

Other Optimization algorithms

April 10, 2006

1 Trust region methods

Trust region methods are approaches to optimization that use a simple model of the function-to-be-optimized limited to a local “trusted” region. The simple model is usually a quadratic function centered on the current guess of the form:

$$\phi_k(p) = J(\mathbf{x}_k) + \frac{\partial J}{\partial x^i}(\mathbf{x}_k) p^i + \frac{1}{2} p^i B_{ij}(\mathbf{x}_k) p^j$$

where B_{ij} is some symmetric operator (matrix) — possibly the Hessian $\frac{\partial^2 J}{\partial x^i \partial x^j}$, or a positive-definite approximation to it. The minimizer of the approximate function constrained to the trust region is solved at each step. Usually, the trust region is an L_p ball: $\|\mathbf{p}\|_p \leq \Delta_k$, where $p = 2$ is common.

While gradient-descent methods find a direction to proceed and then perform a line-search to find how far in that direction to proceed, trust-region methods pick a maximum step-size and then find solve a smaller optimization problem to find the direction to proceed to the next step.

At each iteration of the algorithm we must solve (at least approximately) the constrained optimization problem:

$$\mathbf{p}_k = \arg \min_{\mathbf{p} \in \mathbb{R}^n} \phi_k(\mathbf{p}) \quad \|\mathbf{p}\| \leq \Delta_k.$$

To evaluate how well the model function agreed with the true function we calculate the ratio of true reduction to the predicted reduction in the function value:

$$\rho_k = \frac{J(\mathbf{x}_k) - J(\mathbf{x}_k + \mathbf{p}_k)}{\phi(0) - \phi(\mathbf{p}_k)}.$$

The denominator is always non-negative. If the numerator is negative (or smaller than some small value) then the model is a poor representation of the function and we need to reduce the trusted region as well as throw away the predicted update. If ρ_k is 1 or near it then the model function is doing a good job of predicting the function (at least at these two points). If the constraint is active, then we should expand the trusted region and solve again. The overall algorithm can be written as

Trust Region Algorithm

Choose $\bar{\Delta} > 0$, $\Delta_0 \in (0, \bar{\Delta})$, $\eta \in [0, \frac{1}{4})$, and \mathbf{x}_0

Start iterations in k

Solve $\mathbf{p}_k = \arg \min_{\mathbf{p}} \phi_k(\mathbf{p}) \quad \|\mathbf{p}\| \leq \Delta_k$

Evaluate $\rho_k = \frac{J(\mathbf{x}_k) - J(\mathbf{x}_k + \mathbf{p}_k)}{\phi(0) - \phi(\mathbf{p}_k)}$

If $\rho_k < \frac{1}{4}$

$$\Delta_{k+1} = \frac{1}{4} \|\mathbf{p}_k\|$$

else if $\rho_k > \frac{3}{4}$ and $\|\mathbf{p}\| = \Delta_k$

$$\Delta_{k+1} = \min(2\Delta_k, \bar{\Delta})$$

else

$$\Delta_{k+1} = \Delta_k$$

If $\rho_k > \eta$
 $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$
else
 $\mathbf{x}_{k+1} = \mathbf{x}_k$

The most time-consuming step in this algorithm is the solving of the constrained optimization function of the approximate model function. The solution to this problem can be fully characterized when the l_2 -norm is used. However, an exact solution is not generally needed. A “good-enough” solution will allow convergence just as a line search in the case of gradient-descent methods did not need to return an exact minimizer (just something that satisfied the Wolfe conditions).

There has been recent research suggesting that if ρ_k is too large, then the trust region should not be expanded. Better rules on when the trust region should be expanded can result in 2-3 times fewer function evaluations. The rule suggested by Walmag and Delhez in their SIOPT paper (“A Note on Trust-region Radius Update”, Volume 16, Issue 2, pp. 548-562) is

$$\Delta_{k+1} = \Lambda(\rho_k) \Delta_k$$

where

$$\Lambda(\rho) = \begin{cases} \alpha_1 & \rho \leq 0 \\ \alpha_1 + (1 - \alpha_1) \left(\frac{\rho}{\beta}\right)^2 & 0 < \rho < \beta \\ \alpha_3 + (\alpha_2 - \alpha_3) \exp\left[-\left(\frac{\rho-1}{\beta-1}\right)^2\right] & \rho \geq \beta \end{cases}$$

where $\alpha_1, \alpha_2, \alpha_3$ and β are all positive parameters. For their paper, they chose $\alpha_1 = \frac{1}{2}$, $\alpha_2 = 2$, $\alpha_3 = 1.01$ and $\beta = 0.95$. This rule does not reward large values of ρ by expanding the trust region (after all if the model matches the function then $\rho \approx 1$).

