Implementation recommendations:

- Compiled implementations generally faster than interpreted equivalents, and back-ends (for linear algebra, AD) matter
- Taylor-mode AD critical for efficient high-order derivatives
- Pre-compute smoothed approximant when multiple derivatives needed

B High-Order Derivatives (Orders 6-7)

High-order derivatives (6th and 7th order) present a particularly challenging test case for numerical differentiation methods. These orders are rarely required in practice but serve as a valuable stress test to understand method behavior at the limits of numerical stability.

B.1 Challenges at High Orders

Computing 6th and 7th order derivatives numerically is fundamentally difficult due to:

- **Noise Amplification**: Each order of differentiation approximately multiplies the noise effect by a factor related to the inverse of the sampling interval, leading to exponential growth in uncertainty.
- **Numerical Precision**: Finite precision arithmetic introduces round-off errors that compound with each derivative order.
- **Method Limitations**: Many methods are theoretically limited to lower orders (e.g., low-degree polynomial approximations) or become numerically unstable at high orders.

As a result, only 24 out of 62 tested methods successfully produce valid predictions for both orders 6 and 7 across all test configurations.

B.2 Performance Across Noise Regimes

Figure 6 presents a three-panel comparison showing how the top 15 methods perform on orders 6-7 across different noise regimes. The panels reveal three key insights:

1. **Overall Performance (All Noise Levels)**: Gaussian Process methods (`GP-TaylorAD-Julia` and `GP-RBF-Python`) substantially outperform all alternatives, achieving mean nRMSE ≈ 0.5 compared to ≈ 0.9 for the next-best Fourier methods.
2. **Low-Noise Regime ($\leq 0.1\%$)**: In clean data conditions, GP methods maintain excellent performance (nRMSE $\approx 0.32 - 0.34$), demonstrating their ability to extract high-order derivatives when signal quality permits. Savitzky-Golay variants emerge as the next-best alternatives (nRMSE $\approx 0.65 - 0.79$).
3. **High-Noise Regime ($\geq 1\%$)**: At realistic noise levels, all methods struggle significantly. GP methods remain the most robust (nRMSE ≈ 0.88), but even they approach the threshold of practical utility. Fourier methods show comparable robustness (nRMSE $\approx 0.95 - 0.97$), suggesting that frequency-domain approaches may be competitive when high precision is unattainable.

