Lecture 7: Global Optimization

Fatih Guvenen

University of Minnesota

May 27, 2022

Global Optimization

Why Global Optimization?

- Calibration and structural estimation are multidimensional optimization problems.

Why Global Optimization?

- Calibration and structural estimation are multidimensional optimization problems.
- Except in very special cases, the objective function can feature kinks, jaggedness, ridges, and even jumps, and is likely to have a large number of local minima.

Why Global Optimization?

- Calibration and structural estimation are multidimensional optimization problems.
- Except in very special cases, the objective function can feature kinks, jaggedness, ridges, and even jumps, and is likely to have a large number of local minima.
- Local maximization algorithms are not the right tool to use. They get stuck in local minima—they are **not** looking for a global optimum.
  - True for Newton- (derivative-) based methods as well as DFNLS or Nelder-Mead type non-derivative based methods.

Why Global Optimization?

- Calibration and structural estimation are multidimensional optimization problems.
- Except in very special cases, the objective function can feature kinks, jaggedness, ridges, and even jumps, and is likely to have a large number of local minima.
- Local maximization algorithms are not the right tool to use. They get stuck in local minima—they are **not** looking for a global optimum.
  - True for Newton- (derivative-) based methods as well as DFNLS or Nelder-Mead type non-derivative based methods.
- $\therefore$ Using a good global optimizer is essential!

Why Global Optimization?

- Calibration and structural estimation are multidimensional optimization problems.
- Except in very special cases, the objective function can feature kinks, jaggedness, ridges, and even jumps, and is likely to have a large number of local minima.
- Local maximization algorithms are not the right tool to use. They get stuck in local minima—they are **not** looking for a global optimum.
  - True for Newton- (derivative-) based methods as well as DFNLS or Nelder-Mead type non-derivative based methods.
- $\therefore$ Using a good global optimizer is essential!
- A good option is TikTak, an algorithm I developed in collaboration with my coauthors on different papers.

Why Global Optimization?

- Calibration and structural estimation are multidimensional optimization problems.
- Except in very special cases, the objective function can feature kinks, jaggedness, ridges, and even jumps, and is likely to have a large number of local minima.
- Local maximization algorithms are not the right tool to use. They get stuck in local minima—they are **not** looking for a global optimum.
  - True for Newton- (derivative-) based methods as well as DFNLS or Nelder-Mead type non-derivative based methods.
- $\therefore$ Using a good global optimizer is essential!
- A good option is TikTak, an algorithm I developed in collaboration with my coauthors on different papers.
  - It is very fast & fully parallelizable without knowing MPI, OpenMP, CUDA, etc.. (see Arnaud, Guvenen, Kleineberg (2019))

Parallel Scaling Performance: Close to Linear!

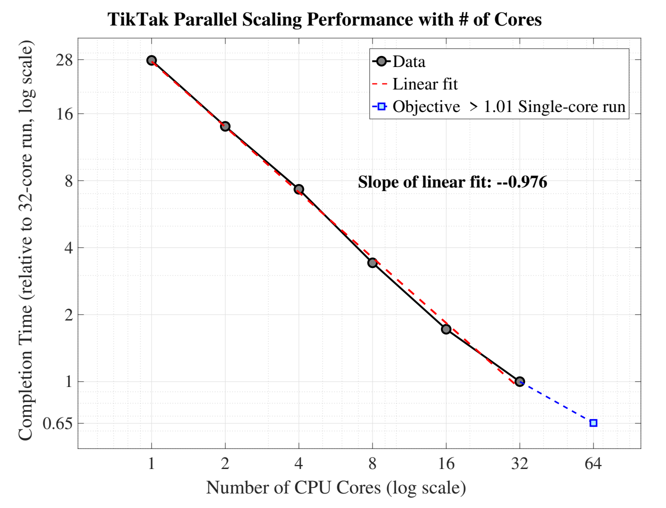
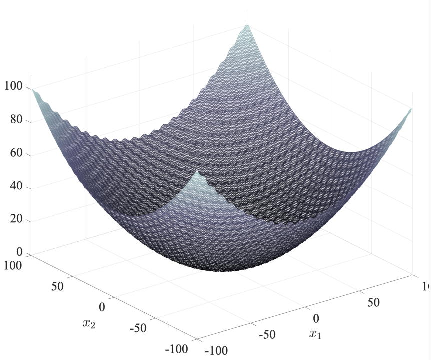
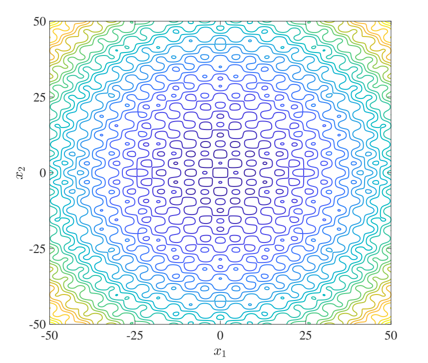


