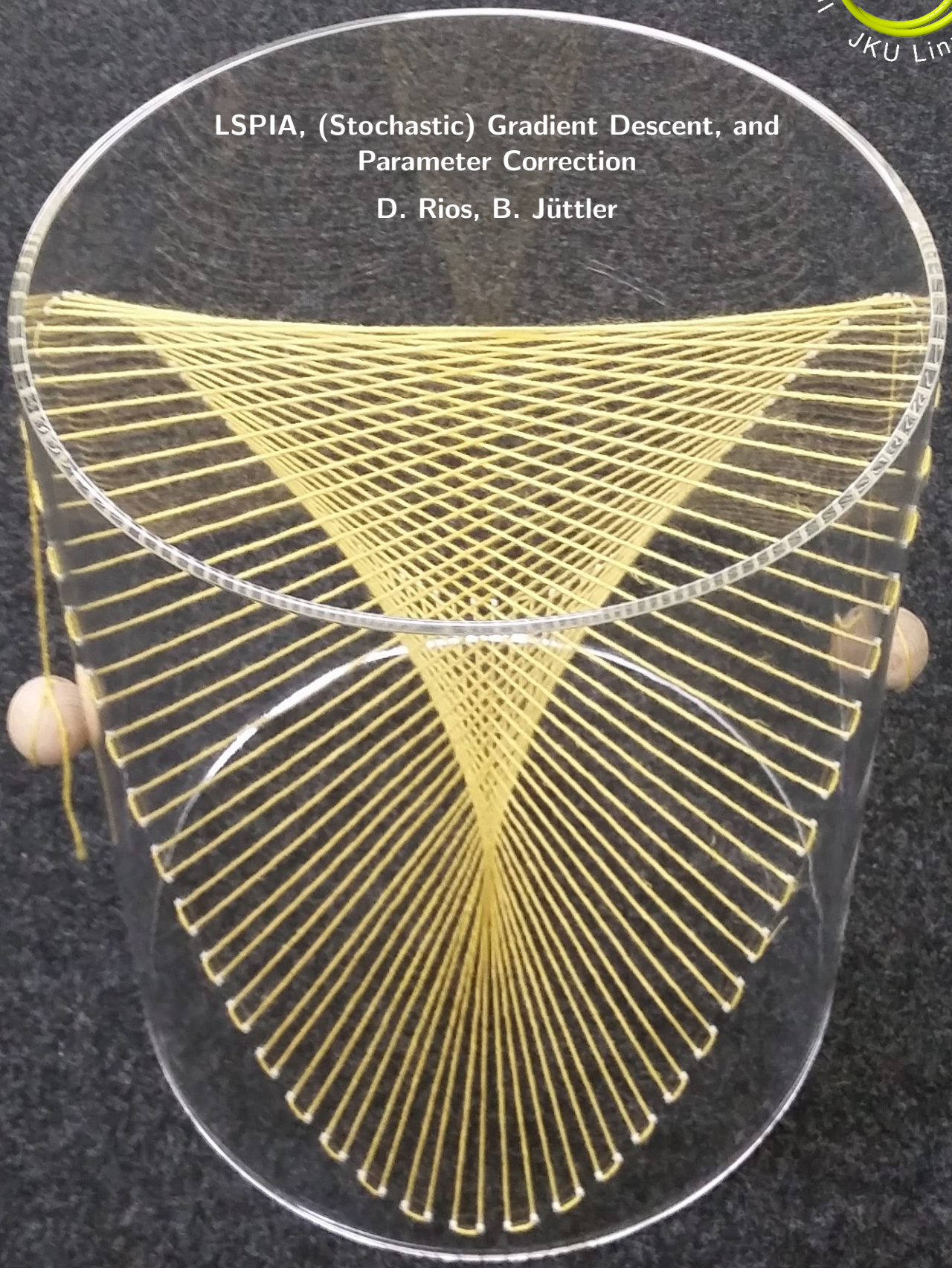
JKU

Institute of Applied Geometry
JKU Linz

LSPIA, (Stochastic) Gradient Descent, and Parameter Correction

D. Rios, B. Jüttler



AG Report No. 94
July 13, 2021
www.ag.jku.at

LSPIA, (Stochastic) Gradient Descent, and Parameter Correction

Dany Rios, Bert Jüttler*

Institute of Applied Geometry, Johannes Kepler University, Linz / Austria

Abstract

We show that the LSPIA method for curve and surface approximation, which was introduced by Deng and Lin (2014), is equivalent to a gradient descent method. We also note that Deng and Lin's results concerning feasible values of the stepsize are directly implied by classical results about convergence properties of the gradient descent method. We propose a modification based on stochastic gradient descent, which lends itself to a realization that employs the technology of neural networks. In addition, we show how to incorporate the optimization of the parameterization of the given data into this framework via parameter correction (PC). This leads to the new LSPIA-PC method and its neural-network based implementation. Numerical experiments indicate that it gives better results than LSPIA with comparable computational costs.

Keywords: progressive iterative approximation, neural networks, parameter correction

1. Introduction

Geometric iterative methods for the construction of curves and surfaces have recently been surveyed by Lin, Maekawa and Deng [1]. These methods are a valuable alternative to linear algebra-based procedures, since they possesses a straightforward geometric interpretation and make it easy to incorporate constraints on control points. Consequently, they have become quite popular and useful for the shape modeling community.

Motivated by de Boor's analysis of Agee's method [2], Lin et al. [3] presented the *progressive and iterative approximation* (PIA) method as an iterative procedure for performing interpolation by non-uniform B-spline curves (surfaces). The method starts with a non-uniform B-spline curve (surface), using the given set of points as its control points. Employing a recursive procedure, it then adjusts the control points, thereby generating a sequence of control polygons that converges to the control polygon of the interpolating non-uniform B-spline curve (surface).