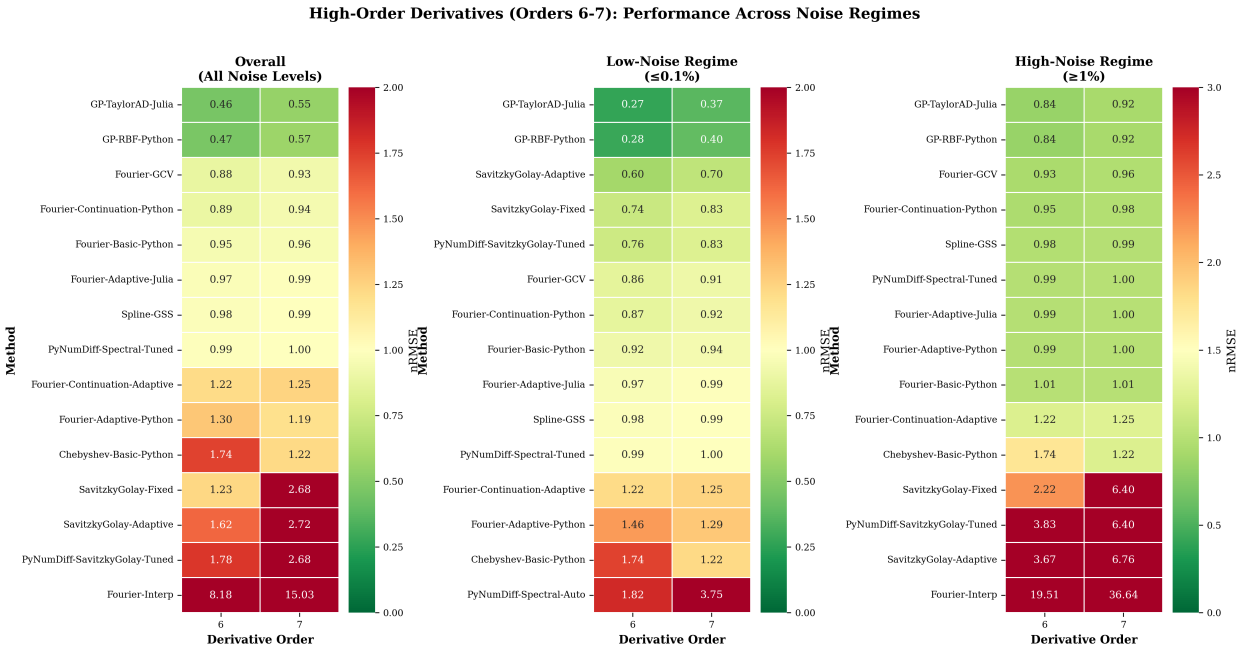

Figure 6: **High-Order Derivatives (Orders 6-7) Performance Across Noise Regimes.** Three-panel heatmap showing the top 15 methods for computing 6th and 7th order derivatives. **Left:** Overall performance averaged across all six noise levels. **Center:** Low-noise regime ($\leq 0.1\%$: noise levels 1e-8, 1e-6, 1e-4, 1e-3) where clean data permits better extraction of high-order information. **Right:** High-noise regime ($\geq 1\%$: noise levels 0.01, 0.02) representing challenging real-world conditions. The heatmap reveals that while GP methods dominate across all regimes, the absolute performance gap narrows in high-noise conditions where all methods struggle. Note the different color scale on the high-noise panel (0-3.0 vs 0-2.0) reflecting the overall degradation in performance.

B.3 Practical Recommendations

Based on this analysis, we offer the following guidance for practitioners needing high-order derivatives:

- **First choice**: `GP-TaylorAD-Julia` provides the best performance across all noise regimes and is the recommended method when computational cost is acceptable.
- **Low-noise alternative**: In clean data environments ($< 0.1\%$ noise), `SavitzkyGolay-Adaptive` offers a computationally cheaper alternative with acceptable accuracy.
- **High-noise reality**: At noise levels $\geq 1\%$, computing reliable 6th and 7th order derivatives may be fundamentally impractical. Consider whether the application truly requires these high orders, or if lower-order approximations suffice.
- **Avoid high orders when possible**: The dramatic performance degradation at orders 6-7 (compared to orders 1-5 shown in Figure 3) suggests that applications should avoid requiring these orders unless absolutely necessary.

B.4 Comparison to Primary Results

The methods ranked highly for orders 1-5 (Figure 3) generally maintain their relative performance at orders 6-7, with one notable exception: Savitzky-Golay methods (both `SavitzkyGolay-Adaptive` and `SavitzkyGolay-Fixed`) move from mid-tier performers at low orders to being among the top alternatives to GP methods at high orders in low-noise conditions. This suggests that local polynomial fitting, while not optimal overall, has specific advantages for high-order differentiation when data quality is sufficient.

Acknowledgments

We thank the developers of GaussianProcesses.jl, Dierckx.jl, FFTW.jl, and scikit-learn for their open-source implementations.

References

[1] Kyle Barbary and contributors. Dierckx.jl: A julia wrapper for the Dierckx FITPACK library. GitHub repository, 2014.

[2] Oren Bassik, Yosef Berman, Soo Go, Hoon Hong, Ilia Ilmer, Alexey Ovchinnikov, Chris Rackauckas, Pedro Soto, and Chee Yap. Robust parameter estimation for rational ordinary differential equations, 2023. arXiv preprint arXiv:2303.02159.

[3] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[4] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[5] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[6] Rick Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011:164564, 2011.

[7] JuliaDiff contributors. TaylorDiff.jl: Taylor-mode automatic differentiation for higher-order derivatives. GitHub repository, 2022.

[8] JuliaDSP contributors. Wavelets.jl: A julia package for fast discrete wavelet transforms and utilities. GitHub repository, 2015.

[9] Paul Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, 1993.

[10] David L. Donoho and Iain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[11] Jamie Fairbrother, Christopher Nemeth, Maxime Rischard, Johanni Brea, and Thomas Pinder. GaussianProcesses.jl: A nonparametric bayes package for the Julia language. *Journal of Statistical Software*, 102(1):1–36, 2022.

[12] Dougal Frost, Matthew Johnson, and David Meles. Efficiently computing higher-order derivatives with Taylor-Mode AD in JAX. In *ICLR 2021 Workshop: "Beyond Backpropagation: Novel Approaches to Training Deep Neural Networks"*, 2021.

[13] Theo Gasser, Lothar Sroka, and Christine Jennen-Steinmetz. Residual variance and residual pattern in nonlinear regression. *Biometrika*, 73(3):625–633, 1986.

[14] David Gottlieb and Steven A. Orszag. *Numerical Analysis of Spectral Methods: Theory and Applications*. Society for Industrial and Applied Mathematics (SIAM), 1977.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Christoph Gohlke, 'Abd A. Zaid, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[16] Ian Knowles. A comparative study of methods for numerical differentiation of noisy data. In *Electronic Journal of Differential Equations, Conference 21*, pages 17–31, 2009.

