Multidimensional Evaluation

This section evaluates the runtime and accuracy of our approach relative to the parameter d (the number of input dimensions). The behavior in relation to parameter d is *not* a fundamental novelty of our approach and therefore not the main contribution of our paper. All other mentioned approaches share the runtime complexity of $\mathcal{O}(d^2)$ with our approach.

At the same time, there are multiple key insights that are noteworthy to share. As representative real data for multidimensional datasets is hard to find and obtain, this analysis focuses on synthetic data. The general setup resembles the experiments done in Section 6.1 of the main paper.

Discussion of the Sherman-Morrison Rank-1 Update

For the evaluation with $d = 2$, all relevant matrices are at most symmetric matrices of dimension 3×3, the matrix A, which must be inverted—or decomposed and solved for a specific system of equations—is a symmetric matrix of dimension 2×2. The specific approach to do these operations has a negligible impact on the overall runtime of the algorithm. This drastically changes when scaling up d.

Runtime Complexity

To achieve a runtime complexity of $\mathcal{O}(d^2)$, the exact dynamic program (DP) and the last optimization step of our approach rely on precomputing the inverse of matrix A and updating it when adding or subtracting one sample. The naïve way of inverting a square matrix would result in a runtime of $\mathcal{O}(d^3)$. While there are approaches to reduce this runtime complexity for large matrices, no known method reduces it to $\mathcal{O}(d^2)$. The update of an inverted matrix with one additional sample is also called a *rank-1 update* (R1U) and can be done more efficiently than a matrix inversion, using the Sherman-Morrison formula. However, this requires special attention in the detailed implementation.

Given the matrix A with a precomputed matrix A^{-1}, and the vectors u and v, the updated inverse value of $A + uv^T$ can be computed as follows[1]:

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

Directly using the above formula typically results in a trivial evaluation from left to right in the individual subterms. In case of the term $A^{-1}uv^TA^{-1}$ in the dividend, this is detrimental, since it results in a multiplication of two matrices of size $d \times d$, which cannot be computed in quadratic time complexity. This can be solved using the following equation:

$$(A + uv^T)^{-1} = A^{-1} - \frac{(A^{-1}u) \cdot (v^TA^{-1})}{1 + v^TA^{-1}u}$$

Since matrix multiplication (including matrix-vector multiplication) is an associative operation, the above formula is mathematically identical to the first formula, but it consists of two matrix-vector multiplications and one final vector-vector multiplication. All these operations are trivially computable in quadratic time complexity.

Although the optimized rank-1 update is widely recognized, this particular implementation detail is easy to miss. For example, Acharya et al.'s DP algorithm overlooks this aspect—likely an unintentional oversight that could affect both accuracy and performance of the baseline algorithm, though its impact is minimal for $d = 2$ (the primary case evaluated in their work). Despite being theoretically quadratic in time complexity, their implementation does not achieve such efficiency.

Accuracy

Using the Sherman-Morrison formula may raise concerns regarding numerical stability. Continuously updating A^{-1} instead of recalculating it from all available data can lead to a gradual buildup of rounding errors over time. Although this issue appears less severe for *well-conditioned* matrices, even these cases have shown signs of reduced stability.

In our evaluation, we used 64-bit fixed-size floating-point numbers—commonly known as *doubles*—for the computations. This approach aligns with standard practices in similar calculations, though it can introduce the rounding errors mentioned earlier. Although a detailed analysis of the Sherman-Morrison formula's outcomes is beyond the scope of this paper, we provide several anecdotal insights based on our specific use case.

Low Impact on Well-Conditioned Data: In the final stage of our algorithm, we ensure that nearly all samples come from a single segment, yielding a regression function $\hat{f}$ that closely approximates the true underlying function—with only a few outliers misassigned during the initial segment placement. Under these conditions, we assume that our matrices are well-conditioned. Employing the Sherman-Morrison rank-1 update never lead to any noticeable drop in accuracy—even when dealing with very high dimensions d or large numbers of samples n. This is clearly demonstrated by comparing the performance of our algorithm both with and without the rank-1 update.

