**Calibration and Estimation: Avoid Local Search**

Economists often need to minimize an objective function of multiple variables that has lots of kinks, jaggedness, and deep ridges. Consequently, the global minimum is often surrounded by a large number of local minima. A typical example of such a problem arises when a researcher tries to calibrate several structural parameters of an economic model by matching some data moments. Algorithms based on local optimization methods (e.g., Newton-Raphson style derivative-based methods or Nelder-Mead simplex style routines) very often get stuck in local minima because the objective surface is typically very rough (non-smooth).

It is useful to understand some of the sources of this roughness. For example, linear interpolation that is often used in approximating value functions or decision rules generates an interpolated function that is non-differentiable (i.e., has kinks) at every knot point. Similarly, problems with (borrowing, portfolio, etc.) constraints can create significant kinks. Because researchers use a finite number of individuals to simulate data from the model (to compute moments), a small change in the parameter value (during the minimization of the objective) can move some individuals across the threshold—from being constrained to unconstrained or vice versa—which can cause small jumps in the objective value. And sometimes, the moments that the researcher decides to match would be inherently discontinuous in the underlying parameters (with a finite number of individuals), such as the median of a distribution (e.g., wealth holdings). Further compounding the problems, if the moments are not jointly sufficiently informative about the parameters to be calibrated, the objective function would be flat in certain directions. As can be expected, trying to minimize a relatively flat function with lots of kinks, jaggedness, and even small jumps can be a very difficult task indeed.[47]

While the algorithm described here can be applied to the calibration of any model, it is especially useful in models with heterogeneous agents—since such models are time consuming to solve even once, an exhaustive search of the parameter space becomes prohibitively costly (which could be feasible in simpler models).

[47] One simple, but sometimes overlooked, point is that when minimizing an objective function of moments to calibrate a model, one should use the same "seeds" for the random elements of the model that are used to simulate the model in successive evaluations of the objective function. Otherwise, some of the change in objective value will be because of the inherent randomness in different draws of random variables. This can create significant problems with the minimization procedure.

## A Simple Fully Parallelizable Global Optimization Algorithm

Here, I describe a global optimization algorithm that I regularly use for calibrating models and I have found it to be very practical and powerful. It is relatively straightforward to implement, yet allows full parallelization across any number of central processing unit (CPU) cores as well as across any number of computers that are connected to the Internet. It requires no knowledge of MPI, OpenMP, or related tools, and no knowledge of computer networking other than using some commercially available synchronization tools (such as DropBox, SugarSync, etc.).

A broad outline of the algorithm is as follows. As with many global algorithms, this procedure combines a global stage with a local search stage that is restarted at various locations in the parameter space. First, we would like to search the parameter space as thoroughly as possible, but do so in as efficient a way as possible. Thoroughness is essential because we want to be sure that we found the true global minimum, so we are willing to sacrifice some speed to ensure this. The algorithm proceeds by taking an initial starting point (chosen in a manner described momentarily) and conducting a local search from that point on until the minimization routine converges as specified by some tolerance. For local search, I typically rely on the Nelder-Mead's downhill simplex algorithm because it does not require derivative information (that may be inaccurate given the approximation errors in the model's solution algorithm).[48] The minimum function value as well as the parameter combination that attained that minimum are recorded in a file saved on the computer's hard disk. The algorithm then picks the next "random" starting point and repeats the previous step of local minimization. The results are then added to the previous file, which records all the local minima found up to that point.

Of course, the most obvious algorithm would be to keep doing a very large number of restarts of this sort and take the minimum of all the minima found in the process. But this would be very time consuming and would not be particularly efficient. Moreover, in many cases, the neighborhood of the global minimum can feature many deep ridges and kinks nearby, which requires more extensive searching near the global minimum, whereas the proposed approach would devote more time to points far away from the true global minimum and to the points near it. Further, if the starting points are chosen literally randomly, this would also create potentially large efficiency losses, because these points have a non-negligible chance of falling near points previously tried. Because those areas have been previously searched, devoting more time is not optimal.

