FEDERAL RESERVE BANK OF MINNEAPOLIS | Research Division

WORKING PAPER
No. 801

Benchmarking Global Optimizers

December 2023

Antoine Arnoud
International Monetary Fund

Fatih Guvenen
University of Minnesota, Federal Reserve Bank of Minneapolis, and NBER

Tatjana Kleineberg
World Bank

DOI: https://doi.org/10.21034/wp.801
Keywords: Global optimization; Multistart algorithms; NLopt; Calibration; Estimation; Parallelized optimizer
JEL classification: C61, C63, D58

Benchmarking Global Optimizers*

Antoine Arnoud† Fatih Guvenen‡ Tatjana Kleineberg§

October 19, 2023

Abstract

We benchmark six global optimization algorithms by comparing their performance on challenging multidimensional test functions as well as on a method of simulated moments estimation of a panel data model of earnings dynamics. Five of the algorithms are from the popular NLopt open-source library: (i) Controlled Random Search with local mutation (CRS), (ii) Improved Stochastic Ranking Evolution Strategy (ISRES), (iii) Multi-Level Single-Linkage (MLSL), (iv) Stochastic Global Optimization (StoGo), and (v) Evolutionary Strategy with Cauchy distribution (ESCH). The sixth algorithm is TikTak, which is a multistart global optimization algorithm used in some recent economic applications. For completeness, we add three popular local algorithms to the comparison—the Nelder-Mead downhill simplex algorithm, the Derivative-Free Nonlinear Least Squares (DFNLS) algorithm, and a popular variant of the Davidon-Fletcher-Powell (DFPMIN) algorithm. To give a detailed comparison of algorithms, we use benchmarking tools recently developed in the optimization literature. We find that the success rate of many optimizers varies dramatically with the characteristics of each problem and the computational budget that is available. Overall, TikTak is the strongest performer both on the test functions and the economic application. The next-best performing optimizers are StoGo for the test functions and MLSL and ISRES for the economic application.

JEL Codes: C61, C63, D58.
Keywords: Global optimization, Multistart algorithms, NLopt, Calibration, Estimation, Parallelized Optimizer.

*Special thanks to Anthony Smith, Serdar Ozkan, and Fatih Karahan for their collaboration with Fatih Guvenen on projects that employed earlier versions of TikTak and for making significant contributions to the algorithm; to Michelle Rendall, Rocio Madera, David Domeij and Chris Busch for collaborations with Guvenen on subsequent projects that used TikTak; and to Steven Johnson and Egor Malkov for helpful comments. Leo Stanek and Arun Kandanchatha provided outstanding research assistance, and Arun rewrote parts of the TikTak code as a generic, stand-alone, and user-friendly program. This project started while Guvenen was a visiting professor in the economics department at Yale University, whose support and hospitality are gratefully acknowledged. The views expressed herein are those of the authors and do not represent those of the World Bank or the International Monetary Fund.

†The International Monetary Fund; aarnoud@imf.org; antoinearnoud.com
‡University of Minnesota, FRB of Minneapolis, and NBER; guvenen@umn.edu; fatihguvenen.com
§The World Bank; tkleineberg@worldbank.org; sites.google.com/view/tkleineberg

1 Introduction

We benchmark the performance of six global and three local optimizers by applying them to several difficult multidimensional optimization problems. We first apply the algorithms to four test functions—Levi, Griewank, Rastrigin, and Rosenbrock—that are commonly used in the applied mathematics literature for benchmarking optimizers. We choose these four functions out of a larger suite of commonly used test functions because they are known to be particularly challenging to optimize.[1]

The characteristics of the test functions can differ substantially from the calibration or estimation problems commonly encountered in economics. The test functions are, for example, continuous and differentiable (despite having many local minima), which is not necessarily the case in economic applications in which objective functions are often based on moments that are computed with data that is simulated from the numerical solution of a complex model. Because of truncation and other approximation errors in numerical solutions, as well as the economic features of some models (e.g., discrete choice or binding constraints), the resulting objective function often displays kinks, jaggedness, deep ridges, flat valleys, and even jumps, posing challenges to optimizers.

We therefore assess the performance of the optimizers in an economic application—a method of simulated moments (MSM) estimation of a panel data income dynamics model taken from Busch et al. (2015), which has 297 moments and estimates 7 parameters. This estimation problem is not among the most complex ones we could have chosen, and this choice is intentional. Our goal is to show that even a relatively benign optimization problem commonly encountered in economics can be challenging for many global optimizers.

Five of the six global optimizers are from the NLopt library, which is an open-source library for nonlinear optimization that contains many state-of-the-art optimization routines.[2] These five optimizers are (i) Controlled Random Search with local mutation (CRS), (ii) Improved Stochastic Ranking Evolution Strategy (ISRES), (iii) Multi-Level Single-Linkage (MLSL), (iv) Stochastic Global Optimization (StoGo), and (v) Evolutionary Strategy with Cauchy distribution (ESCH).[3]

[1] Commonly used test function suites include *CUTEr* (Constrained and Unconstrained Testing Environment, revisited) or *COPS* (Constrained Optimization Problem Set).

[2] See Johnson (2018) for an overview of the NLopt library. Further documentation and codes are available at http://ab-initio.mit.edu/wiki/index.php/NLopt.

[3] CRS belongs to the group of random search algorithms, ISRES and ESCH are evolution strategy algorithms, MLSL and TikTak are multistart algorithms, and StoGo uses branch-and-bound techniques to search for global optima. StoGo uses the derivative of the function in the optimization routine. All other algorithms are derivative-free.

The sixth global optimizer, named TikTak, was developed by one of us and refined with coauthors through applications to various estimation/calibration problems in economics.[4] TikTak has been developed specifically for economic applications (medium- to large-scale structural estimation and calibration problems) and has been improved over the years as it was applied to a different problem in each paper. TikTak belongs to the class of multistart algorithms, which conducts local searches from carefully selected points in the parameter space. The algorithm starts with a uniform exploration of the (parameter) space and uses the information it accumulates to increasingly focus the search on the most promising region. We consider two variants of TikTak that differ only in the local optimization routine. "TikTak-nm" uses the Nelder-Mead downhill simplex algorithm and "TikTak-d" uses the Derivative-Free Nonlinear Least Squares (DFNLS) algorithm of Zhang et al. (2010). An important advantage of TikTak is that it is fully parallelizable without requiring special software or coding by the user. In this paper, we only test its single core performance to be consistent with the remaining optimizers.[5]

Even in large-scale estimation and calibration problems, it is fairly common for researchers to use a local optimizer alone (not as part of a global algorithm). One would then restart the local optimizer several times and pick the best objective. While this approach resembles multistart global algorithms, in practice, the number of restarts can be fairly small, and there is no systematic procedure for selecting restart points; in fact, it is not uncommon to do a single restart from the last local optimum. Given the popularity of these approaches that rely on local optimizers alone, we include three widely used local algorithms in the benchmarking analysis: the Nelder-Mead and DFNLS algorithms mentioned above and the DFPMIN optimizer taken from Press et al. (1996), which is based on a quasi-Newton algorithm.[6]

One notion that we have used so far without defining it is the "performance" of an optimizer. There are at least four practical considerations. The first consideration,

[4] See Guvenen (2011) for a description of an early version of TikTak. The algorithm was used to estimate a structural model of consumption-savings choice with Bayesian learning via indirect inference in Guvenen and Smith (2014) and to estimate an equilibrium model of marriage/divorce, educational attainment, and labor supply in Guvenen and Rendall (2015), with the method of simulated moments. TikTak was further used to estimate panel data econometric models of earnings dynamics in Guvenen et al. (2014), Guvenen et al. (2015), and Busch et al. (2018) (with up to 1,200 moments and 35 parameters in Guvenen et al. (2015)). In each case, the objective function displayed several challenging features such as kinks, jumps, ridges, and so on.

[5] For information about the parallel implementation of TikTak, including information on how its performance scales with the number of cores, visit `https://www.fatihguvenen.com/tiktak`

[6] More precisely, DFPMIN implements the Broyden-Fletcher-Goldfarb-Shanno variant of the Davidon-Fletcher-Powell minimization algorithm. For details, see Press et al. (1992, Chapter 10.7).

and arguably the most important, is an optimizer's *reliability*—or the likelihood that it will find the global optimum of the problem that a researcher faces. A proxy for reliability commonly used in the benchmarking literature is the fraction of test problems for which the optimizer successfully finds the global optimum (its "success rate"). A second consideration is *speed*: in practice, researchers have a finite computational budget that they can afford for a given problem, so what we really want to know is the success rate of an optimizer for different computational budgets (measured, for example, in time or number of function evaluations, or FEs). To capture this trade-off we use "data profiles," introduced by Moré and Wild (2009), which plot the success rate of an optimizer as a function of the computational budget. A third consideration is how an optimizer's *speed compares* to other optimizers for given test problems. In particular, we would like to know the fraction of problems for which a given optimizer is the fastest among all available optimizers, as well as the fraction of problems for which it is at most two times (or three, four, and so on) slower than the fastest optimizer of each problem. We capture this information in "performance profiles" introduced by Dolan and Moré (2002) and used, for example, in Ali et al. (2005) and Zhang et al. (2010).[7]

To consider a minimization "successful," we focus on two different metrics: the distance between the function values of the returned and true minima or the distance between the parameter values of the returned and true minima. If the respective distance is smaller than a given threshold, the minimization is considered a success for that metric/threshold combination. Clearly, the choice of a threshold involves a judgement call, so the *fourth* consideration is how well or poorly an optimizer does when it fails to attain the specified threshold. As we shall see, some optimizers will technically fail (sometimes on most problems) but end up coming very close to the threshold, whereas others stop far away. To analyze such differences, we construct what we refer to as "deviation profiles." These are analogous to the data profiles, but instead of the success rate, they report the average of the distance measure over each algorithm's unsuccessful implementations at different computational budgets.

Using these three benchmarking tools, we find that overall, TikTak-d has the strongest performance on the test functions and the economic application—in terms of reliability and speed. The second-best optimizer is TikTak-nm, which performs well on the test

[7] A data profile plots optimizers' success rates—the fraction of problems the optimizer solves successfully—as a function of the computational budget, capturing the trade-off between reliability and speed for each optimizer. A performance profile compares optimizers by plotting the fraction of successfully-solved problems for which a given optimizer is at most α times slower than the fastest optimizer for that problem.

functions and on the income process for most but not all success criteria. TikTak-nm is less efficient than TikTak-d, as it requires a larger computational budget. The relative performance of the NLopt algorithms varies across test functions and the economic application. StoGo performs best on the test functions.[8] MLSL and ISRES perform better on the economic application but are relatively less successful in minimizing the test functions with ISRES performing poorly on all four test functions and MLSL performing poorly on two of them. CRS struggles with the same two test functions as MLSL and is less reliable on the economic application. ESCH performs less well on the economic application and the test functions. All local algorithms (which we use only on the test functions) have low success rates under all success criteria, and their performance does not improve as computational budgets increase. Among the local algorithms, Nelder-Mead performs best, followed by DFNLS, and then by DFPMIN.

Parallel Implementation of TikTak All the benchmarking results presented in this paper are based on running each optimizer on a single CPU core to provide a level playing field. However, an appealing feature of TikTak is that it has a parallel implementation, which can be much faster than the single-core version benchmarked here and has some other desirable features.[9] Although we do not delve into the details of this parallel implementation of TikTak, we briefly mention four of its features, which may be of interest to researchers. First, its performance *scales up nearly linearly* with the number of cores available, so the completion time halves as the number of cores is doubled. This can be seen in Figure (A.1) in Appendix (A), which plots the completion time against the number of cores used. For large size problems, the parallel version of TikTak can be up to one or two *orders of magnitude* faster. Second, using the parallel version does not require any special software or knowledge of parallel programming (such as MPI, OpenMP, CUDA, and so on). Third, it is an "asynchronously parallel" implementation, which means that the number of CPU cores can be increased or reduced during run-time without any problems. For example, the optimizer can be launched on 4 CPU cores and several hours or days later, another set of CPU cores (which may not have been available when the optimizer was first launched) can be added to further speed up the computation. Similarly, some CPU cores can be "retired" during run-time without problems (except of course the natural slow down resulting from using fewer cores). The code is written to

[8] We exclude StoGo from the benchmarking on the economic application because it is the only optimizer that uses gradient information. The analytical gradient is not available in most economic applications, and numerically computing the gradient can be difficult and computationally demanding.

[9] See https://www.fatihguvenen.com/tiktak for more information and a link to the source code for this parallel implementation.

automatically adjust to the number of available cores in both cases. Fourth, and finally, the optimizer can be run on CPUs that reside in different machines possibly located in different physical locations, which could be running different operating systems and/or different compilers of the same language.[10] We leave a fuller exploration of TikTak's parallel performance for future research.

Related Literature. An active literature in applied mathematics benchmarks global optimization algorithms using various collections of well-known test problems. For example, Mullen (2014) compares the performance of different algorithms—including CRS, MLSL and StoGo, as well as algorithms based on annealing and particle swarm optimization methods—in optimizing 50 objective functions. Ali et al. (2005) test five different stochastic optimization algorithms on the same suite of 50 test problems. The considered algorithms are based either on simulated annealing methods (Hit-and-Run and Hide-and-Seek) or on population sets (Controlled Random Search, Real Coded Genetic Algorithm and Differential Evolution). Ali et al. (2005) show that the performance of optimizers depends crucially on the computational budget (i.e., function evaluations, or FEs). CRS performs better with fewer FEs, whereas other algorithms (such as genetic algorithms) perform better when more FEs are used. Kaelo and Ali (2006) compare different versions of the CRS algorithm. They conclude that CRS with local mutation (CRS-LM) performs best in terms of efficiency (number of FEs) and reliability (success rates) among all versions of CRS.

Our paper makes the following contributions. First, we benchmark optimizers not only for a collection of test problems but also for an economic application, which can help applied economists select the best algorithm to estimate structural economic models. Second, we benchmark a new global optimization algorithm (TikTak), analyze its performance, and find that it outperforms most of the other optimizers on test functions and the economic application. Third, we use "deviation profiles" (in addition to commonly used data and performance profiles) throughout our benchmarking exercise to document how far away failed implementations are from the true global optimum.

Section 2 describes the TikTak algorithm (it omits the others, since they are already well known). Section 3 describes the tools that we use to compare the performance of optimizers. Section 4 provides the benchmarking results for the test functions. Section 5 discusses the results for the economic application. Section 6 concludes.

[10] In the previous applications cited in footnote 4, the algorithm was run in parallel on a few dozen to several hundred CPUs distributed across servers, clusters, and personal computers located in Zurich, Minneapolis, New Haven, New York, and Washington DC.

2 Algorithms

The five global algorithms from the NLopt library and the three local algorithms are widely used in different scientific applications and hence are well known. For brevity, we omit their descriptions in this section but discuss them in more detail in Appendix B.[11] TikTak is a new algorithm that is not well known, so we describe it in some detail here as well as in Appendix B.

2.1 The TikTak Algorithm

TikTak belongs to the class of multistart algorithms. A multistart algorithm first picks a point in the parameter space at which it implements a local optimization until a local optimum is found. The algorithm then picks the next starting point, implements another local optimization, and finds a new local optimum. This procedure is repeated many times. At the end, the algorithm returns the point with the lowest value function among all local optima as the global optimum. The main distinguishing features among multistart algorithms are how they choose the next starting point and how they use the information that is provided by the history of local searches. These algorithms typically have two stages: (i) a *global* stage, which selects the starting points for new local searches, and (ii) a *local* stage, which implements local searches, by choosing a local search algorithm and local stopping criteria.

TikTak aims to balance the need for reliability (high success rates) and efficiency (low computational budgets). To achieve reliability, it is important to search broadly over the entire parameter space. To achieve reliability and efficiency, it is important to identify promising regions and to search more intensively in these regions. To search broadly and uniformly early on, TikTak evaluates the objective function at points in the parameter space that are drawn from quasi-random variables, which are deterministic sequences that are designed to cover the parameter space as uniformly as possible.[12] In particular, TikTak uses the Sobol' sequence (Sobol' (1967)), which has several desirable properties and is known to perform particularly well in high dimensions.[13]

[11] For TikTak, we use either Nelder-Mead or DFNLS in the local stage (i.e., local searches). For MLSL, we use either Nelder-Mead or BOBYQA in the local stage. Finally, we implement a "polishing phase" at the end of all global optimizations, which consists of a final local search with a stringent convergence criterion. For these polishing searches, we use DFNLS or/and BOBYQA (see Section 3.3).

[12] It is well known that random numbers drawn from a uniform distribution are not effective ways to sample a space uniformly, especially in higher dimensions. See Zhigljavsky and Žilinskas (2008) for a thorough discussion.

[13] For further details, see, e.g., Liberti and Kucherenko (2005) and Kucherenko and Sytsko (2005).

The global stage of TikTak comprises two phases. The first phase is pre-testing, which consists of drawing and evaluating N Sobol' points and selecting among these the $N^*(\ll N)$ "seed" points that have the lowest (best) function values. (In practice, N will be a large number that scales up with the dimensionality of the problem, whereas N^* is much smaller—for example, 1% to 10% of N. These seed points are then sorted in ascending order, $(s_1, \ldots, s_{N^*})$, with $f(s_1) \leq \cdots \leq f(s_{N^*})$. The remaining Sobol' points are discarded, as the space in their immediate vicinity seems less promising.

In the second phase, the algorithm sequentially implements local searches from N^* starting points, denoted $(\widetilde{s}_1, ..., \widetilde{s}_{N^*})$. Let z_j^* denote the minimum found by the local search that started from $\widetilde{s}_j$. The starting point for the next local search is chosen as a convex combination of the next Sobol' seed point, s_{j+1}, and the *best* minimum found in the previous j local searches up to that time, denoted $Z_j^* = \min(z_1^*, z_2^*, ..., z_j^*)$:

$$\widetilde{s}_{j+1} = (1 - \theta_j)s_{j+1} + \theta_j Z_j^*,$$

where $\theta_j \in (0, 1]$ is the mixing weight. Early on in the second phase, θ_j is chosen to be very small, possibly zero, to allow time for the algorithm to conduct a broad search of the parameter space. As the algorithm progresses and the information accumulated from past local searches grows, θ_j is gradually increased to concentrate local searches around the space that includes the best local minima, so that the most promising parts of the parameter space are explored more and more thoroughly. A useful heuristic is to stop the algorithm when the absolute difference between the last two different values of Z_j^* are sufficiently close to each other—in other words, when a new best-local-minimum is not too different from the previous one. This is the basic idea of the TikTak algorithm; additional details are in Appendix B.

In the benchmarking analysis, we use four different variants of TikTak, which differ in the implementation of the local searches. The variants use either the local Nelder-Mead or the local DFNLS algorithm. Another important decision is the stopping tolerance of each local search. A high tolerance will lead to shorter, and hence less costly, local searches but may stop too soon without fully exploring the region it started in. A lower tolerance implies the opposite trade-off (exhaustive but costly search). To explore these trade-offs, we consider two tolerance levels, 10^{-3} and 10^{-8}, for each local optimization

algorithm.[14,15] We refer to these four versions as TikTak-nm3 and TikTak-nm8 when Nelder-Mead is used in the local stage, and as TikTak-d3 and TikTak-d8 when DFNLS is used.

3 Measurement Preliminaries

In this section, we discuss in more detail how we define and measure the performance of an optimizer. We already mentioned two notions of performance: reliability and speed. The *reliability* of an algorithm measures the success rate—that is, the percentage of problems that the algorithm solves successfully. The *efficiency* (or speed) of an algorithm measures the computational budget (i.e., the number of function evaluations or FEs) that the algorithm requires to reach certain success rates.

We now define what it means to solve a problem successfully. The goal of an optimization is to find the true global minimum of the objective function. We know the true global minima for the standard test functions, but not for the objective function of the economic application. For the economic application, we therefore consider the "true" minimum to be the point with the lowest function value that is found by any of the optimizers and with any of the computational budgets that we consider. Let $f_s(\mathbf{x})$ denote the function we wish to minimize in a given problem $p \in \mathcal{P}$, $\mathbf{x}_p^*$ denote the (unique) parameter vector at the minimum, and $y_p^* = f_p(\mathbf{x}_p^*)$ be the minimized function value. Finally, let $\hat{\mathbf{x}}_{p,s}^*$ be the global minimum returned by optimizer (or solver) s and $\hat{y}_{p,s}^* = f(\hat{\mathbf{x}}_{p,s}^*)$ be the corresponding function value. We define two different success criteria, one based on discrepancy in function values and the other based on discrepancy in the parameter vector. Specifically, we say the optimizer $s \in S$ solved the problem p successfully according to the F-val criterion if

$$|y_p^* - \hat{y}_{p,s}^*| < \tau,$$

where τ is the desired tolerance we choose. Similarly, the optimizer solves the problem successfully according to the X-val criterion if the maximum discrepancy across all

[14] One could conjecture that the ideal approach would start with a high tolerance early in the search process, when most of the local searches are likely to take place far away from the global optimum, and then gradually tighten the tolerance as the algorithm progresses and narrows down the search area. We have not pursued this approach here, but it is implemented in the parallel version of the TikTak code that we refer to in footnote 5.