Lin et al. in [4] proved the convergence of the method for any normalized and totally positive basis, which includes the case of B-splines. Later, Lu [5] improved the convergence of the PIA method for normalized totally positive bases by introducing a weighted PIA

Email addresses: dany.rios_rodas.jku.at (Dany Rios), bert.juettler@jku.at (Bert Jüttler*)

method. Moreover, Carnicer et al. [6] established a connection of PIA to the classical Richardson's iteration method.

Unfortunately, in the classical PIA method, the number of control points must be equal to the number of data points. This represents a severe disadvantage when the number of data points is large. In order to overcome this difficulty, Lin and Zhang [7] were the first to present an extended method, which is suitable for a large number of data points. The generated limit curve, however, is not equal to the usual least-squares approximation. Motivated by this deficiency, Deng and Lin in [8] introduced the *Progressive and iterative approximation for least square fitting* (LSPIA) method as another improvement of PIA.

Like in the PIA method, the iterations start with an initial curve (or surface) with control points taken from the set of ordered pair. Sequentially, the LSPIA method adjusts the control points, thereby generating a sequence of curves (surfaces) that converges to the least-squares approximation of the data set. It was shown that the limit curve (surface) is equal to the least-squares approximation, and that the convergence is guaranteed for a sufficiently small stepsize. The authors even succeed to establish an upper bound on the stepsize that guarantees convergence. The optimal choice of the stepsize to ensure a fast convergence rate was further investigated in [9]. A generalization of the LSPIA method is presented by Zhang et al. in [10], using generalized B-splines and mutually different stepsizes. Adapted point sampling methods that ensure a high quality of the fitting results in the context of the LSPIA method were discussed by [11].

In order to accelerate the convergence, Huang et al. [12] propose a new PIA method with memory (MLSPIA). The acceleration of the method is achieved by introducing a weighted sum with three different weights that recover information from the previous step. Another recent extension of the LSPIA method was presented by Wang [13]. The author proposed the extended (E-) LSPIA method by introducing global and local relaxation parameters, where LSPIA is a particular instance.

Extensions of the PIA method for tensor product surfaces were presented in [3, 4, 7, 14, 15, 16]. The case of triangular Bézier surfaces was studied by Chen and Wang [17]. Progressive interpolation (PI) methods for various types subdivision surfaces were investigated also, e.g. [18].

In the present paper, we establish the relation between LSPIA and the *gradient descent* method, which has been overlooked so far. More precisely, *we show that LSPIA is equivalent to a gradient descent method. and that Deng and Lin's results concerning feasible values of the stepsize [8] are directly implied by classical results [19, Theorem I] about convergence properties of the gradient descent method.* Furthermore, we demonstrate that the use of *stochastic* (rather than standard) gradient descent – which admits a simple neural-network-based implementation – leads to a more robust version of LSPIA. We also modify the neural network by adding extra layers that allow us to include the optimization of the parameterization into the framework.

The optimization of the parameterization for curve and surface fitting is one of the fundamental problems in digital geometry reconstruction and has continuously attracted the attention of researchers over the years, see e.g., [20, 21, 22]. We discuss how the resulting LSPIA-PC method, which is the procedure obtained by training the augmented

neural network, is related to the Hoschek's classical technique [23] of *parameter correction* (PC).

The paper is organized as follows. The next section presents the standard LSPIA method and analyzes its relationship to the gradient descent method. Subsequently we introduce new stochastic and mini-batch versions of LSPIA and discuss their realizations via the neural networks framework. In Section 4, we present numerical examples that show the advantages of using the stochastic version of LSPIA (which is an instance of the mini-batch version with batch size $B = 1$) in comparison to the standard method (batch size $B = N$). Finally, we propose the new stochastic LSPIA-PC method as an extension of the stochastic LSPIA, which includes parameter correction. In Section 6 we conclude the paper and propose directions for future work.

2. Data fitting by LSPIA

We consider a set of data $\{d_j\}_{j=1}^N$ in $\mathbb{R}^D$ (typically $1 \leq D \leq 3$) with associated parameter values $t_1, ..., t_N$ satisfying $t_i < t_{i+1}$. We want to fit them by a spline curve

$$x(t) = \sum_{i=1}^{n} \beta_i(t) c_i \ , \tag{1}$$

where the B-splines β_i of degree p are defined with respect to a suitably[1] chosen knot vector. The LSPIA (least-squares progressive iterative approximation) method is a simple iterative method for solving this problem:

1. Initialize the control points by selecting a suitable subset of the data,

$$c_i \leftarrow d_{j_i}, \quad i = 1, \ldots, n,$$

   for a monotonically increasing sub-sequence $\{j_1, \ldots, j_n\} \subseteq \{1, \ldots, N\}$ such that $\beta_i(t_{j_i}) > 0$.
2. Perform the following update of the control points, until a certain termination criterion is satisfied:

$$\begin{aligned} \delta_j &\leftarrow d_j - \sum_{i=1}^{n} \beta_i(t_j) c_i && \text{for} \quad j = 1, \ldots, N \ , \\ \Delta c_i &\leftarrow \mu \sum_{j=1}^{N} \beta_i(t_j) \delta_j && \text{for} \quad i = 1, \ldots, n \ , \\ c_i &\leftarrow c_i + \Delta c_i && \text{for} \quad i = 1, \ldots, n \ . \end{aligned}$$

[1] One needs to make sure that the Schoenberg-Whitney conditions $\beta_i(t_{j_i}) > 0$ are satisfied by a sub-sequence $(j_i)_{i=1,\ldots,n}$. We use open knot vectors with boundary knots $t_0 = 0$ and $t_N = 1$.

