MOX-Report No. 05/2025

Greedy reconstruction algorithms for function approximation

Buchwald, S.; Ciaramella, G.; Verani, M.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it https://mox.polimi.it

Greedy Reconstruction Algorithms for Function Approximation

Simon Buchwald[1†], Gabriele Ciaramella[2*†], Marco Verani[2†]

[1]Department of Mathematics, Universität Konstanz, Universitätsstr. 10, Konstanz, 78464, Germany.
[2]MOX, Dipartimento di Matematica, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano, Piazza Leonardo da Vinci 32, Italy
GC and MV are members of the GNCS Indam group.

*Corresponding author(s). E-mail(s): gabriele.ciaramella@polimi.it;
Contributing authors: simon.buchwald@uni-konstanz.de;
marco.verani@polimi.it;
†These authors contributed equally to this work.

Abstract

Two key elements in any function approximation problem are the selection of data points and the choice of the structure of the ansatz within a given family of approximation functions. This paper is devoted to the development and analysis of greedy reconstruction algorithms that address both aspects to improve approximation accuracy and efficiency. The general idea of these methods is to select an optimal set of data points while simultaneously identifying a minimal structure that is able to accurately approximate the selected data. Theoretical and numerical studies on polynomial interpolation and function approximation by neural networks demonstrate the efficiency of the proposed algorithms.

Keywords: Function approximation, polynomial interpolation, data-driven methods, neural networks, greedy reconstruction algorithms

1 Introduction

When dealing with approximation problems, one needs to consider two main aspects: the computation (or selection) of the data and the optimization of the structure of the approximation ansatz, within a given family of approximation functions. For instance,

the first issue is known for polynomial approximation, where one searches for a set of interpolating nodes that leads to the best approximation quality (see, e.g., [1, 2]). Here, while the structure of the approximation ansatz is fixed (a polynomial of a given order), the computation of the data (interpolating nodes) is a crucial aspect that strongly affects the approximation error. Another prominent field where both aspects play an important role is machine learning. Before training a neural network (the approximation ansatz) for a specific task, one must determine its structure (number and dimensions of the network layers) and choose an appropriate set of training data to avoid over/underfitting; see, e.g., [3–5]. This paper deals with developing and analyzing greedy algorithms designed to handle these aspects. More specifically, given the task of finding an approximation (a surrogate) $g_\star$ of a function f, the goal of our methods is to construct a set of data points $\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^K$ while optimizing the structure of the ansatz set $\mathcal{G}$, within a given family of approximation functions, in which one searches for $g_\star$. Given these two objects, one obtains $g_\star$ by computing an element in $\mathcal{G}$ that minimizes the distance from f when evaluated at the points $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$.

Our new greedy algorithms are inspired by those introduced in [6–10] in the field of Hamiltonian identification and inverse problems, and can be divided into two classes. A first class, closer to the methods developed in [6–10], splits the process into online and offline phases. We call them greedy reconstruction (GR) algorithms. Here, the evaluation of f is allowed only in an online phase (in which $g_\star$ is computed) after the set $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$ and the structure of $\mathcal{G}$ (within a given family of approximation functions) have been designed by a GR algorithm in an offline phase. Note that GR algorithms do not use f to build $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$, and the data set $\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^K$ is built only afterwards. Moreover, we will also introduce an optimized version of GR, called OGR, where the greedy character of the method is fully exploited and extended to better optimize the structure of the approximation ansatz. The second class assumes that a (very) large data set $\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^J$, with $J \gg K$, is given. We call algorithms in this second class Data Greedy Reconstruction (DGR) algorithms. These aim to select the smallest subset of K elements from $\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^J$ while simultaneously optimizing the structure of $\mathcal{G}$. Note that, unlike the first class, DGR strategies use the (available) data $\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^J$, but select only those that are needed to build $g_\star$.

To study the applicability of our new numerical strategies and analyze their performance, we consider two classes of problems: polynomial interpolation and function approximation by neural networks. Since polynomial interpolation is a well-understood branch of function approximation problems, it represents a concrete benchmark for studying our GR, OGR and DGR algorithms and obtaining concrete theoretical results. In particular, we prove that GR and OGR compute exactly a set of (tensor-product) Leja points (see, e.g, [11–14] and references therein). Moreover, we show that for polynomial interpolation, OGR is equivalent to the well-known Empirical Interpolation Method (EIM); see, e.g,. [15–20] and references therein. Although the two methods can be equivalent, GR, OGR and DGR have one main advantage over EIM: EIM (and also other greedy algorithms used for function approximation, see, e.g., [21]) is limited to linear parametrizations of the ansatz function, while GR, OGR and DGR can also handle nonlinear parameterizations, such as weights and biases of neural networks. This is exactly the second class of problems to which we apply our greedy

algorithms, and where it is possible to see how nicely they can simultaneously select the data set and optimize the network structure. In particular, we develop a combination of OGR and DGR tailored for neural network applications, called Network OGR (NOGR) algorithm. We provide a detailed discussion about the network structure optimization and illustrate how our algorithms can optimize the depth and number of neurons of a network. Our paper is organized as follows: in Section 2 we set our notation. In Section 3, we introduce our GR algorithms in a general setting. In Section 4, the behavior of our GR algorithms for polynomial interpolation (in univariate and multivariate settings) is studied. In Section 5, we detail our GR algorithms for function approximation by neural networks and study their behavior through extensive numerical experiments. Finally, we present our conclusions in Section 6.

2 Notation

We denote by $\widetilde{x}_j \in \mathbb{R}$ and $\widetilde{\boldsymbol{x}}_j = [\widetilde{x}_{j,1}, \ldots, \widetilde{x}_{j,N}]^\top \in \mathbb{R}^N$ any interpolation point, while using x and $\boldsymbol{x} = [x_1, \ldots, x_N]^\top$ for general points in $\mathbb{R}$ and $\mathbb{R}^N$, respectively. For $N, M \in \mathbb{N}$, we let $\mathcal{G} = \mathcal{G}(X, \mathbb{R}^M)$ be a class of functions over some closed and bounded set $X \subset \mathbb{R}^N$. In particular, for $M = 1$ we denote by $\mathcal{P}_n := \mathcal{P}_n(X)$ the space of polynomials p over X with degree $\deg(p) \leq n$, meaning that $p : X \to \mathbb{R}$, $x \mapsto p(x) = \sum_{|\boldsymbol{\beta}|=0}^{n} a_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}}$, for coefficients $a_{\boldsymbol{\beta}} \in \mathbb{R}$ with $\boldsymbol{\beta} \in \mathbb{N}^N$, $|\boldsymbol{\beta}| = \sum_{i=1}^{N} \beta_i$, and $\boldsymbol{x}^{\boldsymbol{\beta}} = \boldsymbol{x}^{(\beta_1,\ldots,\beta_N)} := x_1^{\beta_1} \cdots x_N^{\beta_N}$. For a monomial $a_{\boldsymbol{\beta}} x^{\boldsymbol{\beta}}$, we call $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N) \in \mathbb{N}^N$ its componentwise degree.

We denote by $F_{W,b}$ a residual neural network of depth m, with input size $N \in \mathbb{N}$, output size $M \in \mathbb{N}$ and hidden layers of (the same) width $d \in \mathbb{N}$, represented by weights $W = (W^{[1]}, \ldots, W^{[m]}) \in \mathbb{R}^{N\times d} \times \cdots \times \mathbb{R}^{d\times M}$ and biases $b = (\boldsymbol{b}^{[1]}, \ldots, \boldsymbol{b}^{[m]}) \in \mathbb{R}^d \times \cdots \times \mathbb{R}^d \times \mathbb{R}^M$. We call $\mathcal{W}(\boldsymbol{d}) := (\mathbb{R}^{N\times d} \times \cdots \times \mathbb{R}^{d\times M}) \times (\mathbb{R}^d \times \cdots \times \mathbb{R}^d \times \mathbb{R}^M)$ the joint space of the weight matrices W and the bias vectors b.

3 Greedy algorithms for function approximation

For a (finite-dimensional) function space $\mathcal{G}$ over $X \subset \mathbb{R}^N$, the interpolant of a function $f : X \to \mathbb{R}^M$ for a set of points $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K \subset X$ is the element $g_\star \in \mathcal{G}$ such that $g_\star(\widetilde{\boldsymbol{x}}_j) = f(\widetilde{\boldsymbol{x}}_j)$, $j = 1, \ldots, K$. A general approach of finding this interpolant is to solve the least-squares problem

$$\min_{g\in\mathcal{G}} \sum_{j=1}^{K} \|f(\widetilde{\boldsymbol{x}}_j) - g(\widetilde{\boldsymbol{x}}_j)\|_2^2. \tag{1}$$

The overall goal is to find a $g_\star$ that approximates f well at all other points $\boldsymbol{x} \in X \setminus \{\widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_K\}$. In other words, the interpolant $g_\star$ must ideally solve

$$\min_{g\in\mathcal{G}} \max_{\boldsymbol{x}\in X} \|f(\boldsymbol{x}) - g(\boldsymbol{x})\|_2^2.$$

Algorithm 1 GR for function approximation

Require: A set $\mathcal{B} = \{g_1, \dots, g_K\} \subset \mathcal{G}$ with corresponding parameterizations $\{\widetilde{\boldsymbol{\alpha}}^1, \dots, \widetilde{\boldsymbol{\alpha}}^K\} \subset \mathbb{R}^{\widetilde{K}}$ with $K, \widetilde{K} \in \mathbb{N}$ with $\widetilde{K} \leq K$ and such that $g_k = g_{\widetilde{\boldsymbol{\alpha}}^k}$ for $k = 1, \dots, K$.

Set $k = 1$ and find $\widetilde{\boldsymbol{x}}_1$ that solves

$$\max_{\boldsymbol{x} \in X} \|g_1(\boldsymbol{x})\|_2^2. \tag{2}$$

while $k \leq K - 1$ **do**

Fitting step: Find an $\boldsymbol{\alpha}^k$ that solves

$$\min_{\boldsymbol{\alpha} \in \operatorname{span}(\widetilde{\boldsymbol{\alpha}}_i)_{i=1}^k} \sum_{j=1}^{k} \|g_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{x}}_j) - g_{k+1}(\widetilde{\boldsymbol{x}}_j)\|_2^2. \tag{3}$$

Splitting step: Find an $\widetilde{\boldsymbol{x}}_{k+1}$ that solves

$$\max_{\boldsymbol{x} \in X} \|g_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - g_{k+1}(\boldsymbol{x})\|_2^2. \tag{4}$$

Update $k \leftarrow k + 1$.

end while

In that regard, the choice of the points $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$ plays a crucial role. One reasonable assumption is that for an increasing number of interpolation points K, the approximation error $\max_{\boldsymbol{x} \in X} \|f(\boldsymbol{x}) - g_\star(\boldsymbol{x})\|_2^2$ decreases. However, depending on the application, the evaluation of the function f at a large number of points might be very expensive or create an imbalance in the approximation properties of the interpolant $g_\star$ with respect to X. Thus, our goal is to find an efficient set of points $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$, i.e., the fewest number of points such that the approximation error is still reasonably small.

For this purpose, we introduce adapted versions of the so-called greedy reconstruction (GR) algorithms (compare [6–10]). These methods compute the inputs $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$ in an offline phase, i.e., without having access to the outputs of f (namely the data $\{f(\widetilde{\boldsymbol{x}}_j)\}_{j=1}^K$), but they require a set (or basis) $\mathcal{B} := \{g_1, \dots, g_K\} \subset \mathcal{G}$.

In order to handle different function spaces $\mathcal{G}$, we assume that the set $\mathcal{B}$ can be parameterized, i.e., there exists $\widetilde{K} \leq K$, a closed and convex subspace $\mathcal{A} \subset \mathbb{R}^{\widetilde{K}}$, and an isomorphism $\widetilde{g} : \mathcal{A} \to \mathcal{G}, \boldsymbol{\alpha} \mapsto \widetilde{g}(\boldsymbol{\alpha}) := g_{\boldsymbol{\alpha}}$. In this way, we can identify $\mathcal{B} = \{g_1, \dots, g_K\}$ with a set $\{\widetilde{\boldsymbol{\alpha}}_1, \dots, \widetilde{\boldsymbol{\alpha}}_K\} \subset \mathcal{A}$ such that $g_k := g_{\widetilde{\boldsymbol{\alpha}}^k}$ for $k = 1, \dots, K$. A simple example for such a parameterization is the case where $\mathcal{B}$ is a linear basis of $\mathcal{G}$. Then we can choose $\widetilde{K} = K$, $\mathcal{A} = \mathbb{R}^K$ and $\widetilde{\boldsymbol{\alpha}}^k = \boldsymbol{e}_k$ the canonical vectors in $\mathbb{R}^K$. For a general $\boldsymbol{\alpha} \in \mathbb{R}^K$ we obtain $g_{\boldsymbol{\alpha}} = \sum_{k=1}^K \alpha_k g_k$.

Now, we can formulate the standard version of GR as described in [7–10]. GR applied to the setting of problem (1) is stated in Algorithm 1. GR builds the set $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$ recursively. Assume that after k iterations a set $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$ has been computed. The new point $\widetilde{\boldsymbol{x}}_{k+1}$ is obtained in two substeps. First, one solves the interpolation problem (3), which attempts to find a parameterization $\boldsymbol{\alpha}^k$ such that the function $g_{\boldsymbol{\alpha}^k}(\widetilde{\boldsymbol{x}}_j)$ fits $g_{k+1}(\widetilde{\boldsymbol{x}}_j)$ in all previously computed points $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^k$. Second, $\widetilde{\boldsymbol{x}}_{k+1}$ is computed as the solution to (4) by maximizing the difference between $g_{\boldsymbol{\alpha}^k}(\widetilde{\boldsymbol{x}}_{k+1})$ and $g_{k+1}(\widetilde{\boldsymbol{x}}_{k+1})$.