[15] For Nelder-Mead, this stopping criterion corresponds to the distance between the function values at all points of the simplex that is constructed in the optimization routine (see Press et al. (1996)). For DFNLS, this stopping criterion corresponds to the smallest radius of the trust-region that is used in the optimization routine (see Zhang et al. (2010)).

elements of the parameter vector is less than the chosen tolerance:

$$\max|\mathbf{x}_p^* - \hat{\mathbf{x}}_{p,s}^*| < \tau.$$

We next describe the data profiles and performance profiles which we use to benchmark the performance of optimizers along different dimensions.

3.1 Data Profiles

A data profile (Moré and Wild (2009)) plots the fraction of test problems that are solved successfully by a given solver (for a given success criterion) for a given computational budget—that is, for a given number of function evaluations or FEs, γ. First define the performance measure, $t_{p,s} > 0$, as the number of FEs that optimizer s needs to solve problem p successfully. Higher values of $t_{p,s}$ imply worse performance, and if the optimizer is not able to solve a problem at any budget, we set $t_{p,s} = \infty$. Next, define the "success rate" of the optimizer for a given number of FEs γ as

$$d_s(\gamma) \equiv \frac{1}{|\mathcal{P}|}\text{size}\left\{p \in \mathcal{P} \ : \ t_{p,s} \leq \gamma\right\},$$

where $|\mathcal{P}|$ denotes the cardinality of the set of all problems considered in the benchmarking study. To construct the data profile, we then plot each optimizer's success rates $d_s(\gamma)$ as a function of γ.

In addition to the success rate, we are also interested in measuring how poorly algorithms perform when they do not satisfy given success criteria; that is, How far away are failed implementations from the true optimum? To measure this, we define a complementary tool, which we call "deviation profiles," that reports the value of discrepancies (e.g., $|y_p^* - \hat{y}_{p,s}^*|$) averaged over all failed problems. We compute this measure for different FEs to measure how deviations evolve along the entire set of considered computational budgets. This provides information about optimizers' ability to get into the close neighborhood of an optimum and about possible difficulties of optimizers in locating the precise global optimum at different computational budgets.

3.2 Performance Profiles

A *performance profile* (Dolan and Moré (2002) and Moré and Wild (2009)) provides a more direct comparison of optimizers with each other. Whereas the data profile shows how the success rate of a given optimizer varies with computational budgets, a performance profile shows how the distribution of performance measures of a given optimizer

compares with those of other optimizers. To this end, define the *performance ratio* for solver s on problem p as

$$r_{p,s} \equiv \frac{t_{p,s}}{\min\{t_{p,s'} : s' \in \mathcal{S}\}}.$$

The denominator is the *performance measure* for the fastest solver for problem p among all solvers, so the ratio expresses performance relative to the best available solver. The ratio naturally has $r_{p,s} = 1$ for the best/fastest solver and $r_{p,s} > 1$ for all slower solvers. For an optimizer who fails to solve problem p at any considered budget, we set $r_{p,s} = \infty$. The *performance profile* of an optimizer $s \in \mathcal{S}$ measures the fraction of problems for which $r_{p,s}$ is smaller than or equal to α so that

$$\rho_s(\alpha) \equiv \frac{1}{|\mathcal{P}|}\text{size}\{p \in \mathcal{P} \ : \ r_{p,s} \leq \alpha\},$$

where $|\mathcal{P}|$ denotes the cardinality of the set of all considered problems $\mathcal{P}$. For example, $\rho_s(1)$ is the fraction of problems for which optimizer s is the best/fastest optimizer—that is, s is the optimizer that solves the problem with the smallest number of FEs among all optimizers considered—and $\rho_s(2)$ is the fraction of problems that solver s solves successfully for budgets that are at most twice as large as the budget of the fastest optimizer. More generally, $\rho_s(\alpha)$ is the cumulative distribution function of $r_{p,s}$, showing the probability that solver s can solve a problem p successfully for budgets that lie within a factor α of the best solver's budget for the same problem p. Hence, for a given α, higher values of $\rho_s(\alpha)$ mean better performance.

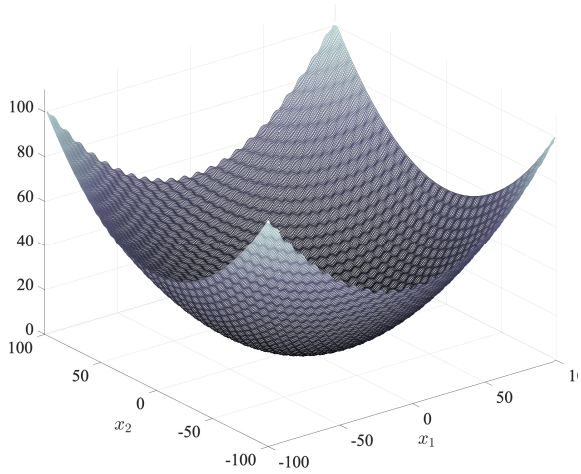
3.3 Coding Language and Specifications

NLopt algorithms are programmed in C, with the exception of StoGo, which is programmed in C++. We use a Fortran wrapper to use the optimizers from NLopt. The TikTak optimization algorithm is written in Fortran. We used the Gfortran compiler with no additional optimization flags. The code was run on a Linux cluster at Yale University.

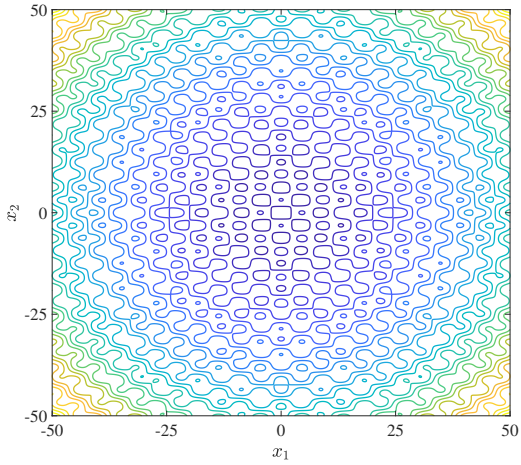
For all global algorithms, we implement a "polishing" local search as the very last step of the optimization routine which starts from the best (smallest) minimum that was found so far by the global algorithm. For the global NLopt algorithms, we use the local BOBYQA algorithm for the last polishing search. For TikTak, we use the local DFNLS for the last polishing search. Since the local BOBYQA algorithm seems to perform better in some application, we further polish the TikTak results with BOBYQA for the

FIGURE 1 – Griewank Function

(A) 3-D Plot



(B) Contour Map



economic application (but not for the test functions).[16]

4 Benchmarking Results for Standard Test Functions

For the benchmarking analysis, we select four test functions—Griewank, Levi, Rastrigin and Rosenbrock—that are among the most challenging ones for global optimization algorithms (see Ali et al. (2005)). Each function exhibits a combination of challenging features such as a large number of local minima, deep ridges, or very flat valleys where algorithms can get stuck. The functions are well defined for any number of dimensions. Section 4.2 benchmarks performance via data profiles for the 10-dimensional versions of each test function. Appendix C further reports the data profiles for 2-dimensional test functions. The performance profiles in Section 4.3 pool the results from the two- and 10-dimensional test functions.

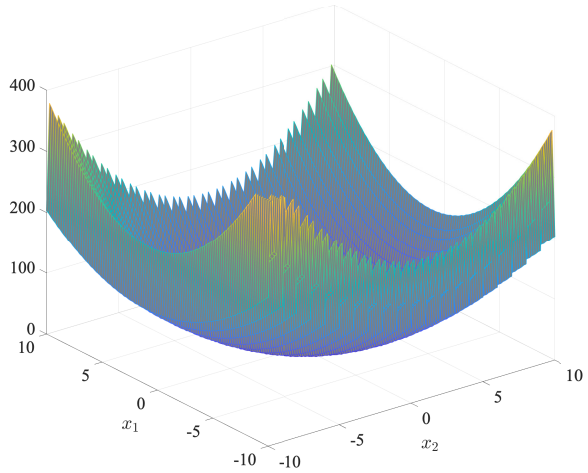
4.1 The Test Functions

The first three test functions display a large number of local minima, whereas the fourth one, Rosenbrock, is "valley-shaped" with a flat surface near the global minimum.

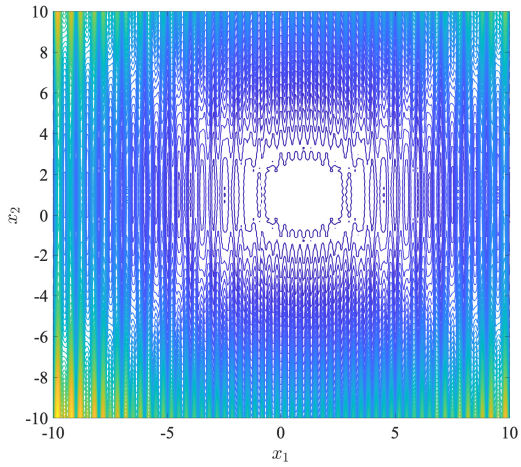
[16]The last polishing search is useful, as we sometimes use lower tolerances as stopping criteria in the local stages (i.e., 10^{-3}) to increase the speed of the global algorithm. It is therefore possible that the global algorithm comes into the close neighborhood of the true minimum, but it might require the additional polishing search to find the exact minimum.

FIGURE 2 – Levi No. 13 Function

(A) 3-D Plot



(B) Contour Map



Griewank function. The Griewank function in n dimensions is

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{a} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 2,$$

where we use the conventional choice of $a = 200$. We will focus on the hypercube domain $x \in [-100, 100]^n$. The global minimum is at $x = (0, ..., 0)$ with function value $f(0, ..., 0) = 1$.

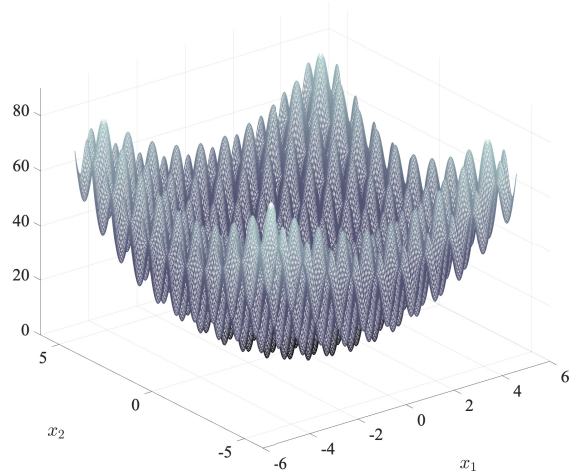
The left panel of Figure 1 plots the Griewank function in two dimensions. Globally, it has a very clear bowl shape owing to the quadratic first term, so the general location of the global minimum is hard to miss. However, thanks to the product of cosine terms, the function also exhibits a large number of small "ripples," which give rise to a large number of local minima spread throughout its domain. These ripples can be seen more clearly in the right panel, which plots the contour maps of the function.[17] Each closed circle on the map contains at least one local minimum, of which there are many. These challenges are already evident in two dimensions; how the function looks and what further complications arise in three or more dimensions are impossible to visualize or imagine.

Levi No. 13 Function. The Levi function has several variants. We use a version called Levi No. 13 that is commonly used for benchmarking optimizers. In n dimensions, it is

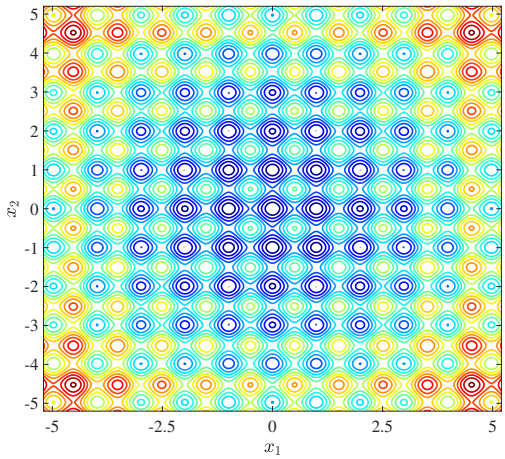
[17] To make the details visible, we plot the contour map on a smaller domain: $x \in [-50, 50]^2$.

FIGURE 3 – Rastrigin Function

(A) 3-D Plot



(B) Contour Map



given by

$$f(x) = \sin^2(3\pi x_1) + (x_n - 1)^2[1 + \sin^2(2\pi x_n)] + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + 1,$$

and we focus on the domain $x \in [-10, 10]^n$. The global minimum is located at $x = (1, ..., 1)$ with function value $f(1, ..., 1) = 1$.

The left panel of Figure 2 plots the Levi function in two dimensions. Like Griewank, the Levi function is also globally bowl-shaped, but it has another characteristic feature: a large number of deep ridges that run along the x_2 direction, each ridge showing a well-defined local minimum. These ridges are (somewhat) visible in the contour map in the right panel. Because the function value changes sharply from the sides of the ridge to the bottom, the contour map is dense in colors. The narrow bottom between ridges can be seen as the vertical white strips, indicating the lower objective values. The larger white circle in the middle is where the global minimum is located; the function becomes flatter in that region, further complicating the task of finding the optimum.

Rastrigin Function. The Rastrigin function in n dimensions is defined as

$$f(x) = An + \sum_{i=1}^{n}[x_i^2 - A\cos(2\pi x_i)] + 1,$$

where we set $A = 10$ and focus on the traditional choice of domain $x \in [-5.12, 5.12]^n$. The global minimum is at $x = (0, ..., 0)$ with function value $f(0, ..., 0) = 1$. Figure 3 plots the function values in two dimensions as well as the contour map. In some ways, Rastrigin combines the challenging features of the Griewank and Levi functions: like Griewank, it has a large number of local minima, each of which is buried at the bottom of a deep bowl (or silo), making it hard to "see" around, similar to Levi's ridges. As we shall see, Rastrigin will prove to be an especially challenging test for many of the optimizers we benchmark.

Rosenbrock Function. The Rosenbrock function in n dimensions is defined as

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] + 1,$$

and we focus on the domain $x \in [-100, 100]^n$. The global minimum is at $x = (1, ..., 1)$ with function value $f(1, ..., 1) = 1$.

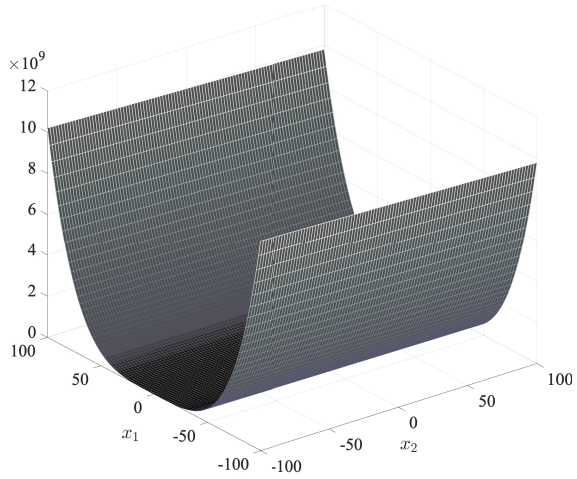
Figure 4 plots the function values and contour map. Rosenbrock is different from the previous functions in that its challenge is not the proliferation of local minima but rather the flat and long valley that contains the global optimum. The extremely large range of variation in the function values (from 1 to 10^{12}) makes it hard to see the shape of this valley. This is where the contour map becomes useful. To make the contours visible, we plot it only in the neighborhood of the global optimum, which lies not only on a flat surface but also one that actually branches off into two near the global optimum. This would make it easy for an optimizer to take the wrong branch and stop at a nearby point without finding the optimum. Figure 9 shows a different cut of the function on a log scale, which further illustrates this point.

4.2 Results: Data and Deviation Profiles

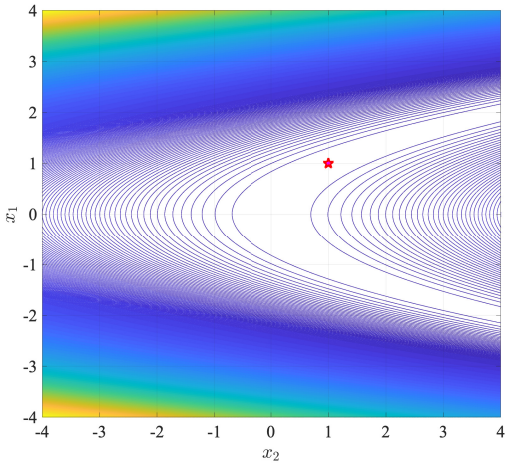
Let us briefly restate the terminology and abbreviations of the algorithms that we benchmark. The five global optimizers taken from the NLopt library are CRS, ISRES, ESCH, StoGo, and MLSL. For the local stage of MLSL, we use either Nelder-Mead or BOBYQA with two different convergence criteria (10^{-3} and 10^{-8}) for each version. We refer to these versions as MLSL-nm3, MLSL-nm8, MLSL-b3, or MLSL-b8. For the TikTak algorithm, we use either Nelder-Mead or DFNLS in the local stage, and we again use 10^{-3} and 10^{-8} as different local convergence criteria. We refer to the versions as TikTak-nm3, TikTak-nm8, TikTak-d3, and TikTak-d8. The three local optimization routines are the Nelder-Mead simplex algorithm, DFPMIN, and DFNLS.

FIGURE 4 – Rosenbrock Function

(A) 3-D Plot



(B) Contour Map



Note: The global optimum is marked with the red $*$ marker on the contour map.

Implementation Details and Definition of Success

We implement each minimization from 100 randomly drawn starting points (which are the same for all algorithms). Each starting point is counted as a different problem p, so that the set of problems P consists of 100 problems for each test function. We construct the data profile for each test function and each optimizer by implementing all minimizations at 30 different computational budgets. We then trace the data profiles by linearly interpolating between these data points.[18] For the 10-dimensional test functions, we set an upper bound of 100k FEs.[19] As mentioned above, we consider two success criteria based on the distance from either the true function value (F-val success) or the true parameter values (X-val success). For the test functions, we set the tolerance level for success at $\tau = 10^{-6}$. We report data and deviation profiles for each optimizer. Deviation profiles are meaningful only for failed searches, so no values are plotted if an optimizer reaches a 100% success rate at given levels of FEs. Figures 5, 7, 8, and 10 present the

[18] We can specify the number of FEs as explicit stopping criteria for the NLopt algorithms but not for TikTak. For TikTak, we instead increase the number of Sobol' points that are generated at the beginning, which increases the number of local searches and the number of FEs. The last "polishing" local search adds additional FEs to the computational budgets, which can vary across optimizers and problems p. To plot the data profiles, we therefore compute the average number of FEs for each optimizer at each stopping criteria across all 100 starting points from which we start each minimization. We then plot these averages on the x-axis of the data profiles.

[19] For the two-dimensional test functions, documented in Appendix C, we allow up to 20k FEs.

data profiles (top panel) and deviation profiles (bottom) for each test function. The left and right panels report the results for the F-val and X-val success criteria. This section documents the results for the 10-dimensional test functions. Appendix C shows the results for the two-dimensional test functions.

4.2.1 Griewank Function

Figure 5 shows the data and deviation profiles for the Griewank function. TikTak has the strongest performance under the F-val and X-val success criterion. All four versions of TikTak have steep data profiles and reach a 100% F-val success rate at low budgets (460 FEs for -d3, 710 for -d8, 1.3k for -nm3, and 2.6k for -nm8). The algorithm performs similarly well under the X-val criterion. CRS ranks next but requires higher computational budgets. CRS reaches a F-val success rate of 89% at 1.3k FEs and nears 100% for budgets above 5.3k FEs. X-val success rates are 0% for budgets below 10k FEs and fluctuate between 97% and 100% for budgets above 15k FEs.

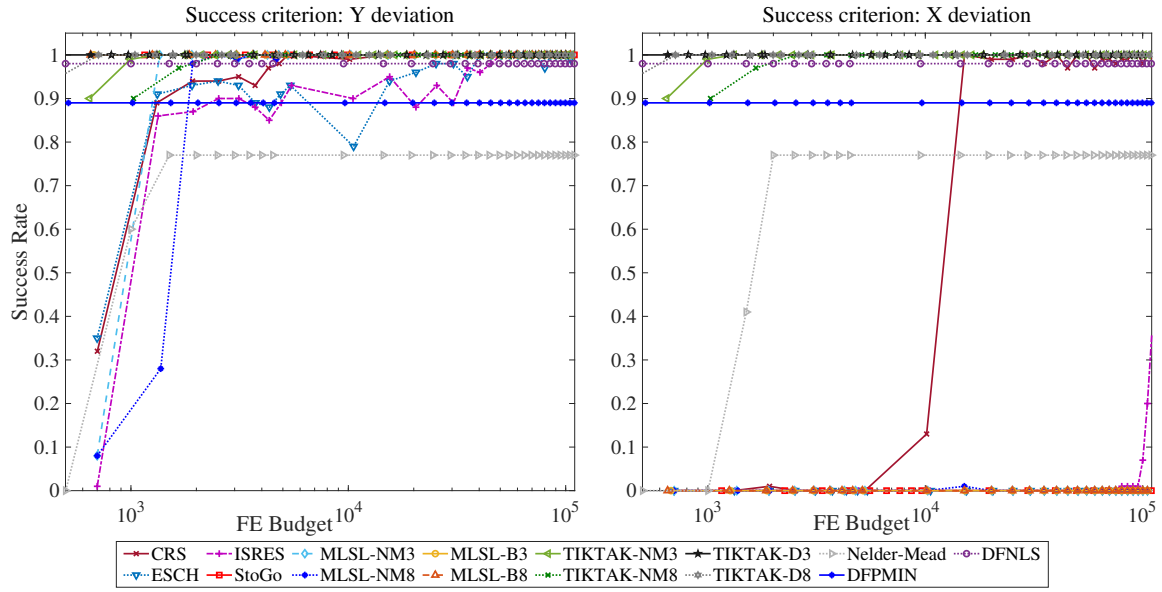
The performance of StoGo and MLSL is good but requires a closer inspection. The algorithms attain a 100% F-val success rate at small budgets, requiring between 700 (MLSL-b3), 1.1k (StoGo), and 2.5k FEs (MLSL-nm8). However, the algorithms never attain X-val success, as X-deviations always exceed the required tolerance level (10^{-6}). The deviation profiles (bottom panel) show that failed implementations of StoGo have the smallest deviations from the true X-values with values of $10^{-4.7}$ for all computational budgets. For MLSL, deviations are larger at smaller budgets and they decrease to 10^{-4} at large computational budgets ($10^{-4.5}$ for MLSL-b3). If deviations of this size are acceptable in a given application, then StoGo and MLSL could rank before CRS, with StoGo in second, MLSL-b in third, and MLSL-nm in fourth place.

