Multivariate Optimization Algorithms

The problem here is to determine the minimum value of the function $F(\mathbf{x})$, as well as the location of that minimum in the space of design variables, $\mathbf{x}^*$. Notice that the notation x used here contradicts the use of **x** for random variables in other documents. Also note that we must usually assume that F behaves nicely, i.e., that it is smooth and convex.

The Downhill Simplex Algorithm

The key advantage of this algorithm is that derivatives of F are not needed. That is great, but naturally, this leads to slower convergence than gradient-based algorithms. Nelder-Mead is another name of this algorithm, because of the paper "A simplex method for function minimization" published by paper Nelder and Mead in Volume 7 of the Computer Journal in 1965. Their work built on work by Spendley, Hext, and Himsworth published in Volume 4 of Technometrics in 1962 entitled "Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation." A simplex is easy to explain for the case of two design variables; then the simplex is a triangle like the one shown in Figure 1. For each design variable that is added, one point is added to form a simplex. In the downhill simplex algorithm we repeatedly discard one point of the simplex, and add another point to form a new simplex. Ideally, the simplex then steadily moves downhill, closer to the optimum at the bottom of the convex function $F(\mathbf{x})$. There are four ways to reshape the simplex: reflection, expansion, contraction, and shrink. Because those are not explained here at this time, perhaps the explanation of this algorithm on Wikipedia could provide help. Perhaps more importantly, the algorithm is implemented in the Python code that follows this document.

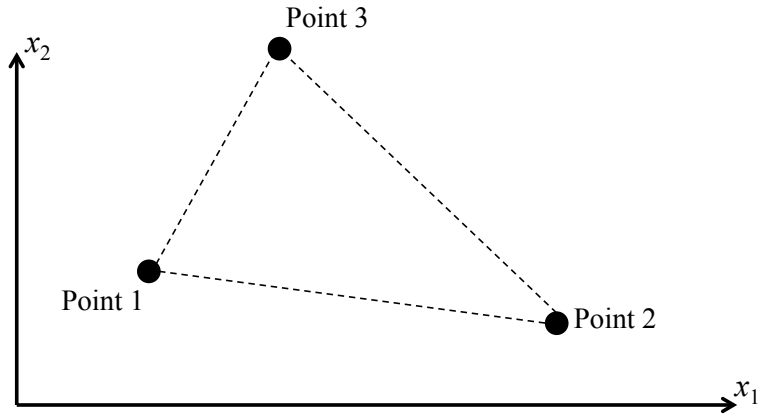

Figure 1: A simplex for the case of two design variables.

The Directional Line Search Algorithm

The effectiveness of iterative searches for x* depends on the availability of derivatives of F with respect to $\mathbf{x}$. If no derivatives are available then the downhill simplex algorithm may be an option. If first-order derivatives are available, i.e., the gradient vector $\nabla F \equiv \partial F / \partial \mathbf{x}$, then the gradient-based methods covered here are available. Most gradient-based algorithms take the form

$$\mathbf{x}_{n+1} = \mathbf{x}_n + s_n \cdot \mathbf{d}_n \tag{1}$$

where n=counter for the iterations, $\mathbf{x}_{n+1}$=new trial point in the space of design variables, $\mathbf{x}_n$=old trial point, s_n=step size along the search direction $\mathbf{d}_n$. A reasonable stopping criterion might be $||\mathbf{x}_{n+1}-\mathbf{x}_n||<\varepsilon$, i.e., when the new trial point is sufficiently close to the previous one. Various gradient-based algorithms differ in how they select the search direction, $\mathbf{d}_n$, and how they select the step size, s_n. The selection of s_n is essentially a line-search along the search direction, hence the name "directional line search algorithm." That line search is a single-variable optimization problem for which algorithms such as the golden section and secant methods are available. This means that the line search algorithms in Rts can be used for three purposes:

1. Root-finding
2. Single-variable optimization
3. Step size selection in multi-variable optimization

To facilitate this, Rts contains classes called "merit function" and "call-back assistant." A merit function specifies the objective function along the direction of a line search, i.e., it tells the "merit" of each trial point in the line search. One might think that the original objective function $F(\mathbf{x})$ should be used, but that is not always best in constrained optimization. The call-back assistant is introduced because when the directional line search algorithm calls the line search algorithm, that algorithm needs to know the original objective function $F(\mathbf{x})$ and possible constraints. To facilitate this, the following calls take place in Rts:

1. The directional line search algorithm, such as FORM, selects a search direction.
2. The directional line search algorithm, such as FORM, calls the line search algorithm, such as golden section and passes itself as an instantiation of the pure virtual callback assistant class.
3. The line search algorithm, such as golden section, calls the evaluate-objective method of the call-back assistant, i.e., FORM in this case, each time it needs to evaluate the objective, or its gradient.
4. In the evaluate-objective method in the directional line search algorithm, such as FORM, the merit function is being called with values for the original objective and possible constraints.
5. The merit function uses the original objective and possible constraints to evaluate the objective of the line-search along the current search direction.

While the step size, s_n, in Eq. (1) is determined by line search algorithms, the search direction, $\mathbf{d}_n$, is determined by formulas involving the gradient vector, ∇F.

