SN Computer Science (2023) 4:230
https://doi.org/10.1007/s42979-022-01664-2

ORIGINAL RESEARCH

Visualizing High-Dimensional Functions with Dense Maps

Mateus Espadoto[1] · Francisco C. M. Rodrigues[1] · Nina S. T. Hirata[1] · Alexandru C. Telea[2]

Received: 3 July 2022 / Accepted: 30 December 2022 / Published online: 22 February 2023

Abstract
Multivariate functions have a central place in the development of techniques present many domains, such as machine learning and optimization research. However, only a few visual techniques exist to help users understand such multivariate problems, especially in the case of functions that depend on complex algorithms and variable constraints. In this paper, we propose a technique that enables the visualization of high-dimensional surfaces defined by such multivariate functions using a two-dimensional pixel map. We demonstrate two variants of it, OptMap, focused on optimization problems, and RegSurf, focused on regression problems in machine learning. Both our techniques are simple to implement, computationally efficient, and generic with respect to the nature of the high-dimensional data they address. We show how the two techniques can be used to visually explore a wide variety of optimization and regression problems.

Keywords Machine learning · Operations research · Optimization · Regression · Dimensionality reduction · Visualization · Dense maps

Introduction

Machine Learning (ML) and Operations Research (OR) methods are key ingredients in the data scientist's toolset. Machine Learning has reached high popularity in the last decade, growing from a field initially of interest to mathematicians and statisticians to the point of being a generic technology for many fields [1, 2]. Operations Research also plays a crucial role in many industries, from logistics to finance. Since its inception in the 1950s, it has delivered many tools for practitioners to improve their productivity, such as algebraic modeling languages like GAMS [3], AMPL [4], and JuMP [5], which enable the use of notation very close to the mathematical definition of optimization problems.

However, while being different approaches to solve problems with data, both ML and OR deal with functions having high-dimensional input spaces and sophisticated algorithms whose inner workings can be hard to understand. *Visualization* techniques help understanding such complex data pipelines by allowing users to explore the large, high-dimensional, and complex spaces they deal with. However, while many visualization techniques exist for high-dimensional *data* [6], only a few such techniques target the visualization of high-dimensional *functions*. To the best of our knowledge, we are not aware of any work that allows for the visualization of high-dimensional continuous functions by considering all their dimensions at once, rather than only a few sets of dimensions at a time.

Recently, we proposed OptMap [7], a technique for the visualization of surfaces generated by high-dimensional *explicit* functions of the form $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that model OR problems, whose input variables have known domains. In

Mateus Espadoto, Francisco C. M. Rodrigues, Nina S. T. Hirata, and Alexandru C. Telea have contributed equally to this work.

This article is part of the topical collection "Computer Vision, Imaging and Computer Graphics Theory and Applications" guest edited by Jose Braz, A. Augusto Sousa, Alexis Paljic, Christophe Hurter, and Giovanni Maria Farinella.

✉ Mateus Espadoto
mespadot@ime.usp.br

Francisco C. M. Rodrigues
caiomr@ime.usp.br

Nina S. T. Hirata
nina@ime.usp.br

Alexandru C. Telea
a.c.telea@uu.nl

[1] Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, São Paulo 05508-090, Brazil

[2] Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands

this paper, we complement OptMap with RegSurf, a visualization technique for high-dimensional *implicit* functions f which are inferred from data, as in ML regression problems. We show how OptMap and RegSurf are instances of a more general visualization framework for high-dimensional functions. This commonality allows us to propose efficient, easy-to-implement, and generic implementations for both our visualization techniques.

We next outline the joint contributions and specifics of OptMap and RegSurf. OptMap enables the OR practitioner to literally see the decision variables and constraint spaces using a two-dimensional dense map, regardless of the number of variables and constraints of the problem. RegSurf uses a similar 2D dense map to enable the ML practitioner to visualize the surface generated by a single-output regressor. In both cases, the map enables the user to *visually* explore the high-dimensional surface, regardless of the dimensionality (number of variables) of the input space.

OptMap can be used in several ways, such as a debugging aid to help diagnose errors in constraint definition; to provide insight in an optimizer's inner workings by plotting the path taken from a starting point to a solution; and to visually explore the high-dimensional space of the decision variables in terms of objective function value and constraint feasibility. In parallel, RegSurf enables the user to assess the goodness of fit of a regressor with respect to the training data, thereby complementing the existing methods that use a single-valued metric such as mean squared error or the R^2 score. RegSurf can be used to find regions of the input space where the data are underrepresented, which may cause the regressor's results to be biased, and to visually gauge if and where a regressor is under- or overfitting the data. Both OptMap and RegSurf can also be used to compare the performance of different regressors and solvers over the same data, and to gain insight over their inner workings.

The joint visualization framework that enables OptMap and RegSurf has the following features, which, to our knowledge, are not achieved by existing visualization techniques for regression and/or optimization:

Quality (C1) Our visualizations create images that encode information at every pixel, using a combination of dense maps, direct, and inverse projection techniques;

Genericity (C2) We can handle many kinds of high-dimensional, single-valued functions, such as the ones common in optimization or regression problems. The only requirements we have are that the user provides implementations of the objective function, constraints (if any), and, for OptMap, the range for each variable;

Simplicity (C3) Our framework is based on existing projection techniques which have a straightforward implementation, allowing easy replication and deployment;

Ease of use (C4) Our framework has few hyperparameters, all with given presets. In most cases, users do not have to adjust these to obtain good results;

Scalability (C5) Using a fast projection technique and caching results when possible, our framework is fast enough to allow its use during the rapid development-test cycle of optimization and ML models.

We structure our paper as follows. "Background" presents the used notations and discusses related work in visualization for multivariate functions, optimization, and ML regression. "Method" details our framework. "Results" presents applications and results of both OptMap and RegSurf on a number of non-trivial problems. "Discussion" discusses our proposal. "Conclusion" concludes the paper.

Background

Related work concerns optimization techniques ("Optimization"), regression problems in machine learning ("Regression in Machine Learning"), and visualization of high-dimensional objects ("Visualization of High-Dimensional Objects").

Optimization

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function to be minimized. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be an n-dimensional vector of n of the so-called decision variables x_i, $1 \leq i \leq n$, i.e., the input variables of the function f. An optimization problem O involving the function f can be described as

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{x}) \\ &\text{subject to} \quad \mathbf{x} \in S. \end{aligned} \tag{1}$$

In other words, solving the problem O means finding specific points $\mathbf{x} \in \mathbb{R}^n$ where f has a minimal value and which also obey the constraints defined by the set S. This set is also called the *feasible set* of all points that can be considered as valid for O. When such a set is provided, it *constraints* the range of inputs of f over which one searches for a minimum—that is, one should not search for minimum points for f outside the set S (see more details below).

Optimization problems come in many forms with respect to the kind of function to be optimized, type of decision variables, and presence of constraints:

- *Function type* Functions f are typically grouped into linear, convex, and non-convex. Linear functions are of the form $f(\mathbf{x}) = \mathbf{a}\mathbf{x} + b$, which defines a hyperplane. Convex functions have many forms, and can be defined as those where the points above their graph form a convex set.

Non-convex functions are neither convex nor linear (linear functions are also convex).
- *Decision variables*: These can be discrete ($x_i \in \mathbb{Z}$) or continuous ($x_i \in \mathbb{R}$). Problems with only discrete variables are called Integer Programs (IP) [8]. Problems with both discrete and continuous variables are called Mixed Integer Programs (MIP).
- *Constraints* For unconstrained problems, the feasible set S is $\mathbb{R}^n$. Constrained problems have a set of K constraint functions $c_k(\mathbf{x}) \in \{0,1\}, 1 \leq k \leq K$, where 0 means that the point $\mathbf{x}$ is infeasible with respect to constraint c_k. For constrained problems, the feasible set is defined as $S = \{\mathbf{x} : \prod c_k(\mathbf{x}) = 1, 1 \leq k \leq K$. Constraints can be characterized just as functions (linear, convex, and non-convex). Additionally, we have box constraints, which are simple restrictions on the variables' domains. Problems with continuous variables, linear objective functions, and linear constraints are called Linear Programs (LP). Other problems are solved by Non-linear Programming (NLP) techniques.

Solvers are algorithms that find one of several (approximate) solutions to a problem O. To do this efficiently, solvers use the type of decision variables, objective function, and constraints to pick adequate heuristics that avoid exploring all possible $\mathbf{x} \in S$, which is impractical in most cases. A very popular solver algorithm for linear problems is Simplex [9, 10], implemented by software such as Clp [11], Cbc [12], and GLPK [13]. For non-linear problems, there are other algorithms, such as Gradient Descent, Nelder–Mead [14], and L-BFGS [15], to name a few. Many solvers work iteratively, i.e., start from a given point $\mathbf{x}_0$ and evolve this point until sufficiently close to the solution of O.

A *solution* is an n-dimensional point $\mathbf{x}_{sol}$ found by the solver which is both feasible ($\mathbf{x}_{sol} \in S$) and *optimal*. The definition of optimality depends on the type of problem and solver used: For linear functions with linear constraints, solvers are guaranteed to find a *global optimum* solution $\mathbf{x}_{sol}$, which means that no other n-dimensional point $\mathbf{x} \neq \mathbf{x}_{sol}$ yields a lower value for the objective function f, given those constraints. For non-linear functions, solvers may return different *local optima*, depending on the starting point $\mathbf{x}_0$ used and the shape of the objective function.

Finally, solvers may provide a *trace*, or *path to solution*, which is the set $T = (\mathbf{x}_0, \dots, \mathbf{x}_k)$ of all k n-dimensional points where they evaluated the objective f, starting from $\mathbf{x}_0$ and ending with the solution $\mathbf{x}_k = \mathbf{x}_{sol}$, if one was found, else ending with the last point $\mathbf{x}_k$ evaluated by the solver.

Real-world optimization problems have many variables and constraints. Users typically rely on numerical analysis to understand if a problem is modeled correctly and if the results make sense. Visualization greatly expands the possibilities of model analysis and debugging, giving a quick way to check, e.g., if constraints are correctly defined, i.e., not under- or over-constraining by mistake; or, for NLP problems, to check how stable are the optima found, i.e., how close they are to peaks or troughs in the data.

Regression in Machine Learning

Regression is one of the two most common problems in ML, alongside classification. A regression model can be seen as a function

$$\rho : \mathbb{R}^n \to \mathbb{R}^p \tag{2}$$

used to predict a continuous response variable. In contrast, in classification problems, the response variable is categorical. That is, a classifier can be seen as a function

$$\kappa : \mathbb{R}^n \to C, \tag{3}$$

where C is a categorical dataset containing the values of the class labels to be inferred from the data.

Regressors and classifiers share many similarities in ML. Both functions ρ and κ are typically created by a training algorithm, based on a training dataset D_T that samples several independent variables and the response variable, the latter being the regressed value ρ or the inferred class label κ. Also, both functions are tested on a test set constructed much in the same way as the training set. Ideally, both functions should reach the response variables recorded by the test set. Many of the standard ML algorithms used in classification, e.g., Decision Trees [16], Random Forests [17], and Gradient Boosting [18], can be used for regression as well. This is so since classifiers use internally a regressor to compute the confidence of a sample being of a certain class and next output the final class labels my maximizing this confidence over all available classes. Examples of classifiers and regressors abound in the ML literature, such as separating malignant from benign images of various tissues in medical science (classification) and image-to-image transformation and text translation in natural language processing (regression).

