Studying the Performance of the Jellyfish Optimiser for the Application of Projection Pursuit

Alice Anonymous[a,*], Bob Security[b], Cat Memes[b], Derek Zoolander

[a]Some Institute of Technology, Department Name, Street Address, City, Postal Code

[b]Another University, Department Name, Street Address, City, Postal Code

Abstract

This is the abstract. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum augue turpis, dictum non malesuada a, volutpat eget velit. Nam placerat turpis purus, eu tristique ex tincidunt et. Mauris sed augue eget turpis ultrices tincidunt. Sed et mi in leo porta egestas. Aliquam non laoreet velit. Nunc quis ex vitae eros aliquet auctor nec ac libero. Duis laoreet sapien eu mi luctus, in bibendum leo molestie. Sed hendrerit diam diam, ac dapibus nisl volutpat vitae. Aliquam bibendum varius libero, eu efficitur justo rutrum at. Sed at tempus elit.

Keywords: projection pursuit, optimization, jellyfish optimiser, data visualisation, high-dimensional data

Let's use British English ("American or British usage is accepted, but not a mixture of these")

1. Introduction [Nicolas and Jessica]

The artificial jellyfish search (JS) algorithm [1] is a swarm-based metaheuristic optimisation algorithm inspired by the search behaviour of jellyfish in the ocean. It is one of the newest swarm intelligence algorithms [2], which was shown to have stronger search ability and faster convergence with few algorithmic parameters compared to classic optimization methods [1]-[3].

Effective optimisation is an important aspect of many methods employed for visualising high-dimensional data (X). Here we are concerned about computing informative linear projections of high-dimensional (p) data using projection pursuit (PP) (Kruskal [4], Friedman and Tukey [5]). This involves optimising a function (e.g. Hall [6], Cook et al. [7], Lee and Cook [8], Loperfido [9], Loperfido [10]), called the projection pursuit index (PPI), that defines what is interesting or informative in a projection.

These PPI are defined on projections (XA), which means that there is a constraint that needs to be considered when optimising. A projection of data is defined by a $p \times d$ orthonormal matrix A, and this imposes the constraint on the elements of A, that columns need have norm equal to 1 and the product of columns need to sum to zero.

Cook et al. [11] introduced the PP guided tour, which enabled interactive visualisation of the optimisation in order to visually explore high-dimensional data. It is implemented in the R [12] package `tourr` [13]. The optimisation that is implemented is fairly basic, and potential problems were highlighted by Zhang et al. [14]. Implementing better optimisation functionality is a goal, but it needs to be kept in mind that the guided tour also has places importance on watching the projected data as the optimisation progresses.

*Corresponding author

Email addresses: `alice@example.com` (Alice Anonymous), `bob@example.com` (Bob Security), `cat@example.com` (Cat Memes), `derek@example.com` (Derek Zoolander)

Here we explore the potential for a jellyfish optimisation to be integrated with the guided tour. Section 2 explains the optimisation that is used in the current the projection pursuit guided tour. Section 3 provides more details on the jellyfish optimiser and formalises several characteristics of projection pursuit indexes that are help to measure optimisaer performance. Section 5 describes a simulation study on performance of the jellyfish for several types of data and index functions. Section 6 summarises the work and provides suggestions for future directions.

2. Projection pursuit, index functions and optimisation [Di and Sherry]

A tour on high-dimensional data is constructed by geodesically interpolating between pairs of planes. Any plane is described by an orthonormal basis, A_t, where t represents time in the sequence. The term "geodesic" refers to maintaining the orthonormality constraint so that each view shown is correctly a projection of the data. The PP guided tour operates by geodesically interpolating to target planes (projections) which have high PP index values, as provided by the optimiser. The geodesic interpolation means that the viewer sees a continuous sequence of projections of the data, so they can watch patterns of interest forming as the function is optimised. There are five optimisation methods implemented in the `tourr` package:

- `search_geodesic()`: provides a pseudo-derivative optimisation. It searches locally for the best direction, based on differencing the index values for very close projections. Then it follows the direction along the geodesic path between planes, stopping when the next index value fails to increase.
- `search_better()`: is a brute-force optimisation searching randomly for projections with higher index values.
- `search_better_random()`: is essentially simulated annealing [15] where the search space is reduced as the optimisation progresses.
- `search_posse()`: implements the algorithm described in Posse [16].
- `search_polish()`: is a very localised search, to take tiny steps to get closer to the local maximum.

