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

H. Sherry Zhang[a,*], Dianne Cook[b], Nicolas Langrené[c], Jessica Leung[b]

[a] *University of Texas at Austin, Department of Statistics and Data Sciences, Austin, United States, 78751*

[b] *Monash University, Department of Econometrics and Business Statistics, Melbourne, Australia, 3800*

[c] *BNU-HKBU United International College, Department of Mathematical Sciences, Zhuhai, China, 519087*

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

README:

- British English ("American or British usage is accepted, but not a mixture of these")*
- No "we" - always third person
- Check if the affiliation information is correct
- word use:
  - Use jellyfish search optimiser, or JS optimiser, not jellyfish optimiser (my bad for leading the way)
  - "hyper-parameter" rather than "hyperparameter"

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

*Corresponding author

Email addresses: huize.zhang@austin.utexas.edu (H. Sherry Zhang), dicook@monash.edu (Dianne Cook), nicolaslangrene@uic.edu.cn (Nicolas Langrené), Jessica.Leung@monash.edu (Jessica Leung)

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.

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 optimiser performance. **?@sec-simulation** 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

The data structure proposed in Zhang et al. [14] is implemented for JS optimisers and additional information such as index function used, data dimension, and jellyfish hyper-parameters can be included as additional columns. Below prints an example data collected from finding the sine-wave structure in 6D data (`d = 6`) using a spline index function(`splines2d`):

```
Rows: 10
Columns: 12
$ n_jellies <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50, 50
$ max_tries <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50, 50
$ d         <dbl> 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
$ 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> 0.036664532, 0.005121912, 0.058683104, 0.056527991, 0.015734~
$ 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> 4.362133 mins, 4.362133 mins, 4.362133 mins, 4.362133 mins, ~
$ index     <chr> "spline2d", "spline2d", "spline2d", "spline2d", "spline2d", ~
```

The jellyfish hyper-parameters used are 50 jellyfishes (`n_jellies`) and a maximum of 50 tries (`max_tries`). The full data contains 50 simulations, recorded by the variable `sim`, with different seeds (`seed`). Recorded in columns `basis`, `index_val`, `tries`, `loop` and `time`, are the sampled projection basis, calculated index value, iteration ID, jellyfish ID, and time taken to find the optimum, respectively. The basis column records every visited basis before comparing with the current (last) basis. If a visited basis has a smaller index value than the current (last) one, the JS optimiser keeps the current basis for the next iteration in the search but still records the visited inferior basis. This data can be used to derive numerical summaries and plots:

- **success rate**: Success rate is defined as the proportion of simulations that achieve a final index value within 0.05 of the best index value found among all 50 simulations. In the dataset above, the optimal index value is 0.999. Out of the 50 simulations, 44 achieved an index value within [0.949, 0.999], resulting in a success rate of 0.88. Figure 1 shows the best projection found in each simulation, sorted by the index value. Most of the projections show a clear sine-wave structure, while the last six shows only a straight line with clusters at the end. When local optima exist in the projection pursuit problem, plotting the projections found in each simulation or by each jellyfish can help identify these local optima.
- **Projection distance**: The projection distance between projection bases and the theoretical best basis (or the empirical best basis if the theoretical best is unknown) is calculated as the Frobenius norm, which measures how close a visited basis is to the best basis. Plotting this projection distance against the index value for each basis visited reveals how the index value changes as the basis approaches the optimal basis, helping to identify the squintability of the index function. Figure 2 shows the index value against the projection distance for the given example and its binned version. As the projection distance decreases, the index value initially increases with an increasing slope and then level off. The index value shows greater variation at medium projection distances compared to small and large distances.

