big data and cognitive computing

MDPI

Article

High-Speed Scientific Computing Using Adaptive Spline Interpolation

Daniel S. Soper

Department of Information Systems & Decision Sciences, California State University, Fullerton, CA 92831, USA; dsoper@fullerton.edu

Abstract

The increasing scale of modern datasets has created a significant computational bottleneck for traditional scientific and statistical algorithms. To address this problem, the current paper describes and validates a high-performance method based on adaptive spline interpolation that can dramatically accelerate the calculation of foundational scientific and statistical functions. This is accomplished by constructing parsimonious spline models that approximate their target functions within a predefined, highly precise maximum error tolerance. The efficacy of the adaptive spline-based solutions was evaluated through benchmarking experiments that compared spline models against the widely used algorithms in the Python SciPy library for the normal, Student's t, and chi-squared cumulative distribution functions. Across 30 trials of 10 million computations each, the adaptive spline models consistently achieved a maximum absolute error of no more than 1×10^{-8} while simultaneously ranging between 7.5 and 87.4 times faster than their corresponding SciPy algorithms. All of these improvements in speed were observed to be statistically significant at $p < 0.001$. The findings establish that adaptive spline interpolation can be both highly accurate and much faster than traditional scientific and statistical algorithms, thereby offering a practical pathway to accelerate both the analysis of large datasets and the progress of scientific inquiry.

Keywords: scientific computing; computational statistics; big data; spline interpolation

Academic Editor: Fabrizio Messina

Received: 22 October 2025

Revised: 24 November 2025

Accepted: 27 November 2025

Published: 2 December 2025

Citation: Soper, D.S. High-Speed Scientific Computing Using Adaptive Spline Interpolation. *Big Data Cogn. Comput.* **2025**, *9*, 308. https://doi.org/10.3390/bdcc9120308

1. Introduction

In the modern scientific and technological landscape, the ability to perform complex calculations with exceptional speed has evolved from a matter of convenience to a matter of critical necessity. The confluence of several key factors, including the exponential growth of datasets, the increasing complexity of analytical models, and the demand for rapid, data-driven decision-making, has placed unprecedented pressure on traditional computational methods [1,2]. While this challenge can potentially be addressed through the acquisition of more expensive hardware, faster scientific and statistical calculations can also be realized through the design and use of more efficient algorithms [3,4]. Adopting a hardware-centric approach, while effective, is often not feasible for many researchers and organizations due to prohibitively high costs [5]. By contrast, innovative algorithms that perform scientific and statistical calculations more quickly than traditional algorithms can be universally adopted, thus yielding improved computational speed for all researchers, regardless of their hardware budgets [6].

The rise in "big data" has perhaps been the most significant driver of the growing gap between the computational needs of modern scientific research and the performance limitations of common hardware and software tools. Fields ranging from genomics and particle physics to finance and climate science are now routinely generating and working with datasets that contain terabytes or even petabytes of information [7–10]. Analyzing these massive volumes of data, whether for pattern recognition, hypothesis testing, or predictive modeling, requires a scale of computation that can quickly overwhelm standard statistical packages and programming libraries [11]. Moreover, the computational costs associated with processing such data can be enormous, particularly when employing computationally intensive techniques like bootstrapping, high-dimensional regression, or deep neural networks [11,12]. Hours or even days spent waiting for calculations to be completed can also bring research to a standstill, hindering the pace of discovery and innovation. This issue is further compounded by the iterative nature of scientific inquiry, in which a single question often requires hundreds or thousands of repeated calculations before a researcher can arrive at a statistically defensible conclusion. Rigorous model validation, for example, might require k-fold cross-validation, where the entire model fitting or training process must be repeated multiple times, which can easily multiply the computational burden by an order of magnitude or more [13,14]. The challenges associated with analyzing big data have, therefore, created a pressing need for both faster hardware and faster analytical algorithms.

Beyond the purely scientific realm, the need for rapid computation is also being driven by a fiercely competitive business environment. In industries such as finance, a delay of even a few milliseconds in analyzing market data can result in significant financial losses [15]. Indeed, the pressure to arrive at conclusions quickly and with a high degree of statistical confidence is critical for gaining and maintaining competitive advantage and making timely, impactful decisions [16,17]. Achieving this goal requires not only efficient algorithms but also a computational framework that can execute these algorithms with minimal latency, thereby transforming data into actionable insights as quickly as possible. Put differently, the ability to make near-real-time decisions based on statistical evidence is now a key differentiator in business, making computational speed a highly valuable strategic asset.

Broader considerations notwithstanding, the calculations for foundational statistical tests have also become computationally expensive in the era of big data. Arriving at statistically defensible conclusions for tests associated with the normal, Student's t, and chi-squared distributions, for example, typically requires numerous iterative calculations on massive datasets [11,18–20]. These challenges, of course, are not linked solely to the number of data points but are also driven by the high dimensionality of modern datasets. In a genome-wide association study, for example, thousands of t-tests or chi-squared tests may need to be performed on millions of genetic variants, with each test requiring significant computational resources [21]. While these methods have demonstrated remarkable success, their application in research and industry is often constrained by the computational time and resources that are required for robust estimation and validation. A methodology that can significantly reduce the time needed for these foundational calculations would, therefore, open up new possibilities for research, enabling scientists to explore more intricate models and perform more thorough analyses than might otherwise be feasible.

In light of the issues described above, the current paper seeks to motivate a strategic shift in scientific and statistical computing away from traditional algorithms toward those that prioritize speed and efficiency without sacrificing accuracy or statistical rigor. To do so, this paper demonstrates how a technique known as spline interpolation can be modified and adapted to perform statistical calculations much more quickly than standard algorithms to any arbitrarily chosen degree of precision. The viability of the proposed method is demonstrated through a large and rigorous set of benchmarking experiments that compare the accuracy and execution time of the adaptive spline interpolation method against widely used traditional algorithms when computing the cumulative distribution functions (CDFs) for the normal, Student's t, and chi-squared distributions. The results indicate that the adaptive spline interpolation method is not only much faster than traditional algorithms when performing these calculations, but also that the proposed method yields results that are highly accurate. The proposed method, therefore, represents a step forward in scientific and statistical computing by offering researchers a pathway to faster and more accessible analyses of large datasets, which, in turn, can accelerate the pace of scientific discovery and empower a new generation of data-driven decision-making.

The structure of this paper is organized as follows: Section 2 presents the foundational ideas and related work upon which this study is built. Section 3 describes the methods that were used to adapt the spline interpolation technique to the context of high-speed statistical computing, and the experiments that were conducted in order to compare the adaptive spline interpolation method against traditional statistical algorithms. The results of the experiments are presented and discussed in Section 4, followed by a summary of the study's findings, its limitations, and opportunities for future research in Section 5.