[17] Kim Listmann, Andreas Kugi, and Jürgen Böhm. A comparison of methods for higher-order numerical differentiation. In *2013 European Control Conference (ECC)*, pages 3676–3681, 2013.

[18] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

[19] Colin MacDonald and contributors. BaryRational.jl: Barycentric rational approximation and interpolation. GitHub repository, 2018.

[20] Yuji Nakatsukasa, Olivier Sète, and Lloyd N. Trefethen. The AAA algorithm for rational approximation. *SIAM Journal on Scientific Computing*, 40(3):A1494–A1522, 2018.

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[23] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[24] Abraham Savitzky and Marcel J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[25] Floris Van Breugel, Yuying Liu, Bingni W. Brunton, and J. Nathan Kutz. PyNumDiff: A Python package for numerical differentiation of noisy time-series data. *Journal of Open Source Software*, 7(71):4078, 2022.

[26] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Martin Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, , et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

[27] Grace Wahba. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics (SIAM), 1990.

[28] J. A. Walker. Estimating velocities and accelerations of animal locomotion: a simulation experiment comparing numerical differentiation algorithms. *The Journal of Experimental Biology*, 201(Pt 7):981–995, 1998.

C Detailed Results Tables

This appendix contains detailed tables of the mean normalized Root-Mean-Square Error (nRMSE) for all contender methods across all noise levels, for each derivative order from 0 to 5. Each table shows performance across seven noise levels (1e-08 to 2e-02) with the mean across all noise levels in the final column.

D Reproducibility

This study was designed to be fully reproducible. All code, data, analysis scripts, and this manuscript's complete version history are available at the public Git repository: https://github.com/orebas/deriv-estimation-study.

D.1 Software Environment

The computational environment uses Julia 1.12.1 and Python 3.13.1. All package dependencies are managed by lock files (`uv.lock` for Python, `Manifest.toml` for Julia) to ensure exact package versions are preserved. A complete catalog of all methods and their dependencies is available in Appendix A.

All sources of randomness are deterministically seeded to ensure identical datasets are generated on each run. Specifically:

- Noise generation for each trial uses a fixed seed derived from the system name, noise level, and trial number
- ODE integration uses deterministic solvers with fixed tolerances
- Random parameter initializations (where applicable) use consistent seeds

D.2 Computational Workflow

The entire workflow, from running the comprehensive benchmark to generating all figures, tables, and this PDF, is fully automated. The main script `scripts/build_all.sh` executes the complete pipeline in the following stages:

1. `01_run_pilot.sh`: Quick validation test (∼1 minute)
2. `02_run_comprehensive.sh`: Full benchmark across all methods, systems, noise levels, and derivative orders (∼7-10 minutes)
3. `03_generate_figures.sh`: Generate all plots and visualizations (∼30 seconds)
4. `04_generate_tables.sh`: Generate LaTeX tables from results (∼10 seconds)

5. `05_compile_paper.sh`: Compile this manuscript (∼20 seconds)

All intermediate results are stored in the `build/` directory with timestamps, allowing verification of the complete data pipeline.

D.3 Hardware Specifications

All benchmarks were run on an AMD Ryzen 7 2700 Eight-Core Processor with 64 GB RAM, running Ubuntu 24.04.3 LTS under WSL2.

D.4 Data Availability

The complete dataset including:

- Raw prediction outputs for all 28 methods across all test configurations
- Aggregated summary statistics (`comprehensive_summary.csv`)
- All generated figures in publication-quality format
- LaTeX table sources

is preserved in the `build/results/` directory and available in the repository.

Table 4: Derivative Order 0 (function approximation): Mean nRMSE across noise levels and methods.