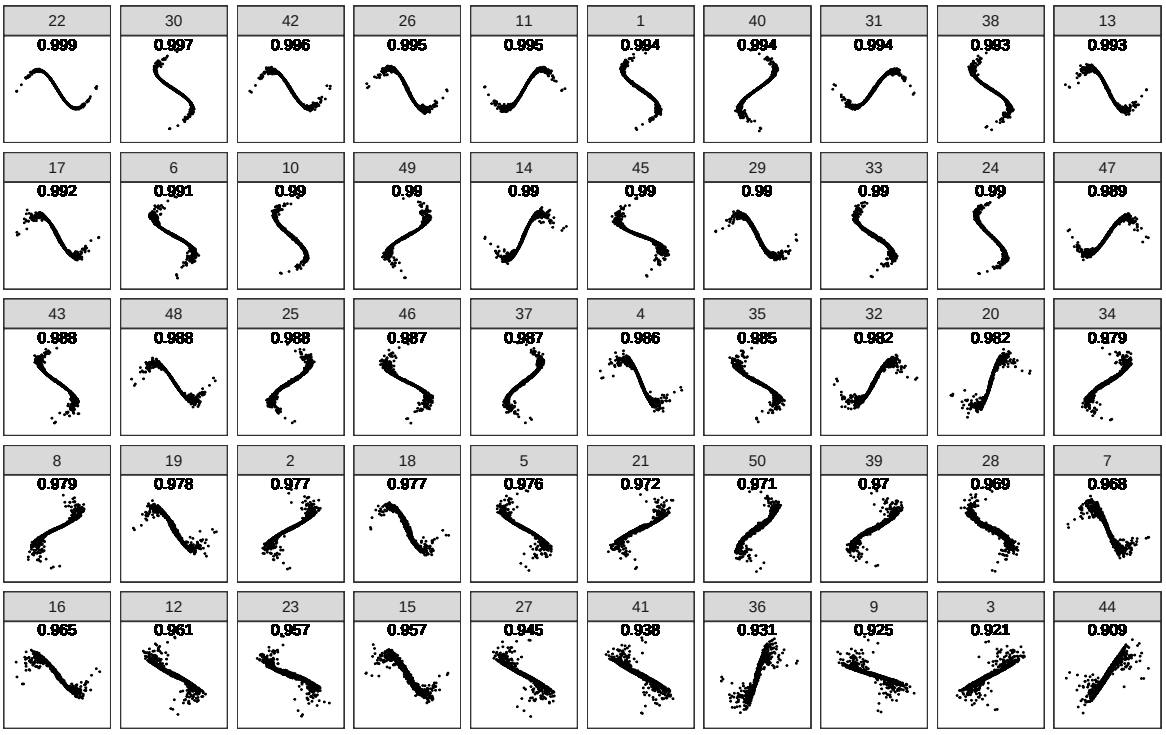

Figure 1: Best projections found by the JS opitmiser in each simulation, sorted by the index value. The projection pursuit problem is to detect the 6D sine-wave structure using the spline index. Most simulations have detected a clear sine-wave pattern, while the last six show a straight line with clustering at the ends.. In this simulation, the success rate is 0.88 (44/50).