It has been shown by Deng and Lin in [8, Theorem 2.4] that the sequence of the control points generated by LSPIA converges to the solution $c = (c_1, \ldots, c_n)$ of the least-squares fitting problem

$$E(c) = \sum_{j=1}^{N} \|d_j - x(t_j)\|_2^2 \to \min \ , \tag{2}$$

provided that the parameter μ is sufficiently small. More precisely, the sufficient condition

$$0 < \mu < \frac{2}{\lambda_{\max}} \tag{3}$$

for the choice of μ has been provided, where $\lambda_{\max}$ is the largest eigenvalue of the symmetric matrix with the elements

$$a_{ij} = \sum_{k=1,\ldots,N} \beta_i(t_k)\beta_j(t_k) \ .$$

Here we add the observation that LSPIA is actually equal to the iterations of the standard gradient descent method, applied to the problem (2). Indeed, a short computation confirms that the gradient of the objective function satisfies

$$\begin{aligned} \nabla E(c) &= \left(-2 \sum_{j=1}^{N} \left(d_j - \sum_{k=1}^{n} \beta_k(t_j) c_k \right) \beta_i(t_j) \right)_{i=1,\ldots,n} \\ &= \left(-2 \sum_{j=1}^{N} \beta_i(t_j) \delta_j \right)_{i=1,\ldots,n} \\ &= -\frac{2}{\mu} \Delta c_i \ , \end{aligned} \tag{4}$$

thus the update of the control points c_i performed by LSPIA in Step 2 adds a scaled multiple of the gradient vector,

$$c_i \leftarrow c_i - \varrho \, \nabla E(c) \ .$$

with stepsize $\varrho = \frac{\mu}{2}$.

According to classical results from optimization theory [19, Theorem I], the gradient descent method converges unconditionally to a local minimum if the step-size satisfies

$$0 < \varrho < \frac{2}{L} \tag{5}$$

where L is the smallest possible Lipschitz constant of the gradient,

$$\|\nabla E(x) - \nabla E(y)\|_2 \leq L \|x - y\|_2 \ .$$

We rewrite the gradient (4) as

$$\nabla E(c) = 2A^T(Ac - d)$$

with

$$A = (\beta_i(t_j))_{j=1,\dots,N;i=1,\dots,n}, \quad c = (c_i)_{i=1,\dots,n}, \quad d = (d_j)_{j=1,\dots,N} \ .$$

Thus

$$\begin{aligned} \|\nabla E(c) - \nabla E(c')\|_2 &= 2\|A^T(Ac - d) - A^T(Ac' - d)\|_2 \\ &= 2\,\|A^T A(c - c')\|_2 \\ &\leq 2\,\|A^T A\|_2\,\|c - c'\|_2 \ , \end{aligned}$$

where the spectral norm $\lambda_{\max} = \|A^T A\|_2$ represents the largest singular value of the matrix A. Consequently, the sufficient condition (3) follows directly from the general bound (5) for the gradient descent method, since the inequalities

$$\varrho = \frac{\mu}{2} < \frac{2}{2\lambda_{\max}}$$

and (3) are equivalent.

3. Stochastic and mini-batch LSPIA

Several natural generalization of the gradient descent method exist, which include the use of adaptive step-size control (not considered here) and stochastic gradient or mini-batch descent. Both variants are based on the observation that the gradient (4) is the sum of partial gradients

$$\nabla_j E(c) = (-2\,\beta_i(t_j)\delta_j)_{i=1,\dots,n} \ . \tag{6}$$

In each step, only a small subset (of size 1 for the stochastic case) of the data is used for estimating the gradient (4):

- *Stochastic Gradient Descent*:
  - Pick up randomly one single index j_0 from the index set $\mathcal{J} = \{1, \dots, N\}$
  - Compute the corresponding update for the control points,

$$\Delta c = -\eta \nabla_{j_0} E(c)$$

  - Repeat until a suitable stopping criterion is satisfied.
- *Mini-Batch Gradient Descent*:
  - Pick randomly a small subset $\mathcal{J}_0 \subset \{1, \dots, N\}$ of batch size B.
  - Compute the corresponding update for the control points,

$$\Delta c = -\eta \frac{1}{B} \sum_{j \in \mathcal{J}_0} \nabla_j E(c) \ .$$

– Repeat until a suitable stopping criterion is satisfied.

Efficient implementations of the resulting *stochastic and mini-batch LSPIA methods* (which include the standard LSPIA method for batch size $B = N$) can be established with the help of *neural networks*. More precisely, we consider a neural network with only one hidden layer and B-spline activation functions.

Formally, a feed-forward neural network is define as a directed graph with ℓ hidden layers of interconnected nodes (neurons). The information is processed forward, which means each neuron feeds the signal produced by a weighted-linear combination into an activation function and then the output is passed to each neuron in the next layer [24]. More precisely, the hidden layers correspond to vectors $a^1, \ldots, a^\ell$, which are evaluated recursively via

$$a^\ell = \sigma_\ell(W^\ell a^{\ell-1} + b^\ell), \ell = 1, \ldots, \ell + 1$$

starting from a vector of input data a^0 and generating a vector of output data $a^{\ell+1}$. The elements w_{ij}^ℓ and b_j^ℓ of the matrices W^ℓ and vectors b^ℓ are called the *weights* of the edges and the *biases* of the nodes, respectively. The functions σ_ℓ are called the *activation functions*.

We use a neural network with only one hidden layer for implementing the stochastic and mini-batch LSPIA method, see Fig. 1. Note that we have a different activation function (i.e., the B-splines β_i) for each neuron.

