Institute for Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Transformed Sparse Grids for High-Dimensional Models

Christopher Schnick

Course of Study:	Informatik
Examiner:	Prof. Dr. Dirk Pflüger
Supervisor:	Dr. Michael Rehme
Commenced:	April 26, 2021
Completed:	October 26, 2021

Abstract

Computer experiments provide a convenient framework for investigating real world phenomena, and making more information available to a simulation enables it to represent reality more accurately. However, many complex simulations suffer from the curse of dimensionality, which makes an accurate simulation difficult and requires the use of a model surrogate. In many cases, a simulation mainly depends only on a subset of the input parameters and is therefore a possible target for dimensionality reduction techniques. These techniques aim to identify such input parameter structures and try to adapt the simulation accordingly, for example by constructing a lower-dimensional model surrogate. Done well, this approach can ease the curse of dimensionality while still preserving a required level of simulation accuracy.

Various approaches to dimensionality reduction already exist in the context of sparse grids, such as adaptive sparse grids or sparse grid based analysis of variance, where the goal is to neglect or eliminate less important dimensions of a model function. Active subspaces are another dimensionality reduction technique, which in contrast to sparse grid based methods, can replace input parameters through linear combiniations of others, thus making them more flexible than purely dimension oriented approaches.

In this thesis, the active subspace method is applied in combination with sparse grids to obtain the name-giving transformed sparse grids and create a more flexible dimensionality reduction technique that can exploit the best of both worlds. This combined approach views the model from a black-box perspective and can therefore be employed on a wide variety of high-dimensional models. To evaluate the quality and properties of the produced transformed sparse grid surrogates in different contexts, we also perform a range of experiments on a variety of high-dimensional models that are used for real world applications.

Contents

1 Introduction

Computer simulations for complex models are in many cases very computationally expensive and are therefore limited in the feasible amount of simulation runs given the available computing power. Because of this limitation, a common approach is to construct a surrogate model and adapt the simulation to work on the surrogate instead. Done properly, this can massively reduce the required resources, such as runtime and memory, to perform one simulation run. Discretization methods [**Ste+73**] are one such way of constructing a surrogate by discretizing the continuous model domain and approximating the output while trying to keep the introduced discretization error to a minimum. Full grids, a simple discretization scheme for functions on hypercubes, suffer from the curse of dimensionality [**Bel61**], as the amount of grid points required to generate a uniform grid grows exponentially with the model dimensionality. As a result, the amount of simulation runs needed for a full grid construction also grow exponentially, severely limiting the viable model dimensionality for full grids.

While all grid-based methods share the common problem of being affected by the curse of dimensionality to some degree, the sparse grid technique [**Zen91**] manages to reduce the impact of the curse of dimensionality and therefore delay the point at which their application becomes infeasible. To decouple the model dimensionality from the actual sparse grid dimensionality, at least to some extent, various methods for the purpose of performing a dimensionality reduction on sparse grids are available to choose from. For example, there exist several kinds of adaptive sparse grids, which are able to eliminate or neglect certain dimensions of the model and are able to tailor the constructed sparse grid to the model. These grid based techniques in general are however limited in their effectiveness, as the grid structures itself, the associated basis functions, and adaptivity rules are all axis-aligned. As we will see later on, axis-aligned basis functions, and therefore all techniques that are based on them, become rather ineffective when dealing with model functions that primarily consist out of structures that are not axis-aligned.

This limitation is used as a motivation to look into ways of transforming the input parameters of the model, such that sparse grid-based methods can be applied more effectively on all kinds of models. By conducting a parameter study to identify potential starting points and exploring the influence of different input parameters on the model output, we try to construct and apply a transformation function on the model inputs prior to feeding them into sparse grids. The goal is to obtain transformed input parameters that have a better alignment and optionally a reduced dimensionality, such that we can construct a sparse grid surrogate of a lower dimensionality while managing to maintain a similar approximation error compared to standard sparse grids.

The concrete method presented in this thesis combines the process of determining so-called active subspaces in the inputs [**Con15**] with dimensionality reduction techniques. It takes inspiration from already established ideas in data science, such as the concept of intrinsic dimensionality [**Ben69**] and principal component analysis (PCA) [**AW10**]. This allows us to transform the original model approximation problem into a lower-dimensional one while still preserving input structures that are suitable for sparse grids. Over the course of this thesis, we will first introduce and explore the name-giving transformed sparse grid technique and then put it to the test in multiple experiments with high-dimensional models for real-world phenomena afterwards.

1.1 Outline

We start off in chap. 2 by covering all the fundamentals needed to understand and use sparse grids in conjunction with different kinds of basis functions and spatial adaptivity to create model surrogates. Furthermore, we will also take a look at other components commonly needed for model approximation tasks, such as sampling techniques and error metrics.

Next, we extensively introduce input transformations, which are the central focus of this thesis, in chap. 3 through a simple motivational example function that causes various established grid-based methods to underperform. We show how an input transformation can be used to fix the underlying problem and improve the approximation. Following that, we define various concrete types of input transformations with the main focus lying on transformations generated with the active subspace method.

Over the course of chap. 4, we look at the process of generating sparse grid surrogates for already transformed inputs to obtain the name-giving transformed sparse grids. We focus especially on the unique characteristics of this construction process and its resulting surrogates compared to the normal sparse grid workflow.

Chap. 5 then combines input transformations with an iterative approach to obtain the more powerful compounded transformed surrogates. We also provide a formal algorithm description that is used as the basis for the practical implementation. Moreover, some basic properties of the algorithm are also investigated.

Then, in chap. 6, we model the concrete structure of the practical implementation for the transformation process and explore various configuration options that can be used to fine tune different aspects of the algorithm.

The individual components of the implementation are evaluated in chap. 7, in which we apply the new transformation-based approach on two models using a variety of different configurations, with the goal of evaluating the implementation components in an isolated environment to prepare for the practical applications in the succeeding chapter.

The implementation is then put to the test in chap. 8, where we perform multiple experiments on practical, high-dimensional models to draw conclusions about the technique in general and about its comparative quality. To achieve the best possible results, we leverage the insights gained in preceeding chapter.

Chap. 9 concludes the thesis by providing a summary of the newly introduced technique and its results. We also reflect on the limitations and possible improvements of the method.

2 Surrogate construction

The general process of performing computer simulations on mathematical models to study their properties can be complicated by different aspects of the underlying model itself. For example, the complexity of simulations is often heavily dependent on the model dimensionality, which is referred to as the curse of dimensionality [**Bel61**]. Moreover, a simulation run can also require a lot of computational resources when for example a differential equation has to be solved. Discretization methods are one way to mitigate this problem by constructing a continuous surrogate that can fill in the gaps between a discrete set of simulation results. Other times when dealing with black box models, i. e., when the underlying model function is not known and only a fixed set of model outputs is given, we can employ regression algorithms to construct a surrogate instead.

More formally, we assume that a model is represented by a mathematical model function $f\colon \Omega \to \mathbb{R}$, where $\Omega \coloneqq [0,1]^d$ is the model domain. While the original domain of many models may not be the d-dimensional unit hypercube, it can be transformed into one as long as the original domain is a bounded Cartesian product. Furthermore, many models are also subject to uncertainties in their inputs. To incorporate these uncertainties, we model the inputs stochastically with a probability distribution defined by a probability density function $\rho\colon \Omega \to \mathbb{R}_+$. In case the model outputs are also subject to uncertainties, such as random noise or measurement errors, a well fitted surrogate should be able to average out the uncertainties and represent a smoothed version of the original model.

The usual course of action is to construct a surrogate $\hat{f}$ that approximates f and preserves an required amount of accuracy, such that the simulation results are still representative for the original model. In other words, we are trying to construct an approximating function $\hat{f}$ such that

$$\forall x \in \Omega\colon\ f(x) \approx \hat{f}(x). \tag{2.1}$$

Moreover, by constructing surrogates of a certain type, other additional useful properties can be provided at no additional cost, even if they were not available for the original model function. For example, the gradients and moments of spline-based surrogates are very easy to compute and are essentially obtained for free.

Plenty of different surrogate types and construction methods already exist and each one comes with its own specific properties and limitations. These properties can range from their sensitivity to the model dimensionality, over their approximation quality in general, to unique effects when being applied to certain types of model functions. The primary focus of this thesis lies on the already established sparse grid surrogates, their various subtypes, and construction methods, with the goal of enhancing the sparse grid technique by finding and exploiting important lower-dimensional subsets of the model input parameters.

2.1 Notation

Prior to starting out with formalizing the fundamentals of the surrogate construction process, we first have to define some basic notation for working with vectors that will be frequently used. In rare cases where a scalar and vector with the same identifier are used, for example for the level and index notation of sparse grids later on, we use an underlined identifier to differentiate a vector from a scalar. For example, the identifier ℓ would represent a scalar, whereas $\underline{\ell}$ would represent a vector. Other times, when no differentiation between scalars and vectors is necessary, no explicit underlined identifier is used for vectors. An underlined constant value $\underline{k}$ with $k \in \mathbb{R}$ is used to define a d-dimensional vector of the form $\underline{k} \coloneqq (k, \ldots, k)^T \in \mathbb{R}^d$.

Furthermore, other commonly used constructs, namely basic vector norms and vector comparison operators, are defined as follows:

Definition 2.1.1 (Vector norms)
Let $v \in \mathbb{R}^d$ be a d-dimensional vector. We make use of the ℓ_1 norm,

$$|v|_1 \coloneqq \sum_{i=1}^{d} |v_i| \,, \tag{2.2}$$

the Euclidean norm,

$$|v|_2 \coloneqq \sqrt{\sum_{i=1}^{d} v_i^2} \,, \tag{2.3}$$

and the maximum norm,

$$|v|_\infty \coloneqq \max_{1 \leq i \leq d} v_i \,. \tag{2.4}$$

Definition 2.1.2 (Vector comparison)
Let $u, v \in \mathbb{R}^d$ be d-dimensional vectors. We define the vector comparison operator,

$$u \leq v \Leftrightarrow \forall i \in \{1, \ldots, d\} \colon u_i \leq v_i, \tag{2.5}$$

as a component-wise comparison.

2.2 Full grids

Model functions can easily be discretized using a naive full grid approach, which involves creating a uniform isotropic grid that spans Ω and interpolates function values using nearby grid points. More formally, we can first construct a one-dimensional mesh of level ℓ with equidistant points $x_{\ell,i} = i2^{-\ell}$, which results in the mesh size $h = 2^{-\ell}$. Then, by associating appropriate basis functions $\phi_{\ell,i} \colon [0,1] \mapsto \mathbb{R}$ with every grid point $x_{\ell,i}$, we can interpolate functions very accurately for reasonably small mesh sizes. By using a basic tensor product approach, we can obtain a uniform isotropic d-dimensional grid mesh where the grid point indices become vectors and the basis functions $\phi_{\underline{\ell},\underline{i}} \colon [0,1]^d \mapsto \mathbb{R}$ become multivariate functions. The asymptotic L^2 interpolation error for full grids constructed this way with hat basis functions is $O(h^2)$, however this comes at the cost of requiring $O(2^{\ell d})$ grid points. Full grids therefore suffer from the curse of dimensionality, as the required amount of function evaluations for the grid points grows exponentially with the level ℓ and model dimensionality d.

Definition 2.2.1 (Levels and indices)
Let $\underline{\ell} \coloneqq (\ell_1, \ldots, \ell_d) \in \mathbb{N}_0^d$ be a d-dimensional multi-index. For every multi-index $\underline{\ell}$, we define the hierarchical index set,

$$H_{\underline{\ell}} \coloneqq \left\{ \underline{i} \in \mathbb{N}_0^d \mid \begin{cases} i_t = 1, 3 \ldots, 2^{\ell_k} - 3, 2^{\ell_k} - 1, & \ell_k \geq 1 \\ i_t = 0, 1, & \ell_k = 0 \end{cases} \right\}. \tag{2.6}$$

We use these hierarchical index sets in order to define a full grid as the sum of grid increments that are constructed for a given level multi-index $\underline{\ell}$ and its associated index set $H_{\underline{l}}$ such that every point in the grid mesh is assigned to exactly one level multi-index and hierarchical multi-index. An incremental construction like this will make it easier to transition from full grids to sparse grids later on.

Definition 2.2.2 (Grid points)
Let $x_{\ell,i} = i2^{-\ell}$ be a one-dimensional grid point coordinate with $x_{\ell,i} \in [0, 1]$, and let

$$x_{\underline{\ell},\underline{i}} \coloneqq (x_{\ell_1,i_1}, \ldots, x_{\ell_d,i_d}) \tag{2.7}$$

be a d-dimensional grid point. The set of grid points of a regular full grid with boundary is then given by

$$X_\ell^{\mathrm{fb}} \coloneqq \{x_{\underline{\ell'},\underline{i}} \mid |\underline{\ell'}|_\infty \leq \ell, \underline{i} \in H_{\underline{\ell}}\}, \tag{2.8}$$

and the set of grid points of regular full grid without boundary points by

$$X_\ell^{\mathrm{f}} \coloneqq \{x_{\underline{\ell'},\underline{i}} \mid |\underline{\ell'}|_\infty \leq \ell, \underline{1} \leq \ell', \underline{i} \in H_{\underline{\ell}}\}. \tag{2.9}$$

Next, we assign each grid point a supporting multivariate basis function that stems from one family of basis functions, for example a family of pieceiwse linear functions or polynomials.

Definition 2.2.3 (Basis functions)
Let $\phi_{\ell,i}(x)$ be a univariate basis function for the one-dimensional grid point $x_{\ell,i}$.
By applying the tensor product approach on $\phi_{\ell,i}(x)$, we obtain the multivariate $d-$dimensional basis function,

$$\phi_{\underline{\ell},\underline{i}}(x) \coloneqq \prod_{k=1}^{d} \phi_{\ell_k,i_k}(x_k), \tag{2.10}$$

for the d-dimensional grid point $x_{\underline{\ell},\underline{i}}$.

Definition 2.2.4 (Full grid)
The span of the multivariate basis functions $\phi_{\underline{\ell},\underline{i}}$ over all regular full boundary grid points,

$$V_\ell^{\mathrm{fb}} \coloneqq \operatorname{span}\{\phi_{\underline{\ell},\underline{i}} \mid x_{\underline{\ell},\underline{i}} \in X_\ell^{\mathrm{fb}}\}, \tag{2.11}$$

is defined as the regular full boundary grid space of level ℓ and the span over all regular non-boundary full grid points,

$$V_\ell^{\mathrm{f}} \coloneqq \operatorname{span}\{\phi_{\underline{\ell},\underline{i}} \mid x_{\underline{\ell},\underline{i}} \in X_\ell^{\mathrm{f}}\}, \tag{2.12}$$

is defined as the regular full grid space of level ℓ, respectively.

2.3 Sparse grids

Full grids can be decomposed into a direct sum of subspaces in a hierarchical way, which allows for a granular selection of subspaces to keep. By keeping only a specific subset of subspaces, we obtain the so-called sparse grids. Compared to full grids, a sparse grid construction with hat basis functions leads to a worse approximation quality with the asymptotic L^2 interpolation error including an additional logarithmic factor with $O(h^2(\log h^{-1})^{d-1})$. As shown in [**BG04**], the amount of grid points however only grows with $O(h^{-1}(\log h^{-1})^{d-1})$, which is a significant improvement over full grids. As a result, the reduced amount of required grid points makes it possible to apply the sparse grid technique to higher-dimensional problems than the full grid technique.

Definition 2.3.1 (Sparse grid points)
Let ℓ be a level. Using the hierarchical index sets $H_{\underline{\ell}}$, we define the grid points for a regular sparse boundary grid,

$$X_\ell^{\mathrm{sb}} \coloneqq \{x_{\underline{\ell'},\underline{i}} \mid |\underline{\ell'}|_1 \leq \ell, \underline{i} \in H_{\underline{\ell'}}\}, \tag{2.13}$$

and the grid points for a regular sparse grid,

$$X_\ell^{\mathrm{s}} \coloneqq \{x_{\underline{\ell'},\underline{i}} \mid |\underline{\ell'}|_1 \leq \ell, \underline{1} \leq \ell', \underline{i} \in H_{\underline{\ell}}\}. \tag{2.14}$$

As we can see, the sparse grid point set definition is very similar to the full grid definition (def. 2.2.4), with the only difference being the level multi-index selection using the ℓ_1 norm $|\cdot|_1$ instead of the maximum norm $|\cdot|_\infty$, effectively performing a diagonal cut on the available grid increments, as seen in fig. 2.1a. The sparse grid space can then also be defined similar to full grids:

Definition 2.3.2 (Sparse grid)
The span of the basis functions $\phi_{\underline{\ell},\underline{i}}$ over all regular sparse boundary grid points,

$$V_\ell^{\mathrm{sb}} \coloneqq \operatorname{span}\{\phi_{\underline{\ell},\underline{i}} \mid x_{\underline{\ell},\underline{i}} \in X_\ell^{\mathrm{sb}}\}, \tag{2.15}$$

is the regular sparse boundary grid space, and the span over all regular sparse non-boundary grid points,

$$V_\ell^{\mathrm{s}} \coloneqq \operatorname{span}\{\phi_{\underline{\ell},\underline{i}} \mid x_{\underline{\ell},\underline{i}} \in X_\ell^{\mathrm{s}}\}, \tag{2.16}$$

is the regular sparse grid space.

The grid construction results for full grids as well as sparse grids are visualized in fig. 2.1b and fig. 2.1c.

In related sparse grid literature, the hierarchical decomposition, which is also called hierarchical splitting, throughout which sparse grids are realized is achieved through the decomposition of subspaces instead of the decomposition of the grid point set. This allows for easier proofs of various approximation qualities but is not required for the purposes of this thesis. It is sufficient to see that the sparse grid construction used in this thesis is equivalent to the construction through hierarchical subspaces.

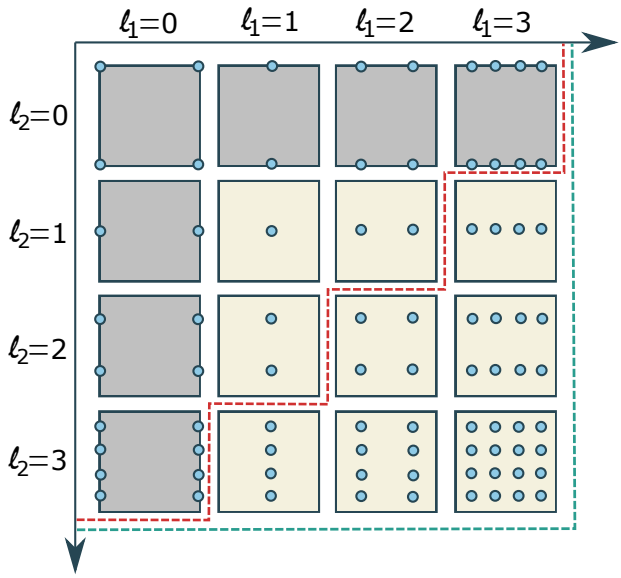
Definition 2.3.3 (Hierarchical subspaces)
The span of the basis functions $\phi_{\underline{\ell},\underline{i}}$ of a grid increment identified by a level multi-index $\underline{\ell}$,

$$W_{\underline{\ell}} \coloneqq \operatorname{span}\{\phi_{\underline{\ell},\underline{i}} \mid \underline{i} \in H_{\underline{\ell}}\}, \tag{2.17}$$

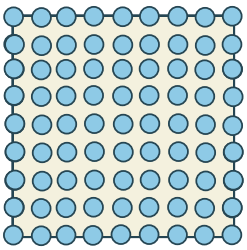
is defined as the hierarchical subspace of $\underline{\ell}$. The sparse grid space can then also be represented as a direct sum of the hierarchical subspaces with

$$V_{\underline{\ell}}^{\mathrm{s}} = \operatorname{span}\left\{\phi_{\underline{\ell},\underline{i}} \mid x_{\underline{\ell},\underline{i}} \in X_{\underline{\ell}}^{\mathrm{s}}\right\} = \bigoplus_{|\underline{\ell}'|_1 \leq \underline{\ell}} W_{\underline{\ell}'}. \tag{2.18}$$

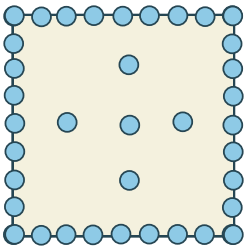
The sparse grid space construction of (2.18) is equal to (2.16) because the only difference between both is that the hierarchical decomposition is realized in different stages.



(a) Hierarchical splitting of the grid point set X_3^{fb}



(b) Grid points of X_3^{fb}



(c) Grid points of X_3^{sb}

Figure 2.1: Construction of different grids from hierarchical grid point set increments. The full boundary grid point set X_3^{fb} is a sum of all grid increments outlined in dotted green ■, whereas the sparse boundary grid point set X_3^{sb} is a sum of all grid increments outlined in dotted red ■. Boundary grid point increments are shaded in grey ■.

2.4 Basis functions

Multivariate basis functions $\phi_{\underline{\ell},\underline{i}}$ (def. 2.2.3), are a central aspect of sparse grids. Many different families of basis functions exist, each with various levels of complexity, capabilities, and properties. In this section we will take a short look at the basis functions used in this thesis.

Hat functions An elementary basis for sparse grids can be constructed with the help of hat functions, which are just piecewise linear functions. A one-dimensional hat function, $h(x) := \max(1 - |x|, 0)$, can be transformed into a basis function for a level ℓ and index i to obtain

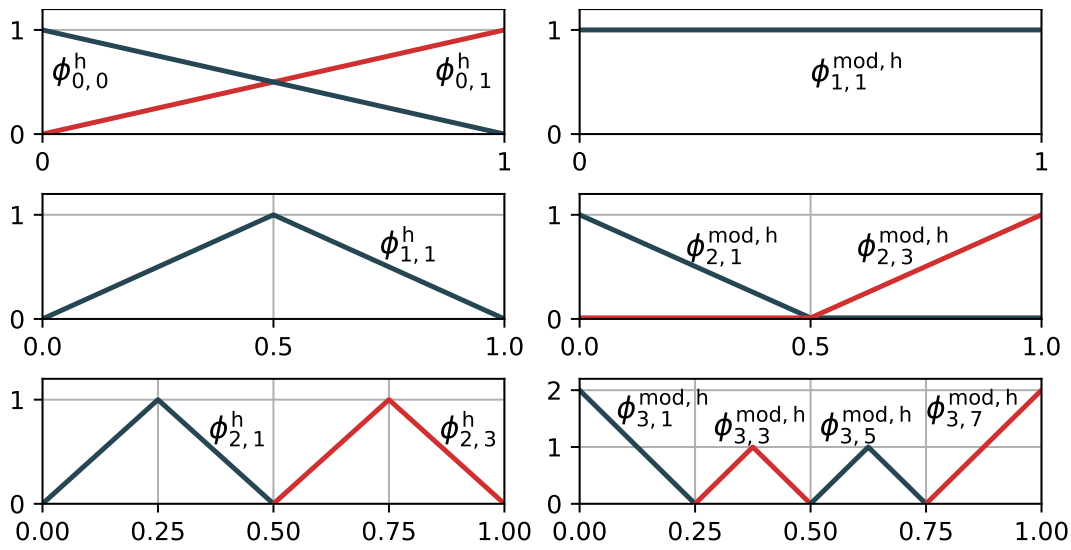
$$\phi_{\ell,i}^{\mathrm{h}}(x) := h(2^{\ell}x - i). \tag{2.19}$$

These basis functions are centered at their associated grid point $x_{\ell,i}$. Furthermore, their supports are disjoint for the same level. Fig. 2.2a shows the hat basis functions for the first few levels.

Modified basis functions One method of removing the need of using boundary grid points to accurately approximate some model functions in the boundary regions are modified basis functions [**Pfl10**]. These basis functions extrapolate the values at the boundaries from the grid points closest to the boundaries by using a modified version of normal basis functions. Extrapolation of this sort can, at least to some degree, be a viable alternative to using boundary grids as a lot of grid points would be allocated just for the boundaries by high-dimensional sparse grid surrogates. Removing all of them while still providing acceptable boundary approximations usually results in a good tradeoff between grid points and accuracy, which is the intuition behind sparse grids. More formally, it is possible to modify any non-boundary basis functions by adding a constant function to it and also defining special boundary extrapolation functions ϕ_ℓ^{left} and ϕ_ℓ^{right} that are usually are similar with regards to their construction compared to the normal basis functions $\phi_{\ell,i}$. For example, modified basis functions for the hat basis can be obtained by adding boundary functions and a constant function to the hat basis with

$$\phi_{\ell,i}^{\text{mod,h}}(x) := \begin{cases} 1, & \ell = 1 \\ \max(2 - 2^\ell x, 0), & \ell > 1, i = 1 \\ \max(2 - 2^\ell (1 - x), 0), & \ell > 1, i = 2^\ell - 1 \\ \phi_{\ell,i}^{\text{h}}(x), & \text{else} \end{cases}. \tag{2.20}$$

Fig. 2.2b shows the modified hat basis functions for the first few levels.



(a) Hat basis for $\ell = 0, \ldots, 2$.

(b) Modified hat basis for $\ell = 1, \ldots, 3$.

Figure 2.2: Hat basis functions and their modified counterparts.

Other basis functions Over time, many kinds of basis functions have been developed. Each one was created with a specific purpose in mind and has the potential to outshine other basis functions for certain applications. There exist polynomial basis functions, spline basis functions that are just piecewise polynomials, wavelet basis functions, and numerous subtypes of them. However, for the sake of simplicity, they are not covered in detail as they will not be primarily used in this thesis. Instead, we will focus on hat basis functions and modified hat basis functions.

2.5 Sparse grid surrogates

Sparse grid surrogates can be constructed using one of multiple different approaches. As a sparse grid surrogate is a linear combination of all basis functions, in order to represent the instance $\hat{f}$, each basis function $\phi_{\underline{\ell},\underline{i}}$ is assigned a weight $\alpha_{\underline{\ell},\underline{i}}$, also called hierarchical surplus.

Definition 2.5.1 (Sparse grid surrogates)
Let $\phi_{\underline{\ell},\underline{i}}$ be multivariate basis functions and X a set of grid points. Furthermore, let $\alpha_{\underline{\ell},\underline{i}} \in \mathbb{R}$ be the hierarchical surplusses. A grid surrogate is then a linear combination with

$$\hat{f}(x) \coloneqq \sum_{x_{\underline{\ell},\underline{i}} \in X} \alpha_{\underline{\ell},\underline{i}}\, \phi_{\underline{\ell},\underline{i}}(x). \tag{2.21}$$

In the case of a regular sparse grid with $X = X^{\mathrm{s}}_{\ell}$, the surrogate instance can also be expressed with

$$\hat{f}(x) = \sum_{|\underline{\ell}|_1 \leq \ell, \underline{1} \leq \ell'} \sum_{\underline{i} \in H_{\underline{\ell}}} \alpha_{\underline{\ell},\underline{i}}\, \phi_{\underline{\ell},\underline{i}}(x). \tag{2.22}$$

The hierarchical surplusses can be computed in a variety of different ways, which results in multiple possible sparse grid construction schemes. Most of the time, the interpolation approach is used. In cases where only model function observations are given, a sparse grid surrogate can also be constructed through regression

Definition 2.5.2 (Fundamental property)
A family of univariate basis functions $\phi_{\ell,i}$ fulfills the fundamental property iff

$$\begin{aligned} &\phi_{\ell,i}(x_{\ell',j}) = 0, \quad \ell' < \ell, j \in H_{\ell'}, \\ &\phi_{\ell,i}(x_{\ell',j}) = \delta_{ij}, \quad j \in H_{\ell'}. \end{aligned} \tag{2.23}$$

As the used multivariate basis functions are constructed using the tensor-product approach, the fundamental property holds for a multivariate basis iff it holds for the univariate one.

Interpolation The interpolation method is the most simple and common sparse grid surrogate construction scheme. It evaluates the model function at every grid point and performs a hierarchization algorithm to obtain the hierarchical surplusses $\alpha_{\underline{\ell},\underline{i}}$. A multivariate basis that fulfills the fundamental property can be hierarchized very efficiently through the unidirectional principle [**Bal94**] by applying a hierarchization operator iteratively on every dimension separately, effectively breaking down the hierarchization process into a series of one-dimensional hierarchizations. The resulting surrogate constructed with the hierarchization method equals the model function at all grid points, i. e., $\forall x_{\underline{\ell},\underline{i}} \in X\colon f(x_{\underline{\ell},\underline{i}}) = \hat{f}(x_{\underline{\ell},\underline{i}})$. As the model function has to be evaluated at every grid point, a grid interpolation surrogate requires exactly $|X|$ function evaluations and can not be used to approximate a function from only model function observations.

Regression An alternative surrogate construction scheme is sparse grid regression [**Pfl10**]. In contrast to interpolation, a regression approach does no longer require the function values at the grid points to interpolate between them. Instead, it uses established regression approaches to iteratively improve the weights $\alpha_{\underline{\ell},\underline{i}}$ with regards to the approximation error by using training and validation observations from the model function. As a result, the amount of required observations does no longer have to be equal to the amount of grid points. This results in a greater level of flexibility

as the amount of available inputs and the amount of grid points can vary. The basic approach is to solve the least squares problem for the observation set, $S \coloneqq \{(x_i, f(x_i)\}_{i=1}^n \subseteq \Omega \times \mathbb{R}$, with the standard square loss function,

$$\varepsilon_{\text{MSE}}(\hat{f}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \left(f(x_i) - \hat{f}(x_i) \right)^2, \tag{2.24}$$

which contains a regularization functional R and a regularization control parameter $\lambda > 0$ that allows for smoothing and can also prevent overfitting. Sparse grid regression therefore tries to solve the regularized least squares problem to find the best possible surrogate for a set of possible regularization factors $\lambda \in \Lambda$ and sparse grid level ℓ to obtain the surrogate with

$$\hat{f} = \underset{\hat{f} \in V_\ell^s,\ \lambda \in \Lambda}{\arg\min}\ \varepsilon_{\text{MSE}}(\hat{f}) + \lambda R(\hat{f}). \tag{2.25}$$

2.6 Spatially adaptive sparse grids

The process of generating a regular sparse grid can be improved upon in many cases by introducing adaptivity. This allows us to spend more grid points, and therefore be more accurate, in regions that are more important and conversely reduce the amount of grid points spent in less important regions. When done correctly, the surrogate can adapt itself to the model function characteristics and deliver a better approximation. One method of implementing adaptivity during the sparse grid construction process are spatially adaptive sparse grids [**Pfl12**]. The central idea is to define a grid point hierarchy where each non-boundary grid point has two child grid points on the next level increment along every dimension. A stepwise algorithm then adds these child grid points to the surrogate if they are needed and do not exist yet. This creation of new grid points that stem from a parent grid point is called the refinement of the parent grid point.

