NBER WORKING PAPER SERIES

BENCHMARKING GLOBAL OPTIMIZERS

Antoine Arnoud
Fatih Guvenen
Tatjana Kleineberg

Working Paper 26340
http://www.nber.org/papers/w26340

NATIONAL BUREAU OF ECONOMIC RESEARCH
1050 Massachusetts Avenue
Cambridge, MA 02138
October 2019

Special thanks to Anthony Smith, Serdar Ozkan, and Fatih Karahan for their collaboration with Guvenen on projects that employed earlier versions of TikTak and making significant contributions to the algorithm; to Michelle Rendall, Rocio Madera, David Domeij and Chris Busch for collaborations on subsequent projects that used TikTak; and to Arun Kandanchatha for writing the TikTak Fortran code used in this paper as a generic, stand-alone, user-friendly program. Leo Stanek provided outstanding research assistance. The views expressed herein are those of the authors and do not necessarily reflect the views of the National Bureau of Economic Research.

Benchmarking Global Optimizers
Antoine Arnoud, Fatih Guvenen, and Tatjana Kleineberg
NBER Working Paper No. 26340
October 2019
JEL No. C13,C15,C51,C53,C61,C63,D52,J31

ABSTRACT

We benchmark seven global optimization algorithms by comparing their performance on challenging multidimensional test functions as well as a method of simulated moments estimation of a panel data model of earnings dynamics. Five of the algorithms are taken 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) algorithm, (iv) Stochastic Global Optimization (StoGo), and (v) Evolutionary Strategy with Cauchy distribution (ESCH). The other two algorithms are versions of 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 Non-linear Least Squares (DFNLS) algorithm, and a popular variant of the Davidon-Fletcher-Powell (DFPMIN) algorithm. To give a detailed comparison of algorithms, we use a set of benchmarking tools recently developed in the applied mathematics literature. We find that the success rate of many optimizers vary dramatically with the characteristics of each problem and the computational budget that is available. Overall, TikTak is the strongest performer on both the math test functions and the economic application. The next-best performing optimizers are StoGo and CRS for the test functions and MLSL for the economic application.

Antoine Arnoud
Yale University
antoine.arnoud@yale.edu

Tatjana Kleineberg
The World Bank
tkleineberg@worldbank.org

Fatih Guvenen
Department of Economics
University of Minnesota
4-101 Hanson Hall 1925
Fourth Street South
Minneapolis, MN, 55455
and NBER
fatihguvenen@gmail.com

1 Introduction

We benchmark the performance of seven global and three local optimizers by applying them to several difficult multidimensional objective functions. We first apply the algorithms to four test functions—named Levi, Griewank, Rastrigin, and Rosenbrock—which 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 were found to be particularly challenging to optimize.[1]

However, the characteristics of these test functions can differ substantially from the calibration or estimation problems commonly encountered in economics. For example, these test functions are continuous and differentiable (despite having many local minima), which is not necessarily the case in economic applications where objective functions are often based on moments that are based on data simulated from the numerical solution of a complex model. Because of truncation and other approximation errors present in numerical solutions, as well as the economic features inherent in 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.

With these considerations in mind, we also assess the performance of these optimizers using 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 7 parameters to estimate. This problem is not one of the more complex ones we could have chosen, and this choice is intentional. Our goal is to show that even such a relatively benign looking optimization problem commonly encountered in economics can be very challenging for many global optimizers.

Five of the seven 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) the Controlled Random Search algorithm with Local Mutation (CRS), (ii) the Improved Stochastic Ranking Evolution Strategy (ISRES) algorithm, (iii) the Multi-Level Single-Linkage (MLSL) algorithm, (iv) the Stochastic Global Optimization (StoGo) algorithm, and (v) the Evolutionary Strategy algorithm 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

The remaining two optimizers are variants of a newer global optimization algorithm, named TikTak, which was developed by one of us (Guvenen) and refined with coauthors through applications to various estimation/calibration problems.[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 broad exploration of the (parameter) space and uses the information it accumulates to increasingly focus the search on the most promising region. The two variants we benchmark differ only in the local optimization routine they use: the Nelder-Mead downhill simplex algorithm ("TikTak-nm") or the the Derivative-Free Nonlinear Least Squares (DFNLS) algorithm of Zhang et al. (2010) ("TikTak-d").

It is fairly common for researchers to use a local optimizer alone (not as part of a global algorithm) even in large-scale estimation and calibration problems. 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 algorithms in the benchmarking analysis. These are Nelder-Mead and DFNLS mentioned above, and a popular quasi-Newton based optimizer called DFPMIN taken from Press et al. (1996).[5]

One notion that we have used so far without defining is the "performance" of an optimizer. In practice, there are at least four practical considerations. The first consideration, and arguably the most important, is an optimizer's reliability—or the likelihood that it will find the global optimum in the problem that a researcher faces. A proxy for

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, but all other algorithms are derivative-free.

[4] See Guvenen (2011) for a description of an early version of the algorithm. 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. It was also 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] More precisely, DFPMIN implements the Broyden-Fletcher-Goldfarb-Shanno variant of the Davidon-Fletcher-Powell minimization algorithm. For details, see Chapter 10.7 of Press et al. (1992).

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 (e.g., measured in time or number of function evaluations). This trade-off is captured by a "data profile," introduced by Moré and Wild (2009), which plots the success rate of an optimizer as a function of the computational budget.

A third consideration is how an optimizer's speed compares to those of the other available optimizers for different test problems. In other words, we would like to know the fraction of test 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 least two times (or three, four, and so on) slower than the fastest optimizer for each problem. A very useful plot introduced by Dolan and Moré (2002) and called "performance profiles" provides exactly this information, and we use it in our benchmarking analysis (see Ali et al. (2005) and Zhang et al. (2010)).

To consider a minimization "successful," we focus on two different metrics: the distance either between the *function values* of the returned and true minima or 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 a particular 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 shall refer to as, "deviation profiles," which is analogous to the data profiles, but rather than the success rate it reports the average of the distance measure over all unsuccessful implementations for a given algorithm at different computational budgets.

Using these three sets of benchmarking tools, we find that overall TikTak-d has the strongest performance on both the test functions and the economic application—in terms of both reliability and speed. The second-best optimizer is TikTak-nm, which performs well on the test functions and on the income process for most but not all success criteria. In addition, TikTak-nm is less efficient than TikTak-d as it requires a larger computational budget. The relative performance of the NLopt algorithms varies across different test functions and the economic application. MLSL and ISRES perform

better on the economic application but are relatively less successful in minimizing the test functions. On the other hand, CRS performs better on the test functions but is less reliable on the economic application. StoGo performs well on the test functions.[6] 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 finally by DFPMIN.

The benchmarking results here are based on running each optimizer on a single CPU core. However, an appealing feature of TikTak is that it can be run in parallel very easily and without needing any special software or requiring any knowledge of parallel programming (such as MPI, OpenMP, CUDA, and so on). Specifically, the optimization routine can distribute its computations (i) across multiple CPUs (ii) that reside in different machines possibly located in different physical locations, (iii) which could be running different operating systems and/or different compilers of the same language. Furthermore, in large scale high-dimensional problems, the speed gains (scaling) can be close to linear with the number of CPU cores.[7] We leave a fuller exploration of TikTak's parallel performance for future research.

As with any study of this kind, the conclusions we draw from our benchmarking analysis inevitably depend on some of the choices we made in the implementation. We have made every effort to minimize the choices we needed to make and relied on the options chosen by the most popular implementations whenever possible. Having said that, there is always a chance that a different assumption can alter some of our conclusions. We hope that these results spur further work that confirms, qualifies, or—if warranted—modifies the conclusions of our analysis.

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

[6] We do not use StoGo on the economic application because it is the only optimizer that uses the gradient. The analytical gradient is unknown in most economic applications, and numerically computing the gradient can be difficult and computationally demanding.

[7] In the previous applications cited in footnote 4, the algorithm has been 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.

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 crucially depends 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 to the literature. First, we benchmark the performance of 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 both text 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 (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.

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. We therefore omit their descriptions here for brevity but discuss them in more detail in Appendix A.[8] TikTak is a new algorithm that is not well known, so we describe it in some detail here as well as in Appendix A.

[8]Note that we also use either Nelder-Mead or DFNLS in the local stage (i.e., local searches) of TikTak. Nelder-Mead is also used in the local stage of MLSL. 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 (see Section 3.3).

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.[9] In particular, TikTak uses the Sobol' sequence (Sobol' (1967)), which has several desirable properties and is known to perform particularly well in high dimensions.[10]

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 of the problem, whereas N^* is much smaller, for example, one to ten percent 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

[9] 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.

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

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 parameters 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 each other. This is the basic idea of the TikTak algorithm; additional details are in Appendix A.

In the benchmarking analysis, we use four different variants of TikTak, which differ in the way in which they implement local searches. In particular, local searches of TikTak use either the Nelder-Mead or the DFNLS algorithm.[11] In addition, an 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, and a lower tolerance implies the opposite trade-off (exhaustive but slow). To explore these trade-offs, we consider two tolerance levels, 10^{-3} and 10^{-8}, for each local optimization algorithm.[12,13] 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

[11] These local algorithms are described in more detail in the Appendix.

[12] As one could conjecture, the ideal approach would be to 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 to further improve the performance of TikTak.

[13] 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)).

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. For the standard test functions, the true global minima are known. However, we do not know the true minimum of the objective function of 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.[14] 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 $\hat{y}_{p,s}^* = f(\hat{\mathbf{x}}_{p,s}^*)$ is the corresponding function value. We define two different success criteria, one based on discrepancy in function values, 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, it is said to solve the problem successfully according to the X-val criterion if the maximum discrepancy across all 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 two tools—data profiles and performance profiles—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 a solver success that are solved successfully (for a given success criterion) for a given computational budget—that is, the number of function evaluations or FEs, γ. Specifically, first define the performance measure, $t_{p,s} > 0$, as the number of FEs that optimizer s needs

[14] Note that we impose an upper bound on the considered number of function evaluations (FEs), which is necessary because it can be very costly to evaluate the objective function.

to solve problem p successfully. Higher values of $t_{p,s}$ imply worse performance, and if the optimizer is not able to solve at any budget, we set $t_{p,s} = \infty$. Next, define the "success rate" of the optimizer for a given 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 optimizers' 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, first define the *performance ratio* for solver s and problem p as

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

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, which naturally has $r_{p,s} = 1$. 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

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

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 that optimizer for which optimizer s is the fastest (i.e., solves the problem with the smallest number of FEs among all optimizers considered). More generally, $\rho_s(\alpha)$ is the cumulative distribution function of $r_{p,s}$, showing the probability that solver s is within α factor of the best solver for problem s; hence, for a given α higher values of $\rho_s(\alpha)$ means 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 the cluster (unix machines) at Yale University.

For all global algorithms, we implement a "polishing" local search as the very last step of the optimization routine. To do this, we implement a local search with DFNLS from the best (smallest) minimum that was found so far by the global algorithm. We set the stopping tolerance of the last local search with DFNLS to 10^{-8} to give all global algorithms the chance to come as close as possible to the true minimum.[15]

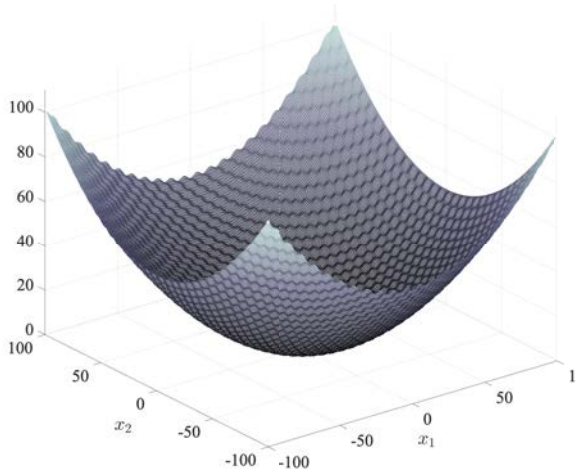
4 Benchmarking Results for Standard Test Functions

For the benchmarking analysis, we select four test functions—named 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. They are well defined for any number of dimensions. In Section 4.2, where we benchmark performance via data profiles, we use 10-dimensional versions of each function. (In Appendix B, we also report the results for the 2-dimensional case). In Section 4.3, where we analyze performance profiles, we use both 2- and 10-dimensional versions of each function.