[48] An alternative that can be much faster but requires a bit more tweaking for best performance is the trust region method of Zhang, Conn, and Scheinberg (2010) that builds on Powell's (2009) BOBYQA algorithm.

A better approach is to use "quasi-random" numbers to generate the starting points. Quasi-random numbers (also called low-discrepancy sequences) are sequences of deterministic numbers that spread to any space in the maximally separated way. They avoid the pitfall of random draws that may end up being too close to each other. Each draw in the sequence "knows" the location of previous points drawn and attempts to fill the gaps as evenly as possible.[49] Among a variety of sequences proposed in the literature, the Sobol' sequence is generally viewed to be superior in most practical applications, having a very uniform filling of the space (i.e., maximally separated) even when a small number of points is drawn, as well as a very fast algorithm that generates the sequence.[50]

Next, how do we use the accumulated information from previous restarts? As suggested by genetic algorithm heuristics, I combine information from previous best runs to adaptively direct the new restarts to areas that appear more promising. This is explained further below. Now for the specifics of the algorithm.

**Algorithm 1** *Let* $\boldsymbol{p}$ *be a* $J$*-dimensional parameter vector with generic element* $p^j$, $j = 1, ..., J$.

- *Step 0.* ***Initialization****:*
  - *Determine bounds for each parameter, outside of which the objective function should be set to a high value.*
  - *Generate a sequence of Sobol' numbers with a sequence length of* $I_{max}$ *(the maximum anticipated number of restarts in the global stage). Set the global iteration number* $i = 1$.
- *Step 1.* ***Global Stage****:*
  - *Draw the* $i^{th}$ *(vector) value in the Sobol' sequence:* $\boldsymbol{s}_i$.
  - *If* $i > 1$*, open and read from the text file* "saved_parameters.dat" *the function values (and corresponding parameter vectors) of previously found local minima. Denote the lowest function value found as of iteration* $i-1$ *as* $f_{i-1}^{low}$ *and the corresponding parameter vector as* $\boldsymbol{p}_{i-1}^{low}$.
  - *Generate a starting point for the local stage as follows:*

[49] Another common application of low-discrepancy sequences is in quasi-Monte Carlo integration, where they have been found to improve time-to-accuracy by several orders of magnitude.

[50] In a wide range of optimization problems, Kucherenko and Sytsko (2005) and Liberti and Kucherenko (2005) find that Sobol' sequences outperform Holton sequences, both in terms of computation time and probability of finding the global optimum. The Holton sequence is particularly weak in high dimensional applications.

  - *If $i < I_{\min}(< I_{\max})$, then use $\mathbf{s}_i$ as the initial guess: $\mathbf{S}_i = \mathbf{s}_i$. Here, $I_{\min}$ is the threshold below which we use fully quasi-random starting points in the global stage.*
  - *If $i \geq I_{\min}$, take the initial guess to be a convex combination of $\mathbf{s}_i$ and the parameter value that generated the best local minima so far: $\mathbf{S}_i = (1-\theta_i)\mathbf{s}_i + \theta \mathbf{p}_{i-1}^{low}$. The parameter $\theta_i \in [0, \bar{\theta}]$ with $\bar{\theta} < 1$, and increases with $i$. For example, I found that a convex increasing function, such as $\theta_i = \min[\bar{\theta}, (i/I_{\max})^2]$, works well in some applications. An alternative heuristic is given later.*
  - *As $\theta_i$ is increased, local searches are restarted from a narrower part of the parameter space that yielded the lowest local minima before.*

- *Step 2:* ***Local Stage****:*
  - *Using $\mathbf{S}_i$ as a starting point, use the downhill simplex algorithm to search for a local minimum. (For the other vertices of the simplex, randomly draw starting points within the bounds of the parameter space.)*
  - *Stop when either (i) a certain tolerance has been achieved, (ii) function values do not improve more than a certain amount, or (iii) the maximum iteration number is reached.*
  - *Open* saved_parameters.dat *and record the local minimum found (function value and parameters).*

- *Step 3.* ***Stopping Rule****:*
  - *Stop if the termination criterion described below is satisfied. If not go to Step 1.*