Definition 2.6.1 (Hierarchical grid point children)
Let X be a set of grid points, and let $x_{\underline{\ell},\underline{i}} \in X$ be a grid point. We define

$$c_k^{\text{left}}(x_{\underline{\ell},\underline{i}}) \coloneqq (x_{\ell_1,i_1}, \ldots, x_{\ell_k+1,2i_k-1} \ldots, x_{\ell_d,i_d}) \tag{2.26}$$

as its left child along the k-th dimension if $i_k > 0$ and

$$c_k^{\text{right}}(x_{\underline{\ell},\underline{i}}) \coloneqq (x_{\ell_1,i_1}, \ldots, x_{\ell_k+1,2i_k+1} \ldots, x_{\ell_d,i_d}) \tag{2.27}$$

as its right child along the k-th dimension if $i_k < 2^{\ell_k}$. In addition, we define

$$c_k(x_{\underline{\ell},\underline{i}}) \coloneqq \begin{cases} \left\{ c_k^{\text{right}}(x_{\underline{\ell},\underline{i}}) \right\} & , i_k = 0 \\ \left\{ c_k^{\text{left}}(x_{\underline{\ell},\underline{i}}) \right\} & , i_k = 2^{\ell_k} \\ \left\{ c_k^{\text{left}}(x_{\underline{\ell},\underline{i}}),\ c_k^{\text{right}}(x_{\underline{\ell},\underline{i}}) \right\} & , else \end{cases} \tag{2.28}$$

as the set of children along the k-th dimension of the grid point $x_{\underline{\ell},\underline{i}}$, and

$$c(x_{\underline{\ell},\underline{i}}) \coloneqq \bigcup_{k=1}^{d} \left\{ c_k(x_{\underline{\ell},\underline{i}}) \right\} \tag{2.29}$$

as the set of all children of a grid point. It always holds that $\left|c(x_{\underline{\ell},\underline{i}})\right| \leq 2d$*, and in case there is no boundary involved,* $\left|c(x_{\underline{\ell},\underline{i}})\right| = 2d$ *holds as well. Furthermore, we denote a grid point* $x_{\underline{\ell},\underline{i}} \in X$ *as a leaf if it holds that*

$$c(x_{\underline{\ell},\underline{i}}) \cap X = \emptyset, \tag{2.30}$$

i. e., no children of it are already contained in X*.*

To choose the best grid points to refine, a refinement criterion is introduced. This abstract refinement criterion assigns a value to each grid point such that the grid points with the highest values can be refined.

Definition 2.6.2 (Refinement criterion)
Let X *be a set of grid points. We define* $\xi\colon X \mapsto \mathbb{R}$ *as the refinement criterion, which assigns each grid point a value that determines the suitability of a refinement. The commonly used surplus criterion,*

$$\xi_s(x_{\underline{\ell},\underline{i}}) \coloneqq \left|\alpha_{\underline{\ell},\underline{i}}\right|, \tag{2.31}$$

uses the absolute hierarchical surplus value, as a high value indicates steep rates of change in the area of a grid point.

To prepare for an iterative adaptivity algorithm, we define the grid points of adaptive sparse grids after m steps as $X_a^{(m)}$. A spatially adaptive sparse grid usually starts out as a regular sparse grid with a coarse level, i. e., $X_a^{(0)} = X_\ell^{\mathrm{s}}$. Then, during the m-th iteration, we first order all leaf grid points with

$$\xi(x_1) \geq \cdots \geq \xi(x_p), \quad x_1, \ldots, x_p \in X_a^{(m-1)}, \tag{2.32}$$

and then perform k refinements to obtain $X_a^{(m)} = X_a^{(m-1)} \cup c(x_1) \cup \cdots \cup c(x_k)$. Usually, when a leaf grid point is refined, all children are created because selective children creation would create additional overhead. After q steps, the spatially adaptive sparse grid is finished and consists out of the grid points $X_a^{(q)}$. Important parameters that influence the results are the amount of iterations q, the refinements per step k, the refinement criterion ξ, as well as the initial grid type and size of $X_a^{(0)}$.

There also exist various other established adaptivity methods that adapt the underlying grid in other ways. For example, dimensionally adaptive sparse grids use a more rigid refinement rule that adds the grid points of whole hierarchical subspaces to the surrogate. One small downside of spatially adaptive sparse grids is that usually all child grid points are added in a refinement step. In a dimensionality reduction focused setting this behaviour is not optimal, since we want to focus on more important dimensions when refining a grid point. One more recently introduced method are spatially dimension adaptive sparse grids [**KH16**] that provide a solution for this by only selectively adding child grid points along certain dimensions.

2.7 Radial basis functions

Radial basis function (RBF) surrogates [**BL88**] are an alternative way of approximating a model function. A radial basis function $\varphi'\colon \Omega \mapsto \mathbb{R}$ is a function whose value solely depends on the distance between the input and its associated center point $c \in \Omega$. More formally, there must exist a

function φ such that $\varphi'(x) = \varphi(\|x - c\|)$ holds. Functions that fulfill this property are called radial functions and the associated univariate function φ is called a radial kernel centered at c. Similar to sparse grid surrogates, a radial basis function surrogate is a linear combination of m radial basis functions with center points c_i and weights ω_i with

$$f(x) \approx \sum_{i=1}^{m} \omega_i \varphi(\|x - c_i\|). \tag{2.33}$$

In the context of this thesis, only Gaussian functions with the Euclidean distance $|\cdot|_2$ measure,

$$\varphi(x) \coloneqq e^{-(\epsilon|x-c_i|_2)^2}, \tag{2.34}$$

are used as basis functions. The only parameter that can be tuned is ϵ, which is referred to as the shape parameter and is used to scale the effects of the input radius. Given a set of center points, type of basis functions, and shape parameter ϵ, we can calculate the weight coefficients ω_i by solving a linear least squares problem. This, combined with the fact that exponentially more center points are needed to maintain the same level of coverage when the model dimensionality increases, also causes this method to suffer from the curse of dimensionality.

Radial basis function approximation is covered in this thesis as it is almost a polar opposite to grid-based approximation. Sparse grid surrogates require a hierarchical and axis-aligned grid, while radial basis function surrogates allow us to specify arbitrary center points. Furthermore, radial basis functions are fundamentally different to tensor-product-based basis functions with regards to their orientation. While tensor product basis functions are strictly axis-aligned, radial basis functions are unaware of the concept of axis-alignment or orientation as only the radius matters to them. This property will allow us to use radial basis functions as a reference when investigating the effects of basis function alignment later on.

2.8 Sampling

In the coming chapters we will make use of regression and some Monte-Carlo-based methods, both of which requiring model function observations as inputs. For this purpose, even the relatively simple process of drawing samples from the input parameter probability distribution of the model can improved to achieve better results. We assume that the inputs of model functions are distributed according to a d-dimensional multivariate distribution defined by the probability density function,

$$\rho(x) \coloneqq \prod_{i=1}^{d} \rho_i(x_i), \tag{2.35}$$

which is made up out of d independent distributions ρ_i.

A problem that arises when using a pseudorandom number generator to draw samples from the d-dimensional uniform input distribution is that the sequence does not cover the function domain as evenly and quickly as it theoretically can. One solution are so-called low-discrepancy sequences, which are also called quasirandom sequences to distinguish them from normal pseudorandom sequences. These sequences are more evenly distributed than pseudorandom samples and therefore have a lower discrepancy. Thus, they cause the result of various Monte-Carlo methods to converge faster than it would if we were just using the same amount of pseudorandom samples. A commonly

used quasirandom sequence is the Sobol sequence [**Nie88**], which is also used in this thesis. It is generated using the Antonov and Saleev variant that uses the Gray code of i to construct the i-th sample.

This approach can also be extended to allow for the generation of pseudorandom samples from non-uniform distributions through the commonly used inverse transform sampling method [**Dev86**]. However, this approach is not used in this thesis as the gains are minimal.

2.9 Error metrics

To evaluate the quality of a constructed surrogate $\hat{f}$, we define a couple of error metrics to measure the approximation error and also make it comparable to other surrogates. An established and commonly used error metric in the context of sparse grids is the L^2 error:

Definition 2.9.1 (L^2 error)
Let

$$\|g\|_{L^p}^p \coloneqq \int_\Omega |g(x)|^p \,\mathrm{d}x \tag{2.36}$$

be the L^p norm for a real function $g\colon \Omega \mapsto \mathbb{R}$. We then define

$$\varepsilon_{L^2}(f,\hat{f}) \coloneqq \|f-\hat{f}\|_{L^2} = \sqrt{\int_\Omega \left|f(x)-\hat{f}(x)\right|^2 \mathrm{d}x} \tag{2.37}$$

as the L^2 surrogate approximation error.

In numerics, to evaluate the accuracy of an estimator, the mean squared error and the root mean squared error are commonly used. These metrics can be described as the expected value of the squared errors and its square root, respectively.

Definition 2.9.2 (Mean squared errors)
Let $\hat{f}$ be model function surrogate for f and $S = \{(x_i, f(x_i)\}_{i=1}^n \subseteq \Omega \times \mathbb{R}$ a set of observations. We define

$$\varepsilon_{\mathrm{MSE}}(f,\hat{f}) \coloneqq \frac{1}{n}\sum_{i=1}^{n}\left(f(x_i)-\hat{f}(x_i)\right)^2 \tag{2.38}$$

as the mean squared approximation error of the surrogate. In the same fashion, we also define

$$\varepsilon_{\mathrm{RMSE}}(f,\hat{f}) \coloneqq \sqrt{\varepsilon_{\mathrm{MSE}}(f,\hat{f})} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(f(x_i)-\hat{f}(x_i)\right)^2} \tag{2.39}$$

as the root mean squared error of the surrogate.

The RMSE is closely related to the continuous L^2 error as it is just a discrete version of it in case the observations S are uniformly distributed. Thus, it holds that

$$\lim_{n\to\infty} \varepsilon_{\mathrm{RMSE}}(f,\hat{f}) = \varepsilon_{L^2}(f,\hat{f}) \tag{2.40}$$

when the inputs are uniformly distributed.

One problem of these error metrics is that they are all sensitive to the magnitude of the function values. To improve upon these absolute error metrics and allow for comparisons across model functions with different ranges of function values, we define normalized error metrics.

Definition 2.9.3 (Normalized root mean squared error)
Let $\hat{f}$ be a model function surrogate for f and $S = \{(x_i, f(x_i)\}_{i=1}^n \subseteq \Omega \times \mathbb{R}$ a set of observations. We define

$$\varepsilon_{\text{NRMSE}}(f, \hat{f}) := \frac{\varepsilon_{\text{RMSE}}(f, \hat{f})}{\max\limits_{(x,f(x))\in S} f(x) - \min\limits_{(x,f(x))\in S} f(x)} \tag{2.41}$$

as the normalized root mean squared error.

3 Input transformations

A lot of work has already gone into exploring all kinds of variants and optimizations for sparse grids that resulted in many different families of basis functions and grid types, showcased for example in [**Feu10**; **Val19**]. Moreover, many methods and algorithms that work on sparse grids have also been developed, for example in [**GG98**; **GGT01**; **Pfl10**; **Val19**; **Reh21**], and allow the general sparse grids technique to be applied in a better and more flexible way to many different kinds of problems. In the related literature, optimizing the sparse grid construction process itself was the primary focus most of the time. In contrast to this approach, we instead inspect and adapt the inputs itself prior to constructing any sparse grid. The basic idea that is propagated in this thesis is that instead of directly using the sparse grids to approximate a function with $f(x) \approx \hat{f}(x)$, we insert another layer of flexibility into the process by introducing an input transformation function t and creating a surrogate $\hat{f}_t$ in tandem with t such that a transformed sparse grid approximation $f(x) \approx \hat{f}_t(t(x))$ performs better. This general idea and also the necessary details will be expanded upon in this chapter.

3.1 Intrinsic dimensionality

There already exist several techniques used in signal processing and data science that are related to the idea of input transformations, for example principal component analysis (PCA) [**AW10**], which tries to find and exploit certain characteristics of the input data. Related to PCA is the concept of intrinsic dimensionality [**Ben69**]. The intrinsic dimensionality of a function or dataset is the minimal amount of features required to completely represent its output. It can be used as a guidance to determine the actual dimensionality of the model function and also optionally exploit the knowledge to construct a lower-dimensional surrogate, i. e., performing a dimensionality reduction.

Definition 3.1.1 (Intrinsic dimensionality)
A d-dimensional model function $f\colon \Omega \mapsto \mathbb{R}$ has an intrinsic dimensionality of $r \leq d$ if there exists an r-dimensional function $f_t\colon \Omega_r \mapsto \mathbb{R}$ and a transformation function $t\colon \Omega \mapsto \Omega_r$ such that

$$\forall x \in \Omega\colon\ f(x) = f_t(t(x)). \tag{3.1}$$

Even though we do not explicitly define Ω_r, it should also be a bounded Euclidean product such that it can transformed into the unit hypercube $\Omega_r = [0,1]^r$.

We denote the minimal intrinsic dimensionality of a function as r^*. Even if we are able to find a non-optimal r-dimensional transformation with $r > r^*$ that satisfies the requirement of def. 3.1.1, we still have gained an advantage and can define that function as having an intrinsic dimensionality of at most r. Furthermore, in case it holds that $r = r^*$, we still refer to it as having an intrinsic dimensionality of at most r because the true minimal intrinsic dimensionality r^* is often unknown.

Since we are dealing with function approximation problems in this thesis, we extend def. 3.1.1 to also allow for approximation errors:

Definition 3.1.2 (Approximate intrinsic dimensionality)
Let f be a d-dimensional function, and ε an error metric. We then define f as having an approximate intrinsic dimensionality of $r \leq d$ with an error of e, if there exists an r-dimensional function $f_t : \Omega_r \mapsto \mathbb{R}$ and a function $t : \Omega \mapsto \Omega_r$ such that

$$e = \varepsilon(f, f_t \circ t). \tag{3.2}$$

Similar to the minimal intrinsic dimensionality r^*, we are not concerned about the finding, and also usually do not know, the minimal error $e^* = \min_{t,f_t} \varepsilon(f, f_t \circ t)$ for a given r. The introduction of approximation errors also enables us to determine a good tradeoff between intrinsic dimensionality and approximation error.

To achieve a good convergence rate and general representativeness when performing an operation on the model function that makes use of observations, e. g., a Monte-Carlo quadrature operation, it is important to achieve a low discrepancy and thus cover the input space Ω more evenly. Under normal circumstances, to achieve the same discrepancy in a high-dimensional space as in a low-dimensional space, exponentially more observations are required. However, combining this requirement with the concept of intrinsic dimensionality, we observe that we effectively only need to cover Ω_r and adapt any operation that normally works on f to instead work on the lower dimensional function f_t. Therefore, the amount of observations required to achieve a good coverage of the domain of f_t would depend on r and not on d.

Definition 3.1.3 (Transformed distribution)
Let $t^{-1}(x_t) = \{x \in \Omega \mid t(x) = x_t\}$ be the inverse transformation function. We then define the transformed distribution,

$$\rho_t : \Omega_r \to \mathbb{R}_+, \quad \rho_t(x_t) = \int_{t^{-1}(x_t)} \rho(x)\, \mathrm{d}x. \tag{3.3}$$

This is also a valid probability distribution since

$$\int_{\Omega_r} \rho_t(x_t)\, \mathrm{d}x_t = \int_{\Omega_r} \int_{t^{-1}(x_t)} \rho(x)\, \mathrm{d}x_t = \int_{\Omega} \rho(x)\, \mathrm{d}x = 1 \tag{3.4}$$

and $\rho_t(x_t) \geq 0$ holds.

One thing to take note of is that applying a transformation on the original input distribution ρ has the potential to change the distribution type. For example, in case of a uniform distribution $\rho = \mathcal{U}[0,1]^d$, the associated transformed distribution ρ_t could be a non-uniform distribution. To effectively sample from ρ_t, it is still easier to just draw the samples $(x_1, \ldots, x_n)$ from ρ directly and feed them into the transformation function to obtain the transformed sequence $(t(x_1), \ldots, t(x_n))$, which is distributed according to ρ_t. The general quality and properties of transformed low discrepancy sequences (sec. 2.8) also change compared to its original, untransformed sequence [**WS08**].

3.2 Limitations of sparse grids

All grid-based interpolation methods share the common problem of being affected by the curse of dimensionality [**Bel61**], which effectively puts an upper limit on the feasible dimensionality of sparse grid surrogates. Sparse grids themselves are already an effective strategy to reduce the

consequences of the curse of dimensionality compared to full grids because they provide a good tradeoff between the amount of grid points used and their approximation quality. However, sparse grids can only weaken the curse of dimensionality and delay the effects of it a bit more. Therefore, employing some form of dimensionality reduction is necessary for high-dimensional models. Of course, the feasability and effectiveness of dimensionality reduction techniques is limited by the minimum amount of accuracy that is required for the specific use case. In this section, we take a look at a few already established approaches to dimensionality reduction in the context of sparse grids and take a look at their properties with the help of simple example functions.

The first example function,

$$f_1(x_1, x_2) = \sin(2\pi x_1) + 1, \tag{3.5}$$

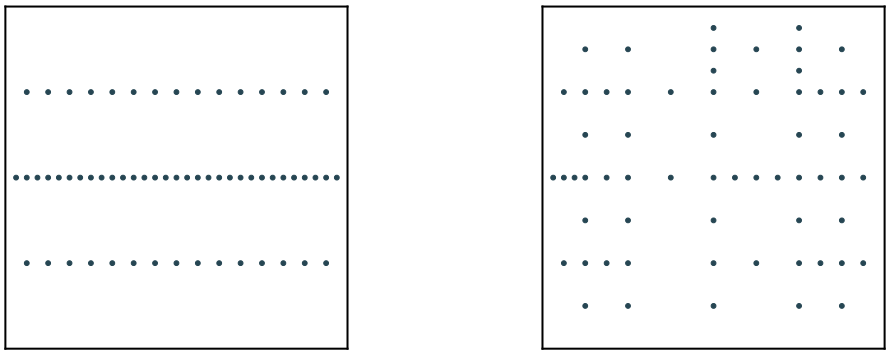
is designed to be a good fit for grid-based methods. It is a two-dimensional function, where only the first component and a constant term contribute to the total function value, while the second component is unused. We can easily see that it has an intrinsic dimensionality of 1 according to def. 3.1.1 since a transformation function could just cut off the second input component. By simply rotating the function by a few degrees, we obtain the second example function,

$$f_2(x_1, x_2) = \sin(2\pi x_1') + 1, \quad x_1' = \cos(0.15\pi)x_1 - \sin(0.15\pi)x_2, \tag{3.6}$$

which is also a two-dimensional function but more importantly still has an intrinsic dimensionality of 1.

3.2.1 Spatially adaptive sparse grids

We start off by interpolating both functions using spatially adaptive sparse grids to investigate how their generated grid points and interpolation errors compare.



(a) Surrogate grid points for f_1 **(b)** Surrogate grid points for f_2

Figure 3.1: Grid points of generated spatially adaptive sparse grids for f_1 and f_2. The sparse grids uses modified hat basis functions and started out as a regular sparse grid with level $\ell = 1$. Moreover, the surplus refinement criterion and 15 refinements steps with one grid point refinement per step were used to adapt the sparse grids.

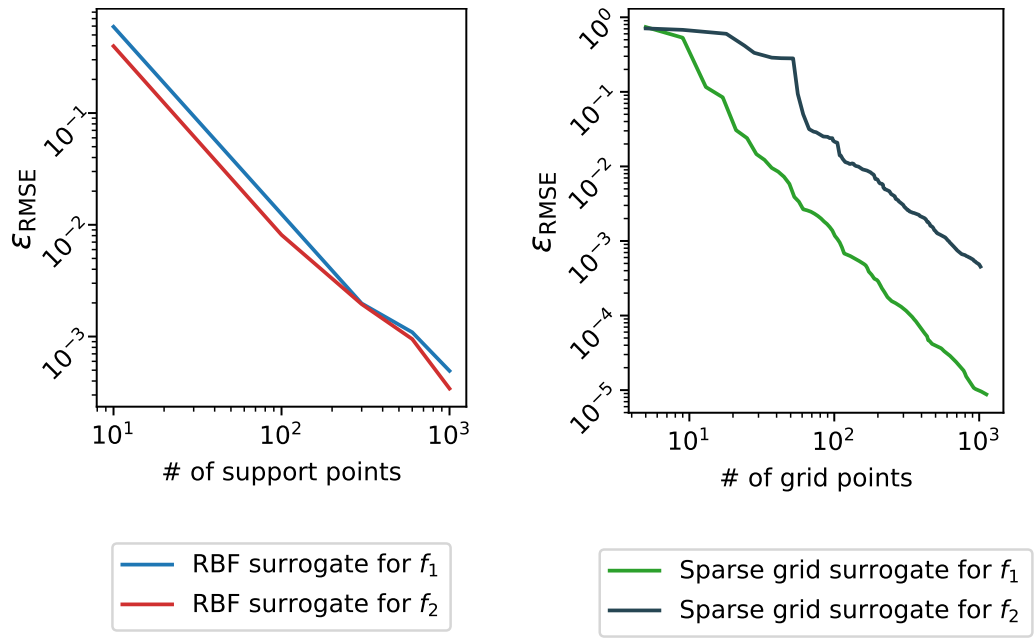
Fig. 3.1 illustrates how spatially adaptive sparse grids are affected by the different alignment of the input functions.

The spatially adaptive strategy is able to exploit the one-dimensional nature of f_1 in fig. 3.1a by only refining along the first dimension and therefore spending the grid points only where they matter. Note that due to the default refinement rule that add all child grid points, the upper and lower grid point rows are created even though they are not necessary. A tweaked refinement rule could easily fix this.

However, as we can see, fig. 3.1b, the intrinsically one dimensional function f_2 can not be exploited as it is no longer axis-aligned. This can be deduced from the fact that grid points all over the grid and along both dimensions are refined. Moreover, this observation is reinforced by the actual approximation errors of surrogates for both functions seen in fig. 3.2b.

3.2.2 Radial basis functions

Radial basis functions behave differently than sparse grids when being applied to the same two functions as their basis functions are not constructed using the tensor-product approach. Instead, they are orientation agnostic because only the radius relative to a support point determines the basis function value. Therefore, as seen in fig. 3.2a, the approximation errors stay almost identical in both cases. However, the general approximation quality for f_1 is worse compared to the sparse grid approximation since radial basis functions are not well suited for approximating functions that are constant along a dimension. Modified basis functions come with a special constant basis function that is well suited for this purpose and therefore have an advantage.



(a) RBF surrogates created with Gaussian basis functions, Euclidean distance measure, and shape parameter $\epsilon = 0.5$.

(b) Spatially adaptive sparse grids with $\ell = 1$, surplus refinement criterion, 280 single refinement steps, and mod. hat basis functions.

Figure 3.2: Approximation error comparison between radial basis functions and spatially adaptive sparse grids for both example functions f_1 and f_2.

3.2.3 Variance-based sensitivity analysis

The method of variance-based sensitivity analysis, a form of global sensitivity analysis introduced by Sobol [**Sob01**], is a well established method in statistics to determine the importance of model input parameters. Given a set of dimension indices $D = \{1, \dots, d\}$, we define the set of ANOVA components with $C \coloneqq \mathcal{P}(D)$ where an ANOVA component $c \in C$ that contains a specific dimension index $i \in D$ indicates that a function term is varying along the i-th dimension. We can then decompose a model function into a componentwise sum with

$$f(x) = \sum_{c \in \mathcal{P}(D)} f_c(x), \tag{3.7}$$

which is called the ANOVA decomposition of f.

A decomposition of a high-dimensional function into $|\mathcal{P}(D)| = 2^d$ lower-dimensional function terms allows for an examination of individual terms with the goal of identifying terms that contribute the least amount of variance to the total function variance. This decomposition can also be employed on sparse grids [**Feu10**] by using special semi-orthogonal basis functions, for example pre-wavelet basis functions, to represent the terms f_c. The terms f_c are required to be semi-orthogonal such that we can define the variance function,

$$\sigma^2(f) \coloneqq \int_\Omega f(x)^2 \rho(x) \, \mathrm{d}x = \sum_{c \in \mathcal{P}(D)} \sigma^2(f_c), \tag{3.8}$$

in an additive manner. For every ANOVA component $c \in C$, we are then able to calculate the variance share, also called Sobol index,

$$S_c \coloneqq \frac{\sigma^2(f_c)}{\sigma^2(f)}, \quad \sum_{c \in \mathcal{P}(D)} S_c = 1. \tag{3.9}$$

Computing the Sobol indices of all terms for both example functions gives insight into the structure of the function and tells us what share of the function we are able to represent with axis-aligned terms of varying dimension. It also helps us to understand why spatial adaptivity only works well for one of them. The Sobol method decomposes the two-dimensional functions into four axis-aligned terms. A constant term $f_\emptyset$, two one-dimensional terms $f_{\{1\}}$ and $f_{\{2\}}$, and a two dimensional term $f_{\{1,2\}}$.

	$2 \notin c$	$2 \in c$
$1 \notin c$	0.667	0.0
$1 \in c$	0.333	0.0

Table 3.1: Sobol indices for all ANOVA components of f_1.

	$2 \notin c$	$2 \in c$
$1 \notin c$	0.636	0.001
$1 \in c$	0.163	0.199

Table 3.2: Sobol indices for all ANOVA components of f_2.

As we can see in table 3.1, the method correctly identifies that only the terms $f_\emptyset$ and $f_{\{1\}}$ have some output variance with $\sigma^2(f_\emptyset) = 0.667 > 0$ and $\sigma^2(f_{\{1\}}) = 0.333 > 0$. Therefore, the other terms can be dropped and the function can be completely represented using only one-dimensional function terms.

However, we can see in table 3.2 that the Sobol method attributes a share of output variance to all terms, since it holds that $\sigma^2(f_c) > 0$ for all ANOVA components. In this case, the Sobol method shows us that no dimensionality reduction can be performed, as the two-dimensional term $f_{\{1,2\}}$ has a decent amount of variance contribution. Note that if the constant term $f_\emptyset$ is excluded from the analysis, as is often done, the effects of the changed alignment of f_2 on the Sobol indices would be even greater.

3.2.4 Summary

We can conclude that a fundamental property of any function approximation method that uses tensor-product basis functions is that the approximation quality significantly depends on the axis-alignment of the model function itself. In contrast to this behaviour, other basis functions, e. g., radial basis functions, do not depend on the alignment of the function terms they are trying to approximate, i. e., they can be seen as orientation agnostic. In turn this means that an effective dimensionality reduction with conventional sparse grid based methods is highly dependent on the alignment of the model function.

Without adding another layer of flexibility, sparse grids can not use their full potential in many cases as we have seen with the function f_2. If we were somehow able to change the alignment of a model function in a more axis-aligned way that matches the underlying sparse grid surrogate, we could improve the approximation in general and also allow a more efficient dimensionality reduction to take place. This will allow us to perform a better dimensionality reduction.

3.3 Active direction transformations

The concept of intrinsic dimensionality allows us, at least in theory, to use arbitrary transformation functions to map into the lower-dimensional input space Ω_r. For this purpose, there are numerous possible types of transformation functions to choose from. As an example, the previous chapter showed how a simple change of basis of the input parameters can have a huge impact on constructed sparse grid surrogates. We will therefore start out with constructing an orthonormal transformation matrix $W \in \mathbb{R}^{d \times d}$ to transform the inputs into a new coordinate system prior to passing them to a sparse grid surrogate. A good first choice of new basis vectors would be some form of directions along which the model function output has high rates of change. Various data analysis methods, such as PCA [**AW10**] or active subspaces [**Con15**], employ some form of a singular value decomposition to obtain these directions of high change. As the active subspace method will be used later on to obtain W, we will refer to the new basis vectors of the matrix W as active directions, which is consistent with the active subspace method. We then refer to the matrix W as the active direction matrix and assume that its basis vectors are ordered by the level of change from highest to lowest, i. e., it should hold that the column vector w_1 is the most active direction while w_d is the most inactive direction.

After performing a change of basis with W on the inputs, we can effectively eliminate the most inactive dimensions, as all active directions are now mapped onto dimensions in the new coordinate system. This results in the possibility of drastically reducing the surrogate dimensionality while only suffering a small increase in the approximation error, as only the most inactive directions, which should not contribute a lot to the model function, are eliminated that way.

Definition 3.3.1 (Reduced active direction matrix)
Let $W \in \mathbb{R}^{d \times d}$ be an active direction matrix with $W^T W = I$, and let $r \leq d$ be the reduced dimensionality. We then define

$$W_r := \begin{bmatrix} | & | & & | & | \\ w_1 & w_2 & \dots & w_{r-1} & w_r \\ | & | & & | & | \end{bmatrix} \tag{3.10}$$

as the reduced active direction matrix.

As we can see, the reduced active direction matrix is simply obtained from a given active direction matrix W and a reduced dimension r by cutting off the last $d - r$ directions from W. For now, we assume that W and r are given as we will cover the process of determining those essential inputs later on in detail.

3.3.1 Unit hypercube projections

To construct a transformation function t that maps from the d-dimensional unit hypercube $\Omega = [0, 1]^d$ to the r-dimensional unit hypercube $\Omega_r = [0, 1]^r$ with $r < d$ for a given transformation matrix W_r, several adjustments have to be made in order to obtain a usable transformation function as the naive linear transformation approach with $W_r^T x$ would map inputs outside of Ω_r in some cases. When applying a transformation $W_r^T x$ on the elements of a hypercube $x \in \Omega$, the result can be described with the help of zonotopes.

