Rational Approximation of Scientific Data [⋆]

Youssef S. G. Nashed[1] [0000−0001−6146−3939], Tom Peterka[1], Vijay Mahadevan[1], and Iulian Grindeanu[1]

Mathematics and Computer Science Division,
Argonne National Laboratory,
Lemont, IL 60439, USA
{ynashed, tpeterka, mahadevan, iulian}@anl.gov

Abstract. Scientific datasets are becoming increasingly challenging to transfer, analyze, and store. There is a need for methods to transform these datasets into compact representations that facilitate their downstream management and analysis, and ideally model the underlying scientific phenomena with defined numerical fidelity. To address this need, we propose nonuniform rational B-splines (NURBS) for modeling discrete scientific datasets; not only to compress input data points, but also to enable further analysis directly on the continuous fitted model, without the need for decompression. First, we evaluate three different methods for NURBS fitting, and compare their performance relative to unweighted least squares approximation (B-splines). We then extend current state-of-the-art B-spline adaptive approximation to NURBS; that is, adaptively determining optimal rational basis functions and weighted control point locations that approximate given input data points to prespecified accuracy. Additionally, we present a novel local adaptive algorithm to iteratively approximate large data input domains. This method takes advantage of NURBS local support to refine regions of the approximated model, acting locally on both input and model subdomains, without affecting other regions of the global approximation. We evaluate our methods in terms of approximated model compactness, achieved accuracy, and computational cost on both synthetic smooth functions and real-world scientific data.

Keywords: Piecewise Approximation · Adaptive Methods · Domain Partitioning · Parallel Algorithms.

1 Introduction

Advancing science through high-performance computing (HPC) depends on managing, analyzing, and visualizing data generated from large-scale simulations or experiments. Much attention has been paid to how best to scale compute capabilities in terms of extreme concurrency and high numbers of operations per second. That, in addition to the prevalence of IoT and scientific observational

[⋆] This work is supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, program manager Laura Biven.

devices, led to unprecedented data volumes and rates. However, there is currently a gap between our increased ability to generate raw data and our ability to store, analyze, and produce scientific results based on these data. This paper seeks to bridge this gap by building upon a fundamentally different kind of data model, termed ***Multivariate Functional Approximation*** (MFA) [19], that conserves resources while improving data understanding and sharing. The new model, which can accommodate many types of scientific datasets on HPC architectures, provides compression and facilitates analytical reasoning not possible before. Moreover, the accuracy of the model is known and guaranteed to user prescription.

The MFA model relies on fitting piecewise smooth functionals to multi-dimensional discrete input data. NURBS basis functions are chosen in MFA because they allow the model to be directly usable in downstream analytics and visualization without the need to evaluate (***decode***) the entire model. This is due to well established NURBS features: NURBS models are continuous across all the input domain, differentiable up to the degree of the used basis functions, and preserve geometric and statistical properties of the input discrete data points. Model continuity provides implicit and inexpensive point evaluation anywhere in the input domain, meaning the model can be sampled at a different mesh resolution than the original discrete data. Differentiability is of particular interest to applications relying on gradient fields, useful for feature detection and tracking, and high-order derivatives, to guide mathematical optimization algorithms for example. Those features are the main reason for NURBS being the *de facto* standard for modeling three-dimensional shapes in Computer Aided Design (CAD) applications.

In this paper, we extend NURBS models to high dimensional scientific data, evaluate different methods for fitting a NURBS model in contrast to a nonrational B-spline model, present adaptive refinement methods to guarantee the accuracy of the model in representing input data, and develop an algorithm to refine locally without resolving the model over the whole input domain or compromising the continuity of the model. The motivation behind this local refinement algorithm is to fit large input domains for which a global solve is expensive, more so if the solve needs to be repeated multiple times to reach a fitting error bound.

2 Background

Parametric representations of curves, surfaces, volumes, and hyper-volumes traditionally involve fitting polynomials to known input points. However, the main drawback of polynomial fitting is that its basis function is 'global'; i.e., the fitted value at a given domain location depends on values from all input points across the whole domain. A solution to this problem is to employ additional basis functions that define a local support of the polynomial fit, such as spline bases [9], and radial basis functions [17]. Here, we choose spline bases for their simplicity and desired features highlighted in the previous section, making them suitable as a new data representation for scientific applications.