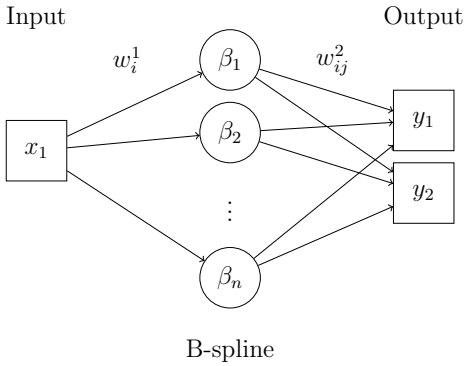


Figure 1: B-spline neural network.

The input vector $a^0 = (t)$ has only one component, and the output $a^2 = x(t)$ possesses dimension D. The weights of the output layer, which are found by training the network, are related to the control points via

$$c_{ij} = w_{ij}^2 \ ,$$

since the output layer (the output of the forward process) satisfies

$$a^2(t) = \left(\sum_{i=0}^{n} \beta_i(t) w_{ij}^2 \right)_{j=1,\ldots,D} . \tag{7}$$

Note that all the biases are zero.

For implementing the stochastic and mini-batch version of LSPIA, we

1. randomly select a single data (t_{j_0}, d_{j_0}) or a small subset (of specified size) of the data, respectively, and
2. use it to compute the (sum of the) partial gradient(s) (6), and to update the weights and biases of the second layer via the *back propagation process*.

This process repeated until a suitable stopping criterion is satisfied.

In our implementation, which is based on the open-source library `Pytorch`, the termination criterion is checked only after each *epoch*, which consists of N/B update steps. This choice ensures that the termination gradient is checked after N partial gradient evaluations.

The iterative method is stopped if there is no longer a significant improvement in the cost function within 100 epochs, i.e.

$$E_{\text{old}} - E_{\text{new}} < e^{-10}$$

in our implementation, or if the maximum (10^3) of epochs is reached. Most codes (including `Pytorch`) that implement this framework perform the minimization of the *mean* square error E/N, and therefore the learning rate $\eta = N\mu/2$ has to be chosen, in order to match the parameter μ of the LSPIA algorithm.

4. Comparison of stochastic, mini-batch, and standard LSPIA

We study the influence of the batch size B and the learning rate to the convergence of the iterative method. We always consider data sets consisting of 128 points, batch sizes $1, 2, 4, \ldots, 128$, and the stopping criterion as described in the previous section.

Example 1 (The zero residual case)**.** We sample the data from the cubic B-spline with 7 control points and uniform knots $(0, 0, 0, 0, 1/5, 2/5, 3/5, 4/5, 1, 1, 1, 1)$ and uniform parameters in the domain $[0, 1]$, which is shown in Fig. 2. The same knot vector and degree is used to define the neural network. The number of required partial gradient evaluations and the resulting mean square error for various values of batch size B and learning rate η are reported in Fig. 3. ♢

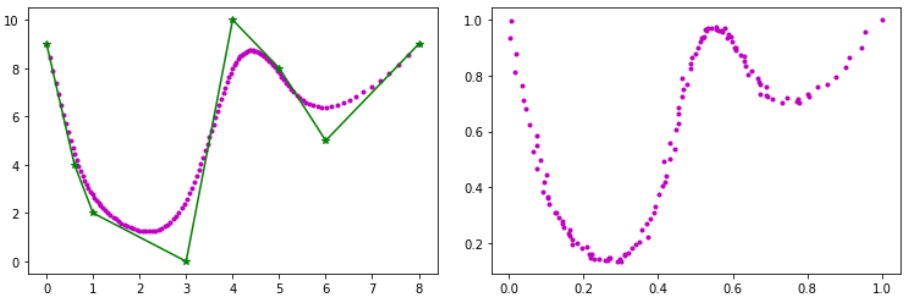


Figure 2: Left: Cubic B-Spline curve with 8 control points. Right: Same curve with noise added.

As shown by this computational result, the performance of the algorithm depends strongly on batch size B and learning rate η. Our goal is to identify the combinations of

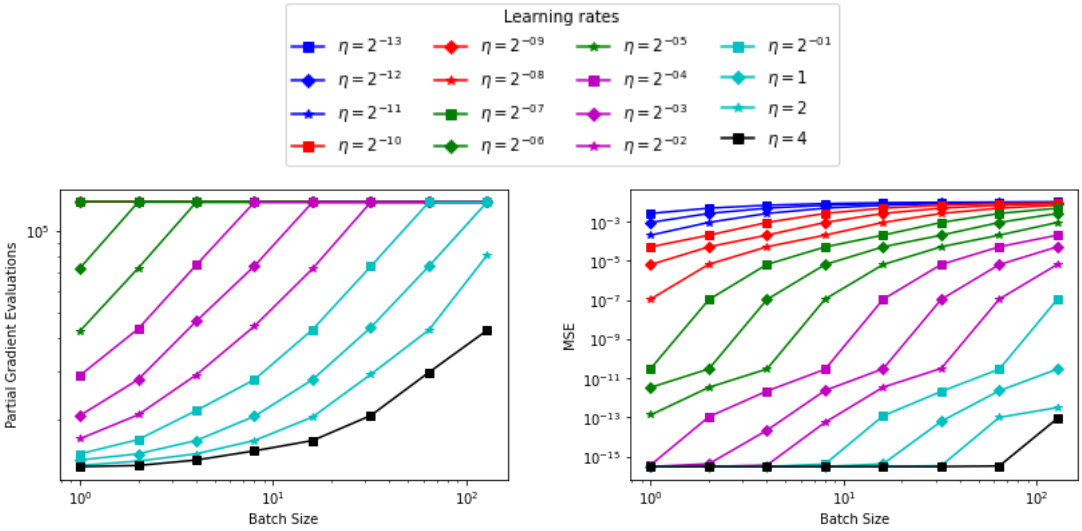


Figure 3: Zero residual case: Number of partial gradient evaluations and MSE per batch size for various values of the learning rate.

both parameters that lead to close-to-minimal value of the objective function while using a relatively small number of partial gradient evaluations. While Fig. 3 does give some preliminary insights, we did not succeed to summarize them in a few observations.

