Lecture 6: Local Optimization

Fatih Guvenen

May 27, 2022

Optimization

Overview of Optimization

- Among many uses, you will need it for:
  - solving a dynamic programming problem.
  - root-finding as a minimization problem (discussed earlier)$\rightarrow$ solving for GE.
  - estimation or calibration by matching moments.

Overview of Optimization

- Among many uses, you will need it for:
  - solving a dynamic programming problem.
  - root-finding as a minimization problem (discussed earlier)$\rightarrow$ solving for GE.
  - estimation or calibration by matching moments.

Two main trade-offs:

- Fast local methods versus slow but more global methods.
- Whether to calculate derivatives (esp. Jacobians/Hessians in multidimensional case!).

Overview of Optimization

- Among many uses, you will need it for:
  - solving a dynamic programming problem.
  - root-finding as a minimization problem (discussed earlier)$\rightarrow$ solving for GE.
  - estimation or calibration by matching moments.

Two main trade-offs:

- Fast local methods versus slow but more global methods.
- Whether to calculate derivatives (esp. Jacobians/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: $f(a), f(c) > f(b)$.

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: $(c_{min} = \epsilon, c_{max} = y - a_{min})$.

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.

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.
- For one-dimensional problems my default choice is Brent's method (e.g. NR's `brent.f90`).
  - It always brackets a minimum and is very fast.
  - There is a version that uses derivatives that's a bit faster (NR's `dbrent.f90`). It can be faster but not as reliable with objectives that are not super smooth.

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.
- For one-dimensional problems my default choice is Brent's method (e.g. NR's `brent.f90`).
  - It always brackets a minimum and is very fast.
  - There is a version that uses derivatives that's a bit faster (NR's `dbrent.f90`). It can be faster but not as reliable with objectives that are not super smooth.
- Newton's method has very poor global convergence properties. Never use it alone!

Multi-Dimensional Optimization

- Multi-dimensional optimization can be a very hard problem because:

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.

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

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.

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.

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.

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:

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!

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.

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. BOBYQA + 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.

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. BOBYQA + 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.

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)

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 $f(\mathsf{P} + \lambda \mathsf{n})$

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 $f(\mathbf{P} + \lambda\mathbf{n})$
- Once you are at this line minimum, call **P**+λ**n**, the key step is to decide what direction to move next.

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 $f(\mathbf{P} + \lambda \mathbf{n})$
- 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).

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!

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).

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

title style=at=(0.75,1.4)

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

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.

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

- 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.

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

- 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.

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").

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 $s \in S$.

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:

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

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:

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

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

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:

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

- $t_{p,s}$ 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,

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

where $|P|$ denotes the cardinality of 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,

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

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

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

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,

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

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.

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,

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

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

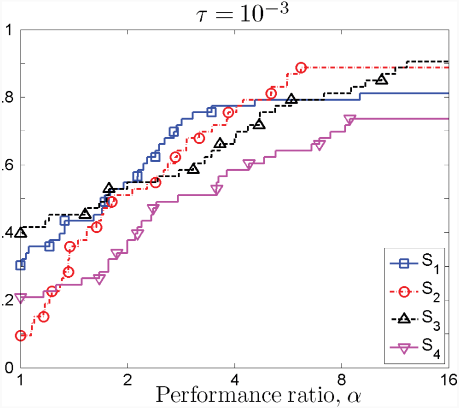

FIG. 2.1. *Sample performance profile $\rho_s(\alpha)$ (logarithmic scale) for derivative-free solvers.*

Data Profiles

- Oftentimes, we are interested in the percentage of problems that can be solved (for a given τ) with μ_f function evaluations.

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

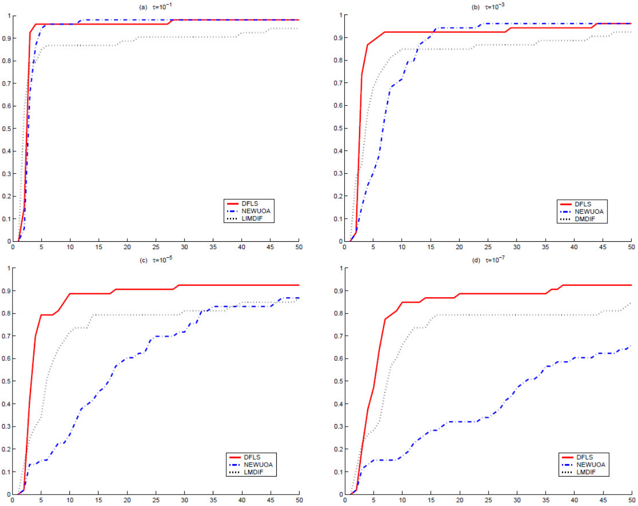

FIG. 5.1. *Data profiles of function* $d_s(\alpha)$ *for smooth problems: (a)* $\tau = 10^{-1}$, *(b)* $\tau = 10^{-3}$, *(c)* $\tau = 10^{-5}$, *(d)* $\tau = 10^{-7}$