2. Foundations and Related Work

2.1. Function Approximation Using Polynomial Interpolation

As noted in the previous section, direct evaluation of statistical functions is often very computationally expensive. This is particularly true when those calculations require numerical integration, which is common when working with the continuous probability distributions that underlie many statistical tests [19,20,22]. Rather than evaluating such functions directly, an obvious alternative is to rely on an approximation that is less computationally expensive to evaluate than the original function. One common approach to approximating continuous functions is through the use of polynomial interpolation [23]. While the general idea of interpolation is ancient, the first systematic, algorithmic approach oriented specifically toward polynomial interpolation was developed by Sir Isaac Newton, first described in around 1676, and was later published in his acolyte James Stirling's *Methodus Differentialis* in 1730 [24]. Polynomial interpolation seeks to approximate a continuous function by constructing a unique polynomial of degree n that passes through a sample of $n + 1$ of the underlying function's data points. This method yields an explicit, computationally efficient proxy model that allows the original function's value to be estimated at any point within the range of the sample data from which the model was constructed.

While effective for low-degree polynomials, polynomial interpolation often exhibits significant instability as the degree of the polynomial increases. This issue is exemplified by Runge's phenomenon, which describes the problem of escalating oscillations at the edges of an interval when a high-degree polynomial is used to interpolate a function over a set of data points [25]. As the degree of the interpolating polynomial increases, the resulting curve can diverge significantly from the true function, thereby yielding an unreliable approximation. This phenomenon is illustrated in Figure 1 below using a polynomial interpolation of the standard normal distribution's CDF.

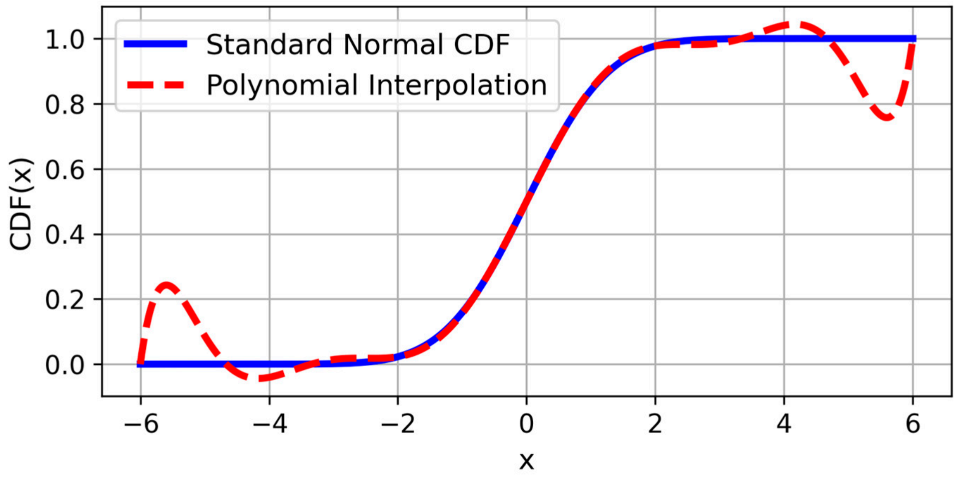

Figure 1. Runge's phenomenon occurs when using a polynomial interpolation of the standard normal distribution's cumulative distribution function.

2.2. Function Approximation Using Spline Interpolation

Evaluating a polynomial interpolation of a function such as the standard normal CDF is much more computationally efficient than evaluating the original function itself, but as Figure 1 clearly illustrates, the polynomial function would yield highly inaccurate results, particularly in the tails of the distribution. By contrast, spline interpolation offers a robust and widely adopted alternative that avoids the limitations of high-degree polynomial interpolation. Although spline interpolation has its origins in 19th-century shipbuilding, the technique was mathematically developed and formalized by Isaac Schoenberg during the middle of the 20th century [26,27]. Building on Schoenberg's foundational work, the subsequent development of B-splines and computationally efficient recursive algorithms transformed spline interpolation from a mathematical construct into a practical and highly useful tool [28,29]. Today, spline interpolation plays a central role in a wide array of applications, including capturing non-linear relationships in machine learning algorithms such as Generalized Additive Models [30], and defining the smooth, complex geometries that are essential in modern computer-aided design and computer graphics [31]. Spline interpolation has also been shown to be highly useful for accelerating numerical integration [32], the calculation of complex functions in optical image processing [33], and the calculation of complex functions in computational electromagnetics and electrical engineering [34].

In its essence, a spline is simply a function that is constructed from a piecewise series of polynomials that have been joined together at points known as knots. Put differently, rather than fitting one high-degree polynomial across an entire dataset, spline interpolation uses multiple low-degree (typically cubic) polynomials to fit smaller subintervals of the original function's data points [35]. This piecewise approach is not only computationally efficient, but it can also effectively eliminate the oscillatory behavior that characterizes Runge's phenomenon, as illustrated in Figure 2 below.

The efficacy of spline interpolation is visually evident in the case of cubic splines (i.e., piecewise, third-degree polynomials), which support specific continuity conditions at the interior knots. For a cubic spline, each cubic function is continuous, with its first and second derivatives also being continuous across the knots. These properties ensure that the piecewise segments can be joined seamlessly at the knots without abrupt changes in value, slope, or curvature. The result, as illustrated in Figure 2, is an approximation that is visually smooth and more accurate than an analogous high-degree polynomial interpolation of the original function.

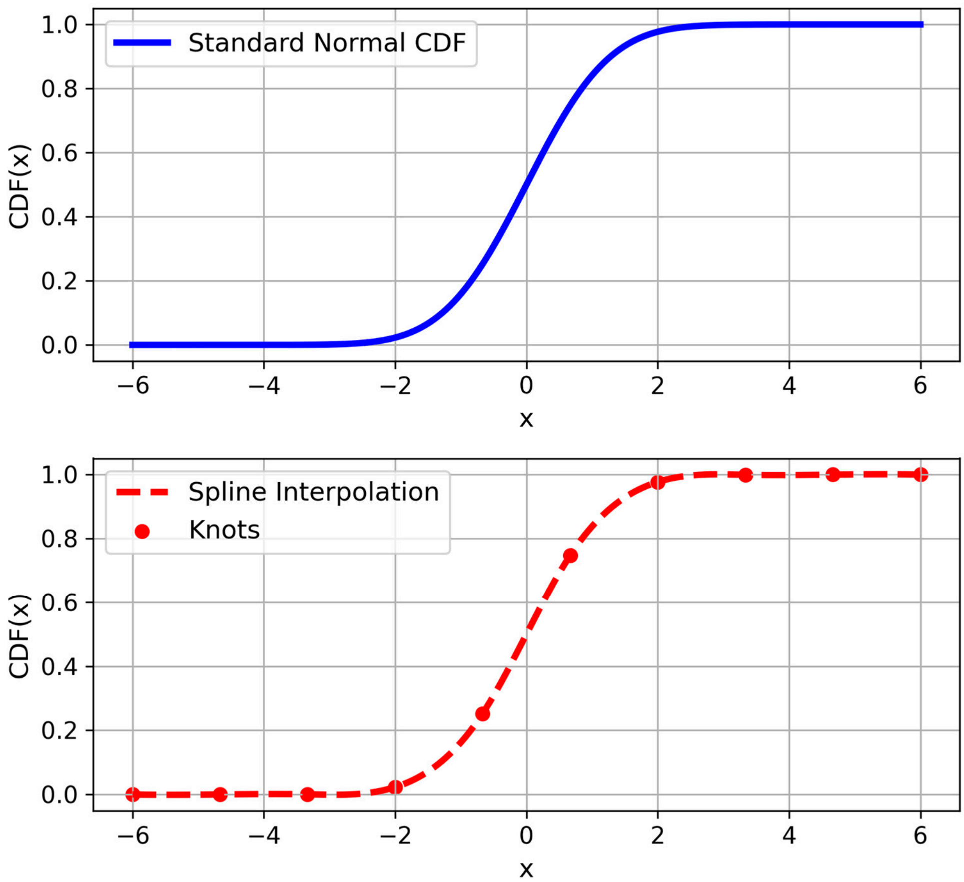

Figure 2. Spline interpolation of the standard normal distribution's CDF.