Figure 1: Griewank Function

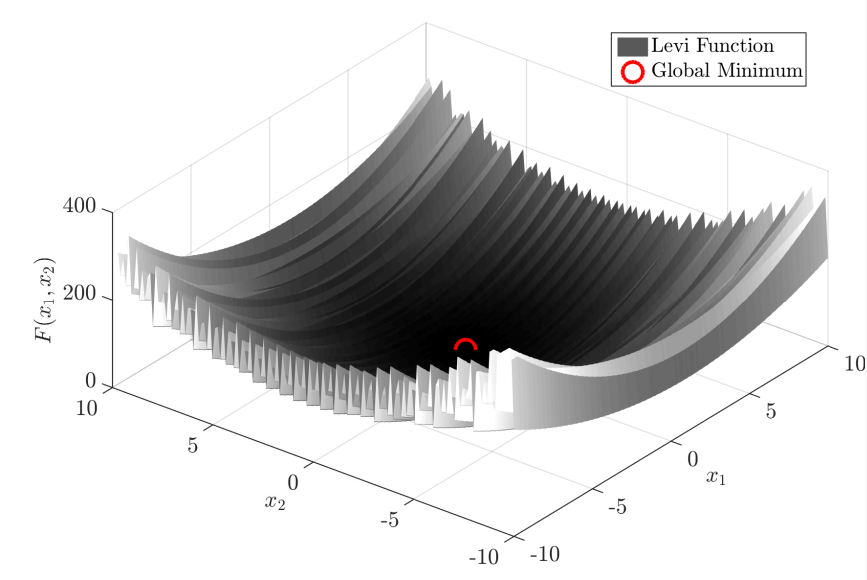


(a) 3-D Plot



(b) Contour Map

How Your OBJ Looks Like



OBJ in Hong and Chernozhukov (JE, 2003)