Algorithm 2 OGR for function approximation

Require: A set $\mathcal{B} = \{g_1, \dots, g_K\} \subset \mathcal{G}$ with parameterizations $\mathcal{A} = \{\widetilde{\boldsymbol{\alpha}}^1, \dots, \widetilde{\boldsymbol{\alpha}}^K\} \subset \mathbb{R}^{\widetilde{K}}$ with $K, \widetilde{K} \in \mathbb{N}$, $\widetilde{K} \leq K$ and such that $g_k = g_{\widetilde{\boldsymbol{\alpha}}^k}$, $k = 1, \dots, K$, and a tolerance $\text{tol} > 0$.

1: Find $\widetilde{\boldsymbol{x}}_1$ and ℓ_1 that solve

$$\max_{\ell \in \{1,\dots,K\}} \max_{\boldsymbol{x} \in X} \|g_\ell(\boldsymbol{x})\|_2^2. \tag{5}$$

2: Swap g_1 and g_{ℓ_1} in $\mathcal{B}$, $\widetilde{\boldsymbol{\alpha}}^1$ and $\widetilde{\boldsymbol{\alpha}}^{\ell_1}$ in $\mathcal{A}$, and set $f_{split} = \|g_1(\widetilde{\boldsymbol{x}}_1)\|_2^2$ and $k = 1$.

3: **while** $k \leq K - 1$ and $f_{split} \geq \text{tol}$ **do**

4: **for** $\ell = k + 1, \dots, K$ **do**

5: Fitting step: Find $\boldsymbol{\alpha}^\ell$ that solves

$$\min_{\boldsymbol{\alpha} \in \text{span}(\widetilde{\boldsymbol{\alpha}}^i)_{i=1}^k} \sum_{j=1}^{k} \|g_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{x}}_j) - g_\ell(\widetilde{\boldsymbol{x}}_j)\|_2^2. \tag{6}$$

6: **end for**

7: Splitting step: Find $\widetilde{\boldsymbol{x}}_{k+1}$ and ℓ_{k+1} that solve

$$\max_{\ell \in \{k+1,\dots,K\}} \max_{\boldsymbol{x} \in X} \|g_{\boldsymbol{\alpha}^\ell}(\boldsymbol{x}) - g_\ell(\boldsymbol{x})\|_2^2. \tag{7}$$

8: Swap g_{k+1} and $g_{\ell_{k+1}}$ in $\mathcal{B}$, $\widetilde{\boldsymbol{\alpha}}^{k+1}$ and $\widetilde{\boldsymbol{\alpha}}^{\ell_{k+1}}$ in $\mathcal{A}$.

9: Update $f_{split} \leftarrow \|g_{\boldsymbol{\alpha}^k}(\widetilde{\boldsymbol{x}}_{k+1}) - g_{k+1}(\widetilde{\boldsymbol{x}}_{k+1})\|_2^2$ and $k \leftarrow k + 1$.

10: **end while**

Notice that GR performs a sweep over the set $\mathcal{B}$, computing a new input $\widetilde{\boldsymbol{x}}$ for each element in $\mathcal{B}$. Hence, the choice and order of the elements $g_1, \dots, g_K$ can have a large impact on the selection of $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$. Since it is usually not clear a-priori which set $\mathcal{B}$ yields the inputs $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^K$ with the best approximation features, we would like to remove or at least reduce the dependence of the algorithm on the functions $\{g_k\}_{k=1}^K$. For this purpose, in Algorithm 2 we introduce a strategy that has also been discussed in [7–10], called optimized GR algorithm (OGR). The main novelty of OGR over the standard GR algorithm is that it considers all remaining elements in the set $\mathcal{B}$ simultaneously at each iteration. In the spirit of greedy procedures, the new splitting step (7) selects simultaneously the next input and element of $\mathcal{B}$ as the ones that return the largest overall cost function value. Additionally, the second condition for the while loop in line 3 allows stopping OGR when a specific error tolerance is reached. This can be particularly useful if the error between f and any function in $\mathcal{G}$ is a priori expected to be of a certain order, meaning that further decreasing the "internal" error within $\mathcal{G}$ does not decrease the overall approximation error.

Both GR and OGR are data-free methods, in the sense that they do not require evaluations of the function f (outputs). However, there are many applications, especially in the field of machine learning, where output data are already available. A common issue in these data-driven settings is that the data is imbalanced, which can lead to biased models (see, e.g., [5, Section 5.3.1]) and thereby a lack of robustness. With the goal of selecting optimal data points that represent well the whole data set and avoid biased models, we introduce a data-driven version of GR (DGR) in Algorithm 3. Notice that the termination criterion, that is $\|g_{\boldsymbol{\alpha}^k}(\widetilde{\boldsymbol{x}}_{k+1}) - \widetilde{\boldsymbol{y}}_{k+1}\|_2^2 < \text{tol}$,

Algorithm 3 DGR for function approximation

Require: A set of data points $\mathcal{Z} := \{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^J \subset X \times \mathbb{R}^M$ with $\boldsymbol{y}_j := f(\boldsymbol{x}_j)$ and $J \in \mathbb{N}$, a set of functions $\mathcal{B} = \{g_1, \dots, g_K\} \subset \mathcal{G}$ and a tolerance $\text{tol} > 0$.

1: Find $(\widetilde{\boldsymbol{x}}_1, \boldsymbol{y}_1)$ that solves

$$\max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} \|\boldsymbol{y}\|_2^2. \tag{8}$$

2: Update $\mathcal{Z} \leftarrow \mathcal{Z} \setminus (\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{y}}_1)$ and set $f_{split} = \|\widetilde{\boldsymbol{y}}_1\|_2^2$, $k = 1$.

3: **while** $k \leq J - 1$ and $f_{split} \geq \text{tol}$ **do**

4: Fitting step: Find an $\boldsymbol{\alpha}^k$ that solves

$$\min_{\boldsymbol{\alpha}\in\text{span}\{\widetilde{\boldsymbol{\alpha}}_i\}_{i=1}^k} \sum_{j=1}^{k} \|g_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{x}}_j) - \widetilde{\boldsymbol{y}}_j\|_2^2. \tag{9}$$

5: Splitting step: Find $(\widetilde{\boldsymbol{x}}_{k+1}, \widetilde{\boldsymbol{y}}_{k+1})$ that solves

$$\max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} \|g_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2. \tag{10}$$

6: Update $\mathcal{Z} \leftarrow \mathcal{Z} \setminus (\widetilde{\boldsymbol{x}}_{k+1}, \widetilde{\boldsymbol{y}}_{k+1})$, $f_{split} \leftarrow \|g_{\boldsymbol{\alpha}^k}(\widetilde{\boldsymbol{x}}_{k+1}) - \widetilde{\boldsymbol{y}}_{k+1}\|_2^2$ and $k \leftarrow k + 1$.

7: **end while**

is similar to that of OGR where the evaluation $g_{k+1}(\widetilde{\boldsymbol{x}}_{k+1})$ is replaced by the data $\widetilde{\boldsymbol{y}}_{k+1}$. This ensures that DGR terminates when the function $g_{\boldsymbol{\alpha}^k}$ is a sufficiently good approximation of the full data set $\{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^J$. In general, it is also possible to modify DGR by using other features of OGR to DGR like, for example, the automatic selection of the next element g_{k+1} from $\mathcal{B}$. However, we omit this discussion here, since the resulting method is very similar to the NOGR strategy introduced in Section 5.1.

4 GR algorithms for polynomial approximation

In this section, we apply our methods to the well-known case of function approximation by polynomials, i.e., $M = 1$ and $\mathcal{G} = \mathcal{P}_n(X)$, and study the corresponding behavior. For a set of linearly independent polynomials $\mathcal{B} = \{p_1, \dots, p_K\} \subset \mathcal{P}_n$, we consider the simple linear parameterization $p_{\boldsymbol{\alpha}} = \sum_{k=1}^K \alpha_k p_k$ for $\boldsymbol{\alpha} \in \mathbb{R}^K$. This implies that, at the k-th GR iteration, the fitting step is interpolating the new polynomial p_{k+1} by a linear combination of the polynomials $p_1, \dots, p_k$. GR for polynomial interpolation is detailed in Algorithm 4. Now, we analyze the behavior of this algorithm in sections 4.1 and 4.2. Finally, in section 4.3 we discuss practical implementation aspects.

4.1 Analysis of GR algorithms

In this section, we investigate the points selected by Algorithm 4 for one-dimensional and multivariate polynomial interpolation. First, we consider the standard case of one-dimensional polynomials in Section 4.1.1, since it is well understood and the notation is simple. This allows us to formulate the main arguments of our analysis in a comprehensible way. Afterwards, we build the analysis for multivariate polynomials in Section 4.1.2 on top of the one-dimensional foundation.

Algorithm 4 GR for polynomial interpolation

Require: A set of linearly independent polynomials $\mathcal{B} = \{p_1, \dots, p_K\} \subset \mathcal{P}_n$.

1: Set $k = 1$ and find $\widetilde{\boldsymbol{x}}_1$ that solves

$$\max_{\boldsymbol{x} \in X} |p_1(\boldsymbol{x})|^2. \tag{11}$$

2: **while** $k \leq K - 1$ **do**

3: Fitting step: Find an $\boldsymbol{\alpha}^k$ that solves

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^k} \sum_{j=1}^{k} |p_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{x}}_j) - p_{k+1}(\widetilde{\boldsymbol{x}}_j)|^2, \tag{12}$$

where $p_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \sum_{i=1}^{k} \alpha_i p_i(\boldsymbol{x})$.

4: Splitting step: Find an $\widetilde{\boldsymbol{x}}_{k+1}$ that solves

$$\max_{\boldsymbol{x} \in X} |p_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x})|^2. \tag{13}$$

5: Update $k \leftarrow k + 1$.

6: **end while**

4.1.1 One-dimensional polynomial interpolation

We begin with the one-dimensional setting $N = 1$ with $X \subset \mathbb{R}$ a closed interval. We also consider the standard assumption that the number of selected points matches the degree of the polynomials in $\mathcal{B}$, i.e., $K = n + 1$.

Recall (e.g., from [2, Theorem 8.2]) that for any set of points $\{\widetilde{x}_j\}_{j=1}^{n+1}$ and the polynomial $p_\star \in \mathcal{P}_n$ interpolating a $(n+1)$-times continuously differentiable function f at these points, there exists for each $x \in X$ a $\xi \in X$ such that

$$|f(x) - p_\star(x)| = \frac{\|f^{(n+1)}(\xi)\|_\infty}{(n+1)!} \prod_{j=1}^{n+1} |x - \widetilde{x}_j|. \tag{14}$$

Thus, to minimize the error $|f(x) - p_\star(x)|$, the points $\{\widetilde{x}_j\}_{j=1}^{n+1}$ should be chosen such that they minimize $\max_{x \in X} \prod_{j=1}^{n+1} |x - \widetilde{x}_j|$. One popular set of points that can be derived directly from this goal, are the so-called Leja points. This class of interpolation points was first introduced by Edrei [12] and later studied by Leja [11]. Starting from any $\widetilde{x}_1 \in X$ the corresponding Leja sequence $(x_j)_{j=1}^{n+1}$ is generated iteratively by the update

$$\widetilde{x}_{k+1} = \arg\max_{x \in X} \prod_{j=1}^{k} |x - \widetilde{x}_j|, \qquad k = 1, \dots, n. \tag{15}$$

The update formula (15) has two main advantages. First, the update formula allows one to incorporate any preexisting set of points $\{\widetilde{x}_j\}_{j=1}^{k}$. This property is of particular interest in applications, where some preliminary data is already available. Second, the

computation of the new point $\widetilde{x}_{k+1}$ allows one to evaluate the term $\max_{x\in X}\prod_{j=1}^{k}|x-\widetilde{x}_j| = \prod_{j=1}^{k}|\widetilde{x}_{k+1}-\widetilde{x}_j|$, which corresponds to the a-priori interpolation error given in (14).

As main result of this section, we show that GR computes exactly a sequence of Leja points if $\mathcal{B}$ is a polynomial basis of increasing order.

Theorem 1 (GR computes the Leja points)**.** *Let $\mathcal{B} = \{p_1, \dots, p_{n+1}\} \subset \mathcal{P}_n$ be such that $\operatorname{span}\{p_1, \dots, p_k\} = \mathcal{P}_{k-1}$ for all $k = 1, \dots, n$. Then the points $\{\widetilde{x}_j\}_{j=1}^{n+1}$ computed Algorithm 4 form a Leja sequence.*

Proof. We begin by constructing the coefficients $\boldsymbol{\alpha}^k \in \mathbb{R}^k$ that solve the fitting step problem (12) at iteration k. Let us define $\widetilde{p}_k(x) := -a_{k+1}\prod_{j=1}^{k}(x-\widetilde{x}_j)$, where a_{k+1} is the leading coefficient of $p_{k+1} \in \mathcal{P}_k$. Then $\widetilde{p}_k + p_{k+1}$ is a polynomial in $\mathcal{P}_{k-1}$. Since $\{p_1, \dots, p_K\}$ form a basis of $\mathcal{P}_{k-1}$, there exists $\boldsymbol{\alpha}^k \in \mathbb{R}^k$ such that $p_{\boldsymbol{\alpha}^k} = \widetilde{p}_k + p_{k+1}$. Thus $p_{\boldsymbol{\alpha}^k}(\widetilde{x}_j) - p_{k+1}(\widetilde{x}_j) = \widetilde{p}_k(\widetilde{x}_j) = 0$ for all $j = 1, \dots, k$, which implies that $\boldsymbol{\alpha}^k = \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^k}\sum_{j=1}^{k}|p_{\boldsymbol{\alpha}}(\widetilde{x}_j) - p_{k+1}(\widetilde{x}_j)|$. The uniqueness follows by the fact that the points $\widetilde{x}_1, \dots, \widetilde{x}_k$ are distinct (which can be shown by a simple iterative argument), meaning that the polynomial $p_{\boldsymbol{\alpha}^k}$ and therefore also $\boldsymbol{\alpha}^k$ are unique.