Despite these advantages and the appearance of accuracy, the question remains as to how precisely a spline interpolation model can reproduce the function it is approximating. Although the example illustrated in Figure 2 appears to be highly accurate, in reality, that particular spline interpolation model has a mean absolute error (MAE) of approximately 0.0016, with a maximum absolute error of approximately 0.0049. Considering that the spline interpolation model is approximating the standard normal CDF—and that the values being estimated are hence *p*-values—it quickly becomes evident that this specific model would be woefully inadequate if it were to be adopted as a computationally efficient substitute for the CDF itself.

An obvious strategy for improving the accuracy of a spline interpolation model is simply to increase the number of knots from which the spline is composed. Increasing the number of equidistantly spaced knots from 10 to 100 for the spline illustrated in Figure 2, for example, yields a model with a maximum absolute error of 3.14×10^{-7} and an MAE of 3.83×10^{-8}, each of which is several orders of magnitude more accurate than the original model. The problem with increasing the number of polynomials in the spline model, of course, is that each additional polynomial increases the model's complexity, thereby making the model less computationally efficient. Moreover, each polynomial in the spline will exhibit its own unique error profile, with some polynomials reproducing their corresponding intervals from the original function more accurately than other polynomials in the spline. Unless these variable error profiles are considered, the resulting spline interpolation model may be unnecessarily complex, thereby slowing the model's computational performance at runtime. What is needed, then, is a method that can construct the most parsimonious and computationally efficient spline interpolation model possible for any arbitrarily chosen degree of accuracy.

2.3. Adaptive Spline Interpolation

Constructing an optimal spline to approximate a continuous function subject to a given maximum error tolerance can be achieved through an adaptive, iterative algorithm [31]. This process is designed to minimize the number of knots, thereby ensuring that the resulting spline interpolation model is as parsimonious and computationally efficient as possible. Formally, the primary objective is to use the fewest possible knots to construct a spline $S(x)$ that approximates a given continuous function $f(x)$ over a specified closed interval $[a, b]$, such that the spline's maximum absolute error does not exceed a predefined tolerance $\varepsilon_{\max}$. This requirement can be mathematically expressed as follows:

$$\max_{a \le x \le b} |f(x) - S(x)| \le \varepsilon_{max}. \tag{1}$$

An algorithm for constructing such a spline is outlined in the following section.

3. Materials and Methods

Having related the history and foundations of spline interpolation in the previous section, this section describes the algorithms and methods that were used in the current study to construct adaptive spline interpolation models that can serve as the basis for high-speed scientific computing. This section also describes the series of benchmarking experiments that were undertaken to assess both the computational speed and the accuracy of the resulting adaptive spline interpolation models in comparison to the highly accurate and widely used algorithms that are included in the Python SciPy library.

3.1. Adaptive Spline Interpolation Algorithm

The algorithm outlined below describes the series of steps that were followed to construct the adaptive spline interpolation models that were used in the current study. While the algorithm itself is context independent, explanatory comments are also provided that are relevant to the topic of the current paper. Importantly, this algorithm assumes that the spline $S(x)$ is constructed from cubic polynomials, which ensures continuity of the polynomials' first and second derivatives [35]. It is important to note that lower-degree (i.e., linear or quadratic) polynomials could also have been used. However, cubic splines were adopted in the current study because, despite causing a minimal increase in computational complexity for each individual polynomial segment, cubic splines' superior approximation properties minimized the total number of knots that were required to satisfy the paper's stringent error tolerance, thereby yielding models that were more parsimonious and computationally efficient overall when compared to models constructed from lower-degree polynomials.

1. Specify the maximum error tolerance $\varepsilon_{\max}$.
   Comments: The maximum error tolerance, $\varepsilon_{\max}$, is the maximum acceptable absolute error between the spline $S(x)$ and the function $f(x)$ for any value of x in the interval $[a, b]$. Since the current study considers the approximation of statistical cumulative distribution functions (which generate p-values), $\varepsilon_{\max}$ was set to a very small value of 1×10^{-8} (or 0.00000001) when constructing the spline interpolation models that were used in the experiments described later in this section.
2. Identify the endpoints of the spline's interval (a and b).
   Comments: For statistical cumulative distribution functions, a can be easily identified by using the distribution's inverse CDF, $F^{-1}(x)$, such that

$$a = F^{-1}(\varepsilon_{max}). \tag{2}$$

Using this approach, any valid values of x that are less than a can be evaluated as *S(a)* without the result exceeding the maximum error tolerance $\varepsilon_{\max}$. If the CDF does not exhibit point symmetry about the mean (e.g., the chi-squared distribution), then b can also be easily identified by using the distribution's inverse CDF, such that

$$b = F^{-1}(1 - \varepsilon_{max}). \tag{3}$$

When b is defined in this way for CDFs that do not exhibit point symmetry about the mean, any valid values of x that are greater than b can be evaluated as *S(b)* without the result exceeding the maximum error tolerance $\varepsilon_{\max}$. If, however, the CDF exhibits point symmetry about the mean μ (e.g., the CDFs for the normal or Student's t distributions), such that

$$F(\mu + x) = 1 - F(\mu - x), \tag{4}$$

Then b can be identified as follows:

$$b = F^{-1}(0.5). \tag{5}$$

When b is defined in this way for CDFs with point symmetry about the mean, any valid values of x that are greater than b can be evaluated as follows without the result exceeding the maximum error tolerance $\varepsilon_{\max}$:

$$1 - S(2\mu - x). \tag{6}$$

Together, these guidelines ensure that the width of the closed interval [a, b] for the spline will be as narrow as possible, thus minimizing the number of required knots and maximizing computational efficiency.