Definition 3.3.2 (Zonotope)
Let $m \in \mathbb{N}$, and let $k \geq m$ be the amount of m-dimensional generator vectors $g_1, \dots g_k \in \mathbb{R}^m$. We define the set

$$Z := \left\{ \sum_{i=1}^{k} g_i x_i \mid -1 \leq x_i \leq 1 \right\} \tag{3.11}$$

as an m-dimensional zonotope with k generator vectors.

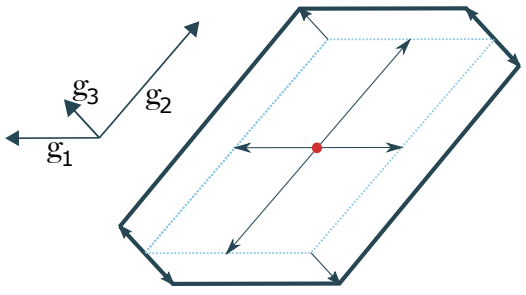


Figure 3.3: Exemplary zonotope with $m = 2$, $k = 3$, and the three generators g_1, g_2, g_3. The outline of the zonotope is coloured in dark blue and the origin in red.

When applying the projection W_r^T to the inputs $x \in \Omega$, we can observe that

$$W_r^T x = W_r^T \sum_{i=1}^{d} x_i e_i = \sum_{i=1}^{d} x_i w_i, \quad w_i = W_r^T e_i, \tag{3.12}$$

where the w_i are just the transformed unit vectors. Therefore, since for all components are contained in the unit interval with $x_i \in [0, 1]$, we can express the transformed inputs with

$$\{W_r^T x \mid x \in \Omega\} = \left\{ \sum_{i=1}^{d} x_i w_i \mid x \in \Omega \right\} = \left\{ \sum_{i=1}^{r} x_i w_i \mid 0 \leq x_i \leq 1 \right\}. \tag{3.13}$$

To get (3.13) closer to the zonotope definition (3.11), we first center the inputs $x \in \Omega$ before applying the transformation and divide the vectors w_i by 2. For this, we define $c_i = \underline{0.5}$ as the center of the i-dimensional unit hypercube. We observe that a linear transformation using W_r on centered inputs will yield the following result:

$$\begin{aligned}
\left\{W_r^T (x - c_d) \mid x \in \Omega\right\} &= \left\{ W_r^T \sum_{i=1}^{d} \left(x_i - \frac{1}{2} \right) e_i \mid x \in \Omega \right\} \\
= \left\{ \sum_{i=1}^{d} \left(x_i - \frac{1}{2} \right) W_r^T e_i \mid x \in \Omega \right\} &= \left\{ \sum_{i=1}^{d} \left(x_i - \frac{1}{2} \right) w_i \mid x \in \Omega \right\}, \quad w_i = W_r^T e_i \\
= \left\{ \sum_{i=1}^{d} \left(x_i - \frac{1}{2} \right) w_i \mid 0 \leq x_i \leq 1 \right\} &= \left\{ \sum_{i=1}^{d} x_i w_i \mid -\frac{1}{2} \leq x_i \leq \frac{1}{2} \right\} \\
= \left\{ \sum_{i=1}^{d} \frac{1}{2} x_i w_i \mid -1 \leq x_i \leq 1 \right\} &= \left\{ \sum_{i=1}^{d} x_i p_i \mid -1 \leq x_i \leq 1 \right\}, \quad p_i = \frac{1}{2} w_i
\end{aligned} \tag{3.14}$$

Therefore, the codomain of a centered transformation with W_r is equal to the zonotope

$$Z_p := \left\{ \sum_{i=1}^{d} x_i p_i, \; -1 \leq x_i \leq 1 \right\}, \quad p_i := \frac{1}{2} W_r^T e_i, \tag{3.15}$$

and the image of the function

$$p \colon \Omega \mapsto Z_p, \quad p(x) = W_r^T (x - c_d) \tag{3.16}$$

is equal to Z_p, i. e., it holds that $Z_p = p(\Omega)$.

3.3.2 Surrogate space

After we obtained a zonotope Z_p, the next step is to determine how to go from this zonotope to a hypercube because sparse grid surrogates require the domain to be a hypercube. To achieve this, the zonotope Z_p is first encased in an r-dimensional hypercube, called the surrogate space S, which is then scaled and aligned to obtain the unit hypercube Ω_r. The generators $q_1, \ldots, q_r$ of the encasing hypercube can be calculated by applying the Gram–Schmidt process on the ordered zonotope generators p_i to obtain the orthogonal vectors,

$$q_i := \frac{q_i'}{\|q_i'\|}, \quad q_i' := p_i - \sum_{k=1}^{i-1} \frac{\langle p_k, q_k' \rangle}{\langle q_k', q_k' \rangle} q_k'. \tag{3.17}$$

The surrogate space, which encases the zonotope Z_p, is then an r-dimensional hypercube,

$$S := \left\{ \sum_{i=1}^{r} s_i q_i x_i, \; -1 \leq x_i \leq 1 \right\}, \quad s_i := \sum_{k=1}^{d} \langle q_i, p_k \rangle, \tag{3.18}$$

where the scaling factors s_i are chosen in such a way that the hypercube S exactly matches the extent of the zonotope Z_p, as seen in fig. 3.4.

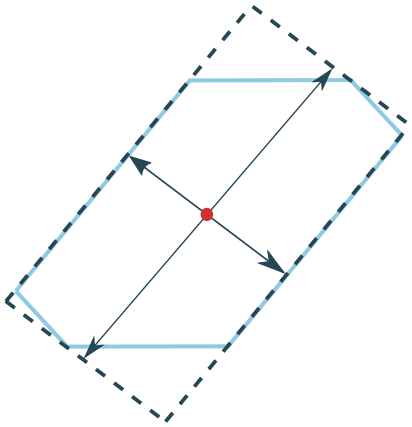


Figure 3.4: Construction of the encasing surrogate space S according to (3.18) for the zonotope Z_p from fig. 3.3. The outline of the zonotope Z_p is coloured in light blue, the encasing surrogate space S in dotted dark blue, and the origin in red .

As we only need r generators to construct the surrogate space but have d generators of Z_p available, not all p_i are used in the surrogate space construction. In theory, the generators p_i that are used to calculate the q_i don't have to be ordered. However, to emphasize the most important generators p_i, they are ordered by their magnitude such that $|p_1| \geq \cdots \geq |p_d|$. This is done to achieve the best possible encasement of Z_p, which is aligned along the dominant generators with respect to their length.

3.3.3 Reduced unit hypercube

The surrogate space S, which is already a hypercube, is then transformed into the unit hypercube Ω_R in the last necessary step. First, we observe that the generators of S are created using the orthogonal vectors q_i and the scaling factors s_i to obtain the generator matrix,

$$G \coloneqq \begin{bmatrix} g_1 & \dots & g_r \end{bmatrix} = Q \operatorname{diag}(s_1, \dots, s_r), \quad Q \coloneqq \begin{bmatrix} q_1 & \dots & q_r \end{bmatrix}, \tag{3.19}$$

in which the column vectors $g_i = s_i q_i$ are the corresponding generators of S. To bring the elements of the surrogate space into a coordinate system of a unit hypercube, we perform a change of basis on them. The change of basis formula to transform the r-dimensional elements $x_p = p(x)$ of S into a representation y in the new basis G is $x_p = Gy \Leftrightarrow G^{-1}x_p = y$, and we can therefore obtain

$$y = G^{-1}p(x) = \operatorname{diag}\left(\frac{1}{s_1}, \dots, \frac{1}{s_r}\right) Q^T p(x), \tag{3.20}$$

where we know that, according to (3.18), $y_i \in [-1, 1]$ holds. Once the inputs are represented in the new basis, we transform the surrogate space into the r-dimensional unit hypercube Ω_r by first scaling S down by 2 and then uncentering it again to finally obtain the usable transformation function $t_{W_r} : \Omega \mapsto \Omega_r$ for an active direction matrix W_r with

$$t_{W_r}(x) = c_r + \frac{1}{2}\operatorname{diag}\left(\frac{1}{s_1}, \dots, \frac{1}{s_r}\right) Q^T p(x), \tag{3.21}$$

which achieves the goal of mapping from Ω into Ω_r just through some scaling and centering operations. This last step of the transformation process that transforms S into a unit hypercube is shown in fig. 3.5.

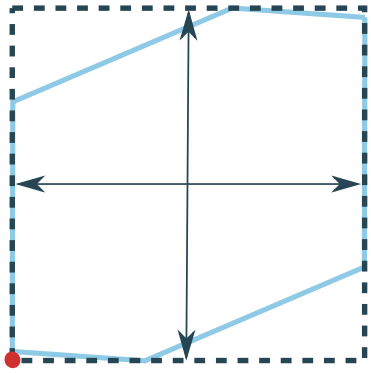


Figure 3.5: Visualization of transformation used to obtain the unit hypercube Ω_r according to (3.20). The outline of the transformed inputs $t_{W_r}(\Omega)$ is coloured in light blue, the unit hypercube Ω_r in dotted dark blue, and the origin in red.

3.3.4 Visualizing transformations

The described transformation process can be visualized by illustrating where points from the lower dimensional space Ω_r would reside in the original input space Ω. While it is normally not required to revert the previously shown transformation process, i. e., going from Ω_r back to Ω, it can be useful for the visualization of transformed surrogates.

We observe that (3.21) can be rearranged to solve for $p(x)$ with

$$p(x) = 2G(t_{W_r}(x) - c_r) = 2Q \operatorname{diag}(s_1, \ldots, s_r)\,(t_{W_r}(x) - c_r). \tag{3.22}$$

To reverse the transformation completely, we have to deal with reversing $p(x)$ and the fact that in many cases the inverse function p^{-1} is not injective. Our goal is to reverse the transformation only onto certain representative elements of the equivalence classes $[x]_p := \{x \in \Omega \mid p(x) = x_p\}$. We then only consider the points

$$C := \{(x - c_d) \in \operatorname{span} W_r \mid x \in \Omega\}, \tag{3.23}$$

i. e., the set of points which lie in the subspace that is spanned by the active directions of W_r and crosses the center c_d. We reverse $p(x) = W_r^T(x - c_d) \Leftrightarrow W_r p(x) + c_d = x$ only onto the representative elements $x \in C$ so that we are able to map one element $x_r \in \Omega_r$ onto one element $x \in C$ to obtain $t_{W_r}^{\text{reverse}} : \Omega_r \mapsto C$ with

$$t_{W_r}^{\text{reverse}}(x_r) = W_r 2Q \operatorname{diag}(s_1, \ldots, s_r)\,(x_r - c_r) + c_d. \tag{3.24}$$

This function is denoted as a reverse function, as it is not a true inverse of t_{W_r}.

In fig. 3.6, we can examine the result of reversing the grid points of a regular two-dimensional sparse grid lying in Ω_r onto a plane in the original three dimensional space Ω with the help of (3.24). As we can see, some grid points lie outside of Ω when being visualized this way. We will focus on this and other peculiarities inherent to transformations in the next chapter.

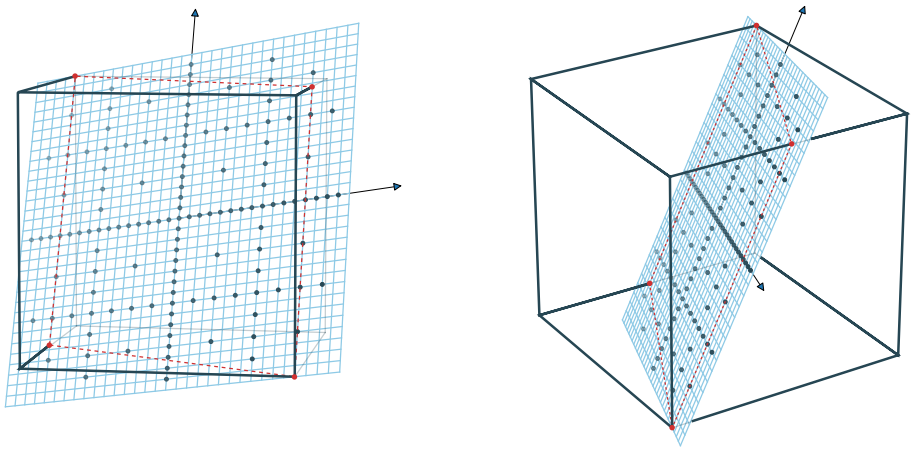


Figure 3.6: Two views of the same transformed regular sparse grid of level 5 with $r = 2$ and $d = 3$. The dark blue ■ cube outline represents the edges of Ω, the arrows are the first two active directions of W_r, and the dotted red ■ lines mark the intersections of C and the edges of Ω.

3.4 Active subspaces

One way to construct the matrix W_r, which is the foundation of the active direction transformation function t_{W_r}, is through the method of active subspaces [**Con15**]. This dimensionality reduction technique conducts a parameter study to identify the most influential directions in the input space to construct a lower-dimensional active subspace of Ω that covers most of the model output variance. The unused part of the domain, which is spanned by the leftover inactive directions, is referred to as the inactive subspace and is cut off. To obtain the orthonormal active directions, the technique performs a singular value decomposition on the average outer product of the gradients,

$$C := \int_{\Omega} (\nabla f)(\nabla f)^T \rho \, \mathrm{d}x, \tag{3.25}$$

a $(d \times d)$ matrix. Since C is a positive semi-definite matrix, it is possible to decompose it into its eigenvectors w_i and their corresponding real eigenvalues λ_i with

$$C = W \Lambda W^T, \quad \Lambda := diag(\lambda_1, \ldots, \lambda_d), \quad W = \begin{bmatrix} w_1 & w_2 & \ldots & w_{d-1} & w_d \end{bmatrix}, \tag{3.26}$$

where a larger eigenvalue indicates a higher rate of change along the direction. More precisely, it is the average squared directional derivative of the function with respect to the eigenvector w_i [**CG14**] with

$$\lambda_i = \mathbb{E}\left[\left((\nabla f)^T w_i \right)^2 \right]. \tag{3.27}$$

We furthermore assume that the eigenvectors in this decomposition are ordered by the magnitude of their eigenvalues, i. e., $|\lambda_1| \geq \cdots \geq |\lambda_d|$ should hold Therefore, the column vectors of the matrix W are ordered from the most the active direction w_1 to the least active direction w_d. By keeping only a specific amount of the most active directions $r \leq d$, we obtain the active direction matrix with

$$W_r = \begin{bmatrix} w_1 & w_2 & \dots & w_{i-1} & w_r \end{bmatrix}. \tag{3.28}$$

The active subspace method itself usually determines r by looking at the largest gaps between the eigenvalues to determine suitable cutoff points. However, we deviate from this approach and will cover the process of determining r later on as the eigenvalue gap approach is not the most optimal choice in the context of this thesis.

To actually compute the active directions, a set of model gradient observations, $S_g := \{(x_i, \nabla f(x_i))\}_{i=1}^n$, is used to derive the matrix C with the help of a classical Monte-Carlo approach from [**CG14**]. With this approach, the matrix C can be approximated by computing

$$C \approx \frac{1}{n} \sum_{i=1}^{n} (\nabla f(x_i)) (\nabla f(x_i))^T . \tag{3.29}$$

3.4.1 Estimating gradients for active subspaces

One potential downside of the active subspace method is that gradient observations are required to approximate the matrix C in (3.29). In cases where the gradient function is not known and can not be easily computed, this would seem like a roadblock limiting the general feasability of the active subspace method. However, as this section will show, the gradients can be approximated in various ways from available observations and can also be used to obtain an acceptable active subspace approximation. We will cover a few different methods of achieving this, where each one has its unique advantages and disadvantages as we will see.

3.4.2 Finite differences

The most common approach for estimating gradients of a function are finite differences. In this thesis, an adaptive mix of forward and backward differences with a fixed distance h are used, where the range of possible spacing is constrained by $0 < h \leq 0.5$. We can approximate $\nabla f(x)$ with

$$\frac{\partial f(x)}{\partial x_i} \approx \begin{cases} \dfrac{f(x_1, \dots, x_i + h, \dots, x_d) - f(x)}{h}, & x_i + h \leq 1 \\ \dfrac{f(x_1, \dots, x_i - h, \dots, x_d) - f(x)}{h}, & \text{else} \end{cases} . \tag{3.30}$$

The downside in the context of an observation-based approach is the requirement to additionally evaluate the model function d times to approximate the gradient at one observation sample point x_i. For high values of d and a large amount of observations n, additional dn function evaluations for finite differences may become too costly.

3.4.3 Directional derivatives

In cases where finite differences are not suitable, the gradients can be determined using a different approach that only works on a given observation sample and does not require further function evaluations. The approach is to roughly approximate the gradient by looking at neighbouring observations and using directional derivatives to create a system of linear equations for the gradient at a certain observation point. Given two inputs $x, y \in \Omega,\ x \neq y$, the simple approximation rule

$$\langle y - x, \nabla f(x) \rangle \approx f(y) - f(x) \tag{3.31}$$

then just extrapolates the gradient along the direction $y - x$ in a linear fashion.

By choosing $m \in \mathbb{N}$ neighbours $N = \{(y_i, f(y_i))\}_{i=1}^{m}$ of an observation $(x, f(x))$ from the available observation set $S = \{(x_i, f(x_i))\}_{i=1}^{n}$ with $N \subseteq S \setminus \{(x, f(x))\}$, we can create a system of linear equations with the linear extrapolation approach to obtain

$$\begin{bmatrix} y_1 - x \\ \vdots \\ y_m - x \end{bmatrix} \widetilde{\nabla} f(x) = \begin{bmatrix} f(y_1) - f(x) \\ \vdots \\ f(y_m) - f(x) \end{bmatrix}. \tag{3.32}$$

Solving this system of linear equations for a residual minimizing value for $\widetilde{\nabla} f(x)$ only provides a very rough approximation of the actual gradient. Furthermore, the linear nature of the approximation, especially when estimating gradients of a non-linear function, can lead to biased gradients. The choice of the neighbours y_i also influences the result as well as the amount of neighbours m.

In the context of active subspaces, the difference between the approximated gradients and the true gradients is not the relevant metric to evaluate the actual quality of approximated gradients. Instead, the primary focus should be the resulting surrogate approximation error, as it is the only metric that matters in the end. Even though the active subspaces computation is based entirely on the average outer product of the gradients, there is not necessarily a strong link between the gradient approximation errors and the quality of the resulting active subspace. Therefore, the evaluation of the active subspace approximations will be done later on when the surrogate quality in general is investigated.

Random neighbour approximation The random neighbour approximation method takes a subset $S_{\text{rn}} \subseteq S \setminus \{(x, f(x))\}$ with $|S_{\text{rn}}| = m < n$ as input. The set $S_{\text{rn}} = \{(y_i, f(y_i)\}_{i=1}^{m}$ is then called a random neighbour set of x. This random neighbour set is used as an input for (3.32) to compute $\widetilde{\nabla} f(x)$. Choosing $m \ll n$ can be useful in cases where a lot more observations are available than are necessary for the system of linear equations to return a usable approximation. This way, the time spent solving the system of linear equations is kept to a minimum.

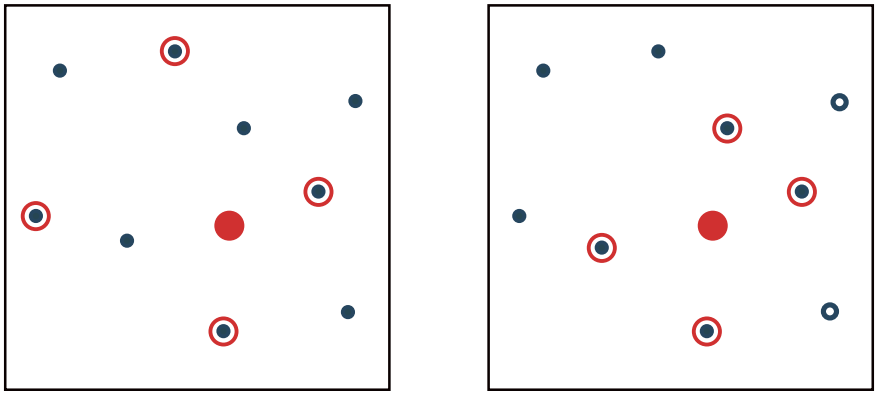
For $n, m \to \infty$, the approximated gradient does not converge to the actual gradients, since the expected distance to the neighbours in a random neighbour set does not approach zero. The problem of determining average neighbour distance is related to finding the mean line segment length $\Delta(d)$ in a d-dimensional unit hypercube [**BBC07**]. There is no closed form solution for $\Delta(d)$, however several approximations have been developed for various dimensionalities in [**Wei**], e. g., $\Delta(1) = \frac{1}{3}$, $\Delta(2) = 0.52$, $\Delta(3) = 0.66$. As $\Delta(d)$ grows with the dimensionality d, this results in higher average neighbour distances for greater values of d.

Nearest neighbour approximation The nearest neighbour variant first picks a subset $S' \subseteq S \setminus \{(x, f(x))\}$ with a manageable size $n' = |S'|$ similar to the random neighbour method and then computes the $m \leq n'$ nearest neighbours $S_{\text{nn}} = \{(y_i, f(y_i)\}_{i=1}^{m} \subseteq S'$ with

$$\|(y_1 - x)\|_2 < \|(y_2 - x)\|_2 < \cdots < \|(y_{n'-1} - x)\|_2 < \|(y_{n'} - x)\|_2. \tag{3.33}$$

The obvious advantage is that the used neighbours lie way closer to x than with the random neighbour method. Fig. 3.7 illustrates the differences between both methods.

Compared to the random neighbour approximation, for $n, n', m \to \infty$, the approximated gradient does converge to the actual gradient because the expected distance to the neighbours does approach zero. However, this property comes at the cost of first having to sort the available n' neighbours by distance to pick the m closest ones. The parameters n' and m also have to be chosen carefully as an unsuitable combination of n' and m would increase the average distance to the neighbours again such that the effect of choosing the nearest neighbours would vanish.



(a) Random neighbour method picks with $m = 4$.

(b) Nearest neighbour method picks with $n' = 7$ and $m = 4$.

Figure 3.7: Differences in the neighbour selection between the two neighbour approximation methods for an observation set S with $|S| = 10$. The dark blue ■ points are the available observations $S \setminus \{(x, f(x))\}$. A hollow point signals that the observation is not considered when looking for neighbours. The red ■ point represents x, and other points circled in red ■ represent observations that were chosen as a neighbour of x by the respective method.

3.4.4 Limitations

Active subspaces, even if the true gradients are used to compute them, are not a perfect indicator on which active directions are suitable for a active direction transformation construction since they can be misled. According to (3.27), a larger eigenvalue indicates a higher rate of change along the direction. More precisely, it represents the average squared directional derivative of f with respect to its eigenvector. However, a large average squared directional derivative does not necessarily indicate that a direction v_i is worth keeping. For example, the function

$$f(x_1, x_2) = 10x_1 + \frac{1}{10}\sin(100\pi x_2) \tag{3.34}$$

illustrates the limitations of active subspaces. The optimal active direction worth keeping would be the first dimension only, as the second term can be seen as a highly changing noise term that does not affect the approximation error a lot if removed as the amplitude of the sine is small. However, as the average squared directional derivative along the second dimension $(0,1)^T$ is way higher than along the first dimension $(1,0)^T$, the calculated active subspace is biased towards the second dimension. Therefore, we can not assume that the computed active subspace is always usable as directions with a high rate of change do not always correspond to directions that cover huge portions of the functions output variance.

A positive aspect of the active subspace neighbour approximation methods is that they effectively introduce a regularization through discretization [**Kre99**], i. e., as the distances between neighbours are comparatively high, they automatically discard noise terms with high frequencies such as the one shown in the previous example. In certain cases, this can lead to more suitable active subspace approximation when using neighbour approximation methods compared to alternatives such as the finite differences method.

3.5 Random active directions

A completely different approach to finding orthonormal active directions w_i to construct a transformation is just generating random ones. This approach is easy to implement, can generate new transformation functions almost instantly, and offers a purely exploratory way of finding the best transformation. It can also serve as a reference for comparison as the active subspace method should ideally, except for rare special cases, output better active directions.

A simple generation algorithm first generates random matrices $R \in \mathbb{R}^{d\times d}$ until all column vectors are linear independent and then orthonormalizes R using the Gram-Schmidt process to obtain the column vectors with

$$w_i = \frac{w_i'}{\|w_i'\|}, \quad w_i' = r_i - \sum_{k=1}^{i-1} \frac{\langle r_k, w_k' \rangle}{\langle w_k', w_k' \rangle} w_k', \tag{3.35}$$

and therefore the orthonormal active direction matrix

$$W = \begin{bmatrix} | & | & & | & | \\ w_1 & w_2 & \dots & w_{r-1} & w_d \\ | & | & & | & | \end{bmatrix}. \tag{3.36}$$

The final step randomly rolls a reduced dimensionality value r with $1 \leq r \leq d$ and cuts off the last $d - r$ column vectors of W to obtain the reduced active direction matrix

$$W_r = \begin{bmatrix} | & | & & | & | \\ w_1 & w_2 & \dots & w_{r-1} & w_r \\ | & | & & | & | \end{bmatrix}. \tag{3.37}$$

Note that the generated random orthonormal matrices are not perfectly uniformly distributed in the orthogonal group $O(d)$ and we would need a different algorithm to achieve a true uniform distribution [**WS08**]. The simple Gram-Schmidt based approach is however perfectly usable for the purpose of generating input transformations.

3.6 Alternative transformations

Until now, the focus lied primarily on active directions because they are relatively easy to use and construct. However, in theory we can construct and use arbitrary transformation functions to improve the model approximation. This section will briefly cover a few alternative types of transformations that will not be covered in detail in this thesis but do show some potential that could be investigated further.

3.6.1 Periodic transformations

A possible use case for alternative transformations are periodic functions. If the model function is for example periodic along one dimension i with the period p_i, i. e., it holds that $f(x_1, \dots, x_i, \dots, x_d) \approx f(x_1, \dots, x_i + p_i, \dots, x_d)$, we can easily define a transformation function $t_{\text{periodic}} \colon \Omega \mapsto \Omega$ with

$$t_{\text{periodic}}(x_1, \dots, x_i, \dots, x_d) = (x_1, \dots, \frac{x_i \bmod p_i}{p_i}, \dots, x_d)^T, \tag{3.38}$$

and create a surrogate $\hat{f}$ in Ω with $f(x) \approx \hat{f}(t_{\text{periodic}}(x))$. This way, the function can be interpolated better by sparse grids since we only approximate one periodic interval and therefore effectively employ $1/p_i$ times more grid points along the i-th dimension. In theory, this concept can be also expanded upon with handling multiple periodic dimensions at once, antiperiodic functions, offsets, and combination with other transformations.

A simple example would be the periodic function $f_{\text{periodic}}(x) = \sin(16\pi x_1) + \cos(16\pi x_2)$, which repeats in intervals of $\frac{1}{8}$ along both dimensions. This can be exploited by constructing a periodic transformation with $p = p_1 = p_2 = \frac{1}{8}$ to obtain

$$t(x_1, x_2) = \left[\frac{1}{p}(x_1 \bmod p), \frac{1}{p}(x_2 \bmod p)\right]^T. \tag{3.39}$$

To illustrate the improvement of the approximation quality in multiple smaller steps, we investigate a periodic transformation with several periods in fig. 3.8. As we can see, a smaller value for p results in a notable improvement, and going from $p = 1$ (no transformation) to $p = 0.125$ (best periodic transformation) improves the approximation error by a factor of around 10^2.

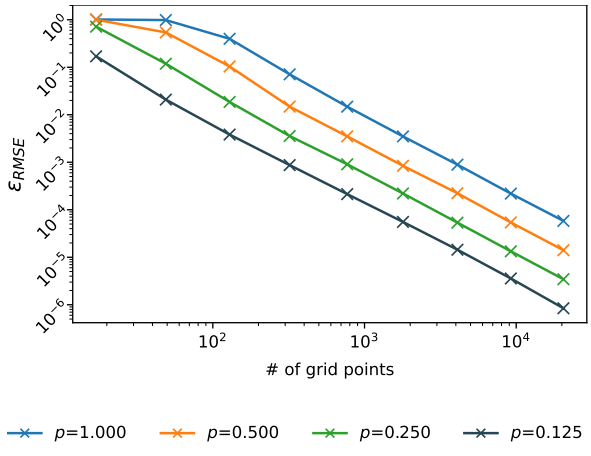


Figure 3.8: Approximation errors of various periodic transformations with periods $p \in \{\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$ in combination with sparse grid surrogates of level $\ell \in \{1, \dots, 11\}$ and modified hat basis functions for the function f_{periodic}.

3.6.2 Manifolds

Manifold-based methods for dimensionality reduction aim to identify nonlinear subspaces in the domain that are suitable for a dimensionality reduction. They have the potential to create more flexible transformations compared to active directions but are also harder to implement and make use of.

Active Manifolds Inspired from active subspaces, which are purely linear in their nature, we can extend the concept to identifying a one-dimensional curve in the domain Ω along the flow of most active change, which is called an active manifold [**BGF+19**]. Compared to active subspaces, calculating the active manifold is more complex and resource intensive. Furthermore, defining a transformation function that maps from Ω onto the local one-dimensional space of the curve is way more complex and can be constructed in possibly many different ways. This technique is currently limited to only one dimension with $r = 1$, which results in the used surrogate construction method becoming irrelevant.

Principal Manifolds A similar concept to active manifolds are principal manifolds [**HS08**]. They are an extension to principal component analysis [**AW10**], which can only compute linear principal components. The principal manifold technique is able to construct nonlinear multi-dimensional manifolds and is therefore more powerful than active manifolds. This technique has already been investigated in conjunction with sparse grids in [**FG09**].

4 Transformed surrogates

Once a transformation function has been created, the next step is creating the surrogate in the reduced input space Ω_r. As we will see in this chapter, this is not a trivial thing to do with sparse grids and requires special care as there are several hurdles that come with the usage of input transformations. Some of these are specific to sparse grids, whereas others are independent of the used surrogate type.

