Polynomial Fitting of Data Streams with Applications to Codeword Testing*†

Andrew McGregor[a], Atri Rudra[b], and Steve Uurtamo[c]

a **Department of Computer Science,**
University of Massachusetts, Amherst, MA.
`mcgregor@cs.umass.edu`

b,c **Department of Computer Science and Engineering,**
University at Buffalo, SUNY, Buffalo, NY.
`{atri,uurtamo}@buffalo.edu`

Abstract

Given a stream of (x, y) points, we consider the problem of finding univariate polynomials that best fit the data. Over finite fields, this problem encompasses the well-studied problem of decoding Reed-Solomon codes while over the reals it corresponds to the well-studied polynomial regression problem.

We present one-pass algorithms for two natural problems: i) find the polynomial of a given degree k that minimizes the error and ii) find the polynomial of smallest degree that interpolates through the points with at most a given error bound. We consider a range of error models including the average error per point, the maximum error, and the number of points that are not fitted exactly. Many of our results apply to both the reals and finite fields. As a consequence we also solve an open question regarding the tolerant testing of codes in the data stream model.

Keywords and phrases Streaming, Polynomial Interpolation, Polynomial Regression

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

In this paper we consider the following problem: given a stream of n input points $(x_i, y_i) \in \mathbb{F}^2$ (where all the x_i's are distinct[1]) for some field $\mathbb{F}$, fit them with a univariate polynomial with low error. This general problem has been intensively studied under at least two broad specifications. The first case is when $\mathbb{F}$ is the set of reals and we want to minimize the least squares or least absolute deviation errors – in this setting the problem is called (polynomial) regression. The second case is when $\mathbb{F}$ is a finite field and we are trying to minimize the number of disagreements– this corresponds to the problem of decoding Reed-Solomon codes. Both of these problems have great practical value: regression is used to build a succinct model of the input points and is perhaps the most widely used statistical tool. Reed-Solomon codes are widely used to guard against corruption of data in everyday use such in communication protocols and in storage media. We present data stream algorithms for both problems.

The case for data stream algorithms as a tool to handle massive data sets has been well made over the last couple of decades (see, e.g., the survey by Muthukrishnan [14]).

* A. McGregor's research was supported by NSF CAREER Award CCF-0953754, while A. Rudra and S. Uurtamo were supported by NSF CAREER Award CCF-0844796.

† The first and the second authors thank the NSF for inviting them to a meeting where the authors first started discussing the problems considered in this paper.

[1] Some of our results hold even without this condition but for simplicity, we make this assumption. We also note that checking whether this assumption holds or not requires $\Omega(n)$ space.

As polynomial fitting is an extremely basic problem, it is natural to consider its data stream complexity. In addition, these problems also have practical motivations. Polynomial regression dates back to work of Legendre and Gauss and can be used to make sense of large data sets or to use a large experimental data set to accurate estimate model parameters, see, e.g., work in epidemiology [7] and marine geography [1]. Polynomial regression is actually a special case of multivariate linear regression since some variables may be polynomial functions of other variables. Multivariate linear regression has been recently considered in the data stream model but existing work is either only applicable to least squares polynomial regression [3] (and we'll observe that a simpler approach works in this case) or is not as time-efficient [6]. By focusing on a special case of the regression problem we are able to a) consider a wider range of error measures such as maximum absolute error and cardinality of errors, b) develop faster and more space-efficient algorithms, c) consider variants of the problem such as fitting the simplest model subject to an error budget, and d) consider fields other than the reals.

Approximation of decoding codes under the umbrella of property testing has been intensively studied since the the advent of interactive proofs more than two decades ago. While Reed-Solomon codes are less interesting from a sampling perspective, the second and third authors recently introduced the problem of designing data stream algorithms for codeword testing [15]. They detailed applications of data stream algorithms for codeword testing in storage systems and network systems. We discuss this further in Section 1.1.

The problems of polynomial regression and decoding of Reed-Solomon codes have inherently different motivations. In particular, let k be the degree of the polynomial that we are trying to fit through the data. In polynomial regression we want to make k as small as possible as that means that our data has a small representation. On the other hand, for Reed-Solomon codes we want k to be as large as possible as that means we introduce as little redundancy as possible. Further, different kinds of error make sense in the two problems. For the Reed-Solomon case, which are defined over finite fields, the measure of error is the Hamming distance, or the ℓ_0 norm. On the other hand, for polynomial regression, ℓ_p and ℓ_∞ measures also make sense. These differences crop up in the kind of algorithms generally used to tackle these two problem. Many of our solutions, however, are "oblivious" as to whether we are working over finite fields or the reals. Some proofs were omitted for space considerations, however, all omitted proofs are available in the full version of the paper.

1.1 Decoding of Reed-Solomon Codes and Related Problems

We now focus on the polynomial fitting problem when $\mathbb{F}$ is a finite field. In this case, we will primarily consider the error distance of Hamming distance (i.e. the number of positions where the fitted polynomial disagrees with in the input point). However, some of our results also extend to the ℓ_p case for $p > 0$.

First consider the problem of *error detection*, i.e., we want to figure out if a polynomial of degree at most k fits exactly through all the n input points. Even though much weaker than the error correction problem, error detection is widely used in practice, e.g., in Internet traffic where one uses a checksum to detect errors. In fact, the error detection algorithm (compute the checksum of the data and compare it with the stored checksum) is a very efficient one-pass data stream algorithm. This feature of checksums is hugely attractive in practice even though checksums have terrible error-correction properties. One of the motivations of [15] was to see if the Reed-Solomon code, which has excellent error-correction capabilities, could also have data stream algorithms for error detection. It was shown in [15] that this problem does indeed have a poly-log space, single-pass data stream algorithm if we allow the algorithm advance knowledge of the x_i's.