3. Generate the training data.
   Comments: Fitting a spline model within a specified error tolerance naturally requires a set of points to use as the basis for evaluating the model's accuracy. The x-coordinates for these data points should span the closed interval [a, b], with their corresponding y-coordinates being directly computed using the original function that the spline model is being trained to approximate. Since the maximum error tolerance $\varepsilon_{\max}$ in the current study was very small, a large dataset containing one million points was used as the basis for constructing each of the spline models that are described in the experiments later in this section.
4. Define and fit the initial spline model.
   Comments: To be parsimonious, a spline model must approximate its original function within the maximum error tolerance $\varepsilon_{\max}$ using the fewest possible knots. The simplest possible spline, of course, is one that consists of a single polynomial, and this simplest model is a rational point of embarkation for the iterative spline construction process. The initial spline model should thus consist of a single cubic polynomial. Fitting a cubic polynomial requires a minimum of four points, so in addition to the two interval endpoints a and b (i.e., the boundary knots), the initial spline models used in the current study's experiments included two additional interior knots that were equidistantly spaced between a and b. These initial models were then fitted using their corresponding training datasets.
5. Iteratively add knots to the spline model until the maximum observed error falls below $\varepsilon_{\max}$.
   Comments: After defining the initial spline, the model is iteratively expanded by adding new interior knots until the maximum absolute error between the spline

S(x) and the function *f(x)* falls below the maximum error threshold ε_{max} for all values of x in the interval $[a, b]$. This is accomplished by iteratively performing the following sequence of steps:

(a) Evaluate the error function for *S(x)* using the training data.
(b) Identify the maximum absolute error and the point within the training dataset at which that maximum error value was observed.
(c) If the maximum observed error is less than ε_{max}, then no additional knots are necessary. Otherwise,

i. Add a new knot to the model at the point at which the maximum error value was observed.
ii. Fit the revised spline model using the training data.
iii. Go to step 5.a.

This approach to constructing the spline model targets the region of poorest fit, thereby ensuring a maximally efficient reduction in error for each additional knot that is added to the model [36]. This approach has also been shown to yield a final model that is vastly more efficient than could otherwise be obtained by using a uniformly spaced vector of knots [37].

6. Prune unnecessary knots from the spline model.
Comments: After a spline model that satisfies Equation (1) has been identified, unnecessary interior knots must be pruned in order to ensure that the spline model is as parsimonious as possible. This is accomplished by performing the following sequence of steps for each interior knot in the model:

(a) Remove the current knot from the spline model.
(b) Fit the revised spline model using the training data.
(c) If the revised model no longer satisfies Equation (1), then restore the current knot.
(d) Proceed to the next interior knot.

After pruning all of the unnecessary interior knots, the resulting spline interpolation model will contain the fewest possible polynomials, thus ensuring that it will be as computationally efficient as possible at runtime.

3.2. Additional Considerations

The experiments described in the following subsection consider the computational performance of adaptive spline interpolation models in comparison to traditional algorithms for calculating the CDFs of the normal, Student's t, and chi-squared distributions, all of which are extremely common in scientific and statistical computing. Given that any normal distribution can be easily converted into the standard normal distribution, it was only necessary to fit a single spline model for the standard normal distribution in order to approximate the CDF of any conceivable normal distribution. While the methods described previously for constructing parsimonious spline models can be directly applied to the CDF for the standard normal distribution, the CDFs for the Student's t and chi-squared distributions also involve a parameter for degrees of freedom (*df*), the values of which change the shapes of the CDFs. A few illustrative examples of this phenomenon are provided in Figures 3 and 4 below.

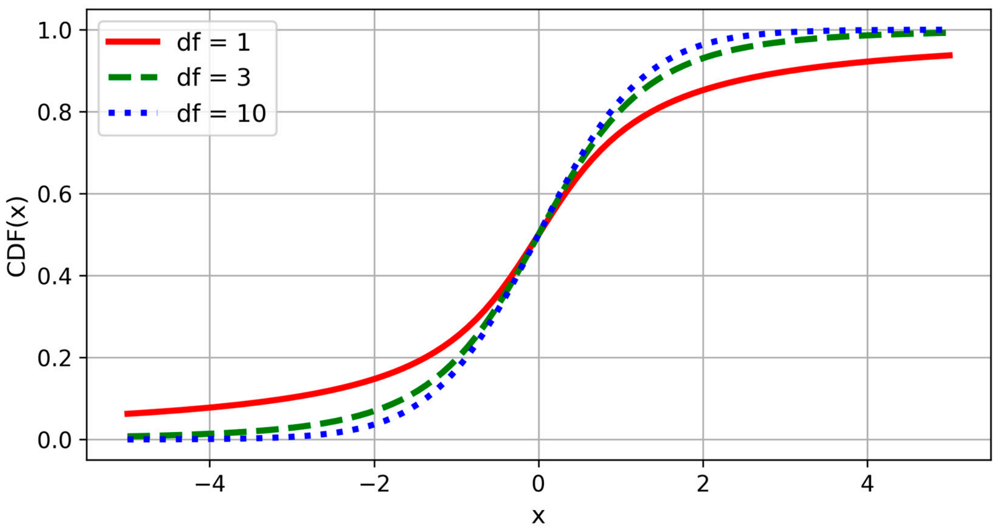

Figure 3. Student's t distribution CDF at different degrees of freedom.

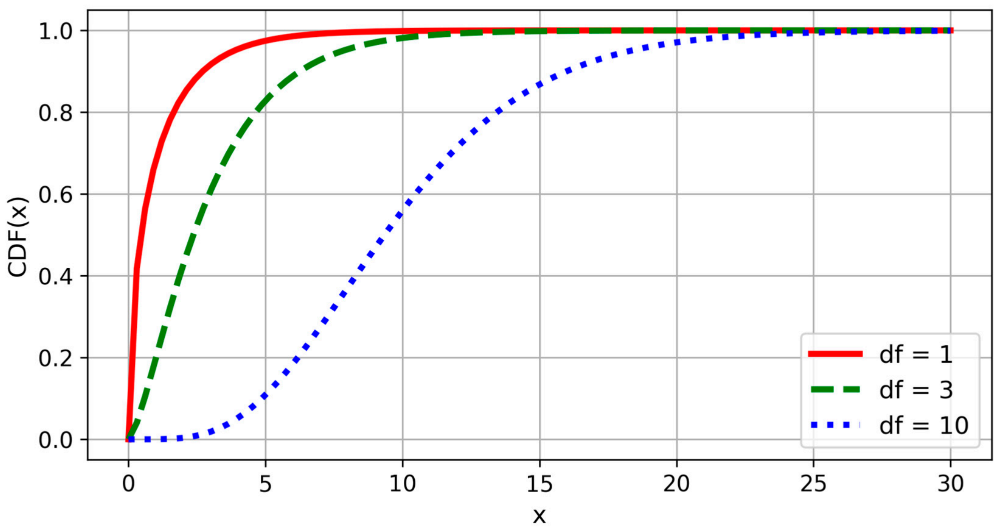

Figure 4. The chi-squared distribution CDF at different degrees of freedom.

To address the issue of the degrees of freedom affecting the shape of the Student's t and chi-squared distribution CDFs, the current study adopted the expedient of treating the function for each *df* value as a unique, independent curve. Put differently, instead of creating one complex, two-variable model, a collection of simple, highly accurate spline models was constructed, with one spline being fitted for each essential *df* value. Each spline was then stored in a dictionary that would enable any individual spline model to be quickly retrieved based on its corresponding degrees of freedom. After constructing the dictionary, any specific value of the CDF could thus be obtained by first retrieving the relevant spline model based on its degrees of freedom and then evaluating the spline at the desired value. This method is efficient because the computationally expensive work of fitting the splines only needs to be performed once, with the runtime application being reduced to a quick lookup and a simple polynomial evaluation.