ISRES has a weaker performance, with F-val success rates fluctuating between 85% and 98% for budgets between 1.3k and 45.3k FEs before reaching 100% for higher budgets. X-val success is very low, with 1% at 80k FEs and 7% at 100k FEs. Deviations are large for small budgets and fluctuate between 10^{-3} and 10^{-5} for budgets above 35k FEs. ESCH's performance lags behind the other global algorithms. The F-val success rate oscillates between 79% and 100% for all budgets above 1.3k FEs, and the algorithm is never successful under the X-val criterion. Deviations among failed implementations are large and stagnate around $10^{-3.5}$ for all budgets above 11k FEs.

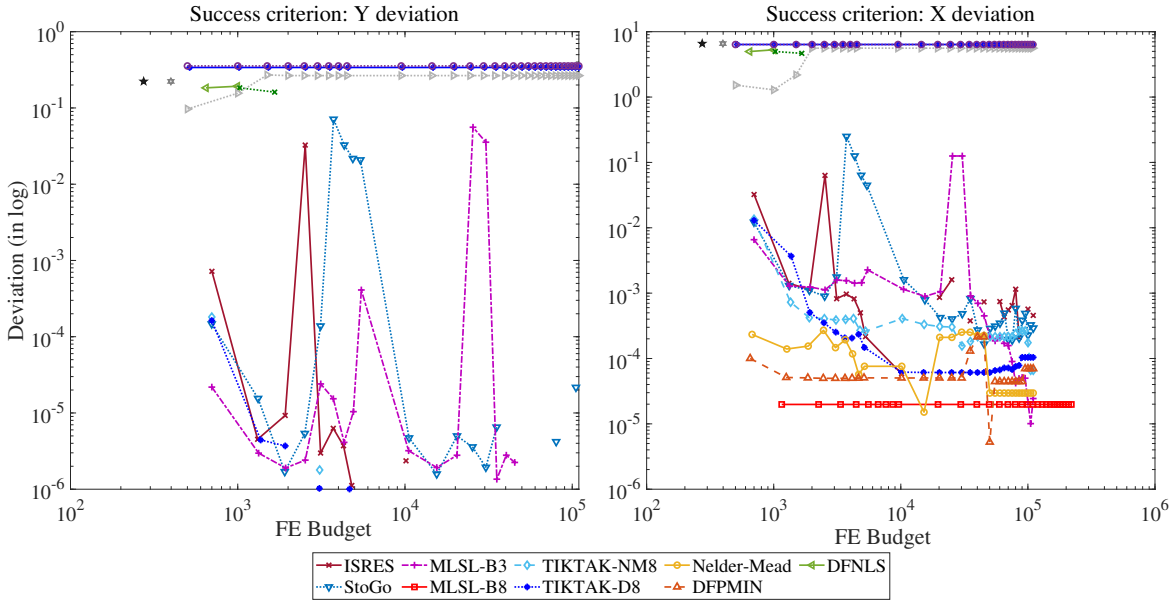
The three local algorithms perform similarly under both success criteria. DFNLS performs better than many global algorithms and reaches F-val and X-val success rates of 98% at all considered budgets. DFPMIN reaches a success rate of 89% at all budgets.

FIGURE 5 – Data and Deviation Profiles: Griewank, 10 Dimensions

(A) Data Profiles



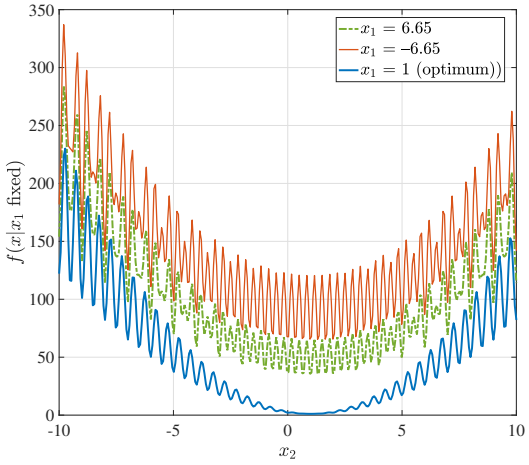
(B) Deviation Profiles



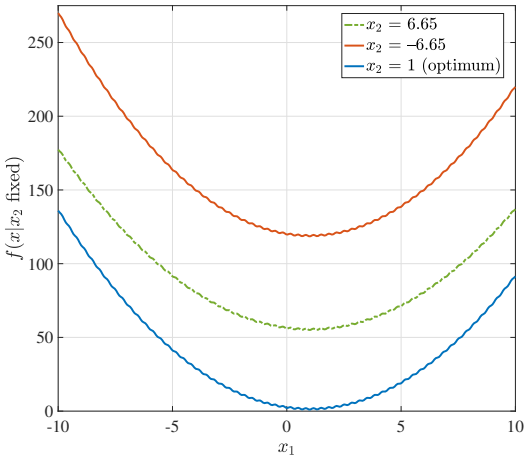
Notes: This figure shows optimizers' performance in minimizing the Griewank test function in 10 dimensions. The x-axis (plotted in logs) shows all computational budgets, ranging from 500 to 100k. The left and right panels show the data profiles under the F-val and X-val criteria, respectively. The bottom panel shows deviation profiles, which document the average distance between the returned and true minimum among all *failed* implementations, where failure is based on the F-val (X-val) criterion in the left (right). For each optimizer, the reported deviation is the average across all problems for which the optimizer failed at the corresponding computational budget. Therefore, no values are plotted if an optimizer has a success rate of 100% at a given computational budget. Both axes are plotted in $\log_{10}$ scale to improve readability.

FIGURE 6 – Levi No. 13 Function, Slices

(A) Values along x_2 dimension



(B) Values along x_1 dimension



Although a success rate of 89% may seem high, DFPMIN will not find the global minimum from 11% of the considered starting points. Nelder-Mead performs worst among the local algorithms, with a success rate that stagnates at 77% for budgets above 2k FEs. Average deviations among failed implementations are very large for all local algorithms (Panel B). The algorithms' success rates and deviations among failed implementations do not improve with larger computational budget. This is due to the local nature of their search, which can cause them to remain stuck at a local minima that can be far from the global minimum. The Griewank function provides a best-case scenario for the local algorithms, which perform worse on the other test functions and in the application to the estimation problem.

4.2.2 Levi Function

Figure 7 shows the data and deviation profiles for the Levi function. TikTak again performs best for both criteria, and all versions of TikTak reach success rates of 100% at low computational budgets. The fastest version is TikTak-d3, which reaches 100% F- and X-val success at 776 FEs, followed by TikTak-nm3 at 1.5k FEs, TikTak-d8 at 2k FEs, and -nm8 at 3.5k FEs.

CRS is the second-best-performing algorithm, but it requires higher budgets, reaching a 17% F-val success rate at 1.4k FEs and a 100% success rate at 5.3k FEs. CRS has the first X-val success of 5% at 4.3k FEs, which increases to 100% at 10k FEs.

The performance of StoGo is good but requires closer inspection. StoGo reaches an

F-val success rate of 100% at 1.2k FEs. It is never successful under the X-val criterion, but failed implementations come very close to the true X-values with deviations of $10^{-5.8}$ for all budgets. To the extent that this is acceptable for a given application, StoGo could rank in second place–ahead of CRS but behind TikTak. Notice that the performance of these three optimizers for the Levi function is quite similar to what we saw for Griewank above. MLSL-b8 ranks next, reaching F-val success rates over 96% at 4.9k FEs. X-val success rates fluctuate between 2% and 98% for all budgets. For budgets above 50k FEs, success rates are very low (3%) but deviations from the true X-values are small (around $10^{-5.5}$) among failed implementations.

ISRES needs larger budgets, reaching a 100% F-val success rate only at 40.5k FEs. X-val success increases slowly, reaching 20% at 60k and 96% at 85k FEs. Deviations from the true X-values among failed implementations remain around $10^{-4}/10^{-5}$ for the highest budgets. The remaining algorithms (MLSL-b3, MLSL-nm, and ESCH) never obtain X-val success and struggle to come close to the true X-parameters in case of failure. Among these, MLSL-b3 performs best, reaching 100% F-val success at 4.3k FEs and coming closest to the true X-parameters, with deviations of $10^{-4.5}(10^{-5.5})$ for budgets above 35k (85k) FEs. Notice, though, that 85k is more than 100 times the budget required by TikTak-d3 to reach the same success rate. ESCH and the MLSL-nm versions reach 100% F-val success at larger budgets (11k FEs for ESCH, 20k FEs for MLSL-nm3, 95k FEs for MLSL-nm8), and they never come close to the true X-parameters, with deviations remaining at 10^{-3} to 10^{-4}.

The three local algorithms perform poorly under both success criteria. Nelder-Mead stagnates with a 32% F-val and X-val success rate at 19.5k FEs. DFNLS reaches a 16% F-val and X-val success rate at 1.5-2k FEs. DFPMIN never exceeds a success rate of 2%. Failed implementation remain very far from the true parameters at all budgets.

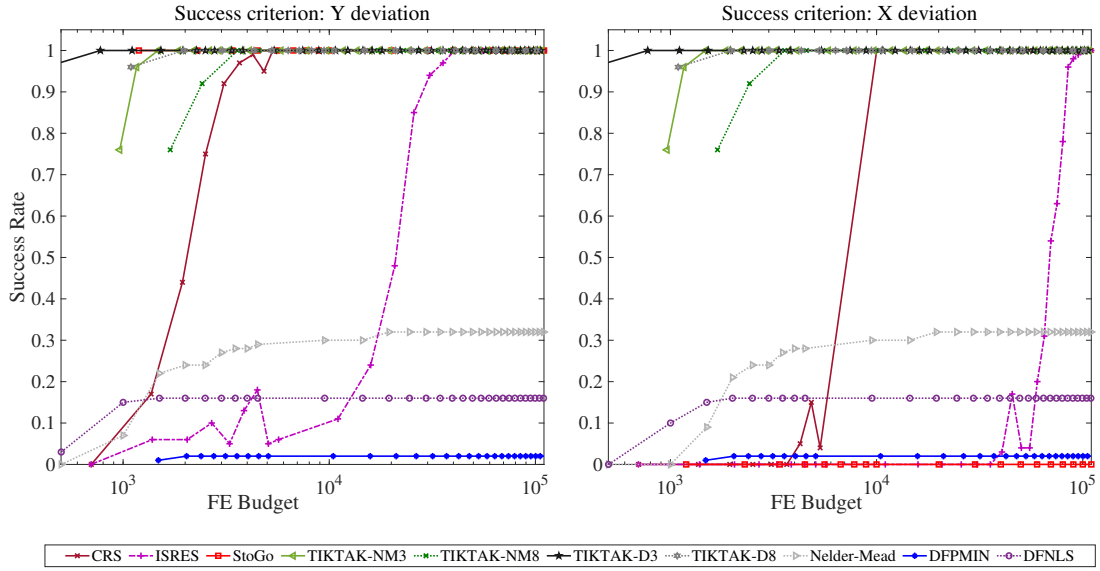
4.2.3 Rastrigin Function

The Rastrigin function has many deeply buried local minima, which make it more challenging for optimizers than the previous two functions. Four of the optimizers reach an F-val success rate of 100% at 10k FEs, two others achieve success rates between 10% and 30%, and the remaining six have success rates below 2%. StoGo has the best performance (Figure 8), as it reaches F-val and X-val success rates of 100% at 1.1k FEs (which is the lowest budget that we consider). TikTak-d3 ranks second, reaching 100% F-val and X-val success rates at 3.8k FEs.[20] It is closely followed by TikTak-d8,

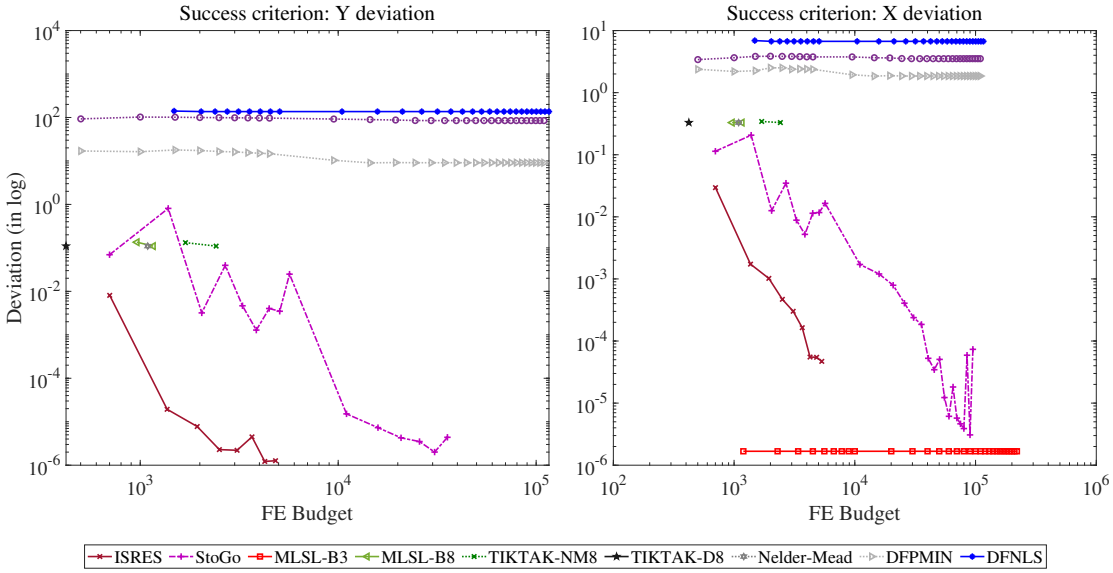
[20] At lower budgets, TikTak-d3 reaches F-val and X-val success rates between 83% and 99% for budgets between 1.2k and 3k FEs.

FIGURE 7 – Data and Deviation Profiles: Levi, 10 Dimensions

(A) Data Profiles



(B) Deviation Profiles



Notes: This figure excludes certain algorithms (ESCH and all 4 MLSL versions -nm3, -nm8, -b3, and -b8) for readability. The performance of these algorithms fluctuates substantially across different budgets of function evaluations, making the figure hard to read. We include the figures with the full set of algorithms in Appendix C.2. This figure shows optimizers' performance in minimizing the Levi test function in 10 dimensions. See the notes to Figure 5 for other details about the construction of these figures.

which has lower success rates at low budgets but also reaches 100% around 6.5k FEs. ESCH arguably ranks fourth, with F-val success rates fluctuating between 10% and 99% for budgets of 2k-30k FEs before reaching 100% for all larger budgets. ESCH's X-val success rate fluctuates substantially and is zero for most computational budgets. Failed implementations have deviations from the true X-parameters around 10^{-5} for budgets above 5k FEs (bottom right figure). TikTak-nm ranks next, as both versions perform poorly for small computational budgets. Tiktak-nm3 reaches F-val and X-val success rates above 94% only at budgets above 65k FEs, and Tiktak-nm8 reaches only 83% F-val and X-val success at 100k FEs. Deviations from the true values are large among failed implementations even for large computational budgets.

The remaining optimizers—CRS, ISRES, MLSL, and the local algorithms Nelder-Mead, DFNLS, and DFPMIN—perform poorly. They have no F-val or X-val success for the computational budgets that we consider, and failed implementations have large deviations from the true F- and X-values.

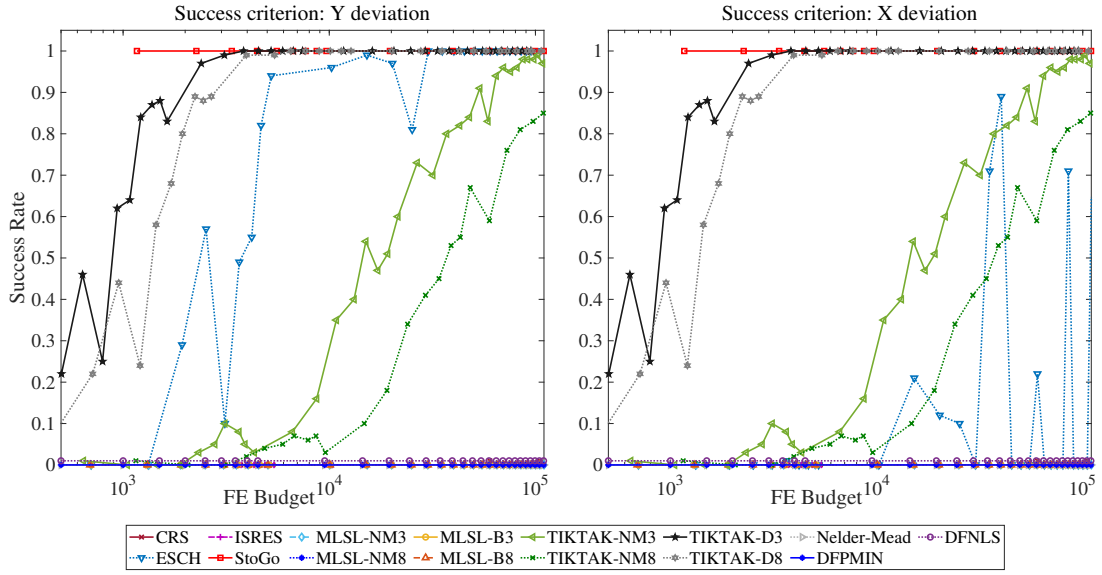
4.2.4 Rosenbrock Function

Rosenbrock provides the toughest test to optimizers. Even the best optimizers require large computational budgets to find the global optimum (cf. Figure 10). TikTak-d performs the best, reaching an F-val and X-val success rate of 100% at 12k FEs for -d3 and at 20k FEs for -d8. TikTak-nm8 follows closely, with 100% F-val success at 23k FEs. X-val success for these budgets lies between 99-100%, and failed implementations (i.e., the 1%) come very close to the true X-parameters, with deviations around $10^{-5.8}$. TikTak-nm3 requires larger budgets for full success: F-val and X-val success rates fluctuate between 9% and 99% for budgets below 55k FEs before settling at 100% at 62k FEs. Failed implementations have larger deviations from the true values.