Recall that in our implementation, which is based on the open-source library `Pytorch`, the termination criterion is checked after each *epoch*, which comprises N/B update steps. The total length of the update vectors between two applications of the termination criterion (i.e., per epoch) can thus be estimated as

$$\begin{aligned}\sum_{i=1}^{N/B} \frac{\eta}{B} || \sum_{j \in \mathcal{J}_0} \nabla_j E(c^{(i)}) || \\ \approx \frac{N}{B} \cdot \frac{\eta}{B} \cdot B \cdot \text{“average individual gradient length”}.\end{aligned}$$

This motivates us to introduce the *effective learning rate*

$$\eta^* = \frac{\eta}{B}.$$

The total length of the update vectors per epoch can then be controlled by the effective learning rate and its dependence on the batch size B is no longer significant.

Example 2 (The zero residual case (continued))**.** We revisit the previous example. The number of required partial gradient evaluations and the resulting mean square error for various values of batch size B and effective learning rate $\eta^\star$ are reported in Fig. 4. For suitable values of the step size, the optimization finds the global minimum, i.e., the curve

with almost zero error. The optimization diverges if the stepsize is too large, and it is aborted too early (after 10^3 iterations) if the the stepsize is too small. The method is more robust for a smaller batch size B, where a larger stepsize can be used, resulting in fewer partial gradient evaluations. ◊

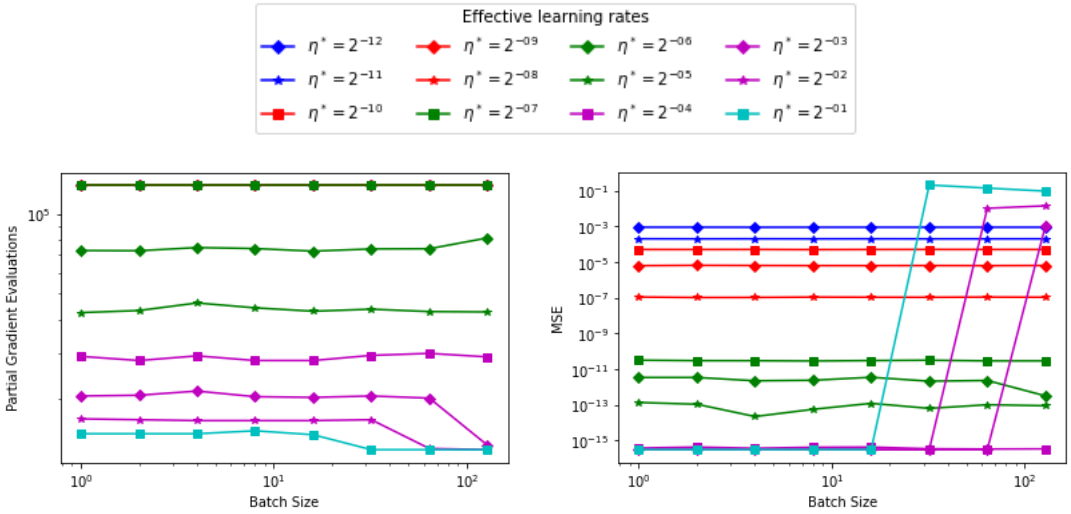


Figure 4: Zero residual case: Number of partial gradient evaluations and MSE per batch size for various values of the effective learning rate.

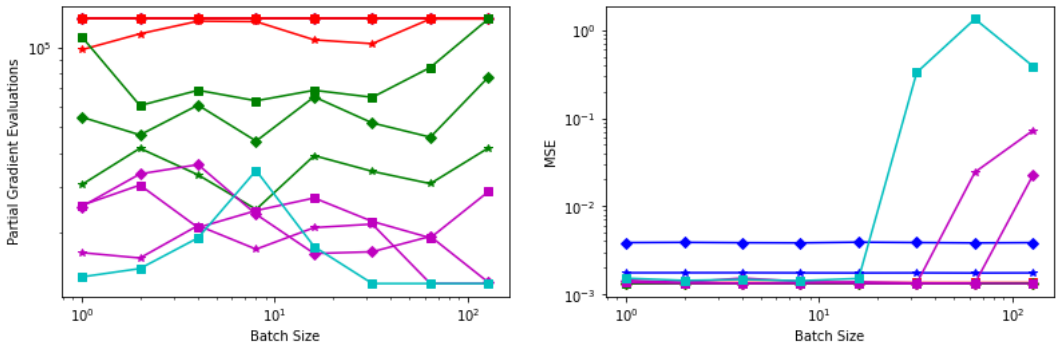


Figure 5: Noisy data: Number of partial gradient evaluations and MSE per batch size for various values of the effective learning rate.

We present a second example:

Example 3 (Data with noise)**.** We use the same configuration for the neural network as in Example 1 but added some noise (using the scaled normal distribution $\frac{1}{20}\mathcal{N}(0,1)$ for each coordinate) to the data set, see Fig. 2. The number of partial gradient evaluations and the resulting mean square error for different batch sizes and effective learning rates is presented in Figure 5. ◊

Three observations are in order:

- First, for configurations of the parameters that lead to optimal fitting results, the total number of partial gradient evaluations is mostly determined by the effective learning rate and does not change much as the batch size B varies.
- Second, the algorithm terminates early for small values of the effective learning rate (since the achieved improvement of the objective function per iteration does not exceed the threshold any more), which leads to fewer partial gradient evaluation but also to less accurate fitting result. In other words, a smaller threshold must be chosen for small learning rates to get the same accuracy.
- Third, the variation of the the batch size B does not have much influence. More precisely, it seems to be beneficial to combine a larger learning rate with a smaller batch size (see Fig. 3), but this effect is caused by the smaller total length of the update vectors between consecutive applications of the termination criterion and it disappears when one considers the effective learning rate. However, the use of a small batch size allows for a larger effective learning rate, making the method more robust.