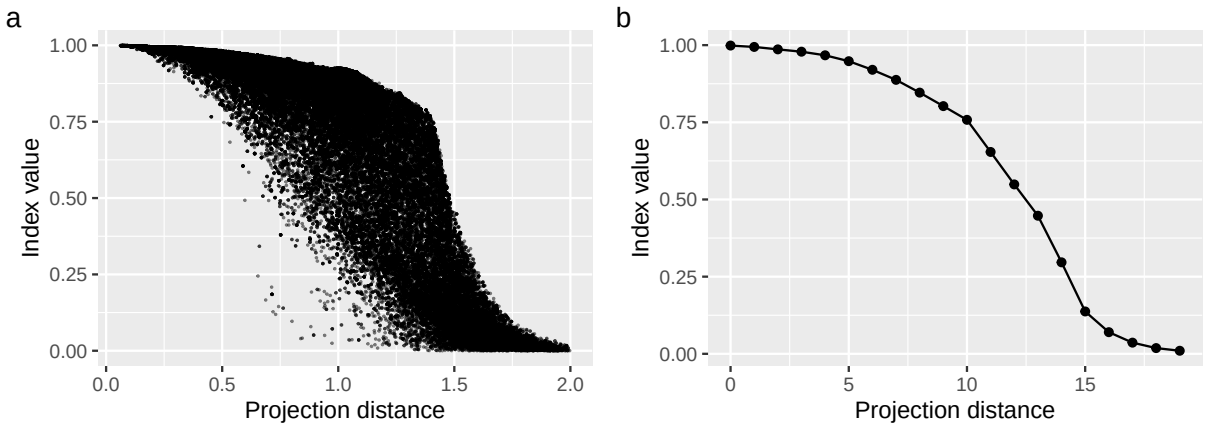

Figure 2: Projection distance plotted against the index value for detecting the 6D sine-wave structure using the spline index function. a) The index values start around 0 with low variance at large projection distances. As the projection distance decreases, the index values increase with a larger variance. Towards the minimum projection distance, the index values increase and cluster around 1, indicating the optimiser is approaching to the optimum. b) Projection distances are binned with a bin width of 0.1, and index values are averaged over each bin. The relationship between the two variables forms a sigmoid curve.

5. Application [Di and Sherry]

The JS optimiser provides a new swarm-based search strategy to optimise projection pursuit problems. Two examples are provided in this section to demonstrate its performance. The first example compares its performance with the search better optimiser and explores how it behaves with different combinations of hyper-parameter. The second example studies the effect of optimisation properties defined in Section 3, along with jellyfish hyper-parameters, on the outcome of the optimisation.

5.1. Performance of JS optimiser in pipe-finding problems

The performance of JS optimiser is investigated in two folds: 1) compare it with a commonly used optimiser: the search better optimiser [14, 23], and 2) examine its success rate under different hyper-parameters (number of jellyfishes and the maximum number of tries). The projection pursuit problem used is finding the pipe shape using the holes index, investigated by Laa and Cook [23].

