Lecture 5: Optimization

Fatih Guvenen

Spring 2019

Optimization

Overview of Optimization

- Most commonly needed for:
  - Solving a dynamic programming problem.
  - Root-finding as a minimization problem (discussed earlier)$\rightarrow$ solving for GE.

Two main trade-offs:

- Fast local methods versus slow but more global methods.
- Whether or not to calculate derivatives (including Jacobians and Hessians in multidimensional case!).

- Some of the ideas for local minimization are very similar to root-finding.
  - In fact, Brent's and Newton's methods have analogs for minimization that work with exactly the same logic.
  - Newton-based methods scale very well to multidimensional case

LOCAL OPTIMIZATION

One-Dimensional Problems

- Note: You only need **two** points to bracket a zero. But you need **three** to bracket a minimum: $f(a), f(c) > f(b)$.
- So first obtain those three points. Many economic problems naturally suggest the two end points: $(c_{min} = \epsilon, c_{max} = y - a_{min})$.
- Sometimes, I use NR's `mnbrak.f90` routine. Nothing fancy.
- In one dimension, Brent's method works well and is pretty fast. **(Figure)**
- I often prefer it to Newton's because I know that I am always bracketing a minimum.
- NR has a version of Brent that uses derivative information very carefully, which is my preferred routine (`dbrent.f90`).

Multi-Dimensional Optimization

- Multi-dimensional optimization can be a very hard problem because:
  - High-dimensional spaces have very unintuitive features. Extrapolating our understanding from 1- or 2-dimensions will get us in trouble.
  - Further: Unlike 1- or 2-dimensional problems, you cannot plot and visualize the objective
  - You can at best plot some “slices”, which are informative (so is essential to do) but they are never conclusive.
  - If there are multiple optima—and very often there are *tons* of them—then you can never guarantee finding the global optimum.
- ∴ Proceed with maximum caution.

Multidimensional Optimizers: Three Good Ones

- I will first talk about local optimizers. Then turn to global ones.
- Key point: There is no one-size fits all optimizers. They each have their advantages and drawbacks:
  1. Quasi-Newton Methods: Very speedy but also greedy: it will either get you to the optima or into a ditch, but will do it quickly!
  2. Nelder-Mead's Downhill Simplex: Slow, patient, methodical. Very good global properties even though it's a local optimizer.
  3. Derivative-Free Nonlinear-Least-Squares (DFNLS): The new kid on the block. Oftentimes very fast and pretty good at finding the optimum. Global properties between the first two.
     - Specifically designed for MSM-like objective functions.
- All three must be in your toolbox. You will use each depending on the situation. Will have more to say.

I. Quasi-Newton Methods: Fast and Furious

- Once you are "close enough" to the minimum, *quasi*-newton methods are hard to beat.
- Quasi-Newton methods reduce the N-dimensional minimization into a series of 1-dimensional problems (line search)
- Basically starting from a point **P**, take a direction vector **n** and find λ that minimizes $f(\mathrm{P} + \lambda \mathrm{n})$
- Once you are at this line minimum, call **P+**λ**n**, the key step is to decide what direction to move next.
- Two main variants: Conjugate Gradient and Variable Metric methods. Differences are relatively minor.
- I use the BFGS variant of Davidon-Fletcher-Powell algorithm.

II. Nelder-Mead Downhill Simplex: Slow and Deliberate

- Powerful method that relies only on function evaluations (no derivatives).
- Works even when the objective function is discontinuous and has kinks!
- It is slow, but has **better global convergence properties** than derivative-based algorithms (such as the Broyden-Fletcher-Goldfarb-Shanno method).
- It **must be** part of your everyday toolbox.

II. Nelder-Mead Simplex

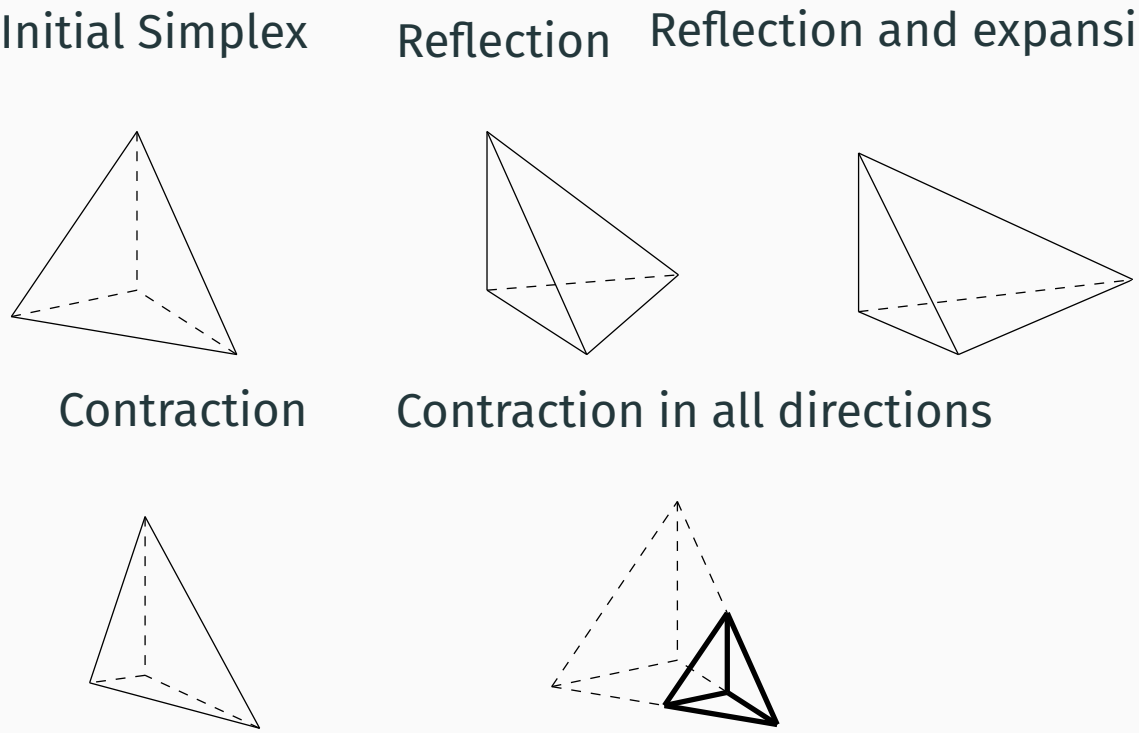


Figure 1: Evolution of the *N*-Simplex During the Amoeba Iterations

III. DFLS Minimization Algorithm: Sweet Spot

A Derivative-Free Least Squares (DFLS) Minimization Algorithm:

- Consider the special case of an objective function of this form:

$$\min \Phi(\mathbf{x}) = \frac{1}{2}\Sigma_{i=1}^{m} f_i(\mathbf{x})^2$$

  where $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2, .., m$.