There are several PP index functions available: `holes()` and `cmass()` [7]; `lda_pp()` [17]; `pda_pp()` [8]; `dcor2d()` and `splines2d()` [18]; `norm_bin()` and `norm_kol()` [19]; `slice_index()` [20]. Most are relatively simply defined, for any projection dimension, and implemented because they are relatively easy to optimise. A goal is to be able to incorporate more complex PP indexes, for example based on scagnostics (Wilkinson et al. [21], Wilkinson and Wills [22]).

An initial investigation of PP indexes, and the potential for scagnostics is described in Laa and Cook [23]. To be useful here an optimiser needs to be able to handle functions which are not very smooth. In addition, because data structures might be relatively fine, the optimiser needs to be able to find maxima that occur with a small squint angle, that can only be seen from very close by. One last aspect that is useful is for an optimiser to return local maxima in addition to global because data can contain many different and interesting features.

3. The jellyfish optimiser and properties of PP indexes [Nicolas and Jessica]

The jellyfish optimiser (JSO) mimics the natural movements of jellyfish, which include passive and active motions driven by ocean currents and their swimming patterns, respectively. In the context of optimization, these movements are abstracted to explore the search space in a way that balances exploration (searching new areas) and exploitation (focusing on promising areas). The algorithm aims to find the optimal solution by adapting the jellyfish's behaviour to navigate towards the best solution over iterations [1].

To understand what the jellyfish optimizer is doing in the context of Projection Pursuit, we first start with a current projection (the starting point). Then, we evaluate this projection using an index function, which tells us how good the current projection is. We then move the projection in a direction determined by the 'best jelly' and random factors, influenced by how far along we are in the optimization process (the trial i and `max.tries`). Occasionally, we might explore completely new directions like a jellyfish might with ocean currents. Then, we compare new potential projections to our current one. If they're better, we adopt them; if not, we stick with our current projection. This process continues and iteratively improves the projection, until we reach the maximum number of trials.

Algorithm: Jellyfish Optimizer Pseudo Code

Input: `current_projections`, `index_function`, `tries`, `max_tries`
Output: `optimized_projection`
Initialize `best_jelly` as the projection with the best index value from `current_projections`, and `current_index` as the array of index values for each projection in `current_projections`
for each try in 1 to max_tries **do**
    Calculate c_t based on the current try and max_tries
    if c_t is greater than or equal to 0.5 **then**
        Define trend based on the best jelly and current projections
        Update each projection towards the trend using a random factor and orthonormalisation
    else
        if a random number is greater than $1 - c_t$ **then**
            Slightly adjust each projection with a small random factor (Type A passive)
        else
            For each projection, compare with a random jelly and adjust towards or away from it (Type B active)
    Update the orientation of each projection to maintain consistency
    Evaluate the new projections using the index function
    if any new projection is worse than the current, revert to the `current_projection` for that case
        Determine the projection with the best index value as the new best_jelly
    if the try is the last one, print the final best projection and **exit**
return the set of projections with the updated best_jelly as the optimized_projection

The JSO implementation involves several key parameters that control its search process in optimization problems. These parameters are designed to guide the exploration and exploitation phases of the algorithm. While the specific implementation details can vary depending on the version of the algorithm or its application, we focus on two main parameters that are most relevant to our application: the number of jellyfish and drift.

Laa and Cook [23] has proposed five criteria for assessing projection pursuit indexes (smoothness, squintability, flexibility, rotation invariance, and speed). Since not all the properties affects the execution of the optimisation, here we consider the three relevant properties (smoothness, squintability, and speed), and propose three metrics to evaluate these three properties.

3.1. Smoothness

If we evaluate the index function at some random points (like the random initialization of the jellyfish optimizer), then we can interpret these random index values as a random field, indexed by a space parameter: the random projection angle. This analogy suggests to use this random training sample to fit a spatial model, a simple one being a (spatial) Gaussian process.

How can we define a measure of smoothness from this? The distribution of a Gaussian process is fully determined by its mean function and covariance function. The way the covariance function is defined is where smoothness comes into play: if an index is very smooth, then two close projection angles should produce close index values (strong correlation); by contrast, if an index is not smooth, then two close projection angles might give very different index values (fast decay of correlations with respect to distance between angles).

Popular covariance functions are parametric positive semi-definite functions, some of which have a parameter to capture the smoothness of the Gaussian field. In particular, consider the Matérn class of covariance functions, defined by

$$K(u) := \frac{(\sqrt{2\nu}u)^{\nu}}{\Gamma(\nu)2^{\nu-1}}\mathcal{K}_{\nu}(\sqrt{2\nu}u)$$

where $\nu > 0$ is the smoothness parameter and where $\mathcal{K}_{\nu}$ is the modified Bessel function. The Matérn covariance function can be expressed analytically when ν is a half-integer, the most popular values in the literature being 1/2, 3/2 and 5/2 . The parameter ν, called smoothness parameter, controls the decay of the covariance function. As such, it is an appropriate measure of smoothness of a random field.