***Termination Criterion***

One useful heuristic criterion relies on a Bayesian procedure that estimates the probability that the next local search will find a new local minimum based on the rate at which new local minima have been located in past searches. More concretely, if $W$ different local minima have been found after $K$ local searches started from a set of uniformly distributed points, then the expectation of the number of local minima is

$$W_{\exp} = W\,(K-1)\,/\,(K-W-2)\,,$$

provided that $K > W+2$. The searching procedure is terminated if $W_{\exp} < W+0.5$. The idea is that, after a while of searching, if subsequent restarts keep

finding one of the same local minima found before, the chances of improvement in subsequent searches is not worth the additional time cost. Although this is generally viewed as one of the most reliable heuristics, care must be applied as with any heuristic.

Notice also that $W_{\text{exp}}$ can be used to adaptively increase the value of $\theta_i$ in the global stage (Step 1 [3] above). The idea is that, as subsequent global restarts do not yield a new local minimum with a high enough probability, it is time to narrow the search and further explore areas of promising local minima. Because jaggedness and deep ridges cause local search methods to often get stuck, we want to explore promising areas more thoroughly.

One can improve on this basic algorithm in various ways. I am going to mention a few that seem worth exploring.

***Refinements: Clustering and Pre-Testing***

First, suppose that in iteration $k$, the proposed starting point $\mathbf{S}_k$ ends up being "close" to one of the previous minima, say $\mathbf{p}_n^{low}$, for $n < k$. Then it is likely that the search starting from $\mathbf{S}_k$ will end up converging to $\mathbf{p}_n^{low}$. But then we have wasted an entire cycle of local search without gaining anything. To prevent this, one heuristic (called "clustering methods") proceeds by defining a "region of attraction" (which is essentially a $J$-dimensional ball centered) around each one of the local minima found so far.[51] Then the algorithm would discard a proposed restarting point if it falls into the region attraction of any previous local minima. Because the local minimization stage is the most computationally intensive step, this refinement of restarting the local search only once in a given region of attraction can result in significant computational gains. Extensive surveys of clustering methods can be found in Rinnooy Kan and Timmer (1987a, 1987b).

Second, one can add a "pre-test" stage where $N$ points from the Sobol' sequence are evaluated before any local search (i.e., in Step 0 above), and only a subset of $N^* < N$ points with lowest objective values are used as seeds in the local search. The remaining points, as well as regions of attraction around them are ignored as not promising. Notice that while this stage can improve speed, it trades off reliability in the process.

***Narrowing Down the Search Area***

The file `saved_parameters.dat` contains a lot of useful information gathered in each iteration to the global stage, which can be used more efficiently as follows. As noted, the Nelder-Mead algorithm requires $J + 1$ candidate

[51] While different formulas have been proposed for determining the optimal radius, these formulas contain some undetermined coefficients, making the formulas less than useful in real life applications.

points as inputs (the vertices of the $J$-dimensional simplex). One of these points is given by $\mathbf{S}_i$, chosen as described above; the other vertices were drawn randomly. But as we accumulate more information with every iteration on the global stage, if we keep finding local minima that seem to concentrate in certain regions, it makes sense to narrow the range of values from which we pick the vertices. One way to do this is as follows: After a sufficiently large number of restarts have been completed, rank all the function values and take the lowest $x$ percent of values (e.g., 10 percent or 20 percent). Then for each dimension, pick the minimum ($p^j_{min}$) and maximum parameter value ($p^j_{max}$) within this set of minima. Then, to generate vertices, take randomly sampled points between $p^j_{min}$ and $p^j_{max}$ in each dimension $j$. This allows the simplex algorithm to search more intensively in a narrower area, which can improve results very quickly when there are ridges or jaggedness in the objective function that make the algorithm get stuck.