2.1 B-spline Approximation

A basis spline (B-spline) function is a piecewise polynomial, where each polynomial piece is defined over a subdomain or partition of the whole input domain. The locations where these partitions meet are termed ***knots***. Knot values are usually stored in D knot vectors, each of the form $T_d = \{t_1, ..., t_{k_d}\}$, where k_d is the number of knots for the d_{th} dimension of domain dimensionality D. Each T_d vector is sorted in nondecreasing order, and defined in parameter space $U \in [0,1]^D$. The domain parameterization function, $M(\bar{x})$, is a mapping function $\mathbb{R}^D \to [0,1]^D$, from a point in input coordinates $\bar{x}$ to a point in parameter space $\bar{u} \in U$. This mapping function is usually determined beforehand, along with knot spacing (uniform vs non-uniform), spline basis degree p, and number of ***control points*** $n = \{n_1, ..., n_D\}$. Control points define the shape of the approximated model, so the main objective of B-spline fitting is to find control point locations, or ***encoding***, that, when decoded, closely represent variations in the input variable field (see Figure 1). In practice, n is usually prespecified by the user, and the number of knots for the d_{th} dimension is directly calculated as $k_d = n_d + p + 1$. Automatically deciding these parameters given input data points is an active research area, and beyond the scope of this paper. The adaptive refinement methods presented here start from an initial number of knots, and control points, then iteratively increase knot resolution where the fitting error is above a certain error metric bound.

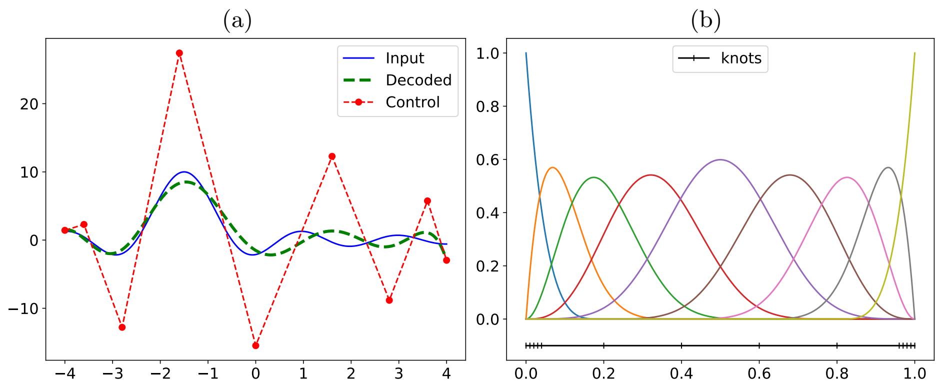

Fig. 1. (a) A 1D curve (blue) and its quartic B-spline approximation (green) using 9 control points (red). (b) A plot showing the 9 4-th degree basis functions (arbitrary colors) used by the B-spline in (a), along with knot locations in parameter space.

A decoded value from a B-spline model is computed as

$$V(\bar{u}) = \sum_{i=1}^{n} P_i \prod_{d=1}^{D} N_{i,p}(u_d), \tag{1}$$

where $V(\bar{u})$ is the decoded hypervolume value at $\bar{u} = \{u_1, ..., u_D\}$, and $N_{i,p}(u_d)$ is the p-degree B-spline basis function for control point P_i at parameter location u_d. $N_{i,p}(\bar{u})$ can be precomputed and stored in memory, or computed as needed on the fly, by the Cox-de Boor recursion formula [7,8].

As seen from Equation 1, decoding one point from the MFA model involves a tensor product of the basis functions with the d-dimensional mesh of control points. Encoding an MFA model for input point field $I = \{y_1, ..., y_m\}$ of size m is traditionally achieved by solving a least squares problem derived from the L_2 norm defining the accuracy of the encoded model by the sum of squared errors (SSE) metric [20],

$$SSE = \sum_{i=1}^{m} \|V(M(x_i)) - I(x_i)\|^2 . \tag{2}$$

2.2 NURBS

NURBS, as the name suggests, is the rational extension of B-splines. It was first introduced and used within CAD software tools because the B-spline formulation is incapable of accurately representing conics [22]. For our specific use case here, using NURBS results in a more compact model than using B-splines.

The difference between the rational equations of NURBS and B-splines is associating a weight variable w_i with each control point P_i. Decoding a NURBS MFA model follows a similar approach to Equation 1 for B-splines, with the exception of substituting $N_{i,p(\bar{u})}$ with rational basis functions $R_{i,p}(\bar{u})$, defined as

$$R_{i,p}(\bar{u}) = \frac{N_{i,p}(\bar{u}) w_i}{\sum_{j=1}^{n} N_{j,p}(\bar{u}) w_j} . \tag{3}$$

NURBS encoding, however, is not as straightforward as the decoding modification. The division in Equation 3 results in a nonlinear problem. That is why most NURBS implementations resort to using uniform weights (all set to 1) that are solved like B-spline models, and then manually tweaked by users for additional model shape control.

3 Adding Weights

There has been some work on approaches for solving the nonlinear problem of finding NURBS control point locations and weights simultaneously. Laurent-Gengoux and Mekhilef [15] manually derive analytical gradients of the nonlinear problem w.r.t. the control point locations and weights, and also the knot locations. The gradients are then employed by a numerical optimization method to minimize a cost function with appropriate geometric continuity constraints. The method relies on a specific complicated and time-consuming optimization algorithm, and the derivation is only valid for cubic (degree 3) NURBS. In order to overcome these shortcomings, Xie et al. [28] propose an alternating linear projection approach, where the control point locations are first determined using least squares, then their corresponding weights are found using conjugate gradient descent optimization methods. These two steps are then repeated until convergence. An alternative approach is an explicit two-step linear scheme proposed by Ma and Kruth [16]. In this explicit method, control point weights are first identified from a homogeneous system using symmetric eigenvalue decomposition and linear programming, and their locations are subsequently found using least squares. A third method involves the use of automatic differentiation to directly calculate the analytical derivative of the error function in Equation 2 w.r.t both P_i and w_i in one step. Other notable approaches that also attempt to directly find control point locations and weights at the same time include the use of metaheuristic techniques, such as evolutionary and swarm intelligence search algorithms [26,23].

