

Visual diagnostics for constrained optimisation with application to guided tours

by H.Sherry Zhang

Abstract Projection pursuit searches for interesting low-dimensional views of high-dimensional data via the optimisation of an index function. The initial paper by Friedman & Tukey in 1974 stated that “the technique used for maximising the projection index strongly influences both the statistical and the computational aspects of the procedure.” However, while many projection pursuit indices have been proposed in the literature, less work has been done on the optimisation procedures. In this paper we introduce visual diagnostics for optimisation algorithms, in particular those available in the projection pursuit guided tour. These diagnostics and workflows can be applied to a broad class of optimisers, to assess their performance. An R package, ferrn, has been created to implement the diagnostics.

Introduction

Visualisation is widely used in exploratory data analysis (REF). Presenting information in a graphical format often allows people to uncover information they would otherwise not be aware of and provides a more comprehensive understanding of the problem at hand. The work presented in this paper combines data visualization with optimization methods. It aims at bringing visualization tools into optimization problems to assist users to better understand the performance of their optimization methods in practice. This motivates our work of creating plots to diagnose optimisation algorithms which in turn can be also used not only to understand optimization performance but also to understand and compare the behaviour of different algorithms.

The goal of continuous optimisation is to find the best solution within the space of all feasible solutions where typically the best solution is decided by an objective function. Broadly speaking optimization can be unconstrained or constrained ([Kelley \(1999\)](#)). The unconstrained problem can be formulated as a minimization (or maximization) problem such as $\min_x f(x)$ where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is an objective function with certain properties defined in an L^p space. In this case, solutions rely on gradient descent or ascent methods. In the constrained optimization problem additional restrictions are introduced via a set of functions that can be convex or non-convex: $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ for $i = 1, \dots, k$ and hence the problem can be written as $\min_x f(x)$ subject to $g_i(x) \leq 0$. Here methods such as Langrange multipliers and convex optimization methods including linear and quadratic programming can be used.

The motivation of this paper is the optimisation problem arising in the projection pursuit guided tour ([Buja et al. \(2005\)](#)) which is an exploratory data analysis tool that is defined to detect *interesting structures* or features in high-dimensional data through a set of lower-dimensional projections that cover the entire high dimensional space using interpolation methods called tours ([cook2008grand](#)). The target of the optimisation is to identify the most *interesting* low-dimensional views of the data given by a corresponding projection matrix. The most *interesting* structures are formally defined by a function of projections, called index function which is optimized to uncover the most revealing structures in a high dimensional space ([Cook et al. \(1993\)](#)).

The optimization challenges encountered in the projection pursuit guided tour problem are common to those of optimization in general. Examples of those include the existence of multiple maxima (local and global), the trade off between computational burden and proximity to the maxima, dealing with noisy objective functions that might be non smooth and non differentiable ([Jones et al. \(1998\)](#)). Those are not unique to this context and therefore the visualization tools and optimization methods presented in this paper can be easily apply to any other optimization problems.

The remainder of the paper is organised as follows. Section 2.2 provides an overview of optimisation methods, specifically line search methods. Section 2.3 reviews projection pursuit guided tour, defines the optimisation problem and introduces three existing algorithms. Section 2.4 presents the new visual diagnostics. A data structure is defined to capture information during the optimisation, and used in different types of diagnostic plots. Section 2.5 shows applications of how these plots can be used to understand and compare different algorithms. We also discuss how these insights contribute to modifications that improve the algorithms. Finally, Section 2.6 describes the R package: ferrn, that implements the visual diagnostics.

Optimisation Methods

Optimization problems are ubiquitous in many areas of study. While in some cases, analytical solutions can be found, the majority of problems rely on numerical methods to find the optimal solution. These numerical methods follow interactive approaches that aim at finding the optimum by progressively improving the current solution until a desirable accuracy is achieved. Although that principle seems uncomplicated, optimization methods need to deal with many challenges such as the existence of multiple maxima (local and global), the trade off between desirable accuracy and computational burden, as well as dealing with noisy objective functions and possible constraints. In addition, optimization results might depend on the algorithm starting values affecting the consistency of results.

Our interest is on constrained optimization (REF) as defined in the introduction section and assume that it is not possible to find a solution to the problem in the way of a closed form. That is, the problem consists of finding the minimum or maximum of a function $f \in L^p$ in the constrained \mathbb{A} space.

Optimization methods can be divided in various classes (REF). In this paper, we focused on methods that rely on the objective function derivatives known as gradient methods, and those which are known as derivative free methods where optimization does not rely on the gradient or derivatives information.

In gradient descent methods the goal is to find the minimum of an objective function while in gradient ascent methods the interest is in the maximum. The minimum is searched for by taken steps proportional to the negative gradient or of the function at the evaluation point (analogously positive gradient for gradient ascent methods), that is $x_{k+1} = x_k - \alpha_k S_k$ where α_k is the step size or the learning rate (Fletcher (2013)). The step size is the key in the success of the optimization problem as inadequate step sizes might lead to a failure in convergence. These methods are known to be relative slow and can have convergence issues when problems are poorly conditioned (Trefethen and Bau III (1997)). In order to solve some of those problems, line minimization or line search methods were proposed (Shi (2004)) where the step size is chosen in such a way that convergence is ensured. In line search methods users are required to provide an initial estimate x_1 , a search direction S_k for each iteration where the objective is to find the value of $\alpha_k \in \mathbb{R}$ that minimizes $f(x_k + \alpha_k S_k)$ with respect to α_k (this step is called the line search). Then move on to the next point following $x_{k+1} = x_k + \alpha_k S_k$ and repeat the process until the desire convergence is reached. Modern development of line search methods focuses on improving the computation of the searching direction S_k , and on approximations of the step size α_k together with reducing the computational storing burden of the intermediate computations while catering for practical optimisation problems. In the next section two line search methods for the optimisation used in the projection pursuit guided tour will be presented.

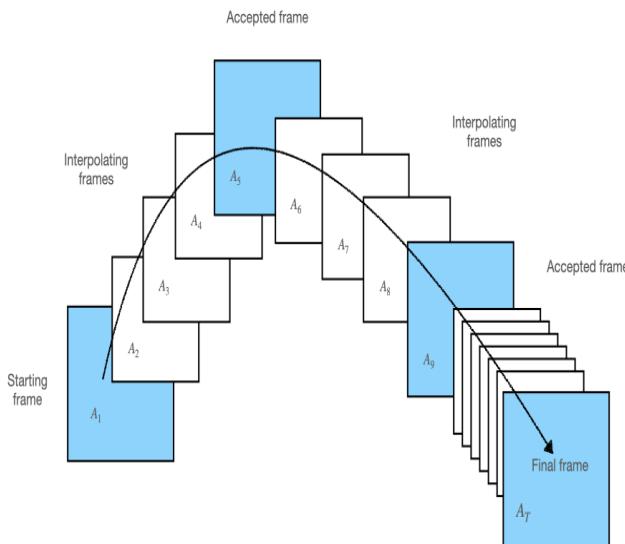
Although gradient optimization methods are very popular, they rely on the existence of the objective function derivatives and on the complexity of the constraints. Because of that, the second big class of optimization problems look at derivative free methods where the emphasis is on finding, in most cases, a near optimal solution. Examples of those include response surface methodology (REF), stochastic approximation (REF), random search (REF) and heuristic methods like tabu search and genetic algorithms (REF). Later, we will present an optimization algorithm within the random search class, namely simulated annealing for optimization with the guided tour. Moreover, we will introduce an algorithm to further improve optimization solutions both in gradient descent methods and in derive free optimization.

Several R implementations address optimization problems with both general purpose as well as tasks specifics solvers. The most prominent one within the general solvers is *optim* in (REF-stats package) which have functions for gradient based and derivative free methods. Other packages having similar functionalities include *nloptr* (REF), XX.

Projection pursuit guided tour

The projection pursuit guided tour combines two different methods in exploratory data analysis, focusing on different aspects. Projection pursuit, coined by Friedman and Tukey (1974), detects interesting structures (e.g. clustering, outliers and skewness) in multivariate data via low dimensions projection. The guided tour is using ideas from projection pursuit to define a particular variation in a broader class of data visualisation methods, building on the grand tour approach (Asimov, 1985).