However, similar regressors and classifiers also have differences. From a goal perspective, a regressor seeks to *interpolate* the sampled values y_i of the response variable between points $\mathbf{x}_i$ in its training set $D_T = \{(\mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}^p)\}$. In contrast, a classifier seeks to *separate* points $\mathbf{x}_i$ in its training set $D_T = (\mathbf{x}_i \in \mathbb{R}^n, y_i \in C)$ according to their class y_i. From a practical perspective, understanding the operation of a classifier typically focuses on showing which aspects of the input data space determine the assignment of a given *class*. This can be done in terms of highlighting the importance of all the n input variables of the classifier [19]; using locally interpretable models [20]; or visually, in terms of depicting the decision zones that the classifier partitions its input space $\mathbb{R}^n$ into [21]. Recently, Garcia et al. [22] presented a survey of visual techniques used for explaining deep learning classification models.

From a practical perspective, understanding a regressor is more complicated than understanding a classifier. Even if we consider regressors that output a single response variable ($p = 1$), this response variable is *continuous* rather than discrete as in the case of classifiers. As such, understanding a regressor implies explaining how the output *varies*, e.g., in terms of variation speed, local minima and maxima, and smoothness of variation, with respect to its input. In contrast, explaining a classifier is easier, as it involves only telling which ranges of the input variables determine the output of one of the (few) class labels. In other words, the explanation of a classifier consists of a small set of *discrete* objects (the class labels), whereas the explanation of a regressor must show the *continuous* mapping between the input and output variables. As a consequence, many methods for visually explaining classifiers exist (as outlined above); in contrast, far fewer methods exist for explaining regressors.

Visualization of High-Dimensional Objects

Visualization of high-dimensional *datasets* is an active topic for several decades, with many types of methods being proposed [6, 23] and analyzed via several quality metrics [24]. Our scope is narrower but, importantly, also broader—we are interested in visualizing multidimensional *functions*. We outline the differences between the two types of visualization next.

Visualizing datasets Visualizing high-dimensional *data* addresses, in the most general case, the task of depicting a dataset $D = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ of N data points or samples where each sample $\mathbf{x}_i \in \mathbb{R}^n$ is an n-dimensional measurement. The main challenge here is fitting a large number of samples (size N of D) of many dimensions n in the limited, low-dimensional, visualization space given by a 2D screen. Several classes of methods offer various trade-offs for this, e.g., table lenses [25, 26], parallel coordinate plots [27], and scatterplot matrices [28] (all three classes address large N, small n), and dimensionality reduction (handle large N, large n, discussed below in more detail). A particular case occurs when the D comes from the sampling of a high-dimensional function $f : \mathbb{R}^n \rightarrow K$, that is, $D = \{(\mathbf{x}_i, f(\mathbf{x}_i))\}$. In this case, several of the dimensions of a point $(\mathbf{x}_i, f(\mathbf{x}_i))$ map to the function's variables $x_1, \dots, x_n$ (also called independent dimensions), whereas the remaining so-called dependent dimensions $f(\mathbf{x}_i)$ represent the function's outputs. Furthermore, when the function's co-domain K is discrete, we have the typical case of visualizing the sampling of a classifier; when K is continuous, we have the case of visualizing the sampling of a regressor. In all cases, key tasks related to visualizing high-dimensional datasets are finding groups of similar samples and dependency patterns between the independent and/or dependent variables.

Visualizing functions Visualizing high-dimensional *functions* generalizes the above-mentioned visualizations of datasets. The key difference here is that the visualization's input is the entire *space* defined by the function's variables, i.e., $\mathbb{R}^n$ or a subset thereof, rather than the *sampling* of this space that a dataset D captures. This is a far more challenging task, since the visualization has to somehow depict the entire dense hypercube in $\mathbb{R}^n$ spanned by the respective variables rather than the sparse point cloud consisting of samples in D. Also, function visualization aims to address additional questions not present for datasets, e.g., finding directions of maximal (or minimal) change or points where the function is locally maximal (or minimal).

The visualization of 2D functions $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ is usually done by means of 3D height plots, contour plots, or color (heatmap) plots. For functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with more than two variables ($n > 2$), there are far fewer options, with Hyperslice [29] being a notable one. Hyperslice presents a multidimensional function as a matrix of orthogonal two-dimensional slices, similar in design to scatterplot matrices [28], each showing the restriction of f to one of the 2D subspaces in $\mathbb{R}^n$, using the 2D function plotting outlined earlier (contour plots, color plots, and 3D height plots). Hyperslice-related approaches have also been used to visualize subspaces of regression models in machine learning contexts [30]. Other methods for visualizing high-dimensional functions aim to reduce the data dimensionality by techniques such as isosurfacing [31] or projections [32]. All these techniques essentially only visualize a *subset* of the entire space of the function's input variables.

Visualizing constrained optimization problems is similar to the above, since not only the function has to be visualized but constraint feasibility as well. Most techniques used for this are based on overlaying contour plots with constraint information, with one case where image-based techniques are used [33]. Still, such techniques cannot work with more than two dimensions ($n > 2$). In our work, we remove all above limitations. RegSurf, proposed in this paper, aims to visualize functions for any n; OptMap does the same for optimization problems and also integrates the visualization of constraints.

Dimensionality reduction (DR) is an area of research concerned with representation of high-dimensional data by a low number of dimensions, enabling different tasks to be performed on the data, such as visual exploration [34]. For a dataset $D = \{\mathbf{x}_i\}, 1 \leq i \leq N$ of N points $\mathbf{x}_i$ with n dimensions each, a dimensionality reduction, or projection, technique is a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q,$$

where $q \ll n$, and typically $q = 2$. The projection $P(\mathbf{x})$ of a sample $\mathbf{x} \in D$ is a point $\mathbf{p} \in \mathbb{R}^q$. Projecting a set D yields thus a qD scatter plot, denoted next as $P(D)$. The inverse of P, denoted $P^{-1}(\mathbf{p})$, maps a point in $\mathbb{R}^q$ to a high-dimensional point $\mathbf{x} \in \mathbb{R}^n$, aiming to satisfy that $P^{-1}(P(\mathbf{x})) = \mathbf{x}$.

Many types of DR methods exist, such as the well-known Principal Component Analysis [35] (PCA) technique, Manifold Learners, Spring Embedders, and Stochastic Neighborhood Embedding (SNE) techniques, among others. Manifold Learners, such as MDS [36], Isomap [37], LLE [38], and more recently UMAP [39], try to reproduce in 2D the high-dimensional manifold on which data are embedded, to capture non-linear structure in the data. Spring embedders, also called force-directed techniques, such as LAMP [40] and LSP [41], have a long history in visualization, with uses in dimensionality reduction but also in graph drawing. The SNE family of methods appeared in the 2000s, and has t-SNE [42] as its most popular member. SNE-class methods produce visualizations with good cluster separation. For extensive reviews of DR methods, and their quality features, we refer to [34, 43]. However, in most cases, DR methods are used to visualize high-dimensional datasets and not high-dimensional functions.

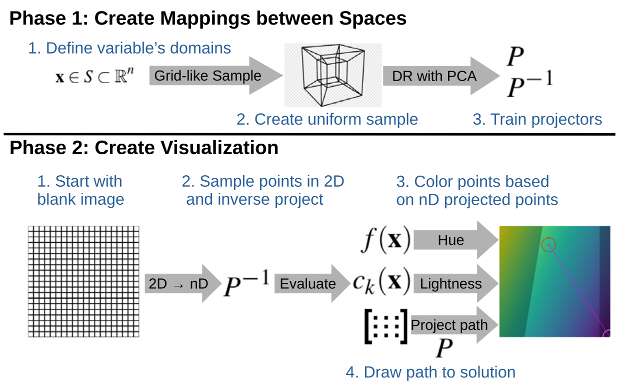
The authors of iLAMP [44] used direct and inverse projection techniques applied to non-linear optimization problems to help users interactively identify good starting points for optimization problems. Yet, iLAMP is computationally expensive, and has several free parameters the user needs to set. The NNInv method [45] performs inverse projections two orders of magnitude faster than iLAMP by deep learning the inverse projection function P^{-1}. A similar deep learning idea was used to accelerate the direct projection P by Neural Network Projections (NNP) [46]. Recently, NNInv was used by an image-based (dense map) technique to visualize the decision boundaries for Machine Learning classifiers [21] for problems with arbitrary dimension n. This can be seen as visualizing a function $f : \mathbb{R}^n \rightarrow C$, where f is a classifier for nD data and C is a class (label) set. In our work, we also use a dense pixel map as visualization model. However, as explained earlier, our aim is to understand the behavior of optimizers (by OptMap) and regressors (by RegSurf), i.e., of continuous real-valued functions, rather than that of discrete-valued classifiers.

Method

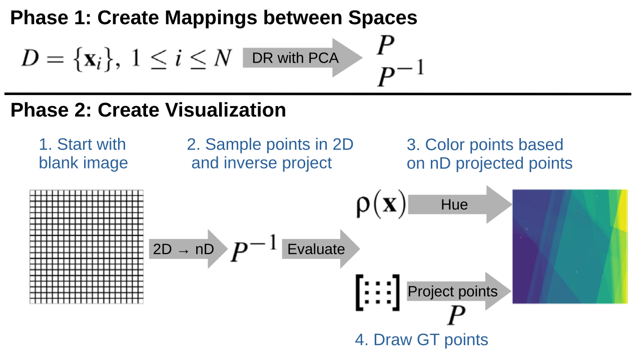
Recently, we proposed a technique called OptMap [7] to visualize high-dimensional continuous functions in the context of optimization problems. OptMap aims to visualize any type of multivariate, single-output function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Its input is a specification of the explicit definition of the function f plus the *domain* of all its input variables $x_1, \ldots, x_n$. In our current work, we propose a new technique RegSurf, which extends OptMap to address ML regression. The key difference between RegSurf and OptMap is as follows: While OptMap requires the user to explicitly specify the expression of the function f to be visualized, RegSurf *constructs* this function f by training an ML regressor from a given set of sample points and next maps f's n-dimensional input space, and the function's continuous values, to a 2D visualization.

Figure 1 shows high-level pipelines of both OptMap and RegSurf. Both methods have the following main workflow (with method-specific steps indicated where present):

1. Define variable ranges (OptMap only) The user specifies the domain $X_i = [x_i^{\min}, x_i^{\max}]$ of each variable x_i of $f(x_1, \ldots, x_n)$. When X_i is the entire real axis $\mathbb{R}$, we select a finite subset thereof to avoid too coarse sampling for x_i in the next step.