- Zhang-Conn-Scheinberg (SIAM, 2010) propose an extension of the BOBYQA algorithm of Powell that does not require derivative information.

- The key insight is to build quadratic models of each f_i individually, rather than of Φ directly.

- The function evaluation cost is the same (order) but it is more accurate, so faster.

How Do We Evaluate/Compare Optimizers?

Judging The Performance of Solvers

- First begin by defining a convergence criteria to use to judge when a certain solver has finished its job.

- Let x_0 denote the starting point and τ, ideally small, the tolerance. The value f_L is the best value that can be attained.
  - In practice, f_L is the best value attained among the set of solvers in consideration using at most μ_f function evaluations (i.e., your "budget").

- Define the stopping rule as :

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_L). \tag{1}$$

- We will consider values like $\tau = 10^{-k}$, for $k \in \{1, 3, 5\}$.

Moré and Wild (2009)

- Performance profiles are defined in terms of a performance measure $t_{p,s} > 0$ obtained for each problem $p \in P$ and solver $s \in S$.

- Mathematically, the performance ratio is:

$$r_{p,s} = \frac{t_{p,s}}{\min \{t_{p,s} : s \in S\}}$$

- $t_{p,s}$ could be based on the amount of computing time or the number of function evaluations required to satisfy the convergence test.

- Note that the best solver for a particular problem attains the lower bound $r_{p,s} = 1$.

- The convention $r_{p,s} = \infty$ is used when solver s fails to satisfy the convergence test on problem p.

Performance Profile

- The **performance profile** of a solver $s \in S$ is defined as the fraction of problems where the performance ratio is at most α, that is,

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

  where $|P|$ denotes the cardinality of P.

- Thus, a performance profile is the probability distribution for the ratio $r_{p,s}$.

- Performance profiles seek to capture how well the solver performs relative to the other solvers in S on the set of problems in P.

- $\rho_s(1)$ is the fraction of problems for which s is the best.

- In general, $\rho_s(\alpha)$ is the % of problems with $r_{p,s}$ bounded by α. Thus, solvers with high $\rho_s(\alpha)$ are preferable.

Performance Profile

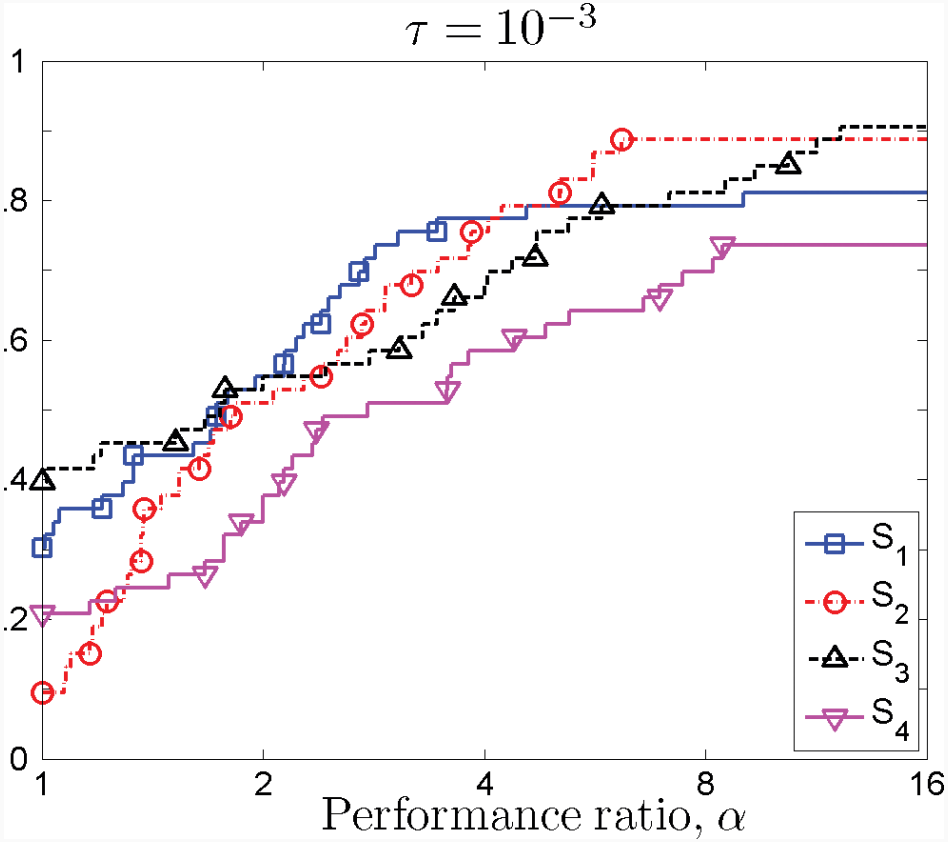


FIG. 2.1. *Sample performance profile $\rho_s(\alpha)$ (logarithmic scale) for derivative-free solvers.*

Data Profiles

- Oftentimes, we are interested in the percentage of problems that can be solved (for a given τ) with μ_f function evaluations.
- We can obtain this information by letting $t_{p,s}$ be the number of function evaluations required to satisfy (1) for a given tolerance τ.
- Moré and Wild (2009) define a data profile as:

$$d_s(\alpha) = \frac{1}{|P|} \text{size} \left\{ p \in P \ : \ \frac{t_{p,s}}{n_p + 1} \leq \alpha \right\},$$

where n_p is the number of variables in problem p.