To define projection pursuit, we first need to establish the notation used. Let $\mathbf{X}_{n \times p}$ be the data matrix, with n observations in p dimensions. A d -dimensional projection can be seen as a linear transformation from \mathbb{R}^p into \mathbb{R}^d , and defined as $\mathbf{Y} = \mathbf{X} \cdot \mathbf{A}$, where $\mathbf{Y}_{n \times d}$ is the projected data and $\mathbf{A}_{p \times d}$ is the projection matrix. Define $f : \mathbb{R}^{n \times d} \mapsto \mathbb{R}$ to be an index function that maps the projected data \mathbf{Y} (corresponding to an associated projection matrix \mathbf{A}) onto an index value I (QUESTION: Isn't f the

**Figure 1:** (ref:tour-path)

(#fig:tour-path)

index function?). This is commonly known as the projection pursuit index function, or just index function, and is used to measure the “interestingness” of a given projection.

A number of index functions have been proposed in the literature to detect different data structures, including Legendre index (Friedman and Tukey, 1974), Hermite index (Hall et al., 1989), natural Hermite index (Cook et al., 1993), chi-square index (Posse, 1995), LDA index (Lee et al., 2005) and PDA index (Lee and Cook, 2010).

As a general visualisation method, a tour produces animations of high dimensional data via rotations between low dimension planes. Different tour types choose these planes differently, for example, a grand tour (Cook et al., 2008) selects the planes randomly and a manual tour (Cook and Buja, 1997) gradually phases in and out one variable, to understand the contribution of that variable in the projection. Guided tour, the main interest of this paper, chooses planes with the aid of projection pursuit. Given a random start, projection pursuit iteratively finds bases with higher index values and the guided tour constructs the geodesic interpolation between these planes to form a tour path.

Mathematical details of the geodesic interpolation can be found in Buja et al. (2005). Figure @ref(fig:tour-path) shows a sketch of the tour path. The blue frames are produced by the projection pursuit optimisation algorithm, and the white frames interpolate between them. The tour method has been implemented in the tourr package in R, available on the Comprehensive R Archive Network at <https://cran.r-project.org/web/packages/tourr/> (Wickham et al., 2011).

(ref:tour-path) Each square (frame) represents the projected data with a corresponding basis. Blue frames are found by an optimisation algorithm iteratively whilst the white frames are constructed between two blue frames by geodesic interpolation.

Optimisation in the tour

The optimisation problem in the tour context is stated as follows: Given a randomly generated starting basis \mathbf{A}_1 , projection pursuit finds the final projection basis \mathbf{A}_T that satisfies the following optimisation problem:

$$\arg \max_{\mathbf{A} \in \mathcal{A}} f(\mathbf{X} \cdot \mathbf{A}) \quad (1)$$

$$s.t. \mathbf{A}' \mathbf{A} = I_d \quad (2)$$

where I_d is the d -dimensional identity matrix and the constraint requires the projection bases \mathbf{A} to be orthogonal matrices.

Several features of this optimisation are worth noticing. First of all, this is a constrained optimisation problem as the decision variables form the entries of a projection basis, which is required to be orthonormal. It is also likely that the objective function may not be differentiable for a constructed

index function and in these cases, gradient-based methods may not work well. Although finding the global maximum is the goal of an optimisation problem, it is also interesting to inspect local maximum in projection pursuit since it could present unexpected interesting projections. Lastly, there is also one computational consideration: the optimisation procedure needs to be fast to compute since the tour animation is played in real-time.

Existing algorithms

Below we introduce three possible algorithms: `search_better`, `search_better_random`, and `search_geodesic`. The first two are derivative free methods that sample candidate bases in the neighbourhood whilst `search_geodesic` is an analogue of gradient ascent on the projection basis space.

Algorithm 1: random search

```

input :  $\mathbf{A}_{\text{cur}}$ ,  $f$ ,  $\alpha$ ,  $l_{\max}$ 
output:  $\mathbf{A}_l$ 
1 initialisation;
2 Set  $l = 1$ ;
3 while  $l < l_{\max}$  do
4   Generate  $\mathbf{A}_l = (1 - \alpha)\mathbf{A}_{\text{cur}} + \alpha\mathbf{A}_{\text{rand}}$  and orthogonalise  $\mathbf{A}_l$ ;
5   Compute  $I_l = f(\mathbf{A}_l)$ ;
6   if  $I_l > I_{\text{cur}}$  then
7     | return  $\mathbf{A}_l$  ;
8   end
9    $l = l + 1$ ;
10 end
```

`search_better` is a random search device that samples a candidate basis \mathbf{A}_l in the neighbourhood of the current basis \mathbf{A}_{cur} by $\mathbf{A}_l = (1 - \alpha)\mathbf{A}_{\text{cur}} + \alpha\mathbf{A}_{\text{rand}}$ where α controls the radius of the sampling neighbourhood and \mathbf{A}_{rand} is a randomly generated matrix with the same dimension as \mathbf{A}_{cur} . \mathbf{A}_l is then orthogonalised to ensure the orthonormal constraint is fulfilled. When a basis is found with index value higher than the current basis \mathbf{A}_{cur} , the search terminates and outputs the basis for guided tour to construct an interpolation path. The next iteration of search begins after adjusting α by a cooling parameter: $\alpha_{j+1} = \alpha_j * \text{cooling}$. The termination condition is when the maximum number of iteration l_{\max} is reached. The algorithm of `search_better` is summarised in Algorithm 1. A slightly different cooling scheme has been proposed by Posse (1995) to include a halving parameter c . Rather than reducing the radius of the searching neighbourhood, α , at each iteration, Posse's design only adjust α if the last search takes more than c times to find an accepted basis to avoid the searching space being reduced too fast.

Algorithm 2: simulated annealing

```

1 Compute  $I_l = f(\mathbf{A}_l)$  and  $T(l) = \frac{T_0}{\log(l+1)}$ ;
2 if  $I_l > I_{\text{cur}}$  then
3   | return  $\mathbf{A}_l$  ;
4 else
5   | Compute  $P = \min \left\{ \exp \left[ -\frac{I_{\text{cur}} - I_l}{T(l)} \right], 1 \right\}$ ;
6   | Draw  $U$  from a uniform distribution:  $U \sim \text{Unif}(0, 1)$ ;
7   | if  $P > U$  then
8     |   | return  $\mathbf{A}_l$  ;
9   | end
10 end
```

Simulated annealing (`search_better_random`) (Kirkpatrick et al., 1983, Bertsimas et al. (1993)) uses the same sampling process as `search_better` but allows a probabilistic acceptance of a basis with lower index value based on the annealing $T(l)$. Given an initial T_0 , the temperature at iteration l is defined as $T(l) = \frac{T_0}{\log(l+1)}$. When a candidate basis fails to have an index value larger than the current

basis, simulated annealing gives it a second chance to be accepted with probability

$$P = \min \left\{ \exp \left[-\frac{|I_{\text{cur}} - I_l|}{T(l)} \right], 1 \right\}$$

where $I(\cdot)$ denotes the index value of a given basis. This implementation allows the algorithm to jump out of a local maximum and enables a more holistic search of the whole parameter space. This feature is particularly useful when local maxima are present. The algorithm can be written as replacing line 5-8 of Algorithm 1 with Algorithm 2.