Phase 1: Create Mappings between Spaces
1. Define variable's domains
$\mathbf{x} \in S \subset \mathbb{R}^n$
Grid-like Sample
DR with PCA
P
P^{-1}
2. Create uniform sample
3. Train projectors
Phase 2: Create Visualization
1. Start with blank image
2. Sample points in 2D and inverse project
3. Color points based on nD projected points
2D → nD
P^{-1}
Evaluate
$f(\mathbf{x})$
Hue
$c_k(\mathbf{x})$
Lightness
Project path
P
4. Draw path to solution

(a) OptMap visualization


Phase 1: Create Mappings between Spaces
$D = \{\mathbf{x}_i\},\ 1 \le i \le N$
DR with PCA
P
P^{-1}
Phase 2: Create Visualization
1. Start with blank image
2. Sample points in 2D and inverse project
3. Color points based on nD projected points
2D → nD
P^{-1}
Evaluate
$\rho(\mathbf{x})$
Hue
Project points
P
4. Draw GT points

(b) RegSurf visualization

Fig. 1 Pipelines for visualization of optimization (**a**) and regression (**b**) problems

2. Sample data (OptMap only) We uniformly sample the ranges X_i defined above, yielding a regular sample grid $G^n \subset \mathbb{R}^n$. We constrain the maximum number of sample points $N_{\max}$ in G^n to avoid combinatorial explosion. In this paper, we used $N_{\max} = 5$M for all experiments. We evaluate f on G^n and call the resulting dataset $D = \{(\mathbf{x}, f(\mathbf{x})) | \mathbf{x} \in G^n\}$.

3. Create mappings We use PCA [35] trained on D to create the mappings P and P^{-1} from $\mathbb{R}^n$ to $\mathbb{R}^q$, and from $\mathbb{R}^q$ to $\mathbb{R}^n$, respectively. For OptMap, the dataset D used in this process is generated as described in step 2 above. For RegSurf, D is the dataset used for training and testing a desired regressor function ρ, as typically done in ML.

4. Create image We create a pixel image $G^2 \subset \mathbb{R}^2$ of some fixed user-chosen resolution (set to 800^2 for all experiments in this paper). Next, we use the trained P^{-1} to map each pixel $\mathbf{p} \in G^2$ to a high-dimensional point $\mathbf{x} = P^{-1}(\mathbf{p}), \mathbf{x} \in \mathbb{R}^n$. Finally, we evaluate the target function to determine the color and luminance of the pixels $\mathbf{p}$. For OptMap, this is the objective function $f(\mathbf{x})$ and optional constraints. For RegSurf, this is the regressor $\rho(\mathbf{x})$ we trained (see step 3 above). Let $v(\mathbf{p})$ denote next the value of f (for OptMap), respectively, of ρ (for RegSurf) at point $\mathbf{x} = P^{-1}(\mathbf{p})$ for pixel $\mathbf{p}$.

5. Color pixels We color all pixels $\mathbf{p} \in G^2$ by the values of $v(\mathbf{p})$ using a continuous color map. For OptMap, we also set the luminance of $\mathbf{p}$ to reflect $f(P^{-1}(\mathbf{p}))$'s membership of the constraint-set S, to indicate constraint feasibility. If the colormap is not isoluminant, as for the Viridis colormap [47] we use in this paper, the luminance of $\mathbf{p}$ actually encodes both f and the constraints. If desired, one can easily use other—more (perceptually) isoluminant colormaps. We leave the question of what the optimal colormap is open as part of future work.

6. Draw path to solution (OptMap only) If the solver provides the trace T to a solution (see "Optimization"), we draw it atop of the 2D image. In detail, we project all the data points of the trace $T = (\mathbf{x}_0, \dots, \mathbf{x}_k)$ and obtain the 2D points $(P(\mathbf{x}_0), \dots, P(\mathbf{x}_k))$. Next, we connect these points by line segments and draw the resulting polyline. As iterative solvers typically take small steps, consecutive solver points in T are very close to each other, so the same will be true for their 2D projections. Hence, using linear interpolation (line segments) to connect these points in the image plane is a good approximation of the projection of the trace.

7. Draw ground truth (RegSurf only) We draw the training set $D_T = \{(\mathbf{x}_i, y_i)\}$ used earlier to train our regressor ρ (see step 3) over the 2D image. Each point $\mathbf{x}_i$ is projected at pixel $P(\mathbf{x}_i)$ and colored by to the absolute error $|y_i - \rho(\mathbf{x}_i)|$ to show the quality of the surface generated by the regressor ρ.

Results

We next present several experiments that show how our techniques perform in different scenarios. First, we show how OptMap can be used to visualize high-dimensional functions that have a *known* shape ("OptMap: Test via High-Dimensional Functions"). As we know the ground truth (i.e., function shape), we can easily tell whether OptMap is working as intended. Next, we test OptMap on several unconstrained and constrained optimization problems ("OptMap: Solvers for Unconstrained Problems" and "OptMap: Constrained Problems", respectively) and show the added value OptMap provides for these use cases. "OptMap Performance" presents a performance evaluation of OptMap. Similarly, we demonstrate RegSurf for the visualization of several regression algorithms using real-world datasets ("RegSurf: Real-World Datasets"). We also select a specific set of datasets and regressors to show the usage of RegSurf to visualize overfitting ("RegSurf: Visualizing Overfitting"). Finally, "RegSurf Performance" explores the scalability of RegSurf.

OptMap: Test via High-Dimensional Functions

To test OptMap, we use the six functions f in Table 1. Figure 2 shows the corresponding six dense maps, computed as explained in "Discussion". In all cases, the domain used for all variables was $x_i \in [-5, 5]$. All these functions have a predictable shape and also generalize to many dimensions. We created dense maps using increasing numbers of dimensions $n \in \{2, 3, 5, 7, 10, 20\}$. The dense map for $n = 2$ was created for reference only, without using OptMap. Indeed, for $n = 2$, we can directly visualize f, e.g., by color coding, similar to [29]. Showing these maps for $n = 2$ is, however, very useful. Indeed, (1) for $n = 2$, we can show f *directly*, without any approximation implied by OptMap; and (2) given the functions' expressions (Table 1), we know that they behave similarly regardless of n. Hence, if, for $n > 2$, OptMap produces images similar to the ground-truth ones for $n = 2$, we know that OptMap works well. And indeed, Fig. 2 shows us exactly this—the OptMap images for $n > 2$ are very similar to the ground-truth ones for $n = 2$. The differences imply some distortion and rotations, which, we argue, are expected and reasonably small, given the inherent information loss when mapping a nD phenomenon to 2D.

OptMap: Solvers for Unconstrained Problems

We next use OptMap to show how different *solvers* perform with different unconstrained problems (that is, variants of Eq. 1). For this, we select a subset of the functions in Table 1, namely Styblinski–Tang, Rastrigin and Sphere functions, with varying dimensionality n. We use the solvers listed in Table 2, grouped there by solver type, namely whether it is gradient-free or if it requires a gradient or a Hessian. Figure 3 uses OptMap to show the *trace* provided by each solver, i.e., all the points evaluated by the solver to get to the solution (see "Optimization"). We see that, for the

Table 1 Definition of n-dimensional selected functions for ground-truth testing

Function name	Definition
Linear	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i$
Sphere	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$
Rosenbrock [48]	$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$
Step	$f(\mathbf{x}) = \begin{cases} 0 & \sum_{i=1}^{n} x_i < 0 \\ 2 & \sum_{i=1}^{n} x_i < 2 \\ 4 & \sum_{i=1}^{n} x_i < 4 \\ 5 & otherwise \end{cases}$
Rastrigin [49]	$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$ where: $A = 10$
Styblinski-Tang [50]	$f(\mathbf{x}) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}$

simple Sphere function with a global optimum, most solvers find an optimal solution, except for the gradient-free methods, which seem to struggle with the high-dimensionality of the problem ($n = 20$). For the Styblinski–Tang function, we see different but close optima were found by most solvers. We also see that both gradient-free methods evaluated many more points than the other methods, but that Nelder–Mead kept moving in the right direction. For the same problem, Simulated Annealing had problems converging to an optimal solution and eventually gave up. For the Rastrigin function, which has many optima, we see that only Gradient Descent and L-BFGS could find the solution in a straightforward way; the other solvers converged to the wrong solution or did not converge.

OptMap: Constrained Problems

We next show how our OptMap performs when dealing with constrained optimization problems—that is, finding the minimum of some n-dimensional function f whose variables are constrained as described in "Optimization". Table 3 shows the definition of constrained problems (objective functions

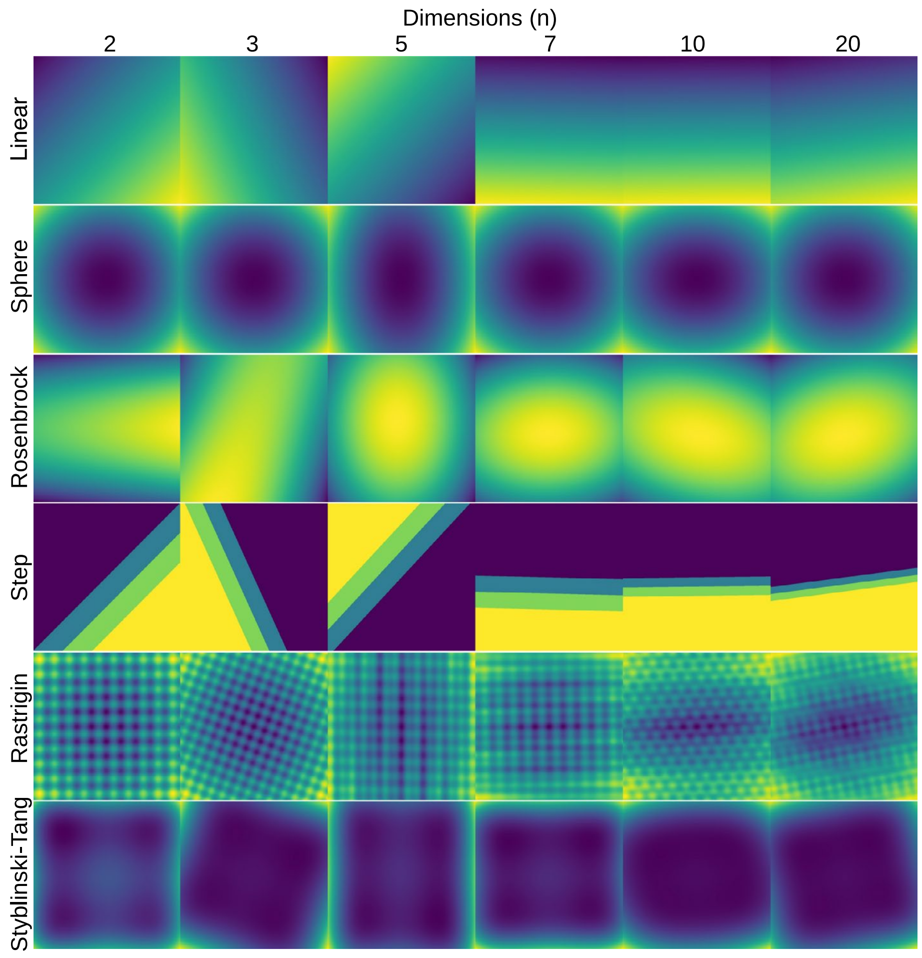


Fig. 2 Dense maps for functions with known shape as defined in Table 1, with increasing dimensionality $n > 2$. Compare these with the ground-truth maps for $n = 2$