3.1 Automatic Differentiation

Automatic differentiation (AD) [21], also known as Algorithmic Differentiation, solves many problems with symbolic and numerical differentiation. It can automatically provide derivatives, high-order derivatives, and partial derivatives with respect to many input parameter functions defined in computer source code [12]. AD approaches typically use a computational graph that is traversed to compute the root node derivative by aggregating the partial derivatives along all paths to leaf nodes, applying the chain rule for every edge weight [4].

The past few years have seen a growing interest in AD from both academia and industry, fueled by a need for generic, user-friendly deep learning toolkits. The backpropagation learning algorithm used to train deep neural networks is a special case of reverse mode AD [24]. Consequently, several libraries available for deep learning also include high performance routines for AD [3,27]. TensorFlow [1] is a Python based deep learning API provided by Google implemented with a GPU backend. In this paper, we extend our previous use of Tensorflow for solving inverse problems [18] to automatically calculate gradients for the NURBS fitting forward model and error metric previously defined in Equation 2.

3.2 Evaluation

In this section we present a comparison of three different methods for fitting an MFA NURBS model: Xie'12 [28], Ma&Kruth'95 [16], and AD [21]. B-spline (unweighted) fitting is also included in the evaluation to assess the benefit of solving for control point weights. Throughout the experiments presented within this paper, we use one synthetic dataset, and one scientific dataset generated from a production HPC simulation code.

Datasets For the synthetic data we use the *sinc* cardinal sine function of the form $y = sin(x)/x$, that we can generate in any dimensionality and resolution. In order to increase the range and slope of the data, we scaled the sinc function by a factor of 10. The 1D sinc function is $f(x) = 10sin(x)/x$. In 2D, $f(x, y) = 10sinc(x)sinc(y)$, and so forth for higher dimensions. The sinc function was

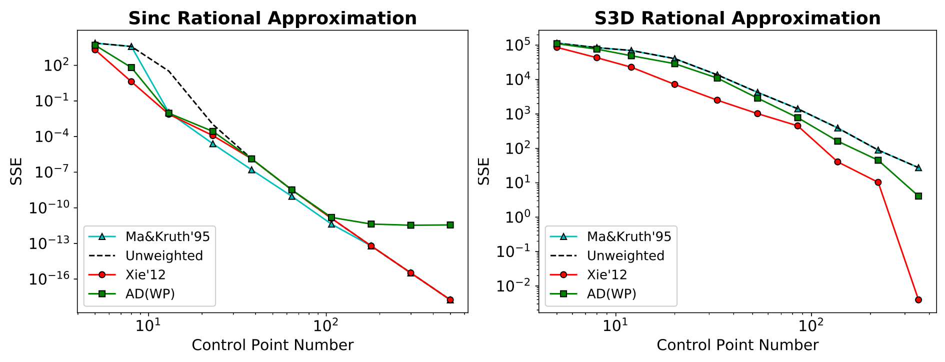

Fig. 2. Rational approximation accuracy plots with increasing control point number. Left plot, synthetic data (sinc). Right plot, combustion simulation data (S3D).

chosen for its smoothness; because of its high degree of continuity, the MFA is able to model such data efficiently. In contrast to the sinc data, *S3D* is a turbulent combustion data set generated by an S3D simulation [6] of fuel jet combustion in the presence of an external cross-flow [13]. This dataset is non-smooth, with sharp edges and high-frequency details, and is representative of actual data one would encounter in scientific experiments or simulations. The domain is 3D (x, y, z) $(704 \times 540 \times 550)$, and the range variable $f(x, y, z)$ is the magnitude of the 3D velocity vector at each domain point. We slice this dataset to produce 1D and 2D cross-sections.

Experiment and Results We ran the four fitting algorithms with sinc, generated for 1D input curve $(m = 1000)$, and S3D 1D curve sliced in the X-axis $(m = 704)$. The experiments were performed by varying the number of control points from the minimum number $(n = p+1)$ to half the input points $(n = m/2)$. For all runs, we use quartic curve fitting $(p = 4)$, and the L-BFGS-B optimizer [5] for the gradients provided by the Xie'12 and AD methods, for which the weights bounds were set within the range $[10^{-4}, 1]$ to keep the weights positive. The plots in Figure 2 report the accuracy of the fitting as defined by the SSE metric in log-log scale.