Well-Conditioned Data Reduces the Impact: The DP often calculates the regression function across multiple segments (e.g., over the whole sample set). Using the optimized

[1]The vectors u and v can be different in the generic version of the formula, but in this scenario they are always the same vector, describing all independent values of a single sample. The matrix A of an individual sample i can be calculated by $A_i = uv^T = x_ix_i^T$.

rank-1 update, resulted in a significantly higher error for DP but not for our algorithm. For $n = 8192$, the error was sometimes large enough to let the matrix seem like a singular matrix, which cannot be inverted at all. We assume this is due to the added samples significantly changing the current inverse. We observed no such issues with our algorithm—even at much larger values for n and d—indicating minimal accuracy loss on well-conditioned matrices.

Worse Accuracy with Trivial Implementation: As mentioned above, Acharya et al.'s original implementation of the rank-1 update suffers more than necessary in terms of speed for large values of d. At the same time, this implementation also seems to yield much worse accuracy. Although these results are not shown here, they can be reproduced using the code provided in the supplemental material.

To ensure a fair comparison between the algorithms, we employ two modified variants of the dynamic program based on Acharya et al.'s implementation. The first variant, *DP*, avoids using the rank-1 update by solving all linear equations explicitly—this guarantees high accuracy at the expense of slower execution times. In contrast, the fast variant, *DP (R1U)*, incorporates fast rank-1 update described above to achieve quadratic time complexity while accepting a slight reduction in accuracy. We also present results for our algorithm with and without the rank-1 update, demonstrating that its performance remains robust under these different configurations.

Fixed Number of Samples

Our experiment setup resembles the synthetic evaluation in the main paper. We measure both accuracy and runtime across 100 randomly generated piecewise functions. Accuracy is quantified using the mean squared error (MSE) between the estimated regression function $\hat{f}$ and the true underlying function f at the positions of the input samples. The samples are uniformly distributed and the breakpoints (BP) are selected at random.

Unlike the original evaluation, we do not fit polynomials; instead, we use input data from multiple truly independent dimensions. For a fair comparison, it is still necessary that $n \gg d \cdot k$, because only one reasonable solution exists at $n = d \cdot k$. Since computational time is a limiting factor in our evaluation—especially when using DP—we reduced the number of breakpoints to $k = 4$, set n to a fixed value, and varied parameter d.

Results for $n = 4096$ are presented in Figure 1. The figure demonstrates that, on occasion, our algorithm produces a suboptimal solution—an effect that is accentuated when the mean value is plotted on a log-log scale. In contrast, plotting the median aligns perfectly with the baseline curve for DP. Additionally, the rank-1 update significantly accelerates both DP and our algorithm for large values of d; however, unlike our approach, DP suffers from reduced accuracy when using the rank-1 update. The results also confirm that our algorithm consistently achieves higher accuracy than