Table 2 Solvers used for unconstrained problems

Solver type	Solver
Gradient-free	Nelder–Mead [14]
	Simulated annealing
Gradient required	Gradient descent
	Conjugated gradient [51]
	L-BFGS [15]
Hessian required	Newton

and constraints) we used. The first three problems used are very common in the optimization literature [8]. The last two problems use the same Sphere and Styblinski–Tang functions defined earlier, but with non-linear constraints added to them. Figure 4 shows how OptMap visualizes the problem space and solution for each problem. Unfortunately, the solvers used in this experiment, namely, Clp [11], Cbc [12], GLPK [13], and Ipopt [52], do not provide trace information to be drawn through the algebraic modeling language

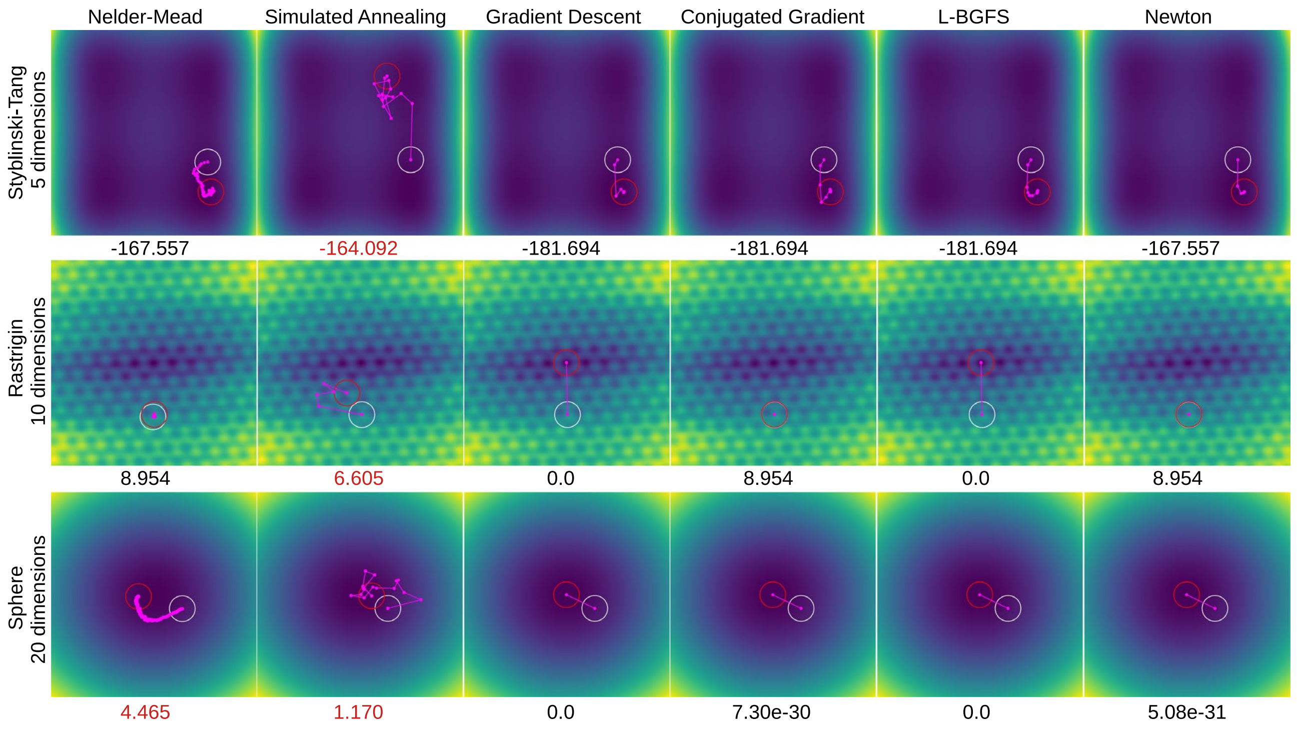


Fig. 3 OptMap dense maps for unconstrained problems (from Table 1) using the solvers in Table 2. White circles indicate starting points (random vectors in 5, 10, and 20 dimensions, respectively). Red circles indicate optimal points found by the solver. The magenta lines and points show the solver traces. The numbers below each image indicate the value of the objective function at the solution; red values indicate that the solver failed to find an optimal solution (converge), so the value is the one the solver stopped at before aborting

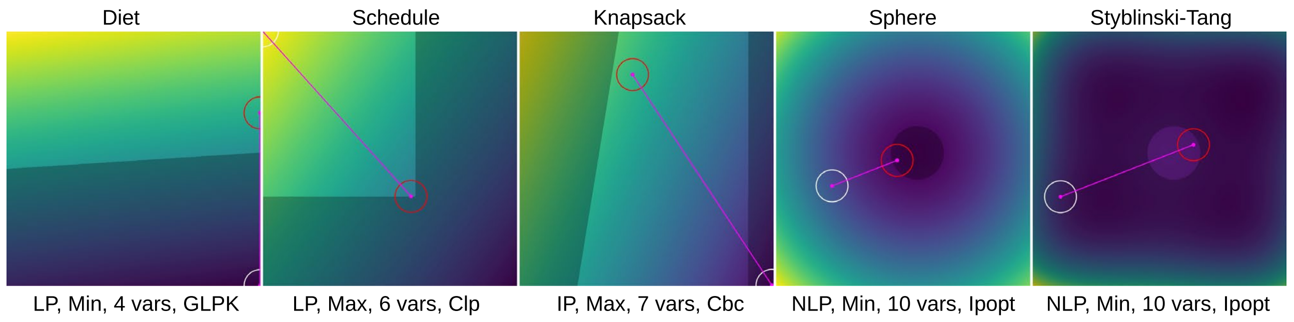


Fig. 4 OptMap dense maps for the constrained problems in Table 3. White circles indicate starting points (zero vector). Red circles indicate optimal points found by the solver. Magenta lines show the solver traces. Darker areas indicate unfeasible regions. Text below each image tells the type of problem, direction (minimization or maximization), number of variables, and solver used

Table 3 Definition of constrained optimization problems

Name	Definition
Diet	Minimize $0.14x_1 + 0.4x_2 + 0.3x_3 + 0.75x_4$ Subject to $23x_1 + 171x_2 + 65x_3 + x_4 \geq 2000.0$ $0.1x_1 + 0.2x_2 + 9.3x_4 \geq 30.0$ $0.6x_1 + 3.7x_2 + 2.2x_3 + 7x_4 \geq 200.0$ $6x_1 + 30x_2 + 13x_3 + 5x_4 \geq 250.0$ $x_1, x_2, x_3, x_4 \geq 0.0$
Schedule	Maximize $300x_1 + 260x_2 + 220x_3 + 180x_4 - 8y_1 - 6y_2$ Subject to $11x_1 + 7x_2 + 6x_3 + 5x_4 \leq 700.0$ $4x_1 + 6x_2 + 5x_3 + 4x_4 \leq 500.0$ $8x_1 + 5x_2 + 5x_3 + 6x_4 - y_1 \leq 0.0$ $7x_1 + 8x_2 + 7x_3 + 4x_4 - y_2 \leq 0.0$ $y_1 \leq 600.0$ $y_2 \leq 650.0$ $x_1, x_2, x_3, x_4, y_1, y_2 \geq 0.0$
Knapsack	Maximize $60x_1 + 70x_2 + 40x_3 + 70x_4 + 20x_5 + 90x_6$ Subject to $x_1 + x_2 - 4y \geq 0.0$ $x_5 + x_6 + 4y \geq 4.0$ $30x_1 + 20x_2 + 30x_3 + 90x_4 + 30x_5 + 70x_6 \leq 2000.0$ $x_3 - 10x_4 \leq 0.0$ $x_1, x_2, x_3, x_4, x_5, x_6, y \geq 0.0$ $x_1, x_2, x_3, x_4, x_5, x_6 \leq 10.0$ $y \leq 1.0$ $x_1, x_2, x_3, x_4, x_5, x_6, y \in \mathbb{Z}$
Sphere	Minimize $\sum_{i=1}^{10} x_i^2$ Subject to $\sum_{i=1}^{10} x_i^2 \geq 5.0$
Styblinski–Tang	Minimize $\frac{\sum_{i=1}^{10} x_i^4 - 16x_i^2 + 5x_i}{2}$ Subject to $\sum_{i=1}^{10} x_i^2 \geq 5.0$

we used, JuMP [5], so we only draw the straight-line path from the (randomly chosen) starting point to solution.

In Fig. 4, we can see for all problems the relationship between the objective function and the constraints of the problem, which provides insight on how close to boundary conditions the solutions are. For example, in the problems Schedule, Sphere, and Styblinski–Tang, we see that the solution found is at the boundary of one or more constraints. This is not the case for the Diet and Knapsack problem, which indicates that some tuning to the solver's settings may be required to obtain better results, or even some adjustments to the problem definition may be done, such as the relaxation of some constraints.

OptMap Performance

OptMap's effort can be divided into two phases (Fig. 1a): In phase 1, most of the time is spent while running PCA for the sample points in the grid G^n to define the mapping between the nD and 2D spaces. This has to be done only once for a given function *f* and can be reused, e.g., when one changes the solver. In phase 2, most time is spent evaluating the objective function *f* and its constraints. Since function evaluation is usually very fast and the pixel grid G^2 is of limited size (800^2 in our experiments), phase 2 takes only a few seconds to run on our platform. Table 4 shows the PCA time in phase 1 for $N_{\max} = 5$M points. Additional implementation and evaluation details are given in Appendix 1. We see in Table 4 that time increases quickly with dimensionality. Yet, since phase 1 is required to be run only once, and since this time is a few minutes only even for a high dimensionality n, we argue that this is not a crucial limitation of OptMap.

Table 4 Time to project $N_{\max} = 5$M points with different dimensionalities n using PCA

Dimensions n	Time (s)
3	1.19
5	0.85
7	1.45
10	2.38
20	6.62
50	32.74
100	108.71

RegSurf: Real-World Datasets

We use RegSurf to visualize the behavior of several regression algorithms, namely, Linear Regression, Decision Trees, Random Forests, Gradient Boosting, k-Nearest Neighbors (kNN), Support Vector Machines (SVM with RBF kernel), and Neural Networks. We selected these regression algorithms based on popularity, availability, and replicability (in terms of stable, documented, open-source implementations), and also on the fact that they use very different approaches to the regression problem, and thus generate different types of surfaces. Also, since the behavior of these regression algorithms is relatively well understood, we can check that the RegSurf visualization works well.