Algorithm 3: search geodesic

```

input :  $\mathbf{A}_{\text{cur}}, f, l_{\text{max}}, n = 5, \delta$ 
output:  $\mathbf{A}_{**}$ 
1 initialisation;
2 Set  $l = 1$ ;
3 while  $l < l_{\text{max}}$  do
4   Generate  $2n$  bases in a small neighbourhood,  $\delta$ , of  $\mathbf{A}_{\text{cur}}$  and ensure orthogonality ;
5   Find the one with the largest index value:  $\mathbf{A}_*$ ;
6   Construct the geodesic  $\mathcal{G}$  from  $\mathbf{A}_{\text{cur}}$  to  $\mathbf{A}_*$ ;
7   Optimise the index value on the geodesic  $\mathcal{G}$  over a 90 degree window to produce
      the optima  $\mathbf{A}_{**}$  ;
8   Compute  $I_{**} = f(\mathbf{A}_{**})$ ,  $p_{\text{diff}} = (I_{**} - I_{\text{cur}}) / I_{**}$ ;
9   if  $p_{\text{diff}} > 0.001$  then
10    | return  $\mathbf{A}_{**}$  ;
11   end
12    $l = l + 1$ ;
13 end
```

Cook et al. (1995) used a gradient ascent algorithm on the space of the projection bases. In gradient ascent, one first finds the direction for improvement via computing the gradient information. In search_geodesic, $2n$ bases are first generated in a tiny neighbourhood of the current basis, controlled by the neighbourhood parameter δ . A geodesic is then constructed using the current basis and the one in $2n$ bases with the highest index value. If the neighbourhood parameter δ is tiny, the geodesic constructed is an analogue of the gradient information in the curved space and works as the searching direction. The next step in gradient ascent is to conduct a line search to find the best improvement along the gradient direction and in search_geodesic, this is replaced by optimising the index value along the geodesic direction over an 90 degree angle from $-\pi/4$ to $\pi/4$. The optima \mathbf{A}_{**} is returned for the current iteration if it meets the termination condition on percentage improvement. The procedure will also terminate if l_{max} is reached. Algorithm 3 summarises the steps in geodesic search.

Visual diagnostics

To be able to make diagnostics on the optimisers, the algorithms need to populate a data structure with key elements of the algorithm. When the algorithms run, key information regarding the decision variable, objective function and hyper-parameters, needs to be recorded and stored as a data object so that it is ready to be supplied to the plotting functions for diagnostics.

Data structure for diagnostics

In the optimisation algorithms for projection pursuit, the three main elements to record are 1) projection bases: \mathbf{A} , 2) index values: I , and 3) State: S , which labels the observation with detailed stage in the optimisation. Possible values for search_better and search_better_random include random_search, new_basis, and interpolation. search_geodesic has a wider variety that includes new_basis, direction_search, best_direction_search, best_line_search, and interpolation.

Multiple iterators are also needed to index the data collected at different levels. t is a unique identifier that prescribes the natural ordering of each observation; j is the counter for each search-and-interpolate iteration, which remains the same within one round and has an increment of one once a new round starts. l is the counter for each search/interpolation, which provides the information of how many bases the algorithm has searched before finding one to return. There are other parameters

of interest, which depend on the particular problem content and they are denoted as V_p . Two most common examples include $V_1 = \text{method}$, which tags the name of the algorithm used, and $V_2 = \text{alpha}$, the neighbourhood parameter that controls the size in sampling candidate bases. A matrix notation of the data structure is presented in Equation 3.

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline t & \mathbf{A} & I & S & j & l & V_1 & V_2 \\ \hline 1 & \mathbf{A}_1 & I_1 & S_1 & 1 & 1 & V_{11} & V_{12} \\ \hline 2 & \mathbf{A}_2 & I_2 & S_2 & 2 & 1 & V_{21} & V_{22} \\ \hline 3 & \mathbf{A}_3 & I_3 & S_3 & 2 & 2 & V_{31} & V_{32} \\ \hline \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & 2 & l_2 & \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & 2 & 1 & \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & 2 & 2 & \vdots & \vdots \\ \hline \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & 2 & k_2 & \vdots & \vdots \\ \hline \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & J & 1 & \vdots & \vdots \\ \hline \vdots & \vdots \\ \hline T & \mathbf{A}_T & I_T & S_T & J & l_J & V_{T1} & V_{T2} \\ \hline \vdots & \vdots & \vdots & \vdots & J & 1 & \vdots & \vdots \\ \hline \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & J & k_J & \vdots & \vdots \\ \hline \vdots & \vdots & \vdots & \vdots & J + 1 & 1 & \vdots & \vdots \\ \hline \vdots & \vdots \\ \hline T' & \mathbf{A}_{T'} & I_{T'} & S_{T'} & J + 1 & l_{J+1} & V_{T'1} & V_{T'2} \\ \hline \end{array} = \begin{array}{|c|c|} \hline \text{column name} & \\ \hline \text{search (start basis)} & \\ \hline \text{search} & \\ \hline \text{search} & \\ \hline \vdots & \\ \hline \text{search (accepted basis)} & \\ \hline \text{interpolate} & \\ \hline \text{interpolate} & \\ \hline \vdots & \\ \hline \text{interpolate} & \\ \hline \vdots & \\ \hline \text{search} & \\ \hline \vdots & \\ \hline \text{search (final basis)} & \\ \hline \text{interpolate} & \\ \hline \vdots & \\ \hline \text{interpolate} & \\ \hline \text{search (no output)} & \\ \hline \vdots & \\ \hline \text{search (no output)} & \\ \hline \end{array} \quad (3)$$

where $T' = T + k_J + l_{J+1}$. Note that there is no output in iteration $J + 1$ since the fact that it is the last iteration means that the optimiser cannot find a better basis and the algorithm terminates. In this notation, the final basis found is A_T with the highest index value I_T .

The data structure constructed above meets the tidy data principle (Wickham et al., 2014) that requires each observation forms a row and each variable forms a column. With tidy data structure, data wrangling and visualisation have been significantly simplified by well-developed packages such as dplyr(Wickham et al., 2020) and ggplot2(Wickham, 2016).

The construction of diagnostic plots uses the concept of grammar of graphics (Wickham, 2010) in ggplot2. In grammar of graphics, plots are not defined by their appearance (i.e. boxplot, histogram, scatter plot etc) but by stacked layers. In the construction of diagnostic plots, there are multiple elements one may want to emphasize and there is no single plot name that would meet the need. The stacked layers concept, on the other hand, allows information to be overlaid on each other and one can build the plot from scratch as long as the variables have been stored in a dataset.

Checking how hard the optimiser is working

A primary interest of diagnosing an optimiser is to study how it progressively finds its optimum. Directly plotting the index value across its natural order will cause the graph to be disproportional to the iteration since it usually takes longer for an optimiser to find a better basis towards the end. Another option is to use summarisation for each iteration. Boxplot is a suitable candidate that can provide five points summary of each iteration. Other additional information not presented in the boxplot can then be added with new layers, for example text information on the number of points can be added at the bottom of each iteration and the position of basis returned by projection pursuit can be highlighted in point. Further, an option to switch between displaying points and boxplot geometry is helpful when the number of observation is small in one iteration and this is achieved via a cutoff parameter.

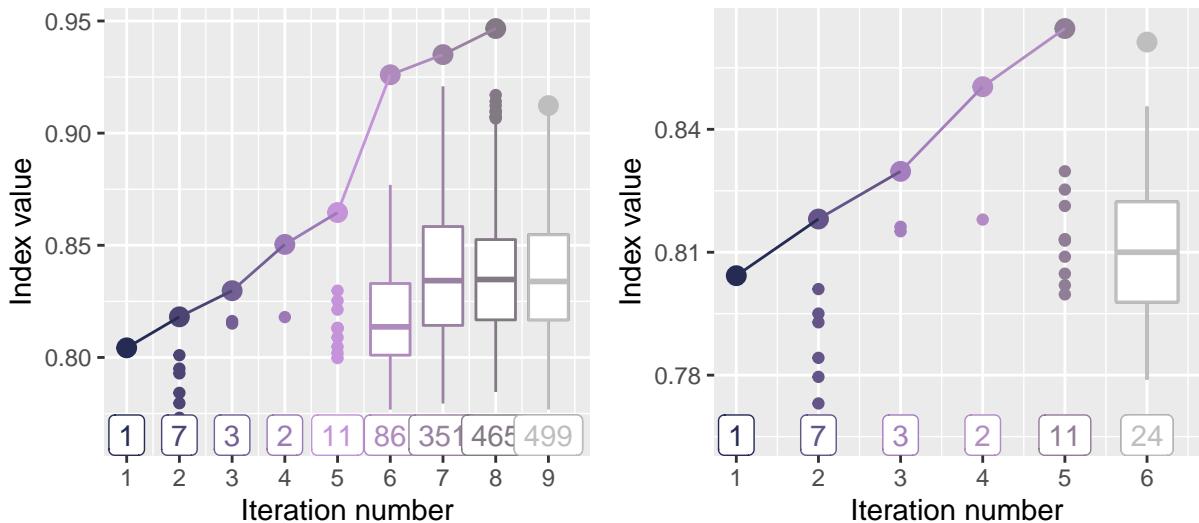


Figure 2: A comparison of `search_better` with different values of the parameter `max_tries`. A six-variable dataset `boa6` is used with holes index on a 2D problem. `max_tries` = 500 allows the optimiser to find better basis with higher index value at iteration six.
(#fig:toy-search)

Figure 2 shows a sample of the plot constructed using `search_better` with different values of the parameter `max_tries`. Comparing the returned index value of each iteration shows that the termination at `max_tries` = 25 is not sufficient for `search_better` to explore the parameter space and a value of 500 is preferred over 25 in this context.