Method	1e-08	1e-06	1e-04	1e-03	1e-02	2e-02	Mean
GP-TaylorAD-Julia	0.00	0.00	0.00	0.00	0.01	0.01	0.00
GP-RBF-Python	0.00	0.00	0.00	0.00	0.01	0.01	0.00
PyNumDiff-PolyDiff-Auto	0.00	0.00	0.00	0.00	0.01	0.01	0.00
PyNumDiff-PolyDiff-Tuned	0.00	0.00	0.00	0.00	0.01	0.01	0.00
PyNumDiff-SavitzkyGolay-Tuned	0.00	0.00	0.00	0.00	0.01	0.01	0.00
PyNumDiff-TVRegularized-Tuned	0.00	0.00	0.00	0.00	0.01	0.02	0.00
SavitzkyGolay-Julia-Adaptive	0.00	0.00	0.00	0.00	0.01	0.01	0.00
PyNumDiff-Spectral-Auto	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-Spectral-Tuned	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-SecondOrder	0.00	0.00	0.00	0.00	0.01	0.02	0.01
Fourier-Adaptive-Julia	0.00	0.00	0.00	0.00	0.01	0.02	0.01
Fourier-Adaptive-Python	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-FourthOrder	0.00	0.00	0.00	0.00	0.01	0.02	0.01
Fourier-Interp	0.00	0.00	0.00	0.00	0.01	0.02	0.01
FiniteDiff-Central	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-FirstOrder	0.00	0.00	0.00	0.00	0.01	0.02	0.01
TVRegDiff-Python	0.00	0.00	0.00	0.00	0.01	0.02	0.01
TVRegDiff-Julia	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-TV-Jerk	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-Spline-Auto	0.00	0.00	0.00	0.00	0.01	0.01	0.01
SavitzkyGolay-Julia-Hybrid	0.00	0.00	0.00	0.00	0.01	0.01	0.01
PyNumDiff-TV-Acceleration	0.00	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-Butter-Auto	0.01	0.00	0.00	0.00	0.01	0.02	0.01
PyNumDiff-Kalman-Auto	0.01	0.01	0.01	0.01	0.01	0.01	0.01
PyNumDiff-SavitzkyGolay-Auto	0.02	0.01	0.00	0.00	0.01	0.01	0.01
PyNumDiff-Gaussian-Auto	0.01	0.01	0.01	0.01	0.01	0.02	0.01
SavitzkyGolay-Julia-Fixed	0.01	0.01	0.01	0.01	0.01	0.02	0.01
PyNumDiff-Friedrichs-Auto	0.01	0.01	0.01	0.01	0.01	0.02	0.01
PyNumDiff-TV-Velocity	0.01	0.01	0.01	0.01	0.01	0.02	0.01
RKHS_Spline_m2_Python	0.01	0.01	0.01	0.01	0.01	0.02	0.01
PyNumDiff-TVRegularized-Auto	0.02	0.02	0.02	0.02	0.01	0.01	0.01
PyNumDiff-Butter-Tuned	0.01	0.01	0.01	0.01	0.01	0.02	0.01
ButterworthSpline_Python	0.01	0.01	0.01	0.01	0.01	0.02	0.01
PyNumDiff-MedianDiff-Tuned	0.01	0.01	0.01	0.01	0.02	0.02	0.01
Spline-Dierckx-5	0.00	0.00	0.00	0.00	0.03	0.06	0.02
SavitzkyGolay-Adaptive	0.01	0.01	0.01	0.02	0.02	0.03	0.02
Fourier-Continuation-Adaptive	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Fourier-Continuation-Python	0.02	0.02	0.02	0.02	0.02	0.02	0.02
PyNumDiff-Spline-Tuned	0.03	0.03	0.03	0.03	0.03	0.03	0.03
PyNumDiff-Gaussian-Tuned	0.03	0.03	0.03	0.03	0.03	0.03	0.03
SavitzkyGolay-Fixed	0.03	0.03	0.03	0.03	0.03	0.03	0.03
Fourier-Basic-Python	0.03	0.03	0.03	0.03	0.03	0.03	0.03
Fourier-GCV	0.03	0.03	0.03	0.03	0.03	0.04	0.03
PyNumDiff-Friedrichs-Tuned	0.04	0.04	0.04	0.04	0.04	0.04	0.04
PyNumDiff-MedianDiff-Auto	0.03	0.03	0.03	0.03	0.04	0.04	0.04
AAA-LowTol	0.04	0.04	0.03	0.04	0.14	0.05	0.06
Chebyshev-AICc	0.06	0.06	0.06	0.06	0.06	0.06	0.06
Butterworth_Python	0.06	0.06	0.06	0.06	0.06	0.06	0.06
AAA-Python-Adaptive-Wavelet	0.00	0.00	0.00	0.01	0.20	0.20	0.07
Whittaker_m2_Python	0.07	0.07	0.07	0.07	0.07	0.07	0.07

Table 5: Derivative Order 1 (first derivative): Mean nRMSE across noise levels and methods.