There are, of course, some nuances of this approach that merit further elaboration. First, an upper limit on the degrees of freedom needs to be identified because it would not be computationally feasible to compute and store a separate spline model for each of the infinite possible *df* values. Happily, this issue can be readily resolved by noting that the CDF of Student's t distribution and the CDF of the standardized chi-squared distribution both converge on the standard normal distribution's CDF as the degrees of freedom approach infinity [38,39]. The equivalence between the standard normal CDF and the standardized chi-squared distribution CDF at high degrees of freedom also motivated the use of the standardized chi-squared CDF when training spline models in the current study, rather than fitting models for the ordinary chi-squared CDF. With this in mind, the upper limits for the degrees of freedom ($df_{\max}$) were determined by using a binary search to identify the df values above which the Student's t distribution CDF and the standardized chi-squared distribution CDF became indistinguishable from the standard normal distribution CDF within the maximum error tolerance $\varepsilon_{\max}$ [40]. Mathematically, this is reduced to finding the smallest df value that satisfies the following expression, where F represents the Student's t distribution or standardized chi-squared distribution CDF, and Φ represents the standard normal distribution CDF:

$$\max_{a \le x \le b} \left| F_{df}(x) - \Phi(x) \right| \le \varepsilon_{max}. \quad (7)$$

After performing these calculations, the degrees of freedom for Student's t distribution beyond which the CDF was indistinguishable from the standard normal CDF within $\varepsilon_{\max}$ was observed to be 15,822,999, with this value being adopted as $df_{\max}$ for the Student's t solution. For the standardized chi-squared distribution, however, the degrees of freedom beyond which the CDF was observed to be indistinguishable from the standard normal CDF within $\varepsilon_{\max}$ was approximately 3.853×10^{14}, due primarily to the persistent skewness of the distribution at high degrees of freedom. Since a spline-based solution that included a theoretical maximum of more than 385 trillion unique spline models would not be computationally efficient, the Wilson–Hilferty transformation was used to reduce the skewness of the distribution, thus allowing for a much more accurate normal approximation at lower degrees of freedom than would otherwise be possible with the direct standardization method [41]. With the addition of this transformation, the degrees of freedom for the standardized chi-squared distribution beyond which the CDF was indistinguishable from the standard normal CDF within $\varepsilon_{\max}$ was observed to be a much more tractable 1,018,679, which was subsequently adopted as $df_{\max}$ for the chi-squared solution. CDF calculations for the Student's t and chi-squared distributions for degrees of freedom exceeding their respective values of $df_{\max}$ could thus be safely obtained by relying on the spline-based solution for the standard normal CDF without exceeding the maximum error tolerance.

Next, fitting a unique spline model for all successive df values between 1 and $df_{\max}$ was unnecessary because, as the degrees of freedom increase, the difference in the shape of the underlying CDF curves between one df value and the next becomes progressively smaller, eventually falling below the maximum error tolerance $\varepsilon_{\max}$. More specifically, rather than fitting a spline model for every possible df value between 1 and $df_{\max}$, a sparse set of models was constructed for the Student's t and standardized chi-squared distribution CDFs by following the sequence of steps outlined below:

1. Fit a spline model for $df = 1$ that is accurate within $\varepsilon_{\max}$. This model becomes the initial reference model S_{ref}.
2. Use a binary search to find the next essential spline model. This will be the model S whose df value is closest to that of S_{ref} for which the maximum absolute error between S_{ref} and S exceeds $\varepsilon_{\max}$. Once identified, this model becomes the new reference model S_{ref}. For any degrees of freedom that fall between the previous reference model and the new reference model, the previous model can be used to calculate values of the corresponding CDF because the maximum absolute error between that model and the hypothetical model for the specified df will always be less than or equal to $\varepsilon_{\max}$.

3. Repeat Step 2 until all essential spline models between $df = 1$ and $df_{\max}$ have been identified, trained, and added to the collection.

Using this approach, it was possible to obtain highly accurate CDF values for any valid degrees of freedom by using a small, sparse set of spline models, thus minimizing memory requirements and maximizing computational efficiency at runtime.

3.3. Evaluative Experiments

A series of empirical benchmarking experiments was conducted in order to assess the computational efficiency and accuracy of the proposed adaptive spline interpolation models. These experiments were designed to compare the performance of the spline interpolation methods described above against the well-established, high-precision algorithms available in the Python SciPy library, with these algorithms serving as the baseline against which both the accuracy and the speed of the spline models were judged. As noted previously, the experiments assessed the performance of spline interpolation models that were trained to approximate the cumulative distribution functions (CDFs) for the standard normal, Student's t, and chi-squared distributions.

For each of the three CDFs, an experiment was conducted in order to compare the performance of the SciPy and spline methods across 30 independent trials, thereby accounting for any minor fluctuations in system performance from trial to trial and ensuring that the distribution of the results would be approximately Gaussian, per the Central Limit Theorem [39]. Each experimental trial involved the computation of 10 million CDF values using both the baseline SciPy algorithm and its corresponding spline-based solution, with these calculations being performed in a serial fashion in order to ensure a fair inter-method comparison. All experiments were conducted in the same Ubuntu Linux-based computational hardware and software environment, utilizing Python (version 3.12.3), SciPy (version 1.15.1) for CDF calculations and spline interpolation, and the NumPy library (version 2.0.2) for numerical operations [42,43]. Importantly, the SciPy libraries that were used for CDF calculations and spline interpolation and the NumPy library that was used for numerical operations rely on compiled C and Fortran code, which ensures exceptional computational speed. To guarantee a comprehensive and unbiased comparison, the input values for each trial were drawn from a uniform random distribution over a range designed to cover the effective domain of each target distribution, including its central body and far tails. For the Student's t and chi-squared CDFs, the degrees of freedom for each of the 10 million computations were also selected at random from a pre-defined integer range, thus representing a realistic workload for high-volume statistical applications. The specific ranges from which the input values and degrees of freedom for each calculation were randomly drawn are shown in Table 1 below. These ranges conform to the rigorous standards that have been established in the literature for testing the accuracy of statistical algorithms [19,20,22,44].

Table 1. Ranges of input values used in the evaluative experiments.

Function	Input Range	Degrees of Freedom Range
Standard Normal CDF	−100 to 100	N/A
Student's t CDF	−10,000 to 10,000	1 to 100,000
Chi-Squared CDF	0 to 1,000,000	1 to 1,000,000

The efficacy of the adaptive spline models was evaluated based on two primary metrics: (1) computational accuracy and (2) computational speed (i.e., wall-clock time). For each experimental trial, the accuracy of the spline models was quantified by determining the minimum, mean, and maximum absolute error between the 10 million CDF values generated by the spline-based solutions and those produced by their corresponding SciPy algorithms. With respect to accuracy, the spline models were considered to be sufficiently precise if their maximum observed error across all experimental trials did not exceed the predefined maximum error threshold, ε_{max}. As noted previously, ε_{max} was set to a very small value of 1×10^{-8} (i.e., 0.00000001) for the current study. Since the adaptive spline models were trained to approximate cumulative distribution functions (which yield p-values), this high level of precision ensured that the resulting spline-based solutions could be usefully applied in almost all practical statistical and scientific scenarios.