Now, using the polynomial $p_{\boldsymbol{\alpha}^k}$ constructed above and solving (12), we obtain

$$|p_{\boldsymbol{\alpha}^k}(x) - p_{k+1}(x)|^2 = |\widetilde{p}_k(x)|^2 = |a_{k+1}|^2 \prod_{j=1}^{k}|x-\widetilde{x}_j|^2.$$

Since a_{k+1} does not depend on x, the splitting-step problem (13) can be written as $\max_{x\in X}\prod_{j=1}^{k}|x-\widetilde{x}_j|^2$, which is equivalent to the Leja update (15). Thus, the splitting step of Algorithm 4 at iteration k computes exactly the point $\widetilde{x}_{k+1}$ in the Leja sequence. □

Notice that $p_1 \in \mathcal{P}_0$ implies that $p_1 \equiv c$ for some $c \in \mathbb{R}$. Thus, any $x \in X$ is a solution to the initialization problem (11). Together with the result from Theorem 1, this implies that one may choose any initial point $\widetilde{x}_1 \in X$ and GR will compute the corresponding Leja sequence $\{\widetilde{x}_j\}_{j=1}^{n+1}$.

4.1.2 Multivariate polynomial interpolation

Let us turn to the multivariate setting, i.e., $N > 1$ and $X = X_1 \times \cdots \times X_N$ with X_i a closed and bounded subinterval of $\mathbb{R}$ for $i = 1, \dots, N$. As in the one-dimensional case, we denote by $\mathcal{B}$ a basis of $\mathcal{P}_n = \mathcal{P}_n(X)$. Notice that $\dim(\mathcal{P}_n) = \binom{N+n}{N}$ (see, e.g., [22, Theorem 2.5]), which implies that GR will compute $K = \binom{N+n}{N}$ points.

To relate the points selected by GR to tensor products of Leja points, we introduce some additional notation. For an initial point $\widetilde{\boldsymbol{x}}_1 \in X$, we denote by $\{y_j^i\}_{j=1}^{n+1} \subset X_i$, y_j^i the Leja sequence initialized by $\widetilde{x}_{1,i}$ for $i = 1, \dots, N$, i.e., $y_1^i = \widetilde{x}_{1,i}$ and

$$y_j^i = \arg\max_{x\in X_i}\prod_{\ell=1}^{j-1}|x-y_\ell^i|, \quad i = 1, \dots, N, \quad j = 2, \dots, n+1. \tag{16}$$

Now, we define the m-first tensor-product Leja points for $m \in \mathbb{N}$ as

$$\mathcal{Y}_m := \Big\{ [y^1_{j_1}, \dots, y^N_{j_N}]^\top \mid j_1, \dots, j_N \in \mathbb{N}^+ \text{ s.t. } \sum_{i=1}^{N} j_i \leq N + m \Big\}, \tag{17}$$

with cardinality $\#\mathcal{Y}_m = \binom{N+m}{N}$. Let us give an example of $\mathcal{Y}_m$.

Example 1 (on the set $\mathcal{Y}_m$)**.** *Let $N = n = 2$, $X = [-1,1]^2$ and $\widetilde{\boldsymbol{x}}_1 = [-1,-1]^\top$. Then $y^1_1 = y^2_1 = -1$, $y^1_2 = y^2_2 = \arg\max_{x \in [-1,1]} |x - (-1)| = 1$ and*

$$y^1_3 = y^2_3 = \arg\max_{x \in [-1,1]} |x+1||x-1| = 0.$$

Thus, the two Leja sequences $\{y^1_j\}_{j=1}^3$ and $\{y^2_j\}_{j=1}^3$ are given by $\{-1, 1, 0\}$. Since

$$\mathcal{Y}_m = \mathcal{Y}_{m-1} \cup \{ [y^1_{j_1}, y^2_{j_2}]^\top \mid j_1, j_2 \in \mathbb{N}^+ \text{ s.t. } j_1 + j_2 = 2 + m \}$$

for $m \geq 1$, we obtain

$$\begin{aligned}
\mathcal{Y}_0 &= \{[y^1_1, y^2_1]^\top\}, = \{[-1,-1]^\top\} \\
\mathcal{Y}_1 &= \mathcal{Y}_0 \cup \{[y^1_1, y^2_2]^\top, [y^1_2, y^2_1]^\top\}, = \mathcal{Y}_0 \cup \{[-1,1]^\top, [1,-1]^\top\} \\
\mathcal{Y}_2 &= \mathcal{Y}_1 \cup \{[y^1_1, y^2_3]^\top, [y^1_2, y^2_2]^\top, [y^1_3, y^2_1]^\top\} = \mathcal{Y}_1 \cup \{[-1,0]^\top, [1,1]^\top, [0,-1]^\top\}. \\
\mathcal{Y}_3 &= \mathcal{Y}_2 \cup \{[y^1_2, y^2_3]^\top, [y^1_3, y^2_2]^\top\} = \mathcal{Y}_2 \cup \{[1,0]^\top, [0,1]^\top\} \\
\mathcal{Y}_4 &= \mathcal{Y}_3 \cup \{[y^1_3, y^2_3]^\top\} = \mathcal{Y}_3 \cup \{[0,0]^\top\}.
\end{aligned}$$

These sets are represented in Figure 1. Notice that for $m \geq 5$ we have $\mathcal{Y}_m = \mathcal{Y}_4$.

Remark 1 (on the set $\mathcal{Y}_n$ and the full tensor product of Leja points)**.** *Notice that the set $\mathcal{Y}_n = \mathcal{Y}_2$ in Example 1 still contains the full Leja tensor product $\{y^1_1, y^1_2\} \otimes \{y^2_1, y^2_2\} = \{-1,1\} \otimes \{-1,1\}$. However, this is not the case for general $n, N \in \mathbb{N}$ since a full Leja tensor product $\{y^1_1, \dots, y^1_m\} \otimes \cdots \otimes \{y^N_1, \dots, y^N_m\}$ of order $m \leq n$ contains all points $[y^1_{j_1}, \dots, y^N_{j_N}]^\top$ with $j_1, \dots, j_N \in \{1, \dots, m\}$. Thus, in order for the set $\mathcal{Y}_n$ to contain this point, we require n to satisfy $\sum_{i=1}^N m \leq N + n$, namely that $Nm \leq N + n$.*

The main result of this section (Theorem 2) is that the points computed by GR coincide with $\mathcal{Y}_n$. To prove it, two assumptions are needed. To clarify their meanings, we sketch the idea of the proof of Theorem 2 in what follows.

The main part of the result is proven by induction on the degree m of the polynomials in the basis $\mathcal{B} = \{p_1, \dots, p_K\}$. For the induction step, we follow the same idea of the proof for Theorem 1 and thus require that the basis elements are ordered according to their polynomial order and the appearing monomials. For this purpose, we introduce the following assumption on the order of $\mathcal{B}$.

Assumption 1. *Let $m \in \{0, \dots, n\}$, $k = \binom{N+m}{N}$ and $K_m = \binom{N+m+1}{N}$. Then the first k elements $p_1, \dots, p_k$ in $\mathcal{B}$ form a basis of $\mathcal{P}_m$. Additionally, for all $\ell \in \{k+1, \dots, K_m\}$ the polynomial $p_\ell \in \mathcal{B}$ introduces exactly one new monomial with respect to all monomials appearing in the polynomials $p_1, \dots, p_k$. In other words, there exists*

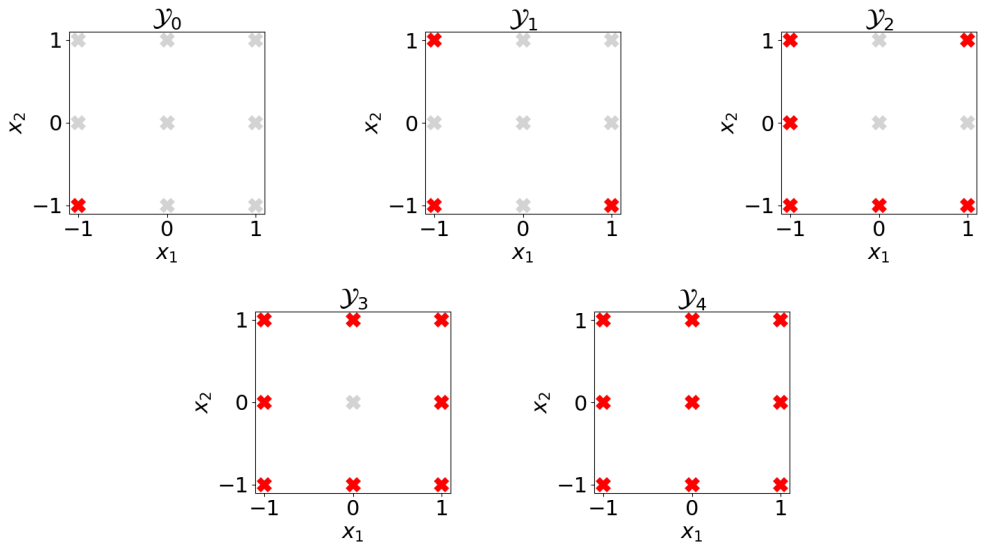

Fig. 1: All tensor-product points (gray crosses) corresponding to the two Leja sequences $\{y_j^1\}_{j=1}^3 = \{y_j^2\}_{j=1}^3 = \{-1, 1, 0\}$ from Example 1, and points $\boldsymbol{x} = [x_1, x_2]^\top \in [-1, 1]^2$ (red crosses) contained in the sets $\mathcal{Y}_0$ (top left), $\mathcal{Y}_1$ (top middle), $\mathcal{Y}_2$ (top right), $\mathcal{Y}_3$ (bottom left) and $\mathcal{Y}_4$ (bottom right) as defined in (17).

a (unique) polynomial $p \in \operatorname{span}(p_1, \ldots, p_k)$ *such that* $p_\ell = a_{d_1^\ell, \ldots, d_N^\ell} \boldsymbol{x}^{(d_1^\ell, \ldots, d_N^\ell)} + p$, *where* $a_{d_1^\ell, \ldots, d_N^\ell} \in \mathbb{R} \setminus \{0\}$ *and* $(d_1^\ell, \ldots, d_N^\ell)$ *denotes the componentwise degree of the new monomial* $\boldsymbol{x}^{(d_1^\ell, \ldots, d_N^\ell)}$ *introduced by* p_ℓ.

Let us clarify Assumption 1 by the following example.

Example 2. *Let* $N = n = 2$ *and consider a basis* $\mathcal{B} = \{p_1, \ldots, p_K\}$ *with* $K = \binom{N+n}{N} = 6$ *such that Assumption 1 holds. Then* p_1 *must be the constant polynomial, i.e.,* $p_1 \equiv a$ *for some* $a \in \mathbb{R}$. *Now, there are two possible choices for* p_2*:*

$$p_2(\boldsymbol{x}) = a_1 x_1 + a_0 \qquad \text{or} \qquad p_2(\boldsymbol{x}) = a_1 x_2 + a_0,$$

for $a_1 \in \mathbb{R} \setminus \{0\}$, $a_0 \in \mathbb{R}$. *Let us assume that* $p_2(\boldsymbol{x}) = a_1 x_1 + a_0$. *Then*

$$p_3(\boldsymbol{x}) = b_1 x_2 + b_0,$$

for $b_1 \in \mathbb{R} \setminus \{0\}$, $b_0 \in \mathbb{R}$. *Now, we have three choices for* p_4*:*

$$p_4(\boldsymbol{x}) = \begin{cases} a_3 x_1^2 + a_2 x_2 + a_1 x_1 + a_0, \\ a_3 x_2^2 + a_2 x_2 + a_1 x_1 + a_0, \\ a_3 x_1 x_2 + a_2 x_2 + a_1 x_1 + a_0, \end{cases}$$

for $a_3 \in \mathbb{R} \setminus \{0\}$, $a_2, a_1, a_0 \in \mathbb{R}$. Depending on the choice of p_4, the polynomials p_5 and p_6 will be of the remaining two forms and thereby introduce the other monomials of degree 2.

Assumption 1 allows us to uniquely identify the basis elements in $\mathcal{B}$ by the componentwise degree of the monomials introduced by the basis elements. For example, if we choose $p_4 = a_3x_1x_2 + a_2x_2 + a_1x_1 + a_0$ in Example 2, then we can identify p_4 by the componentwise degree $(1,1)$ that corresponds to the monomial $a_3x_1x_2$. Let us conclude the discussion on Assumption 1 with the following remark, relating the space of polynomials $\mathcal{P}_{m+1} \setminus \mathcal{P}_m$ with the set of Leja points $\mathcal{Y}_{m+1} \setminus \mathcal{Y}_m$, for $m = 0, \dots, n$.