1.1 Exact solution

The exact solution to the constrained minimization problem needed at each step of the Trust-region method can be characterized by the following theorem (simply replace g_i with $\frac{\partial J}{\partial x^i}$).

Theorem:

The vector \mathbf{p}^* is a global solution of the trust-region problem

$$\min_{\mathbf{p} \in \mathbb{R}^n} J(\mathbf{x}_0) + g_i p^i + \frac{1}{2} p^i B_{ij} p^j \quad \text{s.t. } \|\mathbf{p}\| \leq \Delta$$

if and only if there is a scalar $\lambda \geq 0$ such that the following conditions are satisfied:

$$\begin{aligned} (\mathbf{B} + \lambda \mathbf{I}) \mathbf{p}^* &= -\mathbf{g}, \\ \lambda (\Delta - \|\mathbf{p}^*\|) &= 0, \\ (\mathbf{B} + \lambda \mathbf{I}) &\text{ is positive semidefinite.} \end{aligned}$$

An algorithm to find \mathbf{p}^* based on this theorem can be derived by considering that either $\lambda = 0$ satisfies the last condition or else we define

$$\mathbf{p}(\lambda) = -(\mathbf{B} + \lambda \mathbf{I})^{-1} \mathbf{g}$$

for λ sufficiently large that $\mathbf{B} + \lambda \mathbf{I}$ is positive definite ($\lambda > -\lambda_0$ where λ_0 is the smallest eigenvalue of \mathbf{B} —it is negative). Then, we seek a value to satisfy the second condition of theorem:

$$\|\mathbf{p}(\lambda)\| = \Delta.$$

where $\lambda > -\lambda_0$. The solution to this problem can be obtained with a 1-d root-finding technique producing the following algorithm:

Algorithm:

Given $\lambda^{(0)}, \Delta > 0$

for $l = 0, 1, 2, \dots$

Factor $\mathbf{B} + \lambda^{(l)}\mathbf{I} = \mathbf{R}^T\mathbf{R}$

Solve $\mathbf{R}^T\mathbf{R}\mathbf{p}_l = -\mathbf{g}, \mathbf{R}^T\mathbf{q}_l = \mathbf{p}_l$

Set

$$\lambda^{(l+1)} = \lambda^{(l)} + \left(\frac{\|\mathbf{p}_l\|^2}{\|\mathbf{q}_l\|^2} \right) \left(\frac{\|\mathbf{p}_l\| - \Delta}{\Delta} \right).$$

Only 2-3 iterations of this algorithm are usually necessary to obtain a good-enough approximation. We next look at three other approximate solutions that are good enough to guarantee global convergence of the overall Trust-Region algorithm.

1.2 The Cauchy Point

The Cauchy-point represents the kind of improvement that must be made. A trust-region method will be globally convergent if its steps, \mathbf{p}_k , attain a reduction in the model ϕ_k that is at least some fixed multiple of the decrease attained by the Cauchy step at each iteration.

The Cauchy point, therefore quantifies sufficient reduction just as the Wolfe conditions did in the line-search algorithm. The Cauchy point, \mathbf{p}_k^c is defined as $\tau_k \mathbf{p}_k^s$ where

$$\mathbf{p}_k^s = \arg \min_{\mathbf{p} \in \mathbb{R}^n} J(\mathbf{x}_k) + \frac{\partial J}{\partial x^i}(\mathbf{x}_k) p^i \quad \text{s.t.} \quad \|\mathbf{p}\| \leq \Delta_k$$

and

$$\tau_k = \arg \min_{\tau > 0} \phi_k(\tau \mathbf{p}_k^s) \quad \text{s.t.} \quad \|\tau \mathbf{p}_k^s\| \leq \Delta_k.$$

The solution to this problem (in a linear space where we can say $\nabla J_k \equiv \frac{\partial J}{\partial \mathbf{x}}(\mathbf{x}_k)$) is

$$\mathbf{p}_k^s = -\frac{\Delta_k}{\|\nabla J_k\|} \nabla J_k$$

and

$$\tau_k = \begin{cases} 1 & \nabla J_k^T B_k \nabla J_k \leq 0, \\ \min \left(1, \frac{\|\nabla J_k\|^3}{\Delta_k (\nabla J_k^T B_k \nabla J_k)} \right) & \text{otherwise.} \end{cases}$$

The problem with using the Cauchy-point at each iteration is that it is always in the gradient-descent direction and therefore performs more poorly than necessary. We really want to use B_k in order to alter the direction of search not just the length to step in the gradient-descent direction. If B_k is positive definite then there are two methods that can be used to improve convergence over the Cauchy point.