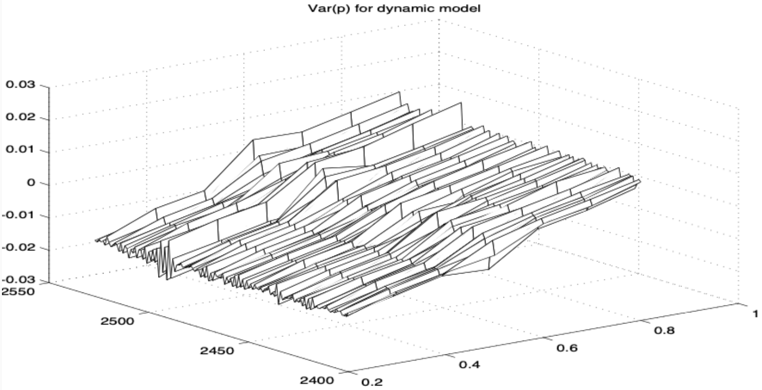


Figure 2: Recursive VaR Surface in time-probability space

Example: Rosenbrock Function

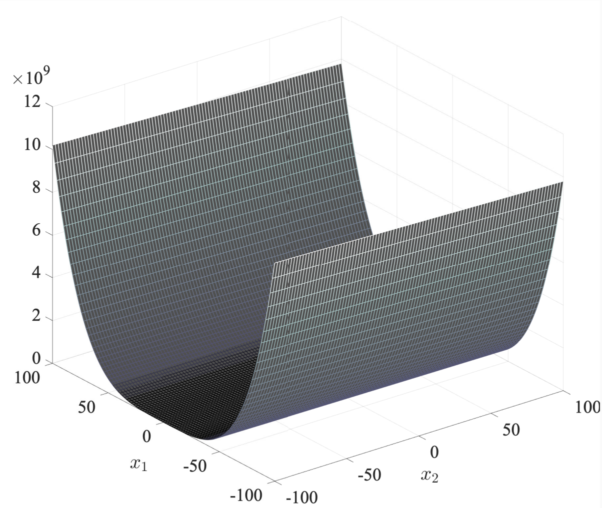
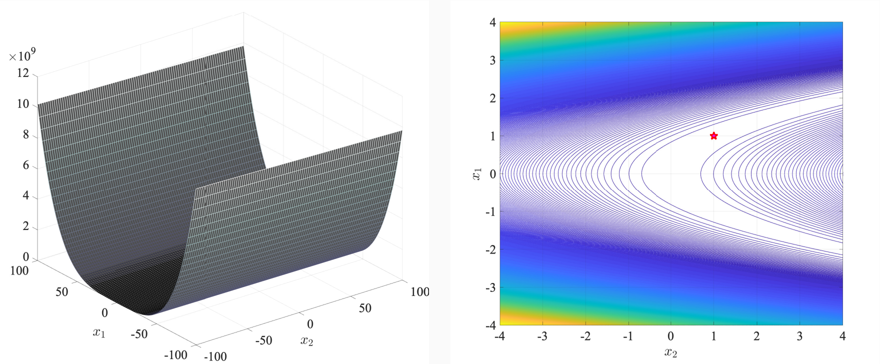


Figure 2: 3-D Plot

Example: Rosenbrock Function

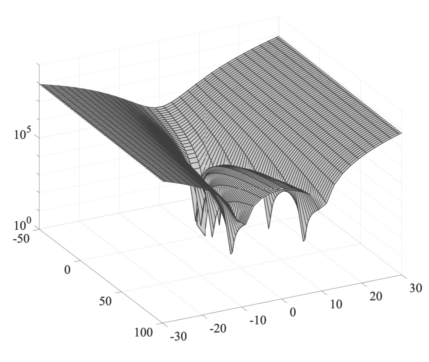


(a) 3-D Plot **(b)** Contour Map

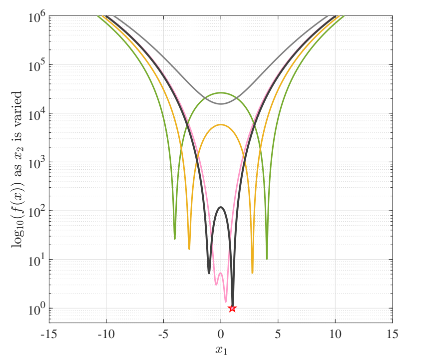
Note: The global optimum is marked with the red $*$ marker on the contour map.