Remark 2 (on the sets $\mathcal{P}_m$ and $\mathcal{Y}_m$)**.** *Let $m \in \{0, \dots, n\}$, $k = \binom{N+m}{N}$ and $K_m = \binom{N+m+1}{N}$. Then Assumption 1 implies that the polynomials $\{p_\ell\}_{\ell=k+1}^{K_m}$ form a basis of $\mathcal{P}_{m+1} \setminus \mathcal{P}_m$. Thus, the componentwise degrees $(d_1^\ell, \dots, d_N^\ell)$ of the monomials $\boldsymbol{x}^{(d_1^\ell, \dots, d_N^\ell)}$ introduced by the polynomials p_ℓ, for $\ell = k+1, \dots, K_m$, coincide with all possible combinations $(d_1, \dots, d_N) \in \mathbb{N}^N$ that satisfy $\sum_{i=1}^N d_i = m+1$. Moreover, we can write*

$$\begin{aligned}\mathcal{Y}_{m+1} \setminus \mathcal{Y}_m &= \left\{ [y_{j_1}^1, \dots, y_{j_N}^N]^\top \mid j_1, \dots, j_N \in \mathbb{N}^+ \text{ s.t. } \sum_{i=1}^N j_i = N + m + 1 \right\} \\ &= \left\{ [y_{d_1+1}^1, \dots, y_{d_N+1}^N]^\top \mid (d_1, \dots, d_N) \in \mathbb{N}^N \text{ s.t. } \sum_{i=1}^N d_i = m + 1 \right\}.\end{aligned}$$

Hence, there exists a one-to-one relation between the componentwise degrees of the monomials introduced by the polynomials $\{p_\ell\}_{\ell=k+1}^{K_m}$ and the Leja points $\mathcal{Y}_{m+1} \setminus \mathcal{Y}_m$.

The next step in the proof of our main result is to show that the splitting-step problem (13) is equivalent to a multivariate version of the Leja update (16). However, notice that maximization problems involving multivariate polynomials in general do not have a unique solution. Consider, for example, the maximization problem

$$\max_{\boldsymbol{x} \in X} |p(\boldsymbol{x})|^2, \; p(\boldsymbol{x}) = \left(\prod_{j=1}^{1} (x_1 - y_j^1) \right) \left(\prod_{j=1}^{0} (x_2 - y_j^2) \right) \cdots \left(\prod_{j=1}^{0} (x_N - y_j^N) \right) = x_1 - y_j^1. \tag{18}$$

Then any $\boldsymbol{x} = [x_1, \dots, x_N]^\top \in X$ with $x_1 \in \arg\max_{x \in X_1} |x_1 - y_j^1|^2$ is a solution to (18). Notice that x_1 does not has to be unique. Thus, we introduce the following assumption that sets all components to the next entry of the corresponding Leja sequence.

Assumption 2. *At iteration $k \in \{1, \dots, K-1\}$ of Algorithm 4, let $(d_1^{k+1}, \dots, d_N^{k+1}) \in \mathbb{N}^N$ be the componentwise degree of the unique monomial that $p_{k+1} \in \mathcal{B}$ introduces, according to the second part of Assumption 1. If the splitting-step problem (13) is given by $\max_{\boldsymbol{x} \in X} |p_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x})|^2$ with*

$$p_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x}) = a \left(\prod_{j=1}^{d_1^{k+1}} (x_1 - y_j^1) \right) \cdots \left(\prod_{j=1}^{d_N^{k+1}} (x_N - y_j^N) \right), \tag{19}$$

for some $a \in \mathbb{R}$, then Algorithm 4 selects the point $\widetilde{\boldsymbol{x}}_{k+1} = \left[y^1_{d_1^{k+1}+1}, \dots, y^N_{d_N^{k+1}+1}\right]^\top$.

Notice that, if the polynomial $p_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x})$ is given as in (19), then

$$\max_{\boldsymbol{x}\in X} |p_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x})|^2 = a\left(\max_{x_1\in X_1} \prod_{j=1}^{d_1^{k+1}} |x_1 - y_j^1|^2\right) \cdots \left(\max_{x_N\in X_N} \prod_{j=1}^{d_N^{k+1}} |x_N - y_j^N|^2\right). \tag{20}$$

Thus, by the Leja update (16), $\left[y^1_{d_1^{k+1}+1}, \dots, y^N_{d_N^{k+1}+1}\right]$ is always a solution to (20) and hence to the splitting-step problem (13). We can now state our main result.

Theorem 2 (multivariate GR computes tensor product Leja points)**.** *Let $n, N \in \mathbb{N}$, $K = \binom{N+n}{N}$, $\mathcal{B} = \{p_k\}_{k=1}^K$ a basis of $\mathcal{P}_n$ and $\widetilde{\boldsymbol{x}}_1 \in X$ the solution to (11). If Assumptions 1 and 2 hold, then the points computed by Algorithm 4 coincide with $\mathcal{Y}_n$.*

Proof. We prove the result by induction on the degree m of the polynomials in $\mathcal{B}$. The case $m = 0$ is trivial since $\mathcal{Y}_0 = \{[y_1^1, \dots, y_1^N]^\top\} = \{\widetilde{\boldsymbol{x}}_1\}$. Let us now assume that for some $m \in \{1, \dots, n\}$ and $k = \binom{N+m}{N}$ the set $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^k$ coincides with $\mathcal{Y}_m$. Then the result follows if we can show that the set $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^{K_m}$ for $K_m = \binom{N+m+1}{N}$ coincides with $\mathcal{Y}_{m+1}$. To do so, recalling Remark 2, it is sufficient to show that Algorithm 4 computes exactly the point $[y^1_{d_1^\ell+1}, \dots, y^N_{d_N^\ell+1}]^\top$ at iteration $\ell \in \{k+1, \dots, K_m\}$. We prove this result by a second induction argument on the index ℓ.

Let us begin with the base case $\ell = k+1$. By Assumption 1, we have

$$p_{k+1} = a_{d_1^{k+1},\dots,d_N^{k+1}} \boldsymbol{x}^{(d_1^{k+1},\dots,d_N^{k+1})} + p,$$

for some $p \in \operatorname{span}\{p_1, \dots, p_K\}$. Now, we define the polynomial

$$\widetilde{p}(\boldsymbol{x}) := -a_{d_1^{k+1},\dots,d_N^{k+1}} \left(\prod_{j=1}^{d_1^{k+1}} (x_1 - y_j^1)\right) \cdots \left(\prod_{j=1}^{d_N^{k+1}} (x_N - y_j^N)\right) \in \mathcal{P}_{m+1}, \tag{21}$$

where y_j^i are the Leja points defined in (16). By the definition of the polynomial degree given in Section 2, we can write $\widetilde{p}(\boldsymbol{x}) = \sum_{|\boldsymbol{\beta}|=0}^m a_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}} - p_{k+1}(\boldsymbol{x})$, for some $a_{\boldsymbol{\beta}} \in \mathbb{R}$. Thus, since the polynomials $\{p_1, \dots, p_K\}$ form a basis of $\mathcal{P}_m$ (by Assumption 1), there exists a unique $\boldsymbol{\alpha}^k \in \mathbb{R}^k$ such that

$$\widetilde{p}(\boldsymbol{x}) = p_{\boldsymbol{\alpha}^k}(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x}) = \sum_{i=1}^k \alpha_i^k p_i(\boldsymbol{x}) - p_{k+1}(\boldsymbol{x}), \qquad \text{for all } \boldsymbol{x} \in X.$$

Recalling (17), we have for any $\boldsymbol{y} \in \mathcal{Y}_m$ that $\boldsymbol{y} = [y_{j_1}^1, \dots, y_{j_N}^N]^\top$ for some indices $j_1, \dots, j_N \in \mathbb{N}^+$ with $\sum_{i=1}^N j_i = N + m$. Since $\sum_{i=1}^N d_i^{k+1} = m+1$, there exists at least one j_i such that $1 \le j_i \le d_i^{k+1}$.[1] Together with (21), this implies that $\widetilde{p}(\boldsymbol{y}) = 0$ for all

[1] This follows by a contradiction argument: assume that all $j_i > d_i^{k+1}$ for all $i \in \{1, \dots, N\}$, then $\sum_{i=1}^N j_i \ge \sum_i^N (d_i^{k+1} + 1) = N + m + 1$, contradicting $\sum_{i=1}^N j_i = N + m$.

$\boldsymbol{y} \in \mathcal{Y}_m$. Since we assume that the set $(\widetilde{\boldsymbol{x}}_j)_{j=1}^k$ coincides with $\mathcal{Y}_m$, we obtain $\widetilde{p}(\widetilde{\boldsymbol{x}}_j) = 0$ for all $j \in \{1, \dots, k\}$. Thus, $\boldsymbol{\alpha}^k$ is the unique solution to the fitting-step problem (12), and the splitting-step problem (13) can be written as $\max_{\boldsymbol{x} \in X} |\widetilde{p}(\boldsymbol{x})|^2$. By Assumption 2, Algorithm 4 will now choose exactly $\widetilde{\boldsymbol{x}}_{k+1} = \left[y^1_{d_1^{k+1}+1}, \dots, y^N_{d_N^{k+1}+1}\right]^\top$.

Next, consider the induction step $\ell \to \ell + 1$. Let us denote by $(\widetilde{\boldsymbol{x}}_j)_{j=k+1}^{\ell}$ the set of points computed by Algorithm 4 from the base case until iteration ℓ. By the induction hypothesis, we have $(\widetilde{\boldsymbol{x}}_j)_{j=k+1}^{\ell} \in \mathcal{Y}_{m+1}$. Now, the induction step $\ell \to \ell + 1$ follows analogously to the base case above. We only have to show that the new polynomial

$$\widetilde{p}(\boldsymbol{x}) := -a_{d_1^{\ell+1}, \dots, d_N^{\ell+1}} \left(\prod_{j=1}^{d_1^{\ell+1}} (x_1 - y_j^1) \right) \cdots \left(\prod_{j=1}^{d_N^{\ell+1}} (x_N - y_j^N) \right) \in \mathcal{P}_{m+1} \tag{22}$$

satisfies $\widetilde{p}(\widetilde{\boldsymbol{x}}_j) = 0$ also for all $j \in \{k+1, \dots, \ell\}$. This can be shown by the following contradiction. Assume that $\widetilde{p}(\widetilde{\boldsymbol{x}}_j) \neq 0$ for some $j \in \{k+1, \dots, \ell\}$. By the assumption of the induction step, we have $\widetilde{\boldsymbol{x}}_j \in \mathcal{Y}_{m+1} \setminus \mathcal{Y}_m$ and $\widetilde{\boldsymbol{x}}_j = [y^1_{d_1^j+1}, \dots, y^N_{d_N^j+1}]^\top$ with $\sum_{i=1}^N d_i^j = m+1$. On the other hand, since $p_{\ell+1} \in \mathcal{P}_{m+1} \setminus \mathcal{P}_m$ we also have $\sum_{i=1}^N d_i^{\ell+1} = m+1$. Thus, $\widetilde{p}(\widetilde{\boldsymbol{x}}_j) \neq 0$ together with (22) imply that $d_i^j + 1 > d_i^{\ell+1}$ for all $i \in \{1, \dots, N\}$. Since $\sum_{i=1}^N d_i^j = \sum_{i=1}^N d_i^{\ell+1}$, we obtain $d_i^j = d_i^{\ell+1}$ for all $i \in \{1, \dots, N\}$.[2] Since $j < \ell + 1$, this is a contradiction to the second part of Assumption 1. □

Notice that the full grid of tensor-product Leja points of order n is given by

$$\mathcal{Y} := (y_j^1)_{j=1}^{n+1} \times \cdots \times (y_j^N)_{j=1}^{n+1} = \{[y^1_{j_1}, \dots, y^N_{j_N}]^\top \mid j_1, \dots, j_N \in \{1, \dots, n+1\}\}.$$

Clearly, we have $\mathcal{Y}_n \subsetneq \mathcal{Y}$, meaning that GR applied to a basis $\mathcal{B}$ of $\mathcal{P}_n$ only computes a subset of the tensor-product Leja points. However, analogously to Theorem 2, one can also show that if we use a basis $\mathcal{B}$ that spans the same space as the monomials up to componentwise degree n^N, then GR computes the full tensor product $\mathcal{Y}$.

4.2 OGR and equivalence with EIM

As in Section 3, we can formulate an OGR for polynomial interpolation problems. The resulting method is stated in Algorithm 5. This algorithm looks similar to a popular greedy algorithm introduced in [15]: the empirical interpolation method (EIM). While EIM is mostly used in the context of reduced-basis applications for parameter-dependent PDEs, it has also been studied for polynomial interpolation (see, e.g., [17, Section 3]). The version of EIM used for the setting of polynomial interpolation (compare [17, Section 2]), adapted to our notation, is stated in Algorithm 6.

Although the two methods are very similar, OGR has one main advantage over EIM: the parameterization discussed in Section 3. While EIM only considers linear

[2] This follows by a contradiction argument: assume that there exists $i \in \{1, \dots, N\}$ with $d_i^j \neq d_i^{\ell+1}$. Then, since $\sum_{i=1}^N d_i^j = \sum_{i=1}^N d_i^{\ell+1}$, there exists $\widehat{i} \in \{1, \dots, N\}$ with $d_{\widehat{i}}^j < d_{\widehat{i}}^{\ell+1}$. This implies that $d_{\widehat{i}}^j + 1 \leq d_{\widehat{i}}^{\ell+1}$, which contradicts $d_i^j + 1 > d_i^{\ell+1}$ for all $i \in \{1, \dots, N\}$.

Algorithm 5 OGR for polynomial interpolation

Require: A set $\mathcal{B}_{OGR} = \{p_1, \dots, p_K\} \subset \mathcal{P}_n$ for $K, n, N \in \mathbb{N}$ and a tolerance tol > 0.