Measuring Actual Performance: DFNLS wins

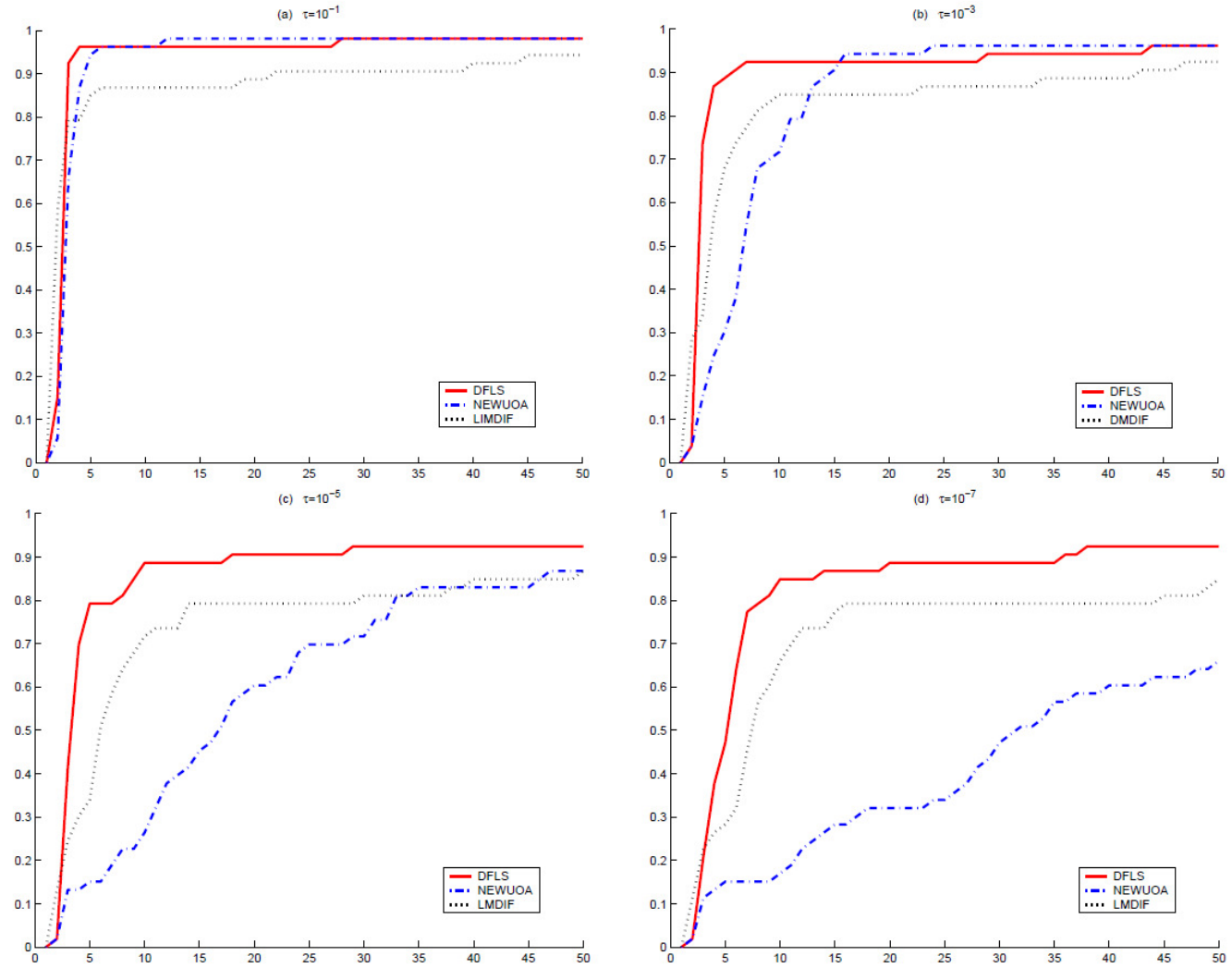
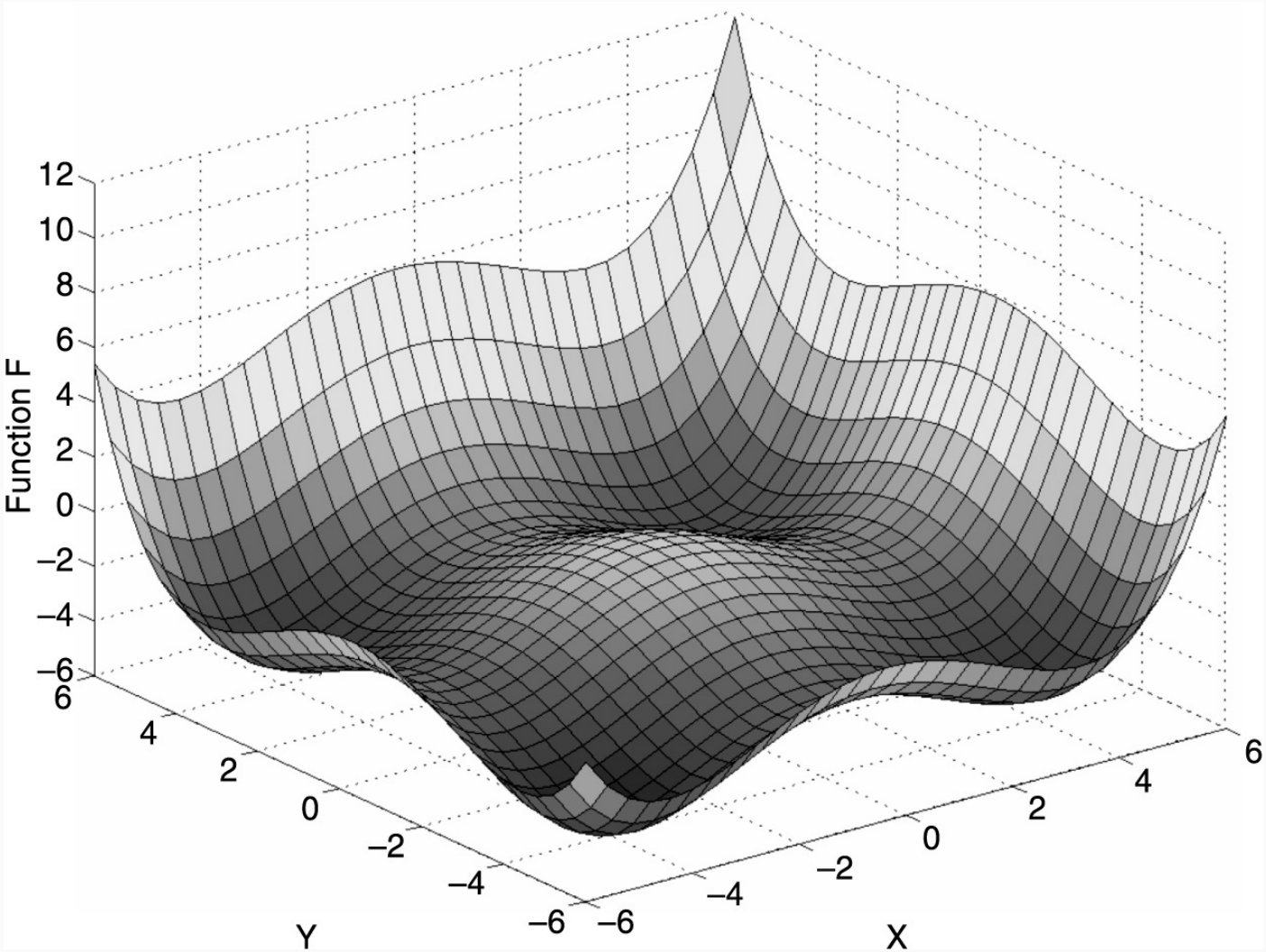