Given the somewhat surprising fact that error detection for Reed-Solomon codes does indeed have efficient data stream algorithms, [15] also looked at the following *tolerant testing* problem: Is it the case that there exists a polynomial of degree at most k that disagrees with at most t points in the input or do all polynomials of degree at most k have to disagree in at least $2t$ points? They showed that the trivial algorithm of trying out all possible error locations can be implemented in $\tilde{O}(t)$ space with one pass. They also used group testing ideas to improve the running time with similar parameters under the additional constraint of $tk \leq O(n)$. It was an open question whether this could be improved (and, in fact, the second author has widely conjectured that one would need $\tilde{\Omega}(t)$ space). In this paper, we show that one can in fact solve this problem in $\tilde{O}(k)$ space *independent* of t.

The main building block for the algorithm above is an algorithm to estimate the F_0 value of a vector where each coordinate is updated by an addition over the field $\mathbb{F}$. This problem of course, is very well studied for the case of the reals and has been implicitly studied for finite fields [10]. Our algorithm is similar to existing algorithms for estimating F_0 (see, e.g., [12]) and works for any field $\mathbb{F}$. The technical ingredient is a subroutine to determine efficiently if a given subset of vector positions has a nonzero value in it (the catch is that for fields in general nonzero elements can add up to zero). Given this subroutine, the algorithm mentioned in the paragraph above is simple: sketch the input y values and then cycle through all the q^k possibilities for the codewords. (The latter can be done in low space if the algorithm has full knowledge of the x_i's.) In general, trying to improve the running time of this algorithm is hopeless as it would solve the maximum likelihood problem for Reed-Solomon codes, i.e., computing the degree k polynomial that disagrees with the minimum number of input points), which is known to be NP-hard [8]. In fact, there are no known approximate maximum likelihood algorithms for any nontrivial codes, and this is a notoriously hard problem [5]. However, under additional constraints on t, we show that one can in a single pass, compute the closest polynomial and estimate t in space $\tilde{O}(k)$.

We also consider the following natural problem related to polynomials: given the coefficients of a degree k polynomial, compute the number of roots of the polynomial. Note that in this case the trivial algorithm of storing the entire input takes $\tilde{O}(k)$ space. In fact, computing the number of non-roots is the same as computing the F_0 value of the stream of the evaluation of the polynomial over all elements of the field, which by our earlier algorithm is easy. However, we show that the complementary problem of computing the number of roots takes $\Omega(k)$ space. In fact, this is true even if we want to solve the simple problem of checking if the polynomial has any roots at all. The reduction is from set-disjointness and makes use of some properties of non-squares in fields.

1.2 Polynomial Regression

We now discuss our results for polynomial regression over the reals (though some of our results work over any field). Given $p \geq 0$, and n points $(x_1, y_1), \ldots, (x_n, y_n)$, the aim is to find a univariate polynomial $f(X)$ of degree at most k that minimizes the ℓ_p error, i.e. the sum $\sum_{i \in [n]} |y_i - f(x_i)|^p$. A fairly easy argument shows that one requires $\Omega(k)$ space to solve this problem. We show that under many scenarios, this lower bound is indeed tight. Contrast this with the general regression problem where the corresponding space bound is $\tilde{\Theta}(k^2)$ [3].

We first consider the case when we are given a bound e on the error we are willing to tolerate and we are interested in computing $f(X)$ of the smallest possible degree k that results in an error of at most e. Note that in this case k is *unknown* but we still want to use space that is $\tilde{O}(k)$. We present two one pass $\tilde{O}(k + e)$-space algorithms for the case when

$e = 0$ and $e > 0$ but $p = 0$. In both cases, our algorithms work under extra conditions on n, k and e. The main idea in both of these algorithms is to build an estimate $\hat{k}$ on k progressively. It is not too hard to figure out when the estimate $\hat{k}$ is smaller than the actual value of k. Our main insight is that under suitable conditions on n, k and e, when we discover that our estimate $\hat{k}$ is insufficient, we can discard all we have seen so far and start from scratch. The conditions guarantee that we never throw away too much information. Also crucial to our algorithms are the known observations that (i) Newton's interpolation formula can be implemented in an "online" fashion and uses $\tilde{O}(k)$ space and (ii) existing decoding algorithms for Reed-Solomon codes can be implemented in linear space.

Next we consider the case when k is specified up-front. For $p = 0$, our algorithm samples $O(k)$ points and then uses the decoding algorithm for Reed-Solomon codes mentioned in the previous paragraph to compute the optimal polynomial. For $p \in [1, 2)$, we use Indyk's p-stable sketching technique to sketch the y and x values. A naive approach would then be to cycle through all possible values for the coefficients (we also provide a simple one pass algorithm to bound the range of values each coefficient can take). While this results in an $\tilde{O}(k)$ space algorithm, the running time is not satisfactory. Using the convexity of the error function, we present an algorithm that effectively does a binary search in a k-dimensional space to compute the best values of the coefficients. This leads to a $O(\log^k n)$-pass $\tilde{O}(k)$ space algorithm, which we then refine to a one-pass, $\tilde{O}(k)$ space and $O(\log^k n)$ time algorithm. Finally, we consider the case of $p = \infty$. We observe that a result due to Chan and Chen [2] can be used to solve the problem *exactly* in constant passes and sub-linear space. We also present a one-pass $\tilde{O}(k)$ space algorithm to approximate the ℓ_∞ error that is in turn based on the fact that one can compute the optimal polynomial for any *even* $p \geq 2$ with one pass and $\tilde{O}(p^2 k)$ space.