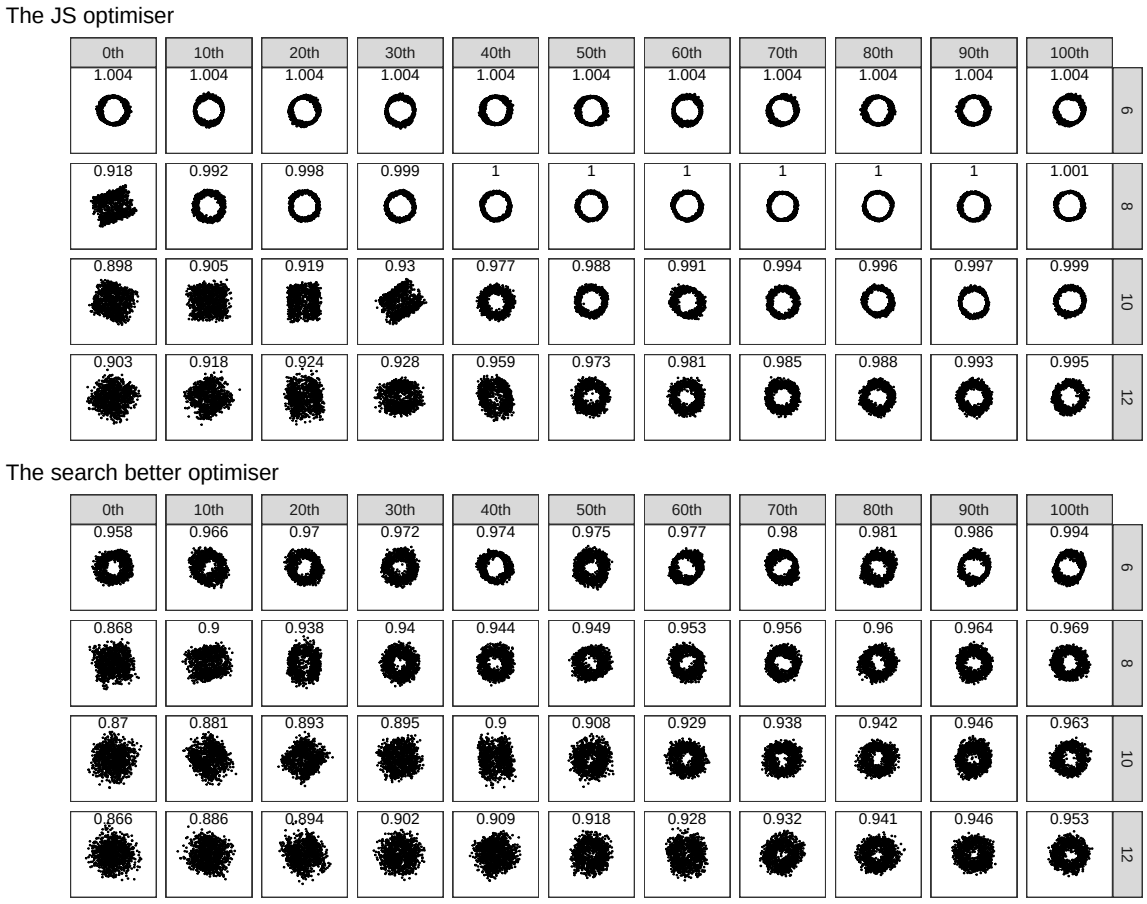

Figure 3: Projections found by the jellyfish and search better optimisers at each 10th quantile across 50 simulations. The projection pursuit problem is to find the pipe shape using the holes index in the 6, 8, 10, and 12-dimensional spaces. The JS optimiser uses 100 jellyfishes and a maximum number of tries of 100. The search better optimiser uses a maximum of 1000 tries in each step of random sampling step before the algorithm terminates. In the 6-D data space, the JS optimiser always finds a clear pipe shape while the search better optimiser also finds the pipe shape but with a wide rim. At higher data dimensions, the JS optimiser finds a higher index value and a clearer pipe shape across all the quantiles than the search better optimiser.

Fifty (50) simulations are conducted with both JS and the search better optimiser, in four dimensions ($d = 6, 8, 10, 12$). The JS optimiser uses 100 jellyfishes and 100 maximum number of tries, while the search