4.1 Transformed input space

All transformed inputs form the transformed input space, which is a part of the reduced input space Ω_r. In the last chapter, the transformation was constructed in such a way that every transformed input lies in Ω_r while also applying appropriate scaling to still use as much space of Ω_r as possible.

Definition 4.1.1 (Transformed input space)
Let $t\colon \Omega \mapsto \Omega_r$ be transformation function. We define the transformed input space as

$$T \coloneqq \{t(x) \mid x \in \Omega\},\ \ T \subseteq \Omega_r. \tag{4.1}$$

The main difference between T and Ω_r is that Ω_r is a unit hypercube, while T might not be one. As we will see later on, T is not a unit hypercube in many cases. We should strive to have T cover Ω_r as much as possible to not make a huge portion of Ω_r go unused when constructing the surrogate.

Definition 4.1.2 (Unused reduced surrogate inputs)
Let $t\colon \Omega \mapsto \Omega_r$ be transformation function, and let $U = T \setminus \Omega_r$ be the unused reduced surrogate space. We then define

$$\tilde{u}_t \coloneqq \frac{|U|}{|\Omega_r|} \tag{4.2}$$

as the share of unused space in the reduced input space.

The greater U, and therefore the unused input share $\tilde{u}_t$, the more grid points of a sparse grid can potentially be generated outside of T. Note that even if a grid point lies outside of T, it can still contribute to some degree to the approximation error of the surrogate if the support of the basis function ϕ intersects with T. Thus, grid points are only completely unused if their basis function support does not intersect with T at all. Other surrogate types do not have the same issue as radial basis functions for example can freely choose their support points and therefore automatically adapt to the shape of T in relation to Ω_r.

4.2 Method selection

Usually, the standard sparse grid technique makes use of interpolation (see sec. 2.5) and is able to generate an accurate interpolant quickly. However, as we will show in this section, sparse grid interpolation is not suitable for many scenarios involving transformations and we will have to differentiate between two cases.

4.2.1 Interpolation

If it holds that $r = d$, i. e., we do not perform a dimensionality reduction, the transformation can be seen as orientation changing only. In that case, if an active direction transformation is created according to the rules of the previous chapter such that all inputs are transformed into a unit hypercube with equal dimension, the transformation function is injective and maps from Ω to $T \subseteq \Omega_r$. Fig. 4.1 illustrates this case.

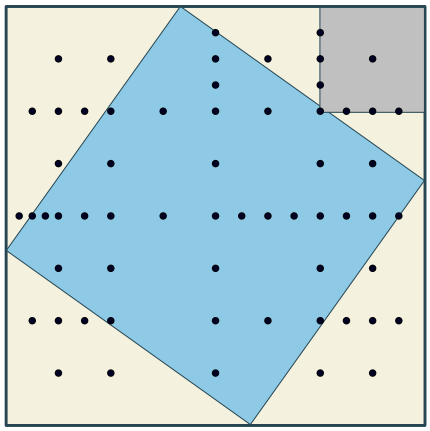


Figure 4.1: Exemplary case of an active direction transformation with $r = d = 2$. The beige □ square represents $\Omega = \Omega_r$, the black dots are the grid points of a sparse grid surrogate, and the rotated light blue □ square represents T. The grey □ square in top right corner represents the basis function support of the grid point in its center.

It is then possible to use normal sparse grid interpolation to create a surrogate for the transformation if we define how to treat grid points outside of T. For that we can either ensure that grid points outside of T never are created or define a rule on how to treat these out of domain function evaluations.

Restricting grid point creation in U is difficult as the grid point hierarchy, which dictates that every grid point has a parent, should be kept intact to work effectively with sparse grids. Thus, restricting grid point creation in U can prevent large areas in Ω being covered by grid points as a basic parent grid point might lie outside of T and would not be added to the grid. Note that even if there are many grid points that were generated outside of T, as seen in figure fig. 4.1, very few grid points are actually completely unused as the support of many grid point basis functions intersects at least slighly with T and therefore contributes to the overall approximation.

The alternative approach would require us to define rules on how to treat grid points outside of T, which when mapped to their corresponding function input, would lie outside of Ω. To evaluate the model function at a grid point for a transformed sparse grid, we need to inverse the transformation. As the transformation function is an automorphism of Ω when $r = d$, it is invertible for elements $t \in T$ and the inverse function $t^{-1} : T \mapsto \Omega$ can easily be constructed. We can also extend the inverse transformation t^{-1} to accept all elements $x \in \Omega_r$ without a problem with $t_{\text{ext}}^{-1} : \Omega_r \mapsto \mathbb{R}^d$, $t_{\text{ext}}^{-1}(x) = t^{-1}(x)$ to be able to inverse a grid point that lies outside of T. In a similar fashion, we also have to extend the model input domain to allow the model function to be evaluated

outside of Ω to create the interpolant surrogate $\hat{f}(x) = f_{\text{ext}}(t_{\text{ext}}^{-1}(x))$. A simple implementation could set all values outside of Ω to 0 with

$$f_{\text{ext},0}(x) \coloneqq \begin{cases} f(x), & \text{if } x \in \Omega \\ 0, & \text{else} \end{cases} \tag{4.3}$$

or alternatively just extend the input domain to allow the model function to be evaluated outside of Ω with $f_{\text{ext},f}(x) \coloneqq f(x)$. Whether the latter approach is feasible or not depends on the model function behaviour when leaving the original input domain Ω.

4.2.2 Regression

In case we perform a dimensionality reduction with $r < d$, we can no longer make use of sparse grid interpolation as the transformation is no longer invertible. We are then dealing with data that might be, depending on the model function and the transformation, very noisy and there also might be multiple different values for the same inputs with

$$t(x_i) = t(x_j),\ f(x_i) \neq f(x_j),\ x_i \neq x_j. \tag{4.4}$$

As a result, the transformed surrogate construction is usually an ill-posed problem for $r < d$ and we have to employ regression based methods to successfully create a surrogate for the transformed data. Fig. 4.2 illustrates this case.

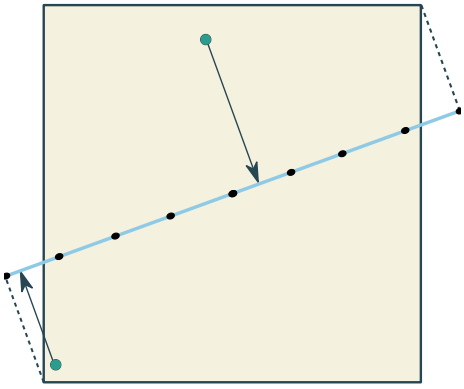


Figure 4.2: Exemplary case with $r = 1$ and $d = 2$. The surrounding beige □ square represents Ω, the light blue ■ line represents $\Omega_r = T$, and the black dots are the grid points of the one-dimensional sparse grid surrogate. The green ■ points are observations that are mapped from Ω onto Ω_r. Note that as Ω and Ω_r are distinct vector spaces, this figure visualizes the one dimensional Ω_r as the elements crossing the center in the two-dimensional space.

Since the focus of this thesis lies on dimensionality reducing transformations and the regression approach is also able to handle the orientation changing case with $r = d$, we will primarily use the regression method for the sparse grid surrogate construction. For this, the original observations,

$$S = \{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\} \subseteq \Omega \times \mathbb{R}, \tag{4.5}$$

which were drawn randomly from the given input probability distribution ρ, are transformed to obtain the transformed observations,

$$S_t \coloneqq \{(t(x_1), f(x_1)), \ldots, (t(x_n), f(x_n))\} \subseteq T \times \mathbb{R}. \tag{4.6}$$

We then construct the surrogate $\hat{f}_t$ for the transformed observation set S_t using sparse grid regression (see sec. 2.5) on a grid point set X and basis functions $\phi_{\underline{l},\underline{i}}$ to obtain the sparse grid surrogate,

$$\hat{f}_t(x) \coloneqq \sum_{x_{\underline{l},\underline{i}} \in X} \alpha_{\underline{l},\underline{i}} \phi_{\underline{l},\underline{i}}(x). \tag{4.7}$$

4.3 Multifidelity surrogates

Compared to sparse grid interpolation, the regression method takes longer to construct a well fitting surrogate because an optimization problem that involves all grid points has to be solved. Furthermore, the regression method also requires parameter calibration since there are many configuration parameters available that can significantly affect the resulting surrogate. Given a set of possible regression configuration parameter combinations, finding the best combination can become very computationally expensive as every parameter combination has to be evaluated to determine the best one. To mitigate this problem, we employ the concept of multifidelity simulations [**RWEH06**; **FSK08**], i. e., reducing the cost of parametrization by using low-fidelity sparse grids and datasets to evaluate a parameter combination and only using high-fidelity sparse grids and datasets when creating the final surrogate using the determined optimal parameters.

A low-fidelity sparse grid surrogate $\hat{f}^l$ has less grid points and employs no adaptivity compared to a high-fidelity sparse grid surrogate $\hat{f}^h$, i. e., the level of $\hat{f}^l$ is lower and no grid points are refined. It is vital that the used low-fidelity surrogates still stay comparable when evaluating their approximation errors for different parameter combinations. They don't have to be representative of the high-fidelity approximation error but should be indicative of the comparative quality of their high-fidelity surrogates, i. e., it should hold that

$$\varepsilon(\hat{f}_1^l) < \varepsilon(\hat{f}_2^l) \Rightarrow \varepsilon(\hat{f}_1^h) < \varepsilon(\hat{f}_2^h). \tag{4.8}$$

This multifidelity approach will also be heavily utilized in later chapters. The ability to evaluate transformed surrogates quickly is especially critical for random active direction transformation (see sec. 3.5) because it allows for the generation and evaluation of many transformations to choose the best from.

4.4 Operations on transformed surrogates

Now that we are able to construct a transformed surrogate and approximate the model function with $f(x) \approx \hat{f}_t(t(x))$, we can look at the differences when working with transformed sparse grid surrogates compared to using sparse grid interpolation surrogates without employing an input transformation function.

Differentiation Assuming that the surrogate uses a basis that can be differentiated easily, such as B-splines, the gradients can be approximated using the chain rule and the gradient function of the transformed surrogate to get

$$\nabla f(x) \approx \nabla \hat{f}_t(t(x)) = \nabla(\hat{f}_t \circ t)(x) = \left(\frac{\partial t(x)}{\partial x}\right)^T \frac{\partial \hat{f}_t(t(x))}{\partial x}. \tag{4.9}$$

Therefore, as long as the transformation function and the surrogate are differentiable, the transformed surrogate can be differentiated as well. This holds true for the primarily used active direction transformations.

Optimization To determine the approximate maximum $x^{\max} := \arg\max_{x \in \Omega} f(x)$ or minimum $x^{\min} := \arg\min_{x \in \Omega} f(x)$ of the model function, we can perform an optimization on the transformed surrogate in Ω_r. There already exist optimization algorithms for sparse grids, especially for B-spline

basis functions, presented in [**Val19**]. However, we have to keep in mind that not every element $x_r \in \Omega_r$ might actually have a corresponding input in $x \in \Omega$ that maps onto x_r with $x_r = t(x)$. Therefore a constrained optimization has to be performed on the surrogate with

$$x_r^{\max} := \arg\max_{x_r \in \Omega_r} \hat{f}(x_r),\ x_r \in T. \tag{4.10}$$

Afterwards, at least for dimensionality preserving transformations with $r = d$, we then can obtain the maximum $x^{\max}$ with $x^{\max} = t^{-1}(x_r^{\max})$, since $x_r^{\max} \in T$. One problem is that in the case of a dimensionality reducing transformation with $r < d$, a computed transformed surrogate maximum or minimum may be a set of many points in the original input domain Ω. Therefore another non sparse grid based optimization has to be performed on the set $t^{-1}(x_r^{\max}) = \{x \in \Omega \mid t(x) = x_r^{\max}\}$ to obtain a good maximum estimate.

Quadrature Given an input distribution ρ and transformation t, we can approximate the integral of the model function with

$$\begin{aligned} \int_\Omega f(x)\rho(x)\,\mathrm{d}x &\approx \int_\Omega \hat{f}_t(t(x))\rho(x)\,\mathrm{d}x \\ &= \int_{\Omega_r} \hat{f}_t(x_r)\left(\int_{t^{-1}(x_r)} \rho(x)\,\mathrm{d}x\right)\,\mathrm{d}x_r \\ &= \int_{\Omega_r} \hat{f}_t(x_r)\rho_t(x_r)\,\mathrm{d}x_r. \end{aligned} \tag{4.11}$$

We have to note that the transformed distribution ρ_t is usually a complex distribution, even in cases where ρ is a uniform distribution. Furthermore, similar to transformed optimization, integration of $\hat{f}_t$ over Ω_r is not representative in case $\Omega_r \neq T$. We therefore lose the ability to easily employ sparse grid based quadrature algorithms. However, since we are aiming to choose r to be a lot smaller than d, a Monte-Carlo based quadrature becomes a much better option for transformed surrogates as the curse of dimensionality is weakened that way.

5 Iterative transformations

We now covered all the required steps to approximate a model function with a transformed sparse grid surrogate. However, we can easily see that using just one transformed surrogate for approximation purposes is not as powerful as it needs to be for many models. Especially if the model function is a sum of terms itself, an iterative approach that generates a sum of multiple different transformed surrogates suits a model function better than a single one. We can extend the definition of intrinsic dimensionality, to obtain a concept for surrogate sums:

Definition 5.0.1 (Compounded intrinsic dimensionality)
A d-dimensional function $f(x_1, \ldots, x_d)$ has a compounded intrinsic dimensionality of $r \leq d$ if there exist p transformation functions $t_i : \Omega \mapsto \Omega_{r_i}$ and functions $f_{t_i} : \Omega_{r_i} \mapsto \mathbb{R}$ with $r = \max r_i$ such that

$$\forall x \in \Omega \colon f(x) = \sum_{i=1}^{p} f_{t_i}(t_i(x)). \tag{5.1}$$

Using multiple surrogates allows us to approximate model functions that consist of multiple terms with a low compounded intrinsic dimensionality. One example for this is the function $f(x_1, x_2) = \sin(2\pi x_1) + x_2$, which consists out of two terms with an intrinsic dimensionality of 1 and therefore also a compounded intrinsic dimensionality of 1.

5.1 Projection pursuit regression

When taking a look at other dimensionality reduction techniques to draw some inspiration from them, one insightful related method that shares some similarities is projection pursuit regression [**Hub85**]. The projection pursuit regression technique comes from statistics and is in its core idea related to our approach with active direction transformations. It tries to represent a dataset,

$$P \coloneqq \{(x_i, y_i)\} = \{(x_i, f(x_i))\}, \tag{5.2}$$

using the statistical model,

$$y_i = \beta_0 + \sum_{k=1}^{m} f_k(\beta_k^T x_i) + \varepsilon_i, \tag{5.3}$$

where β_0 is a constant, β_k are projection matrices, the f_k are lower dimensional functions, and the ϵ_i are the residuals. Using the β_k to map inputs onto a lower-dimensional space is very similar to our concept of creating active direction transformations, which is also based on transformation matrices. The functions f_k and projection matrices β_k of this additive model can be computed iteratively by finding a projection and constructing a surrogate repeatedly for the current error function,

$$e_k(x) \coloneqq f(x) - \sum_{i=1}^{k} f_k(\beta_k^T x). \tag{5.4}$$

A basic implementation of this approach can be seen in alg. 5.1.

function `projectionPursuitRegression`$(f, k_{\max})$

1: $e_0 = f$
2: **for** $k = 1, \ldots, k_{\max}$ **do**
3: $\quad \beta_k \leftarrow$ `determineBestProjection`(e_{k-1})
4: $\quad f_k \leftarrow$ `constructBestSurrogate`(e_{k-1}, β_k)
5: $\quad e_k \leftarrow e_{k-1} - f_k$
6: **end for**
7: $\hat{f}(x) \leftarrow \sum_{k=1}^{m} f_k(\beta_k^T x)$
8: **return** $\hat{f}$

Algorithm 5.1: Pseudocode of the iterative projection pursuit regression algorithm. Parameters are the model function f and the maximum amount of iterations $k_{\max}$.

Alternatively, we could also introduce an error exit condition to alg. 5.1 that exits the loop if the approximation error is small enough with $\varepsilon(e_i) < \varepsilon_{\min}$. We can already spot that the function `determineBestProjection` relates to chap. 3 where we extensively covered finding a suitable active direction transformation. Furthermore, `constructBestSurrogate` relates to chap. 4 that covered the creation of sparse grid surrogates for a given transformation.

5.2 An iterative approach

Going from this statistical model to our function approximation problem, we can formulate the same concept using the notation used in this thesis by removing the constant term β_0 and the residuals ε_i from (5.3) to get

$$\hat{f}(x) = \sum_{i=1}^{m} \hat{f}_{t_i}(t_i(x_i)), \tag{5.5}$$

where the t_i are active direction transformations, which are similar to the projections β_k^T and the $\hat{f}_{t_i}$ are the surrogate equivalents of the f_k. The functions $\hat{f}_{t_i}$ and transformations t_i can then be constructed iteratively by repeatedly executing a stepwise algorithm of first finding a transformation and then constructing a surrogate for the current error function,

$$e_k(x) = f(x) - \sum_{i=1}^{k} \hat{f}_{t_i}(t_i(x)). \tag{5.6}$$

Alg. 5.2 illustrates this iterative approach, which is similar to alg. 5.1, but has been adapted to fit our notation. It also handles the error function observations S_i and transformed observations S_{t_i} that are needed to construct a regression surrogate. The algorithm also includes a flexible exit condition in the form of the function `shouldExit()`. We usually either use a fixed amount of iterations exit condition or a relative convergence condition, which exits the loop if the new approximation error has not improved by a certain amount compared to the last iteration, i. e., it quits if $\varepsilon(e_i) > (1 - \eta)\varepsilon(e_{i-1})$ for some $\eta \in [0, 1]$.

function `transformedSurrogateSum`(f, n)

1: $e_0 \leftarrow f$
2: $S_0 \leftarrow \{(x_j, e_0(x_j)) \mid x_j \sim \rho\}_{j=1}^{n}$
3: $\hat{f} \leftarrow 0$
4: **for** $i = 1, \ldots, \infty$ **do**
5: $\quad t_i \leftarrow$ `generateBestTransformation`(S_{i-1})
6: $\quad S_{t_i} \leftarrow \{(t_i(x_j), e_{i-1}(x_j))\}_{j=1}^{n}$
7: $\quad \hat{f}_{t_i} \leftarrow$ `generateBestSurrogate`(S_{t_i}, t_i)
8: $\quad e_i \leftarrow e_{i-1} - \left(\hat{f}_{t_i} \circ t_i\right)$
9: $\quad S_i \leftarrow \{\left(x_j, e_{i-1}(x_j) - \hat{f}_{t_i}\left(t_i(x_j)\right)\right)\}_{j=1}^{n}$
10: $\quad \hat{f} \leftarrow \hat{f} + \left(\hat{f}_{t_i} \circ t_i\right)$
11: $\quad$ **if** `shouldExit`(S_{t_i}) **then**
12: $\quad\quad$ **break**
13: $\quad$ **end if**
14: **end for**
15: **return** $\hat{f}$

Algorithm 5.2: Pseudocode of the iterative transformed surrogate sum algorithm. Parameters are the model function f and the amount of used observations n.

Note that this algorithm can be applied to problems where either the original model function or only model observations are given. The continuous error function e_i is maintained and available in the algorithm only in case f is known. It is however not required in case only the input observations S_0 are given instead of f .

5.3 Generating the best surrogate

Surrogates can, depending on their type, have possibly many independent configuration parameters that can heavily influence the ultimate result and therefore the approximation error. The amount of possible configuration parameter combinations makes finding the best combination difficult as it would require the examination of a lot of combinations. In most cases, the actual surrogate has to be constructed with a given parameter combination to evaluate the suitability of them. The main driver of the runtime of the function `generateBestSurrogate` is therefore the amount of configuration parameters available. Common configuration parameters usually include various regression parameters, with the smoothing factor λ being the most prominent one. More formally, assuming that we have m independent configuration parameters where each one has a set of possible values C_i, the combined configuration set would be $C = C_1 \times \cdots \times C_m$ and the amount of necessary examinations would be $|C| = |C_1| \ldots |C_m|$.

Furthermore, the general structure and properties of the current regression problem at hand can change with each new iteration and the current error function e_i. Thus, configuration parameters have to examined in every iteration again because there might not exist a set of optimal configuration parameters for all iterations.

As a result, low-fidelity surrogates are employed during the parametrization phase that occurs right before the actual high-fidelity surrogate construction. Alg. 5.3 shows a simple implementation that uses low-fidelity surrogates for parameter evaluation to reduce runtime to determine the optimal low-fidelity configuration,

$$c^* := \arg\min_{c \in C} \varepsilon(\hat{f}_c^l). \tag{5.7}$$

This algorithm works fine presuming that the low-fidelity surrogates $\hat{f}_c^l$ are representative of the high-fidelity ones as already discussed in sec. 4.3. It is satisfactory as long as the actual approximation quality of the generated high-fidelity surrogate is close to the optimal configuration over all parameter combinations, i. e., in a high-fidelity setting it should hold that

$$\varepsilon(\hat{f}_{c^*}^h) \approx \arg\min_{c \in C} \varepsilon(\hat{f}_c^h), \tag{5.8}$$

where c^* was determined with low-fidelity surrogates.

function `generateBestSurrogate`(S_{t_i}, t_i)

1: $c^* \leftarrow ?$
2: $\varepsilon_{min} \leftarrow \infty$
3: **for** $c \in C$ **do**
4: $\quad \hat{f}_c^l \leftarrow$ `generateLowFidelitySurrogate`(S_{t_i}, t_i, c)
5: $\quad \varepsilon \leftarrow \varepsilon_{S_{t_i}}(\hat{f}_c^l)$
6: $\quad$ **if** $\varepsilon < \varepsilon_{min}$ **then**
7: $\quad\quad \varepsilon_{min} \leftarrow \varepsilon$
8: $\quad\quad c^* \leftarrow c$
9: $\quad$ **end if**
10: **end for**
11: **return** `generateHighFidelitySurrogate`(S_{t_i}, t_i, c^*)

Algorithm 5.3: Pseudocode of the surrogate generation algorithm. The parameters are the current error function observations S_{t_i} and the transformation function t_i. The the set of possible surrogate configurations C and Monte-Carlo-based error metric $\varepsilon_{S_{t_i}}$ are fixed.

Some other sparse grid construction properties, like the level, also have to be adapted to handle arbitrary surrogate dimensionalities as a the approximate grid point amount should be decoupled from the used dimensionality r, especially when comparing surrogates with differing dimensionalities. This means that instead of passing a fixed level ℓ when constructing a sparse grid surrogate, we instead specify a fixed grid point amount to control how much grid points the surrogate should approximately have and determine the most appropriate level and refinements from there. We are able to come very close to a targeted amount of grid points g by performing refinements a certain amount of times on a regular sparse grid of the largest possible level, i. e., generating a regular sparse grid with

$$\ell_{\max} = \left\{ \left| X_\ell^s \right| \leq g \mid \ell \in \mathbb{N} \right\} \tag{5.9}$$

and padding up the missing amount of grid points using refinements. That way, we can accurately compare approximation errors of sparse grid surrogates across all possible reduced dimensionalities because their grid point amount is equal.

5.4 Generating the best transformation

Until now, we assumed that the reduced dimensionality value r was given and did not cover ways on how to determine the optimal reduced dimensionality r^* during each iteration step. When covering active subspaces in sec. 3.4, we already mentioned that, even though the active subspace method provides a suggested criterion to determine the optimal cutoff dimension r^* by looking for the largest gap in the sequence of eigenvalues, this criterion is not always optimal and could be influenced by various factors. Furthermore, alternative transformation generation methods, like random active direction transformations in sec. 6.2.3, can not make use of such a cutoff criterion. We are therefore looking for a generalized approach to determine the best value for r_i^* in each iteration that is independent of the employed transformation type. One solution would be to use the exploratory look ahead approach, which we already used to find the best configuration parameters in previous section, to determine the best value for r_i^* in the same way. In alg. 5.4, we can see a simple implementation to generate the best transformation by testing a range of possible values for r.

function `generateBestTransformation`$(S_{i-1}, r_{\min}, r_{\max})$

1: $t_i^* \leftarrow ?$
2: $\varepsilon_{min} \leftarrow \infty$
3: **for** $r = r_{\min}, \ldots, r_{\max}$ **do**
4: $\quad t_{i,r} \leftarrow$ `generateTransformation`(S_{i-1}, r)
5: $\quad \hat{f}^h_{c^*} \leftarrow$ `generateBestSurrogate`$(S_{i-1}, t_{i,r})$
6: $\quad \varepsilon \leftarrow \varepsilon_{S_{i-1}}(\hat{f}^h_c)$
7: $\quad$ **if** $\varepsilon < \varepsilon_{min}$ **then**
8: $\quad\quad \varepsilon_{min} \leftarrow \varepsilon$
9: $\quad\quad t_i^* \leftarrow t_{i,r}$
10: $\quad$ **end if**
11: **end for**
12: **return** t_i^*

Algorithm 5.4: Pseudocode of the transformation generation algorithm. Parameters are the error function observations S_{i-1}, the minimum reduced dimensionality $r_{\min}$, and the maximum reduced dimensionality $r_{\max}$.

The actual implementation of the function `generateTransformation` is usually fixed and can use any of the transformation generation methods presented in chap. 3, for example random active direction transformations.

5.5 Visualizing the algorithm

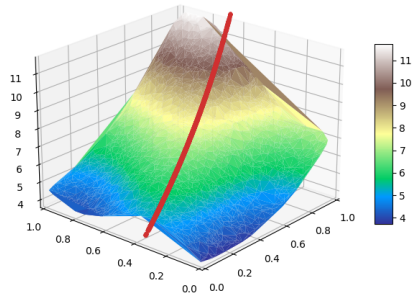
We visualized the iterative algorithm using a simple two-dimensional function,

$$f(x_1, x_2) = e^{x_1'+1} + \sin(2\pi x_2') + 1, \quad \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} +\cos(0.1\pi) & -\sin(0.1\pi) \\ +\sin(0.1\pi) & +\cos(0.1\pi) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{5.10}$$

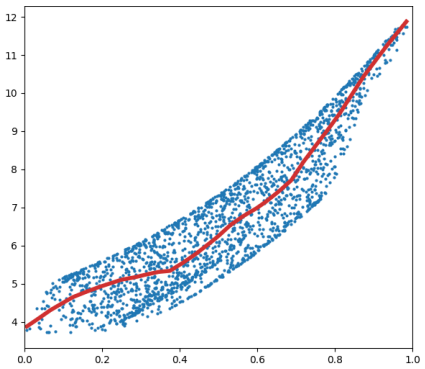
where the first component contributes the most to the total function value while the second one does not. It has a compounded intrinsic dimensionality of 1 and is not axis-aligned to further visualize how the algorithm handles multiple non axis-aligned function terms.

For every iteration, we take a look at the error functions e_{i-1} and at the regression problem that has to be solved by the surrogate generator. To better illustrate the surrogate construction, r is always set to 1, regardless of whether a value of $r = 2$ would make sense or not. Furthermore, the regression surrogate employs heavy regularization and is only plotted as far as it needs to be, i. e., the plot might not cover $[0, 1]$ as the amount of transformed samples in the boundary regions is very low. This also illustrates how the transformed input distribution changes from the original input distribution, a uniform distribution in this case. The active subspaces gradient inputs were approximated using finite differences, and the amount of samples n is set to 4000.

As we can see in fig. 5.1, the first algorithm iteration removes a big chunk of the original function error of $e_0 = f$ to reduce the approximation error in the first step. It identified the most active direction mostly right, constructed a one-dimensional active direction transformation, and was able to create a well fitting regression surrogate for the transformed observations S_{t_1}. The regression surrogate had to employ a lot of regularization, as seen in fig. 5.1b, since there is still some function output contribution by the sine term that will all be projected onto the first active direction.



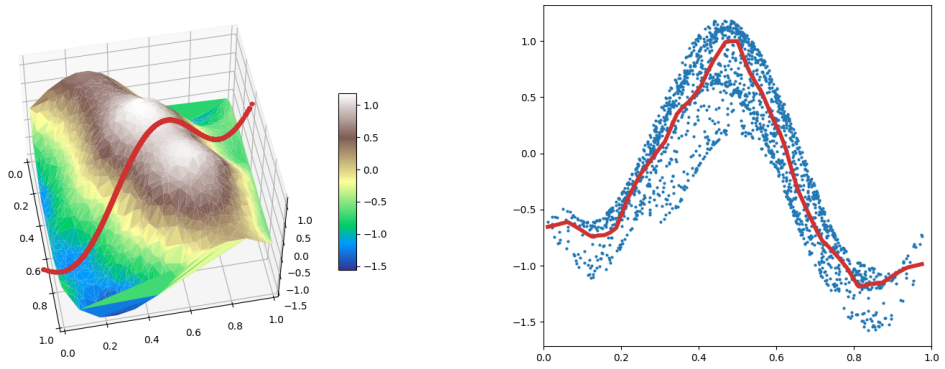
(a) Error function e_0 and its intersection with the first active subspace direction w_1 in red.



(b) Transformed input samples S_{t_1} in blue and the surrogate $\hat{f}_{t_1}$ in red.

Figure 5.1: First iteration of the iterative transformation algorithm.

In fig. 5.2, we can clearly identify the remnants of the rotated sine term that are the leftovers after the exponential term was mostly removed in the first iteration. However, we can also see that the leftover error function is not exactly equal to the sine term as there still exists some noise. In theory, an optimal method should be able to approximate f without resulting in a leftover error function after two iterations since it consists out of two terms and its compounded intrinsic dimension is 1. However, because the method is not perfect and encounters small errors from various sources throughout the algorithm, it can not approximate functions in an optimal way.



(a) Error function e_1 and its intersection with the first active subspace direction w_1 in red.