Examining the optimisation progress

(ref:toy-interp) The resulting trace plot on the interpolated points has been plotted when using three different algorithms to optimise the index. The color represents the number of iteration. It can be observed that each algorithm differs in length in the optimisation and the curvature of the improvement for each algorithm also varies.

Points on the interpolation path are another interest in tour since the projection on these bases will be played by the tour animation. Figure ?? presents the interpolation of three different tour paths each with different curvature. The leftmost plot shows an interpolation where the index value increases progressively and monotonically. The middle path has an increase-then-decrease pattern in the last two iterations and the rightmost path shows even a decrease in the index value at iteration three. The middle situation can be avoid via a construction of the interruption, which will be detailed in section 2.5.2 and the rightmost case is a deliberate construction of `search_better_random` where an inferior basis can be accepted with some probability so as to avoid being trapped in a local maximum.

Understanding the optimiser's coverage of the search space

(ref:toy-pca) 1D projection on the 5-variable dataset `boa5` with two optimisers: `search_better` and `search_geodesic`. The yellow point corresponds to the theoretical best basis [0, 1, 0, 0, 0] with V2 being the only non-normal variable in the dataset. The underlying grey points are randomly generated on the 5D space and reduced to 2D via PCA along with all the search points presented. The enlarged color points that colored as the interpolation are the starting points of each algorithm. They are initially simulated with the same starting points but all the bases in `search_geodesic` has been flipped positive to ensure that bases with the same projected images but a sign difference are represented by the same point in the plot.

Apart from checking the progression of an optimiser, another interesting aspect is to visualise how the search looks like in its parameter space. Given the orthonormality constraint, the projection bases $\mathbf{A}_{p \times d}$ live on the surface of a $p \times d$ dimension sphere, where the dimension can easily go over two or three. While visualising the search paths on the original high dimensional sphere would require skills for the viewers to perceive rotation of geometry in higher dimensional space ($d > 3$), an easier alternative is to view the reduced space via some dimension reduction methods, i.e. principal

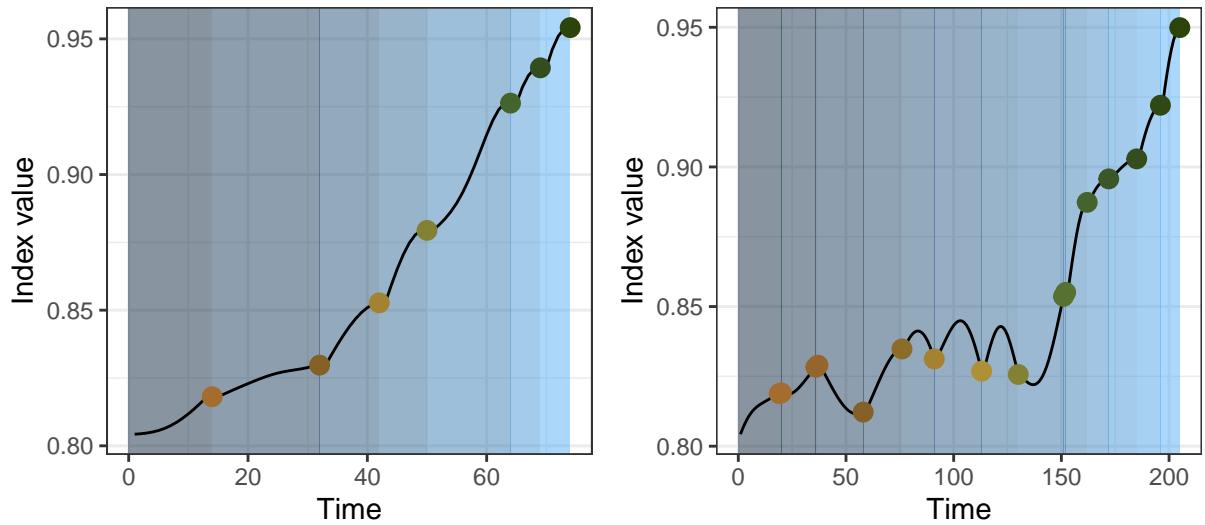


Figure 3: (ref:toy-interp)

(#fig:toy-interp)

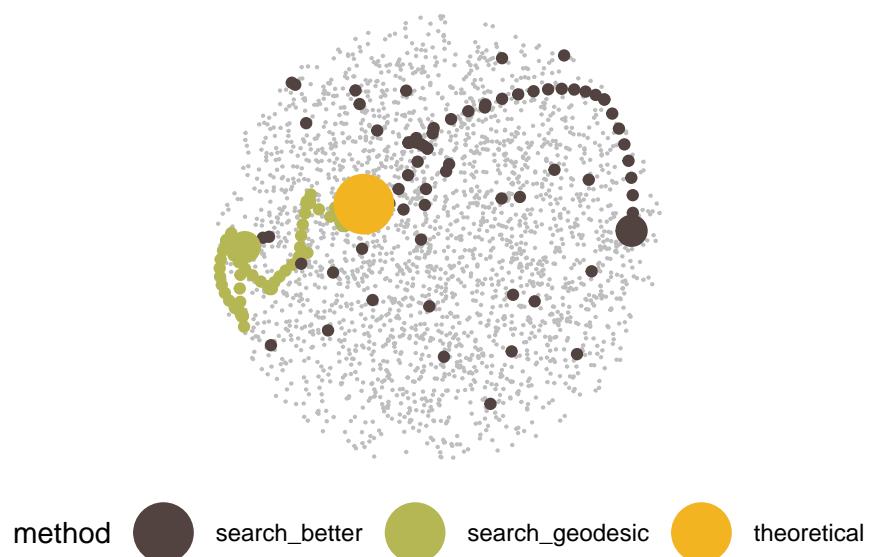


Figure 4: (ref:toy-pca)

(#fig:toy-pca)

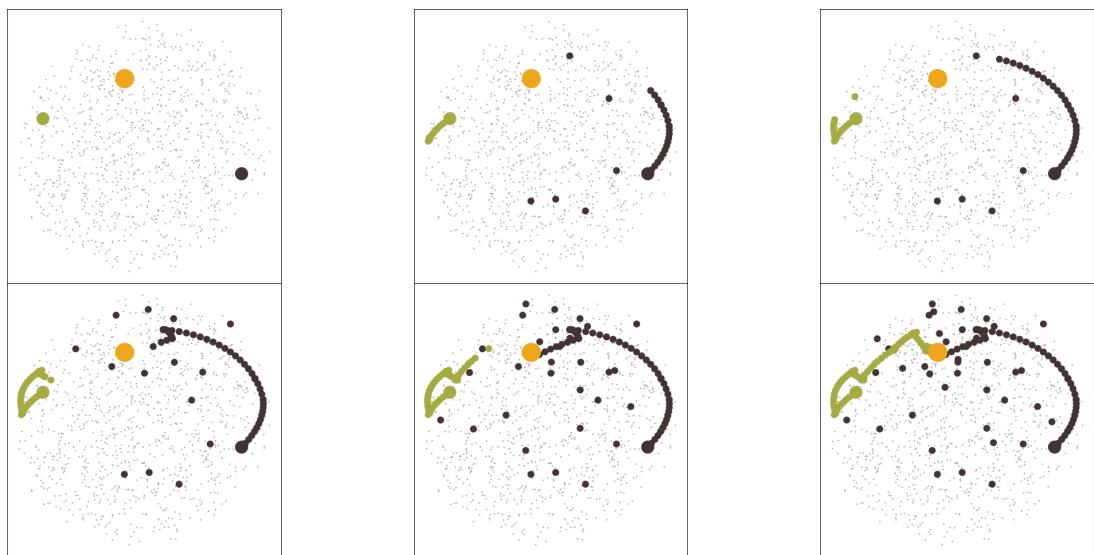


Figure 5: (ref:toy-pca-animated)
(#fig:toy-pca-animated)

component analysis. To better perceive the search path as an embedding of a hollow sphere, random points on the high dimensional sphere is generated using the package `geozoo` and PCA is conducted on both the bases and the points on the surface of the sphere.