FIG. 5.1. *Data profiles of function* $d_s(\alpha)$ *for smooth problems: (a)* $\tau = 10^{-1}$, *(b)* $\tau = 10^{-3}$, *(c)* $\tau = 10^{-5}$, *(d)* $\tau = 10^{-7}$

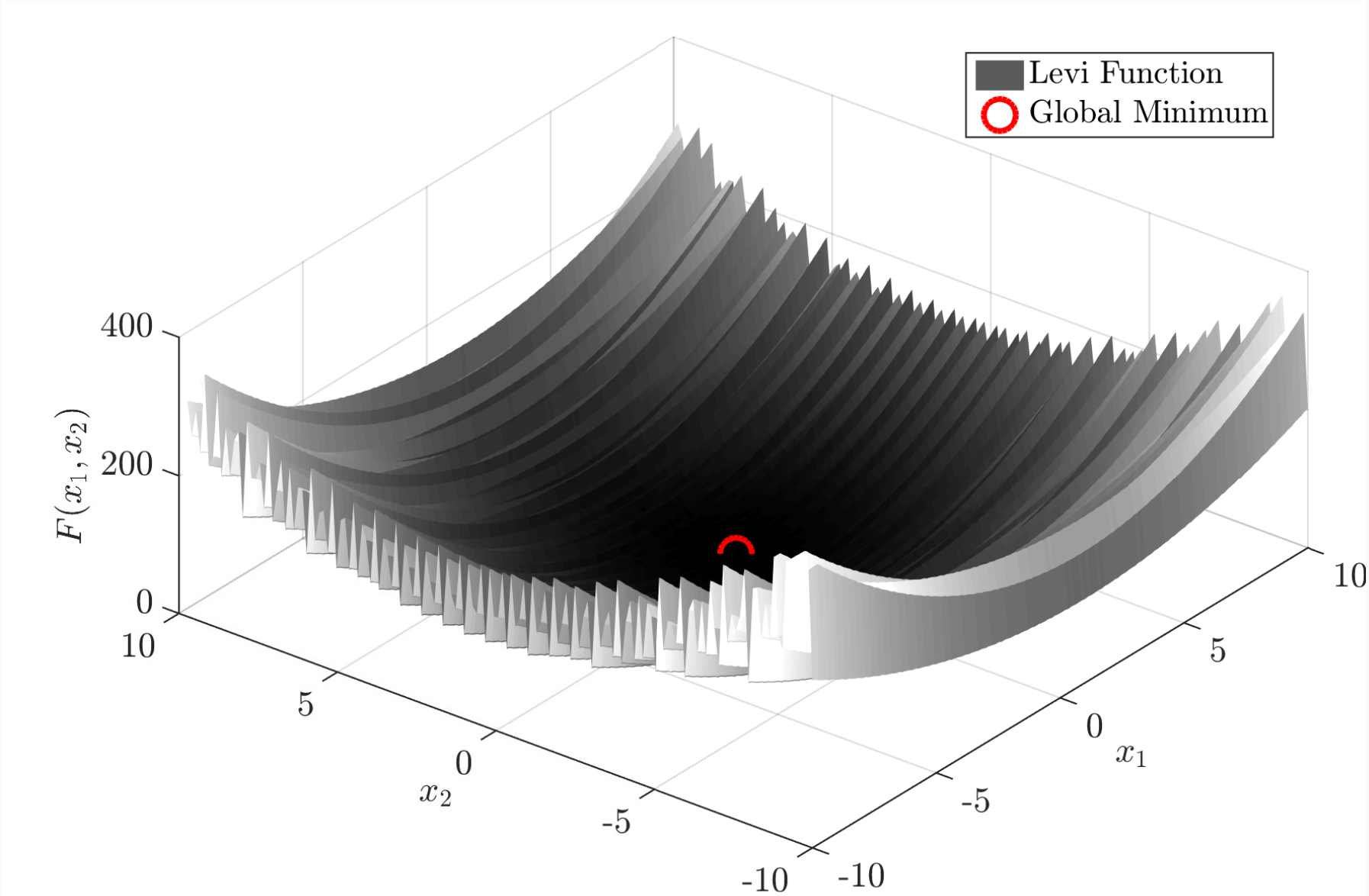
GLOBAL OPTIMIZATION

How Your Objective Function Looks Like

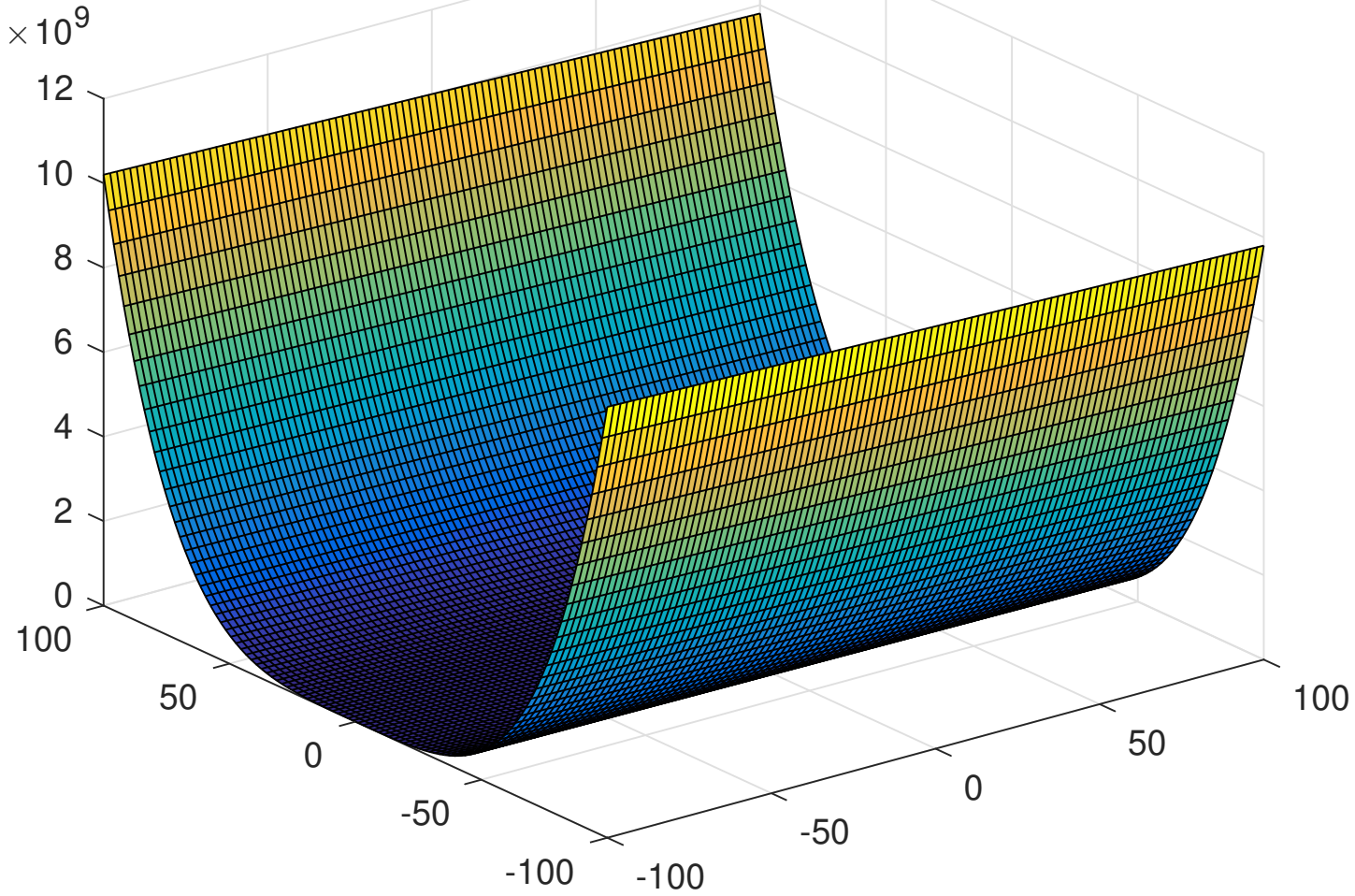
- **Caution:** Very easy to find **a** minimum, very hard to ensure it is ***the*** minimum.



How Your Objective Function Looks Like



How Your Objective Function Looks Like



A Practical Guide

How to proceed in practice?

1. If you can establish some geometric properties of your objective function, this is where you should start.
2. For example, in a standard portfolio choice problem with CRRA utility and linear budget constraints, you can show that the RHS of the Bellman equation has a single peak (no local maxima).
3. Even when this is theoretically true there is no guarantee your numerical objective will have a single peak because of the approximations. (We will see an example in a few weeks).
4. The least you should do is to plot **slices** and/or two-dimensional ***surfaces*** from your objective function.
5. These will give you valuable insights into the nature of the problem.

A Practical Guide