In our context, we suggest to use this parameter as a measure of the smoothness of the index function by fitting a Gaussian process prior with Matérn covariance on a dataset generated by random evaluations of the index function, as in the initial stage of the jellyfish random search. There exist several R packages, such as GpGp or ExaGeoStatR, to fit the hyperparameters of a GP covariance function on data. In this project, we make use of the GpGp package.

The fitted value $\nu > 0$ can be interpreted as follows: the higher ν, the smoother the index function.

3.2. Squintability

From the literature, it is commonly understood that a large squint angle implies that the objective function value is close to optimal even when we are not very close to the perfect view to see the structure. A small squint angle means that index function value improves substantially only when we are very close to the perfect view. As such, low squintability implies rapid improvement in the index value when near the perfect view.

In this study, we propose two metrics to capture the notion of squintability.

[We generate random points that is beyond 1.5 projection distance and interpolate. Then we fit a kernel or use nonlinear least squares.]

First, parametric model.

[Nicolas's pdf]

Second, we consider the product of the largest absolute magnitude of rate of change of f and the corresponding projection angle as a second measure of squintability. Since f is decreasing, the rate of change of f is negative and thus $|\min_x f('x)|$ gives the absolute magnitude of the most negative rate of change.

[Nicolas's pdf]

To the best of our knowledge, this is the first attempt to measure the notion of squintability.

3.3. Speed

The speed of optimizing an index function can be calculated/measured using the computational complexity (in big O notation, with respect to the sample size) of computing the index function.

4. Visualisation of jellyfish optimiser

Information of the jellyfish optimiser is available in a tabular format and below is an example data collected from finding the sine-wave structure in 6D data using a distance correlation index (`docr2d_2`):

```
Rows: 10
Columns: 13
$ idx_f     <chr> "dcor2d_2", "dcor2d_2", "dcor2d_2", "dcor2d_2", "dcor2d_2", ~
$ d         <dbl> 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
$ n_jellies <dbl> 20, 20, 20, 20, 20, 20, 20, 20, 20, 20
$ max_tries <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50, 50
$ sim       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
$ seed      <int> 3462, 3462, 3462, 3462, 3462, 3462, 3462, 3462, 3462, 3462
$ basis     <list> <<matrix[6 x 2]>>, <<matrix[6 x 2]>>, <<matrix[6 x 2]>>, <<m~
$ index_val <dbl> 0.0247212373, 0.0033938502, 0.0463398915, 0.0486230801, -0.0~
$ info      <chr> "initiation", "initiation", "initiation", "initiation", "ini~
$ method    <chr> "search_jellyfish", "search_jellyfish", "search_jellyfish", ~
$ tries     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
$ loop      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
$ time      <drtn> 35.46031 secs, 35.46031 secs, 35.46031 secs, 35.46031 secs, ~
```

Information recorded can be categorised into the following categories:

- projection pursuit variables: the index function used (`idx_f`), the data dimension (`d`)
- jellyfish optimiser parameters: the number of jellies (`n_jellies`), the maximum number of tries (`max_tries`)
- simulation variables: the simulation number (`sim`), the seed used (`seed`)
- optimisation variables: the projection basis in a matrix format (`basis`), the index value (`index_val`), a description of the status - one of "initiation", "current_best", and "jellyfish_update" (`info`), current iteration ID (`tries`), current jelly ID (`loop`), and the time taken to find the optimum (`time`).

The basis column records every basis *visited* by the jellyfish optimiser prior to comparing with the current basis. In each iteration, if the index value of a visited basis is smaller than that of the current one, the jellyfish optimiser will retain the current basis for the next iteration, while still documenting the visited basis.

Numerical information to compute:

- angular distance between the projection basis and the theoretical best basis,
- the proportion of simulation that found the optimal basis,
- the proportion of jellies, within each simulation, that found the optimal basis,

Visualisation to inspect:

- inspect the basis visited by each jellyfish in the reduced PCA space,
- inspect the final 2D projections reached by each jellyfish,
- plot the index value against the angular distance between the projection basis and the theoretical best basis

Plotting the basis in the space and the projected data can help to understand 1) whether each simulation finds the same optimum or some simulations find local optima; and 2) whether the index function used can detect the structure in the data and the projection contains the structure of interest.