1.3 The dogleg method

This method looks at the effect of the trust-region radius on the exact solution to the constrained subproblem (for positive semi-definite \mathbf{B}). Denote the solution for a given trust-region radius as $\mathbf{p}^*(\Delta)$. Define $\mathbf{g} = \nabla J$. The unconstrained minimizer of ϕ is $\mathbf{p}^B = -\mathbf{B}^{-1}\mathbf{g}$. If this solution is within the feasible set then clearly this is the solution to the constrained problem as well thus,

$$\mathbf{p}^*(\Delta) = \mathbf{p}^B \quad \text{when } \Delta \geq \|\mathbf{p}^B\|.$$

When Δ is very small, then the constraint guarantees that the quadratic term in ϕ will have little effect. And thus,

$$\mathbf{p}^*(\Delta) \approx -\Delta \frac{\mathbf{g}}{\|\mathbf{g}\|} \quad \text{when } \Delta \text{ is small.}$$

For intermediate values, the path of $\mathbf{p}^*(\Delta)$ will be a curved trajectory. The dog-leg method replaces this curved trajectory with two line-segments. The first line-segment runs from the origin to the unconstrained minimizer along the steepest-descent direction:

$$\mathbf{p}^U = -\frac{\mathbf{g}^T \mathbf{g}}{\mathbf{g}^T \mathbf{B} \mathbf{g}} \mathbf{g},$$

while the second line segment runs from \mathbf{p}^U to \mathbf{p}^B . Thus,

$$\tilde{\mathbf{p}}(\tau) = \begin{cases} \tau \mathbf{p}^U & 0 \leq \tau \leq 1 \\ \mathbf{p}^U + (\tau - 1)(\mathbf{p}^B - \mathbf{p}^U) & 1 \leq \tau \leq 2. \end{cases}$$

The dog-leg method chooses \mathbf{p} to minimize ϕ along this path subject to the trust-region bound. It can be shown that

1. $\|\tilde{\mathbf{p}}(\tau)\|$ is an increasing function of τ
2. $\phi(\tilde{\mathbf{p}}(\tau))$ is a decreasing function of τ .

Because of these two facts, it is clear that the path $\tilde{\mathbf{p}}(\tau)$ intersects the trust-region boundary $\|\mathbf{p}\| = \Delta$ at exactly one point if the unconstrained optimizer is outside of the trust region and nowhere otherwise. Thus, the new direction is either \mathbf{p}^B (if it lies within the trust region radius) or the point of intersection of the second leg of the dog-leg path and the boundary. This is the solution to

$$\begin{aligned} \Delta^2 &= \|\mathbf{p}^U + (\tau - 1)(\mathbf{p}^B - \mathbf{p}^U)\|^2 \\ &= (\mathbf{p}^U + (\tau - 1)(\mathbf{p}^B - \mathbf{p}^U), \mathbf{p}^U + (\tau - 1)(\mathbf{p}^B - \mathbf{p}^U)) \\ \Delta^2 &= \|\mathbf{p}^U\|^2 + 2(\tau - 1)(\mathbf{p}^B - \mathbf{p}^U, \mathbf{p}^U) + (\tau - 1)^2 \|\mathbf{p}^B - \mathbf{p}^U\|^2. \end{aligned}$$

The solutions are ($b \neq 0$)

$$\tau = 1 - \frac{b}{a} \left[\sqrt{1 + \frac{ac}{b^2}} - 1 \right].$$

where

$$\begin{aligned} a &= \|\mathbf{p}^B - \mathbf{p}^U\|^2 \\ b &= (\mathbf{p}^B - \mathbf{p}^U, \mathbf{p}^U) = (\mathbf{p}^B, \mathbf{p}^U) - \|\mathbf{p}^U\|^2 \\ c &= \Delta^2 - \|\mathbf{p}^U\|^2. \end{aligned}$$