Method	1e-08	1e-06	1e-04	1e-03	1e-02	2e-02	Mean
GP-TaylorAD-Julia	0.00	0.00	0.00	0.01	0.04	0.06	0.02
GP-RBF-Python	0.00	0.00	0.00	0.01	0.04	0.06	0.02
PyNumDiff-SavitzkyGolay-Tuned	0.01	0.01	0.01	0.01	0.05	0.09	0.03
PyNumDiff-PolyDiff-Auto	0.01	0.01	0.01	0.01	0.05	0.09	0.03
PyNumDiff-PolyDiff-Tuned	0.02	0.02	0.02	0.02	0.05	0.09	0.03
Fourier-Interp	0.02	0.02	0.02	0.02	0.05	0.10	0.04
PyNumDiff-TVRegularized-Tuned	0.02	0.02	0.02	0.02	0.06	0.13	0.04
PyNumDiff-SecondOrder	0.01	0.01	0.01	0.02	0.08	0.15	0.05
FiniteDiff-Central	0.01	0.01	0.01	0.02	0.08	0.15	0.05
PyNumDiff-TV-Jerk	0.03	0.03	0.03	0.03	0.05	0.12	0.05
PyNumDiff-Spline-Auto	0.02	0.02	0.02	0.02	0.06	0.16	0.05
SavitzkyGolay-Julia-Fixed	0.02	0.02	0.02	0.02	0.09	0.16	0.05
PyNumDiff-FourthOrder	0.00	0.00	0.00	0.01	0.10	0.21	0.05
SavitzkyGolay-Julia-Hybrid	0.01	0.01	0.01	0.02	0.10	0.19	0.06
PyNumDiff-SavitzkyGolay-Auto	0.04	0.02	0.01	0.01	0.07	0.20	0.06
PyNumDiff-Butter-Auto	0.05	0.05	0.05	0.05	0.05	0.12	0.06
PyNumDiff-TV-Acceleration	0.05	0.05	0.05	0.05	0.05	0.12	0.06
SavitzkyGolay-Fixed	0.05	0.05	0.05	0.05	0.07	0.11	0.06
SavitzkyGolay-Julia-Adaptive	0.01	0.01	0.01	0.01	0.12	0.23	0.06
PyNumDiff-Kalman-Auto	0.05	0.05	0.05	0.05	0.05	0.14	0.07
Spline-Dierckx-5	0.00	0.00	0.01	0.03	0.14	0.24	0.07
PyNumDiff-Gaussian-Auto	0.05	0.05	0.05	0.06	0.08	0.15	0.07
PyNumDiff-Friedrichs-Auto	0.06	0.06	0.06	0.06	0.08	0.15	0.08
SavitzkyGolay-Adaptive	0.03	0.03	0.03	0.04	0.12	0.23	0.08
ButterworthSpline_Python	0.08	0.08	0.08	0.08	0.08	0.09	0.08
RKHS_Spline_m2_Python	0.06	0.06	0.06	0.06	0.04	0.23	0.09
PyNumDiff-TV-Velocity	0.09	0.09	0.09	0.09	0.05	0.12	0.09
PyNumDiff-Butter-Tuned	0.09	0.09	0.09	0.09	0.09	0.10	0.09
PyNumDiff-MedianDiff-Tuned	0.08	0.08	0.08	0.08	0.11	0.15	0.10
PyNumDiff-TVRegularized-Auto	0.12	0.12	0.12	0.12	0.05	0.09	0.11
PyNumDiff-Gaussian-Tuned	0.12	0.12	0.12	0.12	0.12	0.13	0.12
PyNumDiff-Spline-Tuned	0.13	0.13	0.13	0.13	0.13	0.15	0.13
PyNumDiff-Friedrichs-Tuned	0.14	0.14	0.14	0.14	0.14	0.15	0.14
PyNumDiff-FirstOrder	0.11	0.11	0.11	0.11	0.19	0.32	0.16
Fourier-Continuation-Adaptive	0.15	0.15	0.15	0.15	0.16	0.16	0.16
PyNumDiff-Spectral-Auto	0.14	0.14	0.14	0.15	0.13	0.25	0.16
Fourier-Continuation-Python	0.17	0.17	0.17	0.17	0.17	0.18	0.17
PyNumDiff-MedianDiff-Auto	0.16	0.16	0.16	0.16	0.18	0.21	0.17
Fourier-Adaptive-Julia	0.13	0.13	0.13	0.21	0.29	0.31	0.20
Fourier-GCV	0.22	0.22	0.22	0.22	0.23	0.23	0.23
TVRegDiff-Julia	0.22	0.22	0.22	0.22	0.23	0.25	0.23
TVRegDiff-Python	0.23	0.22	0.23	0.23	0.23	0.25	0.23
Fourier-Basic-Python	0.23	0.23	0.23	0.23	0.24	0.24	0.23
Fourier-Adaptive-Python	0.19	0.19	0.20	0.26	0.35	0.36	0.26
Whittaker_m2_Python	0.27	0.27	0.27	0.27	0.27	0.27	0.27
Butterworth_Python	0.28	0.28	0.28	0.28	0.28	0.28	0.28
PyNumDiff-MeanDiff-Tuned	0.28	0.28	0.28	0.28	0.28	0.28	0.28
Chebyshev-AICc	0.31	0.31	0.31	0.31	0.31	0.31	0.31
PyNumDiff-MeanDiff-Auto	0.38	0.38	0.38	0.38	0.38	0.38	0.38
PyNumDiff-Spectral-Tuned	0.39	0.39	0.39	0.39	0.39	0.39	0.39

Table 6: Derivative Order 2 (second derivative): Mean nRMSE across noise levels and methods.