The visualisations above can be faceted by the projection pursuit variables and jellyfish optimiser parameters to compare the performance of different indexes to detect the same structure and how the jellyfish optimiser parameters affect the optimisation process.

[example plots]

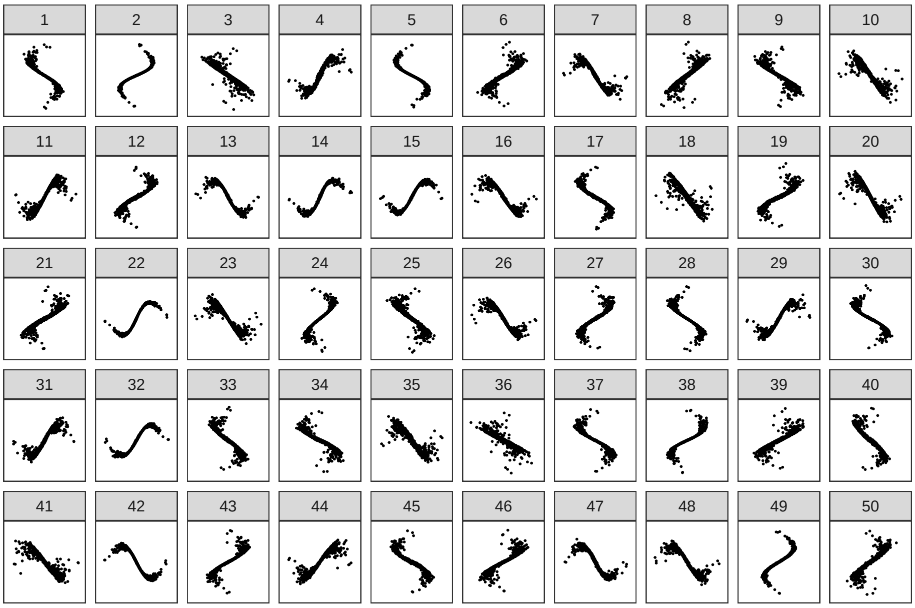

Figure 1: sdfjsflk

5. Application [Di and Sherry]

The jellyfish optimiser has been implemented in the tourr package [24] and we will use the diagnostic plots proposed in the ferrn package [14] to visualise the optimisation process.

5.1. *Going beyond 10D*

The pipe-finding problem is initially used to investigate indexes and optimisers in Laa and Cook [23], and we extend it from a 6D problem to a 12D problem.

Jellyfish optimiser, as a multi-start algorithm, is efficient in [...] for high-dimensional problems