Finally, we note that use of stochastic (rather than standard) gradient descent is well justified since it possesses approximately the same computational costs.

5. LSPIA-PC: Stochastic LSPIA with parameter correction

We propose an extension of the LSPIA methods, focusing on the stochastic case. Starting from the neural network-based implementation, we add another hidden layer to optimize the parameterization of the curve (1).

More precisely, we re-visit the fitting problem described in Section 2. The given data will be approximated by a spline curve

$$x(t) = \sum_{i=1}^{n} \beta_i(\tau(t)) c_i \ ,$$

with the reparameterization function,

$$\tau(u) = \sum_{k=0}^{m-1} \sigma_k(u) r_k \ ,$$

where σ_k are piecewise linear step functions defined by

$$\sigma_k(u) = \max\left(0, \min\left(1, m\Big(u - \frac{k}{m}\Big)\right)\right) \ ,$$

see Fig. 6.

Two constraints are imposed on the weights r_k. The first constraint ensures that the reparameterization τ is increasing monotonically. The second one guarantees that the transformed parameter belongs to the closed interval $[0, 1]$. More precisely, we ensure that the weights r_k satisfy

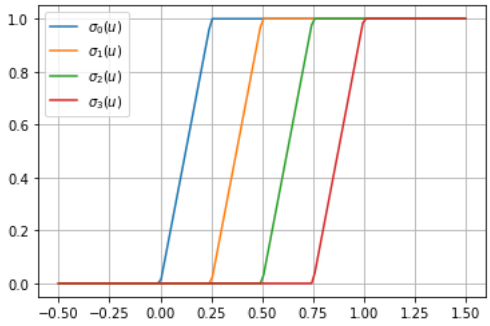


Figure 6: Four step functions σ_k defined over the interval $[0, 1]$.

- $r_k > 0$, and
- $\sum_{k=0}^{m} r_k = 1$.

Using a suitable neural network, we compute the weights r_k and the control points c_i by solving the constrained optimization problem

$$E(r, c) = \sum_{j=1}^{N} \|d_j - x(\tau(t_j))\|_2^2 \to \min \ , \tag{8}$$

using the following constrained gradient descent method:

1. We fix $m > 1$ step functions σ_k in the interval $[0, 1]$ and initialize the weights r_k,
$$r_k \leftarrow \frac{1}{m}, \quad k = 0, \ldots, m-1 \ ,$$
2. We initialize the control points by selecting a suitable subset of the data,
$$c_i \leftarrow d_{j_i}, \quad i = 1, \ldots, n \ ,$$
for a monotonically increasing sub-sequence $\{j_1, \ldots, j_n\} \subseteq \{1, \ldots, N\}$ such that $\beta_i(t_{j_i}) > 0$.
3. We apply the following update steps to the weights r_k and to the control points c_i:

$$\begin{aligned}
\delta_j &\leftarrow d_j - \sum_{i=1}^{n} \beta_i(\tau(t_j)) c_i, && j = 1, \ldots, N \ , \\
\Delta r_k &\leftarrow \mu \sum_{j=1}^{N} \sigma_k(t_j) \Big(\sum_{i=1}^{n} \dot{\beta}_i(\tau(t_j)) c_i \Big) \cdot \delta_j, && k = 0, \ldots, m-1 \ , \\
r_k &\leftarrow r_k + \Delta r_k, && k = 0, \ldots, m-1 \ , \\
\Delta c_i &\leftarrow \mu \sum_{j=0}^{N} \beta_i(\tau(t_j)) \delta_j, && i = 1, \ldots, n \ , \\
c_i &\leftarrow c_i + \Delta c_i, && i = 1, \ldots, n \ .
\end{aligned}$$

4. We clamp the parameter for the first layer $r_k \leftarrow \max(r_k, \epsilon)$, where ϵ is a small non-negative constant.
5. We normalize the weights r_k by

$$r_k \leftarrow \frac{r_k}{\sum_{k=0}^{m-1} r_k}, \qquad\qquad k = 0, \ldots, m-1 \ .$$

6. We continue with step 3 if the stopping criterion is not satisfied.

Note that the update step of the weights r_k is based on the projections of the error vectors δ_j onto the tangent vectors of curve. Since this is similar to Hoschek's method of *parameter correction* (PC) [25], we call the resulting procedure the LSPIA-PC method.

This method can be seen as the training process for a neural network possessing three hidden layers, see Fig. 7. The weights w_i^1 and w_i^3 are fixed parameters equal to 1 and weights w_i^2 and w_{ij}^4 are the trainable weights r_k and c_i, respectively. The corresponding activation functions for the first and third layer are the step functions σ_k and the B-splines β_i, respectively. While the training method described above is in fact the standard (i.e., full batch size) gradient descent, we employ stochastic gradient descent (batch size $B = 1$) to generate approximations of the trainable weights. This is motivated by the observations from the previous section.

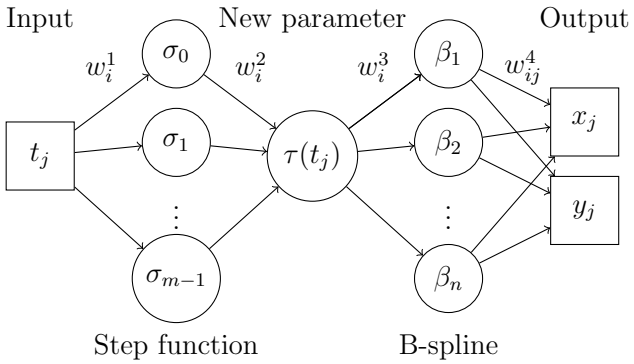


Figure 7: B-spline neural network plus reparameterization

In the remainder of this section, we consider the same data sets as in the previous section 4, and we investigate the influence of the number m of step functions σ_k and of the learning rate to the convergence of the iterative method. We always consider data sets consisting of 128 points, number of step functions ranging from $1, 2, 4$ to 128, and we use the same stopping criterion as in the previous section. The figures report the number of partial gradient evaluations and the resulting mean square error.