Figure @ref(fig:toy-pca) plots the first two principal components of two search paths, one using `search_better` and the other using `search_geodesic`. The search in the space reduced by PCA matches with the optimiser description before where the random sampling in `search_better` is broader, controlled by `alpha` parameter, which is default to 0.5 while the directional search in `search_geodesic`, controlled by `delta` with a default of 0.01, is so tiny that it can barely be seen.

Animating the diagnostic plots

(ref:toy-pca-animated) A selected number of frames from the animated PCA plot. With animation, it is easier to track the progression from the start to finish in each algorithm.

Animated plots can be informative in diagnostics, especially in the case of PCA plot when the starting and ending of the search is not clear. Figure @ref(fig:toy-pca-animated) shows six frames of an animated version of Figure @ref(fig:toy-pca) and this time, it shows that `search_better` finds the optimum quicker than `search_geodesic`.

The tour looking at itself

(ref:toy-tour) A selected number of frames from the tour animation for viewing the 5D space of all the projection bases. The second frame on the top row views the space from a direction that is close to the one in the PCA plot. The tour animation allows for a more holistic view of the full space in high dimensions from different angles.

While viewing the bases on the reduced space via PCA shed some lights on the space the optimisers have explored, the visualisation on the original $p \times d$ dimension enables a more holistic stereoscopic view of the search. To view a high dimensional ($d \geq 3$) object on a screen, an approach is to play the rotation of the object in animation and this can be done via a regular grand tour. Compared to the PCA plot, the animated rotation (tour) displayed in Figure @ref(fig:toy-tour) gives a more well-rounded view of the search and one can view the curved region of the tour path from different angles, which may not be presented in the PCA plot. Also the grand tour animation encompasses the PCA projection since the rotation from PCA is just one angle that maximises the variance of the bases and the grand tour produces a sequence of angles that view the search from different directions. As an evidence, the last frame in Figure @ref(fig:toy-tour) is a frame select from the tour animation that is close to the PCA angle and the projection looks similar to the one in Figure @ref(fig:toy-pca).



Figure 6: (ref:toy-tour)
(#fig:toy-tour)

Diagnosing an optimiser

For a particular index function, the best algorithm to optimise relates to the character of the index and the data. If the index function is smooth and has a single maximum, all of the three algorithms introduced above can find the maximum. When multiple optima are present, `search_better` may get stuck at a local maximum and in the case where the index function is non-smooth, `search_geodesic` may even fail to find the maximum. In this section, examples will be presented to outline how the diagnostic plots can be used to compare the performance of optimisers in different scenarios.

Simulation setup

Random variables with different structures have been simulated and the distribution of each is presented in Equations 4 to 10. Variable x_1, x_8, x_9 and x_{10} are normal distributed with zero mean and unit variance and x_2 to x_7 are mixtures of normal distributions with varied weights and locations. The mixture variables have been scaled to have an overall unit variance before running the projection pursuit.

$$x_1 \stackrel{d}{=} x_8 \stackrel{d}{=} x_9 \stackrel{d}{=} x_{10} \sim \mathcal{N}(0, 1) \quad (4)$$

$$x_2 \sim 0.5\mathcal{N}(-3, 1) + 0.5\mathcal{N}(3, 1) \quad (5)$$

$$\Pr(x_3) = \begin{cases} 0.5 & \text{if } x_3 = -1 \text{ or } 1 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$x_4 \sim 0.25\mathcal{N}(-3, 1) + 0.75\mathcal{N}(3, 1) \quad (7)$$

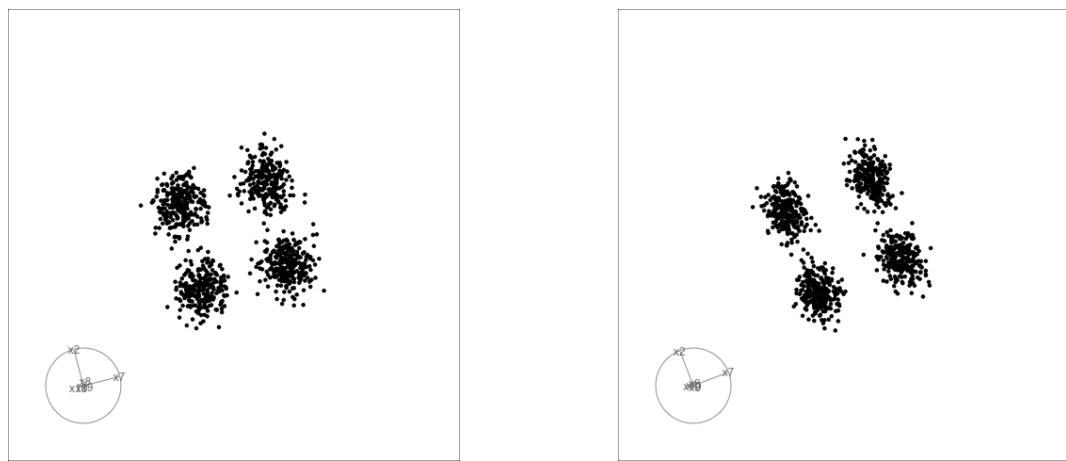
$$x_5 \sim \frac{1}{3}\mathcal{N}(-5, 1) + \frac{1}{3}\mathcal{N}(0, 1) + \frac{1}{3}\mathcal{N}(5, 1) \quad (8)$$

$$x_6 \sim 0.45\mathcal{N}(-5, 1) + 0.1\mathcal{N}(0, 1) + 0.45\mathcal{N}(5, 1) \quad (9)$$

$$x_7 \sim 0.5\mathcal{N}(-5, 1) + 0.5\mathcal{N}(5, 1) \quad (10)$$

A problem of non-monotonicity

In section 2.4.3, an interpolation with increase-then-decrease pattern has been presented. This pattern is undesirable since the optimiser could have started the next iteration from the highest basis on the tour path, as annotated as the interpolated basis in the plot, but instead, it is forced to start from the target basis. This motivates the design of an interruption to check the index value on the tour path so that the interpolating bases is accepted only up to the one with the largest index value. After

**Figure 7:** (ref:polish)

(##fig:polish)

implementing this interruption, the search finds a higher final index value with fewer steps as shown in the right panel of Figure @ref(fig:interruption).

(ref:interruption) Two-D projection on `boa6` data with holes index optimised by `search_better`. A comparison of the trace plot with and without interruption. When the interruption does not take place, the index value of the target basis can be smaller than the interpolated basis. The interruption forces the interpolation to finish at the highest interpolated basis on the tour path.

Close but not close enough

(ref:polish) Two-D projection on `boa6` data with holes index optimised by `search_geodesic`. The left panel shows the final projected data before polish and the right panel shows the one after. The separation of the clusters on the y axis becomes sharper after the polish.

Once the final basis has been found by an algorithm, one may want to push further to investigate whether there is an even better basis in the close neighbourhood. This motivates the polish search where the final basis is supplied as the start of a new guided tour to search for any local breakthrough.