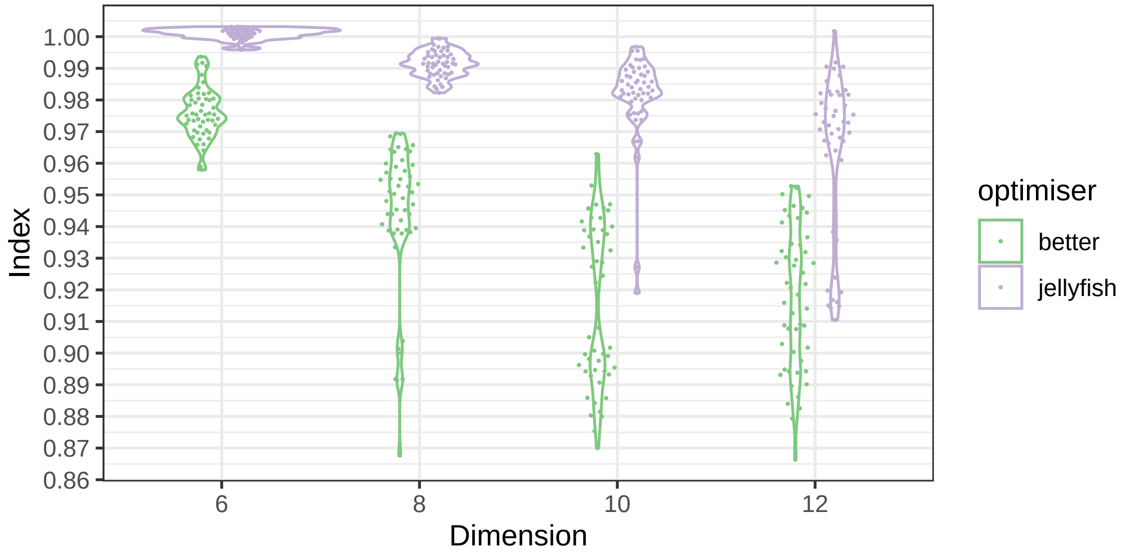

Figure 2: sthis sdfaksdlf

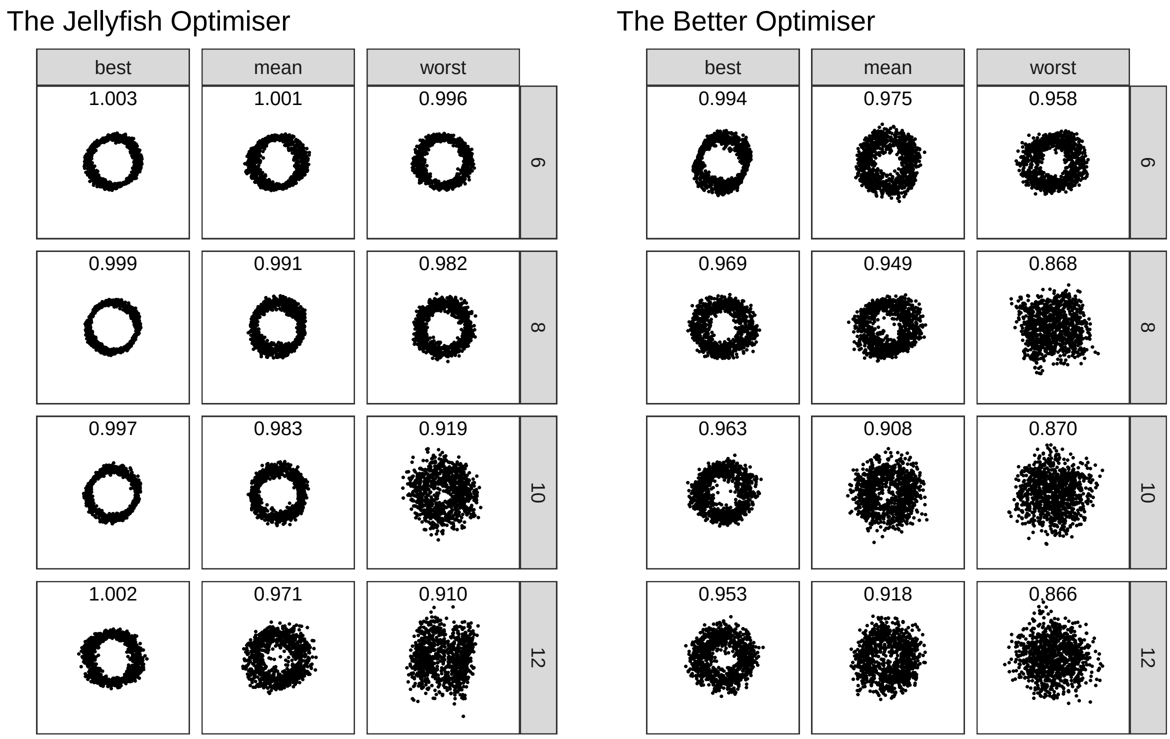

Figure 3: sthis sdfaksdlf

5.2. Connecting optimisation success with its properties and jellyfish parameters

The optimisation properties (smoothness, squintability, and speed) proposed in the Section 3 provide numerical measures to characterise the difficulty of the optimisation for a given projection pursuit problem. This example investigates how these measures, combined with the jellyfish parameters, affect the success of the jellyfish optimisers and how they can guide the decisions made in hyperparameter tuning with the jellyfish optimiser. Simulations are conducted with different jellyfish optimisers to obtain the proportion of successful optimisation across a collection of scenarios, for which smoothness, squintability, and speed are calculated. A generalised linear model is used to link the proportion of success rate with the jellyfish parameters and the optimisation properties.

In addition to the pipe-finding problem, also investigated is the sine-wave finding problem in 6D and 8D spaces with six indices considered: `dcor2d_2`, `loess2d`, `MIC`, `TIC`, `spline`, and `stringy`. Combining with different jellyfish optimiser parameters (`n_jellies` and `max_tries`), a total of 52 cases is produced, comprising of 24 pipe-finding cases and 28 sine-wave finding cases. For each case, the jellyfish optimiser is run fifty (50) times and summary statistics are calculated on the best index value found across all 50 runs and the proportion of runs that find a close (with a difference less than 0.05) best index value (`P_J_hat`).

Smoothness and squintability are computed for each case, following the procedures outlined in Section 3.1 and Section 3.2. To calculate smoothness, three hundred random bases are simulated. Index values and projection distance to the optimal basis are calculated for each basis to fit a Gaussian process model. For squintability, fifty random bases are sampled and interpolated to the optimal basis with a step size of 0.005. The interpolated bases are first binned with a bin width of 0.005 to average the index values before feeding into a 4-parameter scaled logistic function, estimated by non-linear least square, to obtain the squintability measure.