2 Finding Smallest Degree Polynomials

We first consider the following problem: Given n input points (x_i, y_i) $(1 \leq i \leq n)$ and an integer $0 \leq e \leq n$, compute the polynomial $f(X)$ of the smallest degree k such that $|\{i | f(x_i) \neq y_i\}| \leq e$.

2.1 Perfect interpolation

We begin with a one-pass $\tilde{O}(k)$ space algorithm to compute the polynomial of minimum degree k that interpolates through all the points, i.e., we solve the problem above with $e = 0$. This will serve as a warmup for the general case (in addition to giving a slightly better result for this special case.) Note that here we do *not* know k in advance and this is what makes the problem nontrivial. Furthermore, we do not make any assumptions about the range or order of the points nor do we know $\{x_i : i \in [n]\}$ in advance.

▶ **Theorem 1.** *Let $(x_1, y_1), \ldots, (x_n, y_n)$ be n input points such that there is an unknown polynomial $f(X)$ such that for every $1 \leq i \leq n$, $f(x_i) = y_i$. Then there exists a one-pass $\tilde{O}\left(\frac{1}{\epsilon} \cdot \deg(f)\right)$ space algorithm to compute $f(X)$, provided $\deg(f) \leq (1/2 - \epsilon)n$. The amortized update time of the algorithm is $O(\deg(f))$.*

We will use the following well-known result crucially in our algorithm:

▶ **Proposition 2.** *Let the points $(x_1, y_1), \cdots, (x_m, y_m)$ be explained by a polynomial $P(X)$ of degree at most m. Then the points $(x_1, y_1), \ldots, (x_m, y_m), (x_{m+1}, y_{m+1})$ are explained by* *the unique polynomial*

$$Q(X) = P(X) + (y_{m+1} - P(x_{m+1})) \cdot \prod_{i=1}^{m} \frac{X - x_i}{x_{m+1} - x_i}.$$

Further, $\deg(Q) \in \{\deg(P), m\}$.

It is easy to verify that the polynomial $Q(X)$ does indeed work – its uniqueness follows from the fact that two distinct polynomials of degree at most m can agree in at most m points. Further, the claim on the degree of $Q(X)$ follows from the fact that $Q(X) = P(X)$ if $y_{m+1} = P(x_{m+1})$. Finally, note the following corollary that we will use later on: $Q(X)$ can be computed from just the knowledge of $P(X)$, y_{m+1} and $x_1, \ldots, x_{m+1}$.

Proposition 2 implies the following $O(\log(\deg(f)))$-pass algorithm: guess the degree of $f(X)$ in a geometric series and then use Proposition 2 to fit the data with a polynomial with the guessed degree. Our algorithm achieves the result in a single pass.

Proof of Theorem 1. For notational simplicity define $k = \deg(f)$. The algorithm maintains an estimate k' of k. The algorithm also maintains a polynomial $P(X)$ of degree at most k' that explains the last few points (the exact number will be specified later). Now consider the case when the algorithm sees a new point (x_i, y_i). Two things can happen: (i) $P(x_i) = y_i$. In this case, we are good as the current polynomial $P(X)$ explains the new point; or (ii) $P(x_i) \neq y_i$. In this case we want to use Proposition 2 to compute the new polynomial $Q(X)$. Note that in this second case, $\deg(Q) = m$. However, to compute $Q(X)$, we also need to remember all the x_i's we have seen so far.

To implement the idea for part (ii), we will need to keep track of all the x_i's we have seen so far. However, we cannot store all the x_i values if case (ii) never happens (as in that case we would have stored $\omega(\deg(f))$ values). The main observation is to keep track of $O(k')$ x_i values and in case those are not sufficient enough to compute the new $Q(X)$, we update k' accordingly and restart the whole process. The bound of $k \leq (1/2 - \epsilon)n$ is to make sure that by the time we attain $k' = k$, we still have $k + 1$ points left to interpolate through.

We now present the details of the algorithm. Let $c = O(1/\epsilon)$ be a constant that we will fix later on.

```
1. Initialize $k' \leftarrow 1$ and let $P(X)$ be the line that passes through $(x_1, y_1)$ and $(x_2, y_2)$.
2. Set $i, j \leftarrow 2$ and $S \leftarrow \{x_1, x_2\}$. The role of $i$ is to count the total number of points seen
   so far while $j$ counts the number of points seen since last restart.
3. Repeat until $i \leq n - k' - 1$:
   a. Set $i \leftarrow i + 1$ and read $(x_i, y_i)$.
   b. If $P(x_i) == y_i$ then add $x_i$ to $S$ unless $|S| == ck'$. Set $j \leftarrow j + 1$.
   c. Else if $j == |S|$ then set $k' \leftarrow j$ and set $P(X)$ as the $Q(X)$ given by Proposition 2.
      (Note that this be computed from the existing $P(X)$ and $S$.) Finally, add $x_i$ to $S$.
   d. Else set $k' \leftarrow j$ and $j \leftarrow 0$. (the "Restart")
      - Read the points $(x_{i+1}, y_{i+1}), \ldots, (x_{i+k'}, y_{i+k'})$.
      - Set $S \leftarrow \{x_i, \ldots, x_{i+k'}\}$.
      - Set $P(X)$ to be the unique degree at most $k'$ polynomial through $(x_i, y_i), \ldots, (x_{i+k'}, y_{i+k'})$.
      - Set $i \leftarrow i + k'$.
4. If $P(X)$ explains the remaining points then output $P(X)$,
5. Else output $k > (1/2 - \epsilon)n$.
```

Note that if the algorithm halts and outputs $P(X)$ *and* $k' = k$, then it indeed outputs the correct $f(X)$. This is because $P(X)$ explains at least $k + 1$ points and there is a unique polynomial of degree at most k that explains any $k+1$ points. Further, note that the algorithm can be implemented in one pass and uses space $O(ck)$ assuming $k' = k$ at the end of the algorithm.