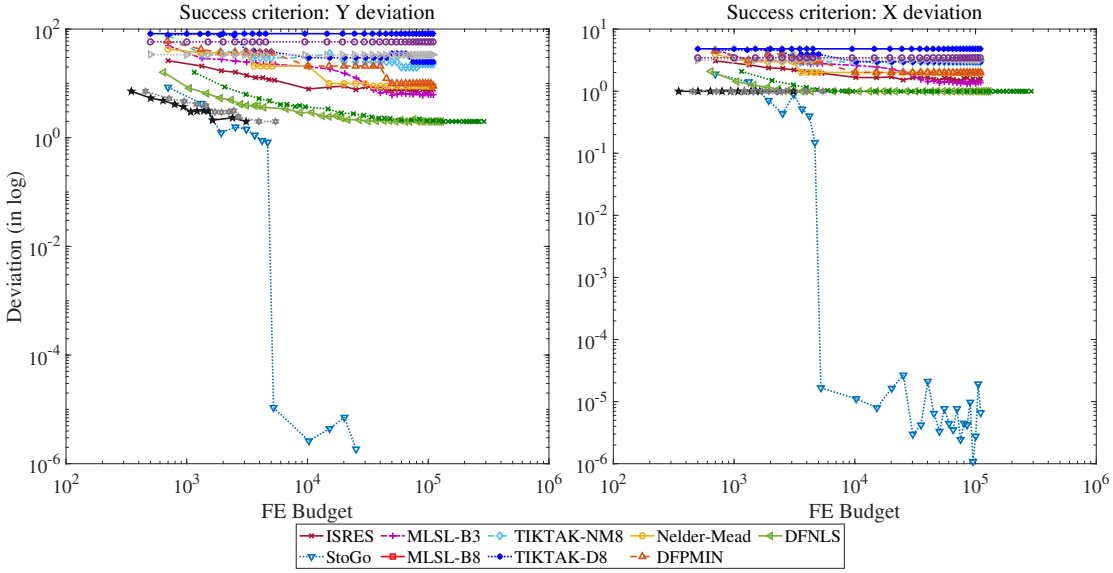
StoGo and MLSL all achieve a 100% F-val success rate–sometimes at lower budgets than TikTak–but they are never successful under the X-val criterion and struggle to come close to the true X-parameters. StoGo reaches 100% F-val success already at 6k FEs but deviations from X-values never improve below 10^{-3}. Both versions of MLSL-b reach 100% F-val success around 25-27k FEs, but they struggle to come close to the true X-parameters for any budgets: for MLSL-b8, average deviations remain around 10^{-4} for budgets above 30k FEs, and for MLSL-b3, they stagnate around $10^{-3.3}$. The MLSL-nm versions require larger budgets and perform worst. For MLSL-nm3, the F-val success rate spikes to 100% at 12k FEs but then drops again to 0% for larger budgets with very small deviations of $10^{-5.5}$ for all failed implementations at budgets above 12k FEs. X-val deviations stay large with values around $10^{-2.6}$ for all budgets. MLSL-nm8 achieves

FIGURE 8 – Rastrigin Function, 10 Dimensions: Data and Deviation Profiles

(A) Data Profiles



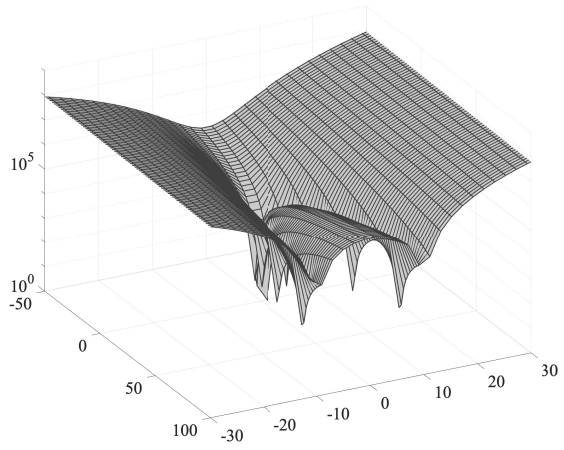
(B) Deviation Profiles



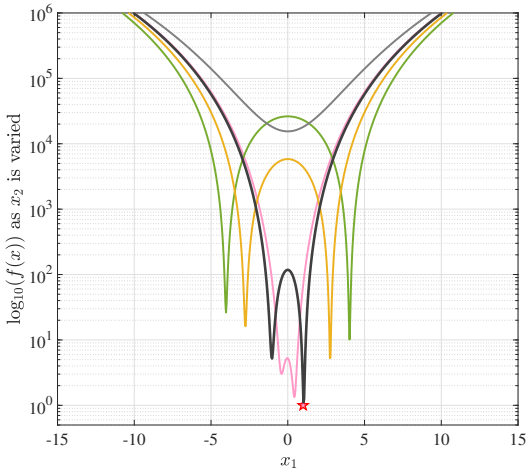
Notes: This figure shows optimizers' performance in minimizing the Rastrigin test function in 10 dimensions. See the notes to Figure 5 for other details about the construction of these figures.

FIGURE 9 – Rosenbrock Function, Different Perspectives, Log Scale

(A) Log Scale: Two Subtle Ridges



(B) Log Scale: Two Ridges Merge into One Near the Global Minimum



100% F-val success at 70k FEs, and X-val deviations are very large for most budgets and stagnate around $10^{-4.2}$ for budgets above 70k FEs.

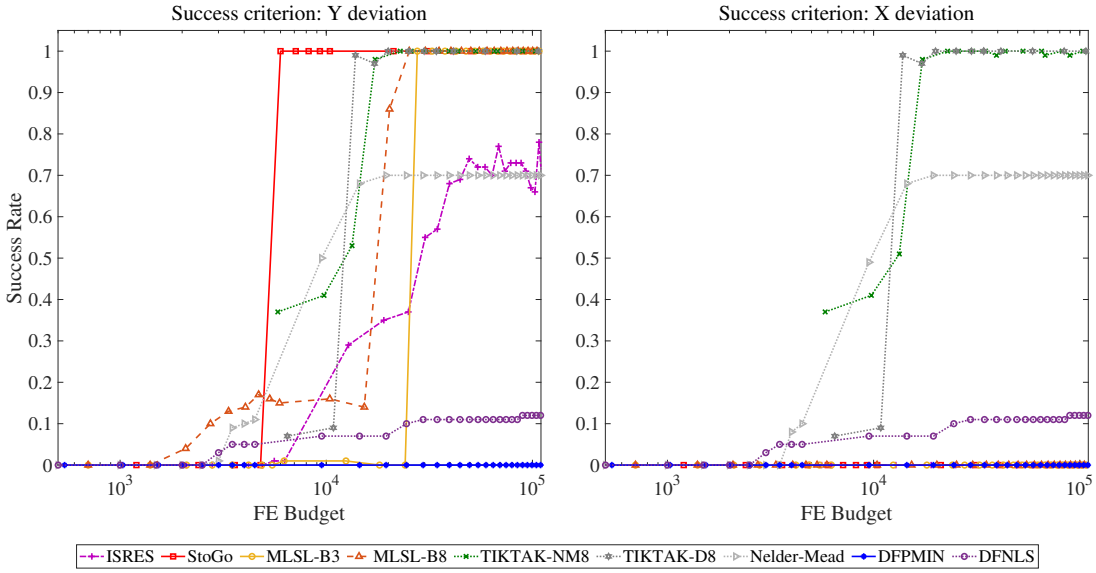
The remaining global algorithms–CRS, ESCH, and ISRES–never reach 100% success rates, and the deviations of failed implementations are large enough that their results are not reliable for any of the considered budgets. Among these algorithms, CRS performs best, reaching 86-96% F-val (X-val) success rates at 20k (30k) FEs. ESCH and ISRES have F-val success rates that fluctuate substantially for all budgets (for ESCH, 52-84% for budgets above 20k FEs; for ISRES, 65-77% for budgets above 60k), and they never achieve X-val success. Failed implementations have large average deviations from the true parameters which makes the results unreliable.

The local algorithms perform poorly under both criteria. Nelder-Mead stagnates at 20k FEs, with F-val and X-val success rates of 50%. DFNLS never exceeds a success rate of 12%. DFPMIN is never successful. Deviations from the true parameter values are large for all budgets.

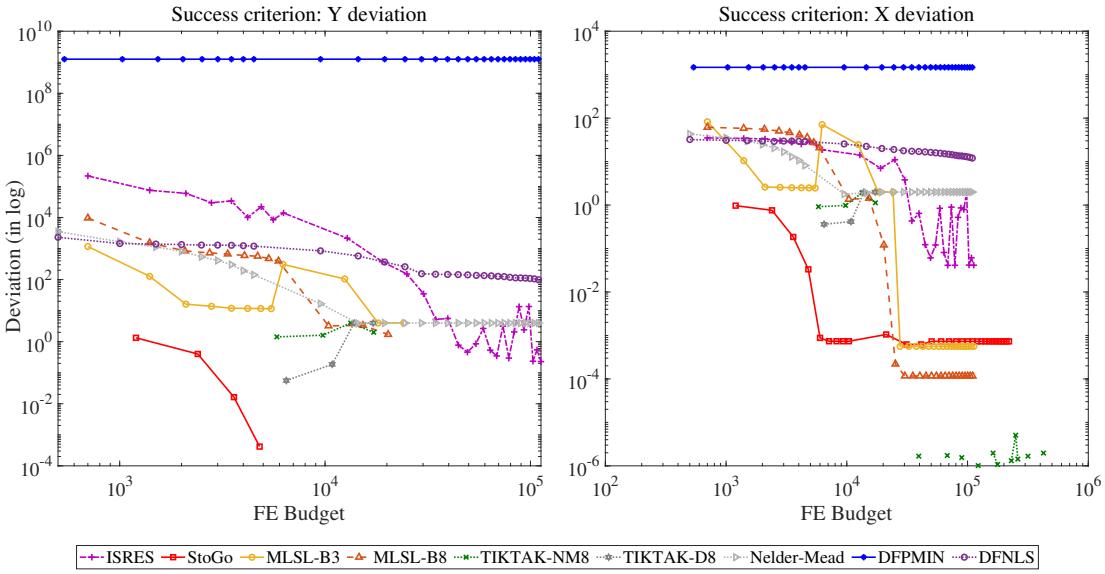
Overall, Rosenbrock presents interesting challenges not present in previous test functions, consistent with its reputation as a very challenging function to optimize. Despite the lack of many local optima and the apparent lack of deep ridges and ripples, it turned out to be harder to locate the global optimum. The global optimum located on a flat valley and the branching off from the objective surface are features that economists commonly face in real-world applications.

FIGURE 10 – Data and Deviation Profiles: Rosenbrock, 10 Dimensions

(A) Data Profiles



(B) Deviation Profiles



Notes: This figure shows optimizers' performance in minimizing the Rosenbrock test function in 10 dimensions. See the notes to Figure 5 for other details about the construction of these figures. This figure excludes certain algorithms (CRS, ESCH, MLSL-nm3, MLSL-nm8, TikTak-nm3, and Tiktak-d3) for readability. The performance of these algorithms fluctuates substantially across different budgets of function evaluations, making the figure hard to read. We include the figures with the full set of algorithms in Appendix C.3.

4.3 Results: Performance Profiles

We now turn to performance profiles, which give a complementary (and more direct) perspective on how optimizers compare with the best performer in each problem. The set of problems $\mathcal{P}$ consists of the four test functions and we pool the results from the two- and 10-dimensional test functions. We again start each problem from 100 randomly selected starting points, yielding a total of 800 problems. To trace the performance profiles, we use implementations of each minimization $p \in \mathcal{P}$ at 30 different computational budgets.

We saw in the previous section that success rates do not always increase monotonically with higher computational budgets.[21] We therefore compute the performance profiles for two different notions of success. In the first case, if an optimizer solves a problem successfully (say, under the F-val criterion) for budget level γ, we automatically define it to be successful at all higher budgets. We call this definition the "first success" criterion. The second case is more demanding: it defines success at a given budget γ only if the optimizer solves the problem successfully at γ and all higher budgets considered. We refer to this as the "permanent success" criterion. We report the performance profiles with both definitions.

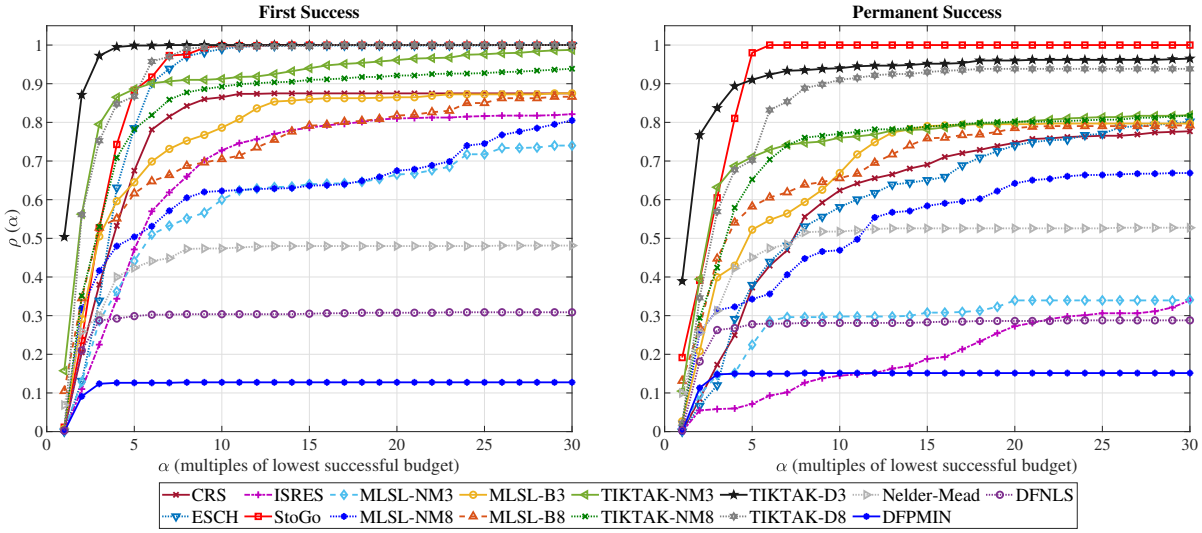
4.3.1 Results

Figure 11 plots the performance profiles for the F-val (top) and X-val (bottom) success criteria, each requiring either first success (left) or permanent success (right). The performance profiles confirm the success of TikTak-d that we have seen in the previous section: the four versions of TikTak share the top four places in the figures. Among the four versions, TikTak-d3 ranks at the top at all levels of α for both first success criteria. TikTak-d3 has the highest probability of being the fastest algorithm for a randomly chosen problem, $\alpha(1)$, which averages about 0.5 across all four panels. For the first success criterion (left panel), it is never more than three times slower than the fastest optimizer for any problem ($\rho(3) = 100\%$). For the permanent-F-val criterion, it has the strongest performance for $\alpha \leq 4$ before it is passed by StoGo. TikTak-d3 has only a 9% (6%) chance of being 5 (10) times or more slower than the best algorithm. For the permanent-X-val criterion, TikTak-d3 performs best at small budgets, but TikTak-d8 takes over for $\alpha \geq 6$ (the interpretation being that the chances for the -d8 version to be more than 5 times slower than the best optimizer are lower than the chances for the -d3 version). We therefore rank TikTak-d8 second, Tiktak-nm3 third, and TikTak-nm8 fourth.

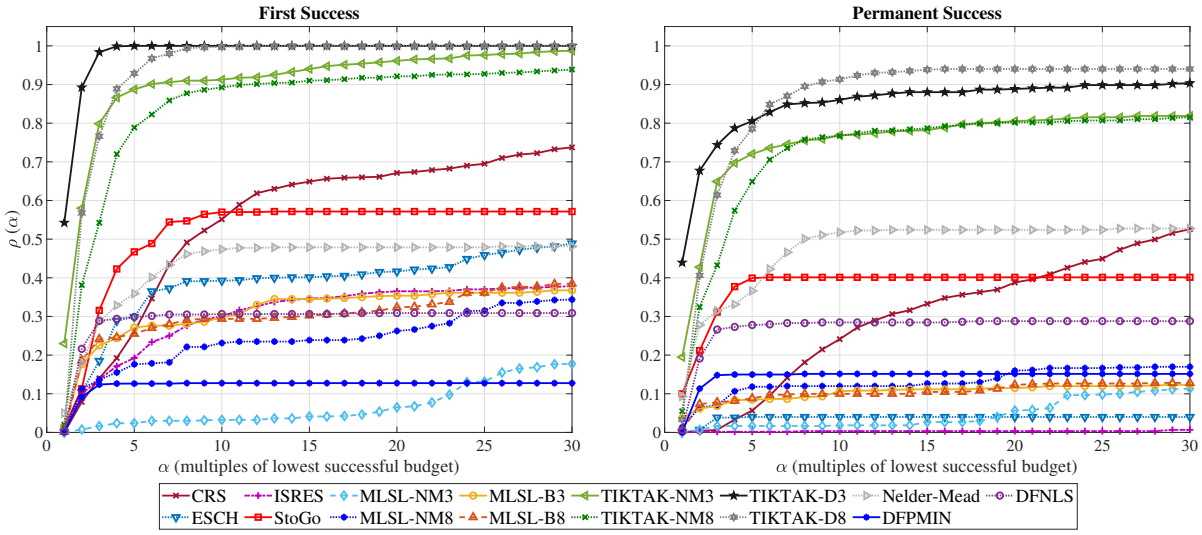
[21] This can happen because optimization routines use different strategies depending on the total budget that is allowed.

FIGURE 11 – Performance Profiles (All Test Functions in 2 and 10 Dimensions)

(A) Success: F-val Criterion



(B) Success: X-val Criterion



Notes: The figure plots the cumulative distribution function of $\rho(\alpha)$, which approximates (for large $\mathcal{P}$) the probability that an optimizer is within α factor of the best optimizer for a randomly chosen problem. $\mathcal{P}$ consists of the four test functions (Levi, Griewank, Rastrigin, Rosenbrock), each in 2 and 10 dimensions, and each minimization starting from 100 randomly chosen starting points. Panel A uses the F-val success criterion, requiring either first success (left graph) or permanent success (right graph). Panel B uses the X-val success criterion, again requiring either first success (left graph) or permanent success (right graph).

All NLopt algorithms have a lower X-val performance than TikTak, while the comparison of F-val performance varies more across algorithms and success criteria. It is hard to rank the NLopt algorithms because their performance profiles cross each other

and vary across success criteria. StoGo may be the best-performing NLopt algorithm across the various criteria. The previous section shows that the failed implementations of StoGo have very small deviations from the true X-parameters, so its performance profiles might understate its performance. Judging by the F-val criterion (top panels), a few optimizers clearly do better in best-case scenarios (i.e., for $\alpha \leq 5$). StoGo reaches a 88% (98%) probability of being within a factor of 5 of the fastest optimizer for the first (permanent) F-value success. For the first (permanent) X-val criteria, these numbers are 47% (40%).

CRS may rank next with a 68% probability for first F-value success and a 37% probability for permanent F-value success. CRS performs worse than StoGo at low budgets but surpasses StoGo at $\alpha = 11$ for first X-val success and at $\alpha = 22$ for permanent-X-val-success.

Arguably, MSLS-b follows next in the ranking, and performance is very similar for -b3 and -b8. MLSL-b has a 62-64% probability of being within a factor of 5 of the fastest optimizer for first F-val-success and 52-58% for permanent F-val-success. For X-val success, these numbers are 25-27% for first success and 8-9% for permanent success. The performance stagnates at larger budgets, and MLSL-b solves at most 39% (13%) of problems successfully for first (permanent) X-val success even at budgets that are 30 times larger than the budget of the best optimizer.

The remaining algorithms–ESCH, MLSL-nm, and ISRES–perform worse: their performance is more volatile, and they require larger budgets. ESCH performs relatively well for the F-val criterion, reaching 79% (38%) probability for first (permanent) F-val success at budgets 5 times larger than the fastest optimizer's budget. ESCH's performance is very poor under the permanent X-val criterion, for which the algorithm never solves more than 4% of problems successfully, even at budgets that are 30 times larger than the fastest optimizer. This indicates that the optimizer does not provide reliable results, even at larger budgets. MLSL-nm ranks next as it performs worse than ESCH for both F-val-criteria and for the first-X-val criterion; however, it performs better than ESCH for the permanent-X-val success criterion. ISRES arguably performs worst among the global optimizers, with particularly low performance under the permanent success criterion, for which the optimizer solves only 1% of problems successfully under the X-val criterion, and only 34% under the F-val criterion even when allowing for budgets 30 times larger than the fastest optimizer.

The performance rankings of local algorithms are similar under all four success criteria: Nelder-Mead comes first, followed by DFNLS and then DFPMIN in last place.

This ordering is not surprising in light of the extensive experience researchers had with these three algorithms over the years; the same ranking has typically emerged in real-life applications. Nelder-Mead does better than some of the NLopt algorithms, although its performance is a long way from the TikTak algorithm.[22] An additional weakness of the local algorithms is that deviations from the true parameters are very large for failed implementations (cf. Section 4.2), which makes it risky to rely on them in challenging optimization problems.

4.4 Taking Stock

Having examined the data and deviation profiles for each (10-dimensional) test function and the performance profiles for the entire set of problems (two- and 10-dimensional functions), we can now summarize the overall performance of each optimizer. To summarize our findings, Table I provides a broad ranking of the optimizers for each test function.

The different variants of the TikTak algorithm perform best across all criteria, with TikTak-d3 ranking at the top more than any other variant. TikTak-nm is typically in second place, slightly held back by the slower performance of Nelder-Mead in the local search stage. That said, Nelder-Mead has high reliability as a local algorithm, so in complex and higher-dimensional problems, it can be prudent to use Nelder-Mead in some of the local searches of TikTak. (This is also consistent with the experience the authors report in the papers listed in footnote 4.) The only exception to TikTak's top performance is Rastrigin for which StoGo is the clear top performer.

Among NLopt algorithms, one can make a case for StoGo, MLSL, and CRS. CRS performs well on relatively easier problems, such as the Griewank and Levi functions, but performs poorly on harder ones, such as Rastrigin and Rosenbrock. MLSL and StoGo are slower on easier problems and can fail to achieve the strict tolerance levels that we impose, but they do better in hard problems. When they "fail," they typically come quite close to the true parameter values. For example, StoGo is the fastest algorithm for Rastrigin (three times faster than the fastest TikTak version) and does well for Rosenbrock for budgets above 6k FEs, whereas CRS fails completely for Rastrigin and fluctuates between 86% and 96% success rates on Rosenbrock. When CRS fails, parameter values are far from the true values. This is an important drawback because an algorithm that cannot attain a (close to) 100% success rate is a risky choice in real-life applications, as increasing

[22] Nelder-Mead performs better in the two-dimensional than in the 10-dimensional test functions, which boosts Nelder-Mead's performance in the performance profiles.