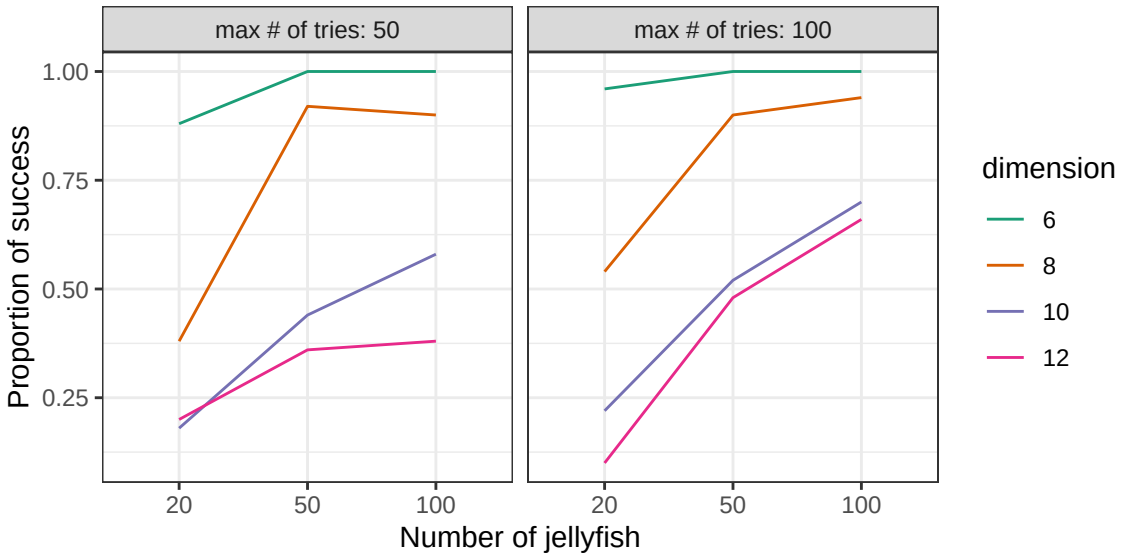

Figure 4: Proportion of simulations reaches near-optimal index values in the pipe-finding problem using the holes index. The proportion is calculated based on the number of simulations, out of 50, that achieve an index value within 0.05 of the best-performing simulation. As the dimensionality increases, the proportion of simulations reaching the optimal index value increases.

better optimiser allows a maximum of 1000 samples at each iteration before the algorithm terminates. Figure 3 presents the final projections found by the two optimisers, broken down by 10th quantile, faceted by the data dimension. In the 6-dimensional data scenario, the JS optimiser consistently identifies a clear pipe shape. The search better optimiser, although also find the pipe shape but with a wide rim, suggesting a further polish search may be required. With increasing dimensions, the JS optimiser may not always identify the pipe shape due to random sampling, it still presents a pipe shape in over 50% of cases. When compared to the search better optimiser, the JS optimiser reaches higher index values and clearer pipe shapes across all quantiles.

In the second experiment, fifty (50) simulations are conducted at each hyper-parameter combination to analyse its effects on the success rate of the JS optimiser. The hyper-parameters tested include: 1) 20, 50, and 100 jellyfishes, and 2) 50 and 100 maximum number of tries. In each simulation, success rate is calculated as defined in Section 4. Figure 4 presents the proportion of success for each hyper-parameter combinations. As the number of jellyfish and maximum tries increase, the success rate also increases. For simpler problems (6 dimensions), small parameter values (20 jellyfishes and a maximum number of tries of 50) can already result in high success rate, while for higher-dimensional problems (8, 10, and 12 dimensions), a combination of 100 jellyfishes and 100 maximum number of tries is necessary for at least half of the jellyfishes to find the optimal projection.

5.2. Factors affecting JS optimiser success rate: optimisation properties and jellyfish hyper-parameters

The optimisation properties, including smoothness, squintability, and speed, offer numerical metrics to characterise the complexity of a projection pursuit optimisation problem. This example investigates how these metrics, alongside jellyfish hyper-parameters (the number of jellyfish and the maximum number of tries), affect the success of the JS optimiser. Simulations are conducted to obtain the success rate across various projection pursuit problems, characterised by shape-to-find, data dimension, and index function used, as well as different hyper-parameter combinations. Smoothness and squintability are calculated for each projection pursuit problem as outlined in Section 3. A generalised linear model is used to construct the relationship between the success rate, jellyfish hyper-parameters and optimisation properties.

In addition to the pipe-finding problem, a new shape, sine wave, is investigated in 6D and 8D spaces with six indices considered: `dcor2d_2`, `loess2d`, `MIC`, `TIC`, `spline`, and `stringy`. Combining with two jellyfish hyper-parameters, a total of 52 cases is produced, comprising of 24 pipe-finding cases and 28 sine-wave finding cases. For each case, the JS optimiser is run fifty (50) times and the success rate is calculated as in Section 5.1.

Note: should we print out the 52 cases investigated somewhere, say appendix, since it is not full grid simulation (otherwise for the sine-wave problem there will be 6 indexes x 6 configurations = 36 cases for 6-D data and 2 indexes [MIC & TIC] x 6 configurations = 12 cases for 8-D data)? We only do a selection of 28 of them to save time/ computational resources.

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 random basis before fitting them into a Gaussian process model. For squintability, fifty random bases are sampled and interpolated to the optimal basis with a step size of 0.005. For these interpolated bases, index values and projection distances to the optimal basis are calculated and averaged with a bin width of 0.005. The squintability measure is then calculated from fitting a 4-parameter scaled logistic function of index value against projection distance, estimated by non-linear least square.