To complete the proof, we will show that assuming $k \le (1/2 - \epsilon)n$, the algorithm indeed outputs $f(X)$. Toward this end, we first note that in the algorithm whenever we update k', there exists a polynomial of degree k' that agrees with the last $k'+1$ points. Now, if at any update we get $k' > k$, then it means that two polynomials – one of degree k' (the polynomial $P(X)$) and another of degree $k < k'$ (the polynomial $f(X)$) agree on $k'+1$ points, which is not possible. Thus, we have that at any stage of the algorithm, $k' \le k$. To prove that at the end, $k' = k$, we claim that the last restart happens at $i \le n-k-1$. Assuming this claim is true, note that the algorithm outputs a polynomial $P(X)$ of degree $k' \le k$ that agrees with the last $k+1$ points. This implies that $f(X) = P(X)$ (and hence $k' = k$).

To complete the proof we need to prove that the last restart happens at $i \le n-k-1$. To this end, we will show that the number of items discarded during restarts is at most $n-k-1$. Indeed, we consider the set of indices $\{i_1, \dots, i_m\} \subseteq [n]$, where during the i_ℓth iteration ($\ell \in [m]$), the value of k' changed. For ease of notation, let the k' value at the i_ℓth iteration for $\ell \in [m]$ be denoted by $k'(\ell)$. Note that for every $1 \le \ell < m$, $k'(i_\ell) \le k'(i_{\ell+1})$. Further, call $j \in [m]$ *bad* if k' changed as a result of a restart. Further, note that the number of discarded points is exactly $\sum_{\ell \text{ bad}} k'(\ell)$. Now, note that when ℓ is bad then since we did not go through Step 3(c), we have the current value of j in Step 3(d) satisfying $j > ck'(\ell-1)$. Thus, this implies that for bad ℓ, $k'(\ell) > ck'(\ell-1)$. Further recall that we had shown earlier that $k'(m) \le k$. Thus, the sum above is bounded by $\sum_{i=0} k/c^i = \frac{c}{c-1} \cdot k \le (1+\epsilon)k - 1$, where the last inequality follows by choosing an appropriate $c \in O(1/\epsilon)$. Thus, we would have proved the claim if $(1+\epsilon)k - 1 \le n-k-1$, which in turn is implied by the assumption that $k \le (1/2-\epsilon)n$. ◀

2.2 Interpolation with outliers

We now present a one-pass $\tilde{O}(k)$ space algorithm to compute the polynomial of minimum degree k that interpolates through all but e points.

▶ **Theorem 3.** *Let $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{F}^2$ be n input points such that there is an unknown polynomial $f(X)$ such that $|\{i | f(x_i) \ne y_i\}| \le e$ for some $0 \le e \le n$. Then there exists a one-pass $\tilde{O}(e + \deg(f))$ space algorithm to compute $f(X)$, provided $(e + \deg(f)) \cdot \log(\deg(f)) \le O(n)$. The amortized update time of the algorithm is $\tilde{O}(k+e)$.*

To prove the theorem above, we will need the error-version of Proposition 2, i.e. a decoding algorithm for Reed-Solomon codes. It is known, for example, that the Berlekamp-Massey algorithm implies the following:

▶ **Theorem 4.** *Let $1 \le K \le N$ be integers. Let $(x_1, y_1), \dots, (x_N, y_N) \in \mathbb{F}^2$ be points. There exists an $\tilde{O}(N)$ space algorithm using $\tilde{O}(N^2)$ field operations that outputs the unique polynomial $P(X)$ of degree at most K such that $|\{i | f(x_i) \ne y_i)\}| < (N-K)/2$.*

The proof of Theorem 3 follows that of Theorem 1, where we use Theorem 4 instead of Proposition 2. The proof is a bit simpler because of the stricter bounds on $\deg(f)$.

3 Polynomial Fitting

In this section we consider finding the degree k polynomial $f(x) = \sum_{i=0}^{k} a_i x^i$ that best fits a stream $(x_1, y_1), \dots, (x_n, y_n)$ of points. We will consider various measures of fit including the total number of points that are not interpolated exactly, the average error on each point, and the maximum error over all points. We introduce the following family of functions $\mathcal{E}_p : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^{k+1} \to \mathbb{R}$

$$\mathcal{E}_p(x,y,a) := \sum_{i=1}^{n} |y_i - f_a(x_i)|^p \qquad \text{where} \qquad f_a(x) = \sum_{i=0}^{k} a_i x^i$$

and write $\mathcal{E}_p^{(k)}(x,y) := \min_{a_k,\ldots,a_0} \mathcal{E}_p(x,y,a)$. We also write $\mathcal{E}_\infty(x,y,a) := \max_{i\in[n]} |y_i - f_a(x_i)|$ and $\mathcal{E}_\infty^{(k)}(x,y) := \min_{a_k,\ldots,a_0} \mathcal{E}_\infty(x,y,a)$.

We start by noting that the case of minimizing $\mathcal{E}_2$ is actually easy in $O(k \log n)$ bits of space! This is because the optimal choices of the a_i coefficients are determined by the following $k+1$ equations:

$$a_0 \sum_{i\in[n]} x_i^j + a_1 \sum_{i\in[n]} x_i^{j+1} + \ldots + a_k \sum_{i\in[n]} x_i^{j+k} = \sum_{i\in[n]} x_i^j y_i \qquad \forall\ 0 \le j \le k\ .$$

The equations correspond to the derivatives of $\mathcal{E}_2(x,y,a)$ with respect to each a_j. It is therefore sufficient to compute the following $O(k)$ values

$$\sum_{i\in[n]} x_i^j \quad \text{for} \quad 0 \le j \le 2k\ , \qquad \text{and} \sum_{i\in[n]} x_i^j y_i \quad \text{for} \quad 0 \le j \le k$$

as the stream is processed. The resulting set of simultaneous equations are then solved in post processing.