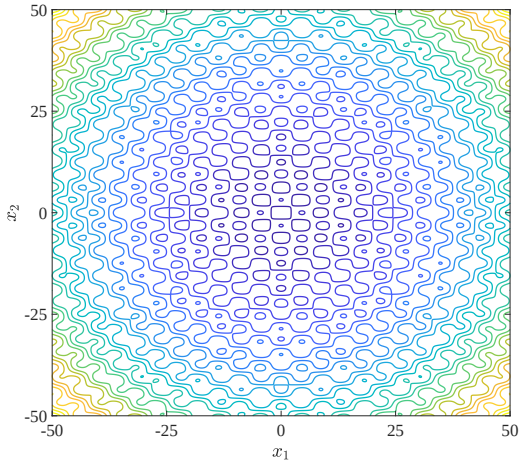
[15]The last polishing search is useful as we sometimes use lower tolerances as stopping criteria in the local stages (i.e. 10^{-3}) to increase the speed of the global algorithm. It is therefore possible that the global algorithm comes into the close neighborhood of the true minimum, but it might require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

FIGURE 1 – Griewank Function

(A) 3-D Plot



(B) Contour Map



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" and therefore has a very flat surface near the global minimum.

Griewank function. The Griewank function in n dimensions is

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

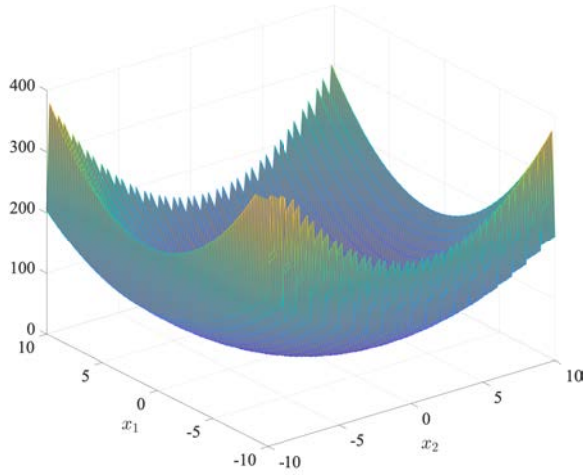
where we use the conventional choice of $a = 200$. We will focus on the hypercube domain $x \in [-100, 100]^n$. Its global minimum is at $x = (0, ..., 0)$ with function value $f(0, ..., 0) = 1$.

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 gives 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.[16] Furthermore, each closed circle on the map contains at least one local minimum, of which there are many. Of course, keep in mind that these are the challenges that are already evident in 2

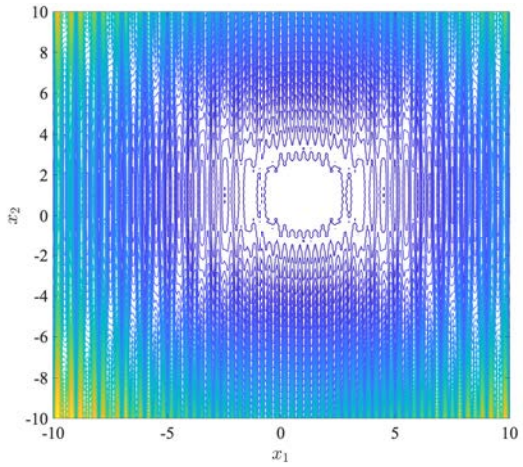
[16]To make the details visible, the contour map is plotted on a smaller domain: $x \in [-50, 50]^2$.

FIGURE 2 – Levi No. 13 Function

(A) 3-D Plot



(B) Contour Map



dimensions; how these functions look and what further complications they may introduce in three or more dimensions are impossible to visualize or even imagine.

Levi No. 13 function. The Levi function has several variants, and the one we use here is a version—called Levi No. 13—that is commonly used for benchmarking optimizers. In n dimensions, it is given by

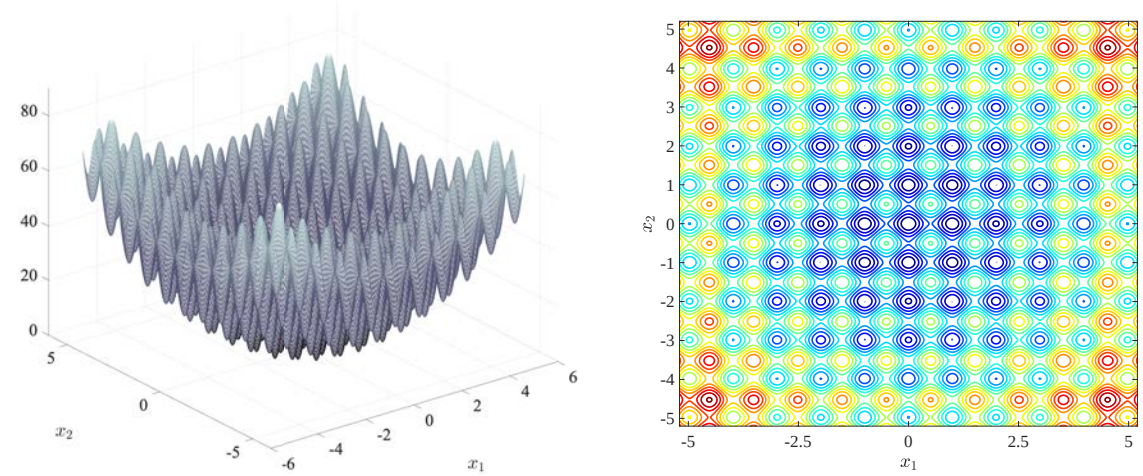
$$f(x) = \sin^2(3\pi x_1) + (x_n - 1)^2[1 + \sin^2(2\pi x_n)] + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + 1,$$

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 that is easily seen here: 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.

FIGURE 3 – Rastrigin Function

(A) 3-D Plot (B) Contour Map



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 (left panel) as well as the contour map (right panel). In some ways, Rastrigin combines the challenging features of 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

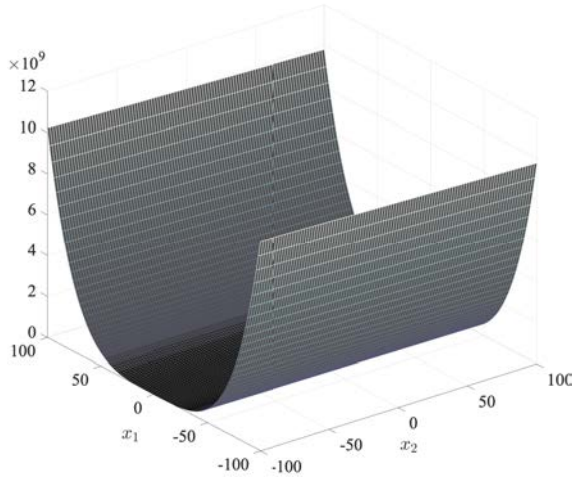
$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] + 1,$$

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

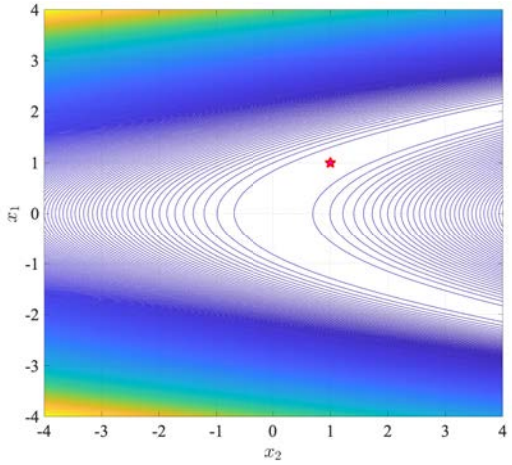
Figure 4 plots the function values and contour map. Unlike the previous functions we have seen, the challenge of Rosenbrock is not the proliferation of local minima but

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.

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, which is where the contour map is 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. Later, in Figure 9, we plot 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 in this section. The five global optimizers taken from the NLopt library are CRS, ISRES, ESCH, StoGo, and MLSL. For the local stage of MLSL, we use the Nelder-Mead algorithm with two different convergence criteria—10^{-3} and 10^{-8}—and we refer to these versions as MLSL3 and MLSL8, respectively. 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 convergence criteria. We refer to the four versions as TikTak-nm3, TikTak-nm8, TikTak-d3, and TikTak-d8, respectively. The three local optimization routines are the Nelder-Mead simplex algorithm, DFPMIN, and DFNLS.

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. To construct the data profile for each test function and each optimizer, we implement all minimizations at 30 different computational budgets. We then trace the data profiles by linearly interpolating between these data points.[17] For the 10-dimensional test functions, we set an upper bound of 100k FEs.[18] As mentioned above, we consider two success criteria: one based on F-val and the other based on X-val, and report them both. For the test functions, we set the tolerance level for success at $\tau = 10^{-6}$. We will report both data and deviation profiles for each optimizer. Because a deviation profile is only meaningful for failed searches, no values are plotted for an optimizer at FEs where the success rate equals 100%. Figures 5-10 present the data profiles (top panel) and deviation profiles (bottom) for each test function. The left and right panels report the results under the F-val and X-val criteria, respectively. This section documents the results for the 10-dimensional test functions. Appendix B shows the results for the 2-dimensional test functions.

4.2.1 Griewank Function

The first main result from the top panel of Figure 5 is that the TikTak algorithm has the strongest overall performance under both success criteria. For example, in the left panel, it reaches a success rate of 100%—at budgets above 460 FEs for TikTak-d3 and above 710 FEs for TikTak-d8. The data profiles are very steep and both versions of TikTak-d reach a success rate of 93% very quickly—at 270 for TikTak-d3 and at 400 for TikTak-d8. The success rate reaches 100% above 1,300 FEs for TikTak-nm3 and above 2,600 for TikTak-nm8. It performs similarly well under the X-val criterion.

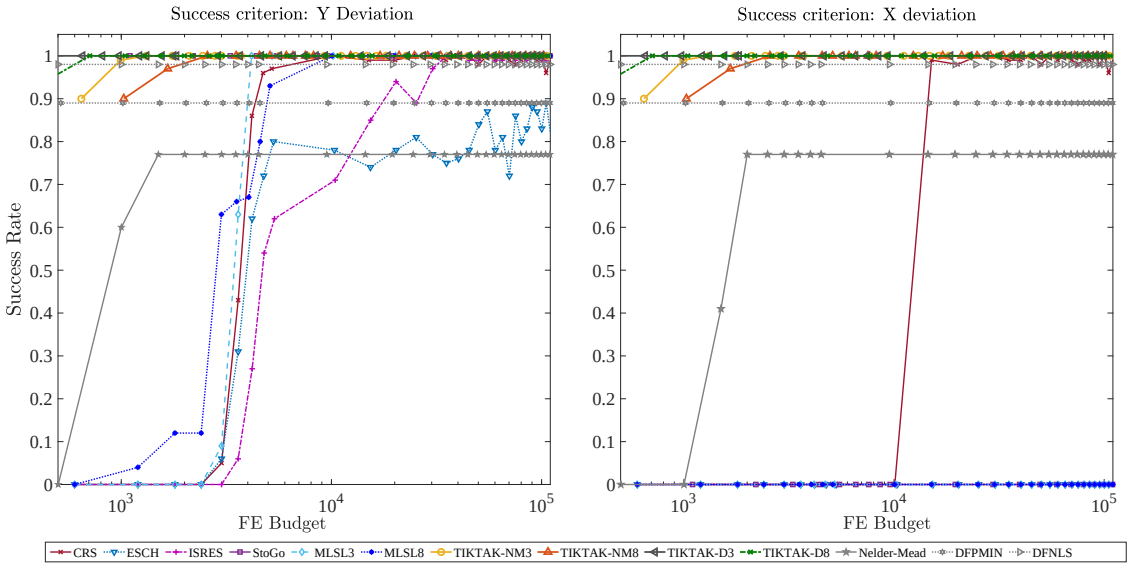
The second-best algorithm is CRS, which reaches high success rates but only at higher computational budgets. CRS reaches F-val success rate of 96% at 4,700 FEs and near 100% above 9,000 FEs. Similarly, the X-val success rates are between 98% and 100% at

[17] For the NLopt algorithms, we can specify the number of FEs as explicit stopping criteria. For TikTak, the number of FEs cannot be directly used as stopping criteria. Instead we 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 that are used by each optimizer and at each computational budget across all 100 problems that are minimized. These averages are then plotted on the x-axis of the data profiles.