Rosenbrock on Log Scale: Different Perspectives

Figure 3: See Arnaud, Guvenen, Kleineberg (2019) for more details



(a) Log Scale: Two Subtle Ridges

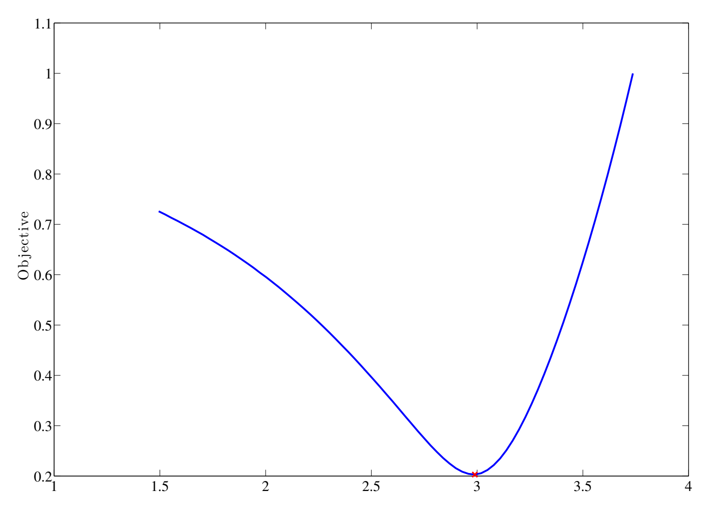


(b) Log Scale: Two Ridges Merge into One Near the Global Minimum

“Visualize” the Objective Surface

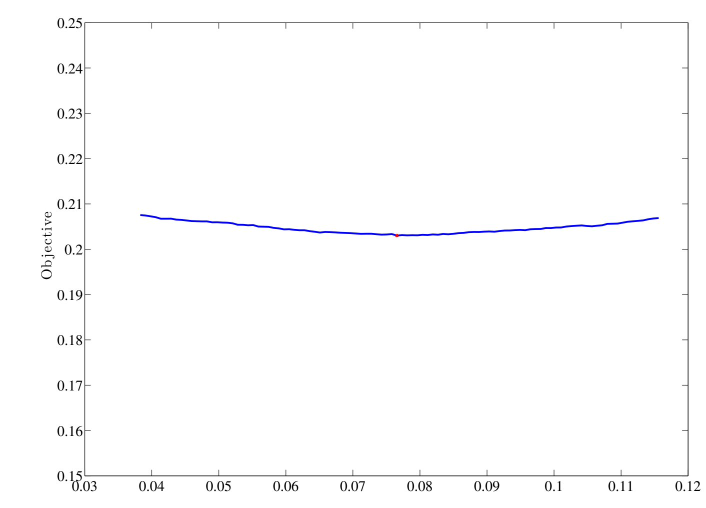
(Necessary but not sufficient!!)

Slicing the Objective in Guvenen et al (2021): Param 1

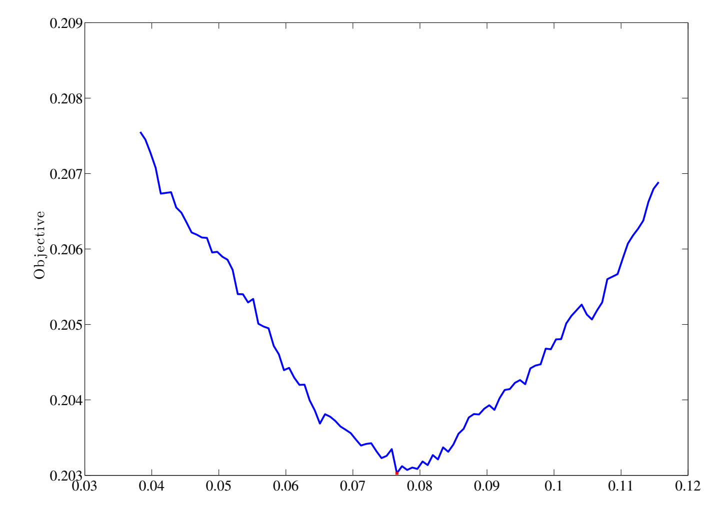


Source: All figures from Guvenen, Karahan, Ozkan, Song (2021).

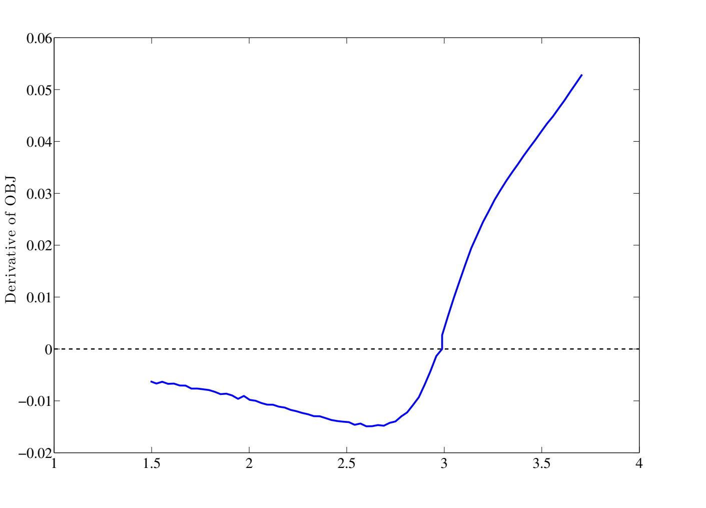
Slicing the Objective: Param 21



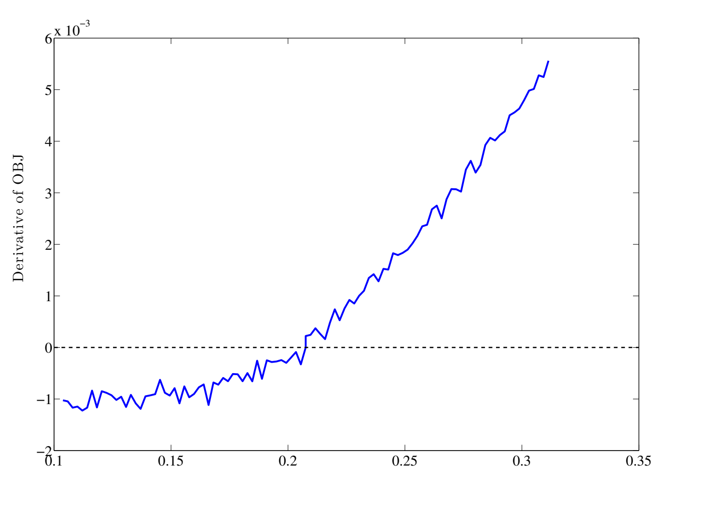
Slicing the Objective, Zooming in (y-axis): Param 21



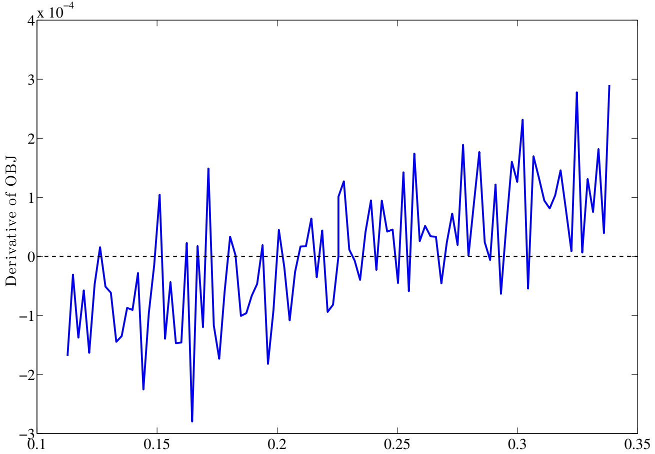
Plot the Derivative! Param 1



Plot the Derivative! Param 6



Plot the Derivative! Param 21



What are the sources of kinks, jaggedness, etc.

Sources from numerical methods used to solve structural model:

What are the sources of kinks, jaggedness, etc.

Sources from numerical methods used to solve structural model:

1. Linear interpolation generates an interpolated function that is non-differentiable (i.e., have kinks) at every knot point.

What are the sources of kinks, jaggedness, etc.

Sources from numerical methods used to solve structural model:

1. Linear interpolation generates an interpolated function that is non-differentiable (i.e., have kinks) at every knot point.
2. Discretizing the choice set (grid search) makes this problem much worse. (Avoid it all cost!)

What are the sources of kinks, jaggedness, etc.

Sources from numerical methods used to solve structural model:

1. Linear interpolation generates an interpolated function that is non-differentiable (i.e., have kinks) at every knot point.
2. Discretizing the choice set (grid search) makes this problem much worse. (Avoid it all cost!)
3. Problems with (borrowing, portfolio, etc.) constraints.

What are the sources of kinks, jaggedness, etc.

Sources from numerical methods used to solve structural model:

1. Linear interpolation generates an interpolated function that is non-differentiable (i.e., have kinks) at every knot point.
2. Discretizing the choice set (grid search) makes this problem much worse. (Avoid it all cost!)
3. Problems with (borrowing, portfolio, etc.) constraints.
4. Models with S-s type behavior (fixed costs, irreversibilities, discrete choice, etc) typically create jumps OBJ

What are the sources of kinks, jaggedness, etc.

Sources from numerical methods used to solve structural model:

1. Linear interpolation generates an interpolated function that is non-differentiable (i.e., have kinks) at every knot point.
2. Discretizing the choice set (grid search) makes this problem much worse. (Avoid it all cost!)
3. Problems with (borrowing, portfolio, etc.) constraints.
4. Models with S-s type behavior (fixed costs, irreversibilities, discrete choice, etc) typically create jumps OBJ
5. When moments are computed from simulated data, small changes in parameter values can move some individuals across threshold and cause jumps in OBJ.
    1. Compute moments using analytical formulas when possible–less susceptible to this problem.

What are the sources of kinks, jaggedness, etc.

6. The moments that we choose match could be inherently discontinuous in the underlying parameters:
   1. the *median* of a distribution (e.g., wealth holdings)
   2. or any percentile/quantile

   - This is one case where targeting central/standard moments (mean, variance, etc) can make sense. **But** only if their data counterpart is well estimated.

7. **Problems (1) to (6) are amplified** when moments are not jointly sufficiently informative about one or more parameters to be calibrated, because OBJ would be very flat in certain directions.

What are the sources of kinks, jaggedness, etc.

6. The moments that we choose match could be inherently discontinuous in the underlying parameters:
    1. the *median* of a distribution (e.g., wealth holdings)
    2. or any percentile/quantile

    - This is one case where targeting central/standard moments (mean, variance, etc) can make sense. **But** only if their data counterpart is well estimated.

7. **Problems (1) to (6) are amplified** when moments are not jointly sufficiently informative about one or more parameters to be calibrated, because OBJ would be very flat in certain directions.

*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.)

Global Optimization Algorithms

Global Optimization Algorithms

*Global Optimization

Definition 1
Let $f : A \to \mathbb{R}$ be a function on some set A. And suppose that

$$\exists! \quad m = \min_{x \in A} f(x)$$

be the unique global minimum point of $f()$ in A, with the associated global minimizer $x*$.

- Construct a sequence of points, $x_1, x_2, ...$ in A such that the sequence of values $y_n = \min_{i=1,..,n} f(x_i)$ approaches the minimum m as n increases.
- y_n is called the record (there is an entire set of tools associated with records and their use).

*Algorithm

1. Generate a random point x_1 according to a probability distribution P_1 on A; evaluate $f(x_1)$; set iteration number $j = 1$.
2. Using the points $x_1, x_2, ...x_j$ and the results of objective function evaluation at these points, check whether $j = n$; that is check if an appropriate stopping condition holds and terminate if yes. If no, continue
3. Generate x_{j+1}according to some probability distribution P_{j+1} and evaluate $f(x_{j+1})$
4. Substitute $j + 1$ for j and return to step 2.

*Different Approaches

1. Pure random search: All distributions $P_j$are the same and the points x_j are independent.
2. Markovian algorithms: P_{j+1} depends only on x_j and $f(x_j)$.
3. More general algorithms: Update P_j after a certain number of points have been evaluated and based on past search information.

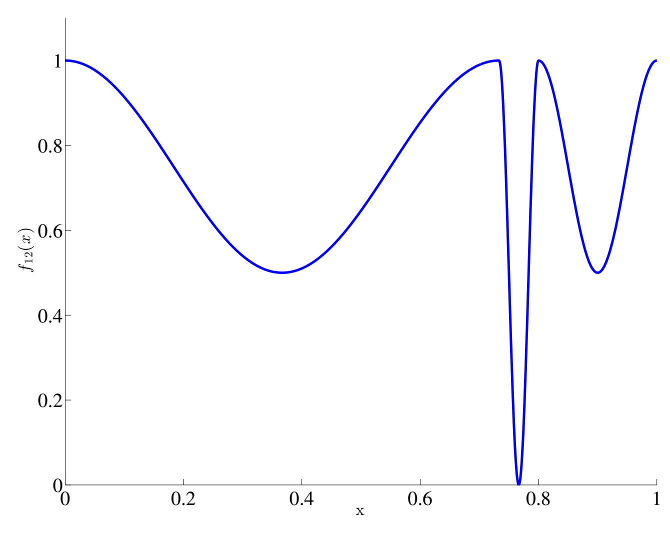
*Regularity Conditions?

- With local optimization, assumptions on f that guarantee continuity or differentiability are useful for convergence.
- With global algorithms, they are a lot less useful.
- For example, consider:

$$f_k(x) = \begin{cases} 1 - \frac{1}{2}\left(\sin \frac{5k\pi x}{4(k-1)}\right)^2 & \text{for } x \in [0, \frac{4(k-1)}{5k}] \\ 1 - \left(\sin \frac{5k\pi x}{4}\right)^2 & \text{for } x \in [\frac{4(k-1)}{5k}, \frac{4}{5}] \\ 1 - \frac{1}{2}\left(\sin 5\pi x\right)^2 & \text{for } x \in [\frac{4}{5}, 1] \end{cases}$$

where $k \geq 2$ is an integer.

*Regularity Conditions



TIKTAK: An Asynchronously Parallelizable Global Optimization Algorithm

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

1. Set $j = 0$ and start the iteration.

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

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.

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

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 (e.g., *Sobol*'s sequence. More on this in a minute).

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

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 (e.g., *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 "record" as of iteration j.

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

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 (e.g., *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 "record" as of iteration j.
5. Update $j = j + 1$, and $x_j = \widetilde{x}_{j-1}$. Go to step 2.
6. Iterate until convergence.

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

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 (e.g., *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 “record” as of 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.

TikTak: A Global Optimization Algorithm

Basic Outline of Algorithm:

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 (e.g., *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 "record" as of 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.

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?

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.

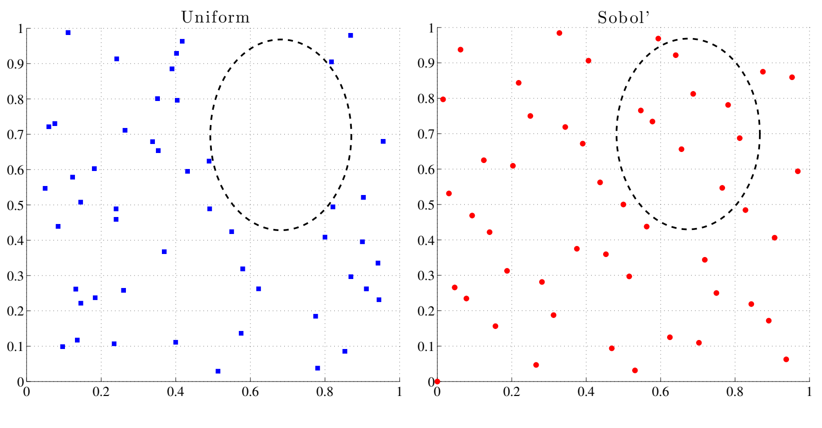
Quasi-Random Numbers

- 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.

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

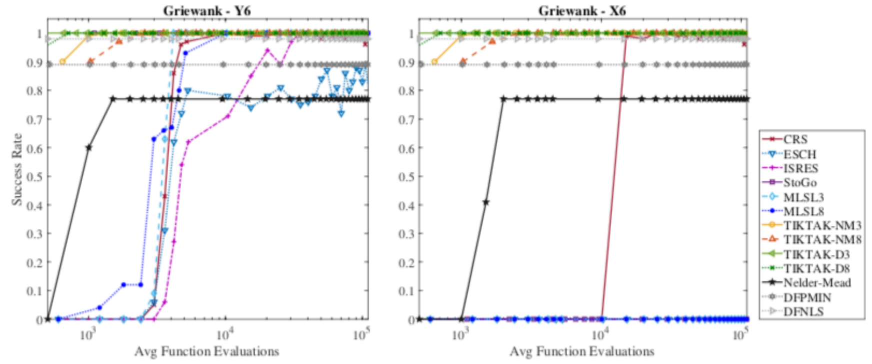
Results from Arnaud-Guvenen-Kleineberg (2019):

- 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.

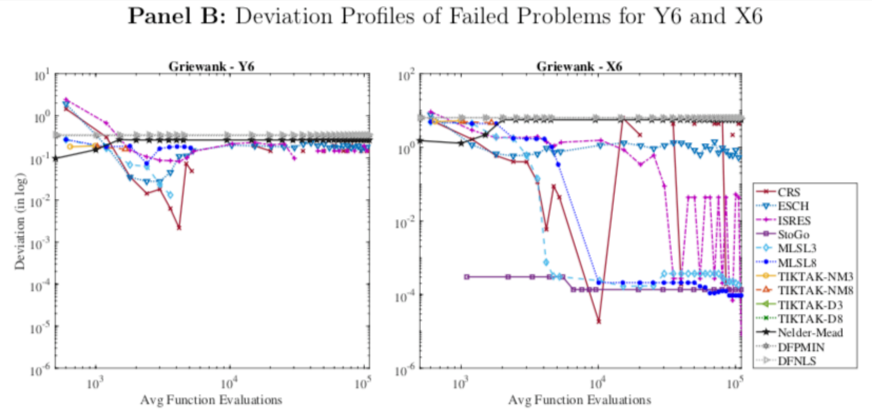
Lots of food for thought in the rankings. TikTak ranks top.

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



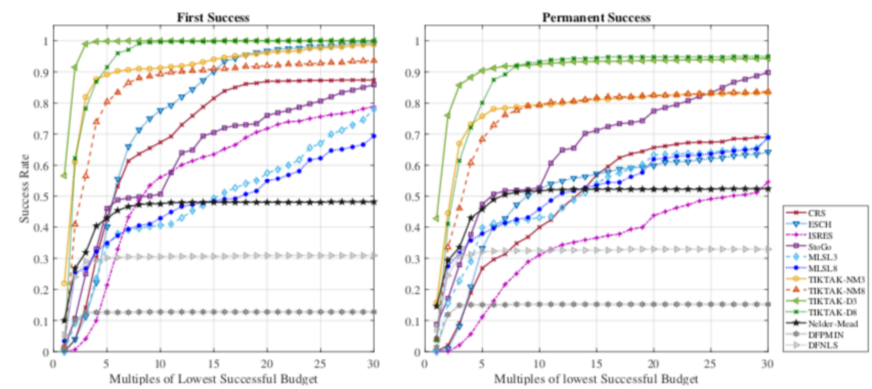
Deviations of Failed Attempts for Griewank

- 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.



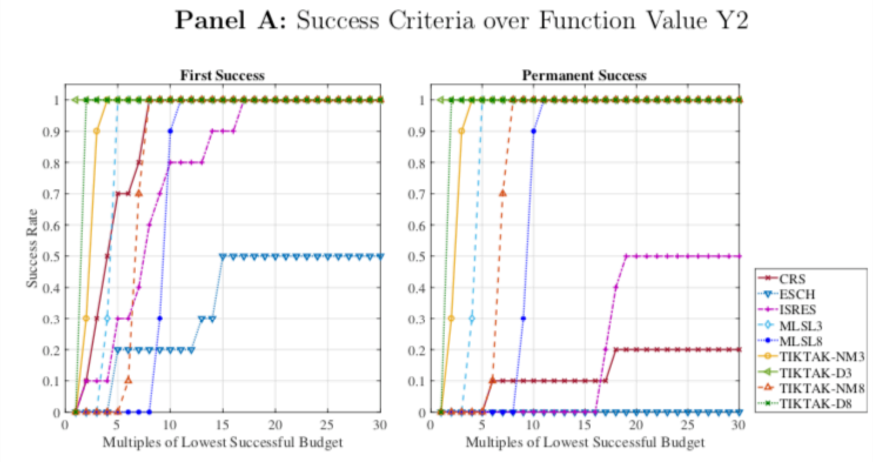
Data Profile for Griewank Test Func.

- 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

A Little Parallel Programming

...Without knowing any parallel programming

- Ingredients you need:

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.

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.

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).

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.

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`.

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!)

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.

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.

Parallel Implementation Posted on Github

- Go to Serdar Ozkan's Github:

  https://github.com/serdarozkan/TikTak#tiktak

- It has all the info and the codes you need to run.
- The version on Github is more efficient than the one in the "Benchmarking" paper.
- How many cores can you parallelize over? Further work needed.
- My rule of thumbs: $\#\text{of cores} \leq \sqrt{\#\text{local restarts}}$
- The following picture says it works pretty well.

Parallel Scaling Performance: Close to Linear!