1: Find $\widetilde{\boldsymbol{x}}_1$ and ℓ_1 that solve

$$\max_{\ell \in \{1,\dots,K\}} \max_{\boldsymbol{x} \in X} |p_\ell(\boldsymbol{x})|^2. \tag{23}$$

2: Swap p_1 and p_{ℓ_1} in $\mathcal{B}_{OGR}$, and set $f_{split} = |p_1(\widetilde{\boldsymbol{x}}_1)|$ and $k = 1$.
3: **while** $k \leq K - 1$ and $f_{split} \geq$ tol **do**
4: **for** $\ell = k + 1, \dots, K$ **do**
5: Fitting step: Find $\boldsymbol{\alpha}^\ell$ that solves

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^k} \sum_{j=1}^{k} \Big| \sum_{i=1}^{k} \alpha_i p_i(\widetilde{\boldsymbol{x}}_j) - p_\ell(\widetilde{\boldsymbol{x}}_j) \Big|^2. \tag{24}$$

6: **end for**
7: Splitting step: Find $\widetilde{\boldsymbol{x}}_{k+1}$ and ℓ_{k+1} that solve

$$\max_{\ell \in \{k+1,\dots,K\}} \max_{\boldsymbol{x} \in X} \Big| \sum_{i=1}^{k} \alpha_i^\ell p_i(\boldsymbol{x}) - p_\ell(\boldsymbol{x}) \Big|^2. \tag{25}$$

8: Swap p_{k+1} and $p_{\ell_{k+1}}$ in $\mathcal{B}_{OGR}$, and set $p_{\boldsymbol{\alpha}^k} = \sum_{i=1}^{k} \alpha_i^{\ell_{k+1}} p_i$.
9: Update $f_{split} \leftarrow |p_{\boldsymbol{\alpha}^k}(\widetilde{\boldsymbol{x}}_{k+1}) - p_{k+1}(\widetilde{\boldsymbol{x}}_{k+1})|$ and $k \leftarrow k + 1$.
10: **end while**

combinations of the elements in $\mathcal{B}_{EIM}$, OGR can handle also nonlinear parameterizations, for example a set $\mathcal{B}_{OGR}$ of neural networks where the parameterizations are given by the weights and biases. We will discuss this example in more detail in Section 5.

Let us now compare OGR (Algorithm 5) and EIM (Algorithm 6) in more detail. The initializations of OGR (line 1 in Algorithm 5) and EIM (lines 1-2 in Algorithm 6) are obviously equivalent, since $\|p\|_{L^\infty(X)} = \max_{\boldsymbol{x} \in X} |p(\boldsymbol{x})|$ for any $p \in \mathcal{P}_n$. A similar analogy also holds for the maximization problems inside the while loops (line 7 in OGR and lines 8-9 in EIM). The difference here is that, while OGR selects elements p_k directly from the set $\mathcal{B}_{OGR}$, EIM generates a new basis of polynomials $\{q_k\}_{k=1}^{K_{max}}$ (see also line 10 in EIM). If we neglect this difference between p_i and q_i for a moment, we notice that the fitting step (line 5 in OGR) solves the strictly convex problem (24), whose optimality system coincides with the linear system (26). On the other hand, if at iteration k the polynomials $\{\widehat{p}_i\}_{i=1}^{k}$ computed by EIM coincide with the set $\{p_i\}_{i=1}^{k}$ selected by OGR, then we also have $\operatorname{span}\{p_i\}_{i=1}^{k} = \operatorname{span}\{q_i\}_{i=1}^{k}$. This is because q_1 is a rescaled version of $\widehat{p}_1$ (see line 3 in EIM) and the q_k for $k \geq 2$ are linear combinations of $\widehat{p}_1, \dots, \widehat{p}_k$ (see line 10 in EIM).

To prove the equivalence of OGR and EIM, we need the following results [17]:

P1 If $K_{max} \leq \binom{N+n}{N}$ then problem (26) is uniquely solvable (see [17, Thm 1]).

P2 For all $k \in \{1, \dots, K_{max}\}$ and all $p \in \operatorname{span}\{q_i\}_{i=1}^{k}$ we have $p - \sum_{i=1}^{k} [\boldsymbol{\alpha}(p)]_i q_i = 0$ (see [17, Lemma 1]).

Theorem 3 (Equivalence of OGR and EIM)**.** *Assume that* $\mathcal{B}_{OGR} = \mathcal{B}_{EIM} =: \mathcal{B}$. *Let* $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^{K_{OGR}}$, $\{p_i\}_{i=1}^{K_{OGR}}$ *and* $\{\widetilde{\boldsymbol{y}}_j\}_{j=1}^{K_{max}}$, $\{q_i\}_{i=1}^{K_{max}}$ *be the points and polynomials selected*

Algorithm 6 EIM for polynomial interpolation

Require: A set of polynomials $\mathcal{B}_{EIM} = \{p_1, \dots, p_K\} \subset \mathcal{P}_n$ for $K, n, N \in \mathbb{N}$, and a maximum number of iterations $K_{max} \leq K$.

1: Compute $\widehat{p}_1 = \underset{p \in \mathcal{B}_{EIM}}{\arg\max} \|p\|_{L^\infty(X)}$.
2: Compute $\widetilde{\boldsymbol{y}}_1 = \underset{\boldsymbol{y} \in X}{\arg\max} |\widehat{p}_1(\boldsymbol{y})|$.
3: Set $q_1 = \frac{\widehat{p}_1}{\widehat{p}_1(\widetilde{\boldsymbol{y}}_1)}$ and $k = 1$.
4: **while** $k \leq K_{max} - 1$ **do**
5: **for** $p \in \mathcal{B}_{EIM}$ **do**
6: Find $\boldsymbol{\alpha}(p) \in \mathbb{R}^k$ that solves the interpolation problem

$$\sum_{i=1}^{k} [\boldsymbol{\alpha}(p)]_i q_i(\widetilde{\boldsymbol{y}}_j) = p(\widetilde{\boldsymbol{y}}_j), \quad j = 1, \dots, k. \tag{26}$$

7: **end for**
8: Compute $\widehat{p}_{k+1} = \underset{p \in \mathcal{B}_{EIM}}{\arg\max} \|p - \sum_{i=1}^{k} [\boldsymbol{\alpha}(p)]_i q_i\|_{L^\infty(X)}$.
9: Compute $\widetilde{\boldsymbol{y}}_{k+1} = \underset{\boldsymbol{y} \in X}{\arg\max} |\widehat{p}_{k+1}(\boldsymbol{y}) - \sum_{i=1}^{k} [\boldsymbol{\alpha}(\widehat{p}_{k+1})]_i q_i(\boldsymbol{y})|$.
10: Set $q_{k+1} = \frac{r_{k+1}}{r_{k+1}(\widetilde{\boldsymbol{y}}_{k+1})}$, where $r_{k+1}(\boldsymbol{y}) = \widehat{p}_{k+1}(\boldsymbol{y}) - \sum_{i=1}^{k} [\boldsymbol{\alpha}(\widehat{p}_{k+1})]_i q_i(\boldsymbol{y})$.
11: Update $k \leftarrow k + 1$.
12: **end while**

by OGR and EIM, respectively. Then for any $\widehat{K} \in \mathbb{N}$, $\widehat{K} \leq \min\{K_{OGR}, K_{max}\}$, *it holds that* $\widetilde{\boldsymbol{x}}_j = \widetilde{\boldsymbol{y}}_j$ *for* $j = 1, \dots, \widehat{K}$ *and* $\operatorname{span}\{p_i\}_{i=1}^{\widehat{K}} = \operatorname{span}\{q_i\}_{i=1}^{\widehat{K}}$.

Proof. Notice that EIM considers all elements of $\mathcal{B}$ in each iteration and does not reorder the set $\mathcal{B}$. Thus, we may assume that the polynomials in the set $\mathcal{B}$ are ordered as OGR selects them, i.e. the $\{p_i\}_{i=1}^{K_{OGR}}$ selected by OGR are exactly the first K_{OGR} polynomials in $\mathcal{B}$.

We now prove the result by induction. Since $\|p\|_{L^\infty(X)} = \max_{\boldsymbol{x} \in X} |p(\boldsymbol{x})|$ for any $p \in \mathcal{P}_n$, OGR and EIM choose the same first point $\widetilde{\boldsymbol{x}}_1 = \widetilde{\boldsymbol{y}}_1$ and corresponding polynomial $p_{\ell_1} = \widehat{p}_1$. By the definition of q_1 in line 3, we obtain $\operatorname{span}(p_{\ell_1}) = \operatorname{span}(q_1)$.

Next, we assume that at iteration $k \leq \widehat{K}$ we have $\widetilde{\boldsymbol{x}}_j = \widetilde{\boldsymbol{y}}_j$ for $j = 1, \dots, k$, and $\widehat{p}_i = p_i$ in EIM for $i = 1, \dots, k$, implying that $\operatorname{span}\{p_i\}_{i=1}^{k} = \operatorname{span}\{q_i\}_{i=1}^{k}$. From **P1** we obtain that (26) has a unique solution $\boldsymbol{\alpha}_{EIM}^{\ell}$ for $\ell = 0, \dots, K$. Since $\operatorname{span}\{p_i\}_{i=1}^{k} = \operatorname{span}\{q_i\}_{i=1}^{k}$, the optimality system of (25) coincides with (26). Thus, also (25) has a unique solution $\boldsymbol{\alpha}_{OGR}^{\ell}$ for $\ell = k+1, \dots, K$ with $\sum_{i=1}^{k} [\boldsymbol{\alpha}_{OGR}^{\ell}]_{i+1} p_i = \sum_{i=1}^{k} [\boldsymbol{\alpha}_{EIM}^{\ell}]_{i+1} q_i$. From **P2** we know that $p_\ell(\boldsymbol{x}) - \sum_{i=1}^{k} \alpha_{i+1}^{\ell} q_i(\boldsymbol{x}) = 0$ for $\ell = 1, \dots, k$. On the other hand, the polynomials $p_{k+1}, \dots, p_K$ are linearly independent from $\{p_i\}_{i=1}^{k}$. Thus, the splitting step (25) of OGR is equivalent to the two maximization problems of EIM in lines 8-9, meaning that we obtain $\widetilde{\boldsymbol{x}}_{k+1} = \widetilde{y}_{k+1}$ and $p_{\ell_{k+1}} = \widehat{p}_{k+1}$. Finally, by the definition of q_{k+1} we obtain $\operatorname{span}\{p_i\}_{i=1}^{\widehat{K}} = \operatorname{span}\{q_i\}_{i=1}^{\widehat{K}}$. □

Algorithm 7 DGR for approximation by polynomials

Require: A data set $\mathcal{Z} = \{(\boldsymbol{x}_j, y_j)\}_{j=1}^{J}$ for $J \in \mathbb{N}$ and a tolerance tol > 0.

1: Find $(\widetilde{\boldsymbol{x}}_1, \widetilde{y}_1)$ that solves

$$\max_{(\boldsymbol{x},y)\in\mathcal{Z}} |y|^2. \tag{27}$$

2: Update $\mathcal{Z} \leftarrow \mathcal{Z} \setminus (\widetilde{\boldsymbol{x}}_1, \widetilde{y}_1)$ and set $f_{split} = |\widetilde{y}_1|$, $k = 1$.

3: **while** $k \leq J - 1$ and $f_{split} \geq$ tol **do**

4: Fitting step: Find $p^k \in \mathcal{P}_{k-1}$ that interpolates $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{y}_j)\}_{j=1}^{k}$.

5: Splitting step: Find $(\widetilde{\boldsymbol{x}}_{k+1}, \widetilde{y}_{k+1})$ that solves

$$\max_{(\boldsymbol{x},y)\in\mathcal{Z}} |p^k(\boldsymbol{x}) - y|^2. \tag{28}$$

6: Update $\mathcal{Z} \leftarrow \mathcal{Z} \setminus (\widetilde{\boldsymbol{x}}_{k+1}, \widetilde{y}_{k+1})$, $f_{split} \leftarrow |p^k(\widetilde{\boldsymbol{x}}_{k+1}) - \widetilde{y}_{k+1}|$ and $k \leftarrow k + 1$.

7: **end while**

4.3 Practical implementation aspects

To solve the initialization problems (11) and (23), and the splitting-step problems (13) and (25) numerically, we consider a strategy used for both the Leja points (see, e.g., [23]) and EIM (see, e.g., [15, 17]). The idea is to replace the input space X by a finite approximation (e.g., a grid), which we denote by X_h, compute the value of the cost function at every $\boldsymbol{x} \in X_h$, and select the point $\widetilde{\boldsymbol{x}}$ corresponding to the maximum value.

Notice that the computational effort of this approach grows exponentially with respect to the dimension of X (increasing with N and n). A possible solution, based on the so-called Fast Leja points (see [24]), is to adaptively enlarge the discrete set X_h in each iteration. A similar strategy can also be applied to our GR algorithms. However, the discussion of these implementation details goes beyond the scope of this work.

4.4 Data approximation GR for polynomial interpolation

Following the idea described in Section 4.3 and replacing the space X by $X_h = \{\boldsymbol{x}_j\}_{j=1}^{J}$, $J \in \mathbb{N}$, the evaluations of the polynomials $p_1, \ldots, p_K$ on X_h can be precomputed, resulting in a discrete set $\mathcal{Z} := \{(\boldsymbol{x}_j, p_i(\boldsymbol{x}_j))\}_{\substack{i\in\{1,\ldots,N\},\\ j\in\{1,\ldots,J\}}}$. In this way, we can provide GR directly with $\mathcal{Z}$, instead of the set of polynomials $\mathcal{B}$. On the other hand, this approach can also be interpreted as GR trying to approximate the "data" $\mathcal{Z}$ by polynomial functions.