**Parallelizing the Algorithm**

The algorithm can be parallelized in a relatively straightforward manner.[52] The basic idea is to let each CPU core perform a separate local search in a different part of the parameter space, which is a time-consuming process. If we can do many such searches simultaneously, we can speed up the solution dramatically. One factor that makes parallelization simple is the fact that the CPU cores do not need to communicate with each other during the local search stage. In between the local stages, each CPU core will contribute its findings (the last local minimum it found along with the corresponding parameter vector) to the collective wisdom recorded in `saved_parameters.dat` and also get the latest updated information about the best local minimum found so far from the same file. Thus, as long as all CPU cores have access to the same copy of the file `saved_parameters.dat`, parallelization requires no more than a few lines for housekeeping across CPUs. Here are some more specifics.

Suppose that we have a workstation with $N$ CPU cores (for example, $N = 4$, 6, or 12). The first modification we need to make is to change the program to distinguish between the different "copies" of the code, running on different CPU cores. This can be done by simply having the program ask the user (only once, upon starting the code) to input an integer value, $n$, between 1 and $N$, which uniquely identifies the "sequence number" of the particular instance of the program running. Then open $N$ terminal windows and launch a copy of the program in each window. Then for each one, enter a unique sequence number $n = 1, 2, ..., N$.

[52] I am assuming here that a compiled language, such as Fortran or C, is used to write the program. So multiple parallel copies of the same code can be run in different terminal windows.

Upon starting, each program will first simulate the same quasi-random sequence regardless of $n$, but each run will pick a different element of this sequence as its own seed. For simplicity, suppose run $n$ chooses the $n$th element of the sequence as its seed and launches a local search from that point. After completion, each run will open the same file `saved_parameters.dat` and record the local minimum and parameter value it finds.[53]

Now suppose that all copies of the program complete their respective first local searches, so there are $N$ lines, each written by a different CPU core, in the file `saved_parameters.dat`. Then each run will start its second iteration and pick as its next seed the $(N+n)$th element of the quasi-random sequence. When the total number of iterations across all CPUs exceed some threshold $I_{\min}$, then we would like to combine the quasi-random draw with the previous best local minima as described in Step 1 (3) above. This is simple since all runs have access to the same copy of `saved_parameters.dat`.[54]

Notice that this parallelization method is completely agnostic about whether the CPU cores are on the same personal computer (PC) or distributed across many PCs *as long as* all computers keep synchronized copies of `saved_parameters.dat`. This can be achieved by using a synchronization service like DropBox. This feature easily allows one to harness the computational power of many idle PCs distributed geographically with varying speeds and CPU cores.

## 8. FUTURE DIRECTIONS AND FURTHER READING

This article surveys the current state of the heterogeneous-agent models literature and draws several conclusions. First, two key ingredients in such models are (i) the magnitudes and types of risk that the model builder feeds into the model and (ii) the insurance opportunities allowed in the economy. In many cases, it is difficult, if not impossible, to measure each component separately. In other words, the assumptions a researcher makes regarding insurance opportunities will typically affect the inference drawn about the magnitudes of risks and vice versa. Further complicating the problem is the measurement of risk: Individuals often have more information than the econometrician about

[53] Because this opening and writing stage takes a fraction of a second, the likelihood that two or more programs access the file simultaneously and create a run-time error is negligible.

[54] It is often useful for each run to keep track of the *total* number of local searches completed by all CPUs—call this $N_{Last}$. For example, sometimes the increase in $\theta_i$ can be linked to $N_{Last}$. This number can be read as the total number of lines recorded up to that point in `saved_parameters.dat`. Another use of this index is for determining which point in the sequence to select as the next seed point. So as opposed to running $n$ by selecting the $(kN+n)$th point in the sequence where $k$ is the number of local searches completed by run $n$, it could just pick the $(N_{Last}+1)$th number in the sequence. This avoids leaving gaps in the sequence for seeds, in case some CPUs are much faster than others and hence finish many more local searches than others.

future changes in their lives. So, for example, a rise or fall in income that the econometrician may view as a "shock" may in fact be partially or completely anticipated by the individual. This suggests that equating income movements observed in the data with risk (as is often done in the literature) is likely to overstate the true magnitude. This entanglement of "risk," "anticipated changes," and "insurance" presents a difficult challenge to researchers in this area. Although some recent progress has been made, more work remains.