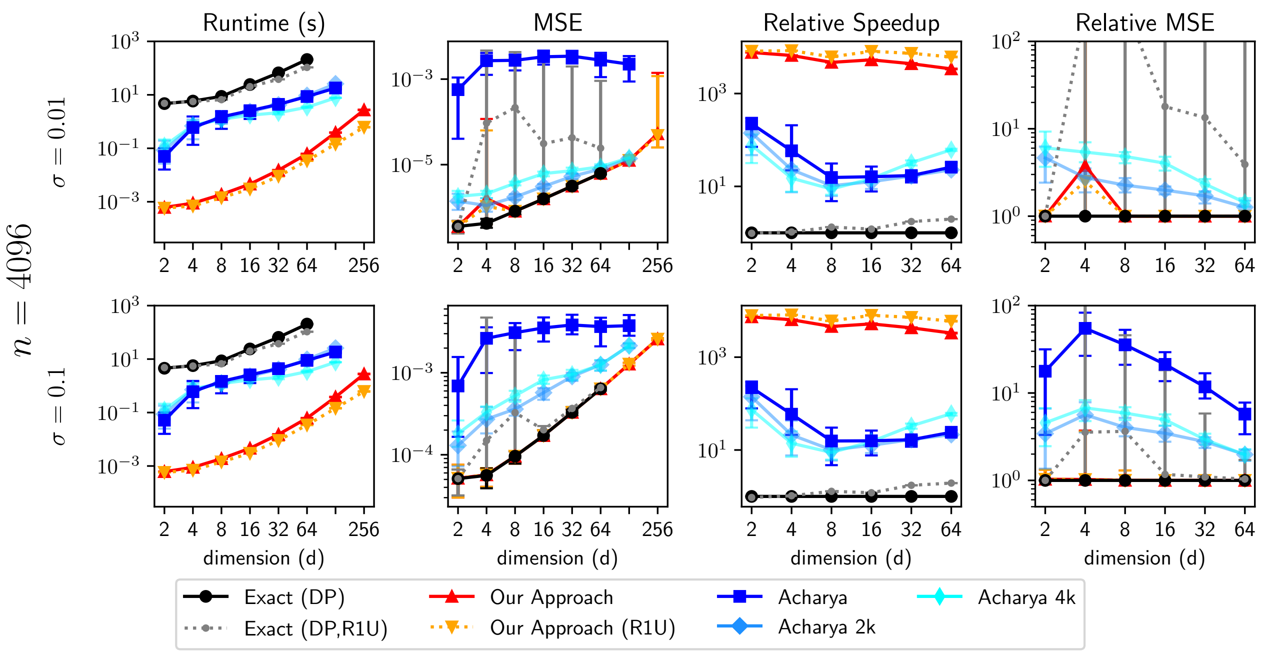

Figure 1. Varying number of arbitrary input dimensions with a constant number of samples. The dotted lines mark versions of the algorithms using the Sherman-Morrison formula for a quick rank-1 update (R1U). An outlier at $\sigma = 0.01, d = 4$ caused a high mean value, which is accentuated by the log-scale.

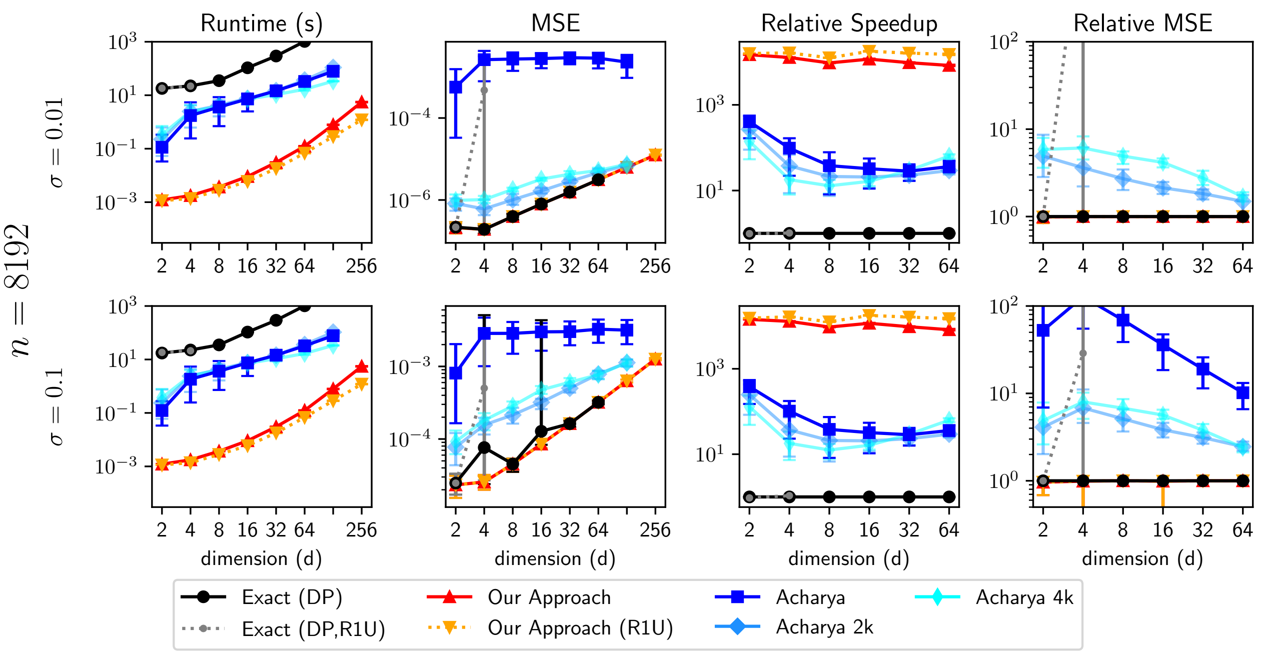

Figure 2. Similar experiment to the one shown in Figure 1, but using 8192 samples. The DP using the rank-1 update formula failed at $d = 2$ due to rounding errors.