For each experimental trial, the computational speed of the spline models and their corresponding SciPy algorithms was measured by recording the total amount of wall-clock time (in seconds) that was required for each method to complete the trial's 10 million CDF calculations. Each experiment thus yielded two samples of 30 wall-clock time measurements, with one sample recording the SciPy algorithm execution times and the other sample recording the execution times for the corresponding spline models. To determine whether the observed differences in wall-clock execution time were statistically significant, the two samples of timing data for each experiment were compared using an independent, two-sample Welch's t-test [45]. This statistical test was specifically chosen because it does not assume equality of variances between the two samples that are being compared. This is a critical consideration, since the computational complexity of spline interpolation (a direct, memory-access-bound operation) is fundamentally different from that of the iterative numerical methods used in standard statistical algorithms, thus making an assumption of equal variance untenable.

4. Results and Discussion

4.1. Model Characteristics

Recalling from the previous section that the spline-based approximations for the Student's t and chi-squared distributions required a collection of spline models (one for each essential *df* value), Table 2 below provides a summary of the characteristics of the spline-based solutions for each CDF after the training process was finished.

Table 2. Characteristics of spline-based solutions.

Function	Number of Spline Models	Knots per Model
Standard Normal CDF	1	90
Student's t CDF	9,857	124 to 268
Chi-Squared CDF	46,418	12 to 193

As shown in the table, the solution for the standard normal CDF required only one spline model with 90 knots in order to approximate the original function within the maximum error tolerance, ε_{max}. By contrast, the solutions for the Student's t and chi-squared distributions each required several thousand unique spline models in order to approximate their original functions within ε_{max} across the entire domain of valid integer degrees of freedom, $[1, +\infty]$. Despite the seeming complexity of these solutions, it is important to remember that each individual calculation required only a quick lookup operation to fetch the appropriate spline model, followed by a simple and computationally efficient polynomial evaluation.

4.2. Experiment Results—Computational Accuracy

Having reviewed the characteristics of the trained spline interpolation models, the results obtained from the experiments with specific respect to the accuracy of those models are presented in Table 3 below, with the data in the table summarizing the observed computational accuracy of the trained spline interpolation models in comparison to their SciPy reference algorithms. As a reminder, the values from the experiments reported below were derived from 30 trials per experiment, with each trial involving the serial calculation of 10 million CDF values.

Table 3. Experimentally observed computational accuracy of spline interpolation models.

		Observed Absolute Error		
Function	**Trials**	**Minimum**	**Mean**	**Maximum**
Standard Normal CDF	30	0.0	9.73×10^{-11}	1.00×10^{-8}
Student's t CDF	30	0.0	1.21×10^{-9}	9.93×10^{-9}
Chi-Squared CDF	30	0.0	3.02×10^{-11}	9.99×10^{-9}

As shown in the table, the adaptive spline interpolation models were observed to be highly accurate in the experiments. Specifically, each of the spline-based solutions exhibited a minimum absolute error of 0.0 and a maximum absolute error that was less than or equal to the maximum error threshold across all of the experimental trials. The mean absolute error (MAE) for the approximation of the standard normal distribution was observed to be 9.73×10^{-11}, while the MAEs for the Student's t and chi-squared distributions were observed to be 1.21×10^{-9} and 3.02×10^{-11}, respectively. All of these values, of course, fall within the maximum error tolerance ($\varepsilon_{\max}$) of 1×10^{-8} that was established prior to the model training process, lending face validity to the adaptive spline model construction process described in the previous section. Collectively, these results establish that the spline-based solutions were able to reproduce the values generated by their corresponding SciPy algorithms with a very high degree of accuracy, thereby confirming that spline interpolation models can be usefully substituted for traditional algorithms in the context of scientific and statistical computing.

4.3. Experiment Results—Computational Speed

While demonstrating the ability of the adaptive spline interpolation models to approximate their corresponding CDFs with a high degree of accuracy is certainly necessary, accuracy alone is insufficient to establish the value of the spline interpolation method for high-speed scientific and statistical computing. Indeed, achieving this more stringent goal requires that the spline interpolation method be demonstrated to be both highly accurate and significantly faster than traditional algorithms. To this end, Table 4 below summarizes the results of the experiments with respect to the comparative computational speed of the SciPy algorithms and their corresponding spline-based solutions. Again, the wall-clock time values reported in the table were derived from 30 trials per experiment, with each trial involving the serial calculation of 10 million CDF values.

Table 4. Experimentally observed computational speed of spline interpolation models.

		Mean Wall-Clock Time (Seconds)	
Function	**Trials**	**SciPy Algorithms**	**Spline Models**
Standard Normal CDF	30	243.117 (sd = 3.551)	2.863 (sd = 0.035) ***
Student's t CDF	30	273.869 (sd = 2.091)	31.364 (sd = 0.231) ***
Chi-Squared CDF	30	270.129 (sd = 2.158)	36.041 (sd = 0.689) ***

*** $p < 0.001$, sd = standard deviation.

As shown in the table, the spline-based solutions were observed to be much faster than their corresponding, widely used SciPy algorithms when computing values for the standard normal, Student's t, and chi-squared cumulative distribution functions, with Welch's *t*-tests revealing these differences in speed to all be highly statistically significant at $p < 0.001$. Specifically, the spline model for the standard normal CDF was observed to be a remarkable 87.4 times faster than its SciPy equivalent. Likewise, the spline-based solution for the Student's t CDF was observed to be approximately 8.7 times faster than its corresponding SciPy algorithm, while the spline-based solution for the chi-squared distribution CDF was observed to be approximately 7.5 times faster than its SciPy equivalent. When considered in conjunction with the results reported in Table 3, these findings suggest that the spline-based approach is both highly accurate and much faster than traditional algorithms when computing values of the cumulative distribution functions that were considered in this study's experiments. In addition to being highly accurate and very fast, the spline-based approach also has the advantage of relying on a single, simple method to achieve these superior results, regardless of the CDF being approximated. By contrast, many traditional algorithms must rely on a combination of methods such as Taylor series, continued fractions, and asymptotic expansions in order to deliver accurate results over a function's entire domain [22]. From the perspective of implementation, this, too, makes the spline-based approach very attractive.