- Having said that, when you solve a DP problem without fixed costs, option values, max operators, and other sources of non-concavity, local methods described above will usually work fine.
- When your minimizer converges, restart the program from the point it converged to. (You will be surprised at how often the minimizer will drift away from the supposed minimum!)
- Another idea is to do random restarts—a bunch of times!
- But this is not very efficient, because the random restart points could end up being very close to each other (general problem with random sampling—small sample issues.)
- Is there a better way? Yes (with some qualifications.)

TikTak: A Global Optimization Algorithm

Here is the algorithm that I use and it has worked well for me. Experiment with variations that may work better for your problem!

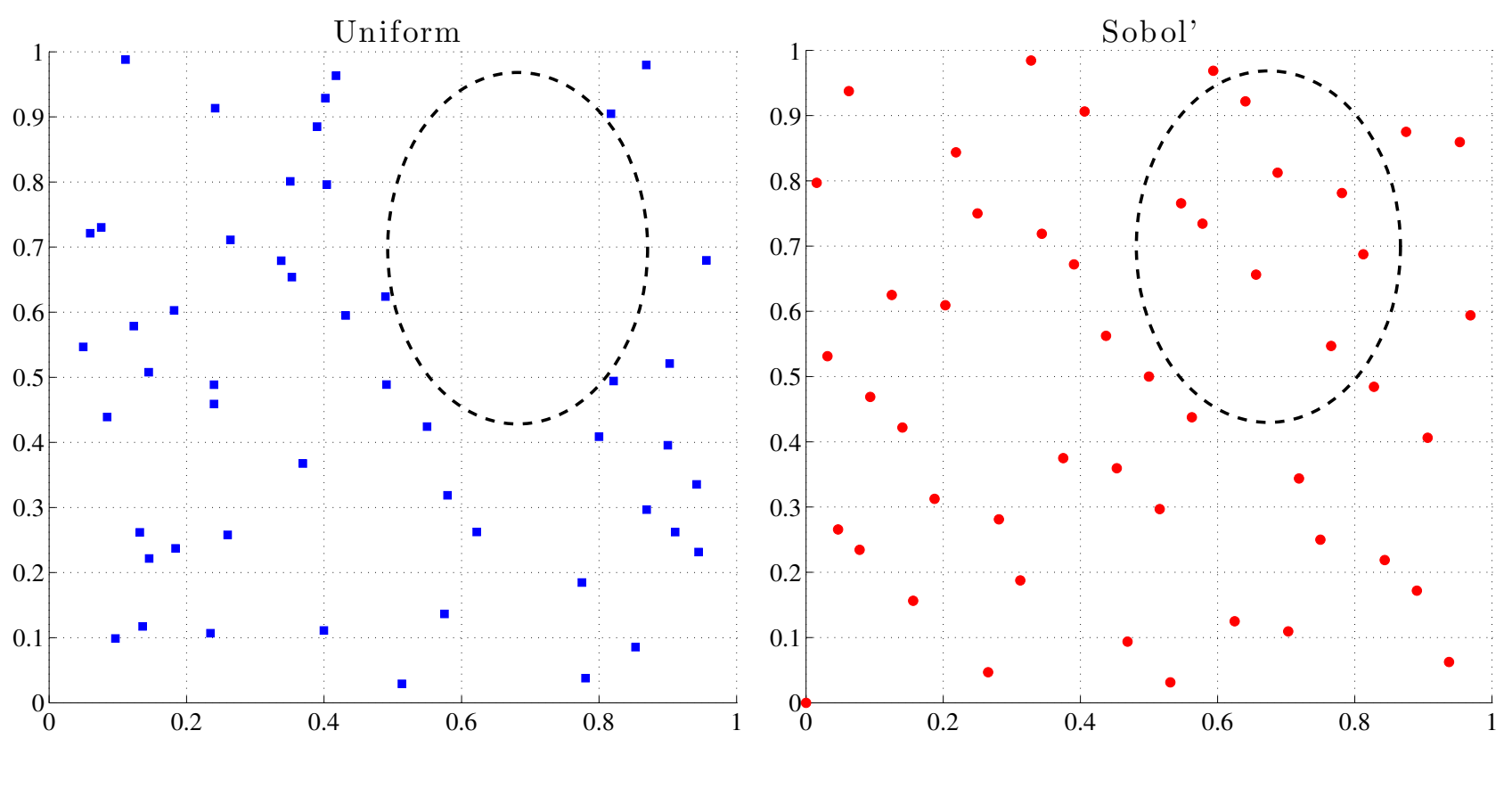
1. Set $j = 0$ and start the iteration.
2. Start a local optimizer with initial guess x_j and run until it "converges" to a new point, call z_j.
3. Draw a **quasi-random** initial guess, y_j (using *Halton*'s or *Sobol*'s sequence. More on this in a minute).
4. Take new starting point as: $\widetilde{x}_j = \theta_j z_j^* + (1 - \theta_j) y_j$ where $\theta_j \in [0, 1]$ and z_j^* is the best point obtained up until iteration j.
5. Update $j = j + 1$, and $x_j = \widetilde{x}_{j-1}$. Go to step 2.
6. Iterate until convergence.