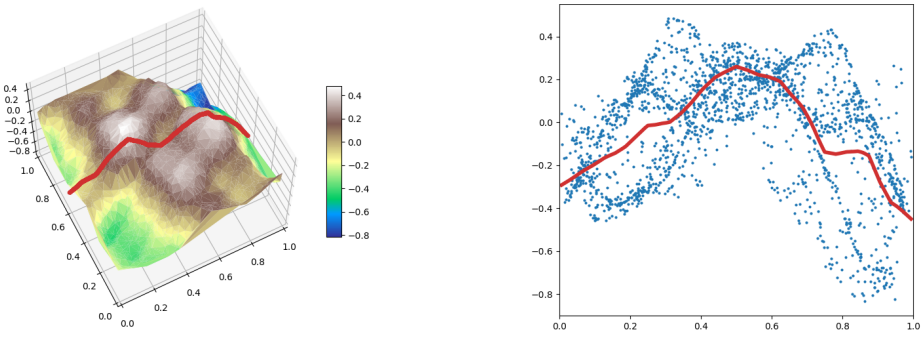
(b) Transformed observations S_{t_2} in blue and the surrogate $\hat{f}_{t_2}$ in red.

Figure 5.2: Second iteration of the iterative transformation algorithm.

These small errors are caused by inaccuracies inherent to the algorithm:

- The active subspace method not finding perfect active directions caused by the nature of the active subspace method itself (see sec. 3.4.4)
- The gradient inputs for the active subspace method being biased caused by using gradient approximation methods
- The regression method not producing ideal surrogates caused by too little or too much regularization or insufficient samples at the boundaries, as seen in fig. 5.1b at the left boundary

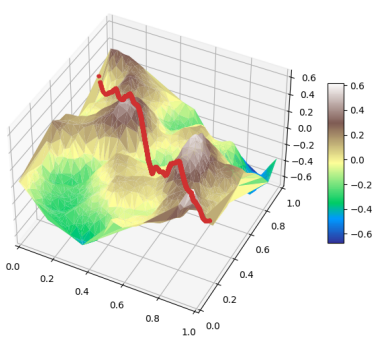
As evident in fig. 5.3 and fig. 5.4, the algorithm eventually hits a point where it can no longer effectively reduce the approximation error by adding another one-dimensional surrogate because the transformed input data is mostly two-dimensional noise that can't be regressed. Even though we can try to fit a heavily regularized surrogate to the transformed observations, the gain is minimal.



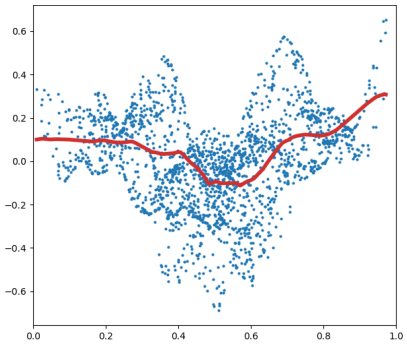
(a) Error function e_2 and its intersection with the first active subspace direction w_1 in red.

(b) Transformed input samples S_{t_3} in blue and the surrogate $\hat{f}_{t_3}$ in red.

Figure 5.3: Third iteration of the iterative transformation algorithm.



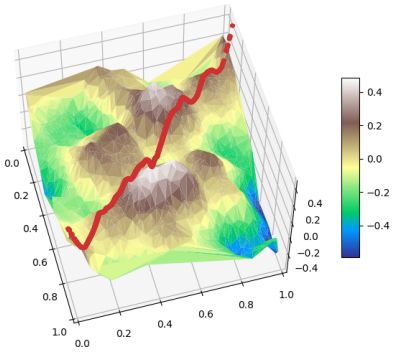
(a) Error function e_3 and its intersection with the first active subspace direction w_1 in red.



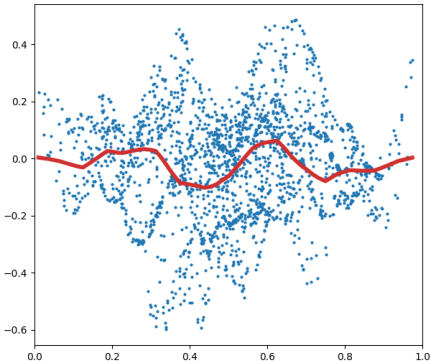
(b) Transformed input samples S_{t_4} in blue and the surrogate $\hat{f}_{t_4}$ in red.

Figure 5.4: Fourth iteration of the iterative transformation algorithm.

In fig. 5.5, we observe that we definitely hit a convergent state as the change compared to the previous iteration is minimal again. Therefore, we stop the algorithm after this iteration and return a sum of five one-dimensional sparse grid surrogates as an approximation for the model function. We can also choose to not include the last few transformed surrogates as they did not vastly improve the approximation.



(a) Error function e_4 and its intersection with the first active subspace direction w_1 in red.

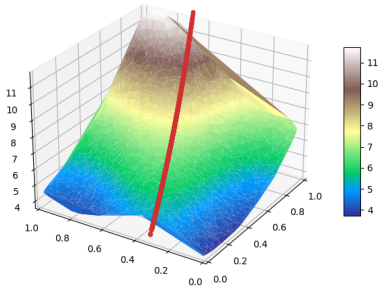


(b) Transformed input samples S_{t_5} in blue and the surrogate $\hat{f}_{t_5}$ in red.

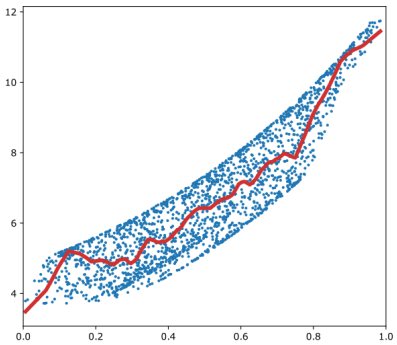
Figure 5.5: Fifth iteration of the iterative transformation algorithm.

5.6 Local vs global optimization

The previously introduced transformation generation algorithm uses a local optimization approach to determine the optimal transformation to use. However, in some cases this approach can lead the algorithm down a wrong path that will result in a worse approximation error in the end than otherwise possible. If we look at the example function from the previous section again and give the algorithm the additional choice of not employing a lot of regularization, the algorithm might fall in to the trap of overfitting when only looking ahead one iteration.



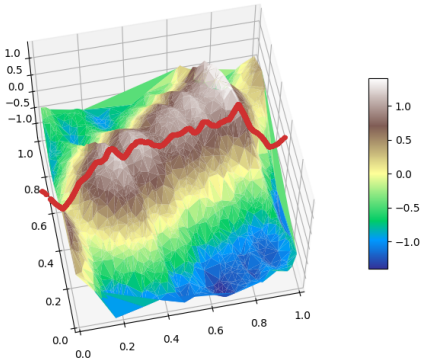
(a) Error function $e_0 = f$ and its intersection with the first active subspace direction w_1 in red.



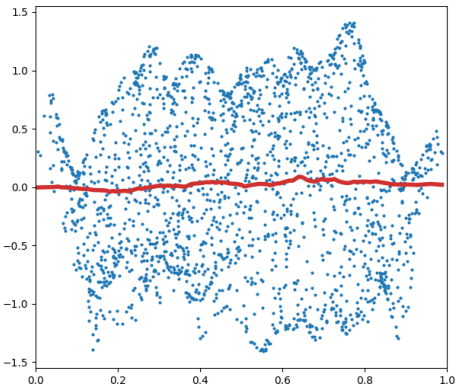
(b) Transformed input samples S_{t_1} in blue and the surrogate $\hat{f}_{t_1}$ in red.

Figure 5.6: First iteration of the iterative transformation algorithm with less regularization than before.

As we can see in fig. 5.6b, the surrogate is overfitted. At this stage, the approximation error is almost equal to the more regularized surrogate from fig. 5.1b due to the noise. However, the consequences of this badly fitted surrogate will propagate into the next iteration. Fig. 5.7b shows that no effective surrogate can be created as the input data is mostly just noise because the surrogate in the previous iteration was overfitted.



(a) Error function e_1 and its intersection with the first active subspace direction w_1 in red.



(b) Transformed input samples S_{t_2} in blue and the surrogate $\hat{f}_{t_2}$ in red.

Figure 5.7: Second iteration of the iterative transformation algorithm with less regularization than before.

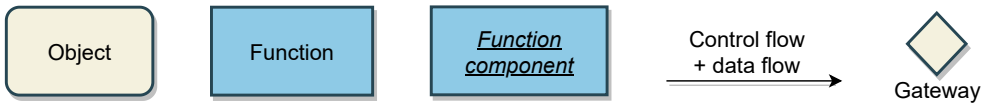
One solution would be to instead look ahead more than one iteration in cases where overfitting is possible. Instead of effectively performing only one look ahead iteration, we could perform i look ahead iterations to better gauge the quality of the produced transformation with regards to the resulting approximation error of the surrogate sum. To guarantee fast execution speed, we would have to use low-fidelity surrogates when looking ahead.

A more simple solution would be to just very carefully treat the smoothing factor λ and prefer more regularization over less regularization. However, this might not deliver optimal results as in some cases, especially if the last iteration is performed, a lower regularization factor might deliver better results.

6 Implementation

The last step missing before we can apply the transformed sparse grid technique to practical models is the actual implementation since all required theoretical aspects have been covered. In this chapter, we will take a look at the practical implementation of the iterative algorithm described in chap. 5 and model its various components using component diagrams to describe the general structure and possible configuration parameters. Such a formal description of the whole implementation will make the experimental results presented later as transparent as possible because it will list and explain possible algorithm configuration parameters that will impact the end result. The used notation for these component diagrams is shown in fig. 6.1.

Notation



Elements

Object: An object can represent any kind of data, for example a model function, a set of function observations, a transformation function, a surrogate, and more.

Function: A function takes some inputs in the form of objects, performs some kind of operation on them, and outputs an object.

Function component: A function component is a also function where the actual implementation can be exchanged for many different function implementations of the same interface.

Control flow + data flow: Arrows signal control flow and also data flow.

Gateway: A gateway evaluates an attached condition and decides which outgoing control flow to follow.

Figure 6.1: Notation used for different elements in the component structure diagrams in this chapter.

The implementation uses the SG^{++} library [1] [**Pfl10**] as a foundation and adds transformation capabilities on top of it. It makes use of many elements of the SG^{++} library and is basically implemented exactly as described in this chapter.

[1] The source code of the SG^{++} library is publicly available at https://github.com/sgpp/sgpp

6.1 Transformation pipeline

The complete transformation process is modeled and implemented as pipeline that is made up of different components to allow for maximum flexibility. The pipeline structure is visualized in fig. 6.2. Every pipeline iteration has the goal of adding another transformed surrogate to the current sum of surrogates to further improve the model approximation. Many pipeline function components can be exchanged for various different implementations that can also be customized by passing configuration parameters.

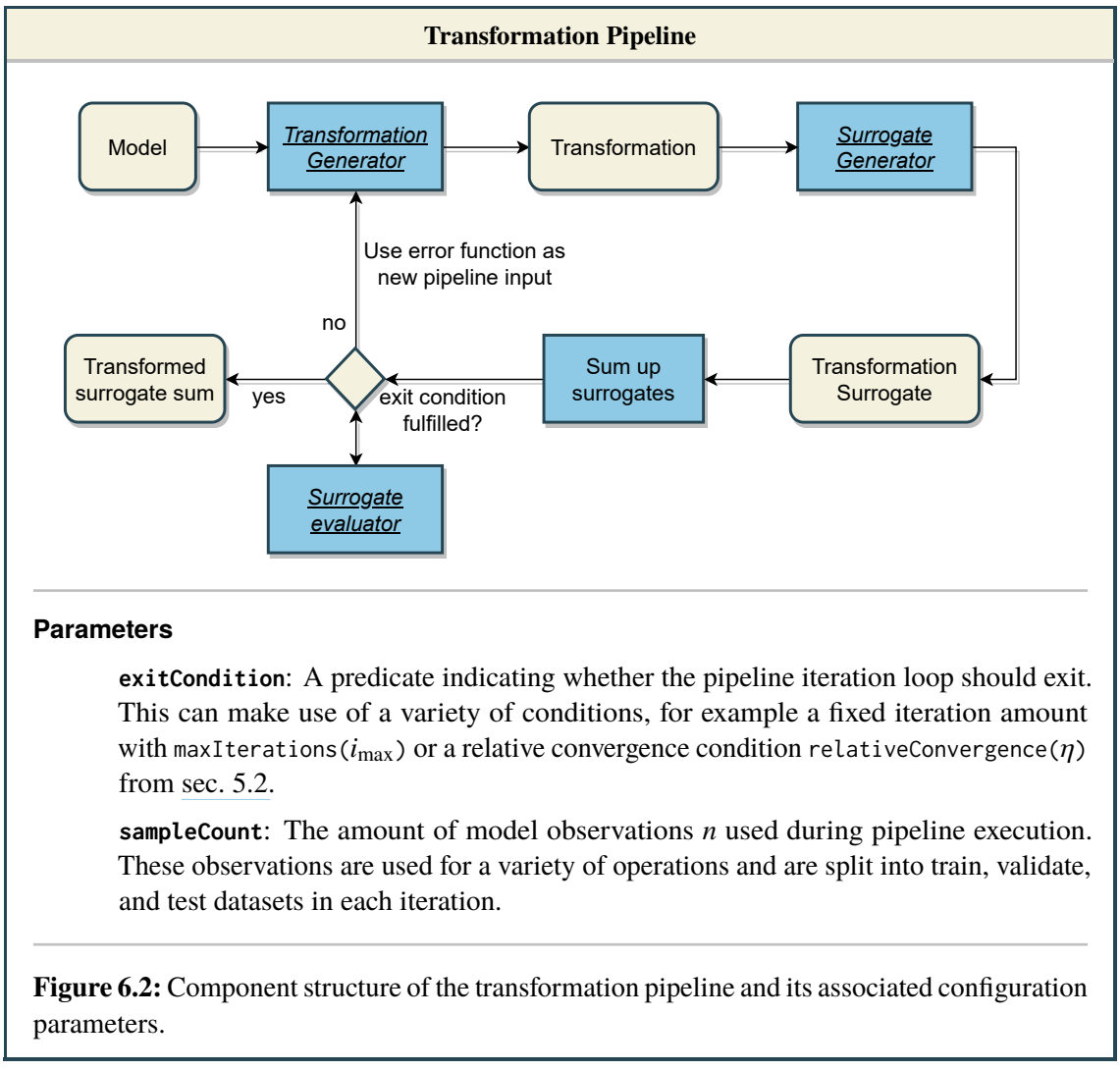


Figure 6.2: Component structure of the transformation pipeline and its associated configuration parameters.

The most important component of the transformation pipeline with regards to achieving good approximation results is the transformation generator component (sec. 6.2) as the quality of a single transformation will affect all later iterations. The generated transformation is fed into a surrogate generator component (sec. 6.3), which will generate the transformed surrogate. Using the appropriate surrogate generator, any type of surrogate, such as sparse grids or RBFs, can be generated. The newly generated surrogate is then added to the sum of previously generated surrogates to obtain the new transformed surrogate sum. At the end of each iteration, we decide whether to exit the pipeline or continue with the resulting residual error function e_k as the new target function.

6.2 Transformation generators

The transformation generator component has the responsibility of finding a good transformation function to use for a pipeline iteration. We covered the creation of input transformations extensively in chap. 4, and transformation generators are a straightforward implementation of the covered methods.

6.2.1 Stream-based transformation generator

We saw previously that we can generate a whole family of transformations for every generated active direction matrix, which is usually accomplished by using different values for the reduced dimensionality r, i. e., cutting off less or more dimensions. To apply this concept and to couple it with other components introduced later on, this implementation makes use of so-called transformation stream generators. A transformation stream generator will generate as much different transformations as it can where the range of generated transformations is usually defined by a minimum and maximum value for r. Every transformation that is generated by a stream generator is first evaluated using a transformation evaluator (sec. 6.4.2) and then compared to the other transformations to choose the best transformation out of all in the end. The stream-based transformation generator component structure can be seen in fig. 6.3.

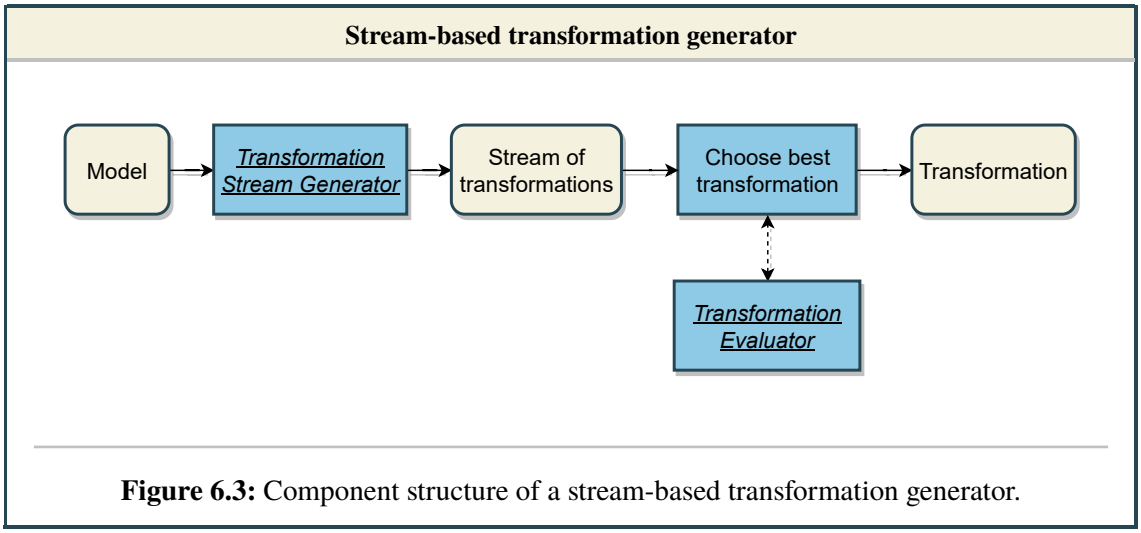


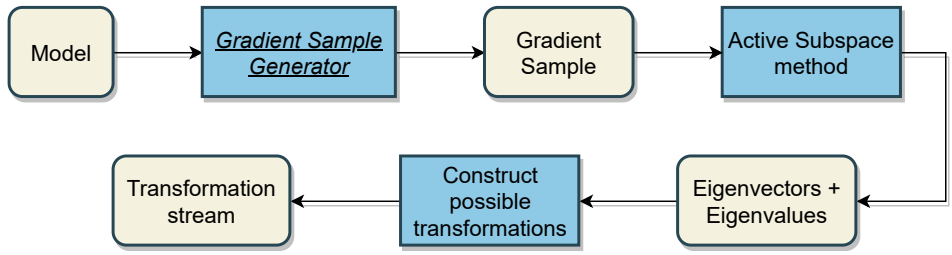
Figure 6.3: Component structure of a stream-based transformation generator.

Such a design allows us, in theory at least, to parallelize the transformation generation process as we can generate and evaluate all possible transformations in parallel.

6.2.2 Active Subspace stream generator

Transformation generation with the active subspaces method, as covered in sec. 3.4, is implemented as a transformation stream generator component since there are multiple different cutoff possibilities. The exact range of the cutoff dimensions can be customized using various parameters, as seen in fig. 6.4. However, the general order of active directions from w_1 to w_d is still fixed however as it is dictated by the magnitude of the eigenvalues.

Active subspace transformation stream generator



Parameters

`maxSampleCount`: The maximum amount of observations used to create a gradient sample.

`minDimensions`: The minimum amount of used directions, i. e., the lower bound for r.

`maxDimensions`: The maximum amount of used directions, i. e., the upper bound for r.

Figure 6.4: Component structure of an active subspace transformation stream generator and the associated configuration parameters.

The Monte-Carlo based active subspaces method requires gradient samples as inputs. These are generated by a gradient sample generator component. If the gradient function of the model function is not known, we can employ various gradient approximation algorithms that we already covered in sec. 3.4.1. Three implementations for gradient approximation are available to choose from as listed in fig. 6.5.

`finiteDifferences(`h`):` Gradient samples can be determined using finite differences if the runtime cost of d additional model function evaluations per sample is acceptable.

`randomNeighbours(`n'`):` Random neighbour approximation using n' random neighbours.

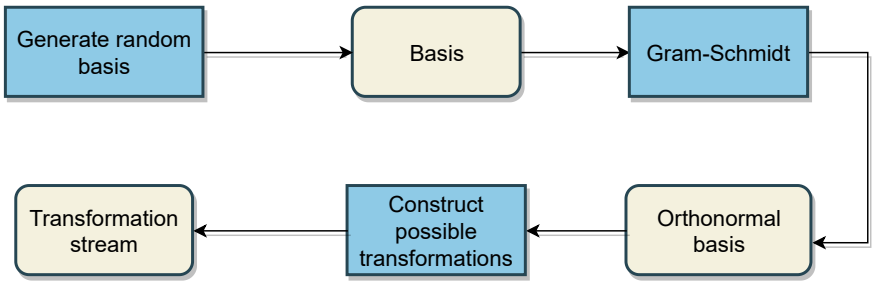
`nearestNeighbours(`n', m`):` Nearest neighbour approximation that selects the m nearest neighbours out of n' possible ones.

Figure 6.5: List of available gradient sample generator implementations.

6.2.3 Random stream generator

An alternative approach to constructing an active direction transformation is just generating a random one (see sec. 3.5). This random transformation generator is easy to implement, as shown in fig. 6.6, can generate new transformation functions almost instantly, and offers a purely exploratory approach to finding the best transformation. Random transformations are only viable if many of them are generated consecutively because the chance of finding a good random transformation in one try is very low. They are therefore mainly used in conjunction with iterative transformation generators (sec. 6.2.4).

Random transformation stream generator



Parameters

`minDimensions`: The minimum amount of used directions, i. e., the lower bound for r.

`maxDimensions`: The maximum amount of used directions, i. e., the upper bound for r.

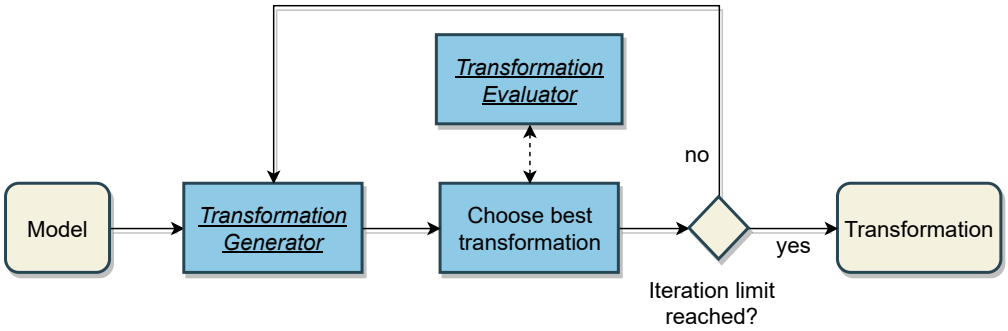
Figure 6.6: Component structure of an random transformation stream generator and the associated configuration parameters.

Note that the random active direction matrix generation is also be affected by the curse of dimensionality in the sense that it will become much harder to find a good random transformation in higher dimensionalities because there are many more degrees of freedom for a single active direction vector and also a lot more possible orderings of multiple active direction vectors. Since the ordering process can be seen as a combination of a set of orthonormal vectors without replacement, the amount of possible combinations given a set of d active directions is $\binom{d}{r}r!$, which increases sharply as r and d grows.

6.2.4 Iterative transformation generator

In some cases, for example when dealing with random transformation generation, it makes sense to introduce an iterative transformation generator component, which can be combined with any other transformation generator component. A transformation evaluator component (sec. 6.4.2) is responsible for assigning comparable quality ratings to each generated transformation such that the best one can be returned after the iteration limit is reached. This process is shown in fig. 6.7.

Iterative transformation generator



Parameters

`numIterations`: The amount of iterations that should be executed.

Figure 6.7: Component structure of an iterative transformation generator and the associated configuration parameters.

Note that this iterative approach is independent from the pipeline iterations, i. e., an iterative transformation generator executes all of its iterations during one pipeline step to determine the best transformation that should be used in the current pipeline step. To achieve fast execution of many iterations, we choose to work on low-fidelity surrogates during the iterations when interacting with transformation evaluator. This means that the approximation errors used to identify the best transformation do not necessarily come close to the actual high-fidelity approximation errors.

6.3 Surrogate generators

The last component needed for the pipeline implementation is the surrogate generator, which constructs the function $\hat{f}_t$ for given configuration parameters and transformed observations,

$$S_t = \{(t(x_i), f(x_i))\}_{i=1}^{n} \subseteq T \times \mathbb{R}. \tag{6.1}$$

The implementation is straightforward and closely matches the construction of sparse grid surrogates in chap. 4 and of radial basis function surrogates in sec. 2.7.

6.3.1 Sparse Grid surrogate generator

Sparse grid generators are implemented using the SG^{++} library [**Pfl10**]. The configuration parameters of a sparse grid surrogate generator contains all the usually required inputs like grid type, basis type, several adaptivity properties, and regression parameters. Instead of passing a fixed level ℓ as a parameter, we instead use the variable `approxGridPoints` to specify the target grid point amount of the surrogate as covered in sec. 5.3. This guarantees that sparse grids of different dimensionalities all contain an almost equal amount of grid points, which is very useful when performing comparisons. Furthermore, for sparse grid regression we similarly express the amount of training samples with the variable `trainSamplesPerGridPoint` to automatically link the amount of grid points and used training samples together.

6.3.2 RBF surrogate generator

Radial basis function surrogate generation is implemented with the help of ALGLIB [**Boc**], a numerical analysis and data processing library that supports fitting RBFs on arbitrary input data. Compared to sparse grids, RBFs do not have a lot of configuration parameters. The main parameters are the shape parameter ϵ and also a regularization factor λ. RBF surrogates can be created with the interpolation approach since they can interpolate from arbitrary model observations.

6.4 Evaluators

To judge the quality of transformations and surrogates, we introduce evaluator components. These take a transformation or surrogate as an input and output a real number that represents their quality and allows for comparisons with others.

6.4.1 Surrogate evaluator

A surrogate evaluator is a simple component that computes the surrogate approximation error by using validation observations and an error metric to compute the Monte-Carlo approximation error of the surrogate, e. g., the MSE, RMSE, or NRMSE. To compute the approximation error, we can either implement a simple train-validate observation split or make use of a more sophisticated cross-validation approach. Furthermore, this straightforward evaluation can also be customized, for example by adding penalty terms for certain surrogate features, e. g., the surrogate dimensionality or regression regularization parameters to steer the transformation pipeline, which relies on these evaluation results, in a certain direction.

6.4.2 Transformation evaluator

The role of a transformation evaluator is to take a newly constructed transformation, evaluate its quality, and make it comparable to other transformations. As discussed in sec. 5.4, evaluating the true quality of a transformation is only really possible by creating a transformed surrogate with it. We therefore generate a surrogate first and then make use of a surrogate evaluator to return the evaluation result. This is illustrated in fig. 6.8. While this is a pretty straightforward process, it can still be customized by changing the surrogate construction rules, for example by using a low-fidelity surrogate generator to speed up the evaluation.

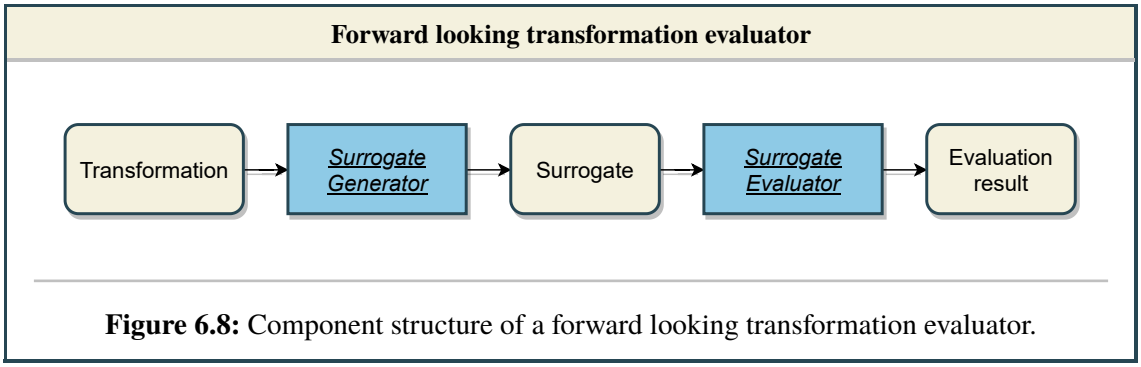


Figure 6.8: Component structure of a forward looking transformation evaluator.

6.4.3 Observation sample splitting

A central aspect of the transformation is pipeline is the handling of observations since they are used in almost every pipeline component. With the heavy use of the same observations across various stages, special care has to be taken to prevent all kinds of overfitting that can occur. For example, the generator components and evaluator components should all work on different observations such that no overfitting can possibly take place.

Furthermore, components that only work together, such as the surrogate generator and surrogate evaluator, are internally designed to coordinate their observation use with each other. For example, a cross validated surrogate evaluator partitions the observation samples multiple times and passes the train partition to the surrogate generator and then evaluates the quality using the associated validate partition. Therefore, the actual implementation control flow is not as simple as shown in fig. 6.9. The observation subset, which is passed down by each component to the next ones, is also randomly shuffled such that partitioning them into train and validation sets is not deterministic when performing multiple pipeline iterations.

In general, the pipeline works fine for any amount of observations as long there is at least a minimum viable amount. It also does not require any additional function evaluations to work if an appropriate transformation generator is chosen. This is an advantage compared to any interpolation based approach and makes the transformed surrogate technique very flexible.

6.5 Example pipeline

To give an example on how an actual transformation pipeline instance could look like, we will construct a simple one in this section. As many components are designed to be forward looking, we can think of the transformation pipeline as a collection of different sub-pipelines. The high-fidelity sub-pipeline is responsible for finding the the best and most expensive possible surrogate that will be returned after each complete pipeline iteration. It achieves this by usually evaluating the quality of multiple transformations from one family of possible transformations using a look-ahead approach. In other words, it executes a low-fidelity sub-pipeline that evaluates the transformation quality by effectively performing a whole iteration with low-fidelity surrogates for every possible transformation. This low-fidelity sub-pipeline has to be quickly executable as it gets executed potentially many times. The structure of an example pipeline can be seen in fig. 6.9.