We have chosen the solution so that $\tau \in [1, 2]$ because $b \leq 0$. If $b = 0$, then $\tau = 1$ is the solution. Notice that to find \mathbf{p}^B we must solve the linear system (which involves the gradient).

$$\mathbf{B} \mathbf{p}^B = -\mathbf{g}.$$

1.4 The two-dimensional sub-space method

One way to expand the dogleg method is to search the entire two-dimensional sub-space spanned by \mathbf{p}^B and \mathbf{p}^U (equivalently $\mathbf{p} \in \text{span}[\mathbf{g}, \mathbf{B}^{-1}\mathbf{g}]$). This results in a two-dimensional constrained optimization problem:

$$\mathbf{p}_k = \arg \min_{\mathbf{p}} \phi_k(\mathbf{p}) = J_k + \mathbf{g}_k^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T \mathbf{B}_k \mathbf{p} \quad \text{s.t.} \quad \|\mathbf{p}\| \leq \Delta, \mathbf{p} \in \text{span}[\mathbf{g}_k, \mathbf{B}_k^{-1} \mathbf{g}_k].$$

Thus, constrain $\mathbf{p} = \alpha \mathbf{g}_k + \beta \mathbf{B}_k^{-1} \mathbf{g}_k$ and solve

$$\alpha_k, \beta_k = \arg \min_{\|\mathbf{p}\| \leq \Delta} \mathbf{g}_k^T (\alpha \mathbf{g}_k + \beta \mathbf{B}_k^{-1} \mathbf{g}_k) + \frac{1}{2} (\alpha \mathbf{g}_k + \beta \mathbf{B}_k^{-1} \mathbf{g}_k)^T \mathbf{B}_k (\alpha \mathbf{g}_k + \beta \mathbf{B}_k^{-1} \mathbf{g}_k).$$

This constrained problem can be solved for the case of positive definite \mathbf{B}_k (remember \mathbf{B} is always symmetric). It is the problem of minimizing

$$J(\alpha, \beta) = \alpha(\beta + 1)a + \beta(\beta + 2)b + \alpha^2c$$

subject to the constraint that

$$\alpha^2a + \beta^2d + 4\alpha\beta b \leq \Delta^2$$

where

$$\begin{aligned} a &= \mathbf{g}^T \mathbf{g} \\ b &= \frac{1}{2} \mathbf{g}^T \mathbf{B}^{-1} \mathbf{g} \\ c &= \frac{1}{2} \mathbf{g}^T \mathbf{B} \mathbf{g} \\ d &= \mathbf{g}^T \mathbf{B}^{-2} \mathbf{g}. \end{aligned}$$

The unconstrained optimum is $\alpha = 0$ and $\beta = -1$. Expanding around this point we can write $\mathbf{z}^T = [\alpha, \beta]$ and

$$J(\mathbf{z}) = -b + \frac{1}{2} \mathbf{z}^T \begin{bmatrix} 2c & a \\ a & 2b \end{bmatrix} \mathbf{z}$$

which because it can be shown that $4cb - a^2 \geq 0$ (Cauchy-Schwartz), the level-curves of the unconstrained problem are ellipses. The constraint is the interior of another ellipse defined as

$$\mathbf{z}^T \begin{bmatrix} a & 2b \\ 2b & d \end{bmatrix} \mathbf{z} \leq \Delta^2$$

which is an ellipse because $ad - 4b^2 \geq 0$ (another application of Cauchy-Schwartz). Thus, the solution to the problem is either $\alpha = 0, \beta = -1$ if this point satisfies the constraint, or it is the point on the boundary of the constraint region that minimizes the function. Thus, it is the minimizer of the augmented Lagrangian:

$$L(\alpha, \beta) = \alpha(\beta + 1)a + \beta(\beta + 2)b + \alpha^2c + \lambda[\alpha^2a + \beta^2d + 4\alpha\beta b - \Delta^2]$$

where λ is found to ensure the constraint is satisfied. Taking the derivatives with respect to α and β gives

$$\begin{aligned} \frac{\partial L}{\partial \alpha} &= (\beta + 1)a + 2c\alpha + 2\alpha\lambda a + 4\lambda\beta b \\ \frac{\partial L}{\partial \beta} &= \alpha a + 2(\beta + 1)b + 2d\lambda\beta + 4\lambda\alpha b \end{aligned}$$

Setting this equal to zero results in the system of equations

$$\begin{aligned} (2c + 2\lambda a)\alpha + (a + 4\lambda b)\beta &= -a \\ (a + 4\lambda b)\alpha + (2b + 2\lambda d)\beta &= -2b \end{aligned}$$

Solving this system of equations results in

$$\begin{aligned} \alpha &= \frac{\frac{a}{2}(b - \lambda(4b^2 - d))}{a\lambda^2(4b^2 - d) + \lambda ab(a - 1) - \lambda cd - bc} \\ \beta &= \frac{\frac{1}{4}(4cb - a^2)}{a\lambda^2(4b^2 - d) + \lambda ab(a - 1) - c\lambda d - bc}. \end{aligned}$$

The value of λ is determined by solving

$$\alpha^2a + \beta^2d + 4\alpha\beta b - \Delta^2 = 0.$$

As both α and β are ratios of polynomials in λ , this can be solved using root finding methods. The result is

$$A_4\Delta^2\lambda^4 + A_3\Delta^2\lambda^3 + A_2\lambda^2 + A_1\lambda + A_0 = 0$$

where

$$\begin{aligned}
A_0 &= -32ab^3c - 16b^2c^2d + 8a^2bcd + 4a^3b^2 - a^4d + 16\Delta^2b^2c^2 \\
A_1 &= 32bc [ab^2c(4b^2 - d) + \Delta^2cd + \Delta^2ab - \Delta^2a^2b] \\
A_2 &= 32\Delta^2abcd + 16\Delta^2a^4b^2 + 16\Delta^2c^2d^2 - 32\Delta^2abc(4b^2 - d) - 32\Delta^2a^3b^2 \\
&\quad + 16\Delta^2a^2b^2 - 32\Delta^2a^2bcd - 4a^3(4b^2 - d)^2 \\
A_3 &= 32(a^3b - a^2b - acd)(4b^2 - d) \\
A_4 &= 16a^2(4b^2 - d)^2
\end{aligned}$$

2 Transition to combinatorial problems

The optimization algorithms we've looked at so far have been largely based on Taylor-series expansions of the functionals and assume an underlying continuous space. These algorithms can be limited (or would need to be adapted) when dealing with countable spaces where a formal gradient does not make sense. Because secant's still make sense, the gradient-descent methods could be adapted to secant-based calculations. However, the limiting process that defines the derivative does not make sense and so some of the theoretical results would need to be adjusted for the discrete case. In theory this could be done, but I have not seen results to this effect.

You could always try a given gradient-based algorithm and calculate the gradient using a secant approximation:

$$\frac{\partial J}{\partial m^i} = \frac{J(m^1, \dots, m^{i-1}, m^i + \delta m^i, m^{i+1}, \dots, m^n) - J(\mathbf{m})}{\delta m^i}$$

taking care to ensure that δm^i is not smaller than the discretization level. Interpolation could also be used to evaluate the function "between sample points" if needed and steps could round to the next discrete value of \mathbf{m} .

So, just because you have discrete data doesn't mean you can't use gradient-descent algorithms, but you may need to adapt them for the circumstance. There are other optimization methods that don't require (or use) gradient information that might be easier to adapt to discrete-data.

3 (Nelder-Mead) Simplex method

The Nelder-Mead, or downhill, simplex method is a simple, but powerful approach to optimization that only requires function evaluations. It is not related to the linear programming simplex method except that both algorithms deal with a simplex of points in an N -dimensional space. A simplex in N -dimensional space is just a polygon with $N + 1$ points that has non-zero volume in the space. Thus, in two-dimensions, a simplex is a triangle, in three-dimensions it is a tetrahedron and so forth. The method creates a new simplex on each iteration by using only function evaluations on the vertices of the current simplex.

For many problems the number of function evaluations is larger than would be necessary if gradient information were available (or if gradients could be estimated). But, because it is so simple it is often easy to implement and/or adapt to a particular problem and so is worth learning about.

This algorithm starts with a simplex of $N + 1$ points. A straightforward approach to finding the initial simplex is to create it using the points $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N$ where \mathbf{x}_0 is the starting guess and

$$\mathbf{x}_i = \mathbf{x}_0 + \lambda_i \mathbf{e}_i$$

with \mathbf{e}_i the coordinate unit vectors and λ_i are scale factors in each direction.