Table 5 lists the datasets used in this evaluation. All these datasets are well known in both classification and regression problems, come from different problems (thus, describe phenomena of different complexity, patterns, data structure), are publicly available (which favors replicability), and cover a range of dimensionalities. All datasets were split 80/20% into training and test sets for our experiments. As visible, the dimensionality of the datasets in Table 5 is relatively low when compared to datasets usually present in deep learning, e.g., MNIST or similar. Technically speaking, RegSurf can handle far higher dimensional datasets directly (see the algorithm steps in "Method". However, as explained there, we use PCA to map the n-dimensional space to the 2D one (projection P) and conversely (inverse projection P^{-1}. PCA can do this reasonably well for low-intrinsic-dimensional datasets. However, high-dimensional datasets in general also have a higher intrinsic dimensionality, which causes PCA to produce poor mappings for P and/or P^{-1}. As such, we limit our experiments to datasets having a lower dimensionality. We detail this point further in "Discussion". A second point is the difficulty of finding high-dimensional *regressor* datasets, i.e., datasets which associate a real-value measurement (ground truth) to every n-dimensional sample point. Many high-dimensional datasets in deep learning, such as MNIST and similar, provide only discrete *class* values, so they cannot be used to test RegSurf.

Figures 5 and 6 show surface maps created using RegSurf for the above-mentioned regression algorithms and datasets. These visualizations can be intuitively interpreted as follows: Consider the training-set $D_T \subset \mathbb{R}^n$ of each of these regressors. For every point $\mathbf{x}_i \in D_T$, we also have a target value $y_i \in \mathbb{R}$. A regression algorithm effectively constructs a function $\rho : \mathbb{R}^n \rightarrow \mathbb{R}$ that aims to ideally reproduce $\rho(\mathbf{x}_i) = y_i$ for all training-set points. Imagine now a 2D surface, embedded in $\mathbb{R}^n$, that passes through all the points $\mathbf{x}_i$. RegSurf effectively takes this surface, flattens it to the 2D image plane, and colors it by the values of ρ at all surface points. Table 6 shows the respective training and testing loss for each dataset and regressor. Let us next interpret these results step-by-step.

Regressor patterns The maps in Fig. 5 suggestively capture the different characteristics of each type of regressor. Linear regressors produce hyperplanes in $\mathbb{R}^{n+1}$, which are shown by the smooth color gradients in the RegSurf maps. Tree-based regressors, such as decision trees, random forests, and gradient boosting, show different kinds of step functions (polygon-like surfaces having near constant color visible in the maps), which form very simple to very complex. Nearest-neighbor regressors produce maps with Voronoi-style cells, which intuitively shows that these regressors, indeed, construct n-dimensional Voronoi partitions of the variable space to compute their output variable. Finally, both SVM and Neural Networks produce smoother, continuous-like, maps. This is in accordance with what we theoretically know about the respective regressors. Hence, we conclude that the RegSurf visualizations accurately capture the behavior of these regressors.

Fit-to-data analysis An additional value of the RegSurf maps is that they convey a global insight on how the studied regressors work and on how good is their fitting to the data, beyond simple aggregated error metrics. Consider the Air Quality dataset. This dataset consists of three distinct sample

Table 5 Datasets used in the evaluation of RegSurf

Dataset	Observations N	Dimensions n	Description
Air quality [53]	9358	13	Measurements from air sensors used to study and predict air quality
MTCars [54]	32	10	Data from several aspects of automobile design, used to predict fuel consumption
Housing prices [55]	506	13	Housing data from the Boston, MA region, used to predict house prices
Concrete [56]	1030	8	Measurements of chemico-physical properties of concrete used to study concrete strength
Superconductors [57]	21,264	81	Measurements of different properties from superconductors used to predict critical temperature
SP urban traffic [58]	135	18	Measurements of urban traffic information used to predict traffic flow
Wine quality [59]	6497	11	Samples of white and red Portuguese *vinho verde* used to describe perceived wine quality

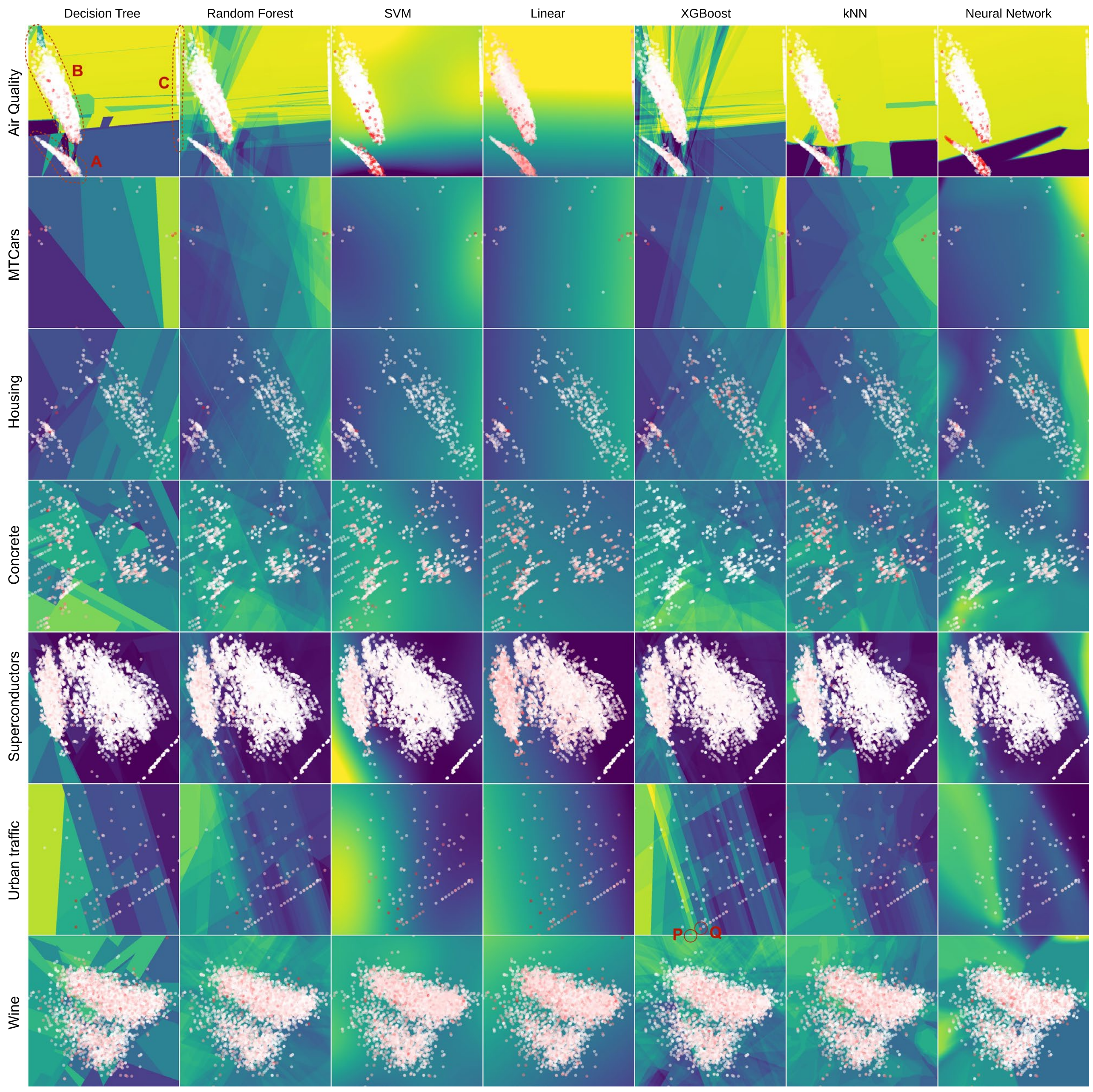


Fig. 5 Maps created with RegSurf for several datasets and regressors. Points represent the training set and are colored according to the absolute error when compared to the generated surface, varying from white (low error) to red (high error)

clusters, with one of them being far away from the other two in the projected space (see markers A, B, C for dotted lines in Fig. 5 top-left cell). We see that the regressors have more complex surfaces in the areas that have a larger concentration of points. As one gets further away from these areas, i.e., if we look at image pixels far away from the red or white dots, the surfaces get simpler to the point of becoming constant. This shows that regressors for this dataset probably will not be able to generalize well for new observations that fall within such sparsely sampled, underrepresented, regions. Second, when looking at the amount of red points (highest approximation error of the regressor as compared to the ground-truth value), we can quickly tell that XGBoost seems to have the best fit to the training data, and that it generated a very complex regression surface to do so. This is confirmed by Table 6 that shows that XGBoost has the lowest training loss for Air Quality from all seven evaluated regressors. Conversely, we see that SVM, Neural Network, and Linear regressors seem to be underfit, as shown by the amount of red points (high errors) they generate and the

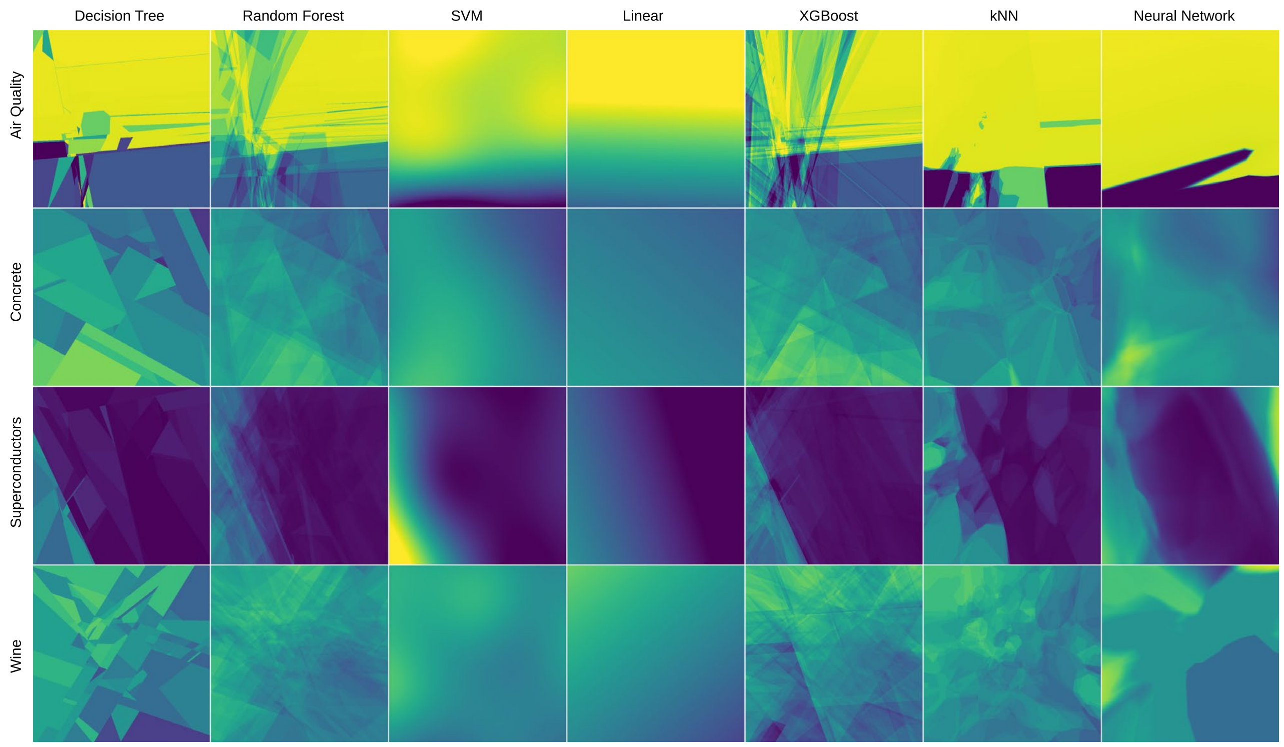


Fig. 6 Maps created with RegSurf for selected datasets shown in Fig. 5. We omitted here plotting the training points to better show the regressor's surfaces