A similar idea works for $p \in \{4,6,8,\ldots\}$ but requires a bit more space. The main idea is simple (and has been observed for the more general regression problem): if one thinks of the coefficients $a_0,\ldots,a_k$ as variables then $\mathcal{E}_p(x,y,a)$ is a $(k+1)$-variate polynomial of degree p. Thus, if we can keep track of all the coefficients in this polynomial, then after the pass over the input, one can estimate $\mathcal{E}_p^{(k)}(x,y)$ by cycling through all possibilities for a. This requires keeping track of roughly p^k values, which is not satisfactory. However, it is easy to check that these roughly p^k coefficients only depend on the following $O(p^2 k)$ sums:

$$\sum_{i\in[n]} y_i^j x_i^\ell \qquad \text{for} \qquad 0 \le j \le p, \qquad \text{and} \qquad 0 \le \ell \le (p-j)k.$$

Thus, we only need to keep track of the above $O(p^2 k)$ sums and $\mathcal{E}_p^{(k)}(x,y)$ can be evaluated in post-processing.

However, it is not possible to find the best coefficients in general in sublinear space. An easy way to see this is to consider $p = 1$ and $k = 0$. Given a set of points $\{(x_i,y_i) : i \in [n]\}$ we seek the value a such that $\sum_{i\in[n]} |x_i - a|$ is minimized. But it is well known that the optimal value of a is the median of the y_i values and computing the median exactly in the data stream model requires $\Omega(n)$ bits of space [9].

Another simple lower bound shows that if, rather than reporting the best $k+1$ coefficients, we just want to multiplicatively estimate $\mathcal{E}_p^{(k)}(x,y)$ then this requires $\Omega(k)$ bits of space. This follows from a reduction from indexing where Alice has a set $A \in [2k]$ of cardinality k and Bob has an index $j \in [2k]$. Alice computes the degree k polynomial $f(x) = \prod_{a\in A}(x-a)$ and defines the first $k+1$ elements of a stream $\{(2k+i, f(2k+i)) : i \in [k+1]\}$. Bob then adds the point $(j,0)$. If $j \in A$ then there is a degree k polynomial, namely f, that interpolates through all the $k+2$ points exactly. Alternatively if $j \notin A$ then any interpolating polynomial must have degree at least $k+1$.

3.1 Maximizing the number of points fitted exactly

We first consider fitting a degree k polynomial to the stream of points with the goal of interpolating exactly through as many of the points as possible. The following result applies to both finite fields $\mathbb{F}_q$ and the reals. The application to finite fields relies on the observation that many sketching algorithms for estimating the number of distinct items, F_0, can be carefully modified to estimate $|\{i : f_i \bmod p \neq 0\}|$ for an arbitrary prime p where f_i is the frequency of the value i in the stream.[2]

▶ **Theorem 5.** *Let $n \geq (1+2\gamma)k$ be integers for $\gamma > 0$. Assume that $\mathcal{E}_0(x, y) \leq ((1-\gamma)n-k)/2$ for some $\gamma > 0$. Then it is possible to find the optimal polynomial and estimate $\mathcal{E}_0(x, y)$ up to a factor $(1 + \epsilon)$ with probability $1 - \delta$ (where $\delta \leq \exp(-\Omega(\gamma^2 k))$) in a single pass using $\tilde{O}(\epsilon^{-2} \log(1/\delta) + \gamma^{-2}k)$ space.*

Proof. Let f be a degree k polynomial that interpolates through the maximum number of points. Note that by the bound on $t = \mathcal{E}_0(x, y)$, f will be unique. Call a point (x_i, y_i) *good* if $f(x_i) = y_i$ and *bad* otherwise. The idea in the algorithm is to essentially sample enough points and run the unique decoding algorithm from Theorem 4 on the sampled points. We next present the details.

First assume that $t \leq \gamma k$. In this case, we just run the algorithm from Theorem 4 with $K = k$ and $N = (1 + 2\gamma)k$ on the first N points. Thus, even if in the worst case all the t errors occur in the first N positions, the algorithm from Theorem 4 will output f in space $\tilde{O}(k)$. Note that once we have computed f, we can check that $t \leq \gamma k$ by verifying that f explains the remaining points. Further, we can compute t exactly.

Next we consider the case when $t > \gamma k$. In this case we first sample each of the n input points with probability $4k/(\gamma^2 n)$. By Chernoff, except with probability $\exp(-\Omega(k))$, we would have sampled $N = ck$ points with $4/\gamma^2(1 - \gamma/2) < c < 5/\gamma^2$. Then we run the algorithm from Theorem 4 on the sampled points. Note that if we sample at most $(ck - k)/2$ bad points, the algorithm will indeed return f. Next, we show that this is indeed the case. Note that the expected number of bad points is $\mu = 4kt/(n\gamma^2)$. We show by a case analysis that the probability we get more than $\Delta := (c - 1)k/2$ bad points is exponentially small.