The left plot shows that all methods perform well on the smooth synthetic input, with rational algorithms generally providing superior approximations, when the ratio of input points to control points, (m/n), is higher; this directly translates to better compression factors. The reason AD accuracy stagnates around $SSE = 10^{-13}$ is because it is running on the GPU and using single-precision (32-bit format), while the other methods are using double-precision (64-bit). S3D results, shown in the right plot of Figure 2, favor iterative gradient-based methods (Xie'12 and AD) over explicit methods (Ma&Kruth'95 and Unweighted) when dealing with nonsmooth real input data.

4 Rational Adaptivity

Next we turn to extending our B-spline adaptive algorithm [19] to NURBS; solving for control point weights in addition to their locations. Adaptivity is achieved through increasing knot resolution where an error metric is above a given error bound ϵ, which, in turn, leads to additional control points in this underrepresented region. Here we use Normalized Squared Error (NSE) as the error metric used for adaptivity, but other metrics can similarly be used depending on the scientific application. We implemented two variants of this approach: one that splits all the knot spans where the user-set error bound is violated, termed ***wga*** for weighted global all; the other approach splits one knot span at each adaptive iteration, termed ***wg1*** for weighted global one. The two variants of our rational adaptive method are presented in Algorithm 1.

Algorithm 1 Global adaptive encoding algorithm

```
function GlobalAdaptive($I, T, \epsilon, splitAll$)
  do
    $R \leftarrow basis(T)$
    $P, w \leftarrow RationalEncode(I, R)$
    $E \leftarrow Decode(P, w, R) - I$
    $NSE \leftarrow E^2/(max(I) - min(I))$
    $toSplit \leftarrow findKnots(NSE, T, \epsilon, splitAll)$
    $T_{new} \leftarrow \emptyset$
    for all $t_i \in T$ do
      $T_{new} = T_{new} \bigcup t_i$
      if $i \in toSplit$ then
        $T_{new} = T_{new} \bigcup \{t_i + ((t_{i+1} - t_i)/2)\}$
    $T = T_{new}$
  while $toSplit \neq \emptyset$
  return $P, w, T$
function findKnots($NSE, T, \epsilon, splitAll$)
  if $splitAll$ then
    return $\{where(NSE > \epsilon, T)\}$          ▷ returns indices in $T$ where $NSE > \epsilon$
  else
    $maxNSE \leftarrow max(NSE)$
    if $maxNSE > \epsilon$ then
      return $\{where(NSE = maxNSE, T)\}$          ▷ returns the index of $maxNSE$ in $T$
    else
      return $\emptyset$
```

The algorithm starts with an initial knot distribution stored in T, then, using one of the methods presented in Section 3, computes a rational encoding, for input field I, in terms of control point locations and weights, P and w. To compute the encoded model accuracy we perform a global decode operation, as specified by Equation 1, and compare the decoded model to the input field. From the calculated error field E, the locations of errors above the error bound ϵ are mapped to knot values in T, and their associated knot spans are split in the middle. This process is repeated until there are no more knot spans to split, which is either due to E being below ϵ everywhere in the input domain, or not enough input points within the spans to be split. The latter scenario means a rational polynomial of degree p is not sufficient to model the discrete input points within the given knot spacing, with the desired error bound ϵ. Remedies to this problem include modifying the overall NURBS degree used for fitting, the use of domain decomposition techniques to assign different degrees to different subdomains, or a hierarchy of different subdomains with varying spline properties at each level of the hierarchy [11]. In this paper, we leave the investigation into NURBS degree and initial conditions for future work. Therefore, the rational adaptive methods, as presented here, might not converge for certain inputs.

4.1 Local Rational Adaptivity

The global adaptive algorithm presented in the previous subsection is computationally inefficient for two main reasons: 1) it includes a global encode and decode at the beginning of every adaptive iteration, and 2) the global rational encode procedure, in Algorithm 1 line 3, ignores the results of the previous iteration, starting from scratch over the whole domain of input points. To address these shortcomings, we develop a local adaptive algorithm that is able to incrementally refine a rational model. The algorithm steps are listed in Algorithm 2, and highlighted in Figure 5.

Algorithm 2 Local adaptive encoding algorithm

```
1:  function LocalAdaptive($I, T, \epsilon, p, n, m$)
2:      $R \leftarrow basis(T)$
3:      $P, w \leftarrow RationalEncode(I, R)$
4:      $E \leftarrow Decode(P, w, R) - I$
5:      $NSE \leftarrow E^2/(max(I) - min(I))$
6:      while $mean(NSE) > \epsilon$ do
7:          $toSplit \leftarrow findKnots(NSE, T, \epsilon, False)$
8:          $T_{new}, P, w, pStart, pEnd \leftarrow knotInsertion(toSplit, T, P, w)$
9:          $R \leftarrow basis(T_{new})$
10:         $localStart \leftarrow max(pStart - p, 1)$          ▷ expand the local domain to include constraints
11:         $localEnd \leftarrow min(pEnd + p, n)$
12:         $R_{local} \leftarrow R[localStart : localEnd]$          ▷ extract local basis from $R$
13:         $I_{local} \leftarrow I[where(R_{local} > 0, I)]$          ▷ extract local input points from $I$
14:         $P_{local}, w_{local} \leftarrow LocalRationalEncode(I_{local}, R_{local}, p)$          ▷ local solve with $p$ constraints
15:         $E_{local} \leftarrow Decode(P_{local}, w_{local}, R_{local}) - I_{local}$
16:         $P[localStart : localEnd] \leftarrow P_{local}$          ▷ update control point locations
17:         $w[localStart : localEnd] \leftarrow w_{local}$          ▷ update control point weights
18:         $NSE[where(R_{local} > 0, NSE)] \leftarrow E^2_{local}/(max(I) - min(I))$          ▷ update error
19:         $T = T_{new}$
20:     return $P, w, T$
```