Note: Shall I repeat the scaled logistic function with our parameterisation here? It can be handy to interpret theta1 - theta4 but they should've already been explained in the squintability section.

Table 1 presents the parameters estimated from the Gaussian process (variance, range, smooth, and nugget) and the scaled logistic function (θ_1 to θ_4) for each case. The column "smooth" is used as the smoothness measure and the column "squint" is calculated as $\frac{\theta_1\theta_2\theta_3}{4}$ as squintability. 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 5. Comparing to other concave shapes, the maximum first derivative happens at a smaller projection distance (θ_2), requiring the optimiser to get closer to the optimal to see a significant change in the index value, hence a small squintability measure.

Note: the stringy squintability doesn't make sense

Table 2 combines all the variables calculated above (the JS optimiser success rate, jellyfish hyper-parameters, and optimisation properties) into one table. Three pre-processing steps are applied to the data: 1) scaling the jellyfish hyper-parameters by a factor of 10 for interpretation, 2) creating a new binary variable `long_time` to indicate cases with an average run time over 30 seconds, and 3) encoding the success rate for the stringy case as 0, since none of the 50 simulations identified the sine-wave shape. A generalised linear model with a binomial family and a logit link function is used to fit the data and Table 3 presents the model outputs. The model suggests that the success rate of the JS optimiser is positively associated with the two jellyfish hyper-parameters, as well as with smoothness and squintability. However, being flagged with long runtime and an increase of data dimension will reduce the success rate. The variable `squintability` and `dimension` are significant, suggesting their importance relative to jellyfish hyper-parameters in the optimisation success.

Table 1: Parameters estimated from the Gaussian process (including variance, range, smoothness, and nugget) and scaled logistic function (θ_1 to θ_4) for the pipe-finding and sine-wave finding problems. The "smooth" column and "squint" column, calculated as $\frac{\theta_1\theta_2\theta_3}{4}$, represent the smoothness and squintability measures.

	shape	index	d	variance	range	smooth	nugget	theta1	theta2	theta3	theta4	squint
1	pipe	holes	6	0.002	0.408	2.364	0.212	0.991	0.860	3.368	0.172	0.718
2	pipe	holes	8	0.000	0.259	2.373	0.613	1.011	0.869	3.264	0.085	0.716
3	pipe	holes	10	0.000	0.144	2.317	1.831	1.011	0.885	3.151	0.153	0.705
4	pipe	holes	12	0.000	0.254	2.173	0.879	0.999	0.878	3.345	0.196	0.733
5	sine	MIC	6	0.016	0.100	2.394	0.087	0.894	0.571	1.623	-0.024	0.207
6	sine	MIC	8	0.016	0.100	2.394	0.087	0.932	0.328	1.314	-0.030	0.100

7	sine	TIC	6	0.124	0.104	2.471	0.086	0.951	0.536	1.719	-0.025	0.219
8	sine	TIC	8	0.124	0.104	2.471	0.086	0.945	0.564	1.723	-0.027	0.230
9	sine	dcor2d_2	6	0.034	0.167	2.663	0.114	0.954	1.039	2.742	-0.019	0.679
10	sine	loess2d	6	0.083	0.307	2.194	0.292	1.016	1.039	2.648	0.080	0.699
11	sine	splines2d	6	0.040	0.189	2.606	0.104	1.014	1.051	2.730	-0.009	0.727
12	sine	stringy	6	0.000	1173.035	1.031	17608.047	1.040	0.011	254.733	0.170	0.731

Table 2: The first 7 rows of the datasets processed for modelling.