Method	1e-08	1e-06	1e-04	1e-03	1e-02	2e-02	Mean
GP-TaylorAD-Julia	0.00	0.00	0.01	0.04	0.15	0.22	0.07
GP-RBF-Python	0.00	0.00	0.01	0.04	0.15	0.22	0.07
PyNumDiff-SavitzkyGolay-Tuned	0.07	0.07	0.07	0.08	0.21	0.34	0.14
SavitzkyGolay-Fixed	0.10	0.10	0.10	0.10	0.20	0.32	0.15
Spline-Dierckx-5	0.02	0.02	0.03	0.10	0.39	0.54	0.18
Fourier-Interp	0.10	0.10	0.10	0.10	0.28	0.50	0.19
SavitzkyGolay-Julia-Fixed	0.10	0.10	0.10	0.12	0.36	0.63	0.24
SavitzkyGolay-Adaptive	0.04	0.04	0.04	0.08	0.43	0.82	0.24
ButterworthSpline_Python	0.23	0.23	0.23	0.23	0.24	0.26	0.24
SavitzkyGolay-Julia-Hybrid	0.05	0.05	0.05	0.08	0.43	0.82	0.25
SavitzkyGolay-Julia-Adaptive	0.02	0.02	0.02	0.06	0.51	0.99	0.27
PyNumDiff-SavitzkyGolay-Auto	0.11	0.08	0.06	0.07	0.32	1.02	0.28
Fourier-Continuation-Python	0.37	0.37	0.37	0.37	0.38	0.40	0.38
TVRegDiff-Python	0.28	0.28	0.33	0.33	0.52	0.71	0.41
Fourier-GCV	0.42	0.42	0.42	0.42	0.43	0.45	0.43
Fourier-Continuation-Adaptive	0.47	0.47	0.47	0.47	0.47	0.47	0.47
Fourier-Adaptive-Julia	0.36	0.36	0.36	0.51	0.59	0.62	0.47
Fourier-Basic-Python	0.46	0.46	0.46	0.46	0.48	0.50	0.47
Whittaker_m2_Python	0.54	0.54	0.54	0.54	0.54	0.54	0.54
Butterworth_Python	0.58	0.58	0.58	0.58	0.58	0.58	0.58
RKHS_Spline_m2_Python	0.20	0.20	0.19	0.20	0.14	2.58	0.58
Fourier-Adaptive-Python	0.63	0.63	0.57	0.58	0.63	0.64	0.61
PyNumDiff-Spectral-Tuned	0.65	0.65	0.65	0.65	0.65	0.65	0.65
PyNumDiff-Spectral-Auto	0.30	0.30	0.30	0.31	0.81	2.14	0.69
Spline-GSS	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Chebyshev-AICc	0.90	0.90	0.90	0.90	0.90	0.93	0.90
Kalman-Gradient	0.96	0.96	0.96	0.96	0.96	0.96	0.96
SVR_Python	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Chebyshev-Basic-Python	1.38	1.38	1.38	1.38	1.38	1.38	1.38
AAA-Adaptive-Diff2	0.18	0.25	1.01	268	597	144	168
AAA-Adaptive-Wavelet	0.02	0.02	1.02	267	>1e3	129	264
AAA-Python-Adaptive-Diff2	0.18	0.25	1.01	268	598	>1e3	>1e3
AAA-Python-Adaptive-Wavelet	0.02	0.02	1.02	267	>1e3	>1e3	>1e3
AAA-LowTol	0.17	0.17	0.18	1.70	>1e3	>1e3	>1e3

Table 7: Derivative Order 3 (third derivative): Mean nRMSE across noise levels and methods.