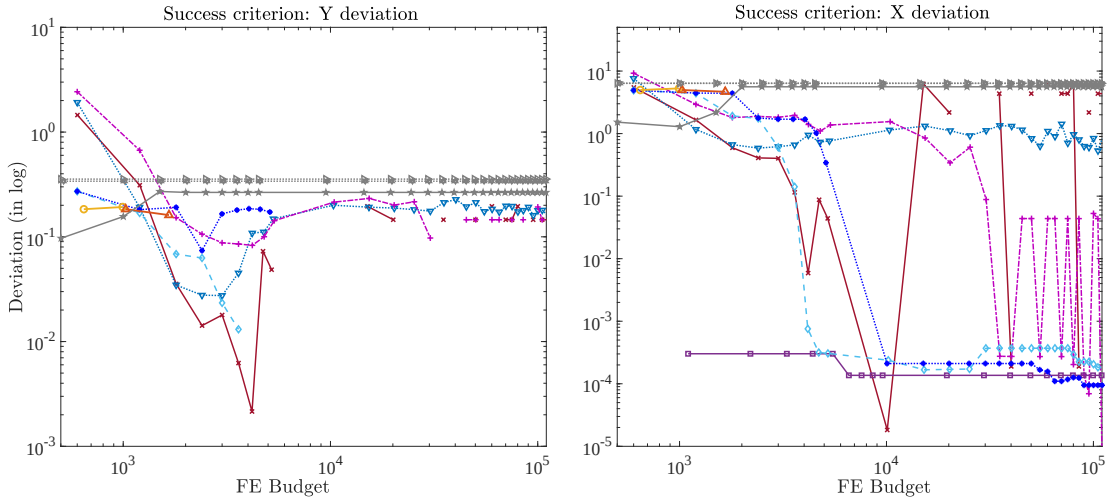
[18] For the 2-dimensional test functions, documented in Appendix B, we allow up to 20k FEs.

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.

budgets above 15,000. Notice that these are twice as many FEs as the slowest TikTak algorithm (-nm8) and 10 times more than TikTak-d3. The performance of StoGo is good as well but requires a closer inspection. Under the F-val criterion, it attains a 100% success rate with 1,100 FEs and higher, only bested by TikTak-d versions. However, it never attains the required tolerance level ($\tau = 10^{-6}$) under the X-val criterion for any budget. That said, when it fails, the deviation profile (bottom left panel) shows that the deviations are generally small, around 10^{-4}. To the extent that this is acceptable in a given application, its success would rank as the second best, behind TikTak.

Both ISRES and MLSL (both versions) require at least 10,000 FEs or more to reach F-val success rates of 100%, similar to CRS. Under X-val, neither one successfully solves a problem. Both versions of MLSL have small deviations (when they fail) from the true parameter values (around 10^{-4}) at budgets above 10k. The same is not true for ISRES: for failed implementations, the deviations vary substantially (between 1 and 10^{-4}), especially at large budgets. Finally, ESCH performs least well among the global algorithms. The F-val success rate is low at small budgets and oscillates substantially as the budget is increased, but never exceeds 88%. Under X-val, it is never successful, and deviations among failed implementations are large.

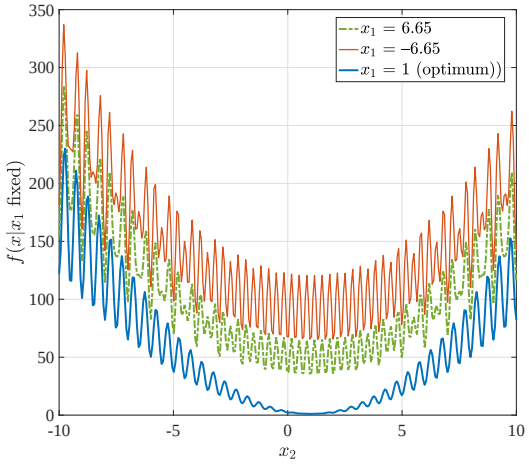
Turning to the local algorithms, they all perform similarly under both success criteria. The local DFNLS algorithm performs very well and reaches a success rate of 98% at all considered budgets. It therefore performs better than many global algorithms. DFPMIN reaches a success rate of 89% at all budgets. Although a success rate of 89% may seem high, DFPMIN will not find the global minimum from 11% of the starting points. Furthermore, because of their local nature, neither one improves its success rate with more FEs. The Nelder-Mead algorithm performs less well with a success rate that stagnates at 77%. Average deviations among failed implementations are very large for all local algorithms (Panel B). It turns out that Griewank is one of the best test functions for local algorithms. As we shall see in a moment, their performance is substantially lower for the other test functions as well as for the economic application.

4.2.2 Levi Function

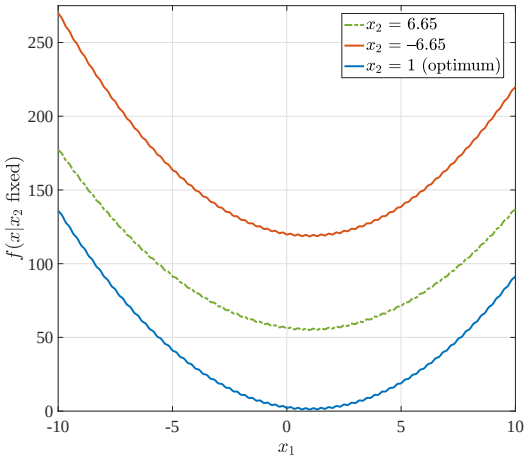
For the Levi function, Figure 7 shows the data and deviation profiles. As before, TikTak performs best for both criteria, and all versions of TikTak reach success rates of 100% already at low computational budgets. The fastest version is TikTak-d3, which reaches that level at 776 FEs, followed by TikTak-nm3 at 1.5k FEs. TikTak-d8 and -nm8 reach the same level at 1.5k and 3.5k FEs, respectively. CRS is the second-best performing algorithm but requires substantially higher budgets: it has no F-val success

FIGURE 6 – Levi No. 13 Function, Slices

(A) Values along x_2 dimension



(B) Values along x_1 dimension



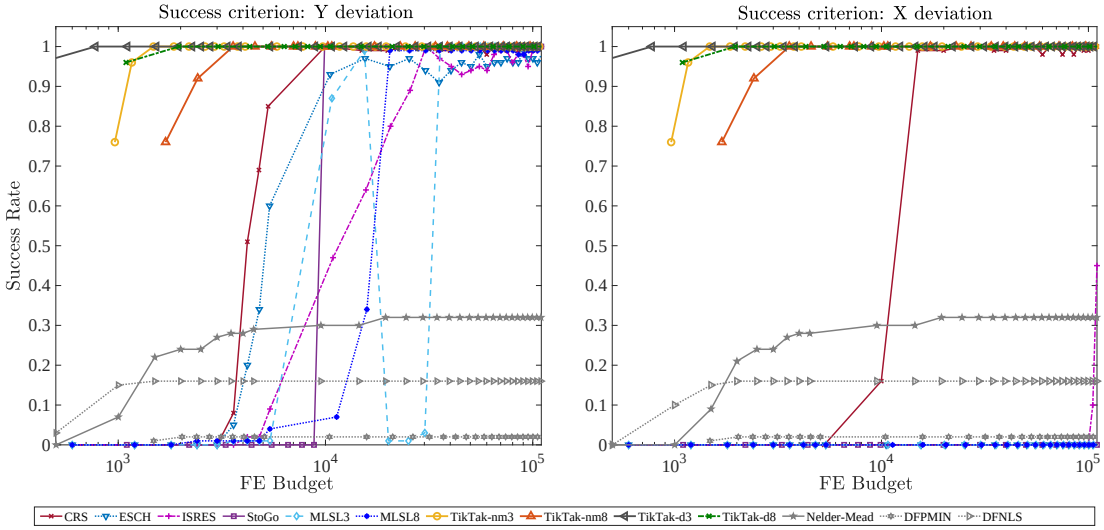
until 3k FEs, but then gradually improves to reach success rates of 99% or higher with budgets over 10k FEs. CRS has no X-val success at low budgets, only a 16% success rate at 10k, before reaching values between 98% and 100% above 15k FEs. StoGo reaches an F-val success rate of 100% at budgets above 10k. The F-val success rate below 10k is zero, but deviations of failed implementations are decreasing toward the success threshold of 10^{-6} as the number of FEs increases (bottom right figure). Under X-val, StoGo is never successful, but deviations among failed implementations decrease and remain at values around 10^{-4} above 10k FEs. Notice that the performance of these three optimizers for the Levi function are quite similar to what we saw for Griewank above.

ESCH, ISRES, and MLSL have F-val success rates that fluctuate, sometimes even at larger budgets. MLSL3, for example, reaches F-val success rates between 87% and 99% between 11k and 15k FEs, but then the success rate drops again sharply to 1%–3% at budgets between 20k–30k before reaching a stable success rate of 100% above 35k. MLSL8 reaches an F-val success rate of 98%–99% at budgets above 21k. ESCH and ISRES reach high F-val success rates, but these fluctuate moderately, even at larger budgets.[19] Deviations among failed implementations vary for ESCH, ISRES, and MLSL (left graph of Panel B). Failed implementations of MLSL have large deviations from the true function values at all budgets. ISRES has large deviations at most budgets, with two exceptions where deviations decrease to values close to 10^{-6}. ESCH has very

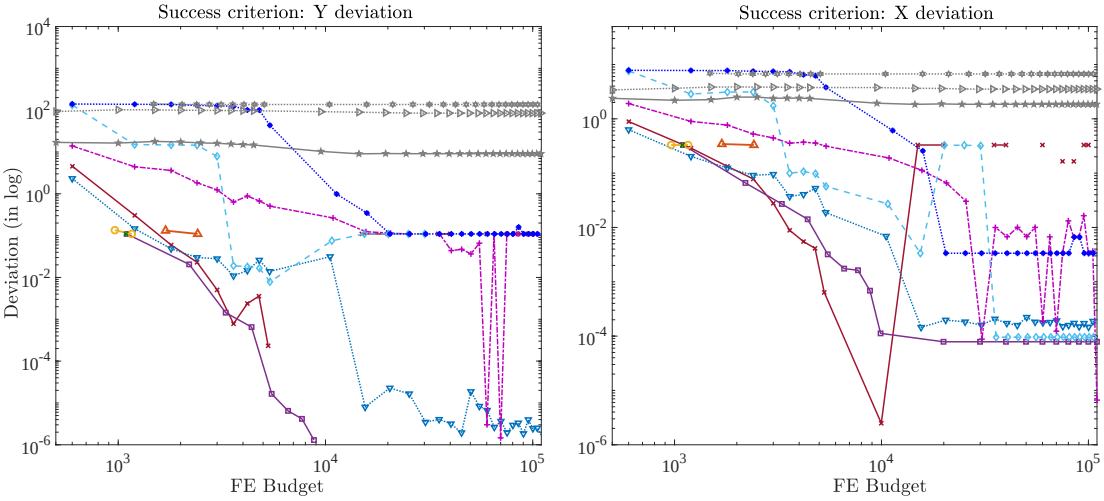
[19]For ESCH, the F-val success rate fluctuates between 91% and 99% at budgets above 11k. For ISRES, it fluctuates between 93% and 100% at budgets above 30k.

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

(A) Data Profiles



(B) Deviation Profiles



Notes: 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.

small deviations (between 10^{-4} and 10^{-6}) at budgets above 15k, implying that failed implementations of ESCH only miss the F-val success criterion by a small margin.

Under X-val, ESCH, ISRES, and MLSL are never successful at the computational budgets that we consider. However, deviations from the true parameters among failed

implementations are small (bottom right figure). For ESCH, deviations are close to 10^{-4} at budgets above 15k. For MLSL3, deviations are around 10^{-4} at budgets above 35k. For MLSL8, deviations are larger with values around 10^{-2} at budgets above 21k. For ISRES, deviations fluctuate between 10^{-2} and 10^{-4} at budgets above 30k.