- Take θ_j to be close to zero initially and increase as you go.
- You could sprinkle some BFGS after step 2 and let it simmer for a while!

Quasi-Random Numbers

- One could imagine that a better approach in the previous algorithm would be take the starting guesses on a Cartesian grid.
- But how to decide on how coarse or fine this grid should be? If x is 6 dimensional and you take 3 points in each direction, you need to start from $3^6 = 729$ different points. And who says 3 points is good enough?
- Random numbers have the advantage that you do not have to decide before hand how many restarts to do. Instead look at the improvement in objective value.
- But a disadvantage of random numbers is that... well, they are random! So they can accumulate in some areas and leave other areas empty.
- This is where quasi-random numbers come into play. They are **not** random, but they spread out maximally in a given space no matter how many of them are generated.

Uniform Random vs. Sobol' Numbers



Benchmarking Global Optimizers

- Most structural estimation/calibration problems with more than a few parameters require global optimization.
- The current approach taken by many is to use Nelder-Mead and restart it from several starting points. If they all converge to the same point it is taken as global optimum.
- But how many restarts are enough?
  - Consider a 10-dimensional objective. And suppose you take 1000 starting points. Is that enough?
  - If we were to constructs a hypergrid (Cartesian) and place 2 points along each axis, since $2^{10} = 1024$, you would get roughly 2 points in the domain of each parameter. This is puny.
  - And it is rare to take 1000 starting points anyway.
- So we need global optimizers as our initial choice. How to compare them?

Benchmarking Global Optimizers

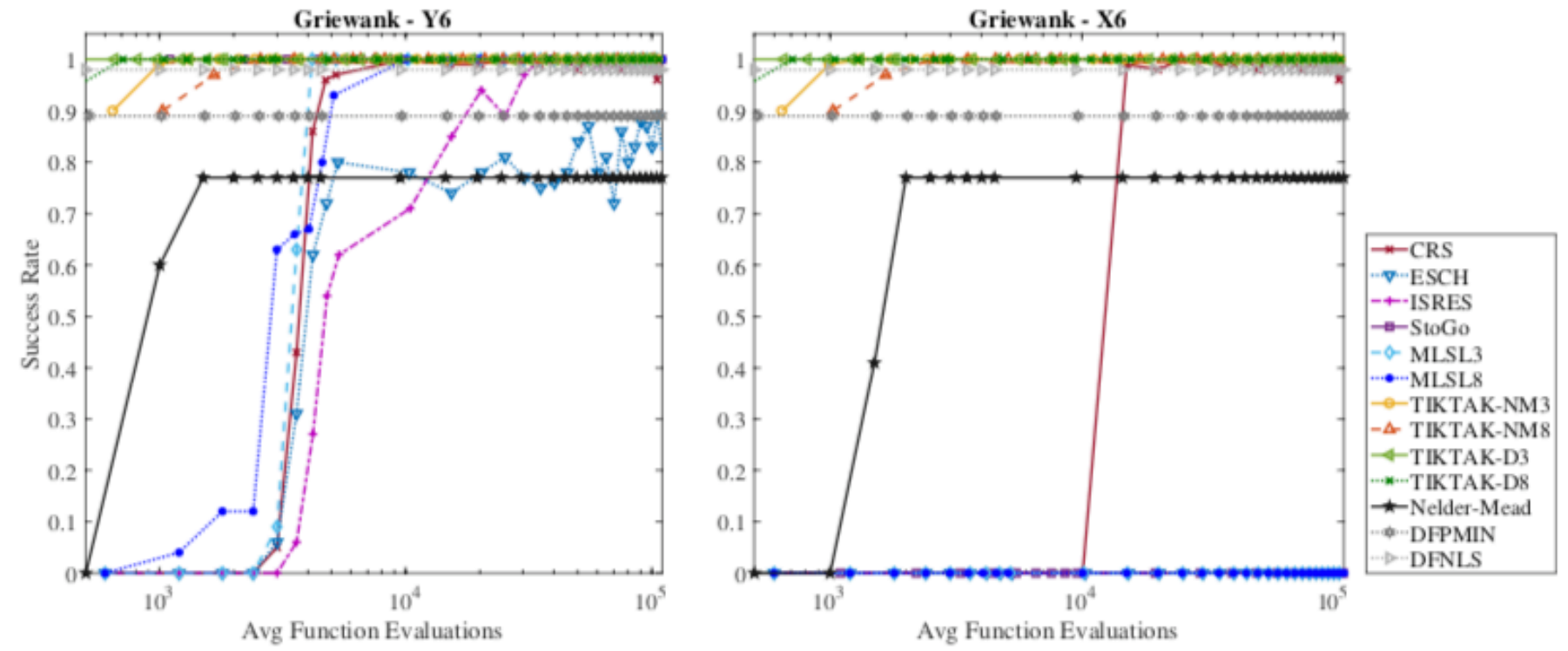
Results from Arnaud-Guvenen-Kleineberg (2019):

- Define "success" either as
  - function convergence to 10^{-6}
  - max deviation in x of 10^{-6}
  - Also analyze failures to see how badly they failed: e.g., did they stop at 10^{-5} or 10^{-1}?
- We will compare 4 versions of TikTak and 6 global optimizers from NLOPT suite. Several of them are award winners.
- We will also add local optimizers, like NM and DFPMIN.