We first consider the sub-case that $\gamma k \leq t < n/(8e)$. Note that in this case $\Delta/\mu > 2e$, which implies that the probability that the number of bad points is more than Δ is at most $2^{-t} = \exp(-\Omega(k))$ [4]. Finally we consider the sub-case that $t \geq n/(8e)$. Note that by the assumption on t, we also have $t \leq (1 - \gamma)n/2$, which implies that in this case $\mu \leq 4k(1 - \gamma)/(2\gamma^2)$. This implies that $\Delta > (1 + \gamma/2)4k(1 - \gamma)/(2\gamma^2) \geq (1 + \gamma/2)\mu$ (as $c > (1 - \gamma/2)4/\gamma^2 \geq 4/\gamma^2(1 - \gamma/2 - \gamma^2/2) + 1$). Thus, the probability that we will have at least Δ bad points by the "usual" Chernoff bound is upper bounded by $\exp(-\Omega(\gamma^2 \cdot \mu)) \leq \exp(-\Omega(\gamma^2 n))$. Thus, in a single pass and $\tilde{O}(k/\gamma^2)$ space we can compute the optimal polynomial $f(X) = \sum_{i=0}^{k} a_i X^i$ with error probability at most $\exp(-\Omega(\gamma^2 k))$.

In parallel with the algorithm above, we compute sketches to estimate the value of t. Compute F_0 sketches (e.g., [13]) of $y = (y_1, \ldots, y_n)$ and $x^j = (x_1^j, x_2^j, \ldots, x_n^j)$ for $0 \leq j \leq k$

[2] In particular, the algorithm detailed in [12] computes $\sum_{i\in S} f_i$ for random subsets S and makes an estimation based on the fraction of random subsets S such that $\sum_{i\in S} f_i \neq 0$ as this indicates that there exists $i \in S$ such that $f_i \neq 0$. However, in the case of $\mathbb{F}_p$ for example, it is possible that $\sum_{i\in S} f_i = 0 \bmod p$ while there exists $i \in S$ such that $f_i \neq 0 \bmod p$. One approach, as taken in Indyk [10] for the case $p = 2$, is to take the probability of this event into account and adjust the estimator appropriately. An alternative approach is to consider $\log(1/\gamma)$ random subsets of each S, $\{S_j : j \in \log(1/\gamma)\}$: if there exists $i \in S$ such that $f_i \neq 0 \bmod p$ then with probability at least $1 - \gamma$, there exists S_j such that $\sum_{i\in S_j} f_i \neq 0 \bmod p$. The results in a factor $\log(1/\gamma)$ increase in the space and time use of the algorithm but it suffices for γ to be $O(\epsilon^{-2})$ so this increase is not significant.

and return an estimate based for t using $\text{sketch}(y - \sum_{j=0}^{k} a_j x^j) = \text{sketch}(y) - \sum_{j=0}^{k} a_j \cdot \text{sketch}(x^j)$.

Repeating the above process $O(\log 1/\delta)$ times and taking the smallest estimate gives a $(1 \pm \epsilon)$ approximation on t with error probability at most δ. Note that we use $\tilde{O}(\epsilon^{-2}\log(1/\delta))$ space in this part of the algorithm. The assumption on δ in the statement of the theorem completes the proof. ◀

3.2 Minimizing the average error

To minimize $\mathcal{E}_p(x, y, a)$ for $p \in (0, 2)$, we first consider the case where we may assume that each a_i comes from some set of t discrete values. Using the p-stable sketching technique [11], construct linear sketches of the $k + 1$ vectors $x^j = (x_1^j, x_2^j, \ldots, x_n^j)$ for $0 \le j \le k$ and $y = (y_1, \ldots, y_n)$. Call these sketches $\text{sketch}(x^j)$ Then for a given setting of $a_0, \ldots, a_k$, we can estimate $\mathcal{E}_p(x, y, a)$ up to factor $1 + \epsilon$ because the sketches are linear:

$$\text{sketch}(y - \sum_{j=0}^{k} a_j x^j) = \text{sketch}(y) - \sum_{j=0}^{k} a_j \cdot \text{sketch}(x^j) \ .$$

If the sketches are of size $\tilde{O}(\epsilon^{-2}\log\delta^{-1})$ then this procedure fails with probability at most δ. Since there are at most t^k settings for a, this procedure works for testing all settings of a with probability at least $1 - t^k\delta$. Rescaling δ gives a $\tilde{O}(\epsilon^{-2}k\log t\log\delta^{-1})$ space algorithm. A similar idea was used in Feldman et al. [6] for multivariate linear regression. The main drawback with this approach is the $O(t^k)$ time required for post-processing.

To ameliorate the situation slightly, we first argue that if we restrict ourselves to finding coefficients up to polynomial precision, we may assume that t is polynomial. In particular we know how to compute a value B in one pass over the input such that all the coefficients of the polynomial $f(X) = \sum_{i=0}^{k} a_i X^i$ minimizing $\mathcal{E}_p(x, y, a)$ satisfy $|a_i| \le B$. Note that in this case $t = O(B/\gamma)$, where $\gamma = 1/\operatorname{poly}(n)$ is the (additive) precision value.

▶ **Lemma 6.** *Let $n > k \ge 0$ be integers, $p \in (0, \infty)$ be a real and (x, y) be the input points. Assume that the polynomial $f(X) = \sum_{i=0}^{k} a_i X^i$ satisfies $\mathcal{E}_p^{(k)}(x, y) = \mathcal{E}_p(x, y, a)$. Then, for every $0 \le i \le k$, $|a_i| \le 6n^{1/p} y_{\max}/\min(1, x_{\min}^k)$, where $y_{\max} = \max_i |y_i|$ and $x_{\min} = \min_i |x_i|$.*

By applying a random shift to the x values we may ensure that the numerator is $\Omega(1)$ and we may subsequently assume that $B = \operatorname{poly}(n)$. For constant k, this ensures that the post-processing step is polynomial in n. In the remainder of this section we show that this dependence on n can be made poly-logarithmic when k is constant and $p \ge 1$.