Similar to `search_better` as a stochastic random search, `search_polish` has a different scheme of reducing the search neighbourhood. In each search-interpolation iteration, `search_better` has a fixed neighbourhood parameter alpha and this alpha is reduced by the cooling parameter only after an iteration finishes. On the contrary, `search_polish` allows alpha to be reduced during each iteration to exploit the search in the neighbourhood. Further, to avoid the case where alpha becomes too small and the further search is meaningless, three more stopping criteria have been added, on top of the original `max.tries` limit. These include:

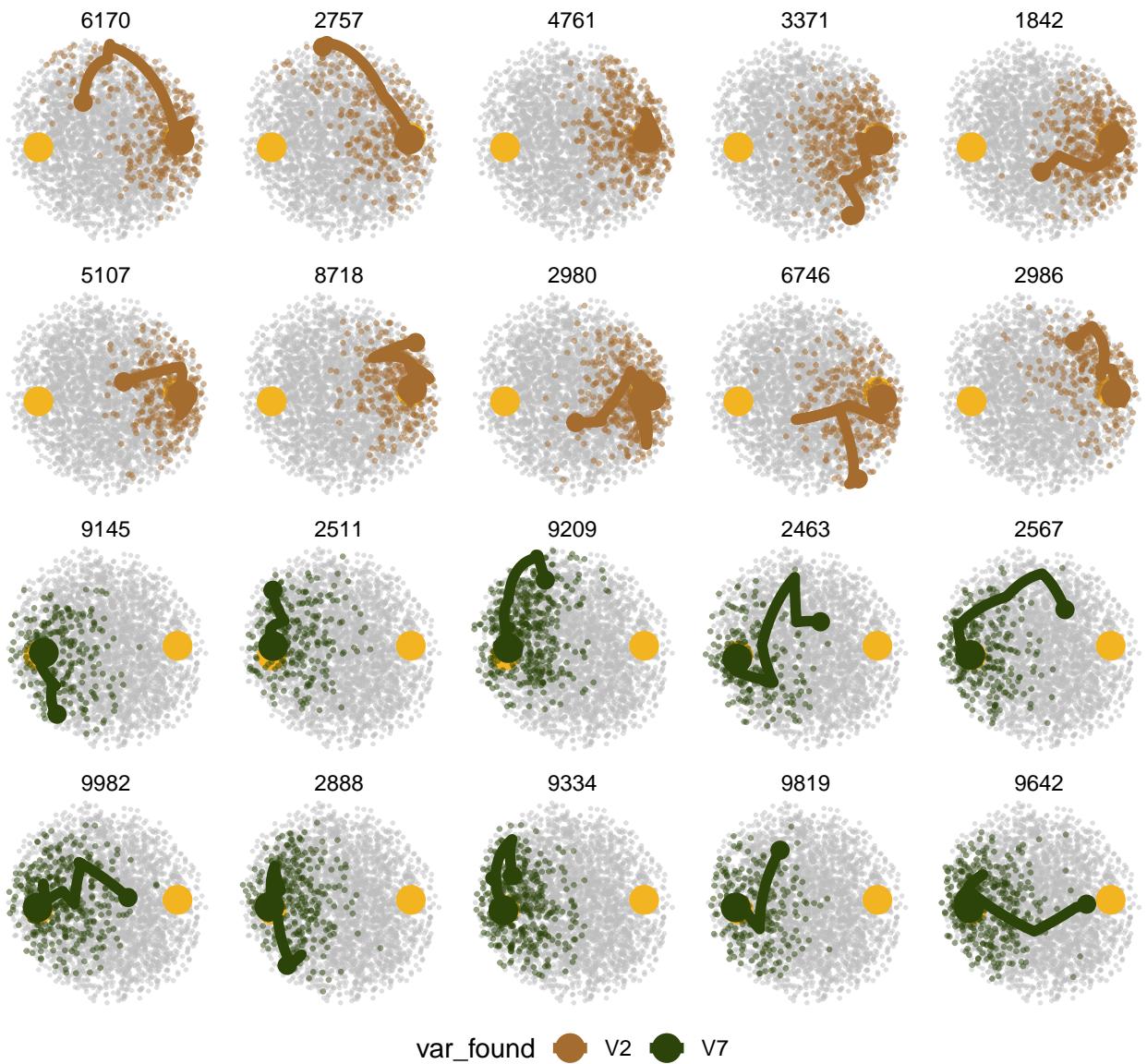
- 1) the distance between the candidate basis and the current basis needs to be larger than 1e-3;
- 2) the percentage change of the index value need to be larger than 1e-5; and
- 3) the alpha parameter on itself needs to be larger than 0.01

Figure @ref(fig:polish) presents the final projections found before and after applying `search_polish` on `search_geodesic`. Polish search improves the index value from 0.9618 to 0.9627 with reduction of weights on the non-informative variables. In terms of the projected data as in Figure @ref(fig:polish), polish works to sharpen the edges of each cluster.

Seeing the signal in the noise

(ref:noisy-better-geo) One-D projection on `boa5` data with noisy index `norm_kol` optimised by `search_geodesic` and `search_better`. The grey dashed line represents the index value of the theoretical best basis. `search_geodesic` fails to optimise the noisy index while `search_better` has made reasonable improvements to reach the index value close to the one of the theoretical best basis.

The index function, up until this point, are all smooth, while this is not the case for all the index functions. `norm_kol`, a 1D projection function based on the Kolmogorov test, compares the difference between the 1D projected data, $\mathbf{Y}_{n \times 1}$ and a randomly generated normal distribution, y_n based on

**Figure 8:** (ref:kol-better)

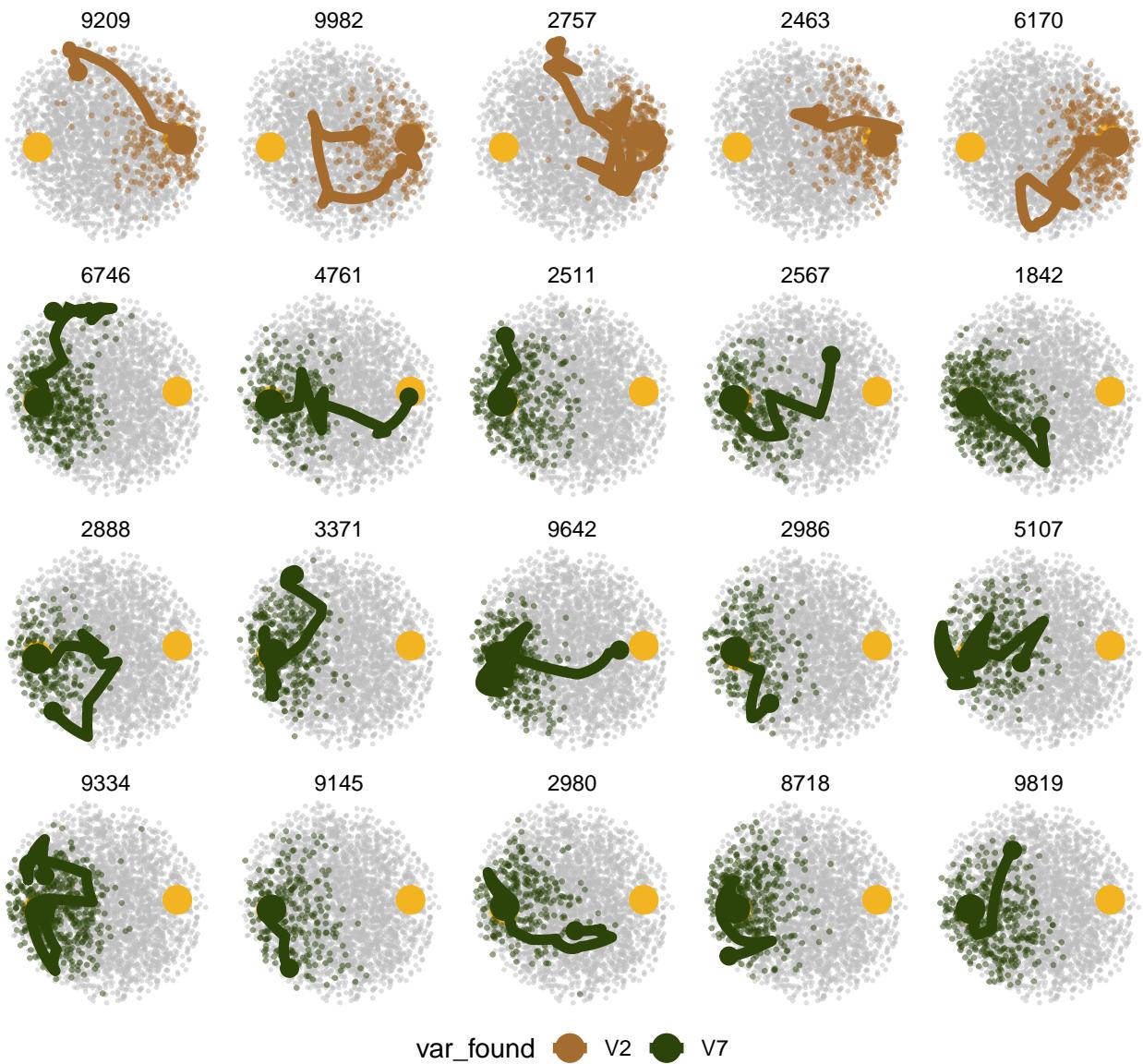
(##fig:kol-better)