The three local algorithms again perform poorly under both success criteria. Nelder-Mead stagnates at a 32% success rate at budgets above 19.5k for both criteria. DFNLS reaches 16% at budgets above 1.5k and 2k for F-val and X-val criteria, respectively. DFPMIN never exceeds a success rate of 2%. Average deviations among failed implementations are very large in every case.

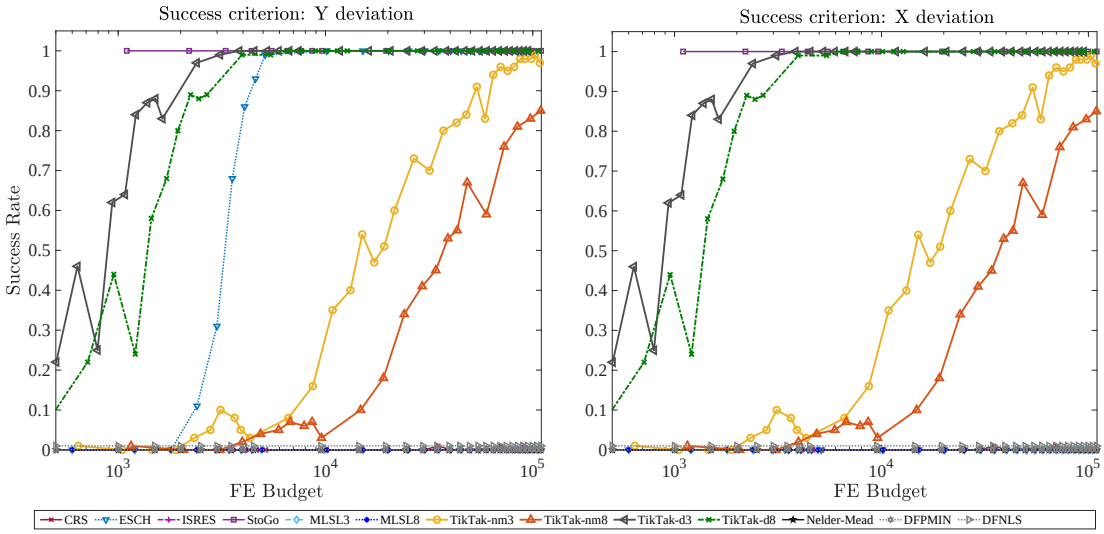
4.2.3 Rastrigin Function

Rastrigin proves to be much more challenging for optimizers than the previous two functions. While four of the optimizers reach an F-val success rate of 100% at 10k FEs, two others are between 10% and 30%, while the remaining six have less than a 2% success rate. Overall, StoGo has the best performance (Figure 8); it reaches F-val and X-val success rates of 100% for budgets above 1.1k FEs (which happens to be the lowest budget that is feasible for the optimizer). TikTak-d3 is the second best, reaching a 100% success rate (both F-val and X-val) above 3.8k FEs.[20] It is closely followed by TikTak-d8, which has a lower success rate at budgets below 4k, but reaches 100% around the same point as the -d3 version. TikTak-nm3 and -nm8 perform very poorly up to about 10k FEs but improve monotonically beyond that point. TikTak-nm3 reaches a 98% success rate above 65k FEs, whereas the -nm8 version reaches only an 83% rate at 100k FEs.

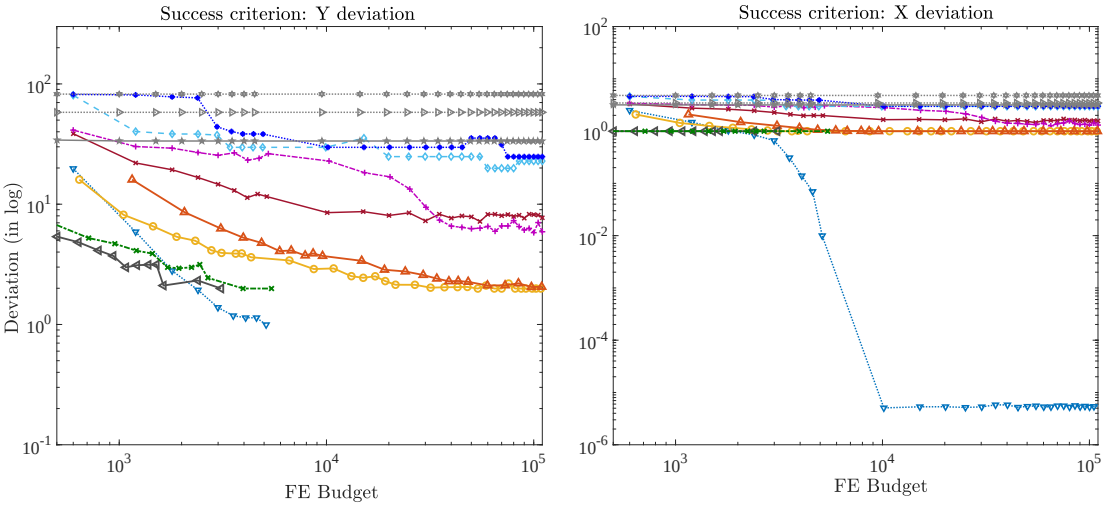
ESCH arguably ranks fourth in performance, only behind StoGo and the two versions of TikTak-d. It reaches an F-val success rate of 99% at 5k FEs and 100% for all larger budgets. Although its X-val success rate is zero, as we have seen for some other optimizers before, the deviations are consistently small, around $10^{-5.3}$ (bottom right figure). The remaining optimizers—CRS, ISRES, MLSL, and all three local algorithms Nelder-Mead, DFNLS, and DFPMIN—perform very poorly and have no F-val or X-val success at any of the computational budgets that we consider. In addition, failed implementations have large deviations that go all the way up to 10^2 and 10 for F-val and X-val values, respectively (bottom left figure). Overall, the deeply buried (many) local minima featured by Rastrigin proves to be too much to overcome for many of the global optimizers.

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.

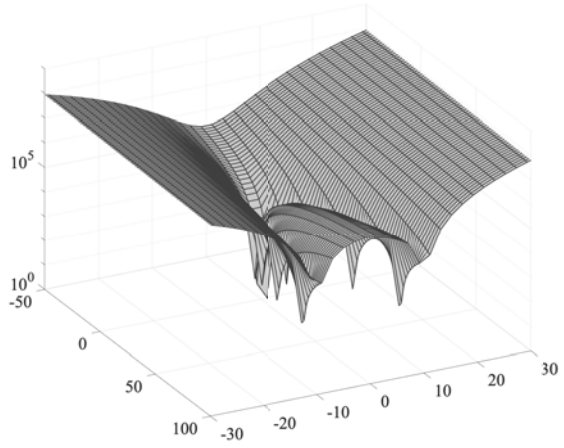
4.2.4 Rosenbrock Function

In many ways, Rosenbrock provides the toughest test to optimizers. Even the best optimizers require large computational budgets to find the global optimum (Figure 10).

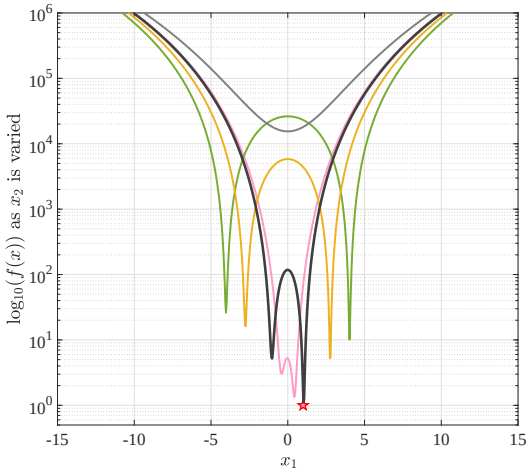
[20] At lower budgets, TikTak-d3 reaches an F-val success rate of 7% at 350 and success rates between 83% and 99% at budgets between 1.2k and 3k.

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



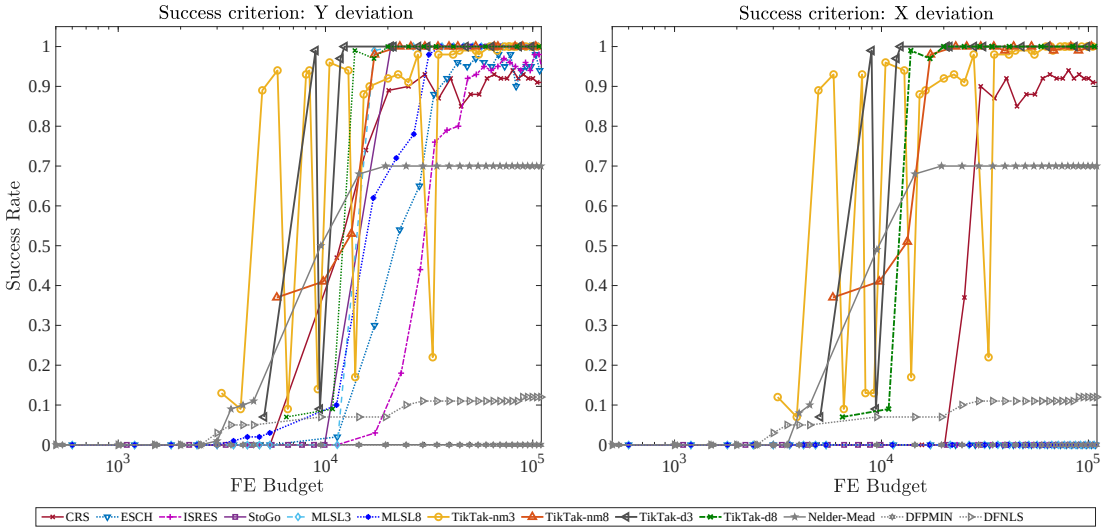
Overall, TikTak-d3 performs the best, reaching an F-val success rate of 100% above 12k FEs. Four optimizers—TikTak-d8, TikTak-nm8, StoGo, and MLSL3—are effectively tied for second place. TikTak-d8 reaches a 99% success rate at 14k FEs and 100% at 20k FEs. The other three have lower success rates below 20k FEs but then all reach 100% as well. As before, the success rates of all TikTak versions are similar under the X-val criterion. The same is not true for StoGo and MLSL3, which have zero success rates under X-val at all budgets; but again as before, beyond 20k FEs, the failed runs have small deviations (10^{-4} to 10^{-5}) from the true values.[21] MLSL8 is slower than these optimizers but eventually attains a 100% success rate—for budgets above 36k.

The remaining optimizers never get to a 100% success rate. CRS stagnates around a 90% success rate (for both F-val and X-val) all the way up to the maximum budget we consider. Failed implementations have large deviations. Similarly, ESCH and ISRES fluctuate between 92% and 98% success rates, and the failed implementations never come close to the true values under either success criterion. (ESCH has parameter deviations between 1 and 10^{-4} and ISRES has between 1 and 0.1). Finally, local algorithms perform poorly under both criteria. Nelder-Mead reaches F-val and X-val success rates of 70% at budgets above 19.5k but then stagnates there. DFNLS never exceeds a success rate of 12%. DFPMIN is never successful. Deviations from the true function and parameter

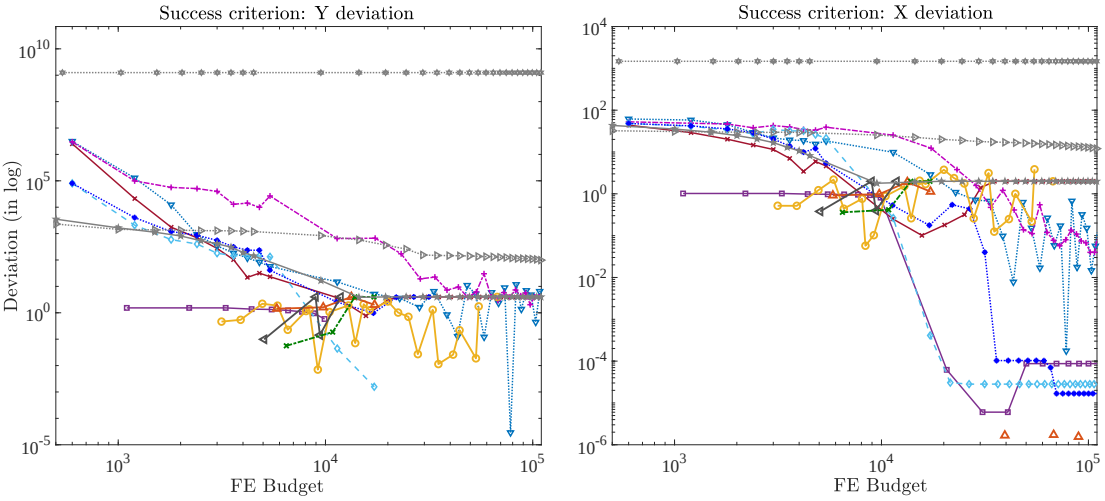
[21] TikTak-nm3 reaches F-val success rates between 98% and 100% above 35k FEs, but its success rate is erratic (ranging from 9% to 90%) at smaller budgets.