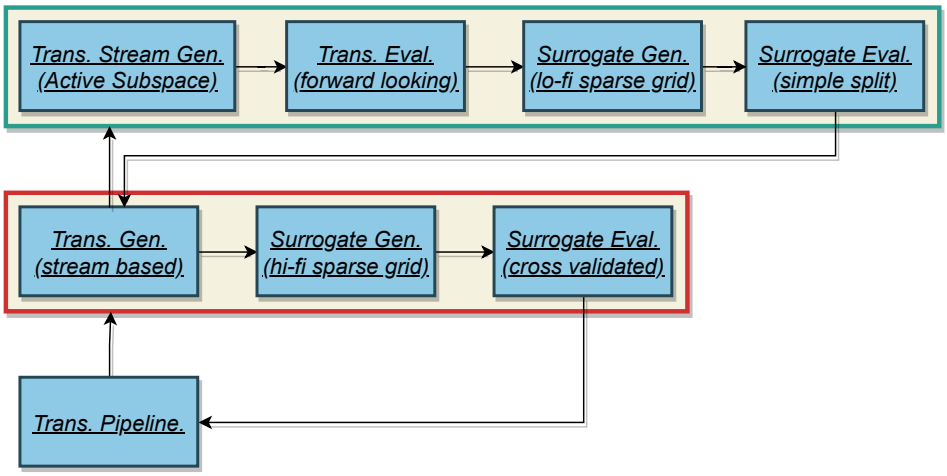


Figure 6.9: Structure and control flow of an exemplary transformation pipeline.
The red box represents the high-fidelity sub-pipeline components, and the green box identifies the low-fidelity sub-pipeline components.

7 Experiments

The construction process for transformed surrogates is realized through the transformation pipeline, which consists out of many different components. These components interact with each other by exchanging data and are therefore vulnerable to compounding inaccuracies and errors. In this evaluation chapter we investigate the qualities and properties of transformation pipeline components in an isolated environment such that we can obtain concrete data for each component that enables us to make use of the best possible pipeline configurations when performing experiments on real-world model functions in the next chapter. Of course, since not all components can be perfectly isolated, we will settle with best effort approaches that try to be as convincing as possible. We will first look at the low-dimensional Ishigami function followed by the higher-dimensional wing weight function.

All approximation error results throughout the chapter are computed using a Monte-Carlo NRMSE surrogate evaluator with 10^5 separate observations. Furthermore, the sparse grid surrogate generator uses a value of `trainSamplesPerGridPoint` $= 1$ when creating the surrogate. The concrete grid point amount of low-fidelity and high-fidelity surrogates varies and is explicitly mentioned in each section.

7.1 Ishigami function

We start off with the three-dimensional Ishigami function,

$$i(x) = \sin(x_1) + a \sin^2(x_2) + b x_3^4 \sin(x_1), \quad a = 7, b = 0.1, \tag{7.1}$$

which has been originally used for uncertainty quantification purposes in [**IH90**]. It has also been used to test sensitivity analysis techniques in [**SLY99**]. To get an idea of the general structure and importance of the inputs, we first apply the active subspace method on this function. It yields the following results:

i	w_i	λ_i	$\lambda_i / \sum_{i=1}^{d} \lambda_i$
1	$(+0.005,\ +0.999,\ +0.007)^T$	967.93	81.7%
2	$(+0.540,\ +0.003,\ -0.841)^T$	112.20	9.5%
3	$(-0.841,\ +0.008,\ +0.540)^T$	104.77	8.8%

Table 7.1: Computed eigenvectors, eigenvalues, and eigenvalue shares of the active subspace matrix for the Ishigami function using the Monte-Carlo method and 10^5 observations with finite differences.

Based on the values in table 7.1, we can see that the most active direction almost exactly corresponds to the second input dimension and is responsible for a big chunk of the average gradient change while the other two directions are less important and are also not axis-aligned. These two less active directions are however not negligible because their combined eigenvalue share still is 18.3%.

7.1.1 Dimensionality preserving transformations

While the main focus of this thesis lies on reducing transformations with $r < d$, we still take a short look at the quality of simple dimensionality preserving transformations with $r = d$, which are effectively only changing the orientation of the function input parameters. For the orientation change we use all computed active directions from table 7.1. In other words, we apply the active direction transformation t_{W_3} (see sec. 3.3.3) on the inputs. As discussed in sec. 4.2, with a few small modifications concerning model function evaluations outside of Ω, we can also apply the interpolation method to construct a transformed sparse grid surrogate. On one hand, the approximation error should be improved by the more optimal orientation of inputs compared to the non transformed interpolation approach. On the other hand, the unused parts of surrogate space (see sec. 4.1) effectively reduce the grid point resolution of the sparse grid surrogate and may make some grid points obsolete. Furthermore, the necessary model function evaluations outside of Ω might worsen the approximation quality if they deviate sharply from function values inside Ω and might create steep value differences at the boundaries of T. In this section, we are interested in answering the following basic questions:

1. How does the approximation error compare when applying an orientation changing transformation with $r = d$?
2. How big is the approximation error difference between interpolation and regression surrogates?
3. How is the approximation error affected when transformations and regression are used together?
4. How much difference is there between regular sparse grids and regular sparse boundary grids?

Fig. 7.1 shows the first comparison results of untransformed and transformed sparse grid surrogates created using interpolation and regression.

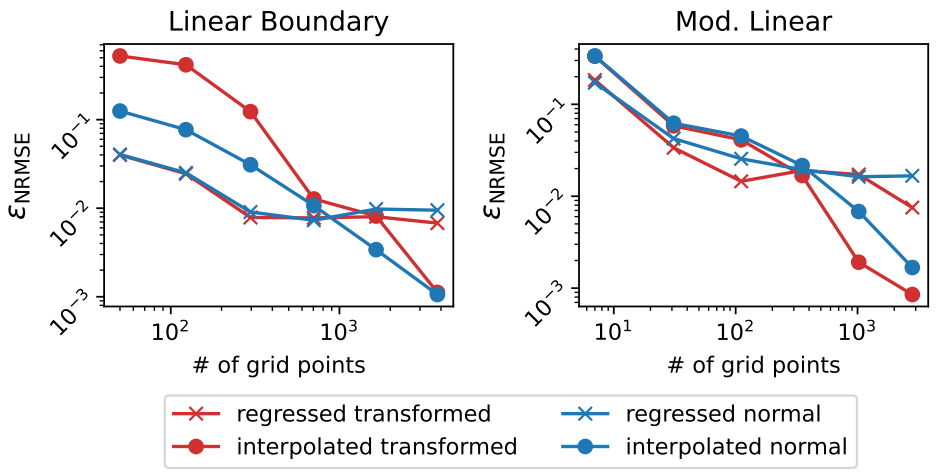


Figure 7.1: Approximation errors of transformed and untransformed sparse grid surrogates with $\ell \in \{2, \ldots, 7\}$ for the Ishigami function using boundary hat basis functions and modified hat basis functions. The transformations were constructed using the active directions from table 7.1 and only one pipeline iteration was performed.

As we can see in fig. 7.1, the pure orientation change does not lead to better interpolation results. In this concrete example, the penalty of the unused transformation space U evens out the gains from a more optimal orientation for sparse grid surrogates. As we do not perform a dimensionality reduction, even if it would be feasible, this results in a constructed surrogate space that covers a lot more than the required transformation space. In this case, the unused surrogate space share is $\tilde{u}_t = 0.44$, which means that T only makes up 56% of the surrogate space S. This effectively lowers the grid point resolution and results in some grid points being unused. Furthermore, the introduced regression penalty factor is approximately 10^1 compared to the interpolation method.

There are also no significant quality differences when using modified hat basis functions instead of a hat basis with boundaries. Therefore, we will only use the modified hat basis for the Ishigami function later in this evaluation section.

Based on these results, we can conclude that, even though there is no improvement with this method, there is also no significant downside even as $\tilde{u}$ is relatively large and regression is employed. This can be seen as a positive insight because it means that having a lot of unused space, which can not be avoided when using active direction transformations, approximately cancels out with the orientation improvements. Moreover, the additional use of regression only results in a slight increase of the approximation error. As the transformation process focuses on reducing transformations that have to deal with unused space and can only use regression, this means that the individual transformed sparse grid surrogates do not become worse just through the orientation change but allow for a much more flexible dimensionality reduction process.

7.1.2 Dimensionality reducing transformations

Next, we look at dimensionality reducing transformations with $r < d$. Note that for this low-dimensional model function, there is no real need to perform a dimensionality reduction as the curse of dimensionality does not really affect sparse grid surrogates for $d = 3$. We can therefore not expect to see any gain from applying a transformation with $r < d$. Even though reducing transformations should not be competitive for the Ishigami function, we are still interested in answering some basic questions:

1. By how much does dimensionality reduction increase the approximation error?
2. How much do the increased approximation errors correlate with the computed active subspace eigenvalues?

Based on the active subspace information from table 7.1, a one-dimensional reduction by cutting off the eigenvectors v_2 and v_3 looks feasible because the first active direction covers the majority of the eigenvalue share. However, as already discussed in sec. 3.4.4, the active subspace method can be biased and may not produce suitable active directions and eigenvalues. Therefore, the primarily used pipeline component uses a look ahead approach to evaluate the suitability of the computed active directions and corresponding eigenvalues instead of relying solely on the active subspaces results.

To better investigate the active subspace properties, we first generate transformations for $r = 1$ and $r = 2$. Since these are dimensionality reducing transformations, we can only create regression surrogates. Then we compare them to regression and interpolation surrogates with $r = 3$ to obtain the added approximation error.

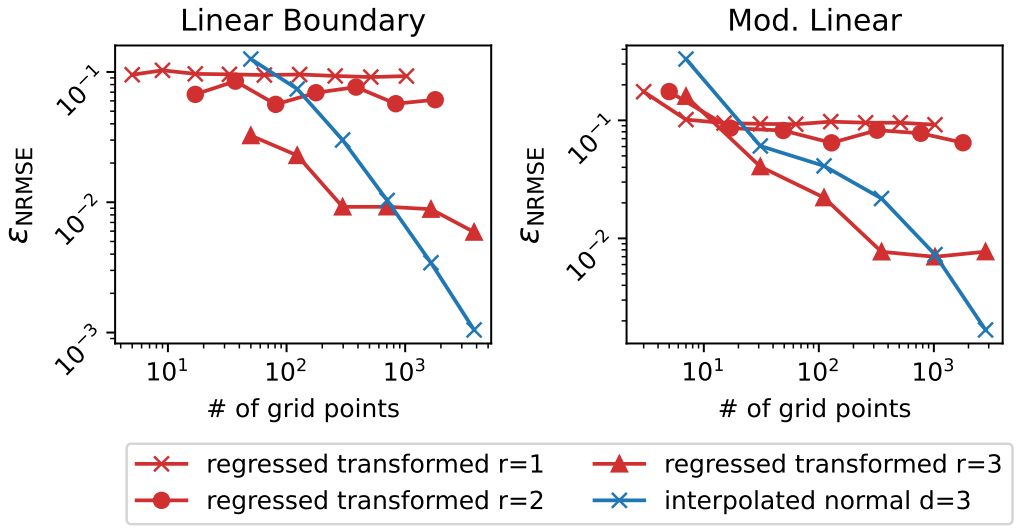


Figure 7.2: Approximation errors of various transformed sparse grid surrogates with $\ell \in \{2, \ldots, 7\}$ for the Ishigami function using boundary hat basis functions and modified hat basis functions. The transformations were constructed using the active directions from table 7.1 and only one pipeline iteration was performed. The interpolation result is included as a reference.

As we can see in fig. 7.2, the dimensionality reducing transformations lead to way worse results and do not provide competitive surrogates, which was to be expected. We see that the approximation error for the reduced surrogates reaches a convergent state very fast and an increase in grid points does not improve the surrogate quality. Furthermore, the actual approximation error of the reduced surrogates does not correlate with the active subspace eigenvalues. Even though the eigenvalue share of the first two eigenvectors is around 90%, the loss of the approximation quality when cutting of the most inactive eigenvector in the $r = 2$ case is already very large.

7.1.3 Multiple iterations

After we looked at the results of a mostly static pipeline in the sense that r was fixed, we now continue to look at a more dynamic pipeline approach. The goal here is to dynamically choose a suitable reduced dimensionality value r_i for each iteration such that we can obtain better results than with a purely static pipeline. By specifying how many grid points should be added in each iteration, we can easily implement a dynamic pipeline that first determines the best r_i by looking ahead and then creates a sparse grid surrogate with the given amount of grid points. The amount of grid points added per iteration can be customized with the goal of finding the best amount. We are interested in answering the following questions:

1. Do multiple iterations deliver better results than a single one?
2. Are there differences between the amount of grid points added per iteration?

Results for various grid point amounts are shown in fig. 7.3, which showcases dynamic pipeline results that use a different amount of grid points per iteration (gp/it).

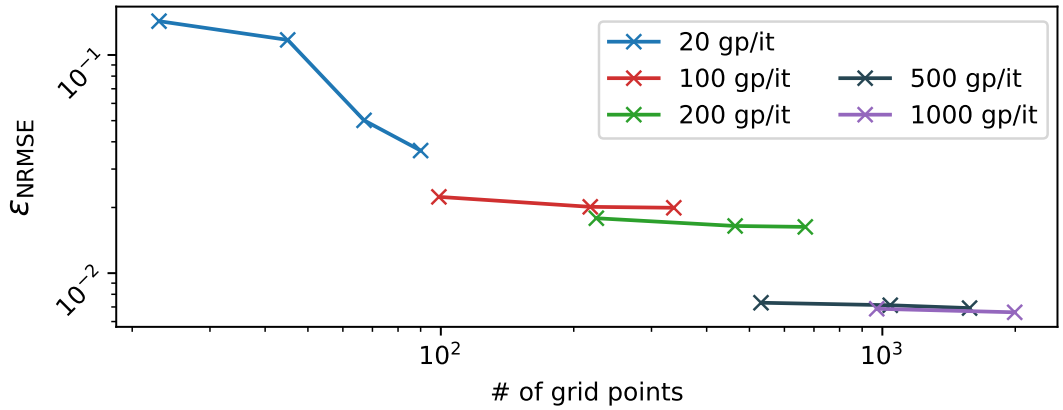


Figure 7.3: Approximation errors of various transformed surrogate sum approximations for the Ishigami function created using a dynamic pipeline.
In contrast to previous figures, every marker of a graph in fig. 7.7 now represents the result of a new iteration with the approximation error of the new surrogate sum and the grid point amount of all surrogate sum elements combined.

As we are able to see, the gains from using multiple iterations are, except in the 20 gp/it case, nonexistent. This might be explained by the fact that reducing transformations are very ineffective at approximating the function and therefore force the iterative algorithm to always construct three-dimensional surrogates as they still provide the best approximation quality. We therefore wait for the high-dimensional function evaluation later in this chapter to judge the true potential of the iterative algorithm because it can not deliver any gains for low-dimensional model functions like this.

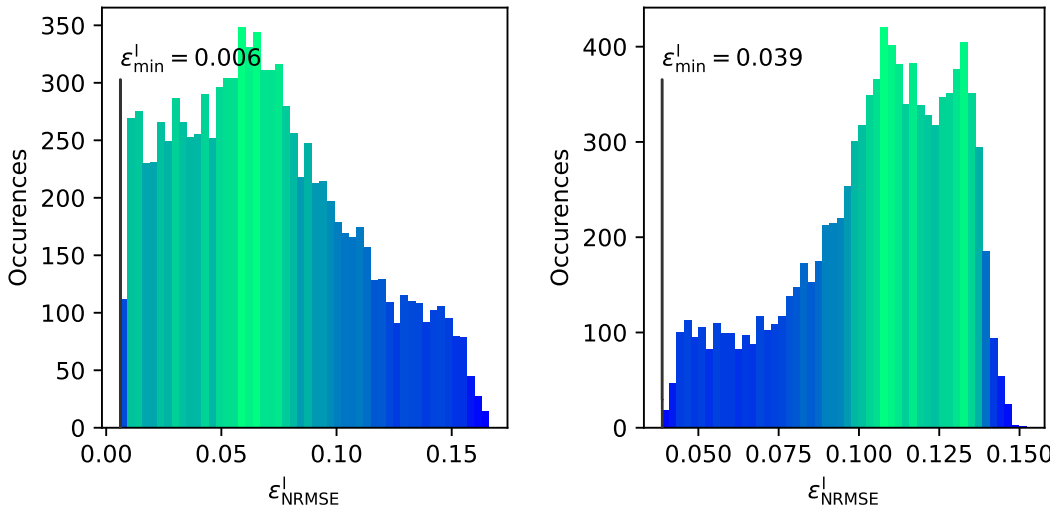
7.1.4 Random transformations

To verify the quality of the active subspace method compared to a true optimal active direction transformation, which we can't necessarily construct deterministically using active subspaces, one approach are random active direction transformations (see sec. 3.5). This generation method is not affected by any potential biases or compounding gradient approximation errors like the active subspace method is. To effectively make use of this strategy, it is vital that we can quickly generate, evaluate, and discard many random transformations, as more generated transformations will allow us to get closer to the potential optimal transformation. As the mainly used transformation evaluator component is forward looking, i. e., generating an actual transformed surrogate, we have to employ very low-fidelity components to achieve a large amount of runs. We are interested in answering the following questions:

1. How do the best found random transformations compare to active subspaces?
2. What does the approximation error distribution of the randomly generated transformations look like?
3. Are random active direction transformations a viable alternative for constructing transformations?

Once the algorithm has determined the random transformation with the lowest error based on the low-fidelity surrogates, we construct an actual high-fidelity surrogate and calculate the error again. The low-fidelity transformation errors are visualized using histograms and while the errors do not come close to the actual high-fidelity errors, the distribution of low-fidelity errors can provide valuable insights into the actual approximation error distribution since the low-fidelity surrogates are chosen to be comparatively representative.

As we will make use of low-fidelity and high-fidelity simulations, we will distinguish the resulting errors by denoting them by $\varepsilon^{\mathrm{l}}_{\mathrm{NRMSE}}$ and $\varepsilon^{\mathrm{h}}_{\mathrm{NRMSE}}$, respectively. For the Ishigami function in the low-fidelity case, we use a sparse grid surrogate generator with modified hat basis functions, and 250 grid points.. In the high-fidelity case, we increase the grid point amount to 10^3. We only perform one pipeline iteration as we already saw that additional iterations do not considerably improve the approximation.

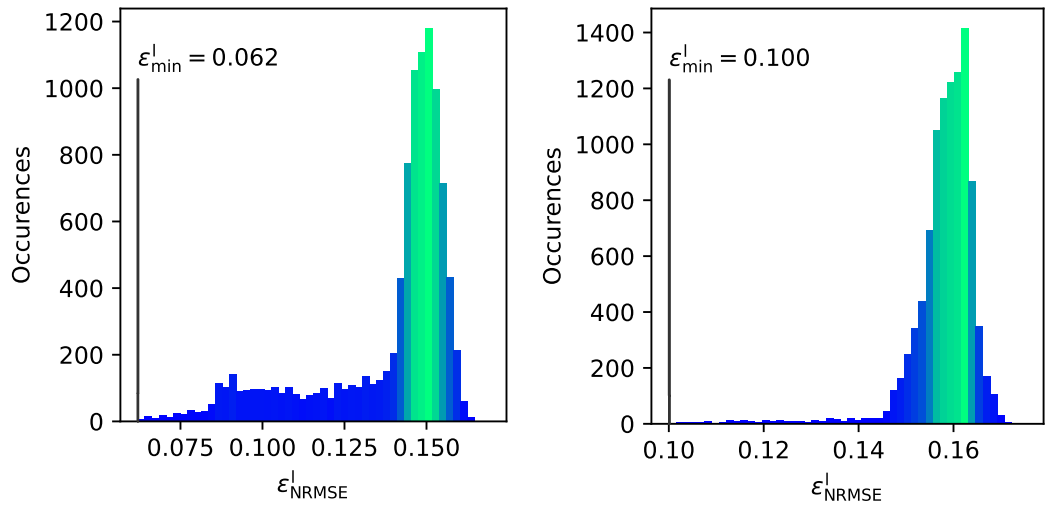


(a) Interpolated transformed sparse grid surrogates with $r = 3$. The resulting high-fidelity error is $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.0007$.

(b) Regressed transformed sparse grid surrogates with $r = 3$. The resulting high-fidelity error is $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.031$.

Figure 7.4: Histogram of low-fidelity approximation errors for 10^4 randomly generated transformations and the associated best possible high-fidelity error.

Fig. 7.4 shows the approximation error distribution of interpolation and regression surrogates for $r = 3$. Comparing the high-fidelity results with 10^3 grid points to the previous active subspace transformation results in fig. 7.2, we see that random transformations can deliver competitive results. We see that the distribution has a broad center of mass and almost no real tails, which makes it easier to find good transformations.



(a) Regressed transformed sparse grid surrogates with $r = 2$. The resulting high-fidelity error is $\varepsilon^{h}_{min} = 0.05$.

(b) Regressed transformed sparse grid surrogates with $r = 1$. The resulting high-fidelity error is $\varepsilon^{h}_{min} = 0.09$.

Figure 7.5: Histogram of low-fidelity approximation errors for 10^4 randomly generated transformations and the associated best possible high-fidelity error.

In fig. 7.5 with $r = 1$ and $r = 2$, the distribution shape changes drastically to a distribution with a high peak, very long tails and a negative skew. As a result, it requires more generation runs to obtain a good transformation because the probability of finding one far to the left of the peak is low. However, compared to the results in fig. 7.2, the best found transformation still delivers similar results to active subspaces.

In conclusion, random transformation generation proves to be a valuable alternative to active subspaces, at least for this example function and enough runs, as it comes close to the active subspace transformations. However, for higher dimensional functions, the required amount of random generation iterations to find an acceptable transformation is expected to grow very fast, as more dimensions will mean more possible directions and also orders of directions. Therefore, we expect the active subspace method to scale better with higher dimensions as it only requires exactly one run and an order of directions is implicitly given.

7.2 Wing weight function

Now that we investigated several important pipeline components with regards to their properties when being applied to a low-dimensional model function, we compare the results to a case study that involves a high-dimensional model function to look out for properties that can't be generalized from one low-dimensional case study. The model function covered in this section is the wing weight function [**FSK08**],

$$w = 0.036\, S_w^{0.758}\, W_{fw}^{0.0035} \left(\frac{A}{\cos^2(\Lambda)}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100 t_c}{\cos(\Lambda)}\right)^{-0.3} (N_z W_{dg})^{0.49} + S_w W_p, \tag{7.2}$$

which models the weight of a wing for a light aircraft based on ten input parameters that are defined in table 7.2. It has already been studied in the context of sensitivity analysis in [**MDS12**].

Variable	Description	Distribution
S_w	wing area (ft^2)	$\mathcal{U}(150, 200)$
W_{fw}	weight of fuel in the wing (lb)	$\mathcal{U}(220, 300)$
A	aspect ratio	$\mathcal{U}(6, 10)$
Λ	quarter-chord sweep (degrees)	$\mathcal{U}(-10, 10)$
q	dynamic pressure at cruise (lb/ft^2)	$\mathcal{U}(16, 45)$
λ	taper ratio	$\mathcal{U}(0.5, 1)$
t_c	aerofoil thickness to chord ratio	$\mathcal{U}(0.08\,0.18)$
N_z	ultimate load factor	$\mathcal{U}(2.5\,6)$
W_{dg}	flight design gross weight (lb)	$\mathcal{U}(1700\,2500)$
W_p	paint weight (lb/ft^2)	$\mathcal{U}(0.025, 0.08)$

Table 7.2: Input parameters of the wing weight model function.

To get an idea of the general structure and importance of the inputs, we apply the active subspaces method on this function as well, which yields the results of table 7.3.

i	λ_i	$\lambda_i / \sum_{i=1}^{d} \lambda_i$
1	28318.959	97.1%
2	478.48	1.6%
3	200.70	0.7%
4 . . . 10	< 71.0	< 0.02%

Table 7.3: Computed eigenvalues and eigenvalue shares of the active subspace matrix for the wing weight function using the Monte-Carlo method and 10^5 samples with finite differences.

Based on the values in table 7.3, we can see that there exists one dominant active direction that takes up the majority of the eigenvalue share and that there are two less important active directions as well. The seven other directions only make up a very small share of the total eigenvalue sum.

However, as we learned for the Ishigami function, these eigenvalues should only be used as rough indicators and not as reliable sources about the suitability for using transformed sparse grids.

7.2.1 Static pipeline

To start off, we first investigate the general quality of active direction transformations with multiple iterations and fixed r to get an idea of the model function properties with regards to its approximate compounded intrinsic dimensionality. To achieve this, we use the directions from table 7.3 and employ the relative convergence criterion with $\eta = 0$, i. e., we perform iterations until the approximation error does not improve anymore. The results can be seen in fig. 7.6.

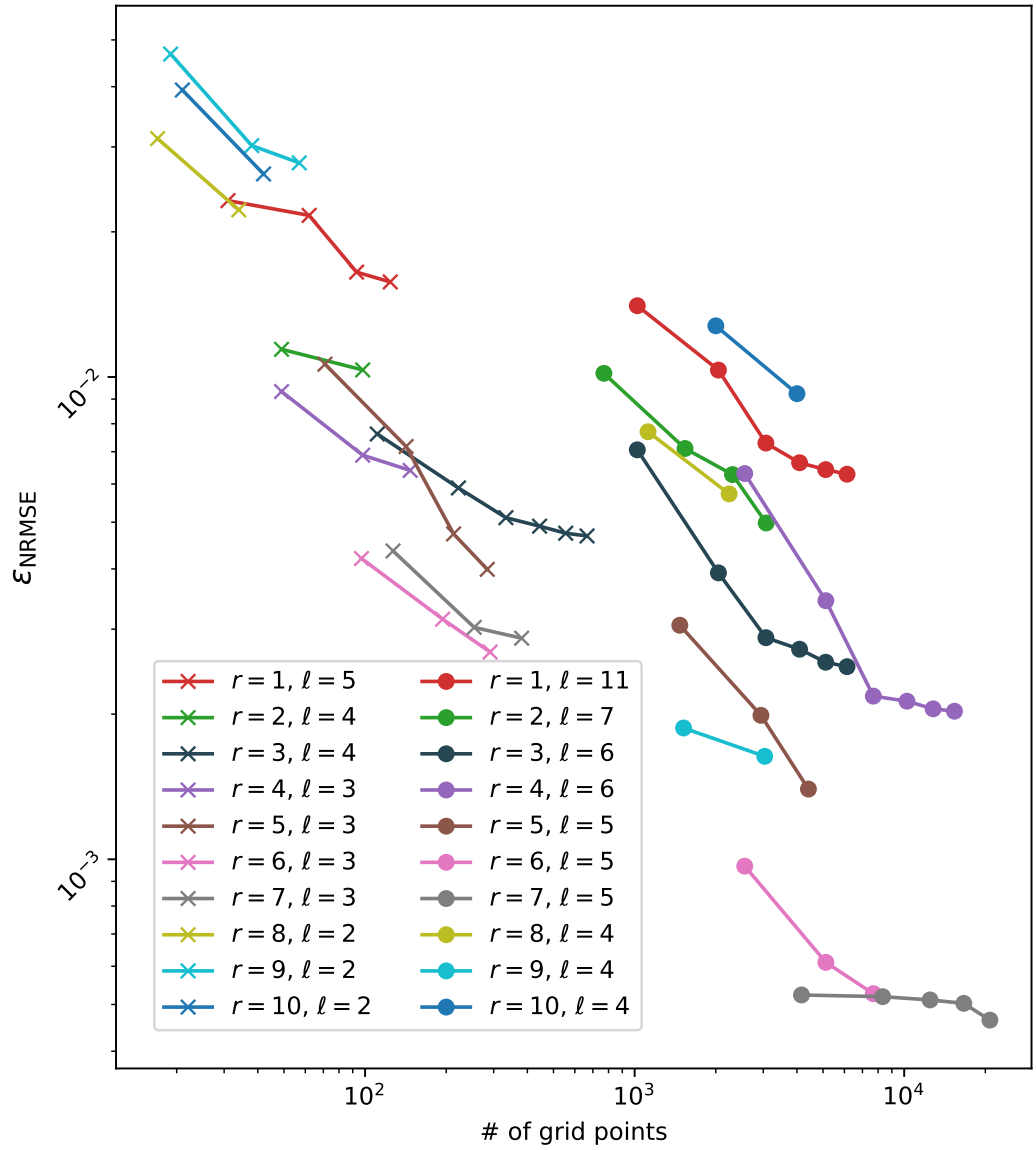


Figure 7.6: Approximation errors of various transformed surrogate sum approximations with fixed values of r for the wing weight function.

Every marker of a graph in fig. 7.6 represents the result of a new iteration with the approximation error of the new surrogate sum and the grid point amount of all surrogate sum elements combined. In the left half we see surrogate sum approximations using a coarse level and in the right half we see the fine level equivalents. The range between coarse and fine level surrogate sums for the same reduced dimensionality r gives insight into what part of the approximation error is based on the compounded intrinsic dimensionality of the model function and what part is based on the surrogate fidelities itself. For example, we see that for $r = 1, \ldots, 4$, the approximation error improvement from low-level to high-level surrogates is relatively small compared to higher values of r. This means that the main limiting factor in these cases is that the model function can not be approximated well with such low-dimensional surrogates. For $r > 4$, we see bigger improvement gaps when going from low levels to higher levels, which means that there is more potential approximation quality for higher-dimensional surrogates.

Moreover, we see that multiple iterations do improve the approximation, in contrast to the Ishigami function. The majority of approximation improvements can be achieved within the first three pipeline iterations as most error graphs converge after the third iteration. As we make use of static pipeline iterations only and therefore do not have the freedom to optimally choose the best reduced dimensionality in each iteration, we also might limit the effectiveness of multiple iterations.

Another interesting trend that can be observed is the behaviour of the approximation errors as r grows. The general approximation quality improves from $r = 1$ to $r = 7$ but hits a point at which the trend reverses and then worsens from $r = 7$ to $r = 10$. This can be explained by the general tradeoff between grid point resolution and dimensionality. As higher-dimensional grids effectively have to spend less grid points for each dimension when keeping the amount of grid points constant, this results in a tradeoff between grid points spent per dimension and reduced dimensionality. This optimal tradeoff point is hit at $r = 7$ in this case.