3.2.1 Poly-logarithmic post processing for constant k:

We start by defining the family of functions $h^j : \mathbb{R}^j \to \mathbb{R}$ for $j = 1, \ldots, k + 1$:

$$h^j(a_k, \ldots, a_{k-j+1}) = \min_{a_0, \ldots, a_{k-j}} \sum_{i=1}^{n} \left| y_i - \sum_{m=0}^{k} a_m x_i^m \right|^p \quad \text{and} \quad h^0 = \mathcal{E}_p(x, y) \ .$$

In other words, h^j is the smallest interpolation error that can be achieved when the j highest coefficients are fixed. We first note that h^j is convex.

▶ **Lemma 7.** *For any $p \ge 1$, $j \in \{0, 1, \ldots, k\}$ and $a_k, \ldots a_{k-j+2} \in \mathbb{R}$, the function $h(x) = h^j(a_k, \ldots, a_{k-j+2}, x)$ is convex.*

```
$Evaluate[j, a_k, \ldots, a_{k-j+1}]$
1. If $j == k+1$, return a $(1+\epsilon)$ approximation based on the sketches.
2. Initialize $a \leftarrow -B$ and $b \leftarrow B$ where $B$ upper bounds the magnitude of all coefficients.
3. Repeat
   a. For $x \leftarrow a + \frac{b-a}{4}$, $y \leftarrow a + \frac{b-a}{2}$, and $z \leftarrow a + \frac{3(b-a)}{4}$
         $e_x \leftarrow Evaluate[j+1, a_k, \ldots, a_{k-j+2}, x]$
         $e_y \leftarrow Evaluate[j+1, a_k, \ldots, a_{k-j+2}, y]$
         $e_z \leftarrow Evaluate[j+1, a_k, \ldots, a_{k-j+2}, z]$
   b. If $e_y < e_z/(1+\epsilon)$ then $b \leftarrow z$ and repeat
   c. If $e_y < e_x/(1+\epsilon)$ then $a \leftarrow x$ and repeat
4. Until $e_y \geq \min(e_x, e_z)/(1+\epsilon)$ or $|b-a| < \gamma$
5. Return $e_y$
```

Figure 1 The Evaluate Algorithm

To find the minimum value of a convex function $h(\cdot)$ in the range $[a, b]$, a natural approach would evaluate h at a few intermediate points, e.g., $a < x < y < z < b$, and recurse on the appropriate subinterval of $[a, b]$ based on the intermediate valuations. If $h(y) \leq h(z)$, we deduce that the minimum lies in the range $[a, z]$ and if $h(x) \geq h(y)$ we deduce that the minimum lies in the range $[x, b]$. Note that one of the above cases must apply since h is convex. If x, y, z are equally spaced in the interval, after $O(\log n)$ iterations we can determine the value that minimizes h.

As a warm-up to the main algorithm of this section, we next present a $O(\log^k n)$ pass algorithm. The algorithm is based on the recursion:

$$h^j(a_k, \ldots, a_{k-j+2}, a_{k-j+1}) = \min_a \left(h^{j+1}(a_k, \ldots, a_{k-j+2}, a_{k-j+1}, a)\right) .$$

We can evaluate h^k for a given $a_k, \ldots, a_1$ in $O(\log n)$ as described above. By appealing to the above recurrence, we can then determine h^{k-1} in $O(\log^2 n)$ passes: we minimize h^{k-1} for a given $a_k, \ldots, a_2$ by performing the quaternary search to find a such that $h^{k-1}(a_k, \ldots, a_2) = h^k(a_k, \ldots, a_2, a)$. Since each evaluation of h^k requires $O(\log n)$ passes, it takes $O(\log^2 n)$ passes to evaluate h^{k-1} for a given $a_k, \ldots, a_2$. Continuing in this manner gives a $O(\log^k n)$ pass algorithm for evaluating h^0. This leads to the following theorem.

▶ **Theorem 8.** *Assume each coefficient may take only t different known values. Then there exists a $O(\log^k t)$ pass algorithm that computes $\mathcal{E}_p(x, y)$ exactly in $\tilde{O}(k)$ space and $O(1)$ per-item processing and $O(1)$ processing at the end of each pass.*

We next transform the multiple pass algorithm into a single pass algorithm where each of the evaluations performed in the quaternary search is computed using a single sketch of the data. In Figure 1, we present the algorithm *Evaluate* for approximating h^j. $Evaluate[j, a_k, \ldots, a_{k-j+1}]$ approximates $h^j(a_k, \ldots, a_{k-j+1})$ by minimizing a sequence of convex functions. Note that *Evaluate* is solely concerned with post-processing: while the points are being streamed it suffices to construct the appropriate sketches. Before we analyze the running time and accuracy of *Evaluate*, we need the following result.

In our algorithm it won't be possible to evaluate h exactly. However, the following lemma demonstrates that when the approximate evaluations become so close that it becomes impossible to evaluate pairwise comparisons, we have identified a sufficiently accurate approximation of the minimum.

▶ **Lemma 9.** *Let $h : [a,b] \to \mathbb{R}$ be a convex function and let $\tilde{h} : [a,b] \to \mathbb{R}$ satisfy $(1-\gamma) \le h(x)/\tilde{h}(x) \le (1+\gamma)$ for all $x \in \mathbb{R}$. Suppose for some a, b, $(1+\epsilon)\tilde{h}(y) \ge \max(\tilde{h}(z), \tilde{h}(x))$ where $x = a + (b-a)/4$, $y = a + (b-a)/2$, and $z = a + 3(b-a)/4$. Then*

$$(1+\epsilon)\tilde{h}(y) \ge \min_{x \in [a,b]} h(x) \ge \tilde{h}(y)/(1+8\epsilon) \ .$$

Proof. Without loss of generality assume that $h(x) \le h(z)$. Note that $h(y) \le h(z)$ because of convexity. We have to analyze the following two cases.