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

(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.

values are large for all local algorithms and at all budgets.

Overall, Rosenbrock presented interesting challenges not present in previous test functions. 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.

4.3 Results: Performance Profiles

We now turn to performance profiles, which gives a complementary (and more direct) perspective on how optimizers compare with the best one available for a problem. The set of problems $\mathcal{P}$ consists of the four test functions, but this time we include both 2- and 10-dimensional versions of each. As before, we start each problem from 100 randomly selected starting points, yielding a total of 800 test 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.[22] 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.

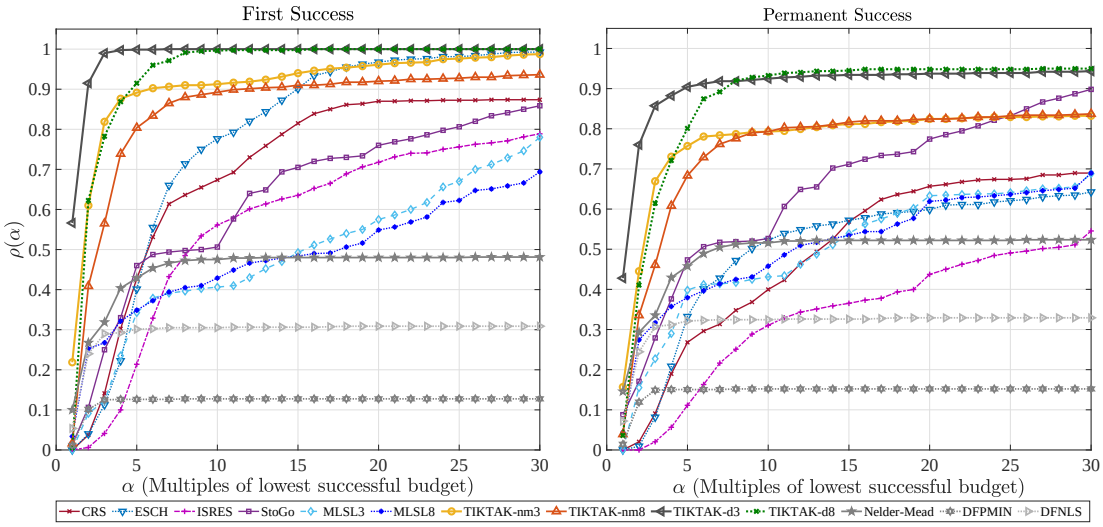
4.3.1 Results

Figure 11 plots the performance profiles for the F-val (top) and X-val (bottom) success criteria, each requiring either first success (left graph) or permanent success (right graph). The success of TikTak-d we have seen in the previous section is also manifested here: the four versions of TikTak share the top four places in the performance profiles. Among these, TikTak-d3 ranks at the top at all levels of α for three out of four success criteria (all but the bottom left panel). In particular, it has the highest probability of being the fastest algorithm for a randomly chosen problem, $\alpha(1)$ averaging about 0.5 across the four panels. By the first success criterion (left panel), it is never more than three times slower than the fastest optimizer for any problem ($\rho(3) = 100\%$). By the permanent/F-val criterion, it has only a 5% chance of being 10 times or more slower than the best, which is only tied with TikTak-d8. For the permanent/X-val, TikTak-d8 takes it over for $\alpha > 5$ (the interpretation being that the chances for the -d8 version to be more than 5

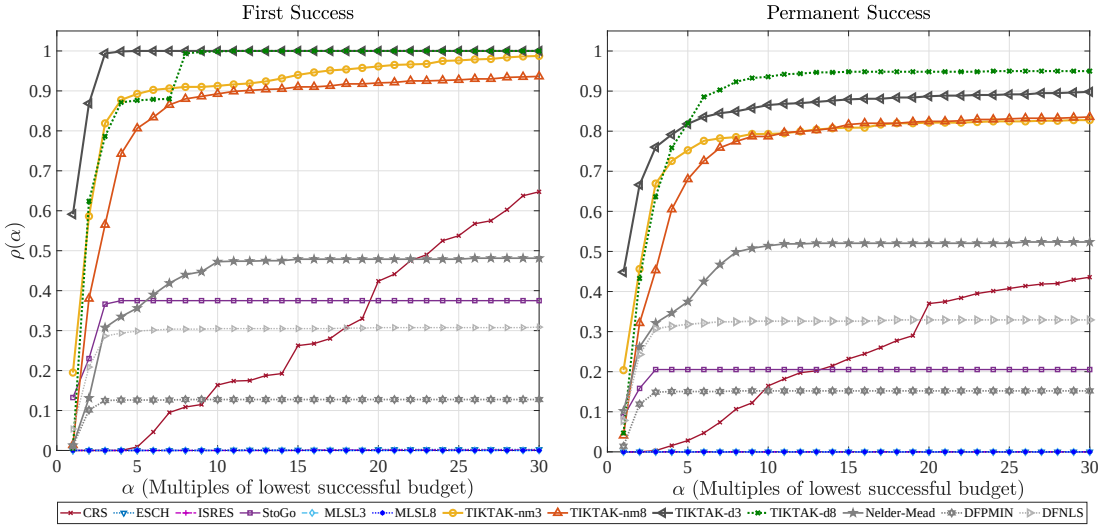
[22] This can happen because optimization routines use different strategies depending on the total budget that is allowed.

FIGURE 11 – Performance Profiles (all Test Functions in 2 and 10 dim)

(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).

times slower than the best optimizer is lower than for the -d3 version, which ranks second best). These two are followed by TikTak-nm3 and -nm8 in third and fourth places.

Turning to NLopt, judging by the F-val criterion and combining the first and permanent success performance (top panels), a few optimizers do better than others in best-case scenarios (i.e., for $\alpha < 5$ or so). In particular, StoGo, both versions of MLSL, and ESCH all reach a 40%–50% probability of being within a factor of 5 of the fastest optimizer (i.e, $\rho(5) \approx 04 - 0.5$). CRS is also within this group by the first success criterion but lags behind when performance is measured by permanent success. Although it is hard to definitively rank the NLopt algorithms given that their performance profiles criss-cross each other and their performance is somewhat sensitive to the particular criteria, StoGo is arguable one of the better ones, ranking near the top among NLopt algorithms across the various criteria. Also recall that when StoGo fails the X-val criteria it is usually by not very much, so its performance in the bottom panel somewhat understates its acceptable performance along that dimension.

CRS also does well, besting StoGo by the first/F-val measure but falling behind in the permanent/F-val measure. By the X-val measure, it does not do as well as StoGo at least for α values less than 20 or so. ESCH does better than CRS by the F-val criterion but poorly by X-val, and from the previous section, we have seen that when it fails, it typically displays large X-val deviations. Finally, despite not ranking at the top in any category in Figure 11, we view MLSL's performance as similar to or better than CRS, mostly based on its performance profile by permanent success and the fact that when it fails, it consistently comes very close to the true parameter values.

The performance rankings of local algorithms are similar under all four success criteria: Nelder-Mead comes first, followed by DFNLS, with DFPMIN in last place. This ordering is not surprising in light of extensive experience researchers have had with these three algorithms over the years, where the same ranking typically emerges in real-life applications. Interestingly, Nelder-Mead does better than some of the NLopt algorithms, although its performance is a long way from the TikTak algorithm. Furthermore, the success rate of Nelder-Mead is boosted somewhat by the inclusion of 2-dimensional versions of test functions, where it does better than it does under 10-dimensional versions. Furthermore, recall from Section 4.2 that deviations among failed implementations are substantially larger for local algorithms, which serves as an important warning for their use in global optimization problems.

4.4 Taking Stock

Having examined the data and deviation profiles for each (10-dimensional) test function as well as the performance profiles for the entire set of problems, we can now summarize the overall performance of each optimizer. Versions of the TikTak algorithm performed the best across different evaluation criteria, with TikTak-d3 ranking at the top more often than any other variant. TikTak-nm is typically in second place because of the slower performance of Nelder-Mead in local search stage. That said, Nelder-Mead has high reliability as a local algorithm, so in complex and higher-dimensional problems, we believe that it would be prudent to use it 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 was Rastrigin where StoGo was the clear top performer.

Among NLopt algorithms, one can make a case for StoGo, and MLSL, as well as CRS. The latter does well on relatively easier problems—such as Griewank and Levi functions, but does poorly on harder ones such as Rastrigin and Rosenbrock. MLSL and StoGo are somewhat slower on easier problems and can fail to achieve the strict tolerance levels we imposed, but they do better in harder problems, and 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 10k FEs, whereas CRS fails Rastrigin completely and gets stuck at a 90% success rate on Rosenbrock. And when it fails, parameter values are far from the true values. In our view, this is an important drawback because an algorithm that cannot eventually attain a 100% (or close) success rate is a very risky choice in real-life applications, as increasing the computational budget does not get us closer to the true minimum. MLSL is similar to StoGo along these lines but fails Rastrigin completely.

ESCH has high F-val success rates (between 80% and 100%) for all test functions; however, the algorithm is never successful under X-val. Those failed instances have small (parameter value) deviations for Griewank, Levi, and Rastrigin, but larger deviations for Rosenbrock. ISRES never successfully solves a problem under X-val at budgets below 30k that we consider, and failed instances have relatively large deviations, especially for Rastrigin and Rosenbrock. In addition, success rates of ISRES (as well as ESCH) fluctuate substantially across computational budgets for several test functions.

In some cases, local algorithms reach higher success rates than some of the global algorithms, but they all stagnate at levels below 100% even when the computational budget is increased (with the minor exception of Griewank for Nelder-Mead). For reasons

just discussed, algorithms that cannot reach a 100% success rate at any budget are not reliable for solving global optimization problems. In addition, failed implementations of the local algorithms always have large deviations. This implies that failed implementations of local algorithms return values that are far away from the true global minima (possibly local algorithms are stuck at a local minima). In comparison, failed implementations of global algorithms are oftentimes able to come into 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, where 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. It is taken from a recent paper by Busch et al. (2015), who studied 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 (seven) function 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 denote the labor income of an individual at time t, and define $y_t \equiv \log Y_t$, 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 such 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] Busch et al. impose $p_2 = p_3, \sigma_2 = \sigma_3$, leaving 7 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 follow the same paper and 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 is 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. Of course, one can increase the number of individuals simulated

[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 only consider 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 for all algorithms), which provides us with the set $\mathcal{P}$. We consider computational budgets up to 70k 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 above, here 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 CRS and is equal to $f(x) = 3.4863$. The corresponding parameter values are shown in Table C.1 in Appendix C.

To get a rough idea about how the objective function varies with each of the seven parameter values, Figure C.5 in Appendix C plots the 1-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. First, for six of the seven parameters (except x_7), the minimum objective value lies close to the bound of each parameter value. Further, 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, the next Figure (C.6) 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, which suggests optimizers can get trapped in a local optimum nearby and stop prematurely.