Example 4 (Zero residual case)**.** Again we consider the data sampled from a cubic curve. However, the parameters are now assigned by the chord-length parameterization. Figure 8

reports the number of partial gradient evaluations and the resulting mean square error for different number of step functions and effective learning rates. The best results (which are two orders of magnitude lower than the error without parameter correction[2]) are obtained for medium to high numbers m of step functions ($\geq N/2$), independently of the chosen learning rate. Theoretically one would expect to arrive at a result with zero error (which should be the same as the one in Figure 3, where we used the original parameterization of the data, which was also used the sampling) for $m = N$, but the optimization did not reach this global minimum. $\Diamond$

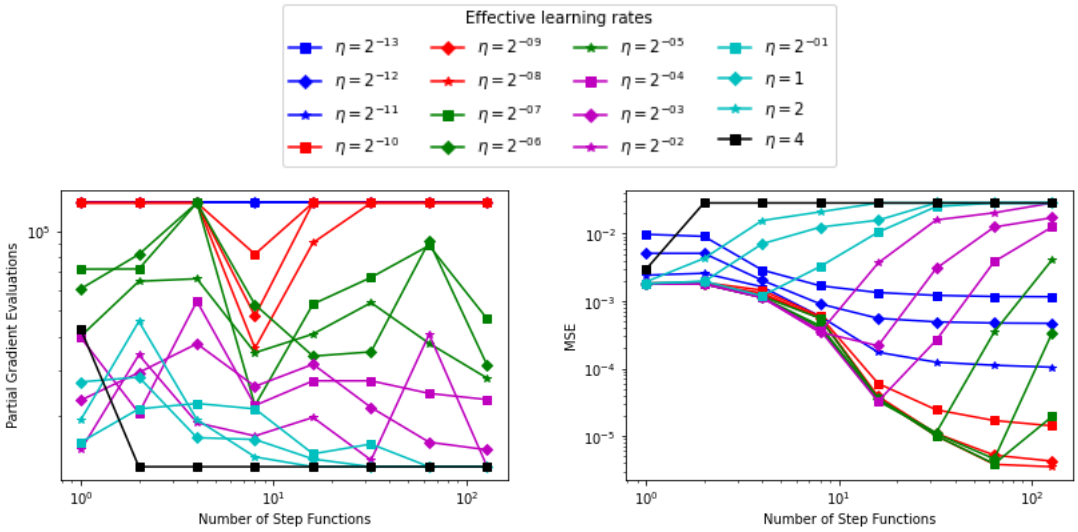


Figure 8: Zero residual case: Number of partial gradient evaluations and MSE per number of step functions in the first layer for various values of the effective learning rate.

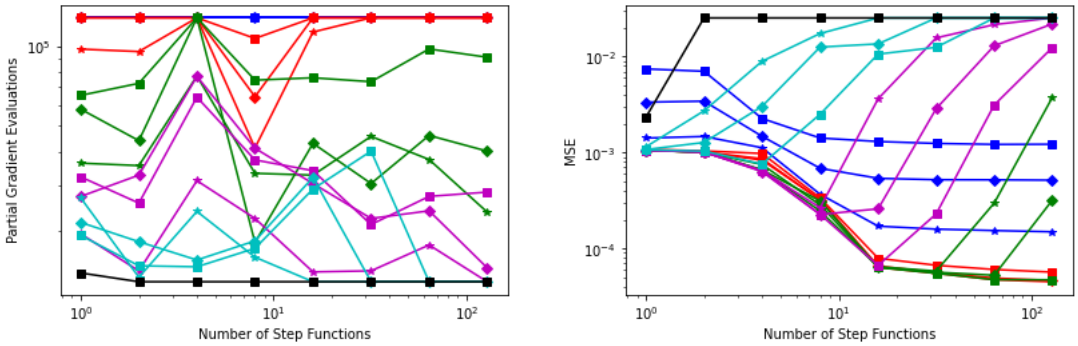


Figure 9: Noisy data: Number of partial gradient evaluations and MSE per number of step function in the first layer for various values of the effective learning rate.

Example 5 (Noisy data)**.** Now we turn to the data with noise, again with a chord-length parameterization. Figure 9 reports the number of partial gradient evaluations and the

[2]This is included in the experiment by considering the case of only one step function, i.e. $m = 1$.

resulting mean square error for different number of step functions and effective learning rates. The best fitting result shows an improvement by one order of magnitude with respect to the MSE without parameter correction (which is again included for $m = 1$). Note that it suffices to use a relatively small number (only 16) of step functions to get a near-optimal result. $\Diamond$

The following observations are in order:

- The use of LSPIA-PC substantially improves the fitting result. The result of the LSPIA algorithm (which is included in the graphs as the limit case of only 1 step function) is improved by almost 3 orders of magnitude in the first case, and by more than one order of magnitude in the second case.

- The method reaches the best accuracy of the fitting result for a large number of step functions ($m \geq N/2$) and a fairly small effective learning rate. Fewer step functions are sufficient for noisy data.

- In terms of computational effort, the LSPIA-PC method needs a similar number of partial gradient evaluations than the standard LSPIA algorithm.