Our local adaptive method, ***wl1***, relies on two important modifications to Algorithm 1 to address its shortcomings. The global rational encode and decode procedures are replaced with their local counterparts; acting on subdomains of I, R, P, and w. In particular, the rational encode optimization problem is amended to include boundary equality constraints, in order to preserve p-continuity at the local subdomain edges [11] (see Figure 5, bottom row). The other modification incorporates the standard knot insertion algorithm [20] instead of a global encode after splitting a knot span. Standard knot insertion is an algorithm similar to DeCasteljau's method for subdividing Bézier curves [10] in that it adds one knot, and in turn one control point, to a spline model without changing the shape of the decoded curve. For a 1D curve, this is achieved by removing $p-1$ control points, belonging to the knot to be split, and calculating positions for p new control points explicitly using a triangular recursion scheme. In the case of 2D surfaces, $p-1$ control point rows and columns are removed, and p control point rows and columns are added as depicted in Figure 5.

4.2 Evaluation

We compare the three variants of our rational adaptive encoding algorithms (wga, wg1, wl1) with their unweighted counterparts (ga, g1, l1), with varying error bounds, in terms of compression factors (m/n), and total runtime. In order to provide a fair walltime comparison, we unify the encoding procedures to use AD for all the six methods, coupled with a Sequential Least Squares Programming (SLSQP) optimizer [14] that is able to handle constrained nonlinear optimization problems. This approach highlights the flexibility of our AD method for solving either nonlinear problems with constraints, without constraints, or the linear problem of determining just control point locations given their weights, with minimal changes to the underlying forward model, and without needing to change the derivative calculations.

Experiments and Results First, we provide an evaluation for the 1D sinc, and S3D datasets with the same parameters as presented in Section 3. For initial conditions for the adaptive algorithms, we use $p = 4$, and $n = p + 1$. The plots in Figure 3 report the different methods compression factors and run times in log-log scale. The results in Figure 3 further reinforce our finding of rational approximations generally outperforming their equivalent unweighted models, i.e. NURBS are better than B-splines. It is also not surprising that the variants of our adaptive algorithm that split one knot span at a time (wg1, wl1) mostly provide better compression factors (fewer control points), than the variants that split all knot spans that violate the error bound. In fact, the flat portions of wga and ga lines signify these methods 'overshooting,' or splitting more knots than needed. In terms of run time, the rational local method is the worst performing among all the tested methods. This is attributed to the overhead needed for solving the constrained optimization problem which seems to outweigh the benefits gained from a local solve. Moreover, since all methods start from the minimum number of control points, the local method, which refines the decoded curve from previous iterations, does not fare well compared to methods that recompute the whole model at every iterations. In order to support these claims, we conduct an experiment where we compare the rational adaptive methods with increasing number of input points. The plots in Figure 4 show the performance of our rational adaptive algorithms on the S3D 1D curve resampled at different domain spacings, employing cubic spline interpolation, $\epsilon = 10^{-4}$, and $n = 50$ initial control points distributed uniformly along the resampled input domain. It is worth noting, this initial number of control points was chosen arbitrarily

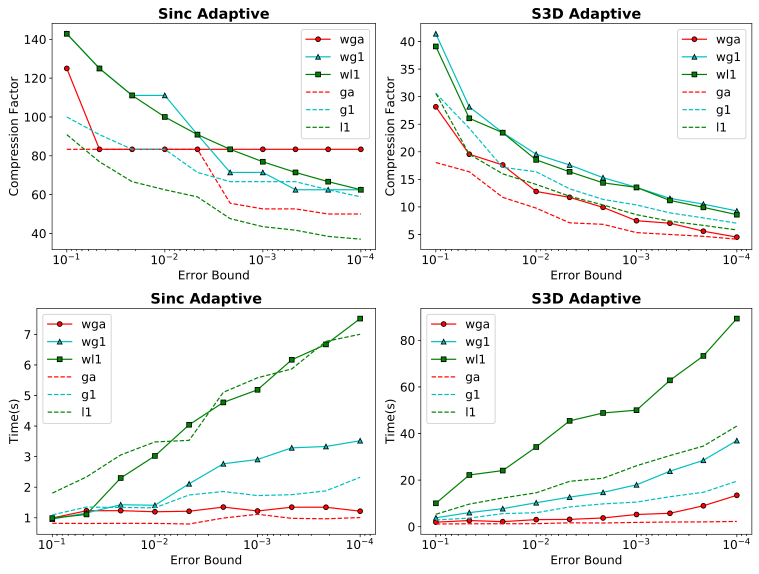

Fig. 3. Adaptive algorithms comparison with varying error bounds. Solid lines are used for the rational methods, while unweighted methods are depicted with dashed lines.