Data and Deviation Profiles

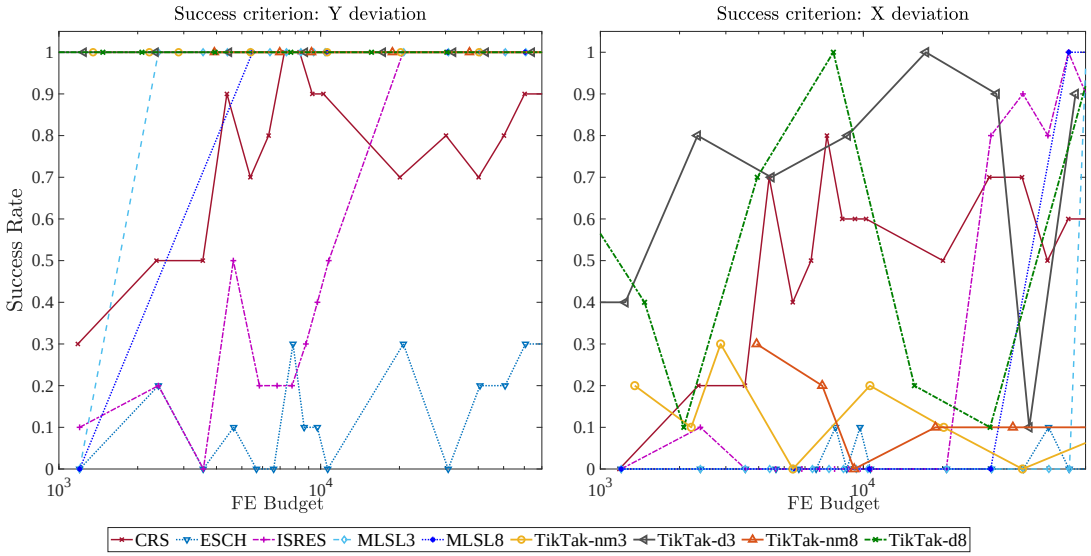
Figure 12 plots the data and deviation profiles. Overall, TikTak-d performs the best, which is perhaps not surprising given the results so far. Among NLopt algorithms,

[25] For the NLopt algorithms, we implement minimizations at 16 different computational budgets using the numbers of FEs as explicit stopping criteria. An exception is MLSL8 for which we only compute 6 steps. For TikTak, we again generate different numbers of Sobol' points to implement minimizations at different computational budgets.

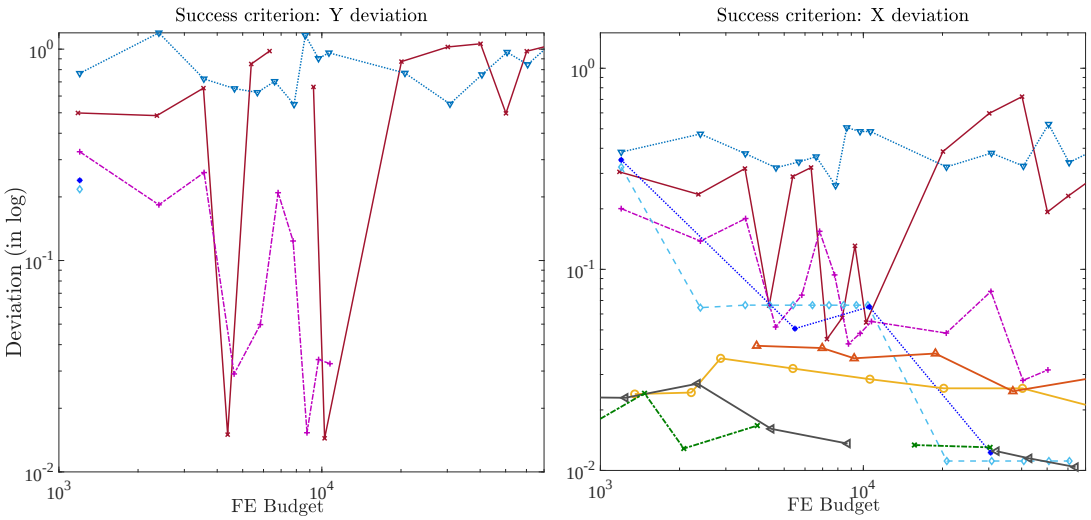
[26] Although this is stating the obvious, we should stress that while visualizing an objective function through these slices is informative, it is nowhere near a complete description of the objective function, so it must be used with care.

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.

MLSL is arguably the top performer. A quick glance at the data profiles shows that all optimizers struggle more than on test functions with many erratic fluctuations in success rates. We now delve into the details.

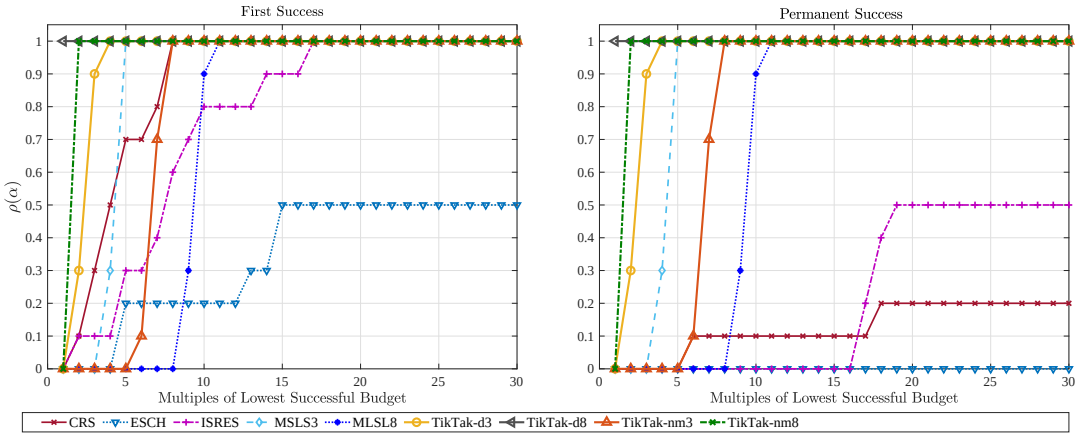
Both versions of TikTak-d perform well under F-val criterion, but their success rates fluctuate substantially under the X-val one. In particular, TikTak-d3 reaches a 100% F-val success rate at all computational budgets (the lowest budget we consider is 600), but is more erratic under the X-val criterion.[27] However, for budgets above 5k FEs, failed implementations are very close to the success threshold, with deviations from true parameter values between $10^{-1.9}$ and $10^{-1.98}$ (bottom right panel). TikTak-d8 is similar in this respect and also comes as close to the parameter values as the -d3 version, so the additional function evaluations it is allowed by the tighter local tolerance only serves to slow down its performance without a clear benefit. The Nelder-Mead versions of TikTak rank clearly behind the -d3 and -d8 versions when judged by the X-val performance. Although the difference is often not substantial, the failed implementations are almost always 3 to 5 times farther from the true values relative to the -d versions, and their X-val success criteria are also quite a bit lower.

Turning to the NLopt algorithms, there are clear differences in performance, although none of them come close enough to the true parameter values in a consistent fashion. Arguably, the most successful one is MLSL (both versions). They both reach 100% F-val success rates at relatively low budgets, although they do not reach high X-val success rates until we reach very high budgets—60k FEs and beyond. That said, similar to what we have seen with the test functions, the deviations from true parameter values are lower than other NLopt algorithms and keep improving with higher FEs. For budgets beyond 20k FEs or more, MLSL attains X-val deviations of $10^{-1.9}$, very close to the success threshold.

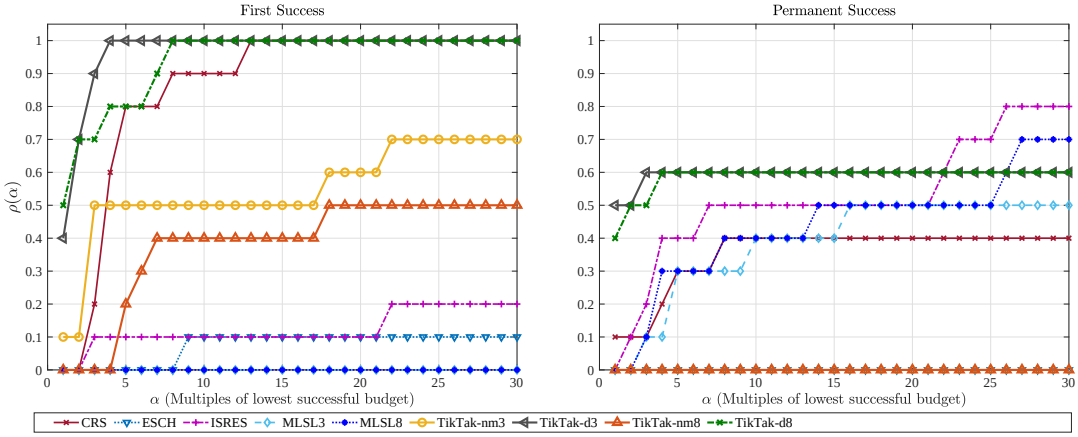
As for CRS, ESCH, and ISRES, none of them do a satisfactory job as judged by their ability to come close to the true parameter values in a consistent manner. ISRES is somewhat better than the other two, eventually reaching an F-val success rate of 100% above 21k FEs, but stagnating between a 0% and 50% X-val success rate below 30k FEs. Its success rate rises above 80% beyond this point, but for failed implementations, it is farther from all TikTak versions at almost all budgets. CRS is further behind, with inconsistent F-val success rates, never reaching above 60% X-val success rates, and displaying large deviations from true values along both dimensions for failed implementations. Finally, ESCH ranks last with very poor performance across the board.

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, with the performance profiles of both TikTak-d versions always lying above others (with the small exceptions of very large budgets and X-val criteria, bottom right panel). One point to keep in mind is that performance profiles rely on success

[27]Specifically, TikTak-d3 reaches X2 success rates of 20% at 600 and 40% at 900. Then X2 success rates vary between 70% and 100% at budgets between 2.3k and 32k; however, the success rate again drops down to 10% at 43k, before increasing again to 90% at 64k (right graph of Panel A).

rates, so they do not always capture the nuances of optimizers that technically fail but come very close, as we have seen for TikTak-nm and MLSL, among others. This pushes the performance profile of some optimizers, such as CRS, above others, even though the data and deviation profiles clearly showed that it is one of the weakest performers. The same feature also affects MLSL, which is the best performer behind TikTak under F-val criteria but ranks low in X-val, the first success criterion.

6 Conclusion

In this paper, we have benchmarked the performance of seven 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. As we are particularly interested in understanding optimizers' performance in typical economic applications, we also use the same optimizers to solve an estimation exercise that is commonly found in economics. We consider seven global optimizers—CRS, ISRES, ESCH, StoGo, MLSL, TikTak-nm, and TikTak-d—as well as three local algorithms—Nelder-Mead, DFPMIN, and DFNLS.

We find that TikTak-d has the strongest performance overall, for both 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. In addition, 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 differs across different test functions and the economic application. MLSL and ISRES perform better in solving the economic application, but they are relatively less successful in minimizing (some of) the test functions. However, even when MLSL fails, it manages to come very close to the success threshold under both the F-val and X-val criteria. Based on this performance, we find MLSL to be one of the better NLopt algorithms that we tested. StoGo is another strong performer, arguably the best NLopt algorithm for test functions, but we have not included it in the economic application for reasons explained earlier.

The performances of CRS, ESCH, and ISRES are a step behind the others. For real-life applications, the minimum that we should expect from a global optimization algorithm is that it finds the true global optimum reliably even if this requires a large computational budget. These algorithms fail this test too often, which raises questions

about their suitability for the complex and high-dimensional problems found in economic applications.

Local algorithms display a similarly 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 sounds a strong cautionary note to discourage 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 Algo-

rithm 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

A Appendix: Detailed Description of Algorithms

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

A.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), and the authors 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 purely based 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 $N = 10(n + 1)$.

A.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 offsprings. 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 is 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)).

A.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.

A.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 substeps. 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 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). 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 $\alpha = 0.2, \gamma = 0.85$.

A.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 are 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 it 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 $M = 60 \times n/10$.

A.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 version, which 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 nonsmooth 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 $\gamma = 0.3$.

A.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 $G = \emptyset$. G represents the set of garbage boxes.
2. Rank boxes $\tilde{B}$ in C based on 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, and $maxGrad$ 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.

A.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[28] 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**:

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

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.[29]

- *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.