Table 6 Training and testing loss for each regressor/dataset in Fig. 5

Regressor dataset	Decision tree		Random forest		SVM		Linear		XGBoost		kNN		Neural network	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Air quality	0.054	0.108	0.059	0.116	0.091	0.088	0.167	0.165	0.048	0.125	0.079	0.105	0.098	0.098
Concrete	0.036	0.066	0.028	0.057	0.050	0.060	0.104	0.104	0.005	0.035	0.068	0.095	0.020	0.040
Housing	0.034	0.063	0.029	0.056	0.038	0.046	0.069	0.077	0.001	0.048	0.058	0.074	0.017	0.049
MTCars	0.065	0.072	0.047	0.078	0.055	0.126	0.082	0.108	0.001	0.089	0.102	0.099	0.011	0.110
Urban traffic	0.082	0.105	0.060	0.095	0.082	0.140	0.101	0.109	0.017	0.096	0.096	0.137	0.025	0.116
Supercond	0.019	0.034	0.018	0.030	0.044	0.044	0.074	0.072	0.020	0.031	0.026	0.032	0.034	0.038
Wine	0.048	0.095	0.040	0.077	0.086	0.091	0.095	0.096	0.032	0.077	0.070	0.089	0.062	0.077

seemingly simple surfaces they created. Without the help of a technique like RegSurf, it is hard to tell how the dataset is spread out in space, and how different regression algorithms spread their errors along that space.

Selecting a good regressor Consider next the Urban Traffic dataset. Table 6 shows that both XGBoost and Neural Network are the best-fit regressors, with a slight edge for XGBoost. However, the RegSurf maps refine and actually nuance this insight. Looking at the XGBoost and Neural Network maps in Fig. 5, we see that Neural Network spreads its errors more evenly, without a specific region with high error. This is not the case for XGBoost, where we can see two dark red dots in the bottom of the image (markers P, Q in Fig. 5). Also, the checkered pattern in the XGBoost map shows that this regressor is far less smooth than Neural Networks. Hence, combining the measured values of the training loss with the visual inspection of the RegSurf maps allows the ML engineer to select the better regressor based on more desirable properties (error spread, smoothness) than if one would study only the aggregated absolute errors.

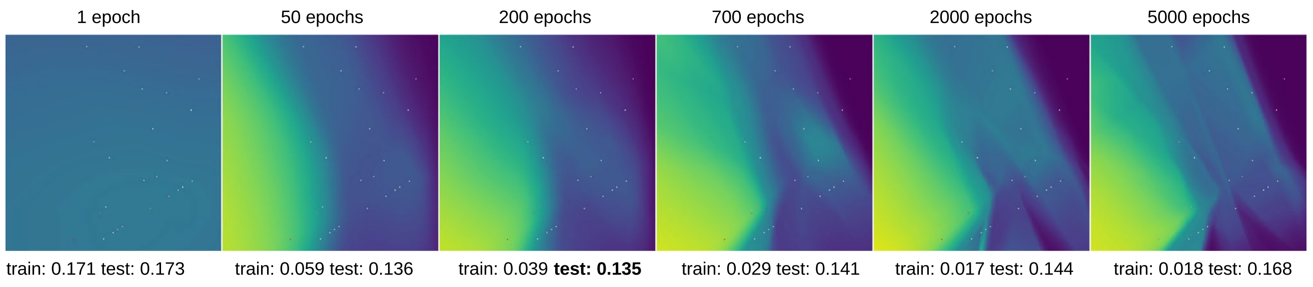


Fig. 7 RegSurf maps for Neural Network trained on the Urban Traffic dataset, increasing number of epochs. Numbers below the images show train and test loss; the minimal loss value is in bold

RegSurf: Visualizing Overfitting

As already indicated in "RegSurf: Real-World Datasets", a practical use case of RegSurf is to visually assess the *goodness of fit* of a regressor to a dataset. This complements information provided by metrics such as mean squared error or mean absolute error. We next select two pairs of (dataset, regressor) to further illustrate this use case. Figure 7 shows a Neural Network regressor trained on the Urban Traffic dataset for increasingly more training epochs. We see that, as the network starts to overfit (700 epochs), the RegSurf maps get increasingly complex, which corresponds to what is intuitively expected of an overfit model. For the second use-case, Fig. 8 shows Decision Tree regressors trained on the MTCars dataset, with varied hyperparameters. The RegSurf maps vary from very simple (small amount of space partitions), indicating underfitting, to very complex (higher amount of space partitions), indicating overfitting. Hence, the visual complexity of the RegSurf maps can be used to detect when overfitting occurs: Take a given problem where one has (rough) knowledge of the smoothness and complexity of the regressor one tries to learn. One can watch how RegSurf's maps change during training and assess, based on how these match the expected nature of the modeled phenomenon, whether, when, and where under- or overfitting occurs.

RegSurf Performance

As for OptMap, RegSurf's computation time can be divided into two phases (Fig. 1b): In phase 1, most of the time is spent while running PCA for the dataset D to construct the mapping between the nD and 2D spaces. These computations are only dataset-dependent and can be reused when, e.g., exploring different regressors. In phase 2, most of the time is spent evaluating the regressor function ρ. As for OptMap's evaluation of the objective function f, this operation is fast (see "OptMap Performance").

To assess RegSurf's scalability, we generated synthetic Gaussian data with varying number of samples and dimensions, and a Decision Tree regressor trained on these different synthetic datasets. Table 7 shows separately the time spent for the PCA mappings (phase 1) and for evaluating the regressor (phase 2). Even for very high-dimensional data, the whole process runs in tens of seconds, with the regressor-evaluation time dominating the process. Importantly, note that the regressor-evaluation time cannot be reduced—unless someone provides a faster implementation of the function ρ. Figure 9 depicts the same

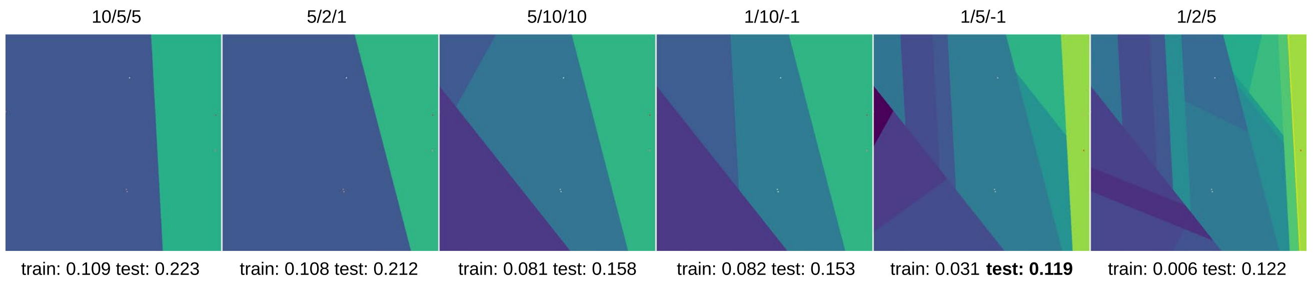


Fig. 8 RegSurf maps for Decision Trees trained on the MTCars dataset. Numbers above the images indicate the hyperparameters min samples leaf, min samples split and max depth, respectively, with − 1 meaning unlimited max depth. Numbers below the images show train and test loss, with the lower loss value shown in bold

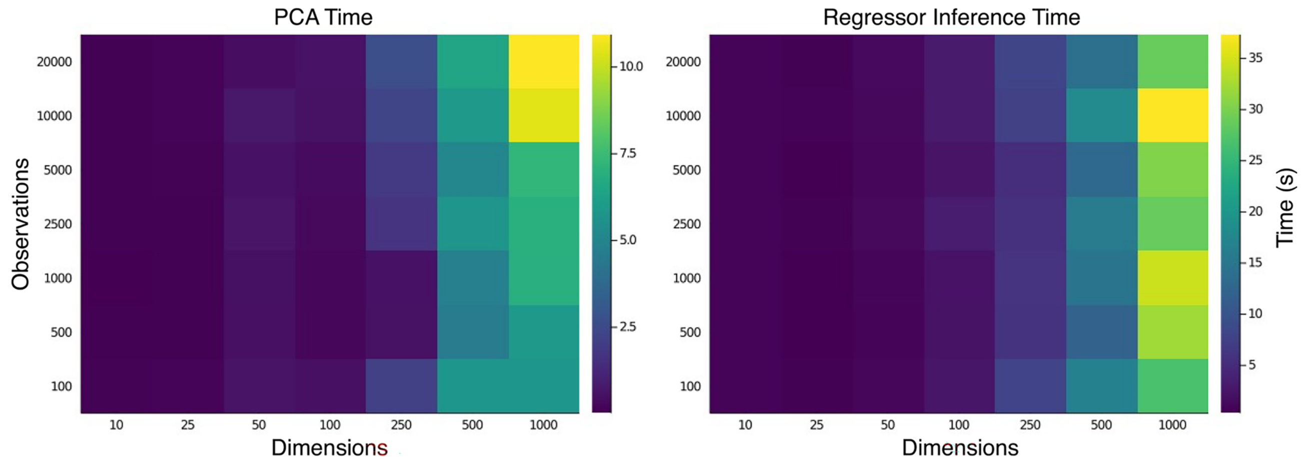


Fig. 9 Heatmap showing the order of growth of time to create surface maps using RegSurf for different dataset sizes

Table 7 Time to create RegSurf maps for different dataset sizes

Dims points	10		25		50		100		250		500		1000	
	P	*ρ*	*P*	*ρ*	*P*	*ρ*	*P*	*ρ*	*P*	*ρ*	*P*	*ρ*	*P*	*ρ*
100	0.09	0.70	0.11	0.41	0.57	0.91	0.47	2.40	2.10	7.76	5.74	16.53	5.74	26.80
500	0.04	0.60	0.04	0.33	0.47	0.69	0.20	2.05	0.50	5.76	4.55	11.87	5.88	32.12
1000	0.02	0.62	0.04	0.33	0.47	0.73	0.21	2.11	0.53	5.82	4.75	14.48	6.89	34.49
2500	0.05	0.69	0.05	0.47	0.61	1.14	0.25	3.21	1.66	5.42	5.71	15.52	6.93	28.75
5000	0.04	0.62	0.09	0.36	0.53	0.91	0.32	2.16	1.83	5.14	5.06	12.81	7.24	30.40
10,000	0.05	0.63	0.12	0.48	0.70	1.02	0.50	3.01	2.24	7.44	5.90	17.88	10.46	37.33
20,000	0.06	0.69	0.13	0.40	0.37	1.17	0.56	2.93	2.66	7.94	6.45	13.82	10.94	28.78

P is the time spent to create PCA mappings; *ρ* is the time spent by the regressor to color all pixels in the image

information as in Table 7 using heatmaps, which shows the above-mentioned trends more clearly. In this experiment, PCA time grows with the number of observations and dimensions, whereas dimensionality seems to be much more important for the Decision Tree regressor used. In the case of lazy regressors, such as k-Nearest Neighbors, we expect this behavior to be even worse. Conversely, for GPU-based Neural Network regressors, we expect this behavior not to be an issue. Given these results, we state that RegSurf is fast enough to be used during iterative ML model development that involves datasets of tens of thousands of observations and up to thousand dimensions.

Discussion

We discuss next how our joint OptMap-RegSurf technique performs with respect to the criteria laid out in "Introduction".

Quality (C1) Figs. 2, 3, and 4 show examples of the quality of the visualizations and the kind of insight they can provide for optimization problems. The same is true for regression problems with Figs. 5, 6, 7, and 8. Our dense maps are pixel-accurate, in the sense that they show *actual* information inferred from the nD function f or ρ under investigation at each pixel, *without* interpolation. This is in contrast with many other dimensionality reduction methods which either show a *sparse* sampling of the nD space (by means of a 2D scatterplot), leaving the user to guess what happens between scatterplot points; or use interpolation in the 2D *image* space to 'fill' such gaps [60–63], which creates smooth images that may communicate wrong insights, since one does not know whether the used projection is continuous.

Genericity (C2) We show how our technique performs for optimization problems with varying nature, complexity, and dimensionality, and for seven types of ML regressors over several real-world datasets. We also show that our method can be used simply for visualizing high-dimensional, continuous, functions by a *single* 2D image, in contrast to multiple images that have to be navigated and correlated by interaction [29]. We also show that our technique is independent of the optimization solvers or regressors being used. For optimization, OptMap only requires the definition of the problem O to solve, range of the input variables to consider, and access to a black-box optimization method. For regression, RegSurf only requires access to a training set and to the black-box regressor trained on it.

Simplicity (C3) We use PCA for direct and inverse projections, which is a very well-known, simple, fast, and deterministic projection method. The complete implementation of each technique has about 250 lines of Julia code. Note that we also experimented with other methods for the direct projection—namely, t-SNE as learned by NNP [46]—and inverse projection—namely, NNInv [45], obtaining good results. However, for the optimization and regression problems presented in this paper, PCA yielded better results (based on ground truth comparison). Since PCA is also simpler and faster than NNP and NNInv, we preferred it in our work.

Ease of use (C4) For OptMap, where sampling is an important step, we executed all experiments using the same maximum number of sample points $N_{\max}$ with good results, which shows that the technique requires little-to-no tuning to work properly. There are also a few options that control visual settings, such as color map and point size, which have sensible defaults and do not affect the quality or performance of the method.

Scalability (C5) “OptMap Performance” and “RegSurf Performance” show that our method is highly scalable with the number of sample points N and dimensions n, which enables its interactive usage during the development cycle of optimization and regression models. For OptMap, scalability is inherently limited by the resolution used to create the dense grid G^n—see “Method”, algorithm step 2. If the number of dimensions n *and* the sampling rate of each dimension become too high, the total number of samples N in the grid G^n becomes prohibitive. To alleviate this, one could (a) consider different sampling rates for the n dimensions, based on prior knowledge on how f depends on each of them; (b) use OptMap interactively by ‘zooming in and out’ of different variable ranges to explore the high-dimensional space; or (c) use multiresolution techniques, akin to those already present in various optimizers. This is not an issue for RegSurf, which does not use the grid G^n (“Discussion”, algorithm step 2) but rather the training and test sets used to construct the regressor ρ to be visualized. In typical ML problems, such training sets have tens of thousands of samples—in any case, far fewer than the size of G^n needed for OptMap. However, even with these limitations, both OptMap and RegSurf create dense maps of any optimizer and regressor, respectively, in tens of seconds for datasets up to 1000 dimensions, on a commodity PC, and both have a complexity linear in the number of evaluated sample points. As such, we deem both methods practical for visualization applications.

Limitations The projected data points, such as the starting, trace, and solution points for OptMap, and training-set points for RegSurf, are placed in the 2D image space at *approximate* positions, due to the inherent discrete nature of the pixel grid G^2. This can cause situations such as the one in Fig. 4 (sphere problem), where the optimal point found by the solver—which is obviously feasible—is placed slightly inside the unfeasible region, which can be misleading. Second, we noticed that due to the inherently imperfect mapping between nD and 2D spaces, equality constraints that compare against constants might not be satisfied during the evaluation, which will make the drawing of feasible regions fail in OptMap.

A separate aspect relates to the fact that P can map multiple different points $\mathbf{x} \in D$ to the same pixel $\mathbf{p} \in G^2$. Hence, the color assigned to $\mathbf{p}$ should ideally reflect the *combination* of values $f(\mathbf{x})$ of all these points $\mathbf{x}$. For categorical-valued functions f, this can be done using voting schemes that compute the confidence of the final coloring [21]. A low-hanging fruit for future work is to (efficiently) extend such schemes to our real-valued functions f using aggregation strategies such as average, min, or max.

Finally, it is known that PCA, used by both OptMap and RegSurf to bidirectionally map between the high-dimensional space and the 2D image, can create distortions when mapping between these two spaces [34]. In turn, these can influence the insights that our dense maps convey, e.g., in terms of observing non-smooth regions that actually correspond to projection errors and not optimizer or regressor non-smoothness. Finding a replacement projection for PCA (and its inverse) that is equally fast, easy to use, generic, parametric, and yields less distortion, is an important direction to explore next.

Applications The aim of the current paper was to introduce two new techniques, OptMap and RegSurf, for the visualization of high-dimensional functions for optimization and regression problems, respectively. To demonstrate these techniques, we have chosen *known* optimizers, regressors, and datasets. Indeed, this was needed, since we need to have functions with known behavior as ground-truth to check the correctness of our novel visualization techniques. As our experiments shown good results for OptMap and RegSurf for these problems, we deem both techniques to be ready for applications on problems with more complex and/or unknown behavior. Several such application areas exist. *Image analysis* by deep learning methods is a particularly interesting application area. The key reason hereof is that images, represented by high-dimensional features extracted by various methods, can be *directly* visualized, unlike other more abstract high-dimensional data. As such, one can imagine simple enhancements of RegSurf—trained for either an image classification or image regression problem—which allow the user to point at every location in the visualization space and generate on-the-fly the image corresponding to the respective high-dimensional point. This would assist users in multiple scenarios, e.g., finding what types of images correspond to low-confidence classification areas or generating new images in such areas in a user-supervised data augmentation process to improve classifier performance. Examples of recent imaging applications which could be assisted by such visualizations include transfer learning for image classification [64], finding relevant features for classification of histopathology images [65], analysis of misclassification results in cell image classification [66], microorganism image segmentation [67], and data augmentation for cancer image classification [68]. Moreover, pipelines in such applications which use *image transformation* modules can be studied separately by RegSurf, given that such transformations are essentially regressors.

Conclusion

In this paper, we presented an image-based visualization technique that enables the visualization of multidimensional, single-valued functions. Our technique represents the high-dimensional domain of the function by means of sampling, either on a provided grid, as in the case of ML training datasets, or computed dense-sampling grid, as in the case of general-purpose functions with given variable domains. This sampling is used to produce a bidirectional mapping from the high-dimensional space to 2D image space. Finally, the image pixels are colored to construct a dense representation of the function of interest.

We proposed two variants of this technique, namely OptMap, for optimization problems, and RegSurf, for regression problems. We show that both variants perform well in varied, realistic scenarios, with several examples that demonstrate the genericity, scalability, parameter-free nature, and simplicity of the underlying technique. Additionally, we showed how the produced maps can be interpreted to gain insights into underlying problems in the two domains, such as comparing solvers for optimization problems and analyzing the smoothness and under-or-overfit properties of ML regressors. Our entire framework is simple to implement and based on open-source components, which favors replicability and ease of use.

Several future work directions exist. First and foremost, it is interesting to consider using more accurate direct and inverse projections for constructing the dense maps. Second, we consider using both variants in concrete applications, and gauging its added value in helping engineers designing better optimization and regression models, as opposed to the existing tools-of-the-trade for the same task. Additionally, we plan to extend this idea to enable the visualization of vector (multivalued) functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, which will expand the range of possible applications for the technique.

Appendix 1: Implementation details

We implemented OptMap and RegSurf in Julia [69] using the open-source software libraries in Table 8. The optimization examples ("OptMap: Test via High-Dimensional Functions", OptMap: Solvers for Unconstrained Problems and "OptMap Performance") were implemented using Optim [70] for the unconstrained problems, and JuMP [5] for the constrained problems, using the solvers Clp [11], Cbc [12], GLPK [13], and Ipopt [52]. The regression examples ("RegSurf: Real-World Datasets" and "RegSurf: Visualizing Overfitting") were implemented using the MLJ library [71]. Our implementation, plus all code used in our experiments, are publicly available at github.com/mespadoto/optmap.

The scalability experiments discussed in "OptMap Performance" and "RegSurf Performance" were executed,

Table 8 Software used for the OptMap and RegSurf implementation

Library	Software publicly available at
Images	github.com/JuliaImages/Images.jl
ColorTypes	github.com/JuliaGraphics/ColorTypes.jl
ColorSchemes	github.com/JuliaGraphics/ColorSchemes.jl
Luxor	github.com/JuliaGraphics/Luxor.jl
CSV	github.com/JuliaData/CSV.jl
DataFrames	github.com/JuliaData/DataFrames.jl
MultivariateStats	github.com/JuliaStats/MultivariateStats.jl
Optim	github.com/JuliaNLSolvers/Optim.jl
Clp	github.com/jump-dev/Clp.jl
Cbc	github.com/jump-dev/Cbc.jl
GLPK	github.com/jump-dev/GLPK.jl
Ipopt	github.com/jump-dev/Ipopt.jl
MLJ	github.com/alan-turing-institute/MLJ.jl
DecisionTree	github.com/bensadeghi/DecisionTree.jl
XGBoost	github.com/dmlc/XGBoost.jl
LIBSVM	github.com/JuliaML/LIBSVM.jl
NearestNeighborModels	github.com/JuliaAI/NearestNeighborModels.jl
NeuralNetworkRegressor	github.com/FluxML/MLJFlux.jl

respectively, on a 4-core Intel Xeon E3-1240 v6 at 3.7 GHz with 64 GB RAM, and on a dual 16-core Intel Xeon Silver 4216 at 2.1 GHz with 256 GB RAM.

Author Contributions Not applicable.

Funding This study was financed in part by FAPESP under Grant Nos. 2015/22308-2, 2017/25835-9, and 2020/13275-1, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001.

Availability of Data and Materials Not applicable.