Data Profile for Griewank Test Func.

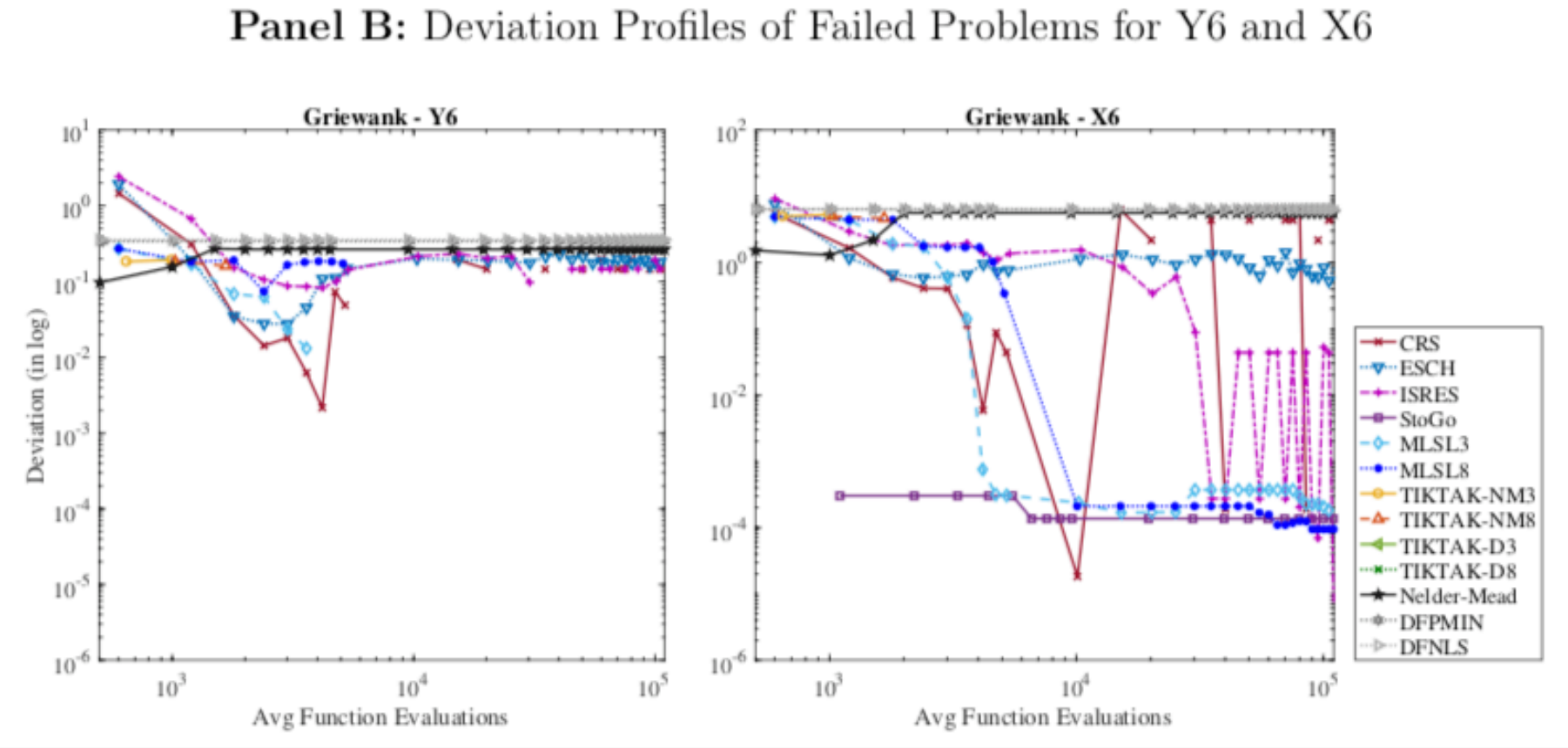
Lots of food for thought in the rankings. TikTak ranks top.

Some others are slow but with large budgets they can solve all problems.



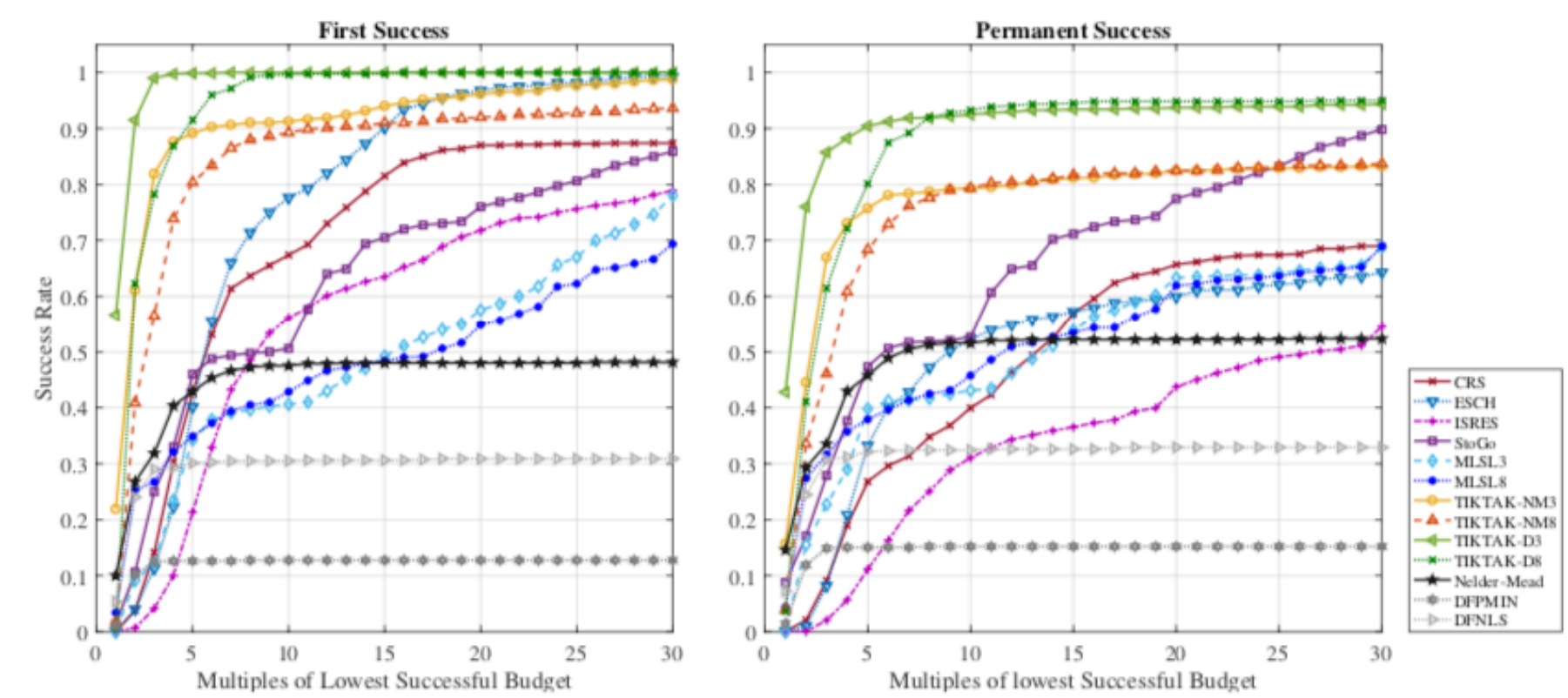
Deviations of Failed Attempts for Griewank