Index	D	Smoothness	Squintability	Num. of jellyfish	Max. Num. of tries	Prop. of success	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 on optimisation properties and jellyfish hyper-parameters from the generalised linear model with a binomial family and a logit link function. The variable smoothness, squintability, number of jellyfish and maximum number of tries are positively associated with the JS optimiser success rate while data dimension and being flagged as long runtime are negatively associated with the success rate. The variable squintability and dimension are significant, suggesting their importance relative to jellyfish hyper-parameters in the optimisation success.

term	estimate	std.error	statistic	p.value
Intercept	-4.849	4.496	-1.078	0.281
Smoothness	1.672	1.527	1.094	0.274
Squintability	6.967	2.173	3.207	0.001
dimension (d)	-0.588	0.257	-2.292	0.022
long time	-0.816	1.333	-0.612	0.540
number of jellyfish	0.229	0.131	1.752	0.080
maximum number of tries	0.108	0.153	0.703	0.482

6. Conclusion [Di and Sherry]

7. Additional plot:

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.

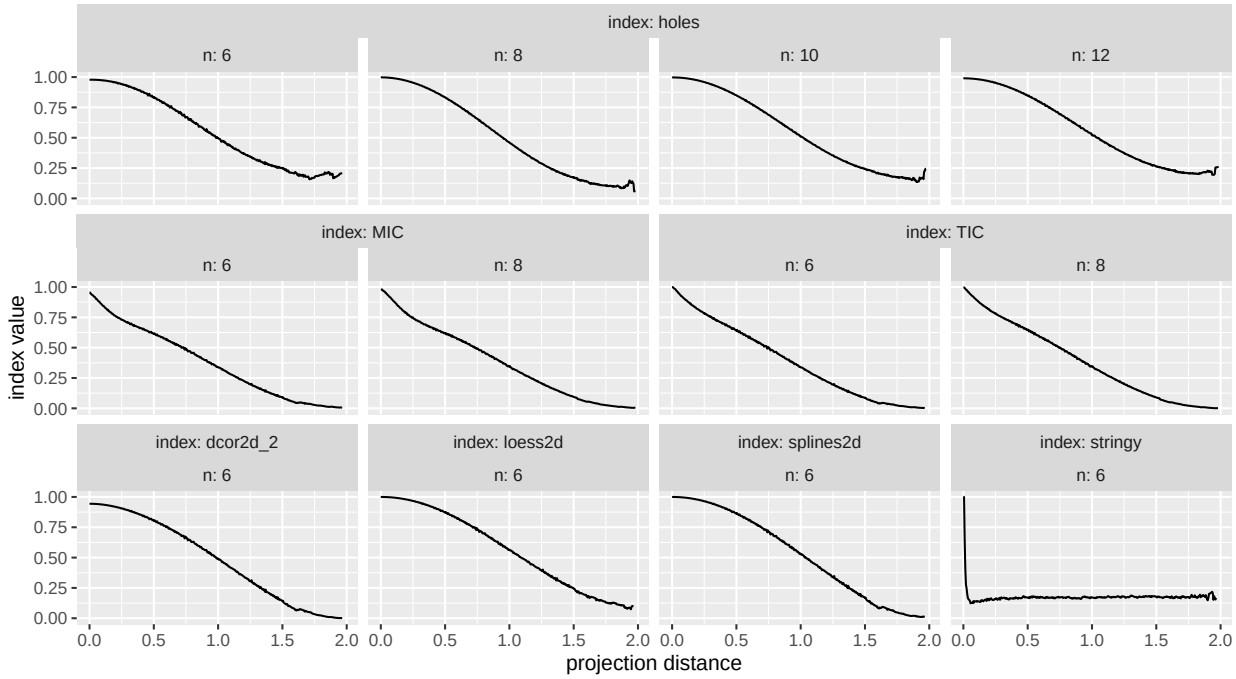

Figure 5: Index values against projection distance for the 12 pipe/sine-wave finding problem after the binning procedure during the estimation of the squintability measure. The index value is averaged at each bin width of 0.005 and the `TIC` index is scaled to 0-1 for comparison. When finding the sine-wave 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.

[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.