Method	1e-08	1e-06	1e-04	1e-03	1e-02	2e-02	Mean
GP-TaylorAD-Julia	0.01	0.01	0.03	0.12	0.32	0.42	0.15
GP-RBF-Python	0.01	0.01	0.03	0.12	0.32	0.42	0.15
Spline-Dierckx-5	0.04	0.04	0.08	0.24	0.72	0.83	0.33
ButterworthSpline_Python	0.39	0.39	0.39	0.39	0.43	0.50	0.42
SavitzkyGolay-Fixed	0.14	0.14	0.14	0.18	0.69	1.27	0.42
PyNumDiff-SavitzkyGolay-Tuned	0.13	0.13	0.13	0.18	0.87	1.56	0.50
Fourier-Continuation-Python	0.55	0.55	0.55	0.55	0.57	0.62	0.56
Fourier-GCV	0.59	0.59	0.59	0.59	0.62	0.65	0.60
Fourier-Interp	0.30	0.30	0.30	0.32	0.96	1.70	0.64
Fourier-Basic-Python	0.67	0.67	0.67	0.67	0.70	0.74	0.69
Fourier-Adaptive-Julia	0.61	0.61	0.61	0.75	0.80	0.82	0.70
Fourier-Continuation-Adaptive	0.71	0.71	0.71	0.71	0.71	0.72	0.71
Whittaker_m2_Python	0.72	0.72	0.72	0.72	0.72	0.72	0.72
Butterworth_Python	0.78	0.78	0.78	0.78	0.78	0.78	0.78
PyNumDiff-Spectral-Tuned	0.82	0.82	0.82	0.82	0.82	0.82	0.82
Spline-GSS	0.85	0.85	0.85	0.85	0.85	0.86	0.85
Fourier-Adaptive-Python	0.96	0.96	0.87	0.82	0.82	0.82	0.87
Kalman-Gradient	0.99	0.99	0.99	0.99	0.99	0.99	0.99
SVR_Python	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SavitzkyGolay-Julia-Fixed	0.15	0.15	0.16	0.31	1.91	3.69	1.06
SavitzkyGolay-Adaptive	0.04	0.04	0.06	0.30	2.61	5.17	1.37
PyNumDiff-SavitzkyGolay-Auto	0.20	0.15	0.13	0.26	1.61	7.92	1.71
SavitzkyGolay-Julia-Hybrid	0.07	0.07	0.10	0.43	3.86	7.75	2.05
PyNumDiff-Spectral-Auto	0.39	0.39	0.39	0.48	1.76	9.02	2.07
SavitzkyGolay-Julia-Adaptive	0.03	0.03	0.07	0.48	4.68	9.39	2.45
RKHS_Spline_m2_Python	0.38	0.38	0.39	0.38	0.32	13	2.55
Chebyshev-Basic-Python	2.61	2.61	2.61	2.60	2.60	2.61	2.61
TVRegDiff-Python	1.08	1.08	1.27	1.73	4.96	6.52	2.77
Chebyshev-AICc	4.34	4.34	4.34	4.34	4.31	4.55	4.37
AAA-Adaptive-Diff2	0.21	2.57	390	>1e3	>1e3	>1e3	>1e3
AAA-Adaptive-Wavelet	0.27	0.11	162	>1e3	>1e3	>1e3	>1e3
AAA-LowTol	0.28	0.28	0.60	84	>1e3	>1e3	>1e3

Table 8: Derivative Order 4 (fourth derivative): Mean nRMSE across noise levels and methods.

Method	1e-08	1e-06	1e-04	1e-03	1e-02	2e-02	Mean
GP-TaylorAD-Julia	0.03	0.03	0.09	0.23	0.50	0.60	0.25
GP-RBF-Python	0.04	0.04	0.09	0.23	0.50	0.60	0.25
Spline-Dierckx-5	0.20	0.20	0.32	0.49	0.90	0.90	0.50
ButterworthSpline_Python	0.52	0.52	0.52	0.52	0.69	0.92	0.62
SavitzkyGolay-Fixed	0.35	0.35	0.35	0.39	0.97	1.60	0.67
Fourier-Continuation-Python	0.71	0.71	0.71	0.71	0.74	0.82	0.73
Fourier-GCV	0.72	0.72	0.72	0.72	0.76	0.80	0.74
Fourier-Adaptive-Julia	0.79	0.79	0.79	0.89	0.91	0.92	0.85
Fourier-Basic-Python	0.83	0.83	0.83	0.83	0.86	0.94	0.85
Whittaker_m2_Python	0.83	0.83	0.83	0.83	0.86	0.94	0.85
Butterworth_Python	0.89	0.89	0.89	0.89	0.89	0.89	0.89
Spline-GSS	0.92	0.92	0.92	0.92	0.92	0.92	0.92
PyNumDiff-Spectral-Tuned	0.92	0.92	0.92	0.92	0.92	0.92	0.92
Fourier-Continuation-Adaptive	0.93	0.93	0.93	0.93	0.94	0.95	0.94
Kalman-Gradient	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SVR_Python	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PyNumDiff-SavitzkyGolay-Tuned	0.35	0.35	0.35	0.44	1.92	3.14	1.09
Fourier-Adaptive-Python	1.51	1.51	1.30	1.01	0.92	0.92	1.20
SavitzkyGolay-Adaptive	0.28	0.28	0.29	0.49	2.73	5.09	1.53
Fourier-Interp	0.67	0.67	0.68	0.74	2.70	4.79	1.71
SavitzkyGolay-Julia-Fixed	0.28	0.28	0.29	0.58	3.75	7.22	2.07
Chebyshev-Basic-Python	2.98	2.98	2.98	2.98	2.97	2.98	2.98
PyNumDiff-Spectral-Auto	0.45	0.45	0.45	0.86	3.59	22	4.66
SavitzkyGolay-Julia-Hybrid	0.19	0.19	0.26	1.02	9.21	18	4.77
SavitzkyGolay-Julia-Adaptive	0.12	0.12	0.20	1.09	11	21	5.47
PyNumDiff-SavitzkyGolay-Auto	0.48	0.42	0.48	1.32	5.04	27	5.76
TVRegDiff-Python	2.90	2.92	3.24	4.95	15	20	8.07
Chebyshev-AICc	13	13	13	13	13	14	14
RKHS_Spline_m2_Python	0.58	0.58	0.65	0.58	0.57	175	30
AAA-Adaptive-Diff2	0.33	76	>1e3	>1e3	>1e3	>1e3	>1e3
AAA-Adaptive-Wavelet	5.97	5.27	>1e3	>1e3	>1e3	>1e3	>1e3
AAA-LowTol	0.37	0.37	3.77	>1e3	>1e3	>1e3	>1e3