This motivates the application of DGR (Algorithm 3 in Section 3) to the setting of polynomial interpolation. The idea is to consider a data set $\mathcal{Z} = \{(\boldsymbol{x}_j, f(\boldsymbol{x}_j))\}_{j=1}^{J} \subset X \times \mathbb{R}$ and find the smallest number of data points $\widetilde{J} \in \mathbb{N}$, $\widetilde{J} \leq J$, such that the polynomial of degree $\widetilde{J} - 1$ interpolating $\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^{\widetilde{J}}$ also approximates well the remaining data $\mathcal{Z}\setminus\{(\widetilde{\boldsymbol{x}}_j, f(\widetilde{\boldsymbol{x}}_j))\}_{j=1}^{\widetilde{J}}$. For brevity, let us denote $y_j := f(\boldsymbol{x}_j)$, $j = 1, \ldots, J$. The new version of DGR is stated in Algorithm 7. Notice that the fitting step in line 4 is no longer formulated as a least squares problem, since the interpolation of k points by a polynomial of degree $k - 1$ is exact.

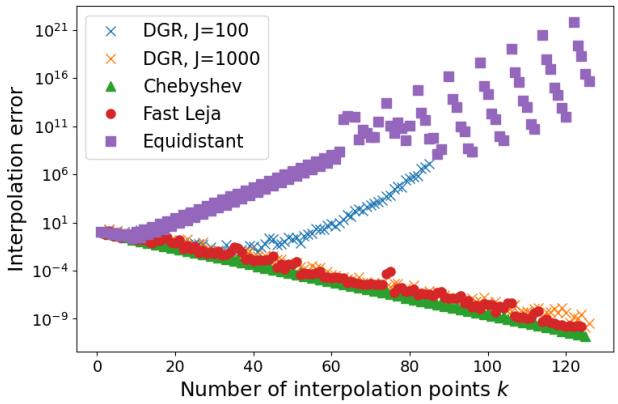

Fig. 2: Interpolation error (30) for the Runge function (29), using the barycentric Lagrange interpolation for the points computed by DGR at iteration k for a data set of size $J = 100$ (blue crosses) and $J = 1000$ (orange crosses), and for Chebyshev nodes (green triangles), Fast Leja points (red circles) and equidistant points (purple squares) of degree k.

Let us now test the performance of DGR to interpolate the famous Runge function

$$f(x) = \frac{1}{1 + 25x^2}, \qquad x \in [-1, 1]. \tag{29}$$

We provide DGR with a tolerance $\text{tol} = 10^{-10}$ and a data set $\{(x_j, f(x_j))\}_{j=1}^{J}$ with equidistant points $x_j = -1 + 2\frac{j-1}{J-1}$, $j = 1, \ldots, J$. To solve the interpolation problem in the fitting step (line 4 of DGR), we use the barycentric Lagrange interpolation (see [25]). To test the performance of DGR, we evaluate the interpolation error

$$\max_{x \in X_h} |f(x) - p^k(x)| \tag{30}$$

at each iteration k of DGR on a fine grid $X_h \subset [-1, 1]$ of 100000 equidistant points. We then compare the result to the interpolation error for the same amount of Chebyshev nodes (compare, e.g., [2, (10.20)]) and Fast Leja points (see [24]). The results for data sets of sizes $J = 100$ and $J = 1000$ are shown in Figure 2.

We observe that DGR for a data set of size $J = 1000$ terminates after $k = 124$ iterations (by reaching the required tolerance $\text{tol} = 10^{-10}$) and is able to match the convergence rate of the Chebyshev nodes and Fast Leja points. If we only provide DGR with $J = 100$ data points, it begins to diverge after 40 iterations. However, this is to be expected since the data $\{(x_j, f(x_j))\}_{j=1}^{J}$ provided to DGR are produced using equidistant points x_j, which are known to have a divergent interpolation error for the Runge function (compare, e.g., [2, Example 8.1]). However, the interpolation error for pure equidistant points in Figure 2 already starts to diverge at 10 interpolation points.

Obviously, the computational effort of DGR is larger than that in computing the Chebyshev nodes or Fast Leja points. Nevertheless, our investigations serve as a proof-of-concept that this data GR framework can perform well for polynomials, with the advantage that it can be extended to other frameworks.

5 A greedy algorithm for function approximation by residual neural networks

The advantage of our class of greedy algorithms is that the approximation ansatz does not have to be linear with respect to the coefficients $\boldsymbol{\alpha}$. This allows us to extend them to the field of deep learning, where a neural network depends nonlinearly on parameters, like weights and biases as in the following example.

We consider the following class of residual neural networks. For a network $F_{W,b}$ of depth m with activation function σ and input $\boldsymbol{x} =: \boldsymbol{z}^0$, we let

$$\boldsymbol{z}^i = \begin{cases} \sigma(W^{[i-1]}\boldsymbol{z}^{i-1} + \boldsymbol{b}^{[i-1]}) & \text{for } i \in \{1, m\}, \\ \boldsymbol{z}^{i-1} + \sigma(W^{[i-1]}\boldsymbol{z}^{i-1} + \boldsymbol{b}^{[i-1]}) & \text{else.} \end{cases}$$

In other words, the input of each hidden layer is summed directly to its output. That automatically implies that the widths of all hidden layers are equal. Thus, the architecture of any such network $F_{W,b}$ is determined by its depth m and the widths of all hidden layers d.

For inputs $(\widetilde{\boldsymbol{x}}_j)_{j=1}^J$, the problem of approximating a function f by a neural network can be formulated as

$$\min_{(W,b)\in\mathcal{W}(d)} L\Big((F_{W,b}(\widetilde{\boldsymbol{x}}_j))_{j=1}^J, (f(\widetilde{\boldsymbol{x}}_j))_{j=1}^J\Big), \tag{31}$$

where L is a loss function, for example, the mean squared error

$$L\Big((F_{W,b}(\widetilde{\boldsymbol{x}}_j))_{j=1}^J, (f(\widetilde{\boldsymbol{x}}_j))_{j=1}^J\Big) = \frac{1}{J}\sum_{j=1}^J \|f(\widetilde{\boldsymbol{x}}_j) - F_{W,b}(\widetilde{\boldsymbol{x}}_j)\|_2^2. \tag{32}$$

In Section 5.1, we formulate a combination of OGR and DGR that is able to simultaneously select a minimal network architecture (namely m and d) and an optimal subset of training data for a given data set. Afterwards, in Section 5.3, we test the performance of our new method, using residual neural networks to learn the Runge function and a spiral data set.

5.1 OGR for minimal neural network structures

In many machine learning applications the training data is already available. However, the choice of an appropriate/optimal neural network structure for the specific data is often only based on heuristics, experience, or determined by trial and error. This leaves open the possibility of choosing a suboptimal network size, which can lead to

Algorithm 8 NOGR for deep neural networks

Require: A set of networks $\{F^1_{W_i,b_i}\}_{i=1}^K$ with depths $\{m_i\}_{i=1}^K$ and widths $\{d_i\}_{i=1}^K$ for $K \in \mathbb{N}$, maximum network depth m_{max} and maximum hidden layer width d_{max}, a set of training data $\mathcal{Z} = \{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^J$ for $J \in \mathbb{N}$, a maximum number of iterations $K_{it} \leq J$ and two tolerances $\mathrm{tol}_{fit}, \mathrm{tol}_{split} > 0$.

Set $\mathcal{L} = \{1, \ldots, K\}$.
$((\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{y}}_1), f_{split}, F^1_{NOGR}) \leftarrow$ splitting_step$(\mathcal{L}, \{F^1_{W_\ell,b_\ell}\}_{\ell\in\mathcal{L}}, \mathcal{Z})$.
Update $\mathcal{Z} \leftarrow \mathcal{Z} \setminus \{(\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{y}}_1)\}$ and set $k = 1$.
while $k \leq K_{it}$ and $f_{split} \geq \mathrm{tol}_{split}$ **do**
    $(\{F^{k+1}_{W_i,b_i}\}_{i=1}^K, \mathcal{L}) \leftarrow$ fitting_step$(\{F^k_{W_i,b_i}\}_{i=1}^K, (\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)_{j=1}^k, \mathrm{tol}_{fit})$
    $((\widetilde{\boldsymbol{x}}_{k+1}, \widetilde{\boldsymbol{y}}_{k+1}), f_{split}, F^{k+1}_{NOGR}) \leftarrow$ splitting_step$(\mathcal{L}, \{F^{k+1}_{W_\ell,b_\ell}\}_{\ell\in\mathcal{L}}, \mathcal{Z})$.
    Update $\mathcal{Z} \leftarrow \mathcal{Z} \setminus \{(\widetilde{\boldsymbol{x}}_{k+1}, \widetilde{\boldsymbol{y}}_{k+1})\}$ and $k \leftarrow k + 1$
end while
return: Training points $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)\}_{j=1}^k$ and the (trained) network $F_{NOGR} = F^k_{NOGR}$ with smallest validation loss f_{split}.

Algorithm 9 splitting_step

Require: A set of indices $\mathcal{L}$, the corresponding networks $\{F_{W_\ell,b_\ell}\}_{\ell\in\mathcal{L}}$, and a set of points $\mathcal{Z} = \{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^{\widetilde{J}}$, with $\widetilde{J} \in \mathbb{N}$.

for $\ell \in \mathcal{L}$ **do**
    Splitting step: Find $(\widehat{\boldsymbol{x}}_\ell, \widehat{\boldsymbol{y}}_\ell)$ that solve

$$\max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{Z}} L(F_{W_\ell,b_\ell}(\boldsymbol{x}), \boldsymbol{y}). \tag{33}$$

    Set $f_\ell = L(F_{W_\ell,b_\ell}(\widehat{\boldsymbol{x}}_\ell), \widehat{\boldsymbol{y}}_\ell)$.
end for
Compute $\widetilde{\ell} = \arg\min_{\ell\in\mathcal{L}} f_\ell$.
Set $(\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}}) = (\widehat{\boldsymbol{x}}_{\widetilde{\ell}}, \widehat{\boldsymbol{y}}_{\widetilde{\ell}})$, $f_{split} = f_{\widetilde{\ell}}$, and $F_{NOGR} = F_{W_{\widetilde{\ell}},b_{\widetilde{\ell}}}$.
return: The training point $(\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}})$ with validation loss f_{split} and the corresponding network F_{NOGR}.

under/overfitting (see, e.g., [26, Section 5.2]). Moreover, even if a "good" network structure is found, it is a common issue that the data is imbalanced, leading to biased models (see, e.g., [5, Section 5.3.1]) and thereby lack of robustness.

To address these issues, we introduce a new method, called Network OGR (NOGR), which combines the ideas of OGR (Algorithm 2) and DGR (Algorithm 3). The main idea of this new greedy algorithm is to select simultaneously a subset of optimal training data points and the smallest network (trained on the selected data) capable of well representing the non-selected data. NOGR is stated in Algorithm 8, while its splitting and fitting steps are detailed in Algorithms 9 and 10. For a better overview, we summarize the role of some variables in Table 1.

Let us describe the first iteration of NOGR in more detail. First, NOGR applies the splitting step (Algorithm 9) to the full set of networks $\{F^1_{W_i,b_i}\}_{i=1}^K$. In this step, for each network a training point $(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{y}}) \in \mathcal{Z}$ is computed by maximizing the validation

Algorithm 10 fitting_step

Require: A set of networks $\{F_{W_i,b_i}\}_{i=1}^K$ with depths $\{m_i\}_{i=1}^K$ and widths $\{d_i\}_{i=1}^K$ for $K \in \mathbb{N}$, a set of points $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)\}_{j=1}^k$ for $k \in \mathbb{N}$, and tolerances $\mathrm{tol}_{fit} \in \mathbb{R}$, m_{max} and d_{max}.

Set $\mathcal{K} = \emptyset$.
while $\mathcal{K} = \emptyset$ **do**
  for $\ell = 1, \dots, K$ **do**
    Fitting step: Train F_{W_ℓ,b_ℓ} on $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)\}_{j=1}^k$, i.e., find $(\widetilde{W}_\ell, \widetilde{b}_\ell)$ that solve

$$\min_{(W,b)\in\mathcal{W}(d^\ell)} L\Big(\{F_{W,b}(\widetilde{\boldsymbol{x}}_j)\}_{j=1}^k, \{\widetilde{\boldsymbol{y}}_j\}_{j=1}^k\Big), \tag{34}$$

    and set $F_{W_\ell,b_\ell} = F_{\widetilde{W}_\ell,\widetilde{b}_\ell}$.
  end for
  Set $\mathcal{K} = \{\ell \in \{1, \dots, K\} \mid L\Big(\{F_{W_\ell,b_\ell}(\widetilde{\boldsymbol{x}}_j)\}_{j=1}^k, \{\widetilde{\boldsymbol{y}}_j\}_{j=1}^k\Big) < \mathrm{tol}_{fit}\}$.
  for $i \in \{1, \dots, K\} \setminus \mathcal{K}$ **do**
    Increase m_i and/or d_i (not beyond m_{max}, d_{max}), and update F_{W_i,b_i} accordingly.
  end for
  if $\min_{i\in\{1,\dots,K\}} (m_i, d_i) = (m_{max}, d_{max})$ **then**
    STOP with message: *"Was not able to fit the data, stopping NOGR!"*
  end if
end while
return: A set of trained networks $\{F_{W_j,b_j}\}_{j=1}^K$ and a set of indices $\mathcal{K}$.