Table 1 presents the parameters estimated for each case from the Gaussian process (variance, range, smooth, and nugget) for smoothness and the scaled logistic function (theta1 to theta4, and squint) for squintability. The column "smooth" and "squint" are used as the smoothness and squintability measure. The low squint value of MIC and TIC index is due to its convex shape of the index value against the projection distance, as shown in Figure 4. Comparing to other concave shapes, the maximum first derivative happens at a smaller projection distance ($\theta 2$), requiring the optimiser to get closer to the optimal to see a significant change in the index value, hence a small squintability measure. [TODO: the stringy squintability doens't make sense]

Table 2 presents the processed data for modelling after augmenting the jellyfish parameters (`n_jellies` and `max_tries`) and the average time spent on each run (`time`) for each case. A generalised linear model is fitted with a binomial family and a logit link function to the data to investigate the relationship between the proportion of jellyfish success and jellyfish parameters and the optimisation properties. Pre-processing of the data include 1) scale the jellyfish parameters (`n_jellies` and `max_tries`) by 10 for interpretation, and 2) create a binary variable `long_time` to flag the cases with average time spent on each run greater than 30 seconds. Table 3 presents the model output with the estimated coefficients. The model suggests a positive relationship between the proportion of jellyfish success and the jellyfish parameters, `n_jellies` and `max_tries`, optimisation property parameter, `smoothness` and `squintability`, and a negative relationship with the average time spent on each run (`long_time`) and the data dimension (`d`). The variable `squintability` and `d` are significant, suggesting their importance over jellyfish parameters to the success of the optimisation.

Table 1: Parameters estimated from the Gaussian process (including variance, range, smoothness, and nugget) and scaled logistic function (theta1 to theta4) for the pipe-finding and sine-wave finding problems. The squint column is calculated as $\frac{\theta 1 \theta 2 \theta 3}{4}$, as described in Section 3.2. The "smooth" and "squint" column represent the smoothness and squintability measures.

shape	index	d	variance	range	smooth	nugget	theta1	theta2	theta3	theta4	squint
pipe	holes	6	0.002	0.408	2.364	0.212	1.001	0.860	3.368	0.823	0.725
pipe	holes	8	0.000	0.259	2.373	0.613	1.001	0.869	3.264	0.811	0.710
pipe	holes	10	0.000	0.144	2.317	1.831	1.000	0.885	3.151	0.806	0.697
pipe	holes	12	0.000	0.254	2.173	0.879	1.000	0.878	3.345	0.806	0.734

sine	MIC	6	0.016	0.100	2.394	0.087	0.894	0.571	1.623	-0.024	0.207
sine	MIC	8	0.016	0.100	2.394	0.087	0.932	0.328	1.314	-0.030	0.100
sine	TIC	6	0.124	0.104	2.471	0.086	0.951	0.536	1.719	-0.025	0.219
sine	TIC	8	0.124	0.104	2.471	0.086	0.945	0.564	1.723	-0.027	0.230
sine	dcor2d_2	6	0.034	0.167	2.663	0.114	0.954	1.039	2.742	-0.019	0.679
sine	loess2d	6	0.083	0.307	2.194	0.292	1.016	1.039	2.648	0.080	0.699
sine	splines2d	6	0.040	0.189	2.606	0.104	1.014	1.051	2.730	-0.009	0.727
sine	stringy	6	0.000	1173.035	1.031	17608.047	1.011	0.011	254.734	0.727	0.711