A.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

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

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

Rosenbrock Function The gradient of the Rosenbrock function is equal to

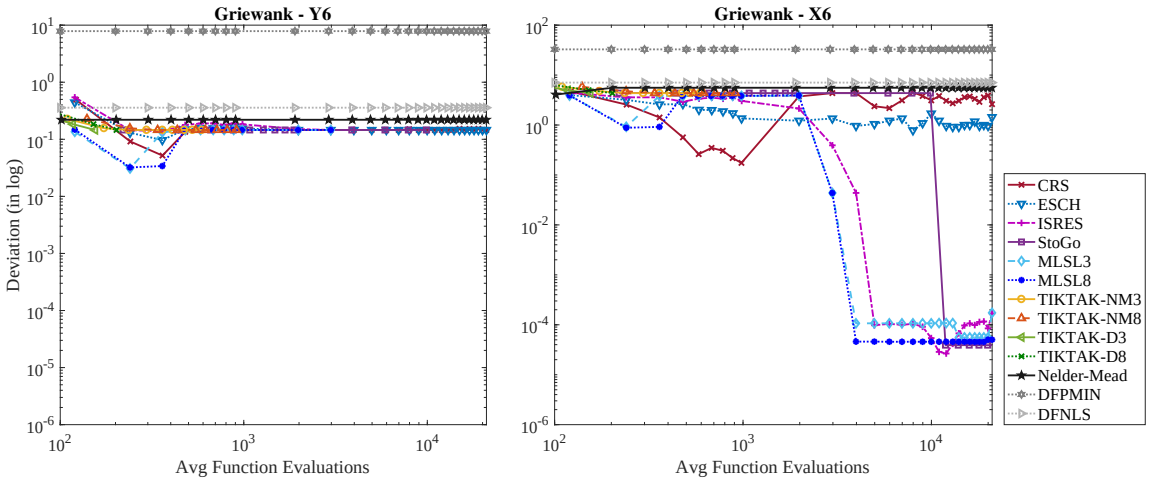
$$\nabla f = \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_i - x_{i-1}^2) - 400x_i(x_{i+1} - x_i^2) - 2(1 - x_i) \text{ for } i \notin \{1, n\} \\ 200(x_n - x_{n-1}^2)^2 \end{pmatrix}.$$

B Appendix: Data and Deviation Profiles for Two-Dimensional Test Functions

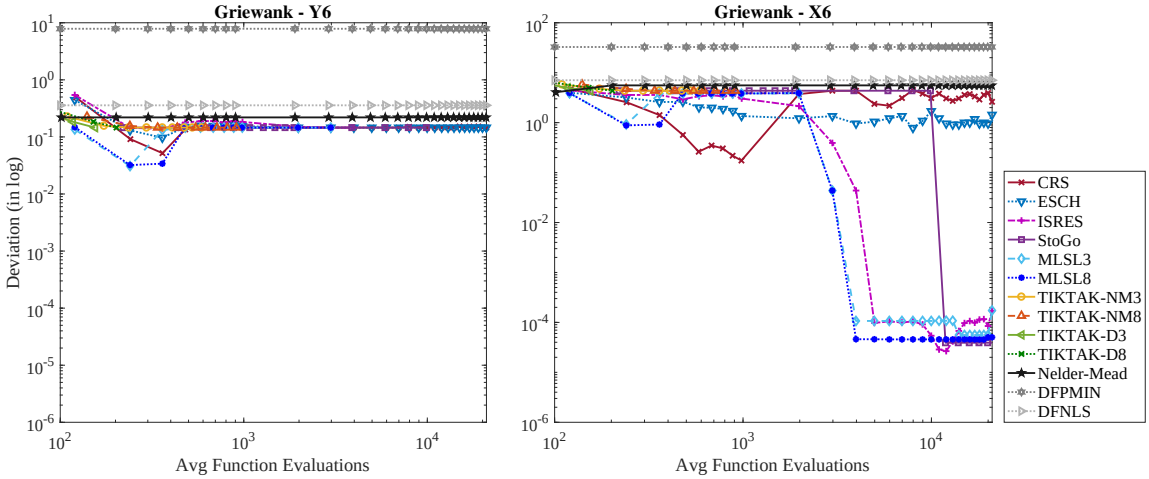
In this section, we provide the data and deviation profiles for each test function in two dimensions. For definitions and explanation of the figures, see Section 4.2. Note that the performance profiles in Section 4.3 already include these two-dimensional test functions (as part of the 800 problems that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

FIGURE B.1 – Data and Deviation Profiles—Griewank (2-dim)

Panel A: Data Profile by Success Criteria



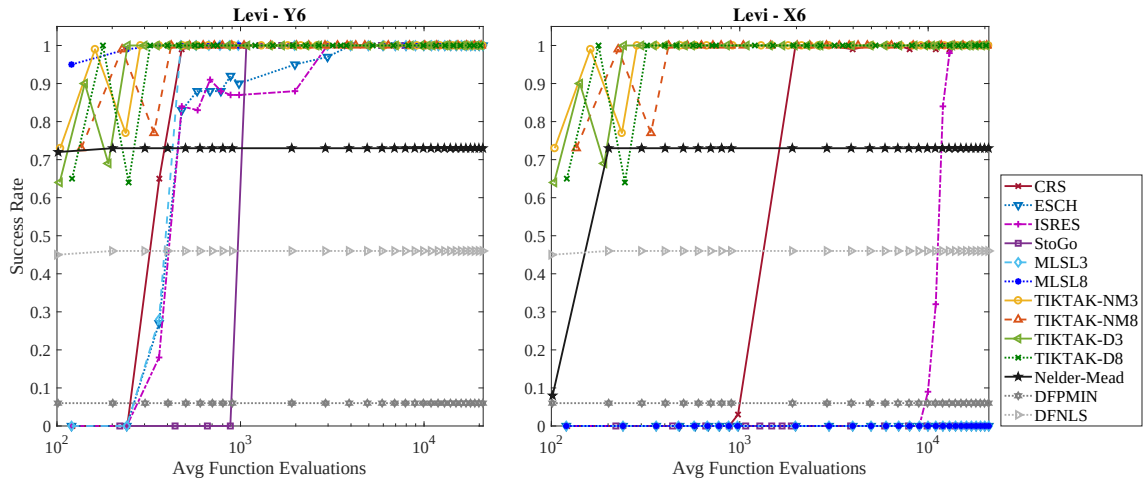
Panel B: Deviations by Success Criteria



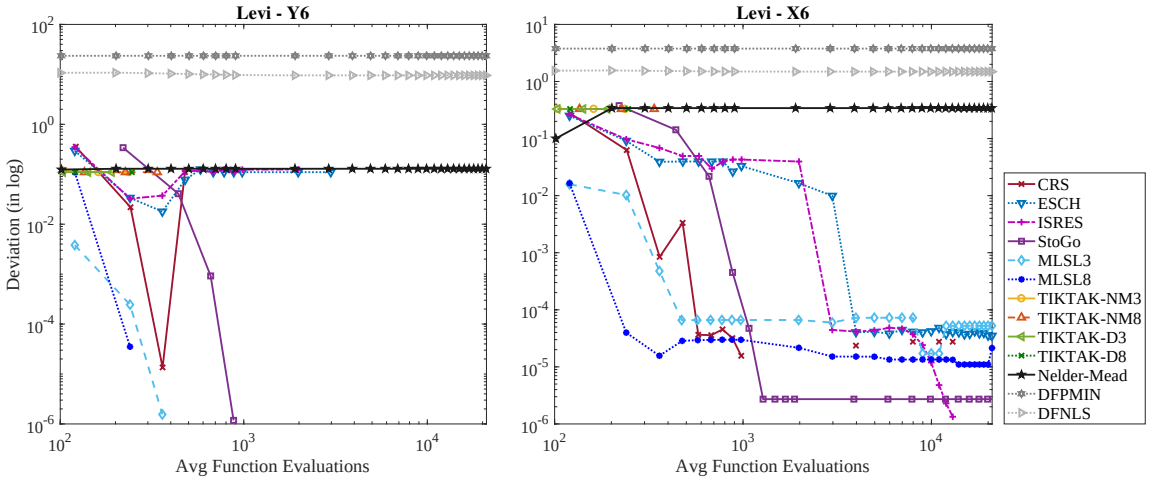
Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

FIGURE B.2 – Data and Deviation Profiles—Levi (2-dim)

Panel A: Data Profile by Success Criteria



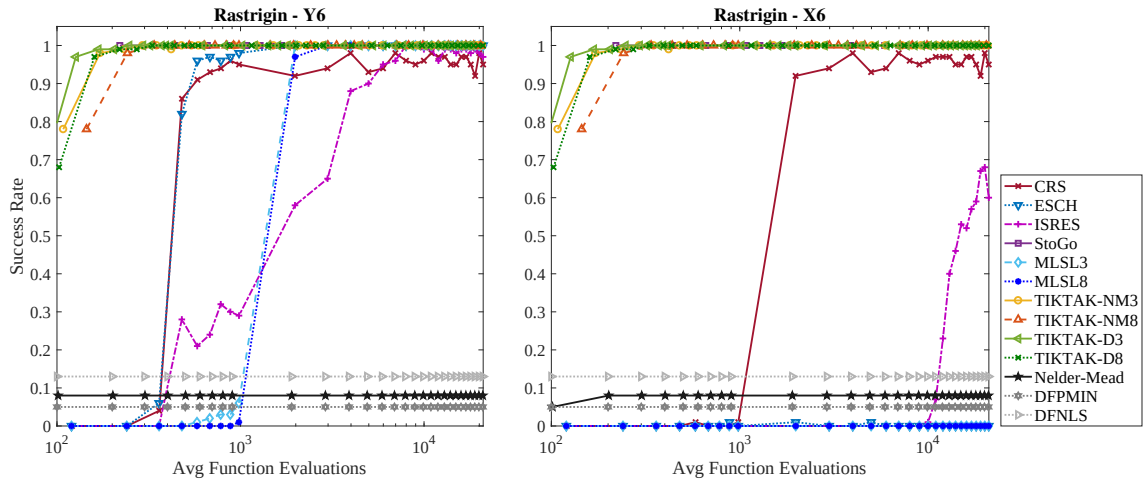
Panel B: Deviations by Success Criteria



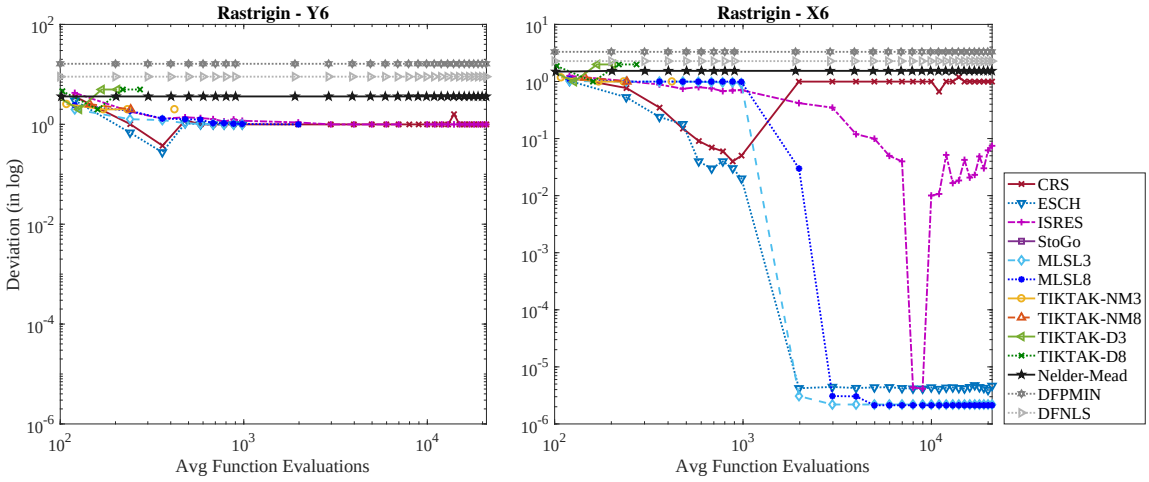
Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

FIGURE B.3 – Data and Deviation Profiles—Rastrigin (2-dim)