6. Conclusions

It is well understood that the progressive and iterative (PIA) method of Lin, Wang and Dong in 2004 [3] is closely related to the use of Richardson's method for solving the associated linear system of equations [6]. More recently, Deng and Lin (2014) have introduced the least-squares (LS) PIA method, which improves the capabilities with respect to the size of the data [8]. No relation to classical methods for solving the fitting problem had been established so far. In the present paper we provided the missing link by establishing a connection to the standard gradient descent method. We also noted that Deng and Lin's observations concerning the feasible values of the stepsize are implied by classical results from the theory of the gradient descent method.

We also proposed a modification based on stochastic gradient descent, which lends itself naturally to a realization that employs the technology of neural networks. Moreover we showed how to incorporate the optimization of the parameterization of the given data into this framework via parameter correction (PC). According to the presented numerical experiments, for non-smooth and noisy data, the resulting new LSPIA-PC method and its neural-network based implementation gives better results than LSPIA with comparable computational costs.

In the future work we plan to improve the framework even further by leveraging the wealth of knowledge that is available in the field of machine learning, especially with respect to the training algorithms. For instance, the use of Nesterov accelerated gradient methods or similar techniques that employ the concept of momentum, which is a well-established approach to improve the speed of convergence [26], might be related to the memory-based (M) LSPIA variant of Huang and Wang [12]. Last, but not least, the extension is the approach to the case of surface fitting in promising and of vital interest.

References

[1] H. Lin, T. Maekawa, C. Deng, Survey on geometric iterative methods and their applications, Computer-Aided Design 95 (2018) 40–51.

[2] C. de Boor, How does Agee's smoothing method work?, in: ARO report 79-3, Proc. 1979 Army Numerical Analysis and Computers Conference, 1979, pp. 299–302.

[3] H. Lin, G. Wang, C. Dong, Constructing iterative non-uniform B-spline curve and surface to fit data points, Science in China Series: Information Sciences 47 (3) (2004) 315–331.

[4] H.-W. Lin, H.-J. Bao, G.-J. Wang, Totally positive bases and progressive iteration approximation, Computers & Mathematics with Applications 50 (3-4) (2005) 575–586.

[5] L. Lu, Weighted progressive iteration approximation and convergence analysis, Computer Aided Geometric Design 27 (2) (2010) 129–137.

[6] J. Carnicer, J. Delgado, J. Peña, On the progressive iteration approximation property and alternative iterations, Computer Aided Geometric Design 28 (9) (2011) 523–526.

[7] H. Lin, Z. Zhang, An extended iterative format for the progressive-iteration approximation, Computers and Graphics 35 (5) (2011) 967 – 975.

[8] C. Deng, H. Lin, Progressive and iterative approximation for least squares B-spline curve and surface fitting, Computer-Aided Design 47 (2014) 32–44.

[9] A. Ebrahimi, G. Loghmani, A composite iterative procedure with fast convergence rate for the progressive-iteration approximation of curves, Journal of Computational and Applied Mathematics 359 (2019) 1–15.

[10] L. Zhang, J. Tan, X. Ge, G. Zheng, Generalized B-splines' geometric iterative fitting method with mutually different weights, Journal of Computational and Applied Mathematics 329 (2018) 331–343.

[11] L. Lu, S. Zhao, High-quality point sampling for B-spline fitting of parametric curves with feature recognition, Journal of Computational and Applied Mathematics 345 (2019) 286–294.

[12] Z.-D. Huang, H.-D. Wang, On a progressive and iterative approximation method with memory for least square fitting, Computer Aided Geometric Design 82 (2020) 101931.

[13] H. Wang, On extended progressive and iterative approximation for least squares fitting, The Visual Computer (2021) 1–12.

[14] H. Lin, Local progressive-iterative approximation format for blending curves and patches, Computer Aided Geometric Design 27 (4) (2010) 322–339.

[15] C. Liu, Z. Liu, X. Han, Preconditioned progressive iterative approximation for tensor product Bézier patches, Mathematics and Computers in Simulation 185 (2021) 372–383.

[16] M. Liu, B. Li, Q. Guo, C. Zhu, P. Hu, Y. Shao, Progressive iterative approximation for regularized least square bivariate B-spline surface fitting, Journal of Computational and Applied Mathematics 327 (2018) 175–187.

[17] C. Liu, X. Han, J. Li, Preconditioned progressive iterative approximation for triangular Bézier patches and its application, Journal of Computational and Applied Mathematics 366 (2020).

[18] Z. Chen, X. Luo, L. Tan, B. Ye, J. Chen, Progressive interpolation based on Catmull-Clark subdivision surfaces, Computer Graphics Forum 27 (7) (2008) 1823–1827.

[19] B. T. Polyak, Introduction to Optimization, Translation Series in Mathematics and Engineering, Optimization Software Inc., Publications Division, New York, 1987.

[20] V. Weiss, L. Andor, G. Renner, T. Várady, Advanced surface fitting techniques, Computer Aided Geometric Design 19 (1) (2002) 19–42.

[21] E. Saux, M. Daniel, An improved Hoschek intrinsic parametrization, Computer Aided Geometric Design 20 (8) (2003) 513–521.

[22] W. Zheng, P. Bo, Y. Liu, W. Wang, Fast B-spline curve fitting by L-BFGS, Computer Aided Geometric Design 29 (7) (2012) 448–462.

[23] J. Hoschek, Intrinsic parametrization for approximation, Computer Aided Geometric Design 5 (1) (1988) 27–31.

[24] C. F. Higham, D. J. Higham, Deep learning: An introduction for applied mathematicians, SIAM Review 61 (4) (2019) 860–891.

[25] J. Hoschek, D. Lasser, Fundamentals of computer aided geometric design, AK Peters, Ltd., 1993.
[26] N. Qian, On the momentum term in gradient descent learning algorithms, Neural Networks 12 (1) (1999) 145–151.