the solution proposed by Acharya et al.—especially when exactly k segments are placed—and remains faster overall, although the gap in both performance metrics narrows as d increases.

The primary reason behind Acharya et al.'s improved performance is the narrowing solution space as d increases while n remains fixed. A valid solution requires each segment to have at least d samples. For example, with $d = 256$, $n = 4096$, and $k = 4$, there are only four times the minimum number of required samples. However, when considering Archarya et al.'s variant that places four times the segments (effectively resulting in $k = 16$), the method is forced into a scenario where only one viable solution exists—namely, an even distribution of breakpoints across the samples. This not only explains why Acharya et al. are closing the performance gap in terms of both speed and accuracy, but it also accounts for why at high d values the $4k$ version outperforms the $2k$ version due to its more rapid collapse of the solution space.

To further validate our analysis, we conducted the same experiment using $n = 8192$. The results are shown in Figure 2. In this trial, the rank-1 update version of DP encountered early failure at $d = 4$ because the accumulated rounding errors resulted in a matrix that behaved like a singular one, i.e., a matrix that has a determinant of 0, which cannot be inverted. Although the naïve DP variant did not fail completely, it too struggled with accuracy issues at $d = 4$ and $d = 16$ with $\sigma = 0.1$. In contrast, our algorithm remained unaffected by these problems.

Observations indicate that the performance gap compared to Acharya et al. remains significantly larger and narrows later. This trend is particularly evident in the relative performance graphs. The impact of a collapsing solution space becomes especially prominent when comparing the runtimes of the 2k and 4k variants of Acharya et al. For $n = 4096$, the 4k variant starts outperforming the 2k variant just beyond $d = 8$; whereas for $n = 8192$, this advantage emerges between $d = 16$ and $d = 32$.

Variable Number of Samples

Due to the algorithm's runtimes, increasing n arbitrarily is not tractable. To keep the solution space size fixed, we adjust by scaling n relative to d instead of maintaining a constant number of input samples. Specifically, we set $n = 64 \cdot d$, which ensures that the solution space remains constant while keeping the DP runtime manageable. The results are shown in Figure 3.

The effects of employing a rank-1 update remain evident in the runtimes of both DP and our approach—with its impact limited solely to DP's accuracy. Moreover, our method again is affected by an isolated outlier in two distinct scenarios, much like when $n = 4096$. In all other cases, however, our accuracy aligns closely with that of the optimal baseline.

Runtime

For small values of d, the gap relative to Acharya et al.'s results is smaller; however, this may be due to measurement