Code Availability Our implementation, plus all codes used in our experiments, are publicly available at github.com/mespadoto/optmap.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

1. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: machine learning in python. JMLR. 2011;12:2825–30.
2. Krizhevsky A, Sutskever I, Hinton G. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems (NIPS). 2012. p. 1097–1105.
3. Brooke A, Kendrick D, Meeraus A, Raman R, America U. The general algebraic modeling system. GAMS Development Corporation. 1998. p. 1050.
4. Fourer R, Gay DM, Kernighan BW. A modeling language for mathematical programming. Thomson: AMPL; 2003.
5. Dunning I, Huchette J, Lubin M. JuMP: a modeling language for mathematical optimization. SIAM Rev. 2017;59(2):295–320.
6. Liu S, Maljovec D, Wang B, Bremer P-T, Pascucci V. Visualizing high-dimensional data: advances in the past decade. IEEE TVCG. 2015;23(3):1249–68.
7. Espadoto M, Rodrigues FCM, Hirata NS, Telea AC. OptMap: using dense maps for visualizing multidimensional optimization problems. In: VISIGRAPP (3: IVAPP). 2021. p. 123–132.
8. Guenin B, Könemann J, Tuncel L. A gentle introduction to optimization. UK: Cambridge University Press; 2014.
9. Dantzig GB. Origins of the simplex method. In: A history of scientific computing. 1990. p. 141–151.
10. Kantorovich LV. Mathematical methods of organizing and planning production. Manage Sci. 1960;6(4):366–422.
11. Forrest J, Vigerske S, Ralphs T, Hafer L, jpfasano Santos HG, Saltzman M, h-i-gassmann Kristjansson B, King A. coin-or/Clp 2020.
12. Forrest J, Vigerske S, Santos HG, Ralphs T, Hafer L, Kristjansson B, jpfasano Straver E, Lubin M, rlougee jpgoncal1 h-i-gassmann, Saltzman M. coin-or/Cbc 2020.
13. Makhorin A. GLPK: GNU Linear Programming Kit. 2008. https://www.gnu.org/software/glpk/glpk.html.
14. Nelder JA, Mead R. A simplex method for function minimization. Comput J. 1965;7(4):308–13.
15. Liu DC, Nocedal J. On the limited memory BFGS method for large scale optimization. Math Program. 1989;45(1–3):503–28.
16. Breiman L, Friedman JH, Olshen RA, Stone CJ. Classification and regression trees. USA: Routledge; 2017.
17. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.
18. Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat. 2001;1189–1232.
19. Lundberg SM, Lee S-I. A unified approach to interpreting model predictions. In: Proceedings of the 31st international conference on neural information processing systems. 2017. p. 4768–4777.
20. Ribeiro MT, Singh S, Guestrin C. Why should I trust you?: explaining the predictions of any classifier. In: Proc. ACM SIGMOD KDD. 2016. p. 1135–1144.
21. Rodrigues F, Espadoto M, Hirata R, Telea AC. Constructing and visualizing high-quality classifier decision boundary maps. Information. 2019;10(9):280.
22. Garcia R, Telea A, da Silva B, Torresen J, Comba J. A task-and-technique centered survey on visual analytics for deep learning model engineering. Comput Gr. 2018;77:30–49.
23. Buja A, Cook D, Swayne DF. Interactive high-dimensional data visualization. J Comput Gr Stat. 1996;5(1):78–99.
24. Bertini E, Tatu A, Keim D. Quality metrics in high-dimensional data visualization: an overview and systematization. IEEE TVCG. 2011;17(12):2203–12.
25. Rao R, Card SK. The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In: Proc. ACM SIGCHI. 1994. p. 318–322.
26. Telea AC. Combining extended table lens and treemap techniques for visualizing tabular data. In: Proc. EuroVis. 2006. p. 120–127.
27. Inselberg A, Dimsdale B. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In: Proc. IEEE visualization. 1990. p. 361–378.
28. Yates A, Webb A, Sharpnack M, Chamberlin H, Huang K, Machiraju R. Visualizing multidimensional data with glyph SPLOMs. Comput Gr Forum. 2014;33(3):301–10.
29. van Wijk JJ, van Liere R. Hyperslice. In: Proc. visualization. IEEE. 1993. p. 119–125.
30. Piringer H, Berger W, Krasser J. HyperMoVal: interactive visual validation of regression models for real-time simulation. Comput Gr Forum. 2010;29(10):983–92.
31. Crawfis PBRWR. Isosurfacing in higher dimensions. In: Proc. IEEE visualization. 2010.
32. Gerber S, Bremer P-T, Pascucci V, Whitaker R. Visual exploration of high dimensional scalar functions. IEEE TVCG. 2010;16(6):1271–80.
33. Wicklin R. Visualize the feasible region for a constrained optimization. SAS. 2018.
34. Espadoto M, Martins RM, Kerren A, Hirata NS, Telea AC. Towards a quantitative survey of dimension reduction techniques. IEEE TVCG. 2019;27(3):2153–73.
35. Jolliffe IT. Principal component analysis and factor analysis. In: Principal component analysis. Berlin: Springer. 1986. p. 115–128.
36. Torgerson WS. Theory and methods of scaling. Oxford: Wiley; 1958.

37. Tenenbaum JB, Silva VD, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science. 2000;290(5500):2319–23.
38. Roweis ST, Saul LLK. Nonlinear dimensionality reduction by locally linear embedding. Science. 2000;290(5500):2323–6.
39. McInnes L, Healy J. UMAP: uniform manifold approximation and projection for dimension reduction (2018). arXiv:1802.03426v1 [stat.ML].
40. Joia P, Coimbra D, Cuminato JA, Paulovich FV, Nonato LG. Local affine multidimensional projection. IEEE TVCG. 2011;17(12):2563–71.
41. Paulovich FV, Nonato LG, Minghim R, Levkowitz H. Least square projection: a fast high-precision multidimensional projection technique and its application to document mapping. IEEE TVCG. 2008;14(3):564–75.
42. Maaten LVD, Hinton G. Visualizing data using t-SNE. JMLR. 2008;9:2579–605.
43. Nonato L, Aupetit M. Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment. IEEE TVCG. 2018.
44. Amorim E, Brazil EV, Daniels J, Joia P, Nonato LG, Sousa MC. iLAMP: exploring high-dimensional spacing through backward multidimensional projection. In: Proc. IEEE VAST. 2012. p. 53–62.
45. Espadoto M, Rodrigues FCM, Hirata NST, Hirata Jr, R, Telea AC. Deep learning inverse multidimensional projections. In: Proc. EuroVA. 2019.
46. Espadoto M, Hirata N, Telea A. Deep learning multidimensional projections. Inf Vis. 2020.
47. Hunter JD. Matplotlib: a 2d graphics environment. Comput Sci Eng. 2007;9(3):90–5.
48. Rosenbrock H. An automatic method for finding the greatest or least value of a function. Comput J. 1960;3(3):175–84.
49. Rastrigin LA. Systems of extremal control. Nauka. 1974.
50. Styblinski M, Tang T-S. Experiments in nonconvex optimization: stochastic approximation with function smoothing and simulated annealing. Neural Netw. 1990;3(4):467–83.
51. Hager WW, Zhang H. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. ACM Trans Math Softw. 2006;32(1):113–37.
52. Wächter A, Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math Program. 2006;106(1):25–57.
53. Vito SD, Massera E, Piga M, Martinotto L, Francia GD. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. Sens Actuators B Chem. 2008;129(2):750–757. https://archive.ics.uci.edu/ml/datasets/Air+Quality
54. Henderson HV, Velleman PF. Building multiple regression models interactively. Biometrics. 1981;391–411.
55. Harrison D Jr, Rubinfeld DL. Hedonic housing prices and the demand for clean air. J Environ Econ Manag. 1978;5(1):81–102.
56. Yeh I-C. Modeling of strength of high-performance concrete using artificial neural networks. Cem Concr Res. 1998;28(12):1797–808.
57. Hamidieh K. A data-driven statistical model for predicting the critical temperature of a superconductor. Comput Mater Sci. 2018;154:346–54.
58. Ferreira R, Affonso C, Sassi R. Combination of artificial intelligence techniques for prediction the behavior of urban vehicular traffic in the city of são paulo. In: 10th Brazilian congress on computational intelligence (CBIC)-Fortaleza, Ceara, Brazil. 2011. p. 1–7.
59. Cortez P, Cerdeira A, Almeida F, Matos T, Reis J. Modeling wine preferences by data mining from physicochemical properties. Decis Support Syst. 2009;47(4):547–53.
60. Martins R, Coimbra D, Minghim R, Telea A. Visual analysis of dimensionality reduction quality for parameterized projections. Comput Gr. 2014;41:26–42.
61. Silva Rd, Rauber P, Martins R, Minghim R, Telea AC. Attribute-based visual explanation of multidimensional projections. In: Proc. EuroVA. 2015.
62. van Driel D, Zhai X, Tian Z, Telea A. Enhanced attribute-based explanations of multidimensional projections. In: Proc. EuroVA. 2020.
63. Tian Z, Zhai X, van Driel D, van Steenpaal G, Espadoto M, Telea A. Using multiple attribute-based explanations of multidimensional projections to explore high-dimensional data. Comput Gr. 2021;98:93–104.
64. Rahaman M, Li C, Yao Y, Kulwa F, Rahman MA, Wang Q, Qi S, Kong F, Zhu X, Zhao X. Identification of COVID-19 samples from chest X-ray images using deep learning: a comparison of transfer learning approaches. J Xray Sci Technol. 2020;28(5):821–39.
65. Chen H, Li C, Wang G, Li X, Rahaman M, Sun H, Hu W, Li Y, Liu W, Sun C, Ai S, Grzegorzek M. GasHis-transformer: a multi-scale visual transformer approach for gastric histopathological image detection. Pattern Recogn. 2022;130:108827.
66. Liu W, Li C, Xu N, Jiang T, Rahaman M, Sun H, Wu X, Hu W, Chen H, Sun C, Yao Y, Grzegorzek M. CVM-Cervix: a hybrid cervical Pap-smear image classification framework using CNN, visual transformer and multilayer perceptron. Pattern Recogn. 2022;130:108829.
67. Zhang J, Li C, Kosov S, Grzegorzek M, Shirahamad K, Jiang T, Sun C, Li Z, Li H. LCU-Net: a novel low-cost U-Net for environmental microorganism image segmentation. Pattern Recogn. 2021;115:107885.
68. Rahaman M, Li C, Yao Y, Kulwa F, Wu X, Li X, Wang Q. DeepCervix: a deep learning-based framework for the classification of cervical cells using hybrid deep feature fusion techniques. Comput Biol Med. 2021;136:104649.
69. Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: a fresh approach to numerical computing. SIAM Rev. 2017;59(1):65–98.
70. Mogensen PK, Riseth AN. Optim: a mathematical optimization package for Julia. J Open Sour Softw. 2018;3(24):615.
71. Blaom AD, Kiraly F, Lienart T, Simillides Y, Arenas D, Vollmer SJ. MLJ: a Julia package for composable machine learning. 2020. arXiv preprint arXiv:2007.12285.