without any manual tuning. From the plots in Figure 4, we can see that wg1 run time starts scaling exponentially at 20k points. As the number of input points increases, while the number of initial control points is kept constant, wg1 requires more iterations to reach the desired fitting accuracy. Since each iteration contains global rational encode and decode operations, wg1 iterations eventually become more expensive than wl1 iterations, which are composed of constrained local rational encode and decode operations.

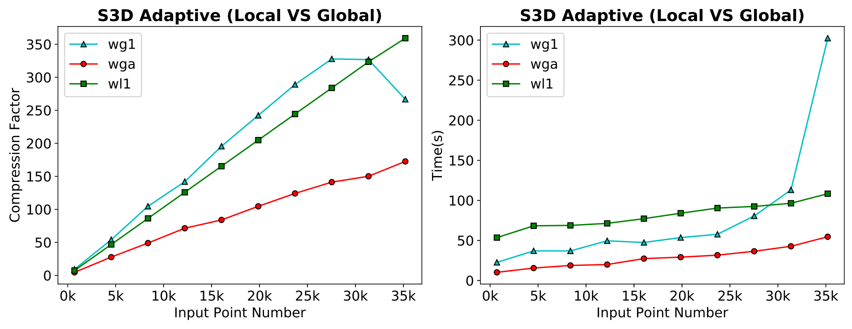

Fig. 4. Rational adaptive approximation of resampled S3D curve.