5. Summary, Limitations, and Concluding Remarks

5.1. Summary and Contributions

This paper described and experimentally validated a high-speed computational method based on adaptive spline interpolation and showed how the method can be used to accelerate the calculation of widely used statistical functions. This was accomplished by constructing parsimonious spline models that were designed to approximate a target function within a predefined, high-precision maximum error tolerance of 1×10^{-8}. The method employs an iterative algorithm that adaptively places knots in regions of highest error, after which unnecessary knots are pruned in order to ensure a maximally efficient model for the specified level of accuracy. For distributions involving a degrees of freedom parameter (such as the Student's t and chi-squared distributions), the proposed approach involves pre-computing and storing a collection of individual spline models only for essential degrees of freedom values, which allows for a solution that consists of a small, sparse set of models and enables rapid, dictionary-based retrieval and execution at runtime.

The efficacy of the proposed approach was evaluated through a series of benchmarking experiments that compared the spline-based solutions against established algorithms in the Python SciPy library for the standard normal, Student's t, and chi-squared CDFs. Across 30 trials, each involving 10 million random CDF calculations, the spline-based methods demonstrated high fidelity, consistently achieving a maximum absolute error that did not exceed the 1×10^{-8} tolerance. Furthermore, Welch's *t*-tests confirmed that the spline models were significantly faster than traditional algorithms, with the spline-based solutions being between 7.5 times and 87.4 times faster than their SciPy counterparts ($p < 0.001$ in all cases).

Collectively, the results of the experiments verify that spline interpolation can be both highly accurate and substantially faster than traditional computational techniques. Spline interpolation, of course, has been well-established for several decades; however, the major contributions of this paper are to show how spline interpolation can be adapted to and usefully applied in the context of scientific functions through the use of a sparse set of spline interpolation models that are constructed by identifying the essential values of a function's parameters. The exceptional speed and high accuracy of the proposed method notwithstanding, another notable advantage of the proposed approach is its methodological simplicity. Adaptive spline interpolation relies on a single, unified framework to perform its calculations, unlike conventional scientific algorithms that often require a complex combination of numerical techniques. By reducing the computationally intensive task of calculating CDF values to a simple lookup and polynomial evaluation, the spline-based approach offers a practical pathway to more accessible and rapid analyses of large datasets, regardless of researchers' hardware budgets.

5.2. Limitations

While the findings of this study are promising, it is important to acknowledge several limitations that define the scope of the current work and present opportunities for future research. First, the experiments were conducted using a single, high-precision maximum error tolerance of 1×10^{-8}. Although the adaptive spline interpolation method could be used to train models with different levels of precision, the specific trade-offs between accuracy and wall-clock time that would result from varying this tolerance remain unquantified. Second, the performance of the spline models was benchmarked exclusively against the algorithms within the Python SciPy library. While SciPy represents a well-established and widely used baseline, the computational performance of the proposed method relative to other statistical packages or custom algorithms has not been explored. Third, the performance of the spline-based solutions was compared to SciPy using hundreds of millions of serial calculations. While this allowed for a fair comparison between the two approaches, how well the adaptive spline interpolation method would perform in a parallel computing environment remains unknown. The simplicity of the method, however, suggests that it could be parallelized with relative ease and that it could also be adapted to run on graphics processing units (GPUs), which could potentially yield even more dramatic improvements in computational speed. Finally, the study's application was confined to approximating the cumulative distribution functions for the standard normal, Student's t, and chi-squared distributions. While there is no reason to anticipate that the spline-based approach could not also be usefully applied to other common, continuous scientific and statistical functions, the extensibility and performance of this method for these other functions were not examined in the current paper. Each of these limitations represents a fruitful avenue for future research into the capabilities and applications of adaptive spline interpolation in the context of high-speed scientific and statistical computing.

5.3. Concluding Remarks

The results of this study represent an initial but nevertheless promising step toward a new paradigm in high-speed scientific computing. The demonstrated accuracy and computational efficiency of the adaptive spline interpolation method suggest the potential for developing an extensive library of spline-based models that could perform a wide array of scientific and statistical calculations much more quickly than traditional algorithms. Such a library could significantly accelerate the pace of scientific discovery by reducing the computational bottlenecks that currently constrain the analysis of large and complex datasets. By providing researchers with faster tools, this approach could enable more thorough model validation, the exploration of more intricate analytical models, and a more rapid, iterative approach to scientific inquiry. Ultimately, the widespread adoption of a freely available, high-speed computational library based on these methods could empower a new generation of data-driven decision-making and scientific discovery. It is hoped that the current work will serve as an impetus for future research aimed at making such a library a practical reality.

Funding: This research was funded by the Office of Research and Sponsored Programs at California State University, Fullerton. Grant number: 002768.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The trained adaptive spline interpolation models described in this paper and the Python source code files for testing the speed and accuracy of those models can be accessed at the following URL: https://github.com/daniel-soper/SplinePy. The raw data from the experiments described in this paper can be accessed at the following URL: https://drive.google.com/drive/folders/1RtQ3aoH-mVzWf80hrxgkduR2JNLkxWHf?usp=sharing.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CDF	Cumulative Distribution Function
df	Degrees of Freedom
GPU	Graphics Processing Unit
MAE	Mean Absolute Error