the empirical cumulated distribution function (ECDF). Denote the ECDF function as $F_p(u)$ with the subscript indicating either the projection or the random normal variable, the `norm_kol` index is defined by

$$\max [F_p(u) - F_y(u)]$$

Figure @ref(fig:noisy-better-geo) compares the tracing plot of two optimisers: `search_geodesic` and `search_better`. This time, the interpolated path is no longer smooth when using either algorithm and `search_geodesic` fails to optimise this index with barely any improvement of the index value. On the other hand, `search_better` is doing relatively well. With the theoretical best basis $[0, 1, 0, 0, 0]$ producing an index value of 0.17, `search_better` finds the final basis $[0.0376, -0.9916, -0.0581, -0.0831, 0.0716]$ with an index value of 0.165. A further polish step will give a marginal improvement of index value to 0.175 with a basis of $[0.0223, -0.9965, -0.0352, -0.0591, 0.0418]$. At this stage, the difference between the theoretical best and what has been found is likely due to simulation error since the best possible basis for a simulated data will be slightly off the theoretical best basis, which is derived based on the distributional assumptions in Equations 4 to 10.

(ref:kol-better) One-D projection of `norm_kol` index on `boa6` data optimised by `search_better` with 20 randomly generated seeds. Each point represents a basis in the original 6D space, reduced to 2D by PCA. The grey points are random bases generated from 6-D hollow sphere and the two yellow points represent the local maximum corresponds to basis when V2 is found and the global maximum

**Figure 9:** (ref:kol-random)

(#fig:kol-random)

where V7 is found. The points produced by the search algorithm is colored by whether the global or local maximum is found with the interpolated bases highlighted as a path and the final basis as an amplified point.

(ref:kol-random) One-D projection of norm_ko1 index on boa6 data optimised by `search_better_random` with the same seeds. `search_better_random` has a probabilistic acceptance implementation that would also accept a basis with lower index value. This design allows the optimiser to jump out of the local maximum and hence more instances find the global maximum.

(ref:kol-random-tuned) One-D projection of norm_ko1 index on boa6 data optimised by `search_better_random` with an larger searching neighbourhood of 0.7. Further tuning of the parameter has been taken place to adjust the neighbourhood parameter alpha from 0.5 to 0.7.

The second experiment with the noisy index is to understand how the optimisers perform when a local maximum is present. The dataset used is boa6 where x_2 and x_7 are informative. The two theoretical best bases are $[0, 1, 0, 0, 0, 0]$ and $[0, 0, 1, 0, 0, 0]$ with index value 0.176 and 0.235, respectively. Hence, the global maximum happens when variable x_7 is found. Simulation is done with 20 randomly generated seeds on two optimisers: `search_better` (Figure @ref(fig:kol-better)) and `search_better_random` (Figure @ref(fig:kol-random) and Figure @ref(fig:kol-random-tuned)). Compared to Figure @ref(fig:kol-better) and @ref(fig:kol-random), Figure @ref(fig:kol-random-tuned) further increases the neighbourhood parameter alpha from 0.5 to 0.7 to enlarge the search space. The data object is collected for each simulation and computation is done to reduce the bases from its

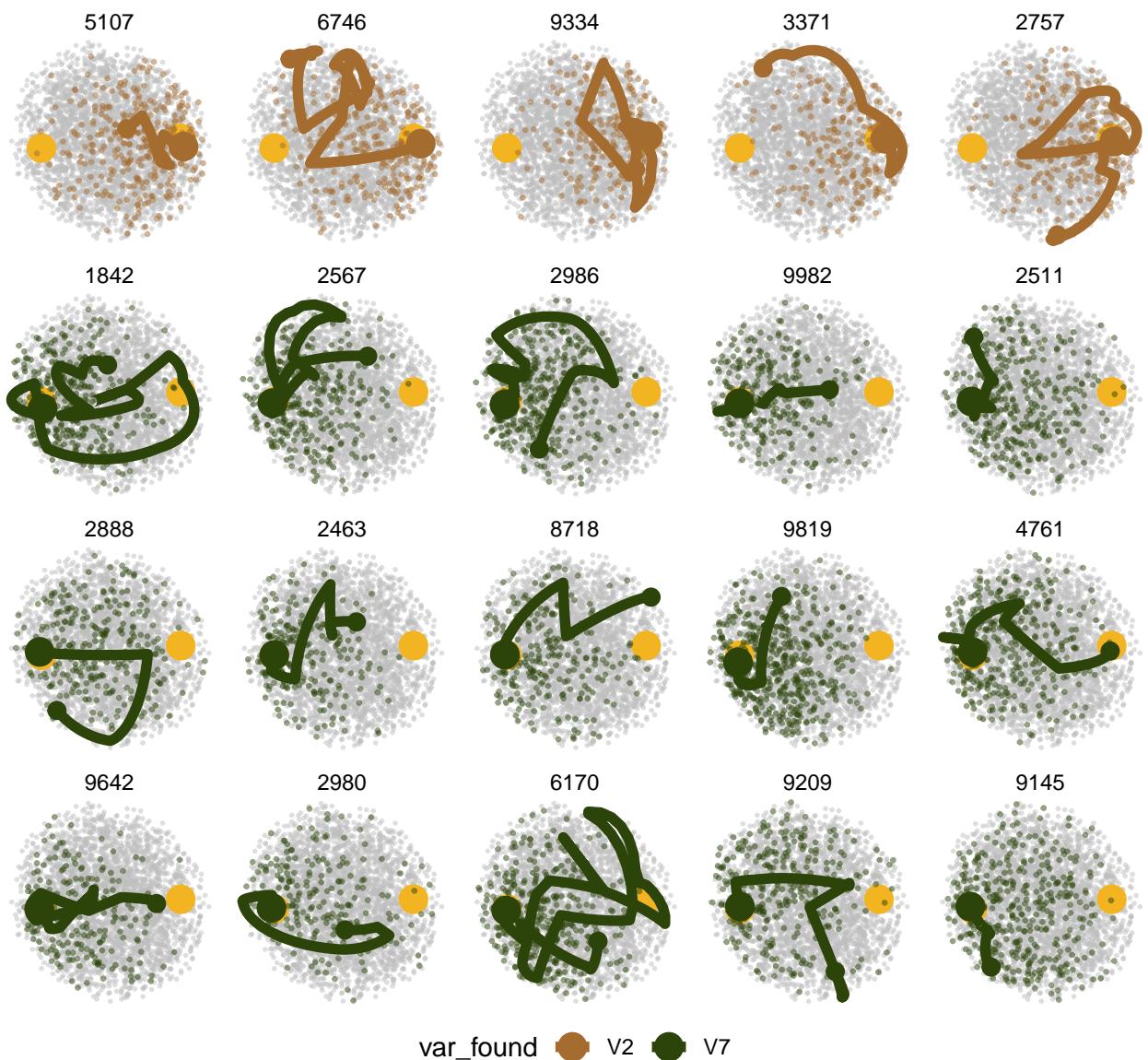


Figure 10: (ref:kol-random-tuned)
(#fig:kol-random-tuned)

original 6D to a 2D points by principal component analysis so as to view the search paths.

In Figure @ref{fig:kol-better}, the starting point largely affects whether `search_better` will find the global maximum. The three seeds on row three: seed 9145, seed 2511, and seed 9209, and three on row four: seed 2888, seed 9334, and seed 9819 easily find V7 because they are born in Rome. On the other end, the ones on the first two rows can only find V2, largely due to the fact that they do not even have chances to search near V7.