All in all, we can conclude that higher reduced dimensionalities up to $r = 7$ provide the best approximation error tradeoff when comparing the results for an equal amount of grid points This does not correlate at all with the active subspace results from table 7.3 and reinforces the evidence that we should not rely on the active subspace eigenvalues. The greater difference between low-fidelity and high-fidelity surrogates for larger values of r means that applying transformation generators that require a lot of low-fidelity surrogates, such as the random active direction generators, will take up more runtime as the amount of grid points for low-fidelity surrogates has to be increased to produce representative surrogates.

7.2.2 Dynamic pipeline

After we looked at the results of a mostly static pipeline in the sense that r was fixed, we now continue to look at a more dynamic pipeline approach. The goal is to dynamically choose a suitable reduced dimensionality r_i for each iteration such that we can obtain better results than with a static pipeline. By specifying how many grid points should be added in each iteration, we can easily implement a dynamic pipeline that determines the best r_i by looking ahead and then creates a sparse grid surrogate with the given amount of grid points. The targeted grid point amount can always be reached by performing refinements a certain amount of times. We differentiate the dynamic pipeline results by the spent amount of grid points per iteration (gp / it).

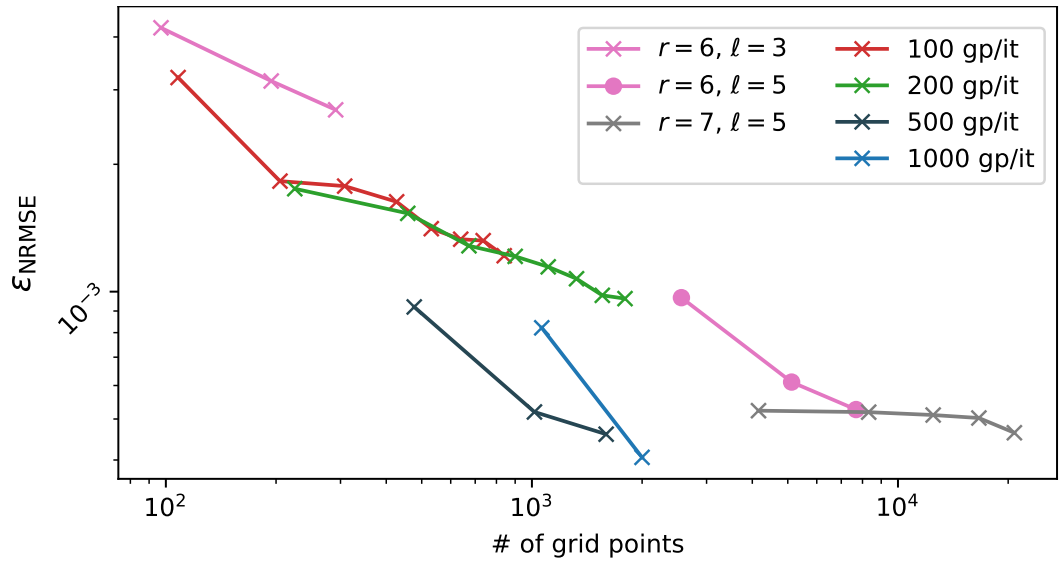


Figure 7.7: Approximation errors of various transformed surrogate sum approximations for the wing weight function created using a dynamic pipeline. The best results from the static pipeline are also included for reference.

Fig. 7.7 shows that by utilizing a more flexible dynamic pipeline, we can improve the approximation quality a little bit compared to a purely static pipeline. Furthermore, we also observe that there are differences in the approximation quality between the different amount of grid points spent per iteration as a pipeline with 500 gp/it delivers better results than one with 100 gp/it and 200 gp/it. Overall, when using the right amount of grid points per iteration, we are able to obtain some improvements. This is in contrast to the Ishigami function where multiple iterations did not improve the approximation at all. Therefore, when we have more possible values of r to choose from and when almost all values of r result in somewhat acceptable approximations, the dynamic pipeline approach becomes the better option.

7.2.3 Active subspace approximations

Another central aspect of the transformation pipeline is the quality of the generated transformations for any given value of r. In cases the transformation is constructed using the active subspace method, we are interested in the quality differences between the available active subspace approximation methods (see sec. 3.4.1) and want to ideally make use of the neighbour approximation methods because they don't require known gradients and no additional model function evaluations. We are interested in answering the following questions:

1. How much do approximated active subspaces differ from the true active subspaces?
2. Do the active subspace differences actually manifest themselves in the approximation error?
3. Which active subspace approximation method performs the best?

To judge the quality of approximated gradients, we first have to define an error metric for them:

Definition 7.2.1 (Gradient approximation error)
Let $S = \{(x_i, f(x_i)\}_{i=1}^n$ be a set of observations, and let $\widetilde{\nabla} f(x_i)$ be the approximated gradient for each observation point x_i. We then define

$$\varepsilon_{\text{grad}}(S) := \frac{1}{n} \sum_{i=1}^{n} \frac{\|\widetilde{\nabla} f(x_i) - \nabla f(x_i)\|_2}{\|\nabla f(x_i)\|_2} \tag{7.3}$$

as the gradient approximation error of the observation set.

In the context of active subspaces, the approximation error of the gradients is not the relevant metric to evaluate a gradient approximation method. Instead, the primary focus should be the resulting surrogate approximation error, as it is the only metric that matters in the end. Even though the active subspaces computation is based entirely on the average outer product of the gradients, there is not necessarily a strong link between $\varepsilon_{\text{grad}}$ and the quality of the resulting active subspace. While the individual approximated gradients can be far off, the average of the outer products of the gradients can still come close to the real one if the general trend of the gradients is still be reflected in the approximated gradients. Therefore, inaccuracies of single gradients can be averaged out, but systematic biases of the gradients will influence the approximated active subspace. The end result doesn't necessarily have to be bad however, as even the true active subspaces might not result in an optimal transformation.

At first glance, using some sort of approximated active subspace deviation looks like a good metric to evaluate approximated active subspaces. This would entail calculating some matrix norm of the approximation error compared to the true active subspace, that is to calculate $\|W - W^*\|$. However, this error calculation is not suitable for cases in which $\lambda_i \approx 0$, since the corresponding eigenvectors can vary greatly in that case. Also, if two eigenvalues λ_i, λ_j are similar in magnitude, the order of eigenvectors might switch arbitrarily and result in a huge matrix norm value. An error calculation of this type would therefore be potentially very unstable and not suitable. We therefore settle with the surrogate look ahead approximation error.

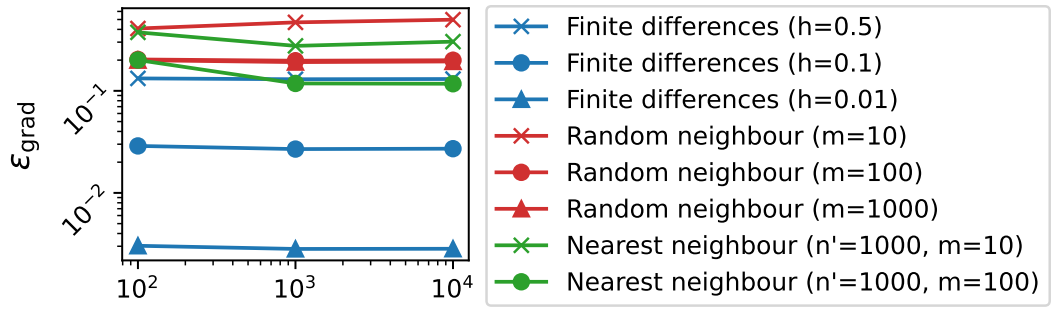
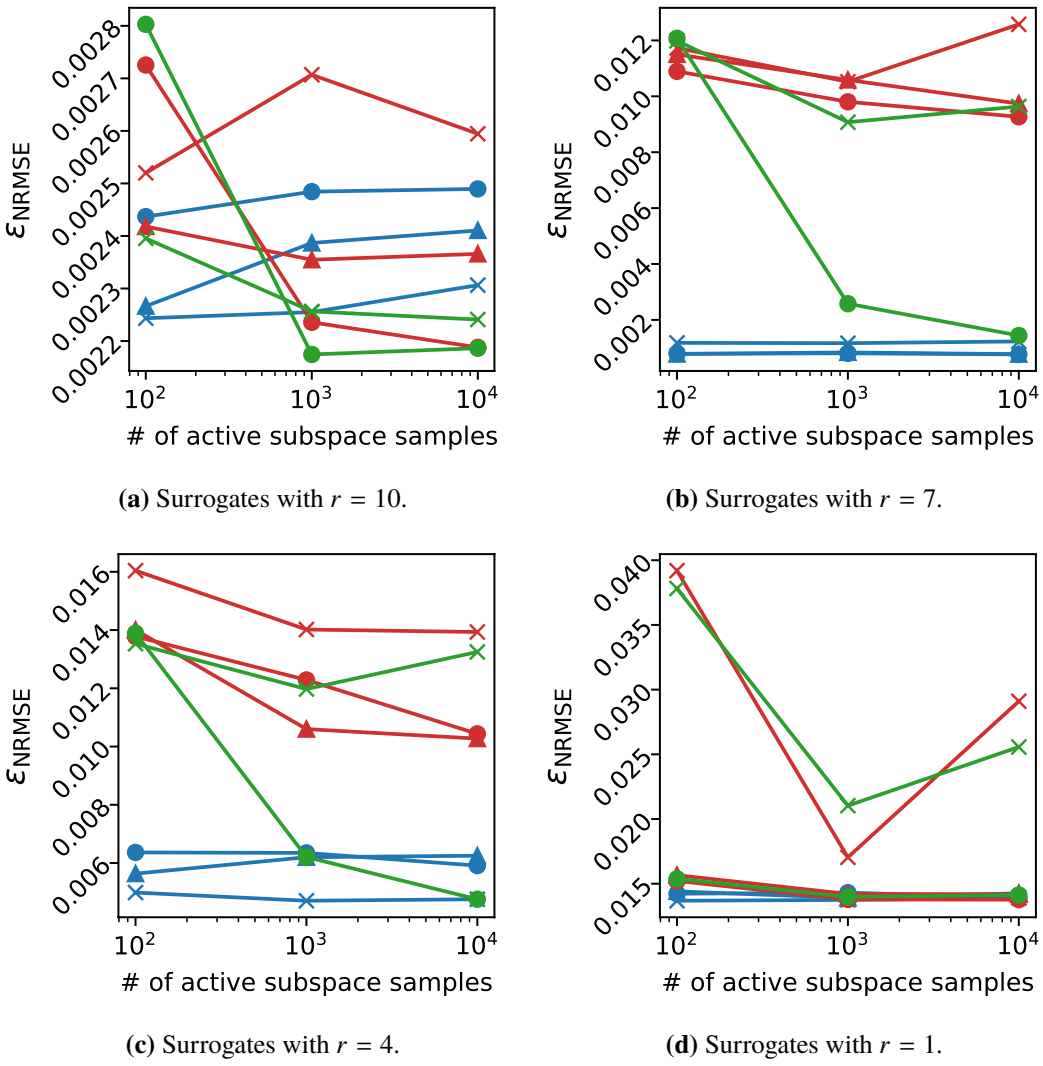


Figure 7.8: Gradient approximation errors of various gradient approximation methods averaged over 10 separate runs.

As seen in fig. 7.8, the finite differences approximation error converges towards zero as the spacing h goes down, which is to be expected. Furthermore, both neighbour methods do deliver approximately the same results with regards to $\varepsilon_{\text{grad}}$. Their gradient errors all lie in a range that is similar to the

finite differences approximation with a very big step size of $h = 5$, which at first glance makes them look not suitable. Different configuration parameters for the neighbour-based methods also seem to only improve the gradient approximation error by a small amount.



(a) Surrogates with $r = 10$.

(b) Surrogates with $r = 7$.

(c) Surrogates with $r = 4$.

(d) Surrogates with $r = 1$.

Figure 7.9: Surrogate look ahead errors for select values of r using active direction transformations, which were created by using various gradient approximation methods, averaged over 10 separate runs. The sparse grid surrogates were created using 10^3 grid points and modified hat basis functions.

Fig. 7.9 shows the actual performance of the different methods by using the look ahead error. It can be observed that the gradient approximation quality from fig. 7.8 does not exactly correlate with the constructed surrogate quality. While the finite differences method with $h = 0.01$ has the best error $\varepsilon_{\text{grad}}$ by far, the constructed surrogate does not perform better than basically the other finite differences methods and also some of the nearest neighbour methods. The random neighbour method lags a little bit behind in terms of the gradient errors and also surrogate errors in some cases

but is still in a comparable range to the other methods. Furthermore, the nearest neighbour method does deliver far better results than the random neighbour method as it shows a similar quality as the finite differences approximation with smaller step sizes in many cases when m is chosen correctly.

Especially important here are the results for $r = 7$ as we have seen that the best approximation errors are achieved for that reduced dimensionality. Overall, the finite differences method performs the best, followed by the nearest neighbour method with $m = 100$. The random neighbour method performs the worst in almost all cases. These results are good news as they indicate that the nearest neighbour approximation method, which is way cheaper with regards to required function evaluations, can deliver almost the same results as more expensive methods when the configuration parameters are chosen right.

7.2.4 Random active directions

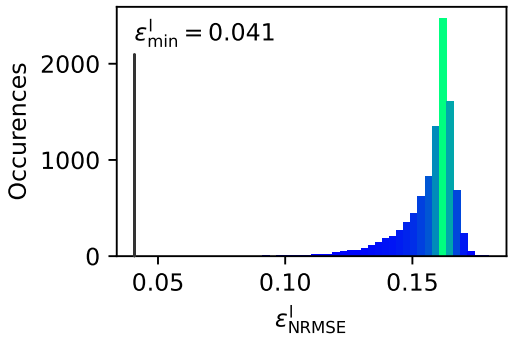
Similar to the first model function, we also try to find good active directions randomly as well. This time however, the results should be way worse because finding good active directions in ten dimensions is more difficult. We saw in the last sections that the wing weight function can already be approximated pretty well with only one transformed surrogate and further iterations do not massively increase the approximation quality. Therefore, we can already assess whether random active directions are potentially viable by only performing one pipeline iteration. Fig. 7.10 shows the low-fidelity approximation error distribution for random active direction generators and the high-fidelity approximation error of the best transformation out of all after one iteration.

When looking at the progression from lower dimensionalities to higher dimensionalities, we see that the distribution shape changes from having long tails and a negative skew to a more balanced bell shape, which makes finding a good transformation located on the left tail harder for higher dimensionalities. Histograms for $r > 6$ are omitted as they behave very similar with regards to their shape and also approximation errors compared to the histogram for $r = 6$.

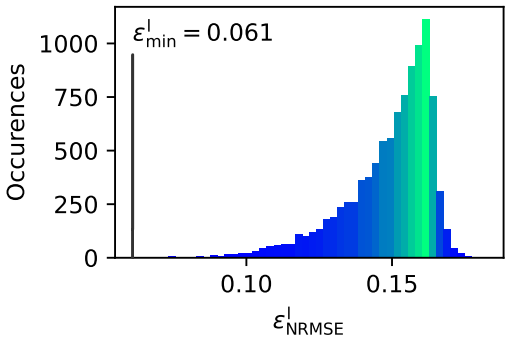
Moreover, we see that the low-fidelity and high-fidelity errors of the best one-dimensional generation result are better than the best ones for $r = 2, \ldots, 5$. We therefore come closer to a good active direction matrix for $r = 1$ than we do in higher dimensionalities. This behaviour can be explained by the fact that we require even more random generation runs than 10^4 to find a good transformation for higher values of r in this case. In conclusion, as the overall approximation errors in general do not come close the active subspace results, the random active direction transformation generation is not a viable alternative for the wing weight function.

7.2.5 Summary

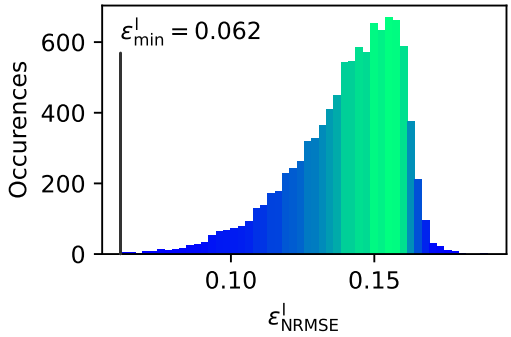
To summarise our findings, we can say that the active subspace method performs much better than the random active direction generation for this high-dimensional model function. This is caused by the sheer amount of possible active direction matrices and might be improved somewhat if more random generation runs were used. As the random approach is subject to the curse of dimensionality, it is not a feasible alternative for high-dimensional models. Active subspaces are therefore the way to go in higher dimensions.



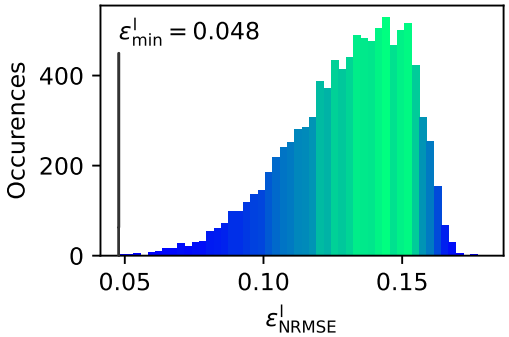
(a) Distribution for $r = 1$ with $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.040$.



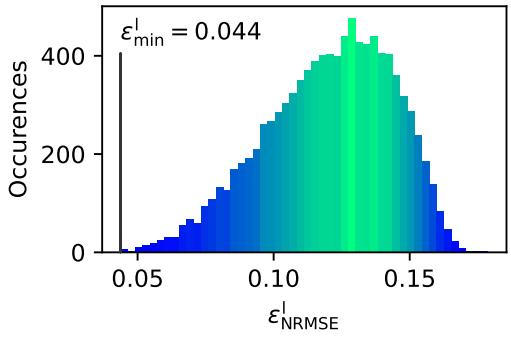
(b) Distribution for $r = 2$ with $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.060$.



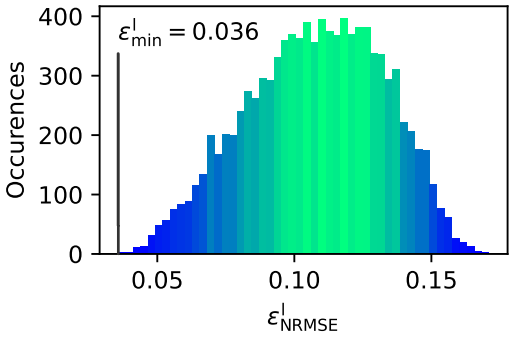
(c) Distribution for $r = 3$ with $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.061$.



(d) Distribution for $r = 4$ with $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.046$.



(e) Distribution for $r = 5$ with $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.041$.



(f) Distribution for $r = 6$ with $\varepsilon^{\mathrm{h}}_{\mathrm{min}} = 0.035$.

Figure 7.10: Histogram of low-fidelity approximation errors for 10^4 randomly generated active direction transformations.

In terms of determining the optimal reduced dimensionalities r_i, we saw that there are huge differences in the tradeoff between approximation error and grid points for various values of r_i. Some values of r_i just perform objectively worse than others and there is no quick heuristic, for example inspecting the active subspace eigenvalues, to gauge their expected quality. We therefore employ the look-ahead approach that works with low-fidelity surrogates because the quality differences for different r_i are already evident at pretty low grid levels.

The results for a baseline amount of 10^3 total grid points after two iteration are visualized in fig. 7.11. To arrive at 10^3 grid points after two iterations, each method spent 500 grid points per iteration. From there it is evident that random active directions are uncompetitive, static pipelines deliver acceptable results if r is chosen correctly, and a dynamic pipeline provides the best results and is also the easiest one to use.

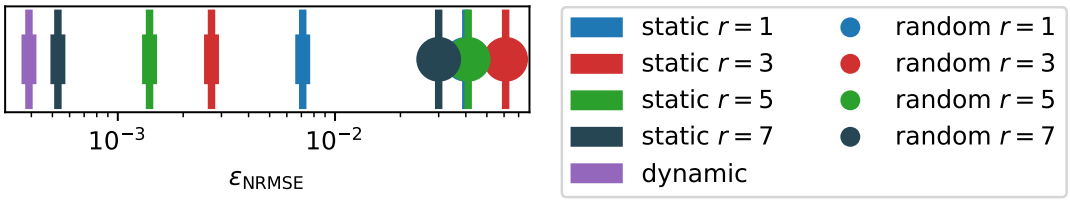


Figure 7.11: Comparison of select results for various methods with 10^3 total grid points.

8 Practical applications

Having investigated the properties of the transformed sparse grid technique in detail, we are now aiming to leverage this knowledge to construct function surrogates that are competitive in their approximation quality when being compared to other already existing techniques. In this chapter we will apply the transformed sparse grid technique on practical models such that we can, at least to some degree, compare the approximation quality to other techniques. Of course, these comparisons are limited in their nature as it is difficult to control for every property unique to an approximation technique across others.

8.1 Borehole function

The initial practical application that we will investigate is the borehole function, which modes the water flow through a borehole that was drilled through aquifers above and below a nuclear waste repository. It was first studied in the context of sensitivity analysis in [**HG83**]. The resulting output Q, which represents the flow in m^3/year, is given by

$$Q = \frac{2\pi T_u (H_u - H_t)}{\ln(r/r_w)\left(1 + \frac{2LT_u}{\ln(r/r_w) r_w^2 K_w} + \frac{T_u}{T_l}\right)}, \tag{8.1}$$

where the eight input parameters and their distributions are defined in table 8.1.

Variable	Description	Distribution
r_w	borehole radius (m)	$\mathcal{N}(\mu = 0.1,\ \sigma = 0.0161812)$
r	influence radius (m)	$\ln \mathcal{N}(\mu = 7.71,\ \sigma = 1.0056)$
T_u	upper aquifer transmissivity (m^2/year)	$\mathcal{U}(63070,\ 115600)$
H_u	upper aquifer potentiometric head (m)	$\mathcal{U}(990,\ 1110)$
T_l	lower aquifer transmissivity (m^2/year)	$\mathcal{U}(63.1,\ 116)$
H_l	lower aquifer potentiometric head (m)	$\mathcal{U}(700,\ 820)$
L	borehole length (m)	$\mathcal{U}(1120, 1680)$
K_w	borehole hydraulic conductivity (m/year)	$\mathcal{U}(9855,\ 12045)$

Table 8.1: Input parameters for the borehole model function.

The borehole function has been thoroughly investigated in the context of sparse grid interpolation in [**Reh21**] using B-spline basis function variants with sparse grids. B-splines are a solution to eliminating the discontinuities of the standard hat basis functions and are piecewise polynomials. Not-a-knot B-splines (NAK B-splines) on spatially adaptive sparse grids were shown to be very competitive compared to various other approximation techniques and serve as a good benchmark.

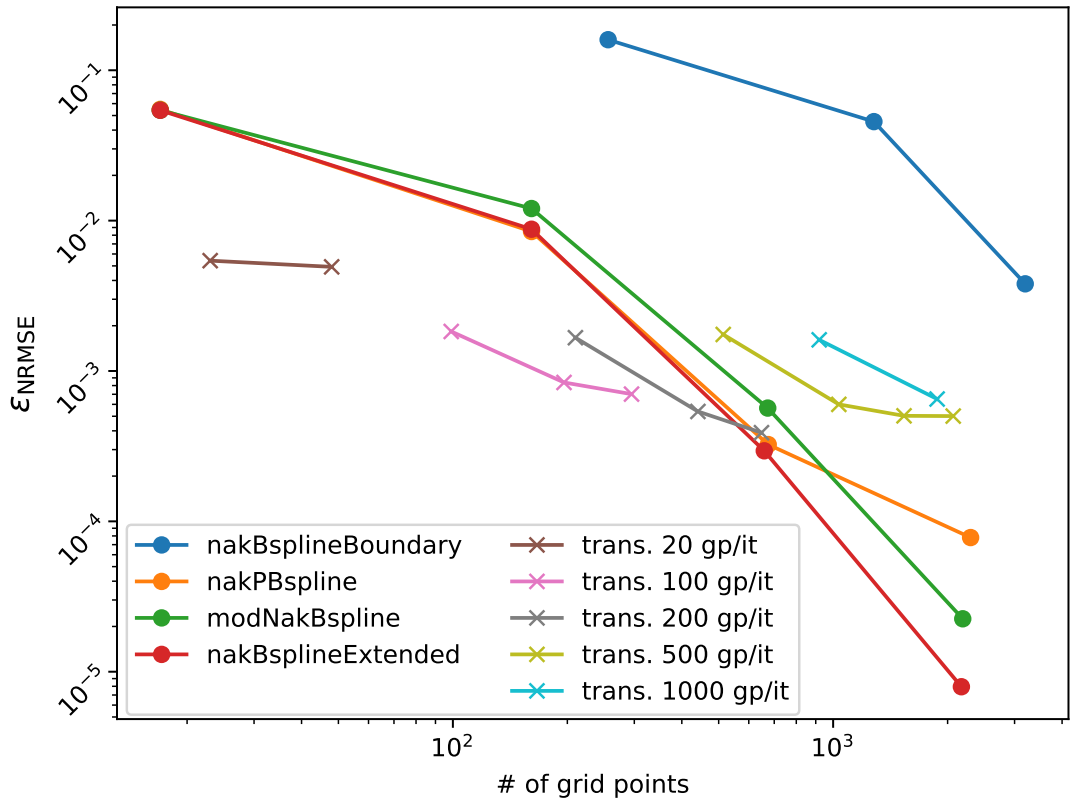


Figure 8.1: Approximation errors for transformed sparse grids constructed with various grid sizes per iteration and for various NAK B-splines.

Fig. 8.1 presents the approximation errors of both NAK B-splines and transformed sparse grids. We see that in the early stages, transformed sparse grids provide a better tradeoff between grid points and approximation quality. However, as more grid points are added to the surrogates, their quality improvement decays very fast and they soon reach a convergent state. This is consistent with previous results. In contrast, the sparse grid interpolation surrogates outshine them as the grid point amount rises and are able to continuously improve the approximation error. This crossover point is reached early on, which results in transformed sparse grids not being a viable alternative to sparse grid interpolation for the borehole function.

8.1.1 Summary

We can deduce that the transformed sparse grid technique comparatively performs much better than various untransformed sparse grid interpolation techniques for a very low amount of grid points. However, the interpolation techniques reach a crossover point at which they are able to continuously improve the approximation and overtake the transformation technique, which reaches a convergent state pretty early on. In this case with $d = 8$, the crossover point is reached too early such that the transformed surrogate technique is not viable as it is easy enough to still create an accurate interpolation surrogate. This crossover point is continuously pushed back as the model dimensionalities grow since untransformed sparse grids will be more heavily affected by the curse

of dimensionality and require a lot more grid points while the transformed sparse grids will only be affected by the best choice for r, which is usually smaller than d. Fig. 8.2 summarizes this hypothesis in a simple way.

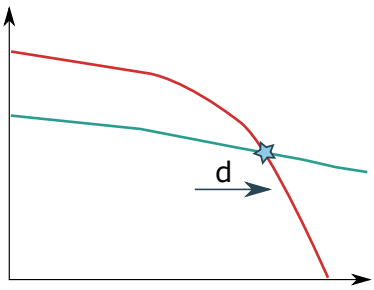


Figure 8.2: Illustration of the approximation quality behaviour. The red plot represents the approximation errors of standard interpolation approaches, the green plot represents transformed sparse grid approaches, and the light blue star represents the crossover point.

Furthermore, as the transformed technique internally uses regression, we are comparing an interpolation-based approach to a regression-based one, which is inherently weaker as it does not evaluate the model function at the most optimal points for the approximation. Therefore, the best applications for transformed sparse grids should be pure, high-dimensional regression problems with optionally only a small number of instances as the transformation technique even outperforms interpolation methods in these cases. Another potential application could be models that can only support a very small amount of function evaluations such that a normal sparse grid of an acceptable level can not be constructed. One such approach of trying to provide an acceptable approximation of the borehole function using only a very small amount of evaluations has already been investigated in [**DP10**].

8.2 Maintenance of naval propulsion plants

The second potential practical application for transformed sparse grids concerns improving the maintenance process of naval propulsion plants. In [**COG+16**], the authors aim to improve the process of maintaining naval propulsion plants that are used in various types of ships by employing the concept of condition-based maintenance.

When maintaining any type of machinery that is subject to wear and tear, there are several possible approaches of maintaining it. A corrective maintenance approach repairs and restores an object only after it breaks. One big disadvantage of this approach is having to deal with unscheduled failures and resulting maintenance, which can have cascading effects. These effects can be failures of other parts or productivity losses due to the unavailability of that machine, both of which having the potential to be costly. Condition-based maintenance improves upon this concept by performing maintenance when the condition degrades below a certain level. This approach allows for a better planning of future maintenance operations and avoids disruptions. However, it requires the availability of accurate condition diagnostics data on which maintenance decisions can be based on.

In [**COG+16**], the authors cover how to implement condition-based maintenance for COmbined Diesel eLectric And Gas (CODLAG) propulsion plants of frigate naval vessels. They do so by obtaining experimental model data through a real-time naval simulator, which is described in detail in [**ABFC09**]. The values of interest are the gas turbine compressor decay state coefficient kM_c and the gas turbine decay state coefficient kM_t. They describe the decay of the gas turbine and its compressor over a fixed period of time and allow us to accurately predict the decay state after

some time when approximated correctly. These coefficients are modeled as the result of 16 input parameters that describe the general state the ship is in and are listed in table 8.2. Through many simulation runs with varying input parameter combinations, the authors obtained a dataset that can be used for a regression such that we can approximate the decay coefficients. However, this regression problem is not simple to solve as the model dimensionality $d = 16$ results in the curse of dimensionality kicking in for many regression methods.