At each iteration, a new simplex is formed. Possible stopping conditions for the iterations are when either the change in the simplex coordinates is below some tolerance or the difference in the highest and lowest function values in the simplex is below some threshold.

At each iteration of the simplex method, a new simplex is constructed from the old simplex in an attempt to move downhill. It is assumed that the function values at each of the vertices of the simplex are available.

Define $J_i = J(\mathbf{x}_i)$ to be the function evaluated at vertex i . At each iteration, let b be the vertex of the best function value and let w be the vertex of the worst so that (for a minimization problem).

$$J_b \leq J_i \leq J_w.$$

A series of simplex-alteration rules are applied in an attempt to get a new simplex where the function is smaller on the vertices. The rules are outlined below.

- 1. Reflection** The first thing attempted is to reflect the worst point through the mid-point of all the other vertices to a new trial point. The mid-point of the best vertices is

$$\mathbf{x}_m = \frac{1}{N} \sum_{i \neq w} \mathbf{x}_i.$$

Then, the new point, \mathbf{x}_r , is chosen as a reflection of \mathbf{x}_w through \mathbf{x}_m so that \mathbf{x}_m is at the midpoint of \mathbf{x}_w and \mathbf{x}_r . Thus,

$$\mathbf{x}_r = 2\mathbf{x}_m - \mathbf{x}_w.$$

If $J_b \leq J(\mathbf{x}_r) < J_w$ (i.e. the point is better but not the “best”), then replace \mathbf{x}_w with \mathbf{x}_r and iterate again.

- 2. Expansion** If $J(\mathbf{x}_r) < J_b$ (i.e. we have a new “best”) then we must be going in approximately the right direction so expand the simplex along that direction. In other words find \mathbf{x}_e so that \mathbf{x}_r is at the midpoint between \mathbf{x}_m and \mathbf{x}_e :

$$\mathbf{x}_e = 2\mathbf{x}_r - \mathbf{x}_m = 3\mathbf{x}_m - 2\mathbf{x}_w = \mathbf{x}_w + 3(\mathbf{x}_m - \mathbf{x}_w).$$

If $J(\mathbf{x}_e) < J(\mathbf{x}_r)$ then replace \mathbf{x}_w with \mathbf{x}_e and iterate again. Otherwise ($J(\mathbf{x}_e) \geq J(\mathbf{x}_r)$) replace \mathbf{x}_w with \mathbf{x}_r and iterate again.

- 3. Contraction** If $J(\mathbf{x}_r)$ is still the worst point in the new simplex (i.e. $J(\mathbf{x}_r) > J_p$) where p is the next-to-largest value of $J(\mathbf{x}_i)$, then contract along the $\mathbf{x}_m - \mathbf{x}_w$ direction by finding

$$\mathbf{x}_{c1} = \frac{1}{2}(\mathbf{x}_r + \mathbf{x}_m) = \frac{1}{2}(3\mathbf{x}_m - \mathbf{x}_w).$$

If $J(\mathbf{x}_{c1}) < J_p$ then replace \mathbf{x}_w with \mathbf{x}_{c1} and iterate again. Otherwise, try a new contraction point by computing

$$\mathbf{x}_{c2} = \frac{1}{2}(\mathbf{x}_m + \mathbf{x}_w).$$

If $J(\mathbf{x}_{c2}) < J_p$ then replace \mathbf{x}_w with \mathbf{x}_{c2} and iterate again.

- 4. Shrink Simplex** If none of the above operations are successful in finding a new vertex to replace the worst-case vertex, then all of the vertices except the best vertex are replaced with vertices at the mid-point on the line connecting the vertex with the current best vertex. Thus,

$$\mathbf{x}'_i = \frac{1}{2}(\mathbf{x}_b + \mathbf{x}_i) \quad \forall i \neq b.$$

The iteration then proceeds again.

These rules allow the simplex to crawl amoeba-like through the solution space progressing towards the minimum point.

4 Simplex Method Figures

On the class web-page there is a link to a file with figures that can help explain the downhill simplex method. These figures were taken from Alberto Gonzalo’s Lecture notes for his Biostatistics 615/815 class at the University of Michigan.

5 Simulated Annealing

Useful adaptation of Metropolis Hastings when you want to find the global optimum.

6 Genetic Algorithms

I think these algorithms are generally over-rated but they can be useful sometimes when you can clearly define a population, a genome, and what it means to “mix” your genomes. Circuit-design problems have been assisted using genetic-algorithms.