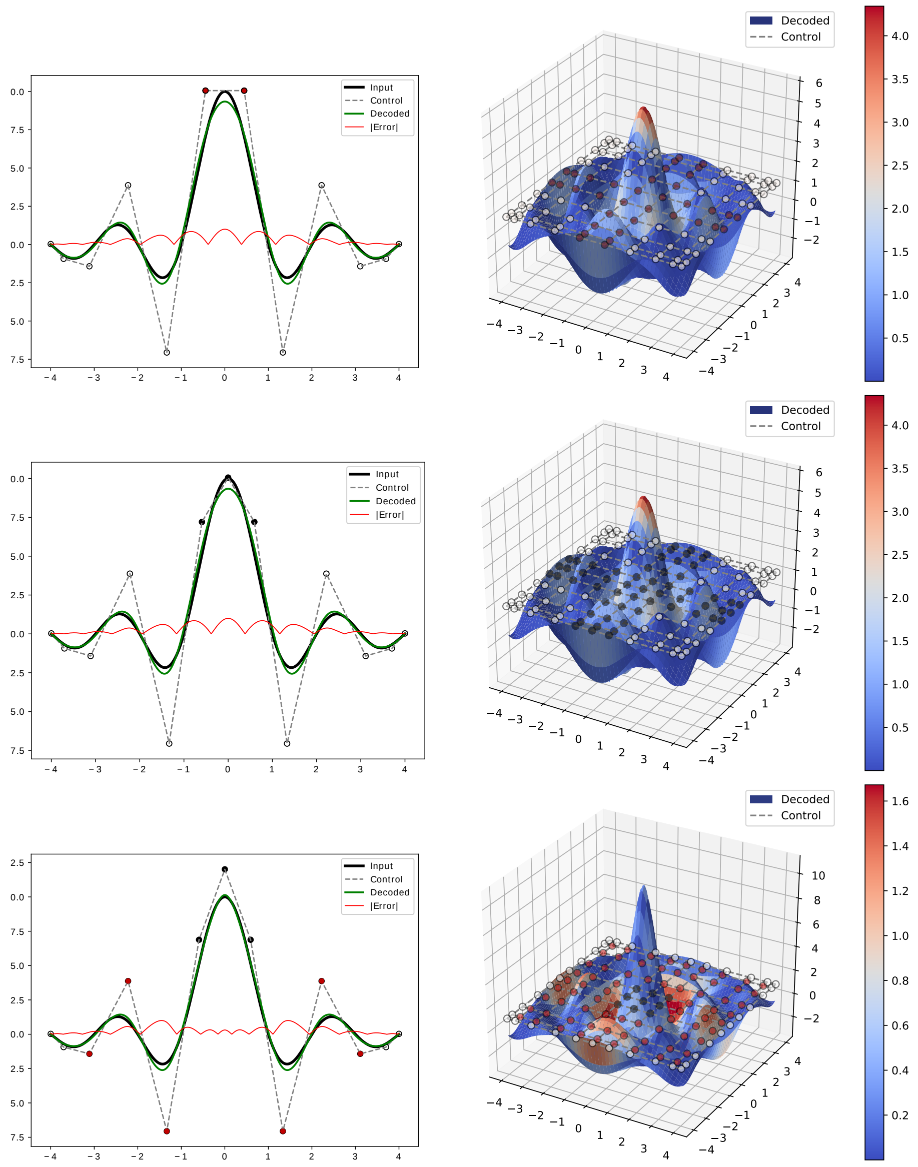

Fig. 5. Local adaptive algorithm steps on 1D (left) and 2D (right) sinc function using a cubic ($p = 3$) NURBS model. The 3D plots on the right show the decoded surface colored by the error magnitude $|E|$, while the plots on the left overlay the error curve (red) with the input (black) and decoded (green) curves. First step: insert a knot where the error is highest by removing $(p - 1)$ control points (top row, red circles) and adding p new control points (middle row, black circles), without changing the decoded curve/surface. Second step: (bottom row) perform a local rational encode for the new control points (black circles) with equality constraints for p boundary control points (red circles). Repeat for next highest error location.

Second, we compare the six adaptive encoding methods on a 2D slice of the S3D dataset, using AD for encoding, SLSQP as the optimization algorithm, $p = 4$, and $n = \{10, 10\}$. These tests show all the methods presented in this paper are directly generalizable to high dimensions, and are able to model real scientific inputs (see Figure 6). The results are presented in Table 1, and are consistent with the results reported for 1D datasets, with the exception of high run time of the wga and ga methods. This is due to the global encoding complexity scaling with the number of control points.

Table 1. 2D S3D Dataset w/ 2.3×10^4 Input Points, Desired $\epsilon = 0.1$

Methods	Output Ctrl Pts	Cmpr Fctr	Actual SSE	Actual $NMSE$	Actual $max(E)$	Time (s)
wga	2.1×10^3	10.8	4.1×10^5	5.2×10^{-2}	3.0×10^{-1}	31899
ga	2.1×10^3	10.8	7.2×10^5	9.1×10^{-2}	4.0×10^{-1}	700
wg1	3.6×10^2	65.8	9.4×10^5	1.1×10^{-1}	3.1×10^{-1}	233
g1	4.0×10^2	59.4	1.1×10^6	1.4×10^{-1}	3.2×10^{-1}	34
wl1	6.2×10^2	38.0	7.5×10^5	9.4×10^{-2}	2.9×10^{-1}	559
l1	6.2×10^2	38.0	9.5×10^5	1.2×10^{-1}	3.2×10^{-1}	129

The results of a sample run of our local adaptive algorithm on S3D is shown in Figure 6. The encoded model is able to capture rapid changes in the input field by increasing knot and control point resolution in high turbulence regions.

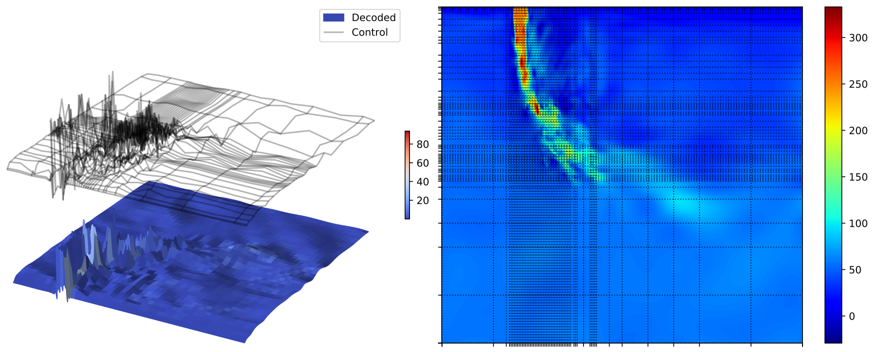

Fig. 6. A visualization of a decoded model for S3D 2D slice. Left, the decoded surface colored by error magnitude, with the control mesh (black) offset upwards for clarity. Right, the decoded surface projected on a 2D image, with the knot locations grid depicted as dotted lines.

The main limitation of the methods presented here, which stems from adopting the standard NURBS model, is their reliance on tensor products of basis function for each domain dimension. While this leads to added cost in terms of extra memory/computation, generally, it results in calculations that can be very easily vectorized. Furthermore, the approximation accuracy should benefit from the overall increased degrees of freedom. The implication of the tensor product formulation for adaptive algorithms is that it requires additional hyper planes of control points in every domain dimension when splitting just one knot span. We are currently working on a T-Splines [2] extension to our work that is compatible with the local adaptive algorithm presented here. The T-Spline formulation will restrict the added control points to the edges of the local region being refined. We are also investigating hybrid local refinement techniques, in which multiple subdomains can be solved in parallel as long as they do not overlap in the unconstrained control points. Additionally, since the quality of the model depends on the nature of the input dataset, we are researching methods for determining initial conditions (parameterization function, knot spacing, number of control points, fitting degree) based on properties of the input data.