- Those that fail, fail a lot. Not always the case.
- For some test functions, many solvers get stucked at 10^{-4} or so. They can still be useful.



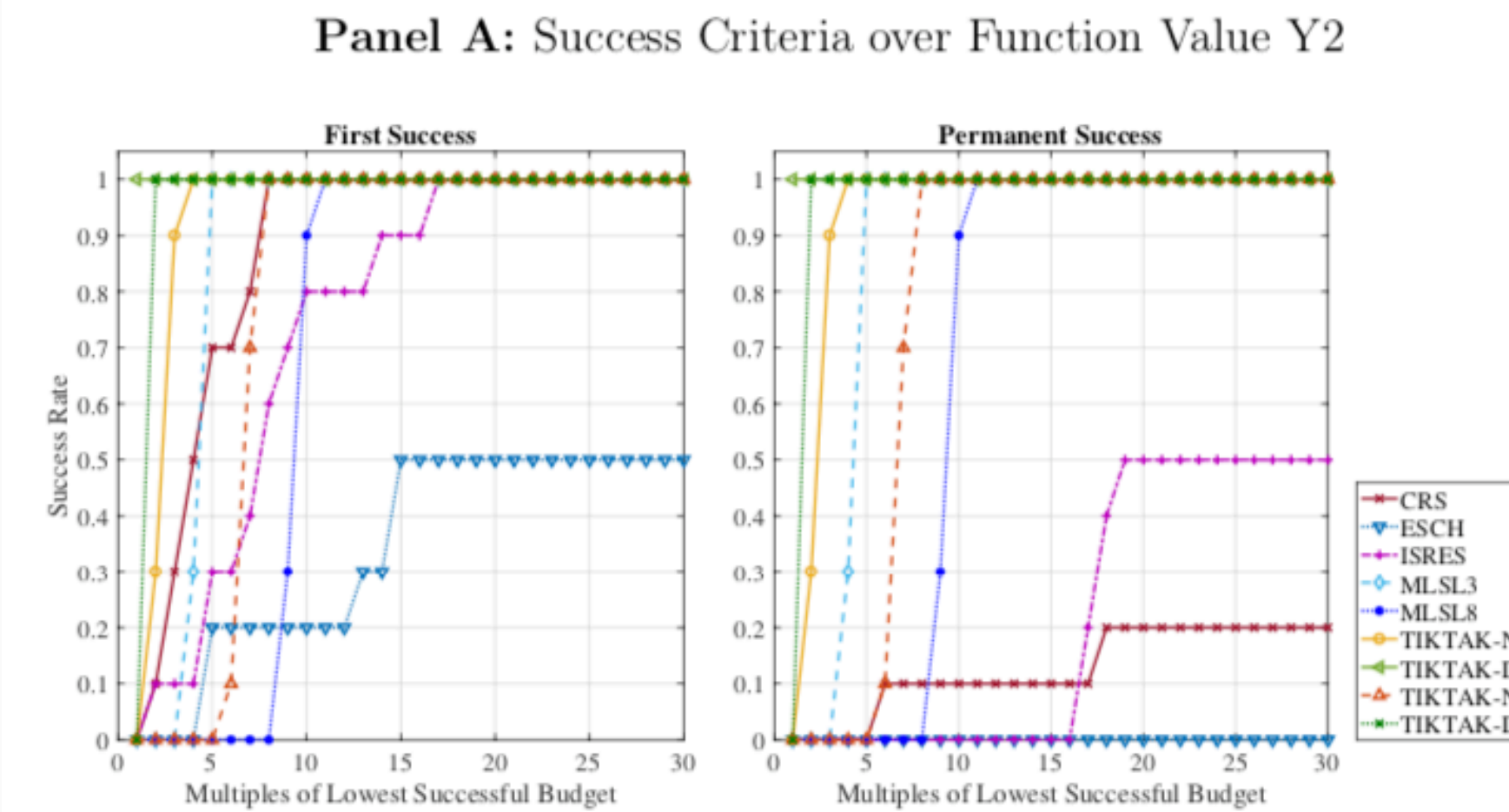
Data Profile for Griewank Test Func.

- Most useful plot. It tells us the worst case performance of each solver relative to others available.



Performance Profile: Income Dyn. Estimation

- Three versions of TikTak performs best. TikTak-NM8 is overkill because it uses the slow NM algorithm with very tight success criteria



A Little Parallel Programming

...Without knowing any parallel programming

- Ingredients you need:
  - Dropbox
  - Friends who will let you use their computers when they are asleep.
- Here is a modified version of my global algorithm that you can use with N computers.

A Little Parallel Programming

1. Generate an empty text file `myobjvals.txt` and put it into automatic sync across all machines using Dropbox.
2. Generate a large number of quasi-random numbers (say 1000).
3. Take the first N of these points and start your program on N machines, each with one of your quasi-random numbers as initial guess.
4. After Nelden-Mead converges on a given machine, write the minimum value found and the corresponding point to `myobjvals.txt`.
5. Before starting the next iteration open and read all objective values found so far (because of syncing this will be the minimum across all machines!)
6. Take your initial guess to the a linear combination of this best point and a new quasi-random number.
7. The rest of the algorithm is as before.