Table 9: Derivative Order 5 (fifth derivative): Mean nRMSE across noise levels and methods.

Method	1e-08	1e-06	1e-04	1e-03	1e-02	2e-02	Mean
GP-TaylorAD-Julia	0.07	0.08	0.17	0.37	0.67	0.76	0.36
GP-RBF-Python	0.09	0.09	0.17	0.37	0.67	0.76	0.36
Spline-Dierckx-5	0.64	0.64	0.82	0.76	0.94	0.94	0.79
Fourier-GCV	0.80	0.80	0.80	0.80	0.85	0.89	0.82
Fourier-Continuation-Python	0.80	0.80	0.80	0.80	0.83	0.92	0.82
Fourier-Basic-Python	0.89	0.89	0.89	0.89	0.93	1.02	0.92
Fourier-Adaptive-Julia	0.90	0.90	0.90	0.96	0.97	0.97	0.93
ButterworthSpline_Python	0.67	0.67	0.67	0.68	1.17	1.76	0.94
Butterworth_Python	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Spline-GSS	0.96	0.96	0.96	0.96	0.96	0.96	0.96
PyNumDiff-Spectral-Tuned	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Kalman-Gradient	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SVR_Python	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Whittaker_m2_Python	0.90	0.90	0.90	0.91	1.23	1.75	1.10
Fourier-Continuation-Adaptive	1.13	1.13	1.13	1.13	1.14	1.15	1.14
Fourier-Adaptive-Python	1.58	1.58	1.41	1.05	0.97	0.97	1.26
SavitzkyGolay-Fixed	0.41	0.41	0.42	0.62	3.04	5.61	1.75
PyNumDiff-SavitzkyGolay-Tuned	0.41	0.41	0.42	0.68	4.02	7.06	2.17
Chebyshev-Basic-Python	2.46	2.46	2.46	2.46	2.45	2.46	2.46
Fourier-Interp	1.41	1.41	1.42	1.58	6.64	12	4.04
SavitzkyGolay-Adaptive	0.28	0.28	0.32	0.98	8.26	16	4.37
PyNumDiff-Spectral-Auto	0.60	0.60	0.60	2.04	5.70	54	11
SavitzkyGolay-Julia-Fixed	0.32	0.32	0.49	2.28	20	40	11
PyNumDiff-SavitzkyGolay-Auto	0.74	0.68	1.13	4.28	15	128	25
Chebyshev-AICc	26	26	26	26	26	27	26
SavitzkyGolay-Julia-Hybrid	0.21	0.22	0.98	7.97	79	161	42
SavitzkyGolay-Julia-Adaptive	0.12	0.12	0.97	8.85	88	180	46
TVRegDiff-Python	36	35	37	76	258	328	128
RKHS_Spline_m2_Python	0.81	0.81	1.00	0.84	0.91	>1e3	221
AAA-Adaptive-Diff2	0.60	>1e3	>1e3	>1e3	>1e3	>1e3	>1e3
AAA-Adaptive-Wavelet	139	313	>1e3	>1e3	>1e3	>1e3	>1e3
AAA-LowTol	0.45	0.45	32	>1e3	>1e3	>1e3	>1e3