5 Conclusion

This paper is primarily concerned with rational approximations, which we show to fit input data more precisely and compactly compared to unweighted ones. Here, we solve a nonlinear problem of finding NURBS optimal control point locations and their associated weights that accurately approximate given input points to user-set error bounds. Additionally, by taking advantage of the local support property of NURBS, we developed an algorithm that is able to locally refine a given approximation on a subset of the input domain. This effectively reduces the computational burden by restricting the iterative gradient-based optimization locally in subdomains of both the approximation and the input domains, and naturally lends the algorithm to a parallel implementation [25].

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
2. Bazilevs, Y., Calo, V.M., Cottrell, J.A., Evans, J.A., Hughes, T.J.R., Lipton, S., Scott, M.A., Sederberg, T.W.: Isogeometric analysis using t-splines. Computer Methods in Applied Mechanics and Engineering **199**(5-8), 229–263 (2010)
3. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: A cpu and gpu math compiler in python. In: Proc. 9th Python in Science Conf. pp. 1–7 (2010)
4. Bischof, C.H., Hovland, P.D., Norris, B.: On the implementation of automatic differentiation tools. Higher-Order and Symbolic Computation **21**(3), 311–331 (2008)
5. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing **16**(5), 1190–1208 (1995)
6. Chen, J.H., Choudhary, A., De Supinski, B., DeVries, M., Hawkes, E.R., Klasky, S., Liao, W.K., Ma, K.L., Mellor-Crummey, J., Podhorszki, N., et al.: Terascale direct

numerical simulations of turbulent combustion using s3d. Computational Science & Discovery **2**(1), 015001 (2009)
7. Cox, M.G.: The numerical evaluation of b-splines. IMA Journal of Applied Mathematics **10**(2), 134–149 (1972)
8. De Boor, C.: On calculating with b-splines. Journal of Approximation theory **6**(1), 50–62 (1972)
9. De Boor, C., De Boor, C., Mathématicien, E.U., De Boor, C., De Boor, C.: A practical guide to splines, vol. 27. Springer-Verlag New York (1978)
10. De Casteljau, P.d.F.: Shape mathematics and CAD, vol. 2. Kogan Page (1986)
11. Forsey, D.R., Bartels, R.H.: Hierarchical b-spline refinement. ACM Siggraph Computer Graphics **22**(4), 205–212 (1988)
12. Griewank, A., Juedes, D., Utke, J.: Algorithm 755: Adol-c: a package for the automatic differentiation of algorithms written in c/c++. ACM Transactions on Mathematical Software (TOMS) **22**(2), 131–167 (1996)
13. Grout, R., Gruber, A., Yoo, C.S., Chen, J.: Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow. Proceedings of the Combustion Institute **33**(1), 1629–1637 (2011)
14. Kraft, D.: A software package for sequential quadratic programming. Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt (1988)
15. Laurent-Gengoux, P., Mekhilef, M.: Optimization of a NURBS representation. Computer-Aided Design **25**(11), 699–710 (1993)
16. Ma, W., Kruth, J.P.: Nurbs curve and surface fitting and interpolation (1995)
17. Mai-Duy, N., Tran-Cong, T.: Approximation of function and its derivatives using radial basis function networks. Applied Mathematical Modelling **27**(3), 197–220 (2003)
18. Nashed, Y.S., Peterka, T., Deng, J., Jacobsen, C.: Distributed automatic differentiation for ptychography. Procedia Computer Science **108**, 404–414 (2017)
19. Peterka, T., Nashed, Y., Grindeanu, I., Mahadevan, V., Yeh, R., Trixoche, X.: Foundations of Multivariate Functional Approximation for Scientific Data. In: Proceedings of 2018 IEEE Symposium on Large Data Analysis and Visualization (2018)
20. Piegl, L., Tiller, W.: The NURBS Book. Springer Science & Business Media (2012)
21. Rall, L.B.: Automatic differentiation: Techniques and applications (1981)
22. Rogers, D.F.: An introduction to NURBS: with historical perspective. Elsevier (2000)
23. Sarfraz, M.: Computer-aided reverse engineering using simulated evolution on NURBS. Virtual and Physical Prototyping **1**(4), 243–257 (2006)
24. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117 (2015)
25. Toselli, A., Widlund, O.: Domain decomposition methods-algorithms and theory, vol. 34. Springer Science & Business Media (2006)
26. Ulker, E.: Nurbs curve fitting using artificial immune system. Int J Innov Comput Inf Control **8**, 2875–87 (2012)
27. Walter, S.F., Lehmann, L.: Algorithmic differentiation in python with algopy. Journal of Computational Science **4**(5), 334–344 (2013)
28. Xie, W.C., Zou, X.F., Yang, J.D., Yang, J.B.: Iteration and optimization scheme for the reconstruction of 3d surfaces based on non-uniform rational B-splines. Computer-Aided Design **44**(11), 1127–1140 (2012)