A number of surveys contain very valuable material that are complementary to this article. First, Heathcote, Storesletten, and Violante (2009) is a recent survey of quantitative macroeconomics with heterogeneous households that is complementary to this article. Second, Browning, Hansen, and Heckman (1999) contains an extensive review of microeconomic models that are often used as the foundations of heterogeneous-agent models. It highlights several pitfalls in trying to calibrate macroeconomic models using microevidence. Third, Meghir and Pistaferri (2011) provides a comprehensive treatment of how earnings dynamics affect life-cycle consumption choice, which is closely related to the issues discussed in Section 3 of this survey. Finally, because heterogeneous-agent models use microeconomic survey data in increasingly sophisticated ways, a solid understanding of issues related to measurement error (which is pervasive in microdata) is essential. Failure to understand such problems can wreak havoc with the empirical analysis. Bound, Brown, and Mathiowetz (2001) is an extensive and authoritative survey of the subject.

The introduction of families into incomplete markets models represents an exciting area of current research. For many questions of empirical relevance, the interactions taking place within a household (implicit insurance, bargaining, etc.) can have first-order effects on how individuals respond to idiosyncratic changes. To give a few examples, Gallipoli and Turner (2011) document that the labor supply responses to disability shocks of single workers are larger and more persistent than those of married workers. They argue that an important part of this difference has to do with the fact that couples are able to optimally change their time (and task) allocation within households in response to disability, an option not available to singles. This finding suggests that modeling households would be important for understanding the design of disability insurance policies. Similarly, Guner, Kaygusuz, and Ventura (2010) show that to quantify the effects of alternative tax reforms, it is important to take into account the joint nature of household labor supply. In fact, it is hard to imagine any distributional issue for which the household structure does not figure in an important way.

Another promising area is the richer modeling of household finances in an era of ever-increasing sophistication in financial services. The Great Recession, which was accompanied by a housing market crash and soaring personal bankruptcies, home foreclosures, and so on, has created a renewed sense of

urgency for understanding household balance sheets. Developments on two fronts—advances in theoretical modeling as discussed in Section 3, combined with richer data sources on credit histories and mortgages that are increasingly becoming available to researchers—will make faster progress feasible in this area.

## REFERENCES

Abowd, John M., and David E. Card. 1989. "On the Covariance Structure of Earnings and Hours Changes." *Econometrica* 57 (March): 411–45.

Acemoglu, Daron. 2002. "Technical Change, Inequality, and the Labor Market." *Journal of Economic Literature* 40 (March): 7–72.

Aguiar, Mark, and Erik Hurst. 2008. "Deconstructing Lifecycle Expenditures." Working Paper, University of Rochester.

Aguiar, Mark, and Mark Bils. 2011. "Has Consumption Inequality Mirrored Income Inequality?" Working Paper, University of Rochester.

Aiyagari, S. Rao. 1993. "Uninsured Idiosyncratic Risk and Aggregate Saving." Federal Reserve Bank of Minneapolis Working Paper 502.

Aiyagari, S. Rao. 1994. "Uninsured Idiosyncratic Risk and Aggregate Saving." *The Quarterly Journal of Economics* 109 (August): 659–84.

Altug, Sumru, and Robert A. Miller. 1990. "Household Choices in Equilibrium." *Econometrica* 58 (May): 543–70.

Arslan, Yavuz. 2011. "Interest Rate Fluctuations and Equilibrium in the Housing Market." Working Paper, Central Bank of the Republic of Turkey.

Athreya, Kartik B. 2002. "Welfare Implications of the Bankruptcy Reform Act of 1999." *Journal of Monetary Economics* 49 (November): 1,567–95.

Attanasio, Orazio, and Steven J. Davis. 1996. "Relative Wage Movements and the Distribution of Consumption." *Journal of Political Economy* 104 (December): 1,227–62.

Attanasio, Orazio, Erich Battistin, and Hidehiko Ichimura. 2007. "What Really Happened to Consumption Inequality in the United States?" In *Hard-to-Measure Goods and Services: Essays in Honour of Zvi Griliches*, edited by E. Berndt and C. Hulten. Chicago: University of Chicago Press.