Panel A: Data Profile by Success Criteria



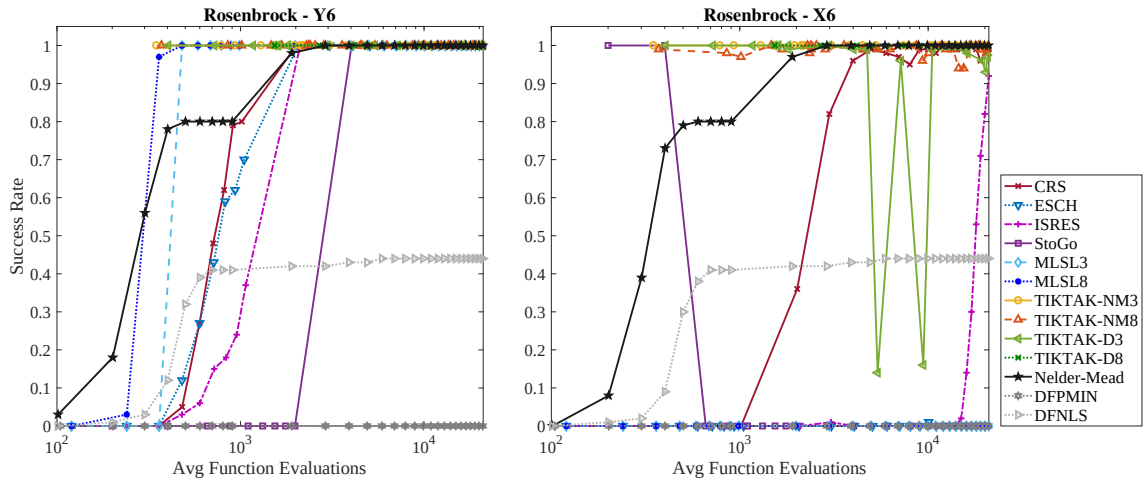
Panel B: Deviations by Success Criteria



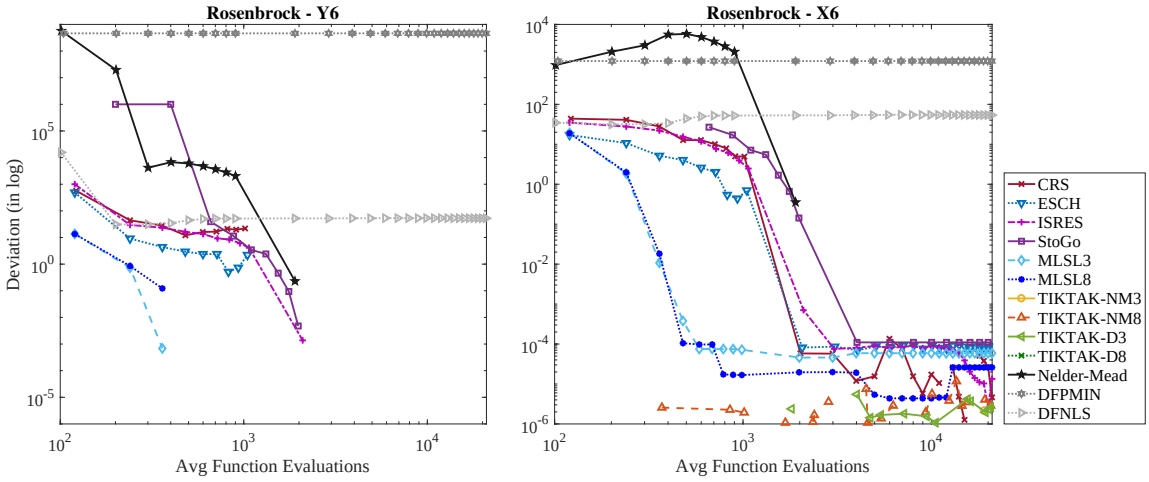
Notes: For explanations and figure notes, see Section 4.2. Data and deviation profiles are defined in the same way. The only difference is that the test functions here are in two (and not 10) dimensions.

FIGURE B.4 – Data and Deviation Profiles—Rosenbrock (2-dim)

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.

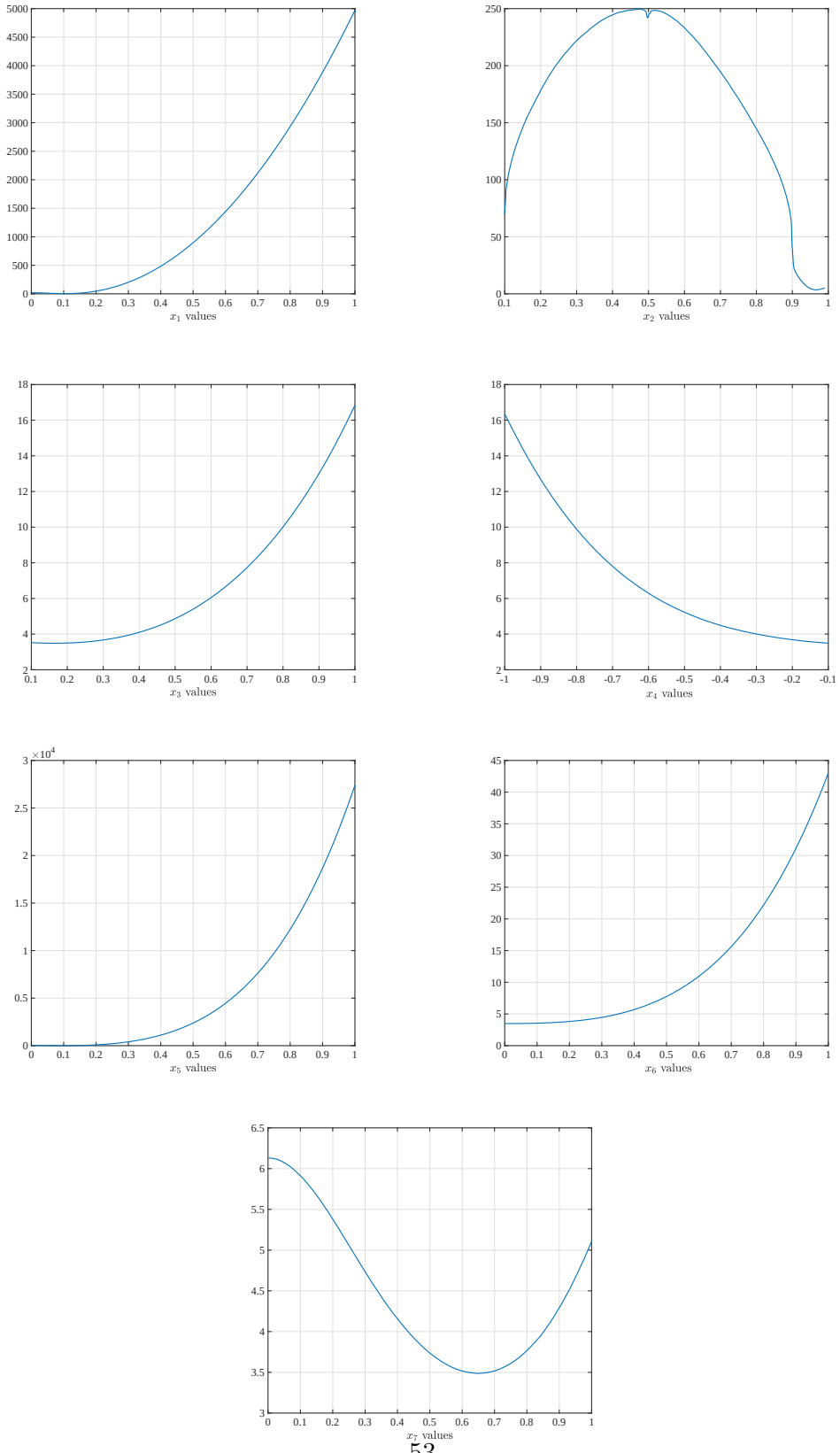
C Economic Application: Slices of the Objective Function Surface

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	0.103324
x_2	p_1	Weight of center of ζ distribution	0.965695
x_3	μ_2	Mean of right tail of ζ distribution	0.160240
x_4	μ_3	Mean of left tail of ζ distribution	–0.1
x_5	$\sigma_{1,\zeta}$	St. dev. of center of ζ distribution	0.095628
x_6	$\sigma_{2,\zeta}$	St. dev. of right tail of ζ distribution	0.00367
x_7	ϕ	Aggregate risk transmission parameter	0.649274

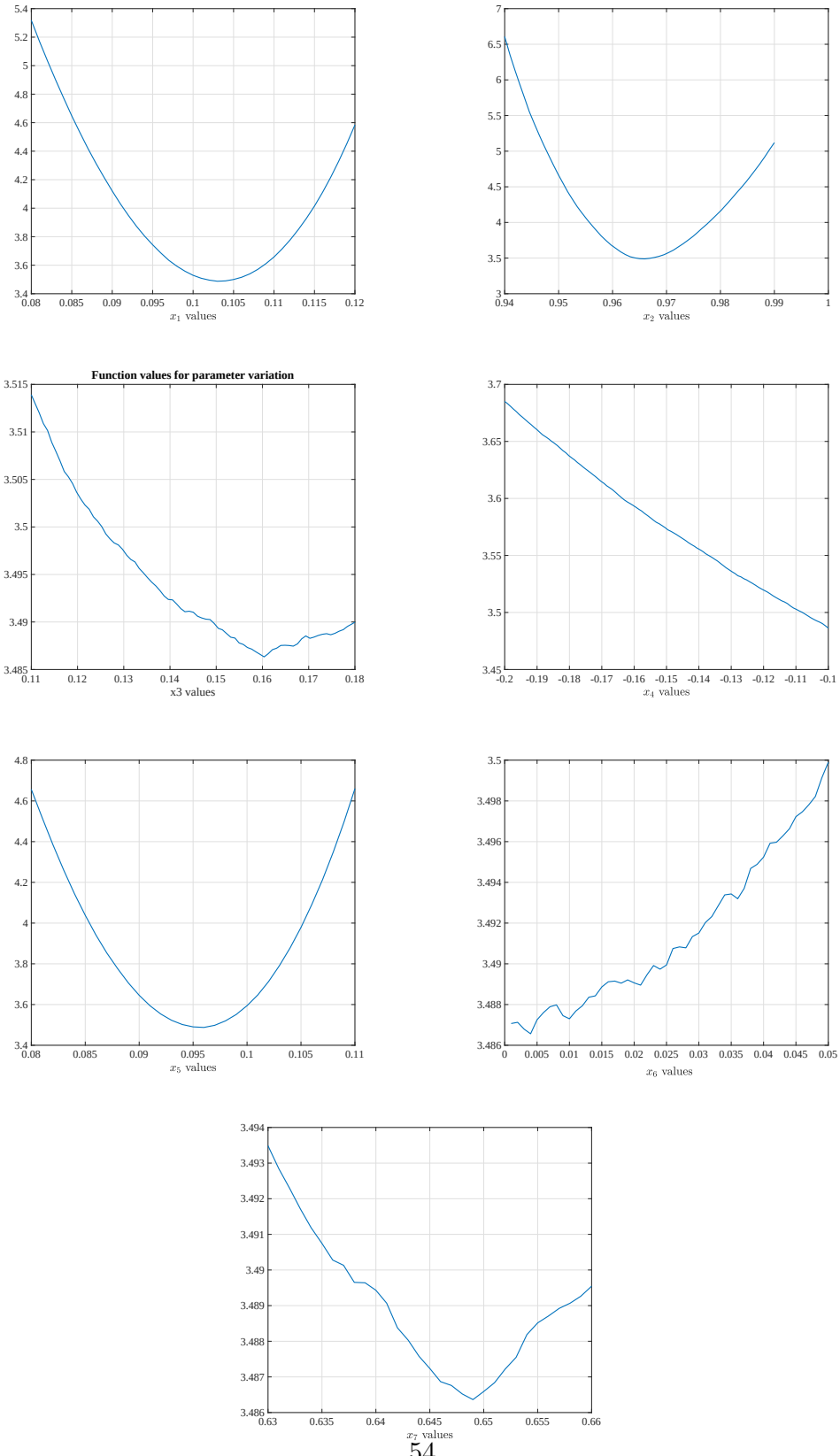
Notes: The parameters are estimated by matching data moments from Sweden. In this table, we report the parameter values that correspond to the smallest function value (equal to 3.4863085) that is found by any algorithm in our benchmarking exercise. We consider this point to be the "true" minimum to define 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 C.5 – 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 C.6 – 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.