References

1. Heinecke, A. Accelerators in Scientific Computing Is It Worth the Effort? In Proceedings of the 2013 International Conference on High Performance Computing & Simulation (HPCS), Helsinki, Finland, 1–5 July 2013; p. 504.
2. Rahman, A.F.B.; Yusof, Z.B. Optimizing Resource Allocation for Big Data Workloads in Cloud Computing Platforms. *Algorithms Comput. Theory Optim. Tech. Appl. Res. Q.* **2024**, *14*, 15–27.
3. Cheng, S.; Liu, B.; Shi, Y.; Jin, Y.; Li, B. Evolutionary Computation and Big Data: Key Challenges and Future Directions. In Proceedings of the International Conference on Data Mining and Big Data, Bali, Indonesia, 25–30 June 2016; pp. 3–14.
4. Prudius, A.; Karpunin, A.; Vlasov, A. Analysis of Machine Learning Methods to Improve Efficiency of Big Data Processing in Industry 4.0. *J. Phys. Conf. Ser.* **2019**, *1333*, 032065. [CrossRef]
5. Geist, A.; Reed, D.A. A Survey of High-Performance Computing Scaling Challenges. *Int. J. High Perform. Comput. Appl.* **2017**, *31*, 104–113. [CrossRef]
6. Pilz, K.F.; Heim, L.; Brown, N. Increased Compute Efficiency and the Diffusion of Ai Capabilities. In Proceedings of the AAAI Conference on Artificial Intelligence, Philadelphia, PA, USA, 25 February–4 March 2025; pp. 27582–27590.
7. Goldstein, I.; Spatt, C.S.; Ye, M. Big Data in Finance. *Rev. Financ. Stud.* **2021**, *34*, 3213–3225. [CrossRef]
8. Knuteson, B.; Padley, P. Statistical Challenges with Massive Datasets in Particle Physics. *J. Comput. Graph. Stat.* **2003**, *12*, 808–828. [CrossRef]
9. Schnase, J.L.; Lee, T.J.; Mattmann, C.A.; Lynnes, C.S.; Cinquini, L.; Ramirez, P.M.; Hart, A.F.; Williams, D.N.; Waliser, D.; Rinsland, P. Big Data Challenges in Climate Science: Improving the Next-Generation Cyberinfrastructure. *IEEE Geosci. Remote Sens.* **2016**, *4*, 10–22. [CrossRef]
10. Zou, J.; Huss, M.; Abid, A.; Mohammadi, P.; Torkamani, A.; Telenti, A. A Primer on Deep Learning in Genomics. *Nat. Genet.* **2019**, *51*, 12–18. [CrossRef] [PubMed]
11. Fan, J.; Han, F.; Liu, H. Challenges of Big Data Analysis. *Natl. Sci. Rev.* **2014**, *1*, 293–314. [CrossRef]
12. Klemetti, A.; Raatikainen, M.; Myllyaho, L.; Mikkonen, T.; Nurminen, J.K. Systematic Literature Review on Cost-Efficient Deep Learning. *IEEE Access* **2023**, *11*, 90158–90180. [CrossRef]
13. Soper, D.S. Greed Is Good: Rapid Hyperparameter Optimization and Model Selection Using Greedy K-Fold Cross Validation. *Electronics* **2021**, *10*, 1973. [CrossRef]
14. Soper, D.S. Hyperparameter Optimization Using Successive Halving with Greedy Cross Validation. *Algorithms* **2022**, *16*, 17. [CrossRef]
15. Laughlin, G.; Aguirre, A.; Grundfest, J. Information Transmission between Financial Markets in Chicago and New York. *Financ. Rev.* **2014**, *49*, 283–312. [CrossRef]
16. Adewusi, A.O.; Okoli, U.I.; Adaga, E.; Olorunsogo, T.; Asuzu, O.F.; Daraojimba, D.O. BBusiness Intelligence in the Era of Big Data: A Review of Analytical Tools and Competitive Advantage. *Comput. Sci. IT Res. J.* **2024**, *5*, 415–431. [CrossRef]
17. Shah, T.R. Can Big Data Analytics Help Organisations Achieve Sustainable Competitive Advantage? A Developmental Enquiry. *Technol. Soc.* **2022**, *68*, 101801. [CrossRef]
18. Bu, Y.; Howe, B.; Balazinska, M.; Ernst, M.D. The Haloop Approach to Large-Scale Iterative Data Analysis. *VLDB J.* **2012**, *21*, 169–190. [CrossRef]

19. Cody, W.J. Algorithm 715: Specfun–a Portable Fortran Package of Special Function Routines and Test Drivers. *ACM Trans. Math. Softw. (TOMS)* **1993**, *19*, 22–30. [CrossRef]
20. Hill, G.W. Acm Algorithm 395: Student's T-Distribution. *Commun. ACM* **1970**, *13*, 617–619. [CrossRef]
21. De, R.; Bush, W.S.; Moore, J.H. Bioinformatics Challenges in Genome-Wide Association Studies (Gwas). *Clin. Bioinform.* **2014**, *1168*, 63–81. [CrossRef]
22. DiDonato, A.R.; Morris, A.H., Jr. Computation of the Incomplete Gamma Function Ratios and Their Inverse. *ACM Trans. Math. Softw. (TOMS)* **1986**, *12*, 377–393. [CrossRef]
23. Gasca, M.; Sauer, T. Polynomial Interpolation in Several Variables. *Adv. Comput. Math.* **2000**, *12*, 377–410. [CrossRef]
24. Stirling, J. *Methodus Differentialis: Sive Tractatus de Summatione et Interpolatione Serierum Infinitarum*; Typis Gul. Bowyer; Impensis G. Strahan: London, UK, 1730.
25. Runge, C. Über Empirische Funktionen Und Die Interpolation Zwischen Äquidistanten Ordinaten. *Z. Math. Phys.* **1901**, *46*, 20.
26. Schoenberg, I.J. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions. Part A. On the Problem of Smoothing or Graduation. A First Class of Analytic Approximation Formulae. *Q. Appl. Math.* **1946**, *4*, 45–99. [CrossRef]
27. Schoenberg, I.J. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions. Part B. On the Problem of Osculatory Interpolation. A Second Class of Analytic Approximation Formulae. *Q. Appl. Math.* **1946**, *4*, 112–141. [CrossRef]
28. Cox, M.G. The Numerical Evaluation of B-Splines. *IMA J. Appl. Math.* **1972**, *10*, 134–149. [CrossRef]
29. De Boor, C. On Calculating with B-Splines. *J. Approx. Theory* **1972**, *6*, 50–62. [CrossRef]
30. Hastie, T.; Tibshirani, R. Generalized Additive Models. *Stat. Sci.* **1986**, *1*, 297–310. [CrossRef]
31. De Boor, C. *A Practical Guide to Splines*; Springer: New York, NY, USA, 1978; Volume 27.
32. Magalhaes, P.A.A.; Magalhaes, P.A.A., Jr.; Magalhaes, C.A.; Magalhaes, A.L.M.A. New Formulas of Numerical Quadrature Using Spline Interpolation. *Arch. Comput. Methods Eng.* **2021**, *28*, 553–576. [CrossRef]
33. Budzinskiy, S.; Razgulin, A. Defocus Optical Transfer Function: Fast Evaluation and Lightweight Storage Based on Cubic Spline Interpolation. *J. Opt. Soc. Am. A* **2019**, *36*, 436–442. [CrossRef] [PubMed]
34. Romano, D.; Kovacevic-Badstuebner, I.; Antonini, G.; Grossner, U. Accelerated Evaluation of Quasi-Static Interaction Integrals Via Cubic Spline Interpolation in the Framework of the Peec Method. *IEEE Trans. Electromagn. Compat.* **2024**, *66*, 829–836. [CrossRef]
35. Wegman, E.J.; Wright, I.W. Splines in Statistics. *J. Am. Stat. Assoc.* **1983**, *78*, 351–365. [CrossRef]
36. Jupp, D.L. Approximation to Data by Splines with Free Knots. *SIAM J. Numer. Anal.* **1978**, *15*, 328–343. [CrossRef]
37. Dierckx, P. *Curve and Surface Fitting with Splines*; Oxford University Press: Oxford, UK, 1995.
38. Ross, S.M. *A First Course in Probability*; Pearson Harlow: London, UK, 2020.
39. Casella, G.; Berger, R. *Statistical Inference*; Chapman and Hall/CRC: New York, NY, USA, 2024.
40. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2022.
41. Wilson, E.B.; Hilferty, M.M. The Distribution of Chi-Square. *Proc. Natl. Acad. Sci. USA* **1931**, *17*, 684–688. [CrossRef] [PubMed]
42. Van Rossum, G.; Drake, F.L., Jr. *The Python Language Reference*; Python Software Foundation: Wilmington, DE, USA, 2014.
43. Harris, C.R.; Millman, K.J.; Van Der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J. Array Programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
44. Marsaglia, G. Evaluating the Normal Distribution. *J. Stat. Softw.* **2004**, *11*, 1–11. [CrossRef]
45. Welch, B.L. The Generalization of 'Student's'problem When Several Different Population Varlances Are Involved. *Biometrika* **1947**, *34*, 28–35.