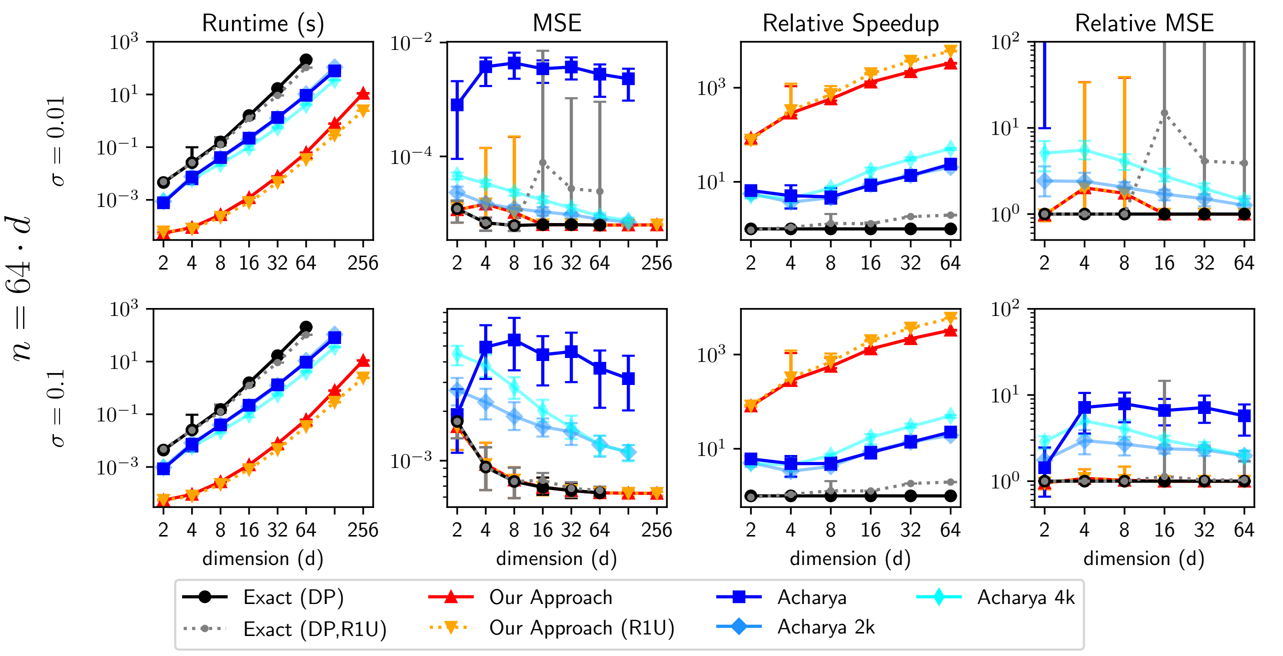

Figure 3. Similar experiment to the ones shown in Figure 1 and Figure 2, but this time the number of samples is not fixed. It is always 64 times the number of dimensions.

overhead since our timings were under one-tenth of a millisecond. For larger values, there are two key observations.

From $d = 8$ onward, the time for solving the linear equations begins to increasingly influence overall runtime. The version without the rank-1 update takes longer and approaches the runtime of Acharya et al., even though the runtime to break even is very large, probably somewhere between 10^4 and 10^5 seconds.

Incorporating the rank-1 update into our algorithm yields a constant speedup relative to Acharya et al.'s method, which is demonstrated by the parallel lines in the log-log runtime graphs. This is expected given that both approaches share identical asymptotic complexity with respect to parameters d and n. In this specific configuration, we observe an approximate $200\times$ speedup, depending on the exact version.

Accuracy

Measuring accuracy proves challenging and appears to be influenced by the noise variance present in the input data. The 2k and 4k variants of Acharya et al.'s method seem to converge toward the baseline as more samples and dimensions are added—this observation holds at least for $\sigma = 0.01$. This behavior may be linked to what is commonly known as the *curse of dimensionality*, where the space defined by increasing sample sizes expands much faster than linearly, resulting in a sparse representation of that space. Unlike experiments conducted with a fixed d, the baseline error does not decrease continuously with additional samples but instead tends to stabilize at a near-constant value. Consequently, even a slight misalignment in the breakpoint has a diminishing impact as the number of samples used for calculating the MSE continues to grow.

The relative error plot does not clearly show that Acharya et al.'s version that places exactly k segments begins to narrow the performance gap when both d and n increase. This uncertainty is especially pronounced for $\sigma = 0.01$.

Conclusion

In evaluations using a fixed-size solution space, we consistently observe a constant speedup factor in runtime. As d and n increase, accuracy tends to level off—likely due to limitations in the experimental design or challenges with high-dimensional data, especially since improvements in the optimal baseline accuracy have plateaued.

Except for some rare outliers, our method is on par with the optimal solution and is significantly more accurate than Acharya et al.—especially compared to their version using exactly k segments. It is also faster than all other algorithms, though its speed advantage relative to Acharya et al. remains consistent.

Our main paper showed that using more samples increases the gap in speed and accuracy between our algorithm and others. We expect a similar trend if we increase the ratio of n to d beyond 64. A much lower value is not suitable in practice, since it approaches the minimum amount of needed samples and probably results in overfitting.