Variable name	Description/Role
m_{max}, d_{max}	Bounds on the depth and hidden layer width of the networks $(F_{W_j,b_j})_{j=1}^K$.
tol_{fit}	Tolerance used in the fitting step to decide whether the training of a network was successful or not.
tol_{split}	Tolerance used to terminate NOGR if the validation error of a network is sufficiently small.
$\mathcal{K}$	Set of indices computed during the fitting step, indicating which networks were successfully trained.
$\mathcal{L}$	Set of indices deciding which networks are considered in the splitting step. At iterations $k \geq 2$ we have $\mathcal{L} = \mathcal{K}$.
$\mathcal{Z}$	Set of training points not selected by NOGR at a certain iteration.

Table 1: Description for some variables used in Algorithms 8, 9, and 10.

error in (33). Among all these training points, the one that corresponds to the network F_{NOGR}^1 with the smallest validation error f_{split} is selected and denoted by $(\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{y}}_1)$. Next, NOGR checks whether the network F_{NOGR}^1 already represents well enough all the training data, namely the validation error f_{split} is below the tolerance tol_{split}. If that is not the case, NOGR continues with the fitting step (Algorithm 10). Here, all networks are trained on $(\widetilde{\boldsymbol{x}}_1, \widetilde{\boldsymbol{y}}_1)$. If the training of some networks is unsuccessful, i.e., the final loss of the corresponding training problems is larger than the tolerance tol_{fit}, then the depths and/or widths of these networks are increased. The fitting step is repeated until one training was successful or the maximum network depth m_{max} and

maximum width of the hidden layers d_{max} are reached. In the latter case, the process is stopped because none of the admissible network architecture was able to represent the selected data. Otherwise, if the training was successful for one or more of the networks, these (trained) networks are registered in the set $\mathcal{L}$ for the next splitting step.

5.2 Practical implementation aspects

The main advantage of the particular class of residual neural networks that we introduced at the beginning of Section 5 is that it limits the amount of choices when upgrading the network. One can either add a new hidden layer with the same width as all the other hidden layers or increase the width of all existing hidden layers by 1.

Let us explain this upgrade strategy in more detail. We initialize NOGR with a set of networks $\{F_{W_i,b_i}\}_{i=1}^{K}$ that have no hidden layers (i.e., $m_i = d_i = 0$), but different weights and biases that are randomly chosen in $[-1, 1]$. In the first architecture update in lines 7-9 of the fitting step (Algorithm 10), we replace all $\{F_{W_i,b_i}\}_{i\in\{1,\dots,K\}\setminus\mathcal{K}}$ with random networks that have exactly one hidden layer of width 1, i.e., $m_i = d_i = 1$. For all further updates, we use the smallest (successfully trained) network as reference size and add either a new hidden layer, or increase the width of all hidden layers by 1. More precisely, by defining $\#F_{W,b}$ as the total number of weights and biases in the network $F_{W,b}$, we compute the index of the smallest (successfully trained) network $\widehat{\ell}$ as

$$\widehat{\ell} = \begin{cases} \operatorname*{arg\,min}_{\ell\in\mathcal{K}} \#F_{W_\ell,b_\ell} & \text{if } \mathcal{K} \neq \emptyset, \\ \operatorname*{arg\,min}_{\ell\in\{1,\dots,K\}} \#F_{W_\ell,b_\ell} & \text{else.} \end{cases}$$

Then we set $(\widetilde{m}_1, \widetilde{d}_1) = (m_{\widehat{\ell}} + 1, d_{\widehat{\ell}})$ and $(\widetilde{m}_2, \widetilde{d}_2) = (m_{\widehat{\ell}}, d_{\widehat{\ell}} + 1)$, and replace the set $\{F_{W_i,b_i}\}_{i\in\{1,\dots,K\}\setminus\mathcal{K}}$ with random networks such that half of them have depth $\widetilde{m}_1$ and hidden layers of width $\widetilde{d}_1$, and the other half have depth $\widetilde{m}_2$ and hidden layers of width $\widetilde{d}_2$.

Remark 3. *NOGR can also be extended to other classes of neural networks. The main challenge here is to design an appropriate strategy to update the network architecture (lines 7-9 in Algorithm 10). For general feedforward neural networks, for example, the number of possible incremental upgrades (by one node) grows proportionally to the number of hidden layers.*

5.3 Numerical experiments

We test the performance of NOGR, using the residual network structure and upgrade strategy discussed in Section 5.2. In Section 5.3.1, we consider again the Runge function. Afterwards, in Section 5.3.2, we apply NOGR to a two-dimensional spiral data set. Our implementations use PyTorch [27].

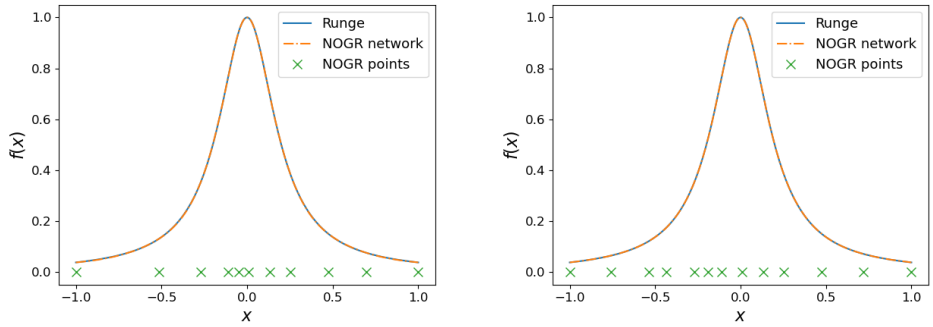

Fig. 3: **Left**: Runge function (blue line), inputs $(\widetilde{x}_j)_{j=1}^{11}$ of the data points selected by NOGR (green crosses), for tolerances $\text{tol}_{fit} = \text{tol}_{split} = 10^{-5}$, and output of the network F_{NOGR} on an equidistant grid X_h of 100000 points (orange dash-dotted line). **Right**: Same as left, but for smaller tolerances $\text{tol}_{fit} = \text{tol}_{split} = 10^{-7}$.

5.3.1 An optimal residual network for the Runge function

Our goal is to find the smallest residual network that is able to learn the Runge function (29). Similar to Section 4.4, we consider a data set $\{(x_j, f(x_j))\}_{j=1}^{J}$ for $J = 100$ equidistant points $x_j \in [-1, 1]$.

For all networks in this section, we fix input and output size to $d_0 = d_m = 1$ and use the hyperbolic tangent as activation function, i.e., $\sigma(x) = \tanh(x)$. Since we want to learn the outputs of a function, we consider the mean squared error loss (32). We solve the fitting-step problem (34) using a BFGS method with the full gradient to have good convergence properties. Notice that there is no need to apply any stochastic optimization approach, since the fitting step at iteration k trains the networks only on the selected training points $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)\}_{j=1}^{k}$.

We run NOGR for a set of $K = 100$ networks, a maximum network width and depth of 20 and $\text{tol}_{fit} = \text{tol}_{split} = 10^{-5}$. The algorithm stops after 11 iterations, selecting data points $(\widetilde{x}_j, f(\widetilde{x}_j))_{j=1}^{11}$ and a network F_{NOGR} of size $(1, 1, 1, 1, 1, 1, 1)$, i.e., 5 hidden layers of width 1, accumulating to 11 trainable parameters. The loss of F_{NOGR} on the full training data set of 100 points is $5.32 \cdot 10^{-7}$, the loss on a fine grid $X_h \subset [-1, 1]$ of 100000 equidistant points is $5.38 \cdot 10^{-7}$. The results are plotted in Figures 3 and 4.

For a fixed network $F_{W,b}$ the interpolation error in Figure 4 is defined as

$$\max_{x \in X_h} |f(x) - F_{W,b}(x)|. \tag{35}$$

To obtain the plots in Figure 4, we train the networks $\{F^k_{W_i,b_i}\}_{i=1}^{100}$ used in the NOGR fitting step in the iterations $k = 3, \ldots, 11$ on the data points $\{(x_j, f(x_j))\}_{j=1}^{k}$, where the inputs x_j correspond to k Chebyshev nodes, Fast Leja points or equidistant points, respectively. We then select, for each iteration and each input set, the network $F^k_{W,b}$ with the smallest interpolation error (35), and plot this error in the left-hand side plot

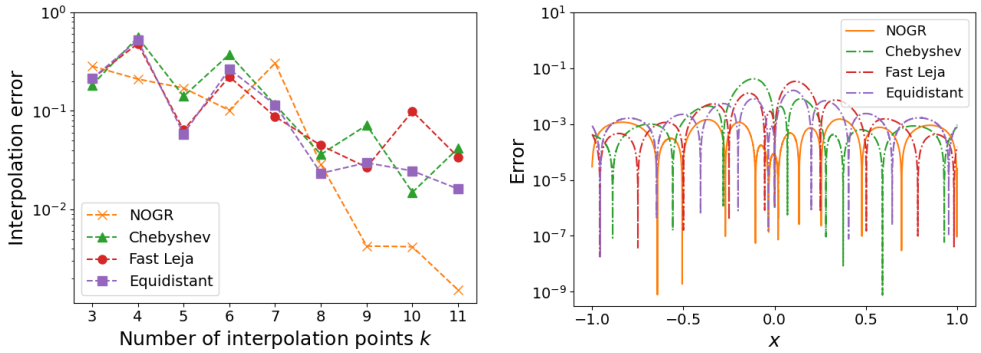

Fig. 4: **Left**: Interpolation error (35) of F^k_{NOGR} at iteration $k \in \{3, \dots, 11\}$ (orange crosses) for tolerances $\text{tol}_{fit} = \text{tol}_{split} = 10^{-5}$, compared to the smallest interpolation error achieved by one of the networks $\{F^k_{W_i,b_i}\}_{i=1}^{100}$ trained on k Chebyshev nodes (green triangles), Fast Leja points (red circles) or equidistant points (purple squares), respectively. **Right**: Error (36) for the final network $F_{NOGR} = F^{11}_{NOGR}$ (orange line) evaluated on a fine grid of 100000 points in $[-1, 1]$, compared to the error for the one network from the set $\{F^{11}_{W_i,b_i}\}_{i=1}^{100}$ that achieved the smallest interpolation error when trained on 11 Chebyshev nodes (green dash-dotted line), Fast Leja points (red dash-dotted line) or equidistant points (purple dash-dotted line), respectively.

of Figure 4. For the final iteration $k = 11$, we additionally plot the absolute difference

$$|f(x) - F_{W,b}(x)|, \qquad x \in X_h, \tag{36}$$

for the networks $F^{11}_{W,b}$, selected for the different sets of inputs, in the right-hand side plot of Figure 4. We observe that all input sets perform similarly until iteration $k = 8$. Afterwards, NOGR outperforms the other input sets by one order of magnitude. One possible reason is the fact that NOGR places more inputs towards the center of the interval, where the Runge function exhibits more rapid changes. Indeed, as we can see from the right-hand side plot in Figure 4, the networks corresponding to Chebyshev nodes, Fast Leja points and equidistant points have the largest error in the center area of the interval. On the other hand, the error peaks for the network F_{NOGR} seem to be the smallest in the center. This could also explain why the equidistant points perform better than Chebyshev and Leja points, which tend to accumulate more points towards the bounds of the interval. However, we also notice that the points, where the error is dropping very low (i.e., the intersection points between the smooth network functions $F_{W,b}(x)$ and the Runge function), do not necessarily coincide with the training points. The reason for this is that there is no guarantee that a trained network will interpolate the training data exactly, especially since we are dealing with rather small networks.

To better understand the role of the tolerances in NOGR, we repeat our test with $\text{tol}_{fit} = \text{tol}_{split} = 10^{-7}$. As expected, the algorithm requires more iterations (13) and a larger network F_{NOGR} of size $(1, 2, 2, 2, 2, 2, 1)$ in order to fit the data in accordance with the smaller tolerances. This means that F_{NOGR} has one more layer than for

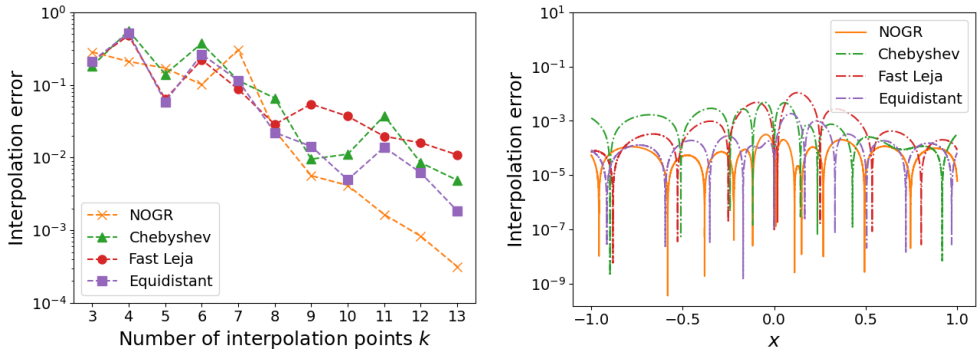

Fig. 5: Same as Figure 4, but for smaller tolerances $\text{tol}_{fit} = \text{tol}_{split} = 10^{-7}$.

the larger tolerances, and the width of all hidden layers is increased by 1. Although this corresponds to "only" two upgrades of the network structure, the number of trainable parameters more than triples to a total of 36. On the other hand, we also obtain a smaller loss on the full training data set ($9.48 \cdot 10^{-9}$) and on the fine grid X_h ($9.56 \cdot 10^{-9}$). The results are plotted in Figures 3 and 5. As before, the networks trained on the data selected by NOGR outperform the ones trained on data corresponding to other input sets.

5.3.2 Optimal learning of a two-dimensional spiral data set

In this numerical experiment, we apply our NOGR to find the smallest residual neural network that separates two spirals in $\mathbb{R}^2$. The data set for this example is adapted from [28], and consists of 600 input vectors $\boldsymbol{x} \in \mathbb{R}^2$ and corresponding labels $\boldsymbol{y} \in \{\binom{0}{1}, \binom{1}{0}\} \subset \mathbb{R}^2$ that indicate whether $\boldsymbol{x}$ belongs to the red or blue spiral. We randomly select $J = 450$ data points for NOGR and set aside the remaining 150 points to test the performance of NOGR a posteriori. For all networks in this section, we set input and output size to $d_0 = d_m = 2$. As activation function, we use again the hyperbolic tangent, i.e., $\sigma(x) = \tanh(x)$. Since we consider a classification problem, we use the cross entropy loss in both sub-problems of NOGR

$$L(\boldsymbol{z}, \boldsymbol{y}) := -\sum_{i=1}^{2} \log \frac{\exp([\boldsymbol{z}]_i)}{\sum_{j=1}^{2} \exp([\boldsymbol{z}]_j)} [\boldsymbol{y}]_i.$$

We solve the fitting-step problem (34) with a (full gradient) BFGS method.

We run NOGR for a set of $K = 100$ networks, a maximum network width and depth of 20 and $\text{tol}_{fit} = \text{tol}_{split} = 10^{-7}$. The algorithm stops after 58 iteration with a network F_{NOGR} of size $(2, 4, 2)$, i.e., one hidden layer of width 4, accumulating to 20 trainable parameters. The loss of F_{NOGR} on the full training data set of 450 points is $1.06 \cdot 10^{-9}$, the loss on the 150 test points is $6.36 \cdot 10^{-9}$ with a classification accuracy of 100%. The spiral training data, the outputs of the trained network F_{NOGR} and the selected data points $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)\}_{i=1}^{58}$ are plotted in Figure 6. As we can see, the relatively

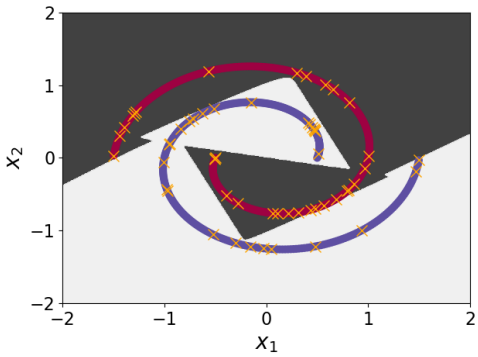

Fig. 6: Full spiral data set (red and blue curve), inputs $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^{58}$ of the data points selected by NOGR (orange crosses) and decision output of the network F_{NOGR} for all points on the domain (dark and light gray areas).

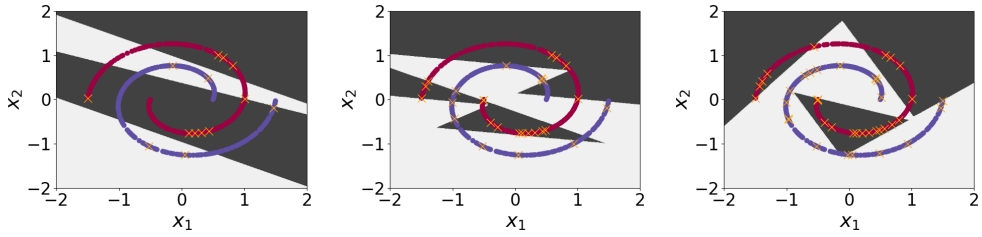

Fig. 7: Spiral training data (red and blue curve), inputs $(\widetilde{\boldsymbol{x}}_j)_{j=1}^{k}$ of the data points selected by NOGR at iterations $k = 14, 28, 42$ (from left to right) and decision output of the network F_{NOGR}^{k} for all points on the domain (dark and light gray areas).

small network F_{NOGR} selected by NOGR is able to separate the two spirals, while using only $\approx 13\%$ of the training data. In fact, the interface drawn by F_{NOGR} between the dark and light area is almost the best possible linear separation for the fewest number of vertices. The placement of $\{(\widetilde{\boldsymbol{x}}_j, \widetilde{\boldsymbol{y}}_j)\}_{j=1}^{58}$ is also very interesting. Many of them seem to be chosen close to vertices and edges of the separation interface. To better understand the interplay of selecting points and training the network during NOGR, we show in Figure 7 the inputs $\{\widetilde{\boldsymbol{x}}_j\}_{j=1}^{k}$ of the data points selected by NOGR and the output of the intermediate network F_{NOGR}^{k}, at iterations $k = 14, 28, 42$.

6 Conclusions

In this paper, we developed and analyzed two classes of greedy reconstruction algorithms to improve the data selection process while optimizing the structure of the ansatz within a given family of approximation functions. Theoretical and numerical studies on polynomial interpolation and function approximation by neural networks demonstrated the efficiency of the proposed algorithms. Our algorithms appear very

promising, especially in machine learning applications. In particular, the structure and optimization features of our NOGR offer significant potential for future research with a wide range of possible applications.

References

[1] Davis, P.J.: Interpolation and Approximation. Blaisdell Publishing Company, Massachussetts (1963)

[2] Quarteroni, A., Sacco, R., Saleri, F.: Numerical Mathematics (Texts in Applied Mathematics). Springer, Berlin, Heidelberg (2006)

[3] Sachdeva, N., McAuley, J.: Data distillation: A survey. Transactions on Machine Learning Research (2023)

[4] DeVore, R., Hanin, B., Petrova, G.: Neural network approximation. Acta Numerica **30**, 327–444 (2021) https://doi.org/10.1017/S0962492921000052

[5] Rasheed, K., Qayyum, A., Ghaly, M., Al-Fuqaha, A., Razi, A., Qadir, J.: Explainable, trustworthy, and ethical machine learning for healthcare: A survey. Computers in Biology and Medicine **149**, 106043 (2022) https://doi.org/10.1016/j.compbiomed.2022.106043

[6] Maday, Y., Salomon, J.: A greedy algorithm for the identification of quantum systems. In: Proceedings of the 48th IEEE Conference on Decision and Control, 2009, Held Jointly Whit the 28th Chinese Control Conference (CDC/CCC 2009). IEEE Conference on Decision and Control, pp. 375–379 (2009)

[7] Buchwald, S., Ciaramella, G., Salomon, J.: Analysis of a greedy reconstruction algorithm. SIAM J. Control Optim. **59**(6), 4511–4537 (2021) https://doi.org/10.1137/20M1373384 https://doi.org/10.1137/20M1373384

[8] Buchwald, S., Ciaramella, G., Salomon, J., Sugny, D.: Greedy reconstruction algorithm for the identification of spin distribution. Phys. Rev. A **104**, 063112 (2021) https://doi.org/10.1103/PhysRevA.104.063112

[9] Buchwald, S., Ciaramella, G., Salomon, J.: Gauss–Newton oriented greedy algorithms for the reconstruction of operators in nonlinear dynamics. SIAM J. Control Optim. **62**(3), 1343–1368 (2024) https://doi.org/10.1137/23M1552929

[10] Buchwald, S., Ciaramella, G., Salomon, J., Sugny, D.: A SPIRED code for the reconstruction of spin distribution. Comput. Phys. Commun. **299**, 109126 (2024) https://doi.org/10.1016/j.cpc.2024.109126

[11] Leja, F.: Sur certaines suites liées aux ensembles plan et leur application à la représentation conforme. Ann. Polon. Math. **4**, 8–13 (1957)

[12] Edrei, A.: Sur les déterminants récurrents et les singularités d'une fonction donnée par son développement de Taylor. Compositio Mathematica **7**, 20–88 (1940)

[13] Bos, L., De Marchi, S., Sommariva, A., Vianello, M.: Computing multivariate Fekete and Leja points by numerical linear algebra. SIAM Journal on Numerical Analysis **48**(5), 1984–1999 (2010) https://doi.org/10.1137/090779024 https://doi.org/10.1137/090779024

[14] Baglama, J., Calvetti, D., Reichel, L.: Iterative methods for the computation of a few eigenvalues of a large symmetric matrix. BIT Numerical Mathematics **36**, 400–421 (1996)

[15] Barrault, M., Maday, Y., Nguyen, N., Patera, A.: An 'empirical interpolation' method: Application to efficient reduced-basis discretization of partial differential equations. Comptes Rendus Mathematique **339**, 667–672 (2004) https://doi.org/10.1016/j.crma.2004.08.006

[16] Grepl, M., Maday, Y., Nguyen, N., Patera, A.: Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. http://dx.doi.org/10.1051/m2an:2007031 **41** (2007) https://doi.org/10.1051/m2an:2007031

[17] Maday, Y., Nguyen, N.C., Patera, A.T., Pau, S.H.: A general multipurpose interpolation procedure: the magic points. Communications on Pure and Applied Analysis **8**(1), 383–404 (2009) https://doi.org/10.3934/cpaa.2009.8.383

[18] Bebendorf, M., Maday, Y., Stamm, B.: In: Quarteroni, A., Rozza, G. (eds.) Comparison of Some Reduced Representation Approximations, pp. 67–100. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-02090-7_3 . https://doi.org/10.1007/978-3-319-02090-7_3

[19] Maday, Y., Mula, O.: In: Brezzi, F., Colli Franzone, P., Gianazza, U., Gilardi, G. (eds.) A Generalized Empirical Interpolation Method: Application of Reduced Basis Techniques to Data Assimilation, pp. 221–235. Springer, Milano (2013). https://doi.org/10.1007/978-88-470-2592-9_13 . https://doi.org/10.1007/978-88-470-2592-9_13

[20] Argaud, J.-P., Bouriquet, B., de Caso, F., Gong, H., Maday, Y., Mula, O.: Sensor placement in nuclear reactors based on the generalized empirical interpolation method. Journal of Computational Physics **363**, 354–370 (2018) https://doi.org/10.1016/j.jcp.2018.02.050

[21] Barron, A.R., Cohen, A., Dahmen, W., DeVore, R.A.: Approximation and learning by greedy algorithms. The Annals of Statistics **36**(1), 64–94 (2008) https://doi.org/10.1214/009053607000000631

[22] Wendland, H.: Haar spaces and multivariate polynomials. In: Scattered Data

Approximation. Cambridge Monographs on Applied and Computational Mathematics, pp. 18–23. Cambridge University Press, Cambridge (2004)

[23] Reichel, L.: Newton interpolation at Leja points. BIT **30**, 332–346 (1990)

[24] Baglama, J., Calvetti, D., Reichel, L.: Fast Leja points. ETNA. Electronic Transactions on Numerical Analysis [electronic only] **7**, 124–140 (1998)

[25] Berrut, J.-P., Trefethen, L.N.: Barycentric Lagrange interpolation. SIAM Review **46**(3), 501–517 (2004) https://doi.org/10.1137/S0036144502417715 https://doi.org/10.1137/S0036144502417715

[26] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA (2016). http://www.deeplearningbook.org

[27] Ansel, J.e.a.: Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. ASPLOS '24, pp. 929–947. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3620665.3640366 . https://doi.org/10.1145/3620665.3640366

[28] Herberg, E., Herzog, R., Köhne, F., Kreis, L., Schiela, A.: Sensitivity-Based Layer Insertion for Residual and Feedforward Neural Networks (2023). https://arxiv.org/abs/2311.15995

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

04/2025 Andrini, D.; Magri, M.; Ciarletta, P.
Nonlinear morphoelastic theory of biological shallow shells with initial stress

02/2025 Corda, A.; Pagani, S.; Vergara, C.
Influence of patient-specific acute myocardial ischemia maps on arrhythmogenesis: a computational study

01/2025 Dede', L.; Parolini, N.; Quarteroni, A.; Villani, G.; Ziarelli, G.
SEIHRDV: a multi-age multi-group epidemiological model and its validation on the COVID-19 epidemics in Italy

109/2024 Liverotti, L.; Ferro, N.; Matteucci, M.; Perotto, S.
A PCA and mesh adaptation-based format for high compression of Earth Observation optical data with applications in agriculture

110/2024 Pederzoli, V.; Corti, M.; Riccobelli, D.; Antonietti, P.F.
A coupled mathematical and numerical model for protein spreading and tissue atrophy, applied to Alzheimer's disease

108/2024 Arostica, R.; Nolte, D.; Brown, A.; Gebauer, A.; Karabelas, E.; Jilberto, J.; Salvador, M.; Bucelli, M.; Piersanti, R.; Osouli, K.; Augustin, C.; Finsberg, H.; Shi, L.; Hirschvogel, M.; Pfaller, M.; Africa, P.C.; Gsell, M.; Marsden, A.; Nordsletten, D.; Regazzoni, F.; Plank, G.; Sundnes, J.; Dede’, L.; Peirlinck, M.; Vedula, V.; Wall, W.; Bertoglio, C.
A software benchmark for cardiac elastodynamics

107/2024 Chen, J.; Ballini, E.; Micheletti, S.
Active Flow Control for Bluff Body under High Reynolds Number Turbulent Flow Conditions Using Deep Reinforcement Learning

106/2024 Brunati, S.; Bucelli, M.; Piersanti, R.; Dede', L.; Vergara, C.
Coupled Eikonal problems to model cardiac reentries in Purkinje network and myocardium

105/2024 Bartsch, J.; Barakat, A.A.; Buchwald, S.; Ciaramella, G.; Volkwein, S.; Weig, E.M.
Reconstructing the system coefficients for coupled harmonic oscillators

104/2024 Cerrone, D.; Riccobelli, D.; Vitullo, P.; Ballarin, F.; Falco, J.; Acerbi, F.; Manzoni, A.; Zunino, P.; Ciarletta, P.
Patient-specific prediction of glioblastoma growth via reduced order modeling and neural

networks