Variable	Description
lp	Lever position
v	Ship speed (knots)
GT_T	Gas turbine shaft torque (kNm)
GT_n	Gas turbine rate of revolutions (r/min)
GG_n	Gas generator rate of revolutions (r/min)
T_s	Starboard propeller torque (kN)
T_p	Port propeller torque (kN)
T_{48}	HP turbine exit temperature (°C)
T_I	GT compressor inlet air temperature (°C)
T_2	GT compressor outlet air temperature (°C)
P_{48}	HP turbine exit pressure (bar)
P_1	GT compressor inlet air pressure (bar)
P_2	GT compressor outlet air pressure (bar)
P_{exh}	GTexhaust gas pressure (bar)
TIC	Turbine injection control (%)
m_f	Fuel flow (kg/s)

Table 8.2: Input parameters for the naval propulsion plant decay model. (HP: high-pressure, GT: gas turbine)

As there are two target regression outputs, kM_c and kM_t, we are dealing with a multi-output regression problem. For simplity however, this problem can be decomposed into two single-output regression problems where the approximation errors are evaluated individually.

In a pure regression setting like this, we can go all out with the amount of grid points and training observations as this does not come with any associated model function evaluation cost in contrast to pure function approximations. Only the general runtime of the regression learner and other algorithms that perform operations on the surrogate is increased when the amounts of grid points and training observations are higher. It is therefore beneficial for the approximation quality that we incorporate as much training data as possible into the surrogate construction process because it would otherwise be unused.

As the data has been obtained through a simulation, some input parameters were varied using fixed size increments. Due to the resulting axis-aligned nature of some of the components of the available observation inputs, the nearest neighbour method has been modified such that it can be applied on this model. In cases when the observation inputs are partly aligned along some form of

grid structure, always choosing the nearest neighbours of an observation can cause the method to only select neighbours along one axis, especially when the input parameter intervals differ along each dimension. This can cause the system of linear of equations to not put a constraint on some of the gradient components, which in turn can cause the solver to return approximated gradients with some arbitrary components. To fix this issue such that we can still use the nearest neighbour method, we add small random vectors to the neighbour observations when solving the gradient system of linear equations.

We investigate the results for various dynamic pipelines that add different amounts of grid points per iteration as we did for the previous experiments. The dynamic pipeline uses the nearest neighbour gradient approximation method with $n' = 1000$ and $m = 100$ as it proved to be better than the random neighbour method. It then constructs sparse grid surrogate with modified linear basis functions using regression with a maximum of 10 train observations per grid point. The dataset consists out of $11,934$ observations where the first 70% of observations are used for training and the last 30% for testing. For comparison purposes we also approximate the model using normal spatially adaptive sparse grid regression surrogates, which effectively are pipeline runs with exactly one iteration and a fixed identity transformation function. Approximation results for the gas turbine compressor decay state coefficient kM_c and the gas turbine decay state coefficient kM_t are visualized in fig. 8.3 and fig. 8.4, respectively.

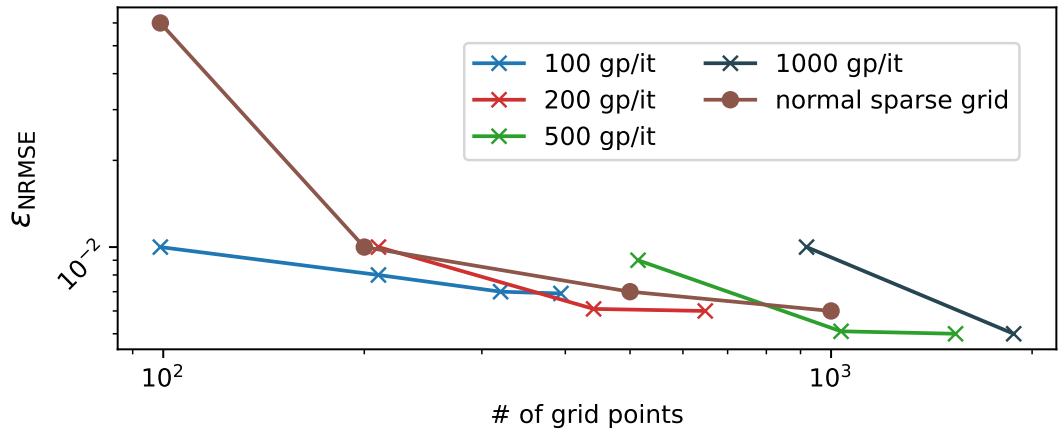


Figure 8.3: Approximation errors of transformed sparse grids constructed with various grid points per iteration for the gas turbine compressor decay state coefficient kM_c. Untransformed sparse grid regression surrogates are included as a reference.

We see that all approximations converge towards a similar error. Compared to normal, untransformed sparse grid surrogates, transformed sparse grids outperform their approximation quality for a low amount of grid points. As we add more grid points to the surrogates, the individual differences even out and the approximations all have an almost equal approximation quality.

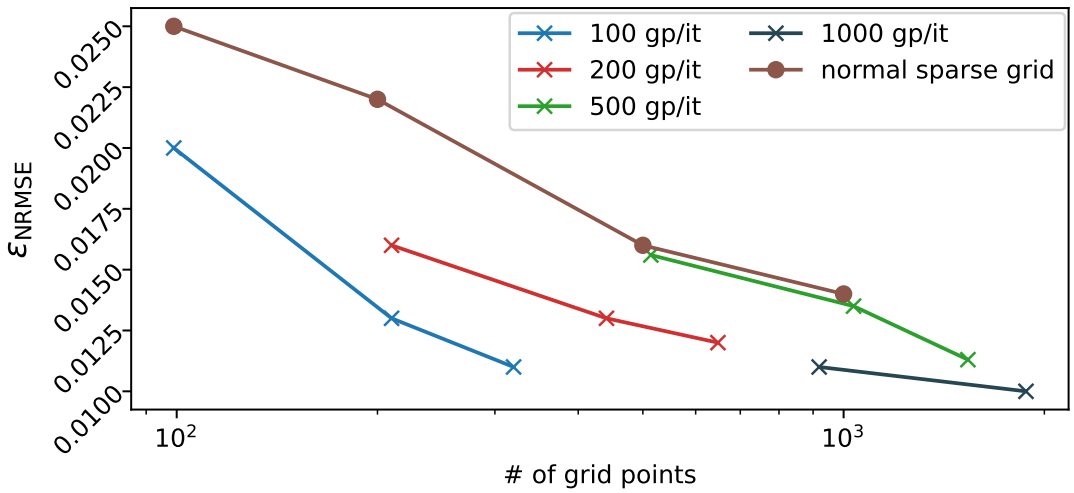


Figure 8.4: Approximation errors of transformed sparse grids constructed with various grid points per iteration for the gas turbine decay state coefficient kM_t. Untransformed sparse grid regression surrogates are included as a reference.

We observe that the approximation errors in fig. 8.4 behave similar to fig. 8.3 with the major difference being that it takes the normal sparse grid surrogates longer to catch up to the transformed sparse grids. Especially the 100 gp/it transformation pipeline delivers very good results.

In summary, we are only able to improve the approximation a little bit by using transformed sparse grids instead of untransformed sparse grids. Bigger improvements only occur for very low amounts of grid points. As the inputs transformations only add another layer of flexibity to the approximation process, this does not necessarily result in guaranteed big improvements. In cases when an identity transformation function works well enough, we are not able to outperform untransformed sparse grids.

8.3 Million songs dataset

For the final application we try to apply the transformed sparse grid technique to a very high-dimensional model to explore the limits of it. We choose to look at the Million Song Dataset [**BEWL11**], which contains various audio features and metadata for a collection of over one million songs. A dataset like this can be used for many potential applications. One common commercial application are tailored song recommendations based on gathered user data, while another application is music identification. In many cases, the processes of identifying and recommending music are interlinked, i. e., instead of always aiming to exactly identify a song, which can be difficult at times, it can also be sufficient to estimate the production year of a song and base recommendations on this value. Therefore, one application is to predict the production year of a song based solely on its observed audio features, which is a straightforward regression task. The limiting factor of this regression problem is the amount of input parameters, which in the case of this dataset are 90 different audio features. These features represent various timbre values that make up the song. A perfect identification of a song is not possible as audio features are not perfect predictors of the production year, i. e., songs with completely different production years may still

have very similar audio features. We can therefore not aim for an approximation error close to zero because this problem is ill-posed. Solving such a high-dimensional regression problem proves to be difficult as the curse of dimensionality is in full effect.

In a regression setting like this, we aim to incorporate as much training data as possible into the surrogate. The part of the dataset that is used for year prediction consists out of 515345 observations where the first 463,715 observations of the dataset are all used for training, and the last 51,630 for testing.

We start off by investigating the results for various dynamic pipelines that add different amounts of grid points per iteration as we did for the previous experiments. The dynamic pipeline uses the nearest neighbour gradient approximation method with $n' = 10000$ and $m = 1000$ as it proved to be better than the random neighbour method. Furthermore, it chooses $1 \leq r_i \leq 25$ and creates a sparse grid surrogate using regression with a maximum of 50 training observations per grid point. The 463,715 available train observations are therefore all used for surrogates with around 10^4 grid points. Results of various pipeline executions are shown in fig. 8.5.

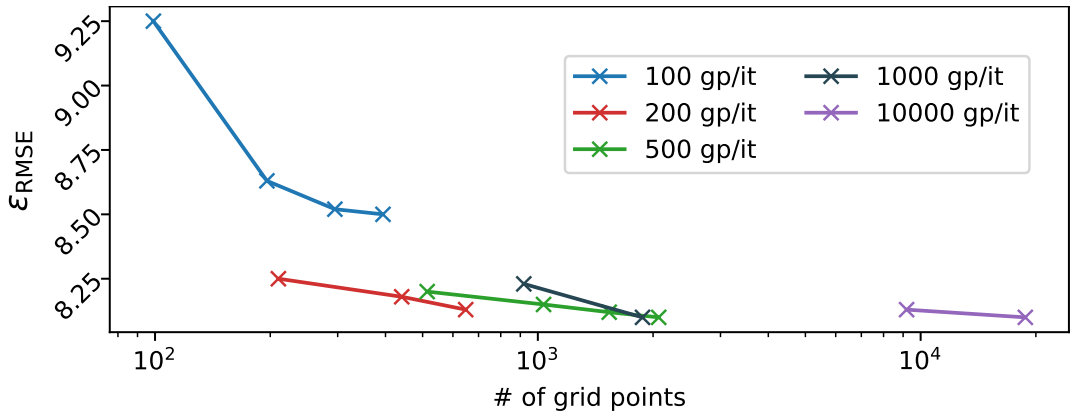


Figure 8.5: Approximation errors of transformed sparse grids constructed with various grid points per iteration for the Million Song Dataset year prediction regression problem.

We observe that we hit a hard limit in terms of the approximation error very early on, at least when exlcuding the 100 gp/it case. Considering the dimensionality of the model, one would expect that the approximation would benefit from more employed grid points, which allows for a better grid point resolution for sparse grid surrogates with a higher dimensionality. However, this does not seem to be the case. We either hit a limit on the effectiveness of the transformed sparse grid technique or are limited by the inherent noise of the rather ill-posed regression problem.

In [**BEWL11**], the authors also included some evaluation results for this regression problem using the k-nearest neighbours (k-NN) method [**Alt92**] and the Vowpal Wabbit method [**LLZ09**], which is a large-scale learning algorithm designed for problems like this. The benchmark they set out to beat was the constant prediction method, which always returns the average release year of the total dataset. Any good regression method should be able to beat this trivial benchmark to provide actual insight for this problem. We compare these results to the best transformed sparse grid technique results that were determined previously. The results, including reference results, are shown in fig. 8.6.

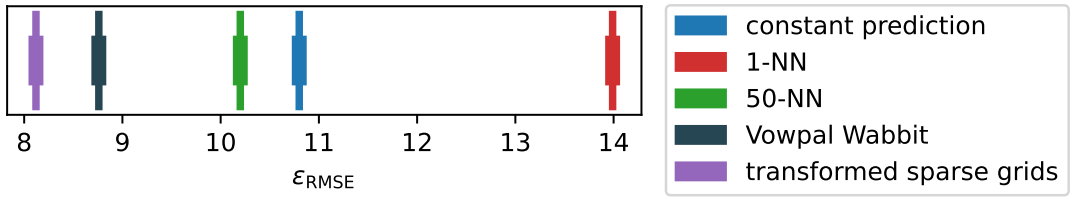


Figure 8.6: Comparison of select approximation results for the Million Song Dataset year prediction regression problem.

We can oberve that the included methods from [**BEWL11**] differ vastly in their approximation quality. The 1-NN regression performs worse than the constant prediction method, making it not feasible to use. When using the 50-NN method, we see that taking more neighbours into account already results in a much better approximation compared to only 1-NN. As it fares better than the constant prediction method, the 50-NN method does provide some insight. The more simple approach of k-NN regression can be improved by utilizing the more complex Vowpal Wabbit method, which incorporates more than just neighbouring observations into its approximation. In fact, it is based on linear transformations that were determined by utilizing gradient descent. The transformed sparse grid method delivers slighly better results compared to the Vowpal Wabbit method. This might be explained by the fact that both methods use a rougly similar approach that is based on linear transformations and are therefore also similar in their approximation qualities.

9 Conclusion and outlook

To summarise this thesis, we discovered that the introduced transformed sparse grid technique is very flexible in the sense that it can be used as pure black box regression method without the need for additional model function evaluations or gradients. We saw that we can substiute required model gradients with approximated ones and use neighbour approximation methods as a basis for the active subspace computations to still obtain comparable results. In case more information about the model is available, such as the model function itself or its gradients, the technique can also incorporate that additional data to achieve slightly better results.

Furthermore, the technique can be applied to both function approximation and regression problems and delivers competitive results in both cases. In a setting where every additional model function evaluation is costly, the technique is able to construct surrogates with a very low amount of required model function observations as seen for the borehole function. When a lot of observations are already available, for example for the million songs dataset, the technique is also able to utilize the additional available training data as well.

Transformed sparse grids also succeed at their primary goal of reducing the impact of the curse of dimensionality. In theory, the underlying sparse grid surrogates can still be affected by it if r is required to be almost equal to d to achieve an acceptable approximation. However, for all model functions we covered this was not the case, i. e., the surrogate approximation quality was rather unaffected by high values for r. In some cases, the approximation quality even dropped when r was too high and resulted in a grid point resolution for each dimension that was too low.

When being compared to other interpolation and regression techniques, we saw that a fundemental property of transformed sparse grids is a very fast approximation error convergence. Even for very high-dimensional models, the transformed sparse grid technique is usually able to converge within the first few hundred grid points. This is a strong indication that the active subspace method is very suitable for determining active directions and works well in conjunction with transformed sparse grids.

The asymptotic approximation error of transformed sparse grids can be seen as competitive compared to other established methods as we have seen in the last chapter. As a result of focusing the attention on the most important active directions, transformed sparse grids always spend the grid points along directions that matter the most and are able to converge very quickly. However, mainly due to the fact that the technique is regression based, we saw that it can not deliver the same asymptotic approximation error when more and more grid points are added to the surrogate compared to interpolation techniques. Relative to normal sparse grid regression, the technique delivers overall better results as it adds more flexibility to the approximation process. In a sense, normal sparse grid regression is a special case of the transformed sparse grid technique with a fixed identity transformation function.

There are also a few more areas of interest regarding transformed sparse grids that warrant further research and might yield potential improvements:

Parallelization The implementation of the transformation pipeline has the potential to be almost perfectly parallelized. A huge portion of the total runtime is spent on evaluating multiple surrogates that either come from transformation stream generators (sec. 6.2.1) or from the process of determining the optimal surrogate configuration parameters (sec. 5.3) . These components can be parallelized by evaluating all of them in parallel as they don't exchange any data or need to coordinate for example by using an API like OpenMP [**Ope08**]. Furthermore, other optimizations like vectorization [**MGG+11**] or GPU execution support with OpenCL [**SGS10**] could also be implemented. This can be combined with the already implemented support of the many of the listed frameworks and technologies by the sparse grid library SG^{++} to obtain a completely optimized and very performant pipeline.

Formalization of concepts The main workflow used to eventually come up with the resulting transformation pipeline was mainly based on ideas and practical experiments. While some formal concepts, like intrinsic dimensionality (sec. 3.1) or active subspaces (sec. 3.4), were used, many parts, especially the transformation construction and the iterative algorithm design, were introduced constructively without any proof of their qualities. Maybe there also exist better ways to construct transformed surrogates compared to the ones used in this thesis or a better way to design an iterative transformation algorithm that works with these surrogates. Methods for determining active directions can also be evaluated in a more formal way. In summary, the general ideas of this thesis can be investigated with a more formal and rigorous approach such that the results can be backed by a theory.

Active subspaces The neigbour-based gradient approximation methods can also be interesting for purposes other than surrogate construction. For example, more general sensitivity analysis applications that active subspaces were originally designed to handle, i. e., one where insights about the input parameter activity are enough or should be used for different purposes, could also benefit from it. The novel approach of just using neighbouring observations to compute an active subspace allows the active subspace method to be applied to regression problems as well. We saw that the active subspace directions computed this way are perfectly usable, at least for the purposes of this thesis. Therefore, the neighbour-based active subspace approximation methods are good targets for further research since they may be potentially suitable for other applications as well.

Surrogate types In this thesis, we focused on the sparse grid technique when constructing reduced surrogates for the simple reason that they are better suited for a following dimensionality reduction after the inputs have been transformed using an orthonormal active direction matrix. Other surrogate types, like radial basis function surrogates, can not make use of the transformed inputs with the same effectiveness as axis-aligned sparse grids. In theory, any surrogate type that uses the tensor-product approach to construct basis functions and supports some form of regression can benefit from the presented active direction transformations. Another common candidate aside from sparse grids would be the polynomial chaos expansion method [**CLM09**]. As the implementation is designed to be very flexible in terms of possible surrogate types, adding support for other methods would not be too difficult.

Automatic parameterization To achieve good approximation results in a timely manner, it is vital to determine good low-fidelity surrogates to guarantee a fast pipeline execution. Suitable low-fidelity surrogates are indicative of the comparative quality of their high-fidelity surrogate equivalents. Right now, these are determined more or less empirically as automating this process is not trivial, especially when dealing with pre-convergent behaviour of the low-fidelity surrogates. It therefore requires some trial and error to get a feeling at which point low-fidelity surrogates become suitable to use. However, it should be possible to develop an algorithm that automatically determines optimal low-fidelity surrogate configuration parameters by generating low-fidelity surrogates with different parameters until a representative one is found. This would make the transformation process even simpler to apply on a variety of models as it would eliminate prior manual probing of the model function.

Bibliography

[**Bel61**] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961. ISBN: 9781400874668. DOI: doi:10.1515/9781400874668. URL: https://doi.org/10.1515/9781400874668 (cit. on pp. 1, 3, 16).

[**Ben69**] R. Bennett. "The intrinsic dimensionality of signal collections". In: *IEEE Transactions on Information Theory* 15.5 (1969), pp. 517–525. DOI: 10.1109/TIT.1969.1054365 (cit. on pp. 1, 15).

[**Ste+73**] H. J. Stetter et al. *Analysis of discretization methods for ordinary differential equations*. Vol. 23. Springer, 1973 (cit. on p. 1).

[**HG83**] W. V. Harper, S. K. Gupta. *Sensitivity/uncertainty analysis of a borehole scenario comparing Latin Hypercube Sampling and deterministic sensitivity approaches*. Oct. 1983. URL: https://www.osti.gov/biblio/5355549 (cit. on p. 75).

[**Hub85**] P. J. Huber. "Projection pursuit". In: *The annals of Statistics* (1985), pp. 435–475 (cit. on p. 39).

[**Dev86**] L. Devroye. "Non-Uniform Random Variate Generation". In: Springer-Verlag, 1986. Chap. Chapter II.2: The lnverslon method. Pp. 27–40 (cit. on p. 13).

[**BL88**] D. Broomhead, D. Lowe. "Multivariable Functional Interpolation and Adaptive Networks". In: *Complex Systems* 2 (1988), pp. 321–355 (cit. on p. 11).

[**Nie88**] H. Niederreiter. "Low-discrepancy and low-dispersion sequences". In: *Journal of Number Theory* 30.1 (1988), pp. 51–70. ISSN: 0022-314X. DOI: https://doi.org/10.1016/0022-314X(88)90025-X. URL: https://www.sciencedirect.com/science/article/pii/0022314X8890025X (cit. on p. 13).

[**IH90**] T. Ishigami, T. Homma. "An importance quantification technique in uncertainty analysis for computer models". In: *First International Symposium on Uncertainty Modeling and Analysis* (1990), pp. 398–403 (cit. on p. 59).

[**Zen91**] C. Zenger. "Sparse grids". In: *Parallel Algorithms for Partial Differential Equations, Proceedings of the 6th GAMM-Seminar Kiel 1990* (Jan. 1991), pp. 241–251 (cit. on p. 1).

[**Alt92**] N. S. Altman. "An introduction to kernel and nearest-neighbor nonparametric regression". In: *The American Statistician* 46.3 (1992), pp. 175–185 (cit. on p. 81).

[**Bal94**] R. Balder. "Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern". PhD thesis. Technische Universität München, 1994 (cit. on p. 9).

[**GG98**] T. Gerstner, M. Griebel. "Numerical Integration using Sparse Grids". In: *Numerical algorithms* 18.3 (1998), pp. 209–232 (cit. on p. 15).

[**Kre99**] R. Kress. "Regularization by Discretization". In: *Linear Integral Equations*. Springer New York, 1999, pp. 308–319. ISBN: 978-1-4612-0559-3. DOI: 10.1007/978-1-4612-0559-3_17. URL: https://doi.org/10.1007/978-1-4612-0559-3_17 (cit. on p. 29).

[**SLY99**] Sobol, I. Levitan, Y. “On the use of variance reducing multipliers in Monte Carlo computations of a global sensitivity index”. In: *Computer Physics Communications* 117.1 (1999), pp. 52–61 (cit. on p. 59).

[**GGT01**] J. Garcke, M. Griebel, M. Thess. “Data Mining with Sparse Grids”. In: *Computing* 67.3 (2001), pp. 225–253. DOI: 10.1007/s006070170007. URL: https://doi.org/10.1007/s006070170007 (cit. on p. 15).

[**Sob01**] I. M. Sobol. “Global Sensitivity Indices for Nonlinear Mathematical Models and Their Monte Carlo Estimates”. In: *Mathematics and computers in simulation* 55.1-3 (2001), pp. 271–280. DOI: 10.1016/S0378-4754(00)00270-6. URL: http://dx.doi.org/10.1016/S0378-4754(00)00270-6 (cit. on p. 19).

[**BG04**] H.-J. Bungartz, M. Griebel. “Sparse Grids”. In: *In: Acta Numerica* 13 (May 2004), pp. 147–269. DOI: 10.1017/S0962492904000182 (cit. on p. 6).

[**RWEH06**] T. Robinson, K. Willcox, M. Eldred, R. Haimes. “Multifidelity Optimization for Variable-Complexity Design”. In: *Collection of Technical Papers - 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* 4 (Sept. 2006). DOI: 10.2514/6.2006-7114 (cit. on p. 36).

[**BBC07**] D. Bailey, J. Borwein, R. Crandall. “Box integrals”. In: *Journal of Computational and Applied Mathematics* 206.1 (2007), pp. 196–208. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2006.06.010. URL: https://www.sciencedirect.com/science/article/pii/S0377042706004250 (cit. on p. 27).

[**FSK08**] A. Forrester, A. Sobester, A. Keane. *Engineering Design Via Surrogate Modelling: A Practical Guide*. July 2008. ISBN: 978-0-470-06068-1. DOI: 10.1002/9780470770801 (cit. on pp. 36, 66).

[**HS08**] X. Huo, A. Smith. “A Survey of Manifold-Based Learning Methods”. In: *Recent Advances in Data Mining of Enterprise Data* (Jan. 2008). DOI: 10.1142/9789812779861_0015 (cit. on p. 31).

[**Ope08**] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 3.0*. May 2008. URL: http://www.openmp.org/mp-documents/spec30.pdf (cit. on p. 84).

[**WS08**] X. Wang, I. H. Sloan. “Low discrepancy sequences in high dimensions: How well are their projections distributed?” In: *Journal of Computational and Applied Mathematics* 213.2 (2008), pp. 366–386. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2007.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0377042707000374 (cit. on pp. 16, 29).

[**ABFC09**] M. Altosole, G. Benvenuto, M. Figari, U. Campora. “Real-time simulation of a COGAG naval ship propulsion system”. In: *Proceedings of the institution of mechanical engineers, part M: journal of engineering for the maritime environment* 223.1 (2009), pp. 47–62 (cit. on p. 77).

[**CLM09**] T. Crestaux, O. Le Maıtre, J.-M. Martinez. “Polynomial chaos expansion for sensitivity analysis”. In: *Reliability Engineering & System Safety* 94.7 (2009). Special Issue on Sensitivity Analysis, pp. 1161–1172. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2008.10.008. URL: https://www.sciencedirect.com/science/article/pii/S0951832008002561 (cit. on p. 84).

[**FG09**] C. Feuersänger, M. Griebel. “Principal manifold learning by sparse grids”. In: *Computing* 85.4 (2009), pp. 267–299. ISSN: 1436-5057. DOI: 10.1007/s00607-009-0045-8. URL: https://doi.org/10.1007/s00607-009-0045-8 (cit. on p. 31).

[**LLZ09**] J. Langford, L. Li, T. Zhang. “Sparse Online Learning via Truncated Gradient.” In: *Journal of Machine Learning Research* 10.3 (2009) (cit. on p. 81).

[**AW10**] H. Abdi, L. J. Williams. “Principal component analysis”. In: *WIREs Computational Statistics* 2.4 (2010), pp. 433–459. DOI: https://doi.org/10.1002/wics.101. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.101. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.101 (cit. on pp. 1, 15, 20, 31).

[**DP10**] H. Dette, A. Pepelyshev. “Generalized Latin Hypercube Design for Computer Experiments”. In: *Technometrics* 52.4 (2010), pp. 421–429. DOI: 10.1198/TECH.2010.09157. eprint: https://doi.org/10.1198/TECH.2010.09157. URL: https://doi.org/10.1198/TECH.2010.09157 (cit. on p. 77).

[**Feu10**] C. Feuersänger. “Sparse Grid Methods for Higher Dimensional Approximation”. PhD thesis. Mathematisch–Naturwissenschaftliche Fakultät der der Rheinischen Friedrich–Wilhelms–Universität Bonn, 2010 (cit. on pp. 15, 19).

[**Pfl10**] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Verlag Dr. Hut, 2010. ISBN: 9783868535556. URL: http://www5.in.tum.de/pub/pflueger10spatially.pdf (cit. on pp. 8, 9, 15, 49, 55).

[**SGS10**] J. E. Stone, D. Gohara, G. Shi. “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems”. In: *Computing in Science Engineering* 12.3 (2010), pp. 66–73. DOI: 10.1109/MCSE.2010.69 (cit. on p. 84).

[**BEWL11**] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, P. Lamere. *The million song dataset*. 2011 (cit. on pp. 80–82).

[**MGG+11**] S. Maleki, Y. Gao, M. J. Garzar´n, T. Wong, D. A. Padua. “An Evaluation of Vectorizing Compilers”. In: *2011 International Conference on Parallel Architectures and Compilation Techniques*. 2011, pp. 372–382. DOI: 10.1109/PACT.2011.68 (cit. on p. 84).

[**MDS12**] H. Moon, A. M. Dean, T. J. Santner. “Two-Stage Sensitivity-Based Group Screening in Computer Experiments”. In: *Technometrics* 54.4 (2012), pp. 376–387. ISSN: 00401706. URL: http://www.jstor.org/stable/41714922 (cit. on p. 66).

[**Pfl12**] D. Pflüger. “Spatially Adaptive Refinement”. In: *Sparse Grids and Applications*. Ed. by J. Garcke, M. Griebel. Lecture Notes in Computational Science and Engineering. Springer, 2012, pp. 243–262. URL: http://www5.in.tum.de/pub/pflueger12spatially_preprint.pdf (cit. on p. 10).

[**CG14**] P. Constantine, D. Gleich. “Computing active subspaces with Monte Carlo”. In: *arXiv preprint arXiv:1408.0545* (2014). arXiv: 1408.0545 [math.NA] (cit. on pp. 25, 26).

[**Con15**] P. Constantine. “Active Subspaces - Emerging Ideas for Dimension Reduction in Parameter Studies”. In: *SIAM spotlights*. 2015 (cit. on pp. 1, 20, 25).

[**COG+16**] A. Coraddu, L. Oneto, A. Ghio, S. Savio, D. Anguita, M. Figari. "Machine learning approaches for improving condition-based maintenance of naval propulsion plants". In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 230.1 (2016), pp. 136–153 (cit. on p. 77).

[**KH16**] V. Khakhutskyy, M. Hegland. "Spatially-Dimension-Adaptive Sparse Grids for Online Learning". In: *Sparse Grids and Applications - Stuttgart 2014*. Ed. by J. Garcke, D. Pflüger. Springer International Publishing, 2016, pp. 133–162. ISBN: 978-3-319-28262-6 (cit. on p. 11).

[**BGF+19**] R. A. Bridges, A. D. Gruber, C. Felder, M. Verma, C. Hoff. "Active Manifolds: A non-linear analogue to Active Subspaces". In: *arXiv preprint arXiv:1904.13386* (Apr. 2019). arXiv: 1904.13386 [stat.ML] (cit. on p. 31).

[**Val19**] J. Valentin. "B-Splines for Sparse Grids: Algorithms and Application to Higher-Dimensional Optimization". PhD thesis. Oct. 2019. DOI: 10.18419/opus-10504. arXiv: 1910.05379 [math.NA] (cit. on pp. 15, 37).

[**Reh21**] M. F. Rehme. "B-Splines on Sparse Grids for Uncertainty Quantification". PhD thesis. Universität Stuttgart, 2021 (cit. on pp. 15, 75).

[**Boc**] S. Bochkanov. *ALGLIB*. URL: www.alglib.net (cit. on p. 55).

[**Wei**] E. W. Weisstein. *Hypercube Line Picking*. URL: https://mathworld.wolfram.com/HypercubeLinePicking.html (cit. on p. 27).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature