1 Tik-Tak Algorithm,following Guvenen (2011)

Denote the number of parameters to be estimated as J. Let $\mathbf{p}$ be a J-dimensional parameter.

1. *Initialization*
   (a) Determine parameter bounds for each parameter, outside of which the objective function should be set to a high value.
   (b) Generate a sequence of Sobol vectors with a sequence length of N.
   (c) Evaluate the function value at each of these N Sobol points. THIS CAN BE DONE IN PARALLEL. Multiprocessing across nodes is fine.
   (d) Keep the set N^* of points that have the lowest function values, and order them in descending order, as $s_1, ..., s_{N^*}$.
   (e) N^* is the number of restarts in the global stage. In one paper, they set $N^* = 0.1 \times N$.
   (f) Save all of these to a JLD or CSV file.
   (g) Set the global iteration number $i = 1$.
2. *Global Stage*
   (a) Draw the i^{th} vector value in the Sobol sequence: s_i.
   (b) Open and read from the text file "saved_parameters..txt" the function values (and corresponding parameter vectors) of the previously found local minima. Denote the lowest function value found as of ieration $i-1$ as f_{i-1}^{low} and the corresponding parameter vector as p_{i-1}^{low} . Denote N_{last} as the total nuber of searches completed by all CPUs.
   (c) Generate an initial point for the local stage as follows:
      - If $i < I_{min}$, then use s_i as the starting guess S_i. (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 starting guess $S_i = (1-\theta_i)s_I + \theta_i p_{i-1}^{low}$, where $\theta_i \in [0, \bar{\bar{\theta}}]$ and $\bar{\theta} < 1$. In one paper, they set $\theta_i = \min[\bar{\theta}, (i/N^*)^2]$, and $\bar{\theta} = 0.995$.
      - As θ is increased, local searches are restarted from a narrower part of the parameter space that yielded the lowest local minima before.
3. *Local Stage*
   (a) Using S_i as a starting point, use the Nelder-Mead 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.)

(b) Stop when either (i) a certain tolerance is acheived (i.e. 10^{-8}), (ii) function values do not improve by more than a certain amount, or (iii) the maximum iteration number is reached.

(c) Open"saved_parameters.txt" and record the local minimum found (function value and parameters).

4. *Stopping Rule*

(a) Repeat Steps 2-3 until local searches are completed from starting points that utilize each of the N^* Sobol points.

(b) Return the point with the lowest function value from "saved_parameters..txt" as the global minimum.

2 Extensions

Narrowing Down the Search Area

As noted, the Nelder-Mead algorithm requires J+1 candidate points as inputs (the vertices of the J-dimensional simplex). One of these points is given by S_i. The other vertices are drawn randomly. After a sufficiently large number of restarts have been compelted, rank all of the function values and take the lowest 10 or 20 percent of values. Then for each dimension, pick the minimum and maximum parameter values within this set of minima (in each dimension). To generate vertices, randomly sample points between the minimum and maximum in each dimension j. This allows the simplex algorithm to search more intensively in a narrower area.

We can alternatively set the bounds implicitly (see old NYFed code) and rescale the parameters so the increments are normaliezed. Decide which setup makes more sense.

Parallelizing (Steps 2-3 of) the Algorithm

- Suppose we have N processes.
- We can distinguish between each process with an integer value $n \in \{1, ..., N\}$. This will be a global variable (this can also be a Slurm environment variable if we end up using a SLURM job array instead of Julia's built-in "pmap" function).
- Upon starting, each process will load the same Sobol sequence.
- Let process n choose the n^{th} element of the sequence for its initial point and launch a local search from that point. After completion, each run will open the same file and record the local minimum and parameter value it finds.

- Keep track of the *total* number of local searches completed by all CPUS as N_{last}. Then we can instead link the increase in θ_i to N_{last}. We can also select Sobol points using this iteration number, but this could be risky since a second process may read the file right after the first process reads it because the first process has not yet written to it.
  - For now, let the n^{th} process use the $N \times k + n^{th}$ Sobol number for setting it's initial point.

3 Questions/Issues

- Would it be more performant to use a SLURM job array rather than Julia's built in pmap functionality for Steps 2-3? The processes are *mostly* independent, except we need to keep track of the *lowest* function value across *all* past iterations (across all processors) to select the intiial point for the local search.
- What is the best way to do file I/O for this step,since there will be several processors reading and writing to the same txt file? Is there a way to lock the file?
- Is it possible to multithread on each process in Julia on the cluster? If so, how do I request this in the Slurm file and in Julia since the number of threads utilized in Julia is an envrionment variable that is set when starting up Julia (at least for the main task)?