A comparison between `search_better` and `search_better_random` shows all four combinations where both, neither, or one of the two algorithms find the global maximum. Two iconic cases that shows how `search_better_random` improves `search_better` are seeds 4761 (position (1,4) in Figure @ref{fig:kol-better}) and (2,2) in Figure @ref{fig:kol-random}) and seed 1842 ((2,2) in Figure @ref{fig:kol-better}) and (4,5) in Figure @ref{fig:kol-random}). In both cases, `search_better_random` accepts inferior points at the initial iteration and this acceptance has later been proved to change the direction of the search and hence allows the optimiser to explore near the global optimum and in the end, find the global optimum. However, this change of search direction may sometimes have an adverse effect on `search_better_random` and change the search to the neighbourhood of V2. This happens on seed 9209 and seed 9982 (position (1,1) and (1,2) in Figure @ref{fig:kol-random}). The two cases that neither algorithm finds the global optimum happens on seed 6170 and seed 2757 (position (1,1) and (1,2) in Figure @ref{fig:kol-better}) and position (1,5) and (1,3) in Figure @ref{fig:kol-random}). This is largely due to the fact that the searching neighbourhood is not large enough to allow the space near V7 to be adequately explored.

In Figure @ref{fig:kol-random-tuned}, the neighbourhood parameter alpha is increased from a 0.5 default to 0.7 to solve the insufficiency of search neighbourhood. Remember that a candidate basis is generated via a linear combination of the current basis and a randomly generated basis on the surface of the sphere, an increase of alpha gives more weights on the randomly generated bases and hence allows a wider search. Seed 9982 ((2,4) in Figure @ref{fig:kol-random-tuned}) and seed 2463 ((3,2) in Figure @ref{fig:kol-random-tuned})) benefit from this and find the global maximum. A noticeable feature of this enlarged search space is that the interpolation paths start to jump around the space but its usefulness is arguable. In seed 2888 (position (3,1)), seed 2986 (position (2,3)), seed 9209 (position (4,4)), just to name a few, this allows a near V2 bases to switch to a near V7 bases and as a result these cases find the global maximum. While in the case of seed 6746 (position (1,2)) and seed 2757 (position (1,5)), the jump happens from near V7 basis to near V2 basis, it does not come back and leaves those simulations in the local maximum.

The conclusion from this experiment is that the usage of `search_better_random` and increasing the search space are methods that can avoid getting trapped in local maxima but the solution will still depend on the starting points of the simulation and the seed used.

Implementation

The implementation of this projection has been divided into two packages: the data collection object is implemented in the existing `tourr` package while the optimiser diagnostics have been implemented in a new package, `ferrn`. When a guided tour is run, the users can choose if the data from optimisation should be collected via the `verbose` argument. Once the data object has been obtained, the package, `ferrn`, can provide four diagnostic plots as shown in Section 2.4. The structure of package functionality has been listed below.

- `explore_trace_search()`: produces summary plots, as shown in Figure @ref{fig:toy-search}
- `explore_trace_interp()`: produces trace plots for the interpolation points, as shown in Figure @ref{fig:toy-interp})
- `explore_space_pca()`: produces plots of projection basis on the reduced space by PCA, as shown in Figure @ref{fig:toy-pca}. Animated version in Figure @ref{fig:toy-pca-animated} can be turned on via the argument `animate = TRUE`
- `explore_space_tour()`: produces animated tour view on the full space of the projection bases, as shown in Figure @ref{fig:toy-tour}).
- `get_*`() extracts and manipulates certain components from the existing data object.
 - `get_best()`: extracts the best basis found in the data object
 - `get_start()`: extracts the starting basis
 - `get_interp()`: extracts the observations in the interpolation
 - `get_search_count()`: produces the summary table of the number of observation in each iteration
 - `get_basis_matrix()`: flattens all the bases into a matrix

- `bind_*`() incorporates additional information outside the tour optimisation into the data object.
 - `bind_theoretical()`: incorporates the best possible basis to the existing data object with the supply of the index function and original data for producing the index value.
 - `bind_random()`: generates 1000 points on the high dimensional surface of a sphere and binds it to the existing data object and output as a tibble object. `bind_random_matrix()` binds the points to the basis matrix.
- Color
 - `botanical_palettes`: a collection of color palettes from Australian native plants. Quantitative palettes include daisy, banksia and cherry and sequential palettes contain fern and acacia.
 - `botanical_pal()`: a color interpolator
 - `scale_color_botanical()`: a ggplot construction for using botanical palettes.

Conclusion

This paper has illustrated setting up a data object that can be used for diagnosing a complex optimisation procedure. The ideas were illustrated using the optimisers available for projection pursuit guided tour. Here the constraint is the orthonormality condition of the projection bases. The approach used here could be broadly applied to understand other constrained optimisers.

Four diagnostic plots have been introduced to investigate the progression and the projection space of an optimiser. The implementation of these visualisations is designed to be easy-to-use with each plot can be produced with a simple supply of the data object. More advanced users may decide to modify on top of the basic plots or even build their own.

Most of the work in this project has been translated into code in two packages: the collection of the data object is implemented in the existing `tourr` package; manipulation and visualisation of the data object are implemented in the new `ferrn` package. Equipped with handy tools to diagnose the performance of optimisers, future work can extend the diagnostics to a wider range of index functions, i.e. scagnostics, association, and information index (Laa and Cook, 2020) and understand how the optimisers behave for index functions with different structures.

Bibliography

- D. Asimov. The Grand Tour: A Tool for Viewing Multidimensional Data. *SIAM Journal of Scientific and Statistical Computing*, 6(1):128–143, 1985. [p2]
- D. Bertsimas, J. Tsitsiklis, et al. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993. [p4]
- A. Buja, D. Cook, D. Asimov, and C. Hurley. Computational methods for high-dimensional rotations in data visualization. *Handbook of Statistics*, 24:391–413, 2005. [p1, 3]
- D. Cook and A. Buja. Manual controls for high-dimensional data projections. *Journal of Computational and Graphical Statistics*, 6(4):464–480, 1997. [p3]
- D. Cook, A. Buja, and J. Cabrera. Projection pursuit indexes based on orthonormal function expansions. *Journal of Computational and Graphical Statistics*, 2(3):225–250, 1993. [p1, 3]
- D. Cook, A. Buja, J. Cabrera, and C. Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155–172, 1995. [p5]
- D. Cook, A. Buja, E.-K. Lee, and H. Wickham. Grand tours, projection pursuit guided tours, and manual controls. In *Handbook of Data Visualization*, pages 295–314. Springer, 2008. [p3]
- R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013. [p2]
- J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 100(9):881–890, 1974. [p2, 3]
- P. Hall et al. On polynomial-based projection indices for exploratory projection pursuit. *The Annals of Statistics*, 17(2):589–605, 1989. [p3]
- D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998. [p1]
- C. T. Kelley. *Iterative methods for optimization*. SIAM, 1999. [p1]
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983. [p4]
- U. Laa and D. Cook. Using tours to visually investigate properties of new projection pursuit indexes with application to problems in physics. *Computational Statistics*, pages 1–35, 2020. [p16]
- E. Lee, D. Cook, S. Klinke, and T. Lumley. Projection pursuit for exploratory supervised classification. *Journal of Computational and Graphical Statistics*, 14(4):831–846, 2005. [p3]
- E.-K. Lee and D. Cook. A projection pursuit index for large p small n data. *Statistics and Computing*, 20 (3):381–392, 2010. [p3]
- C. Posse. Projection pursuit exploratory data analysis. *Computational Statistics & Data Analysis*, 20(6): 669–687, 1995. [p3, 4]
- Z.-J. Shi. Convergence of line search methods for unconstrained optimization. *Applied Mathematics and Computation*, 157(2):393–405, 2004. [p2]
- L. N. Trefethen and D. Bau III. *Numerical linear algebra*, volume 50. Siam, 1997. [p2]
- H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1): 3–28, 2010. doi: 10.1198/jcgs.2009.07098. [p6]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p6]
- H. Wickham, D. Cook, H. Hofmann, and A. Buja. tourr: An R package for exploring multivariate data with projections. *Journal of Statistical Software*, 40(2):1–18, 2011. URL <http://www.jstatsoft.org/v40/i02/>. [p3]
- H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL <https://CRAN.R-project.org/package=dplyr>. R package version 1.0.0. [p6]
- H. Wickham et al. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. [p6]

H.Sherry Zhang
Department of Econometrics and Business Statistics, Monash University

Huize.Zhang@monash.edu