TABLE I – Ranking Optimizers by Performance for Each Test Function

	Griewank	Levi	Rastrigin	Rosenbrock
	Successful Algorithms			
Rank				
1	TikTak-d3/d8	TikTak-d3	StoGo	TikTak-d3
2	TikTak-nm3/nm8	TikTak-nm3	TikTak-d3/d8	TikTak-d8
3	CRS	TikTak-d8	ESCH	TikTak-nm8
4	StoGo	TikTak-nm8	TikTak-nm3/nm8	TikTak-nm3
5	MLSL-b3/-b8	CRS		StoGo*
6	MLSL-nm3/-nm8	StoGo		MLSL*
7	DFNLS	MLSL-b8		
	Unsuccessful Algorithms (Unranked)			
	ISRES	ISRES	CRS	CRS
	ESCH	MLSL	MLSL-nm/-b	ISRES
	DFPMIN	ESCH	ISRES	ESCH
	Nelder-Mead	All Locals	All Locals	All Locals

Note: The ranking in the table summarizes our comparison of the performance of optimizers in Section 4.2. Some judgement calls were necessarily applied in cases in which optimizers' performance was close or overlapped. *For Rosenbrock, StoGo and MLSL have success only under the Y-val criterion. Neither optimizer achieves X-val success and stagnates around values slightly below 10^{-4}.

the computational budget does not get closer to the true minimum. MLSL performs similarly to StoGo but fails Rastrigin completely.

ESCH and ISRES have lower overall performance. ESCH has high F-val success rates (80%–100%) for all test functions; however, the algorithm is almost never successful for X-val, with the exception of some inconsistent success for Rastrigin. Failed implementations have small deviations from the true parameters for Griewank, Levi, and Rastrigin, but larger deviations for Rosenbrock. ISRES has low X-val success for Griewank and Levi and no success for Rastrigin and Rosenbrock. Failed implementations have large deviations, especially for Rastrigin and Rosenbrock. In addition, success rates of ISRES (as well as ESCH and MLSL) fluctuate substantially across computational budgets for several test functions.

Local algorithms can reach higher success rates than some of the global algorithms, but they all stagnate below 100% success, even when the computational budget is increased (with the minor exception of Nelder-Mead for Griewank). As just discussed,

algorithms that cannot reach a 100% success rate at any budget are not reliable for solving complex optimization problems. Another weakness is that the failed implementations of local algorithms return values far away from the true global minima–possibly because they remain stuck at a local minima. Global algorithms oftentimes do better in this dimension, with failed implementations stopping in a closer neighborhood of the true minima.

5 Benchmarking: An Economic Application

We now turn to an economic application to benchmark these global optimizers. A very common use of optimization algorithms in economics is in structural estimation/calibration, in which an objective function based on some distance measure between model and data moments is minimized by the choice of model parameters. The specific example we study in this section is a panel-data estimation of a stochastic process for labor income, taken from Busch et al. (2015), who study the business-cycle variation in higher-order labor income risk. We choose this particular economic application because it involves the minimization of a nonlinear and relatively high-dimensional function with seven parameters that shares many features and challenges that are common to economic applications. We first briefly describe the income process that is estimated and then present the benchmarking results.

5.1 A Stochastic Process for Individual Labor Income

Let $y_t \equiv \log Y_t$ denote the log labor income of an individual at time t, which evolves as follows:

$$y_t = z_t + \theta_t \tag{1}$$

$$z_t = z_{t-1} + \zeta_t, \tag{2}$$

where θ_t is an $i.i.d.$ transitory shock drawn from a Gaussian distribution, $\mathcal{N}(\mu_\theta, \sigma_\theta)$, and μ_θ is chosen so that $\mathbb{E}\left(e^\theta\right) = 1$. The permanent shock ζ_t to the process z_t is drawn from a distribution whose properties vary over the business cycle, modeled as a mixture of three normal distributions:

$$\zeta_t \sim \begin{cases} \mathcal{N}(\mu_{1t,}, \sigma_1) & \text{with probability } p_1 \\ \mathcal{N}(\mu_{2t}, \sigma_2) & \text{with probability } p_2 \\ \mathcal{N}(\mu_{3t}, \sigma_3) & \text{with probability } p_3, \end{cases} \tag{3}$$

with $\sum_{j=1}^{3} p_j = 1$. The business-cycle variation in the means is captured by introducing an indicator for the aggregate economy x_t (which can be GDP growth, the unemployment rate, and so on), which gets transmitted to the means by a factor ϕ. More specifically,

$$\begin{aligned} \mu_{1t} &= \bar{\mu}_t \\ \mu_{2t} &= \bar{\mu}_t + \mu_2 - \phi x_t \\ \mu_{3t} &= \bar{\mu}_t + \mu_3 - \phi x_t, \end{aligned}$$

where $\bar{\mu}_t$ is normalized so that $E(e^{\zeta_t}) = 1$ for all t. The business cycle is captured by $x_t \equiv -(\log \frac{GDP_{t+1}}{GDP_t})$, and GDP growth serves as an empirical measure of aggregate fluctuations.[23] Following Busch et al., we impose $p_2 = p_3$, and $\sigma_2 = \sigma_3$, leaving seven parameters to be estimated:

$$\Theta = (\sigma_\theta, p_1, \mu_2, \mu_3, \sigma_1, \sigma_2, \phi)\,.$$

The parameters are estimated using a method of simulated moments (MSM) estimator that minimizes the distance between 297 data moments and their simulated counterparts. We take the data moments from Busch et al. (2015), who compute them from panel data on individual-level earnings in Sweden.[24] To construct the corresponding model moments, we simulate 10 panels, each containing the income histories of 10,000 individuals. The simulated moments are computed for each panel and then averaged over the 10 panels. The objective function is the sum of squared distances between the data and model moments. The distance measure is the percentage difference, with a small scale adjustment to avoid moments with very small absolute values dominating the objective function; see Busch et al. (2015) for further details.

One particular challenge posed by this objective function is that a large number of moments depend on the percentiles of a distribution. Because a percentile corresponds to data from a single individual, when a finite number of individuals are simulated, these percentiles are not continuous in the underlying parameters of the model. This introduces jaggedness into the objective function, which often cannot be seen with the naked eye but can quickly make the job of optimizers much harder.

[23] Note that log GDP changes are standardized in the estimation.

[24] Key moments include the 10th, 50th, and 90th percentiles of the distribution of earnings changes over one, three, and five years during the 1979–2010 period, as well as the age profile of the cross-sectional variance of log income between ages 25 and 60.

5.2 Results

In this analysis, we consider only the global algorithms. We do not use StoGo, as it requires the gradient of the objective function, which does not have an analytical expression and is costly to compute numerically. For all algorithms, we start the global minimization at 10 randomly selected starting points (the same points for all algorithms), which provide us with the set of problems $\mathcal{P}$. We consider computational budgets up to 50k FEs.[25] The success tolerance, τ, is set to 10^{-2}, which is a sufficiently tight tolerance that the variation in parameter values within this neighborhood of the minimum is small as judged by their potential economic effects. Unlike with the test functions, the true global minimum is unknown. As is often done in the benchmarking literature, we take the smallest objective as the minimum value found by any optimizer and across all budgets. This smallest minimum was found by ISRES and is equal to $f(x) = 3.4416$. The corresponding parameter values are shown in Table C.1 in Appendix D.

To get a rough idea about how the objective function varies with each of the seven parameter values, note that Figure D.8 in Appendix D plots the one-dimensional slices of the objective surface by varying each of the seven parameters over its entire domain while fixing the remaining six parameters at their optimum.[26] There are a few takeaways from this figure. For six of the seven parameters (except x_7), the minimum objective value lies close to the bound of each parameter value. For four of the parameters, (x_1, x_3, x_5, x_6), the objective appears very flat near the boundary where the optimum lies. To get a clearer picture, Figure D.9 zooms in to the immediate neighborhood of the minimum for each parameter. The local view looks much different. It becomes clear that the optima for x_1, x_3, and x_5 are clearly interior, whereas for x_6, it is very close to the boundary. Furthermore, the objective is quite jagged in the x_3 and x_6 directions near the optimum, suggesting that optimizers can get trapped in a local optimum nearby and stop prematurely.

Data and Deviation Profiles

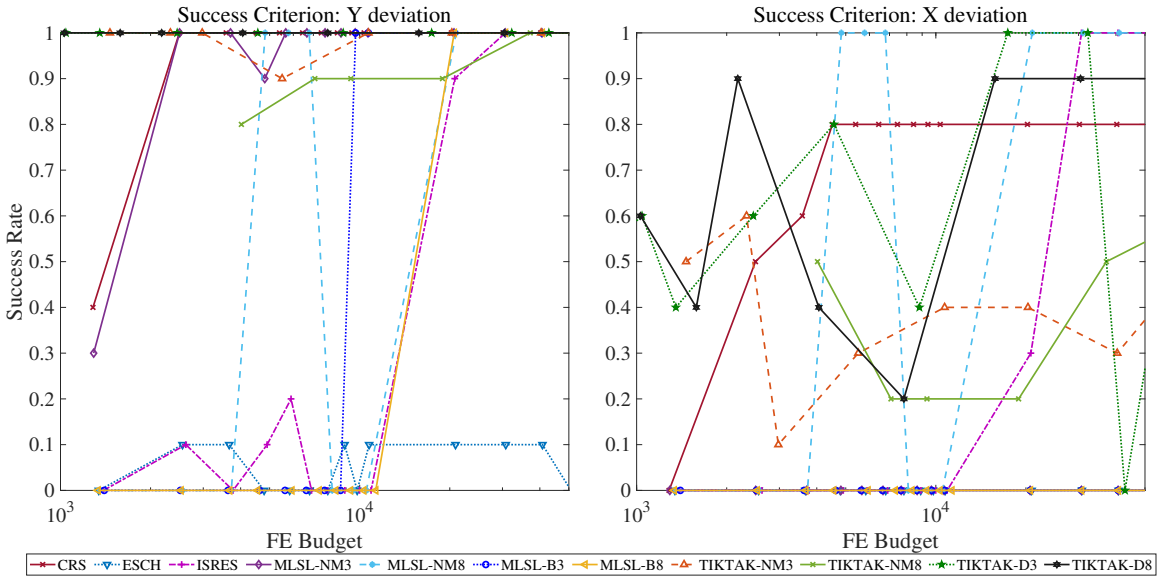
Figure 12 plots the data and deviation profiles. The data profiles show more fluctuations in success rates, indicating that the optimizers struggle more with this application

[25] For the NLopt algorithms, we implement minimizations at 14 different computational budgets using the number of FEs as explicit stopping criteria. For TikTak, we generate different numbers of Sobol' points to implement minimizations at different computational budgets.

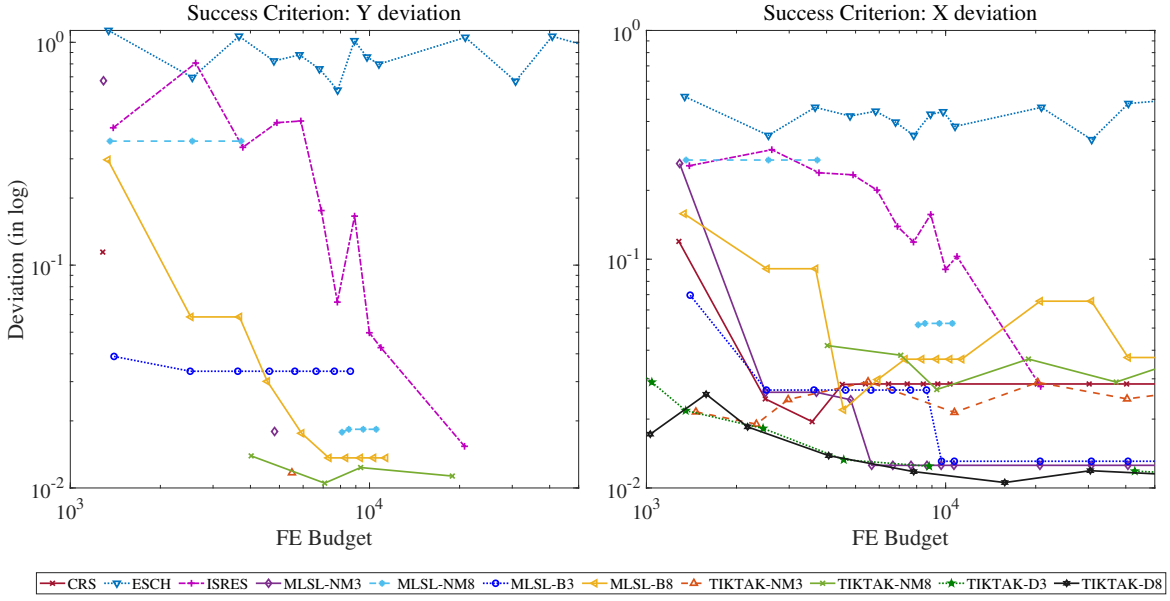
[26] Visualizing an objective function through these slices is informative, but it is nowhere near a complete description of the objective function, so it must be used with care.

FIGURE 12 – Data and Deviation Profiles for the Income Process Estimation

(A) Data Profiles



(B) Deviation Profiles



Notes: The x-axis (plotted in logs) shows computational budgets ranging from 1k to 70k FEs. See the notes to Figure 5 for other details about the construction of these figures.

than with the test functions. It is hard to clearly rank the algorithms, as their performance varies across available budgets and success criteria.

TikTak-d is arguably the best algorithm (-d3 and -d8), reaching a 100% F-val success rate at all computational budgets. TikTak-d comes very close to the true X-parameters for all implementations above 4.5k FEs with deviations around $10^{-1.9}$ among failed implementations. X-val success rates, which require deviations of at most 10^{-2}, fluctuate between 40% to 100% across all budgets (except a one-time drop to 0% at 43k FEs). TikTak-d8 performs similarly and comes as close to the true parameter values as the -d3 version, indicating that the additional FEs used by the tighter local tolerance seem to increase the computational budget without clear benefits.

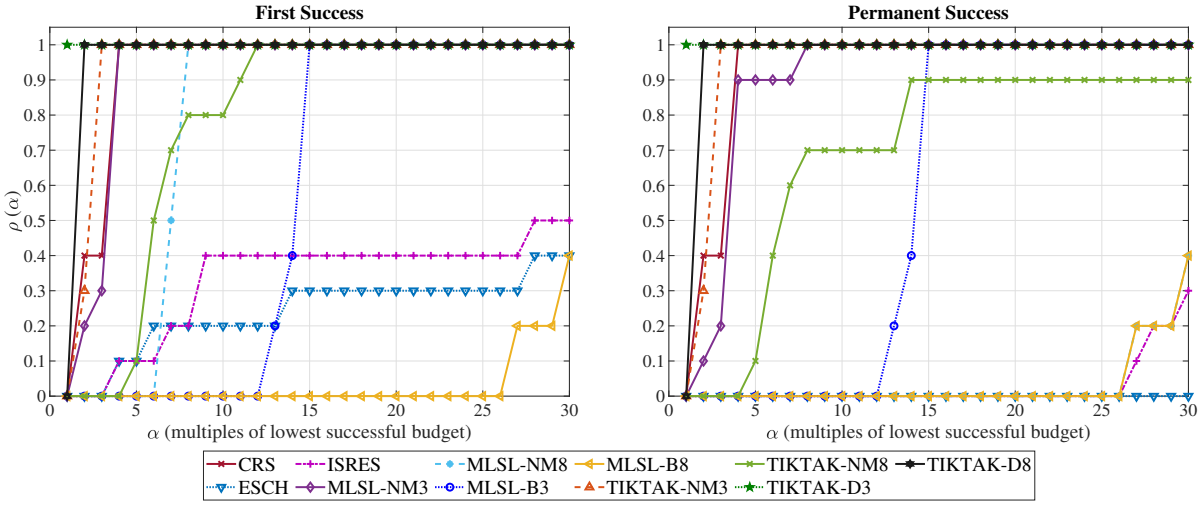
For applications in which small deviations around $10^{-1.9}$ are acceptable, MLSL-nm3 and MLSL-b3 also perform well, reaching F-val success rates close to 100% at 2.5k FEs for MLSL-nm3 and at 10k FEs for MLSL-b3. They never obtain X-val success, but they come close to the true X-parameters, with deviations of $10^{-1.9}$ for budgets above 5.5k FEs for MLSL-nm3 and above 10k FEs for MLSL-b3.

MLSL-nm8 and ISRES are good optimizers at high computational budgets but they perform poorly for low budgets. MLSL-nm8 obtains 100% F-val and X-val success rates for budgets above 20k FEs, and ISRES obtains them for budgets above 31k FEs. For lower budgets, both algorithms have low (ISRES) or widely fluctuating (MLSL-nm8) success rates, and failed implementations have large deviations from the true parameters, making the algorithms less reliable and risky to use at lower budgets.

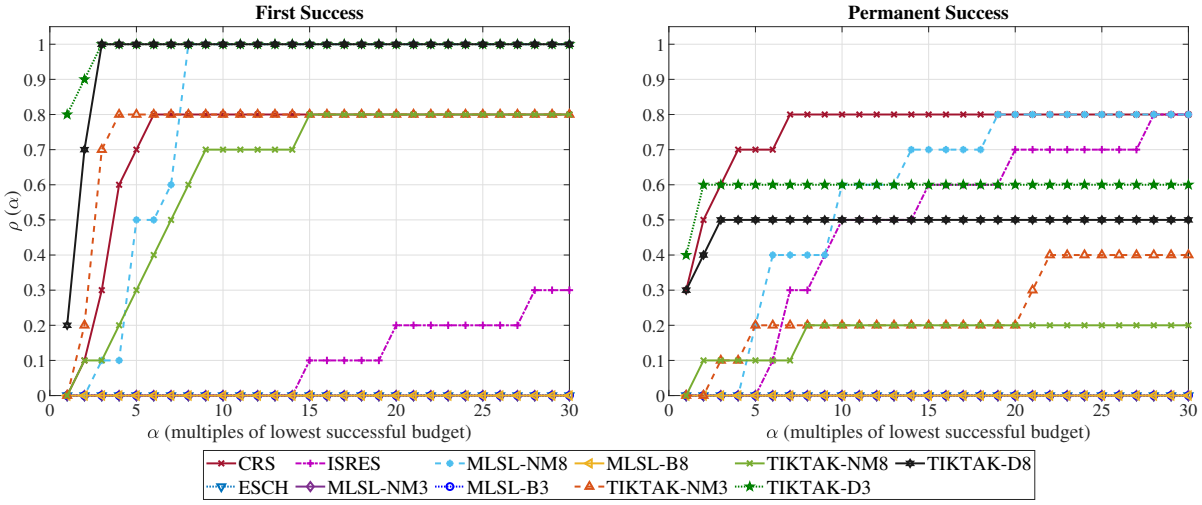
The other algorithms–CRS, TikTak-nm, MLSL-b8, and ESCH–perform less reliably. CRS and TikTak-nm both perform well under the F-val criteria, and they achieve some X-val success, but failed implementations have large deviations (around $10^{-1.5}$) from true X-parameters for all budgets. CRS reaches 80% X-val success rates at budgets above 4.5k FEs. X-val success rates for TikTak-nm fluctuate between 10% and 60% across all budgets. This implies that a substantial fraction of the implementations return X-values that are far from the global minimum even at the largest possible budgets, which question the reliability of the algorithms. MLSL-b8 reaches 100% F-val success at 21k FEs, never reaches X-val success, and has large deviations from the true X-parameters, which fluctuate between $10^{-1.2}$ and $10^{-1.4}$ for all computational budgets. ESCH ranks last and has poor performance across all criteria, with success rates close to zero and large deviations among failed implementations.

FIGURE 13 – Performance Profiles for the Income Process Estimation

(A) Success: F-val Criterion



(B) Success: X-val Criterion



Notes: The figure plots the cumulative distribution function of $\rho(\alpha)$, which approximates (for large $\mathcal{P}$) the probability that an optimizer is within α factor of the best optimizer for a randomly chosen problem. Here, $\mathcal{P}$ consists of the income estimation problem from 10 randomly selected starting points. See notes to Figure 11 for other details.

Performance Profiles

Figure 13 shows the performance profiles. The fast performance of TikTak versions is also seen here. The performance profiles of both TikTak-d versions rank very high, with the exception of the permanent X-val success criterion for which CRS, MLSL-nm8 and ISRES rank higher for larger computational budgets (see bottom right panel). One

point to keep in mind is that performance profiles rely uniquely on success rates, so they do not capture the nuances of optimizers that technically fail our success criteria but come very close to the true parameter values, as we have seen for TikTak-nm and MLSL, among others. This creates higher performance profiles for some optimizers, such as CRS or ESCH, even though we have seen in the data profiles and deviation profiles that their failed implementations tend to have large deviations from the true parameters, which make their performance less reliable.

6 Conclusion

In this paper, we have benchmarked the performance of six global and three local algorithms in optimizing difficult objective functions. In particular, we compare optimizers' performance in terms of reliability (success rates) and efficiency (required computational budgets). We use the algorithms to optimize a small suite of multidimensional test functions that are commonly used to benchmark algorithms in the applied mathematics literature. We are particularly interested in understanding optimizers' performance in typical economic applications, so we use the same optimizers to solve an estimation exercise that is common in economics. We consider the global optimizers CRS, ISRES, ESCH, StoGo, MLSL, and TikTak—with several variants for MLSL and TikTak—and the local algorithms Nelder-Mead, DFPMIN, and DFNLS.