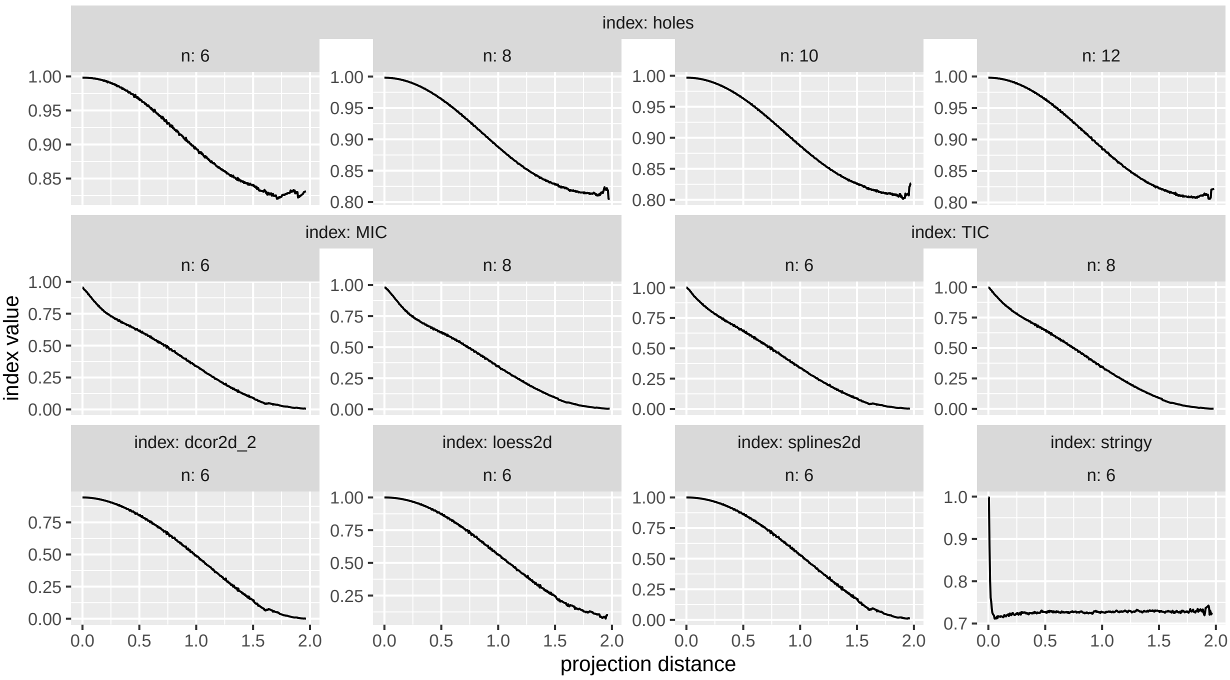

Figure 4: Index values against projection distance for the 12 pipe/sine-wave finding problem after the binning procedure during the estimation of the squintability measure. The index value is averaged at each bin width of 0.005 and the `TIC` index is scaled to 0-1 for comparison. When finding the sine-wave structure using the `MIC` and `TIC` index, a convex curved is observed, as opposed to the pipe-finding problem or the sine-wave finding problem with the `dcor2d_2`, `loess2d`, and `splines2d` index. When finding the sine-wave with the stringy index, the index shows an instantaneous jump to the optimum when close to the best basis.

Table 2: The first few rows of the datasets processed for modelling. The smoothness and squintability variable are uniquely characterised by the index function used and the data dimension, and thus do not vary across `n_jellies` and `max_tries`. The variable `P_J_hat`, and time are calculated at each observation.

index	d	smoothness	squintability	n_jellies	max_tries	P_J_hat	time
MIC	6	2.394	0.207	20	50	0.12	2.479 secs
MIC	6	2.394	0.207	20	100	0.24	8.950 secs
MIC	6	2.394	0.207	50	50	0.52	5.651 secs
MIC	6	2.394	0.207	50	100	0.64	13.223 secs
MIC	6	2.394	0.207	100	50	0.76	19.453 secs
MIC	8	2.394	0.100	20	50	0.08	2.566 secs
MIC	8	2.394	0.100	20	100	0.08	4.960 secs

Table 3: Model estimates of proportion of jellyfish success (`P_J_hat`) on optimisation properties (`smoothness`, `squintability`, and `d`) and jellyfish parameters (`long_time`, `n_jellies`, and `max_tries`) from the generalised linear model with a binomial family and a logit link function. The variable `long_time` is derived to flag the cases with average time spent on each run greater than 30 seconds and the variable `n_jellies` and `max_tries` are scaled by 10 for interpretation.

term	estimate	std.error	statistic	p.value
intercept	-4.672	4.512	-1.036	0.300
smoothness	1.604	1.530	1.048	0.294
squintability	7.062	2.210	3.195	0.001
d	-0.595	0.260	-2.289	0.022
long_time	-0.851	1.348	-0.632	0.528
n_jellies	0.230	0.131	1.754	0.080
max_tries	0.107	0.153	0.700	0.484

6. Conclusion [Di and Sherry]

References

[1] J.-S. Chou, D.-N. Truong, A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean, Applied Mathematics and Computation 389 (2021) 125535. doi:10.1016/j.amc.2020.125535.

[2] K. Rajwar, K. Deep, S. Das, An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges, Artificial Intelligence Review (2023) 1–71. doi:10.1007/s10462-023-10470-y.

[3] J.-S. Chou, A. Molla, Recent advances in use of bio-inspired jellyfish search algorithm for solving optimization problems, Scientific Reports 12 (2022) 19157. doi:10.1038/s41598-022-23121-z.