1. $h(x) \le h(y)$: In this case the minimum value is at least

$$h(y) - 2(h(z) - h(y)) = 3h(y) - 2h(z) \ge \tilde{h}(y)[3/(1+\epsilon) - 2(1+\epsilon)^2] \ .$$

2. $h(y) \le h(x)$: In this case the minimum value is at least

$$h(y) - (h(z) - h(y)) = 2h(y) - h(z) \ge \tilde{h}(y)[2/(1+\epsilon) - (1+\epsilon)^2] \ .$$

In either case, the minimum value is at least $\tilde{h}(y)/(1+8\epsilon)$ assuming $\epsilon < 1/15$. ◀

▶ **Theorem 10.** *The running time of Evaluate$[0]$ is $O(\log^{k+1} n)$ and returns a value that satisfies $1/(1+O_k(\epsilon))^k \le \text{Evaluate}[0]/\mathcal{E}_p(x,y) \le (1+O_k(\epsilon))^k$.*

Using an appropriately rescaled ϵ when sketching the original points leads to a $(1+\epsilon)$ approximation using $O(\epsilon^{-2} \operatorname{polylog}(n))$ space and $O(\operatorname{polylog} n)$ update and post-processing time for constant k. We note that dependence on k is such that this approach is only practical for small values of k.

Proof of Theorem 10. For the running time, note that in each iteration the innermost loop is performed $O(\log n)$ times since $B = O(\operatorname{poly} n)$ and $\gamma = 1/\operatorname{poly}(n)$. The result follows because the depth of the recursion is at most $k+1$. The claim on the accuracy follows by induction on the depth and Lemma 9. ◀

3.3 Minimizing the maximum error

In this section, we consider the problem of finding coefficients a such that the maximum absolute error, $\mathcal{E}_\infty(x,y,a)$, is minimized. We will present two results. The first follows from a straight-forward observation and results in a constant pass algorithm that finds the error and the coefficients exactly. The second algorithm only uses a single pass but returns coefficients that minimize the maximum error up to a constant factor.

The first observation is that the problem can be expressed as a linear program in $O(k)$ variables,

$$\min \epsilon \quad \text{subject to} \quad -\epsilon \le \sum_{j=0}^{k} a_j x_i^j - y_i \le \epsilon \quad \forall i \in [n] \ .$$

Such a problem can be solved in constant passes in $O(n^\delta)$ space for any constant δ using the sub-linear time (for constant k) algorithm of Chan and Chen [2].

▶ **Theorem 11.** *It is possible to minimize $\mathcal{E}_\infty(x,y,a)$ in constant passes and $O(n^\delta)$ space for any constant δ.*

Our single pass algorithm is based on the following relationship between $\mathcal{E}_\infty$ and $\mathcal{E}_p$.

▶ **Proposition 12.** *For* $p \geq \frac{\log n}{\log(1+\epsilon)}$, $\mathcal{E}_\infty^{(k)}(x,y) \leq \sqrt[p]{\mathcal{E}_p^{(k)}(x,y)} \leq (1+\epsilon)\mathcal{E}_\infty^{(k)}(x,y)$.

Proof. The result follows because for any non-negative vector $z \in \mathbb{R}^n$ with $r = \max_i z_i$, $r \leq (\sum_{i\in[n]} z_i^p)^{1/p} \leq (nr^p)^{1/p} \leq (1+\epsilon)r$. ◀

In Section 3, we noted that it is possible to evaluate $\mathcal{E}_p^{(k)}(x,y)$ (and determine the corresponding polynomial) in $O(p^2k)$ space if p was even. Therefore, by choosing $p = 2\lceil(\log n)/(2\log(1+\epsilon))\rceil$ and appealing to Proposition 12, get the following theorem.

▶ **Theorem 13.** $\mathcal{E}_\infty^{(k)}(x,y)$ *can be* $(1+\epsilon)$*-approximated in a single pass with* $O(\epsilon^{-2}k\,\text{polylog}(n))$ *space.*

References

1 P. A. Barker, F. A. Street-Perrott, M. J. Leng, P. B. Greenwood, D. L. Swain, R. A. Perrott, R. J. Telford, and K. J. Ficken. A 14,000-Year Oxygen Isotope Record from Diatom Silica in Two Alpine Lakes on Mt. Kenya. *Science*, 292(5525):2307–2310, 2001.

2 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.

3 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *STOC*, pages 205–214, 2009.

4 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 2009.

5 Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Transactions on Information Theory*, 49(1):22–37, 2003.

6 D. Feldman, M. Monemizadeh, C. Sohler, and D.P. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 2, 2010.

7 Sander Greenland. Dose-response and trend analysis in epidemiology: Alternatives to categorical analysis. *Epidemiology*, 6(4):356–365, 1995.

8 Venkatesan Guruswami and Alexander Vardy. Maximum-likelihood decoding of reed-solomon codes is np-hard. *IEEE Transactions on Information Theory*, 51(7):2249–2256, 2005.

9 Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.

10 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC*, pages 373–380, 2004.

11 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

12 Piotr Indyk. 6.895 Sketching, Streaming and Sub-linear Space Algorithms. *Lecture Notes*, 1, 2007.

13 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52, 2010.

14 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

15 Atri Rudra and Steve Uurtamo. Data stream algorithms for codeword testing. In *ICALP*, 2010.