We find that TikTak-d has the strongest performance for the test functions and the economic application, in terms of reliability and efficiency. The second-best optimizer is TikTak-nm, which performs well on the test functions and on the economic application for most but not all success criteria. TikTak-nm is less efficient than TikTak-d, as it requires larger computational budgets to solve problems successfully.

The relative performance of the NLopt algorithms varies across different test functions and the economic application. StoGo is arguably the best NLopt algorithm for test functions, but we have not included it in the economic application since the algorithm requires the gradient of the objective function which is unknown in the application. The MLSL versions (except -b8) and ISRES perform better in solving the economic application. However, ISRES performs poorly on all four test functions and MLSL performs poorly on two of them (Rastrigin and Rosenbrock). The performances of CRS and MLSL-b8 are a step behind: they perform poorly on two test functions (Rastrigin and Rosenbrock) and in the economic application in which they do not come close enough to the true X-parameters. ESCH's performance is the weakest among the algorithms in the test functions and the economic application. In research applications, we should

expect a global optimization algorithm to find the true global optimum reliably, even if this requires a large computational budget. CRS, MLSL-b8, and ESCH fail this test too often in our application, which raises questions about their suitability for the complex and high-dimensional problems found in economics.

Local algorithms display an unreliable performance, with low success rates, large deviations in failed implementations, and stagnant performance that does not improve with higher computational budgets (especially for DFPMIN and DFNLS). Although this result should not be surprising given that they are not designed for global optimization, these local optimizers are widely used for that purpose in real-life applications. Our analysis strongly cautions against that practice.

References

Ali, Montaz, Charoenchai Khompatraporn, and Zelda B. Zabinsky, "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems," *Journal of Global Optimization*, 2005, *31*, 635–672.

Busch, Christopher, David Domeij, Fatih Guvenen, and Rocio Madeira, "Higher-Order Income Risk and Social Insurance Policy Over the Business Cycle," Working Paper, University of Minnesota 2015.

_ , _ , _ , and _ , "Asymmetric Business Cycle Risk and Government Policy," Working Paper, University of Minnesota 2018.

Dolan, Elizabeth D. and Jorge Moré, "Benchmarking Optimization Software with Performance Profiles," *Mathematical Programming*, 2002, *91* (2), 201–213.

Ghebrebrhan, Michael, Peter Bermel, Yehuda Avniel, John D Joannopoulos, and Steven G Johnson, "Global optimization of silicon photovoltaic cell front coatings," *Optics express*, 2009, *17* (9), 7505–7518.

Guvenen, Fatih, "Macroeconomics with Heterogeneity: A Practical Guide," *Federal Reserve Bank of Richmond Economic Quarterly*, 2011, *97* (3), 255–326.

_ and Anthony A Smith, "Inferring Labor Income Risk and Partial Insurance from Economic Choices," *Econometrica*, November 2014, *82* (6), 2085–2129.

_ and Michelle Rendall, "Emancipation Through Education: A Macroeconomic Analysis," *Review of Economic Dynamics*, 2015, *18* (4), 931–956.

_ , Fatih Karahan, Serdar Ozkan, and Jae Song, "What Do Data on Millions of U.S. Workers Say About Labor Income Risk?," Working Paper 20913, National Bureau of Economic Research 2015.

_ , Serdar Ozkan, and Jae Song, "The Nature of Countercyclical Income Risk," *Journal of Political Economy*, 2014, *122* (3), 621–660.

Johnson, Steven G., "The NLopt nonlinear-optimization package," http://ab-initio.mit.edu/nlopt 2018.

Kaelo, P. and M. M. Ali, "Some variants of the controlled random search algorithm for global optimization," *Journal of Optimization Theory and Applications*, 2006, *130* (2), 253–264.

Kucherenko, Sergei and Yury Sytsko, "Application of Deterministic Low-Discrepancy Sequences in Global Optimization," *Computational Optimization and Applications*, 2005, *30*, 297–318.

Liberti, Leo and Sergei Kucherenko, "Comparison of Deterministic and Stochastic Approaches to Global Optimization," *International Transactions in Operations Research*, 2005, *12*, 263–285.

Madsen, Kaj, Serguei Zertchaninov, and Antanas Zilinskas, "Global Optimization using Branch-and-Bound," *Submitted to Global Optimization*, 1998.

Moré, Jorge J. and Stefan M. Wild, "Benchmarking Derivative-Free Optimization Algorithms," *SIAM Journal on Optimization*, 2009, *20* (1), 172–191.

Mullen, Katharine M., "Continuous Global Optimization in R," *Journal of Statistical Software*, 2014, *60* (6), 1–45.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in Fortran 77: The Art of Scientific Computation*, 2nd ed., New York: Cambridge Univ Press, 1992.

Press, William H, Saul A Teukolsky, William T Vetterling, and Brian P Flannery, *Numerical recipes in Fortran 90*, Vol. 2, Cambridge university press Cambridge, 1996.

Price, Wyn L., "A controlled random search procedure for global optimisation," *The Computer Journal*, 1977, *20* (4), 367–370.

Rinnooy Kan, Alexander and G. T. Timmer, "Stochastic Global Optimization Methods, Part I, Clustering Methods," *Mathematic Programming*, 1987, *39* (27-56).

_ and _ , "Stochastic Global Optimization Methods, Part II, Multilevel Methods," *Mathematic Programming*, 1987, *39* (57-78).

Runarsson, Thomas P. and Xin Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Transactions on evolutionary computation*, 2000, *4* (3), 284–294.

Runarsson, Thomas Philip and Xin Yao, "Search biases in constrained evolutionary optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2005, *35* (2), 233–243.

Silva-Santos, Carlos Henrique, Marcos Sergio Goncalves, and Hugo Enrique Hernandez-Figueroa, "Designing novel photonic devices by bio-inspired computing," *IEEE Photonics Technology Letters*, 2010, *22* (15), 1177–1179.

_ , _ , and _ , "Evolutionary Strategy Algorithm in a Complex Photonic Coupler Device Optimization," *IEEE Latin America Transactions*, 2018, *16* (2), 613–619.

Sobol', Ilya M., "On the Distribution of Points in a Cube and the Approximate Evaluation

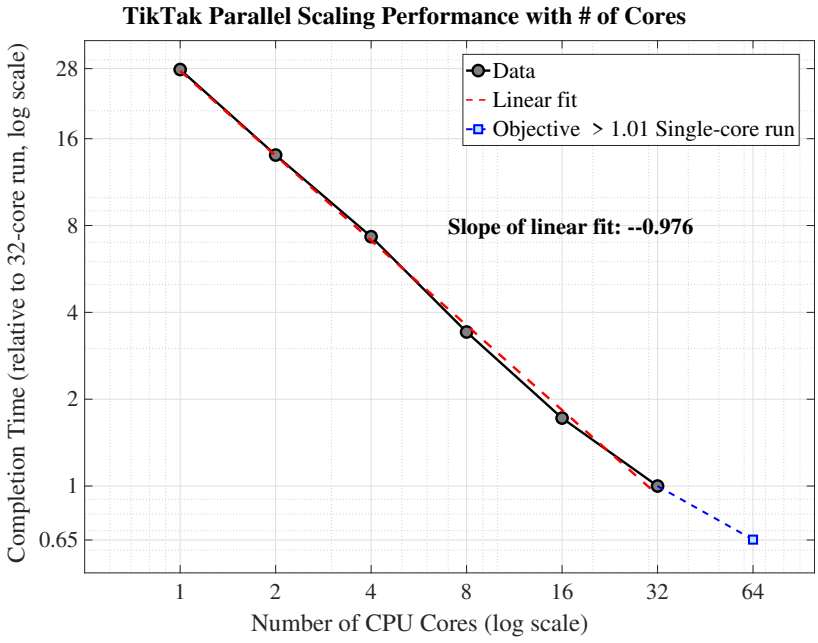
of Integrals," *Computational Mathematics and Mathematical Physics*, 1967, *7* (86-112).

Zhang, Hongchao, Andrew R. Conn, and Katya Scheinberg, "A Derivative-Free Algorithm for Least-Squares Minimization," *SIAM Journal on Optimization*, 2010, *20* (6), 3555–3576.

Zhigljavsky, Anatly and Antanas Žilinskas, *Stochastic Global Optimization* Springer Optimization and Its Applications, New York, NY: Springer, 2008.

ONLINE APPENDIX

FIGURE A.1 – Parallel Implementation of TikTak Scales Up Almost Linearly with Number of CPU Cores



Note: See https://www.fatihguvenen.com/tiktak for more details and a link to the source code.

A Parallel Performance of TikTak

B Appendix: Detailed Description of Algorithms

This section provides a description of the global optimization algorithms used in the paper.

B.1 Controlled Random Search with Local Mutation (CRS2-LM)

Controlled Random Search (CRS) algorithms were first introduced by Price (1977). An advantage of these algorithms is that they do not require much knowledge about the properties (e.g., differentiability) of the objective function that is minimized. The basic CRS algorithm has been modified and improved over time in several ways. More details on all variants of CRS can be found in Kaelo and Ali (2006). We will now describe the basic CRS algorithm and a variant of CRS with local mutation (CRS2-LM). CRS2-LM was developed and benchmarked by Kaelo and Ali (2006), who find that CRS2-LM performs better than all other CRS variants. In this paper, we therefore use the CRS2-LM algorithm from the NLopt library in our benchmarking exercise.

The basic CRS Algorithm The CRS algorithm (Price (1977)) is a direct search technique. Convergence results are based purely on heuristics. Given a bounded n-dimensional objective space X, the algorithm progresses as follows to *minimize* an objective function on this search space:

1. Initialize. Generate N uniformly distributed random points from the search space X and store them in an array S.

2. Rank the N points in S from best (x_b) to worst (x_w), where the best point is associated with the smallest function value $f(x_b)$.

3. Generate trial points $\tilde{x}$:

   (a) Randomly select $n+1$ points $x_1, x_2, ..., x_{n+1}$ with replacements from S.

   (b) Randomly select one vertex of the simplex (say, x_{n+1}) as a pole and reflect it through the centroid of the remaining points in the simplex to obtain the trial point $\tilde{x}$:

$$\tilde{x} = \frac{1}{n} \sum_{i=1}^{n} (2x_i - x_{n+1}) .$$

   (c) If $\tilde{x}$ lies outside of the bounds ($\tilde{x} \notin X$), return to Step 3 (a).

   (d) If the new trial point is worse than the worst point ($f(\tilde{x}) \geq f(x_w)$), return to Step 3 (a).

4. Update S. If this step is reached, then the new trial point $\tilde{x}$ must be better than the worst point; that is, $f(x_w) > f(\tilde{x})$. Therefore, x_w is replaced by $\tilde{x}$. The algorithm then returns to Step 2.

5. Repeat Steps 2 to 4 until a stopping rule is met. Usually, stopping criteria are based on the distance between the best and worst points (e.g., $f(x_w) - f(x_b) \leq \tau$).

Given this basic structure, several modifications aimed at improving the selection of trial points by suggesting changes to Step 3. Kaelo and Ali (2006) develop a new version of CRS, which is referred to as CRS2 with local mutation (CRS2-LM). CRS2-LM modifies Step 3(a) (i.e., the way in which the algorithm selects the $n+1$ points that form the simplex) and Step 3(b) (i.e., the rules for finding new trial points $\tilde{x}$) of the basic CRS algorithm. We will now describe the CRS2-LM algorithm in more detail as we use this version of CRS in our benchmarking exercise.

The CRS2-LM Algorithm (Based on NLopt) The CRS2-LM algorithm changes the method of generating the $n+1$ points that form the simplex. Now, the algorithm generates only n points randomly and always uses the smallest function value in S as the $(n+1)$-st point. The second change affects the rules for updating and discarding unsuccessful trial points. Recall that the basic CRS algorithm discards a new trial point $\tilde{x}$ if its function value $f(\tilde{x})$ is not better than the current worst point in the sample S. In the CRS2-LM version, the unsuccessful trial point $\tilde{x}$ is not discarded but is instead used to obtain another trial point $\tilde{y}$ by coordinate-wise reflecting $\tilde{x}$ through the current best point x_b. The CRS2-LM algorithm can be summarized by the following steps:

1. Initialize as in basic CRS.

2. Rank points as in basic CRS.

3. Generate trial points $\tilde{x}$:

   (a) Randomly select n points $x_2, x_2, ..., x_{n+1}$ with replacements from the search space X. Let $x_b = x_1$.

   (b) Obtain the next trial point $\tilde{x}$ as in CRS given the $n+1$ simplex vertices selected in 3 (a).

   (c) If $\tilde{x}$ lies outside of the bounds ($\tilde{x} \notin \Omega$), return to Step 3(a).

   (d) If the new trial point is worse than the worst point ($f(\tilde{x}) \geq f(x_w)$), then go to Step 4; otherwise, go to Step 5.

4. Local mutation of $\tilde{x}$:

   (a) Generate another trial point $\tilde{y}$ using the "unsuccessful" trial point $\tilde{x}$ and the best point x_b by coordinate-wise reflecting $\tilde{x}$ through the current best point x_b according to the following equation:

   $$\tilde{y_i} = (1 + \omega_i)x_{bi} - \omega_i\tilde{x_i},$$

   where i denotes the i-th coordinate of each point and ω_i is a random number in [0,1] drawn for each i.

   (b) If $f(\tilde{y}) \geq f(x_w)$, then no replacement is done and the algorithm returns to Step 3.

5. Update S. If this step is reached from Step 3, the new trial point $\tilde{x}$ is better than the worst point (e.g., $f(x_w) \geq f(\tilde{x})$). Therefore, x_w is replaced by $\tilde{x}$. If this step is reached from Step 4, the new trial point $\tilde{y}$ is better than the worst point ($f(x_w) > f(\tilde{y})$). Therefore, x_w is replaced by $\tilde{y}$ instead. The algorithm then returns to Step 2.

6. Repeat Steps 2 to 5 until a stopping rule is met. Usually, stopping criteria are based on the distance between the best and worst points, (e.g., $f(x_w) - f(x_b) \leq \tau$).

In our implementation, we use $N = 10(n + 1)$.

B.2 Improved Stochastic Ranking Evolution Strategy (ISRES)

Both ISRES and ESCH (see below) are Evolution Strategy (ES) algorithms. ES algorithms are based on the evolution of a population (set of individuals) along generations. The population is composed of two distinct groups: μ parents and λ offspring. During each generation, the population size changes from μ to $\mu + \lambda$ individuals. After the selection, the population again reduces to μ individuals. ES algorithms differ in the way in which offspring are generated and in the selection of surviving individuals.

The Improved Stochastic Ranking Evolution Strategy (ISRES), developed in Runarsson and Yao (2000, 2005), proposes a novel approach to balance objective and penalty functions stochastically and to rank candidate solutions to the minimization accordingly. The new ranking strategy is tested in Runarsson and Yao (2000) on a suite of 13 benchmark problems using a (μ, λ) evolution strategy. The authors furthermore point out that the new constraint-handling

technique (which is based on the stochastic ranking scheme) can be used in any evolutionary algorithm and is not limited to the evolution strategy. The evolution strategy in the NLopt application is based on a combination of a mutation rule (with a log-normal step-size update and exponential smoothing) and differential variation (a Nelder-Mead-like update rule). Overall, the authors find that using the suitable ranking method improves the performance of the algorithm significantly. The main advantage of the ISRES algorithm is therefore the constraint handling. ISRES supports arbitrary nonlinear inequality and equality constraints, in addition to bound constraints, and it performs well in problems with nonlinear constraints (shown in Runarsson and Yao (2000)).

B.2.1 Basic Evolution Strategy ($(\mu, \lambda)-$ES Algorithm)

The basic $(\mu, \lambda)-$ES algorithm can be summarized by the following steps:

1. Initialize. Generate λ individuals (x_i', σ_i'), where x_i' are uniformly distributed random points from the search space X, and $\sigma_{ij}' = (\bar{x_j} - \underline{x_j})/\sqrt{n}$, where $\underline{x_j}$ and $\bar{x_j}$ are the lower and upper bounds of the search space.

2. Rank the λ points that were generated from best (x_b) to worst (x_w) where the best point is associated with the smallest function value $f(x_b)$. Keep the best μ individuals $(x_i, \sigma_i), i \in \{1, \ldots, \mu\}$.

3. Replication. A new population of λ individuals is reconstituted by mutation of the μ individuals (x_i, σ_i) using a non-isotropic mutative self-adaptation rule:

$$\sigma_{k,j}' = \sigma_{rank(k),j} \exp(\tau' N(0,1) + \tau N_j(0,1)), k = \{1, \ldots, \lambda\}, rank(k) = mod(k-1, \mu) + 1$$

$$x_k' = x_{rank(k)} + \sigma_k' N(0,1)$$

$$\sigma_k' \leftarrow \sigma_{rank(k)} + \alpha(\sigma_k' - \sigma_{rank(k)}) \qquad \text{(exponential smoothing)}$$

4. Repeat Steps 2 to 4 until a stopping rule is met.

B.2.2 Improved Stochastic Ranking Evolution Strategy (ISRES)

Runarsson and Yao (2005) point out that the search by the $(\mu, \lambda)-$ES is biased toward a grid aligned with the coordinate system. To address this issue, the authors introduce a modification to the algorithm (differential variation) that can be thought of as a variation of the Nelder-Mead method. More specifically, Runarsson and Yao (2005) modify Step 3 by subdividing it into two sub-steps. A specific mutation is performed on each of the $\mu - 1$ best parents according to the following equation:

$$x_i' = x_i + \gamma(x_b - x_{i+1}), i \in \{1, \ldots, \mu - 1\}.$$

The search direction is now determined by the best individual and the individual ranked just below the parent being replicated (index $i+1$). The step length is controlled by the parameter γ. For these trials, the parent mean step size σ_i is copied unmodified.

The new algorithm can be described as follows:

1. Initialize. Generate λ individuals (x'_i, σ'_i), where x'_i is uniformly distributed random points from the search space X, and $\sigma'_{ij} = (\bar{x_j} - \underline{x_j})/\sqrt{n}$, where $\underline{x_j}$ and $\bar{x_j}$ are the lower and upper bounds of the search space.

2. Rank the λ points that were generated from best (x_b) to worst (x_w), where the best point is associated with the smallest function value $f(x_b)$. Keep the best μ individuals $(x_i, \sigma_i), i \in \{1, \ldots, \mu\}$.

3. Replication. A new population of λ individuals is reconstituted by mutation of the μ individuals (x_i, σ_i). There are two types of mutations:

   (a) Differential variation. For the $\mu - 1$ first individuals, the strategy parameter is kept unchanged, and the new individual coordinates are a combination of two parents x_i and x_{i+1} and the best point so far x_b:

   $$x'_i = x_i + \gamma(x_b - x_{i+1}), i \in \{1, \ldots, \mu - 1\}.$$

   (b) Standard mutation. For the remaining individuals (x'_k, σ'_k) for $k \in \mu, \ldots, \lambda$, the strategy parameter σ_k and the point x_k are mutated, according to a non-isotropic mutative self-adaptation rule:

   $$\sigma'_{k,j} = \sigma_{rank(k),j} \exp(\tau' N(0,1) + \tau N_j(0,1)), k = \{\mu, \ldots, \lambda\}, rank(k) = mod(k-1, \mu)+1$$

   $$x'_k = x_{rank(k)} + \sigma'_k N(0,1)$$

   $$\sigma'_k = \sigma_{k(i)} + \alpha(\sigma'_k - \sigma_{rank(k)}) \qquad \text{(exponential smoothing)}.$$

4. Repeat Step 2 and 3 until a stopping rule is met.

We use the following values for the parameters of the algorithm: $\lambda = 20(n+1), \lambda/\mu = 1/7$, $\tau = 1/\sqrt{2\sqrt{n}}, \tau' = 1/\sqrt{2n}$, and $\alpha = 0.2, \gamma = 0.85$.

B.3 Evolutionary Strategy with Cauchy Distribution (ESCH)

ESCH (Evolutionary Strategy Algorithm with Cauchy Distribution) is an evolutionary algorithm developed by Silva-Santos et al. (2010, 2018). ESCH is based on an $\mu + \lambda$-evolution strategy algorithm. The algorithm creates an initial population that is then iteratively recombined according to a single point recombination, and individuals undergo mutations generated by a Cauchy distribution. At each generation, the best μ individuals are selected from the entire population ($\mu + \lambda$ individuals).

The ESCH algorithm can be summarized by the following steps:

1. Initialize. Generate μ individuals x_i, where x_i is randomly generated according to a Cauchy distribution in the search space X. These are the parents P.

2. Crossover replication. Generate λ offspring. For each offspring k in $\{1, \ldots, \lambda\}$, randomly choose two parents from P: p_1 and p_2. Randomly choose an index $j_{threshold}$ in $\{1, \ldots, n\}$.

The first j components are copied from parent 1, and the remaining $n - j$ components are copied from parent 2 so that

$$x_{kj} = p_{1j}, j \in \{1, \ldots, j_{threshold}\}$$

$$x_{kj} = p_{2i}, j \in \{j_{threshold} + 1, \ldots, n\}.$$

3. Mutations. Create M mutations. For each mutation, randomly draw an individual among the $\lambda - \mu$ offspring; call the individual i_0. Randomly draw a dimension from the parameter space ($j \in \{1, ..., n\}$). Replace the component $x_{i_0 j}$ with a draw from a Cauchy distribution.