[4] J. B. Kruskal, Toward a practical method which helps uncover the structure of a set of observations by finding the line transformation which optimizes a new 'index of condensation', in: R. C. Milton, J. A. Nelder (Eds.), Statistical Computation, Academic Press, New York, 1969, pp. 427–440.

[5] J. H. Friedman, J. W. Tukey, A Projection Pursuit Algorithm for Exploratory Data Analysis, IEEE Transactions on Computing C 23 (1974) 881–889.

[6] P. Hall, On polynomial-based projection indices for exploratory projection pursuit, The Annals of Statistics 17 (1989) 589–605. URL: https://doi.org/10.1214/aos/1176347127.

[7] D. Cook, A. Buja, J. Cabrera, Projection pursuit indexes based on orthonormal function expansions, Journal of Computational and Graphical Statistics 2 (1993) 225–250. URL: https://doi.org/10.2307/1390644.

[8] E.-K. Lee, D. Cook, A projection pursuit index for large p small n data, Statistics and Computing 20 (2010) 381–392. URL: https://doi.org/10.1007/s11222-009-9131-1.

[9] N. Loperfido, Skewness-based projection pursuit: A computational approach, Computational Statistics and Data Analysis 120 (2018) 42–57. doi:https://doi.org/10.1016/j.csda.2017.11.001.

[10] N. Loperfido, Kurtosis-based projection pursuit for outlier detection in financial time series, The European Journal of Finance 26 (2020) 142–164. doi:https://doi.org/10.1080/1351847X.2019.1647864.

[11] D. Cook, A. Buja, J. Cabrera, C. Hurley, Grand tour and projection pursuit, Journal of Computational and Graphical Statistics 4 (1995) 155–172. URL: https://doi.org/10.1080/10618600.1995.10474674.

[12] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2023. URL: https://www.R-project.org/.

[13] H. Wickham, D. Cook, H. Hofmann, A. Buja, tourr: An R package for exploring multivariate data with projections, Journal of Statistical Software 40 (2011) 1–18. URL: http://doi.org/10.18637/jss.v040.i02.

[14] H. S. Zhang, D. Cook, U. Laa, N. Langrené, P. Menéndez, Visual diagnostics for constrained optimisation with application to guided tours, The R Journal 13 (2021) 624–641. doi:10.32614/RJ-2021-105.

[15] D. Bertsimas, J. Tsitsiklis, Simulated Annealing, Statistical Science 8 (1993) 10 – 15. URL: https://doi.org/10.1214/ss/1177011077. doi:10.1214/ss/1177011077.

[16] C. Posse, Projection pursuit exploratory data analysis, Computational Statistics and Data Analysis 20 (1995) 669–687. URL: https://www.sciencedirect.com/science/article/pii/0167947395000028. doi:https://doi.org/10.1016/0167-9473(95)00002-8.

[17] E. Lee, D. Cook, S. Klinke, T. Lumley, Projection pursuit for exploratory supervised classification, Journal of Computational and Graphical Statistics 14 (2005) 831–846. URL: https://doi.org/10.1198/106186005X77702.

[18] K. Grimm, Kennzahlenbasierte Grafikauswahl, doctoral thesis, Universität Augsburg, 2016.

[19] P. J. Huber, Projection pursuit, Ann. Statist. 13 (1985) 435–475. URL: https://doi.org/10.1214/aos/1176349519. doi:10.1214/aos/1176349519.

[20] A. B. Ursula Laa, Dianne Cook, G. Valencia, Hole or grain? a section pursuit index for finding hidden structure in multiple dimensions, Journal of Computational and Graphical Statistics 31 (2022) 739–752. URL: https://doi.org/10.1080/10618600.2022.2035230.

[21] L. Wilkinson, A. Anand, R. Grossman, Graph-theoretic scagnostics, in: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., 2005, pp. 157–164. doi:10.1109/INFVIS.2005.1532142.

[22] L. Wilkinson, G. Wills, Scagnostics distributions, Journal of Computational and Graphical Statistics 17 (2008) 473–491. URL: https://doi.org/10.1198/106186008X320465. doi:10.1198/106186008X320465. arXiv:https://doi.org/10.1198/106186008X320465.

[23] U. Laa, D. Cook, Using tours to visually investigate properties of new projection pursuit indexes with application to problems in physics, Computational Statistics 35 (2020) 1171–1205. doi:10.1007/s00180-020-00954-8.

[24] H. Wickham, D. Cook, H. Hofmann, A. Buja, tourr: An R Package for Exploring Multivariate Data with Projections, Journal of Statistical Software 40 (2011) 1–18. doi:10.18637/jss.v040.i02.