Simplex of Nelder & Mead
Algorithm

• AKA the “Amoeba algorithm”
• In the class of “direct search” methods
• Unconstrained (although constraints can be added as part of error function) nonlinear optimization
• Very popular:
  – Mathematica: \texttt{NMinimize}[f, \texttt{vars}, \texttt{Method} -> "NelderMead"]
  – MatLab: \texttt{fminsearch}

• The “simplex”
  – For an \(N\)-dimensional search space, create an \((N+1)\)-dimensional shape, typically by displacing points along each primary direction \((N)\) from a single starting guess \((+1)\).

• To evolve
  1. Reflect the point with the highest error function through centroid of all of the other points in the simplex
  2. If the error function at this new point is:
     a. neither best nor worst, accept the point where it lands.
     b. lowest error function among all current points, then \textit{expand} the simplex by moving the point by the same amount again in the same direction. Accept this expansion if it’s better, other wise accept the first location.
     c. highest error function among all current points, then \textit{contract} the simplex by only moving half way to the centroid. Accept this contraction if the contracted point is better than the starting point.
     d. Otherwise (contraction fails), \textit{shrink} the entire simplex by moving every point halfway towards the best point in the simplex.

Simplex of Nelder & Mead
Example
Simplex of Nelder & Mead

General comments

- Quite easy to program and doesn’t require gradient info.
- Deals well with noise since “gradients” calculated on entire convex hull
- Deterministic and can get trapped in local optima
- Convergence can not be proved and there are some cases where it’s terrible, but in moderately well-behaved spaces it often works very well.
Simulated annealing
Concept

- Based on the analogy between metal cooling into a solid with minimum energy and general extremization problem.

- Has some similarities with N&M in that
  - no derivatives are needed
  - new points are chosen based on local topology
  - constraints not directly handled

- Major advantage is ability to find global extrema by allowing for decreases in the merit function with a probability

\[ p = e^{-\frac{\delta f}{T}} \]

Boltzmann distribution.
Why this? No reason, really.

\[ \begin{array}{|c|c|}
\hline
\delta f & \text{Increase in function to be minimized} \\
T & \text{“Temperature”} \\
p & \text{Probability of accepting point} \\
\hline
\end{array} \]
Generation of candidates
Parks’ algorithm

Pick new solution vector \( \vec{x}_{n+1} \) by moving from previous solution \( \vec{x}_n \) by a random variable \(-1 < u_i < 1\) times a maximum change \( D_{ii} \).

\[
\vec{x}_{n+1} = \vec{x}_n + D_{n+1} \vec{u}
\]

Parks’ update rule

At each successful change, update \( D \) to encourage new solutions in directions that have been previously successful

\[
D_{n+1} = (1 - \alpha)D_n + \alpha \omega R
\]

Update \( D \)

where \( R \) is a diagonal matrix whose elements are

\[
|\vec{x}_n - \vec{x}_{n-1}|
\]

and \( \alpha \) is the damping rate and \( \omega \) the weighting.

Note how this rule is similar to the Nelder and Mead simplex growth and contraction rules except that new points here are picked randomly and sometimes allowed to be “worse”.

Cooling schedule

“Black art”

• Initial temperature should “melt”
  system = explore a large fraction of the solution space

• Final temperature should “freeze”
  system = converge deterministically to nearest optimum.

• The rule to reduce temperature should not be too fast (which would miss the global optimum) or too slow (which would waste time).

• Most common (but not necessarily best) cooling schedules are linear
  \( T_{n+1} = T_n - \delta T \) and proportional
  \( T_{n+1} = c T_n \), \( c < 1 \)
Including constraints

Penalty functions

Often your search space has constraints such as physically reasonable bounds on a variable, discrete (often referred to as “integer”) variables or disallowed combinations or areas of the search space.

To include constraints, one can modify the merit function \( f(x) \) with a “penalty”:

\[
f_p(x) = f(x) + \bar{w}^T \bar{c}(x)
\]

where \( \bar{w} \) is a vector of weights and \( \bar{c}(x) \) measures the violation of the constraints. Care must be taken to balance the penalty and the function so as to enforce the constraints but not change the optimum. Unfortunately, sharp constraints tend to confuse algorithms.

A simulated annealing penalty function can deal with this by increasing the strength of the constraint with decreasing temperature, e.g.

\[
f_p(x) = f(x) + \frac{1}{T} \bar{w}^T \bar{c}(x)
\]