4. Selection. Rank the entire population (μ parents and λ offspring) and select the best μ individuals.

5. Repeat Steps 2 to 4 until a stopping rule is met.

We use the following values for the parameters of the algorithm: $\mu = 40, \lambda = 60$, and $M = 60 \times n/10$.

B.4 Multi-Level Single-Linkage (MLSL)

MLSL is a multistart algorithm that starts several local optimizations from a sequence of starting points that can be generated either with a pseudo-random number or with a Sobol' low-discrepancy sequence. The algorithm is proposed by Rinnooy Kan and Timmer (1987a,b). The NLopt version that we are using in this paper relies on Sobol's low-discrepancy sequence, which has been shown to improve convergence rates as Sobol' sequences cover the search space more efficiently (Kucherenko and Sytsko (2005)). The NLopt library allows specifying different local search algorithms, and we use the Nelder-Mead simplex algorithm. In addition, MLSL has a "clustering" heuristic that prevents the algorithm from performing repeated searches that are likely to converge to identical local optima. MLSL has been found to be very effective when used with a fast gradient-based local search algorithm on smooth problems (Ghebrebrhan et al. (2009)). It is not obvious, however, that this performance carries over to non-smooth global optimization problems in economics.

The MLSL algorithm proceeds along the following steps:

1. Draw N elements (from the Sobol' sequence) from the search space X and add them to S (initially empty).

2. Rank the elements in S according to their function values. Select the best $\gamma||S||$ values and store them in $\tilde{S}$.

3. For every point x in $\tilde{S}$:

   (a) Implement a local search starting from x, unless x is a local minimum previously found (i.e., unless x is already in X^*), or if there is another point x_j in S such that $f(x_j) < f(x)$ and $||x - x_j|| \leq r$. If one of these conditions is met, skip this step.

   (b) Add the minimum found by the local search to X^*.

4. Repeat Steps 1 to 4 until a stopping rule is met. Select the best element from X^*.

We use the following parameter values for the algorithm: $N = 4$ and $\gamma = 0.3$.

B.5 Stochastic Global Optimization (StoGo)

StoGo was developed by Madsen et al. (1998), and it uses a branch-and-bound technique. The algorithm proceeds by dividing the search space into smaller hyper-rectangles. Within these areas the algorithm then implements local optimizations, which use a gradient-based local search algorithm. A potential drawback of this algorithm is therefore that the function needs to be differentiable, since the local search algorithm is gradient based.

The main steps of the algorithm are as follows:

1. Initialization. Initialize $C = X$, where C is the set of candidate boxes (hyper-rectangles). Initialize $G = \emptyset$, which represents the set of garbage boxes.
2. Rank boxes $\tilde{B}$ in C according to the minimum function value among all points in $\tilde{B}$ computed during iteration. Store the best box from C in B and remove it from C (i.e., B as the smallest known function value).
3. Randomly draw a set S of N points in B . Evaluate $f(x)$ for $x \in S$. Start local search from each point in S using the Dogleg method (gradients are estimated using forward differences):
   (a) If all local searches end up out of the box B, remove B from C, and add B to garbage set G.
   (b) If all local searches converge to the same point (local minimum) x^*, add x^* to C. Remove B from C, and add B to garbage set G.
   (c) Else (lower bound reduction), there are several local minima found in B:
      i. Estimate the lowest point in B, $lb(B)$ using
      $$lb(B) = \min_{x \in B} \{f(x_{min}) - maxGrad \cdot ||x - x_{min}||\},$$
      where x_{min} is the smallest known function value in B. $maxGrad$ is the maximum value of the gradient, which is estimated at each point generated by the local searches.
      ii. If $lb(B) > fbound$, remove B from C, and add B to garbage set G.
      iii. Else: subdivide. Compute the centroid of two best local minima in C and the dimension-wise dispersion from the local minima in C to the centroid. Select the dimension with the highest dispersion. Split B in two boxes along this dimension at the centroid. By construction of the centroid, each subdivision contains at least one of the local minima. Put these two boxes in candidate set C .
4. Repeat Steps 2 to 3 until C does not contain any boxes (only singletons).

5. Remove an arbitrary box from garbage set G and store it in B. Create two subsets B_1 and B_2 from B in the following way:

   (a) If B has no known local minimum, split B in two along the longest dimension. Add B_1 and B_2 to C.

   (b) If B has exactly one known local minimum x^*, split B in two along the dimension for which x^* is farther away from boundary of B. Add B_1 and B_2 to C.

   (c) If B has several known local minima, compute the centroid of two best local minima in C and the dimension-wise dispersion from the local minima in C to the centroid. Select the dimension with the highest dispersion. Split B in two boxes along this dimension at the centroid. By construction of the centroid, each subdivision contains at least one of the local minima. Put these two boxes in candidate set C .

6. Repeat Step 5 until garbage set G is empty.

7. Repeat Steps 2 to 6 until a stopping rule is met.

B.6 TikTak

In this section, we summarize the main steps of TikTak as it is used in this paper. For an overview of this version of TikTak, see also Section 2. A more general description of the TikTak algorithm is available in Guvenen (2011).

- *Step 0.* **Initialization**:

  1. Determine bounds for each parameter.
  2. Generate a sequence of Sobol' points with length N.
  3. Evaluate the function value at each of these N Sobol' points. Keep the set of N^* Sobol' points[27] that have the lowest function values, and order them in descending order, as $s_1, \ldots, s_{N^*}$, with $f(s_1) \leq \cdots \leq f(s_{N^*})$.
  4. Set the global iteration number to $i = 1$.

- *Step 1.* **Global stage**:

  1. Select the i^{th} value (vector) in the Sobol' sequence: $\mathbf{s}_i$.
  2. If $i > 1$, read the function value (and corresponding parameter vector) of the smallest recorded local minimum from the "`wisdom.dat`" text file. Denote the lowest function value found so far (as of iteration $i-1$) as f_{i-1}^{low} and the corresponding parameter vector as $\mathbf{p}_{i-1}^{low}$.
  3. Generate a starting point (i.e., initial guess) $\mathbf{S}_i$ for the local search by using the convex combination of the Sobol' point $\mathbf{s}_i$ and the parameter value $\mathbf{p}_{i-1}^{low}$ that generated the best local minimum found so far: $\mathbf{S}_i = (1-\theta_i)\mathbf{s}_i + \theta \mathbf{p}_{i-1}^{low}$. The weight parameter $\theta_i \in [0, \overline{\theta}]$ with $\overline{\theta} < 1$ increases with i.[28]

[27] In this paper, we use $N^* = 0.1 \times N$.

[28] In this paper, we use the following function to increase the weight parameter: $\theta_i = \min\left[\max[0.1, (i/N^*)^{1/2}], 0.995\right]$.

- *Step 2:* **Local stage**:
  - Select a local optimizer (in this paper, we use either Nelder-Mead and DFNLS) and implement a local search at the identified starting point $\mathbf{S}_i$ until a local minimum is found.
  - Select a stopping criterion for the local search algorithm (in this paper, we use tolerances of either 10^{-3} or 10^{-8} as convergence criteria).
  - Open the `wisdom.dat` file and record the local minimum (function value and parameters).
- *Step 3.* **Stopping rule**:
  - Repeat Steps 1 and 2 until local searches are completed from starting points that use each of the N^* Sobol' points.
  - Return the point with the lowest function value from `wisdom.dat` as global minimum.

B.7 Gradients of Test Functions

Griewank Function The gradient of the Griewank function is equal to

$$\frac{\partial f}{\partial x_i} = \frac{2x_i}{a} + \left[\prod_{j=1, j\neq i}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)\right] \sin\left(x_i/\sqrt{i}\right) \frac{1}{\sqrt{i}}.$$

Levi Function The gradient of the Levi function is equal to

$$\nabla f = \begin{pmatrix} 6\pi\cos(3\pi x_1)\sin(3\pi x_1) + 2(x_1-1)(1+\sin^2(3\pi x_2)) \\ 2(x_i-1)(1+\sin^2(3\pi x_{i+1})) + 2(x_{i-1}-1)^2 6\pi \sin(3\pi x_i)\cos(3\pi x_i) \quad \text{for } i \notin \{1,n\} \\ 2(x_n-1)(1+\sin^2(2\pi x_n)) + (x_n-1)^2 4\pi\cos(2\pi x_n)\sin(2\pi x_n) + (x_{n-1}-1)^2 6\pi\cos(3\pi x_n)(3\pi x_n) \end{pmatrix}.$$

Rastrigin Function The gradient of the Rastrigin function is equal to

$$\frac{\partial f}{\partial x_i} = 2x_i + 20\pi\sin(2\pi x_i).$$

Rosenbrock Function The gradient of the Rosenbrock function is equal to

$$\nabla f = \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1-x_1) \\ 200(x_i - x_{i-1}^2) - 400x_i(x_{i+1} - x_i^2) - 2(1-x_i) \text{ for } i \notin \{1,n\} \\ 200(x_n - x_{n-1}^2)^2 \end{pmatrix}.$$

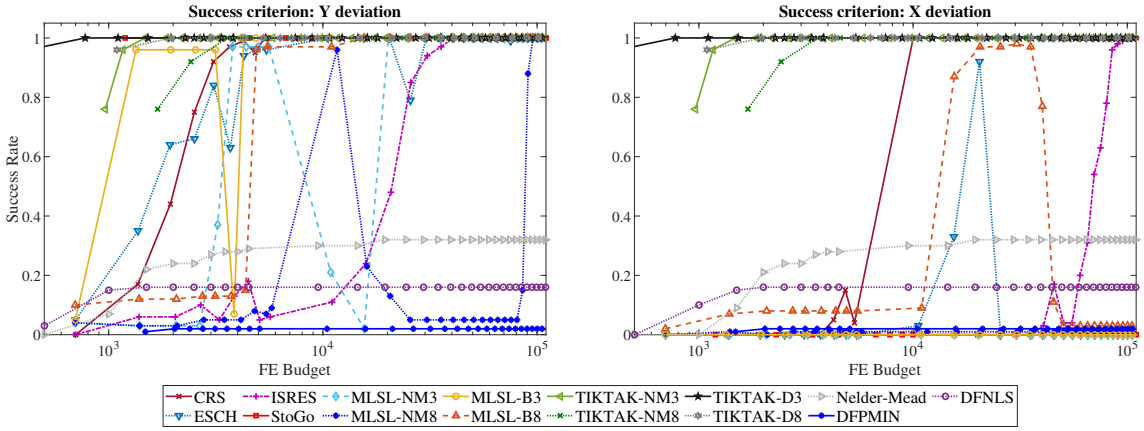
C Appendix: Additional Results on Data and Deviation Profiles

C.1 Complete Results for Levi and Rosenbrock Test Functions in 10 Dimensions

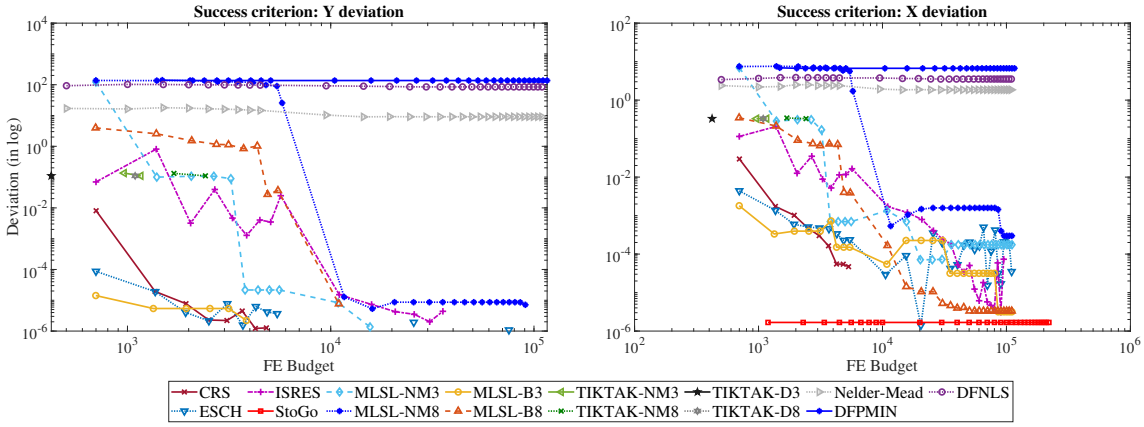
In this section, we provide the complete data and deviation profiles for the Levi and Rosenbrock test functions in ten dimensions. The corresponding figures that we presented in the main text (Figures 7 and 10) excluded selected algorithms for readability.

FIGURE C.2 – Data and Deviation Profiles: Levi, 10 Dimensions

(A) Data Profiles



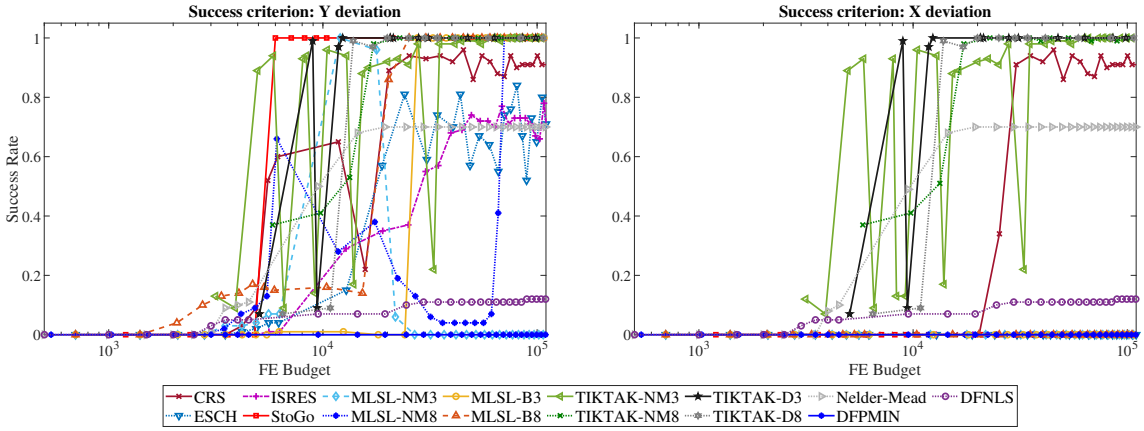
(B) Deviation Profiles



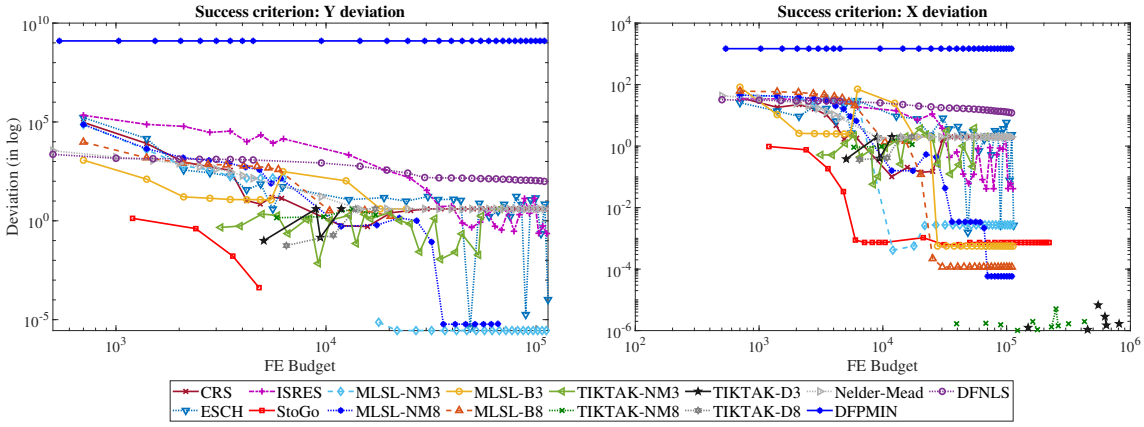
Notes: This figure is the same as 7 but showing all algorithms (while ESCH and the 4 MLSL versions were excluded in the previous figure for readability. This figure shows optimizers' performance in minimizing the Levi test function in 10 dimensions.

FIGURE C.3 – Data and Deviation Profiles: Rosenbrock Function, 10 Dimensions

(A) Data Profiles



(B) Deviation Profiles



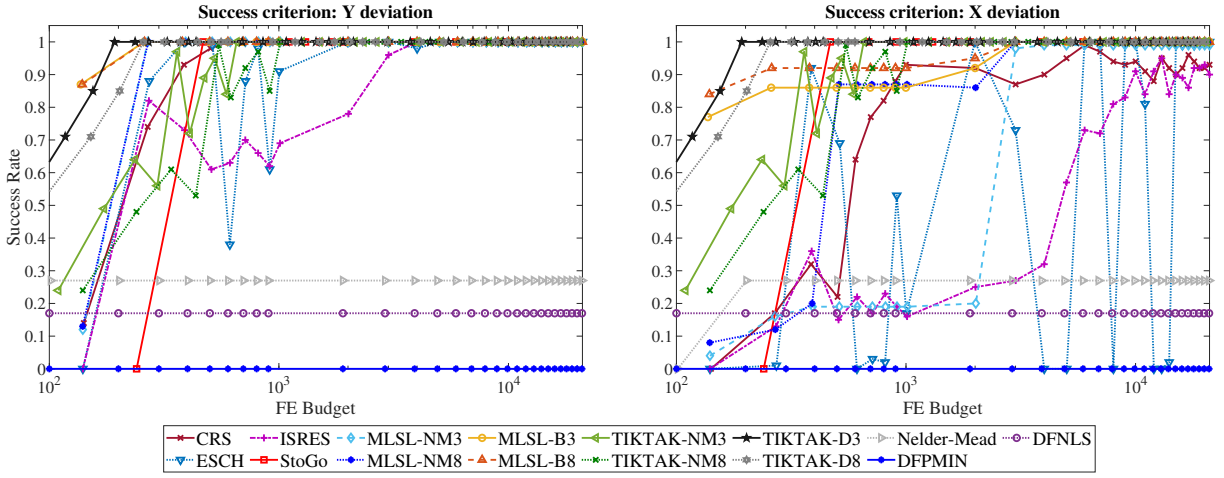
Notes: This figure is the same as 10 but showing all algorithms (while CRS, ESCH, MLSL-nm3, MLSL-nm8, TikTak-nm3, and Tiktak-d3 were excluded from the previous figure for readability. This figure shows optimizers' performance in minimizing the Rosenbrock test function in 10 dimensions.

C.2 Test Functions in 2 Dimensions

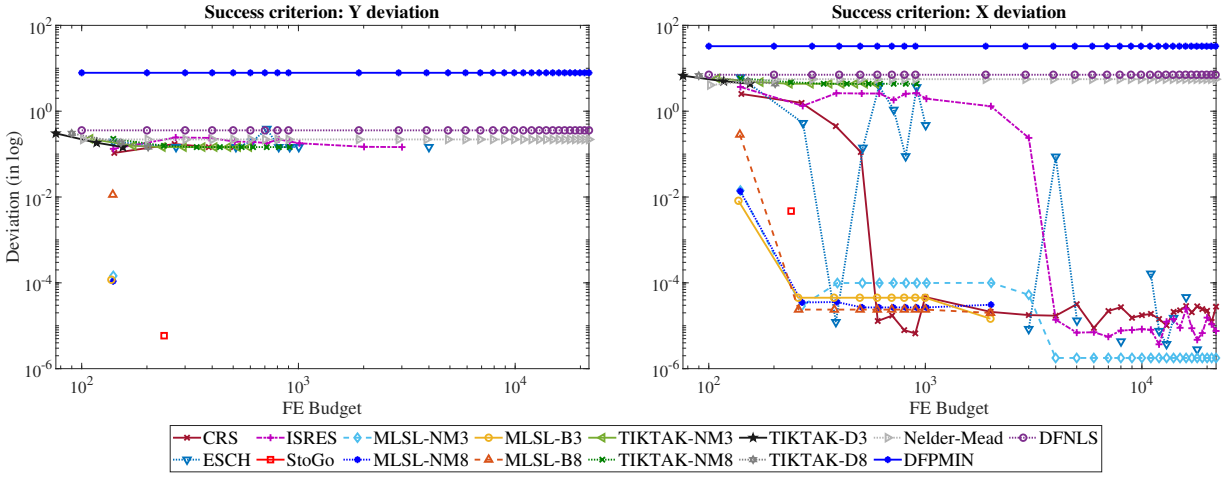
In this section, we provide the data and deviation profiles for each test function in two dimensions. For definitions and explanation of the figures, see Section 4.2. Note that the performance profiles in Section 4.3 include the two- and ten-dimensional test functions (as part of the 800 problems that are included in the full set of problems $p \in P$).

FIGURE C.4 – Data and Deviation Profiles: Griewank, 2 Dimensions

Panel A: Data Profile by Success Criteria



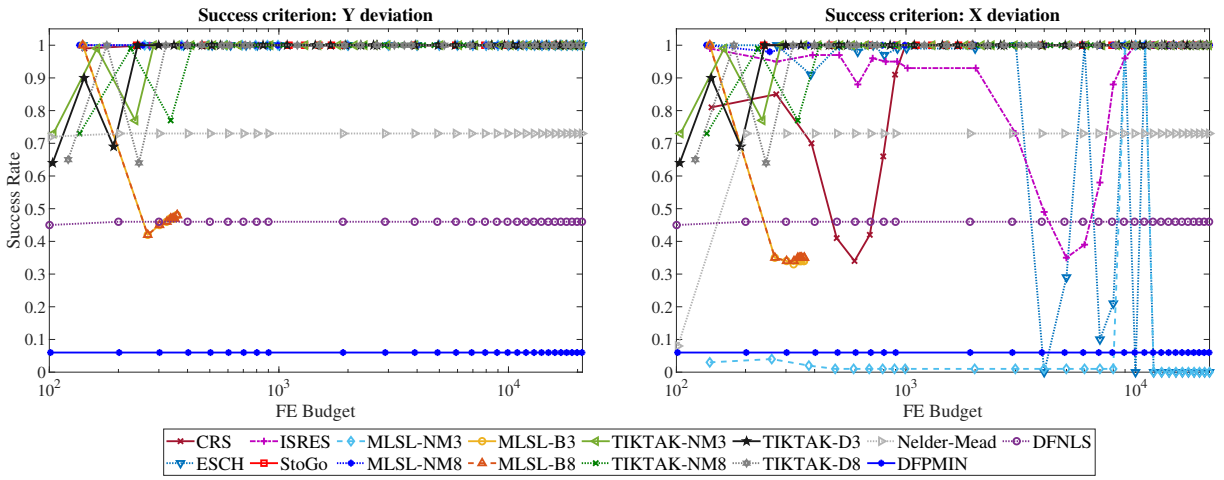
Panel B: Deviations by Success Criteria



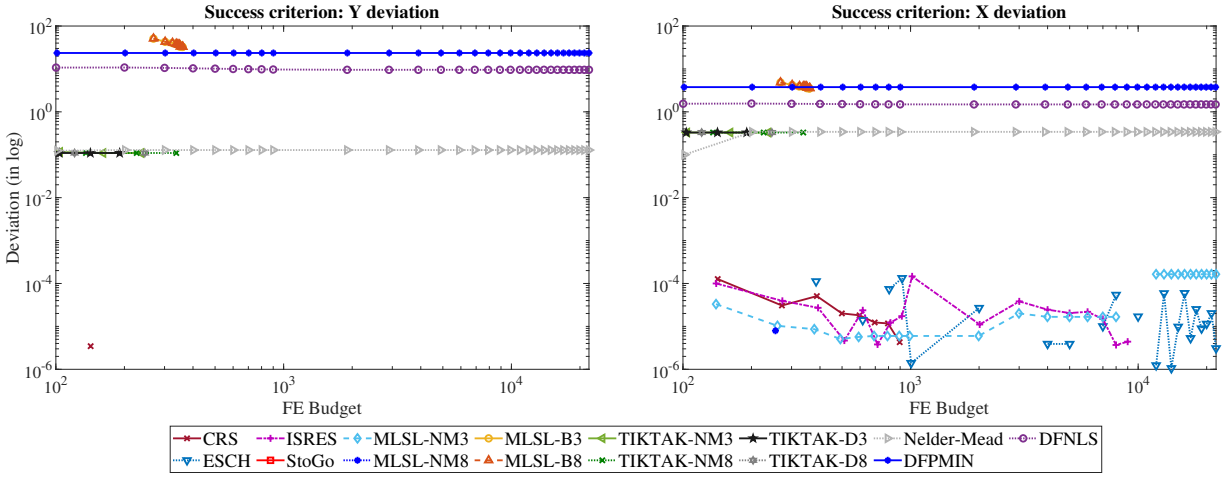
Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

FIGURE C.5 – Data and Deviation Profiles: Levi, 2 Dimensions

Panel A: Data Profile by Success Criteria



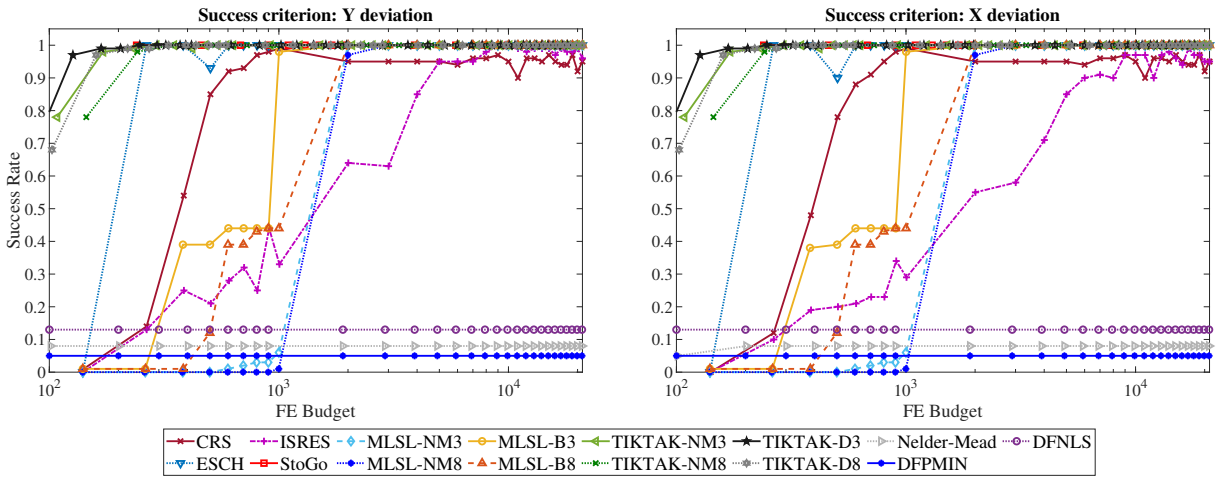
Panel B: Deviations by Success Criteria



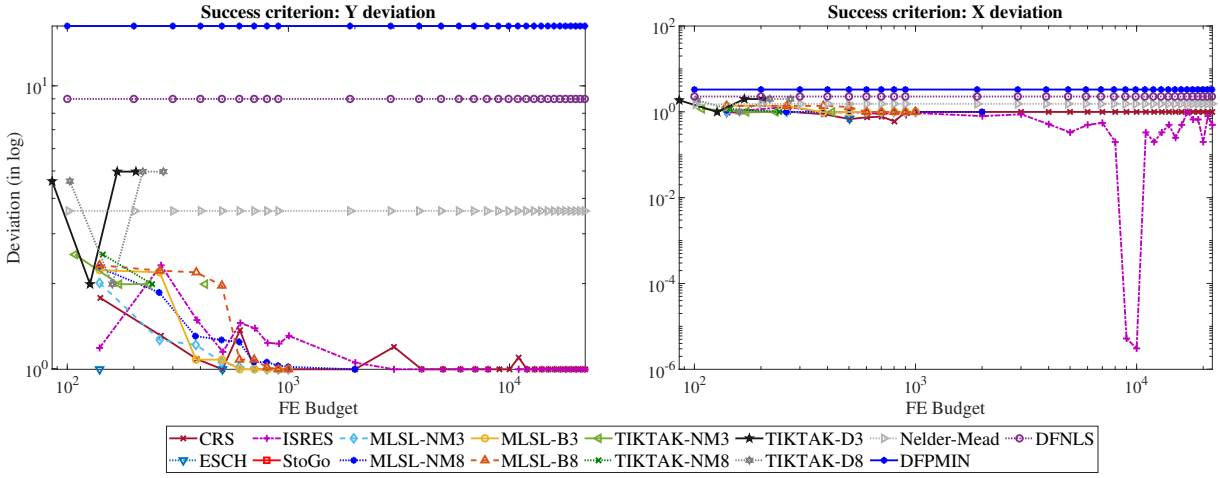
Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

FIGURE C.6 – Data and Deviation Profiles: Rastrigin, 2 Dimensions

Panel A: Data Profile by Success Criteria



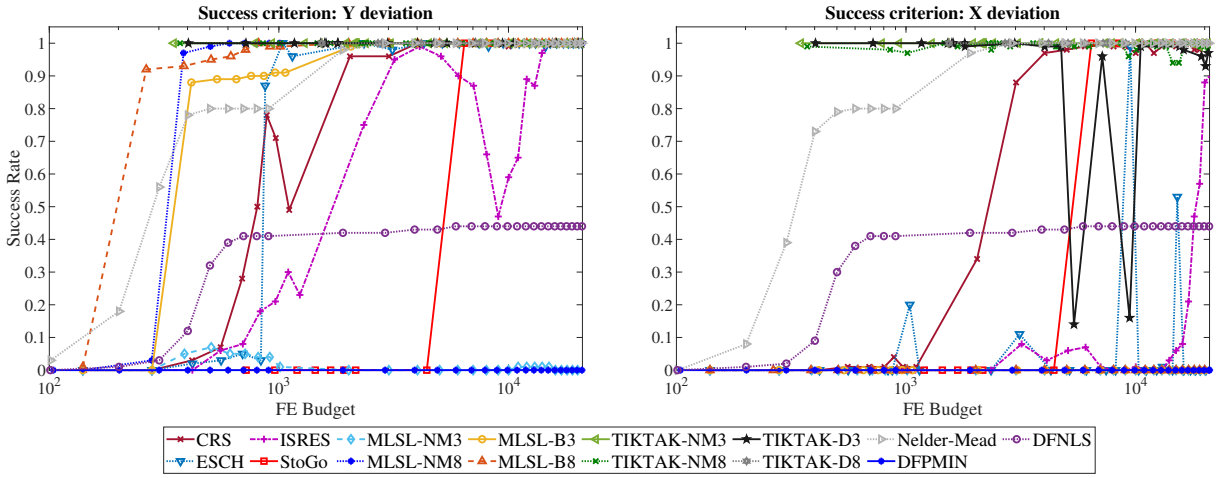
Panel B: Deviations by Success Criteria



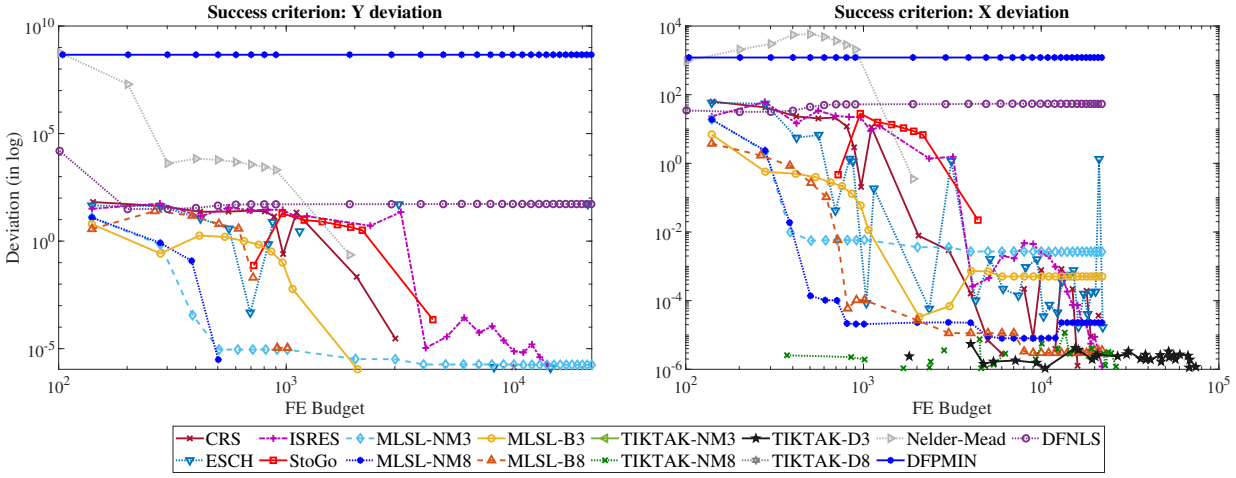
Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

FIGURE C.7 – Data and Deviation Profiles: Rosenbrock, 2 Dimensions

Panel A: Data Profile by Success Criteria



Panel B: Deviations by Success Criteria



Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

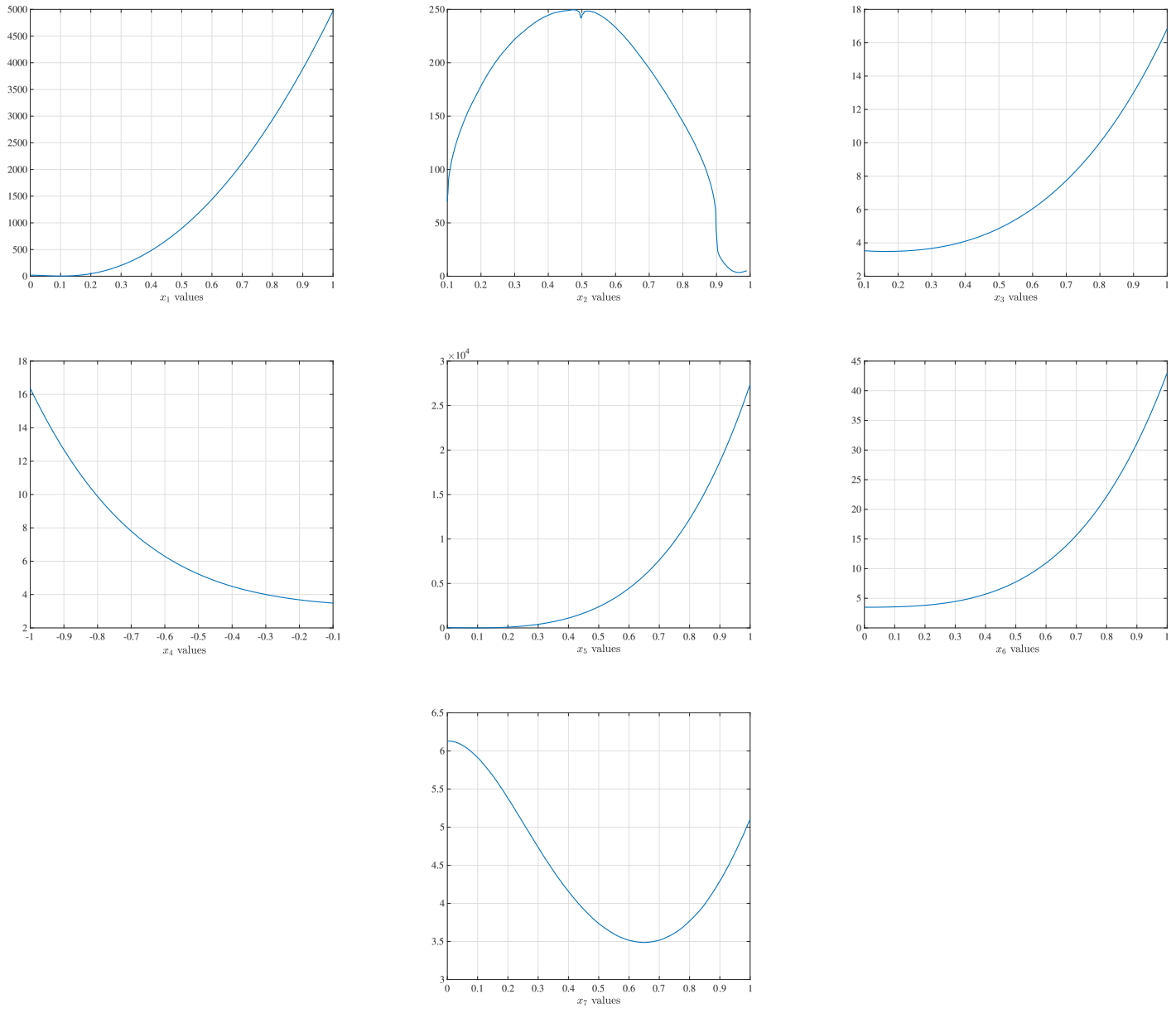
D Economic Application: Slices of the Objective Function Surface

TABLE C.1 – The True Parameter Values of the Income Process

Generic	Parameter	Description	Results
x_1	σ_ε	St. dev. of transitory income shock	0.102687
x_2	p_1	Weight of center of ζ distribution	0.965121
x_3	μ_2	Mean of right tail of ζ distribution	0.170472
x_4	μ_3	Mean of left tail of ζ distribution	–0.10
x_5	$\sigma_{1,\zeta}$	St. dev. of center of ζ distribution	0.09571
x_6	$\sigma_{2,\zeta}$	St. dev. of right tail of ζ distribution	0.005968
x_7	ϕ	Aggregate risk transmission parameter	0.643836

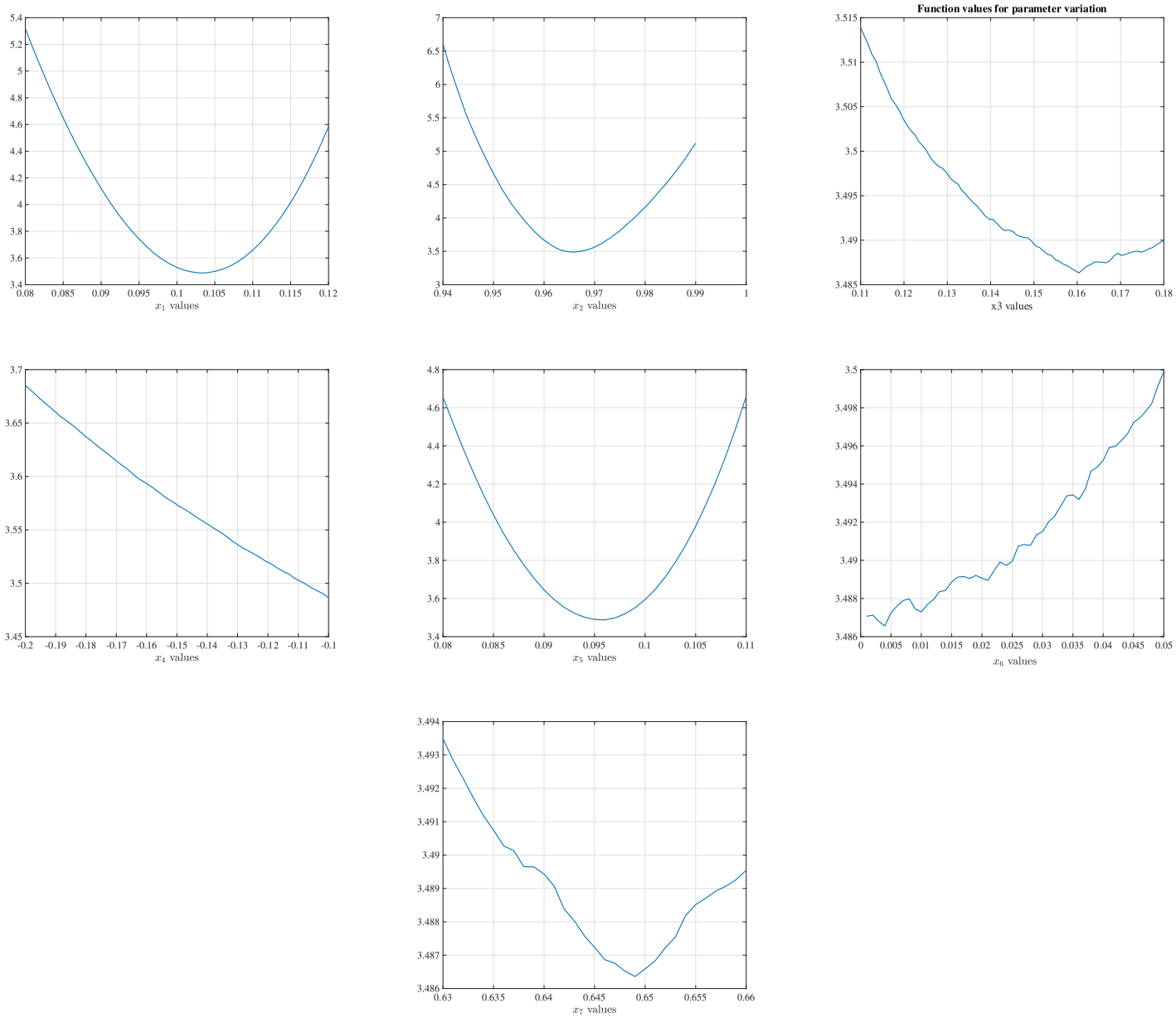
Notes: The parameters are estimated by matching data moments from Sweden. In this table, we report the parameter values that correspond to the smallest function value (equal to 3.4416262) that is found by any algorithm in our benchmarking exercise. We consider this point to be the "true" minimum to define whether minimizations are successful. The whole income process is pinned down by the seven parameters presented in this table. The standard deviations of the center and right tail of the ζ distribution are restricted to be equal, so that $\sigma_{2,\zeta} = \sigma_{3,\zeta}$. Furthermore, we restrict the weight of the right and left tail of the ζ distribution to be equal and the weights have to sum to 1, so that the knowledge of p_1 implies that $p_2 = p_3 = \frac{1-p_1}{2}$.

FIGURE D.8 – 1-Dimensional Slices of the Objective Surface



Note: Each panel plots a slice of the objective surface by varying the parameter value shown on the vertical axis while fixing the remaining six parameters at their global minimum value found. See Table C.1 for the parameter corresponding to each x value.

FIGURE D.9 – 1-Dimensional Slices of the Objective Surface: Zooming In



Notes: Each panel plots a slice of the objective surface by varying the parameter value shown on the vertical axis while fixing the remaining six parameters at their global minimum value found. Each plot is zoomed in to the immediate neighborhood of the true minimum.