Steepest Descent Search Direction

The steepest descent method for selecting the search direction is easy to understand, but often yields slow convergence. It is based on the recognition that the objective function decreases fastest in the direction of its negative gradient:

$$\mathbf{d}_n = -\nabla F(\mathbf{x}_n) \tag{2}$$

Conjugate Gradient Search Direction

A significant improvement to the efficiency of the steepest descent method is obtained by changing the search direction by extracting information about the Hessian from consecutive gradients:

$$\mathbf{d}_n = -\nabla F(\mathbf{x}_n) + \beta \cdot \nabla F(\mathbf{x}_{n-1}) \tag{3}$$

where

$$\beta = \frac{\left\|\nabla F(\mathbf{x}_n)\right\|}{\left\|\nabla F(\mathbf{x}_{n-1})\right\|} \tag{4}$$

While the steepest descent method cannot be guaranteed to converge the conjugate gradient method and other methods that somehow create second-order approximations by employing consecutive gradients are guaranteed to converge, at least to a local optimum, if it exists.

Quasi-Newton Search Direction

A second-order Taylor expansion of the objective function leads to Newton's optimization algorithm:

$$\mathbf{d}_n = -\left[\mathbf{H}_n\right]^{-1} \nabla F_n \tag{5}$$

where $\mathbf{H}_n \equiv \partial^2 F / \partial \mathbf{x}^2$ is the Hessian matrix. Quasi-Newton methods attempt to overcome the difficulties in computing the Hessian, $\mathbf{H}$, by making use of the gradient vector at previous steps to approximate it. The following subsections display some of the most popular approximations. The following notation is employed:

$$\Delta\mathbf{x} = \mathbf{x}_{n+1} - \mathbf{x}_n \tag{6}$$

$$\Delta\mathbf{g} = \nabla F_{n+1} - \nabla F_n \tag{7}$$

The Davidon, Fletcher, Powell (DFP) Hessian Approximation

Here the Hessian is approximated by

$$\mathbf{H}_{n+1} \approx \left(\mathbf{I} - \frac{\Delta\mathbf{g}\Delta\mathbf{x}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}}\right)\mathbf{H}_n\left(\mathbf{I} - \frac{\Delta\mathbf{x}\Delta\mathbf{g}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}}\right) \tag{8}$$

where $\mathbf{I}$ is the identity matrix. This implies that the inverse of the Hessian is approximated by

$$\left[\mathbf{H}_{n+1}\right]^{-1} \approx \left[\mathbf{H}_n\right]^{-1} + \frac{\Delta\mathbf{x}\Delta\mathbf{x}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}} - \frac{\left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}\Delta\mathbf{g}^T\left[\mathbf{H}_n\right]^{-T}}{\Delta\mathbf{g}^T\left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}} \tag{9}$$

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) Hessian Approximation

Here the Hessian is approximated by

$$\mathbf{H}_{n+1} \approx \mathbf{H}_n + \frac{\Delta\mathbf{g}\Delta\mathbf{g}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}} - \frac{\mathbf{H}_n\Delta\mathbf{x}(\mathbf{H}_n\Delta\mathbf{x})^T}{\Delta\mathbf{x}^T\mathbf{H}_n\Delta\mathbf{x}} \tag{10}$$

which implies that the inverse of the Hessian is approximated by

$$\left[\mathbf{H}_{n+1}\right]^{-1} \approx \left(\mathbf{I} - \frac{\Delta\mathbf{g}\Delta\mathbf{x}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}}\right)\left[\mathbf{H}_n\right]^{-1}\left(\mathbf{I} - \frac{\Delta\mathbf{g}\Delta\mathbf{x}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}}\right) + \frac{\Delta\mathbf{x}\Delta\mathbf{x}^T}{\Delta\mathbf{g}^T\Delta\mathbf{x}} \tag{11}$$

The Broyden Hessian Approximation

Here the Hessian is approximated by

$$\mathbf{H}_{n+1} \approx \mathbf{H}_n + \frac{\Delta\mathbf{g} - \mathbf{H}_n\Delta\mathbf{x}}{\Delta\mathbf{x}^T\Delta\mathbf{x}}\Delta\mathbf{x}^T \tag{12}$$

which implies that the inverse of the Hessian is approximated by

$$\left[\mathbf{H}_{n+1}\right]^{-1} \approx \left[\mathbf{H}_n\right]^{-1} + \frac{\left(\Delta\mathbf{x} - \left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}\right)\Delta\mathbf{x}^T\left[\mathbf{H}_n\right]^{-1}}{\Delta\mathbf{x}^T\left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}} \tag{13}$$

The Symmetric Rank-one (SR1) Hessian Approximation

Here the Hessian is approximated by

$$\mathbf{H}_{n+1} \approx \mathbf{H}_n + \frac{(\Delta\mathbf{g} - \mathbf{H}_n\Delta\mathbf{x})(\Delta\mathbf{g} - \mathbf{H}_n\Delta\mathbf{x})^T}{(\Delta\mathbf{g} - \mathbf{H}_n\Delta\mathbf{x})^T\Delta\mathbf{x}} \tag{14}$$

which implies that the inverse of the Hessian is approximated by

$$\left[\mathbf{H}_{n+1}\right]^{-1} \approx \left[\mathbf{H}_n\right]^{-1} + \frac{\left(\Delta\mathbf{x} - \left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}\right)\left(\Delta\mathbf{x} - \left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}\right)^T}{\left(\Delta\mathbf{x} - \left[\mathbf{H}_n\right]^{-1}\Delta\mathbf{g}\right)^T\Delta\mathbf{g}} \tag{15}$$

Constraints

The algorithm in Eq. (1) is maintained when constraints are present, and so are the search direction formulas above. However, the resulting search directions typically differ because the objective function is reformulated to include the constraints.

Method of Lagrange Multipliers

To derive this method, consider the objective function $F(\mathbf{x})$ together with the equality constraint $f(\mathbf{x})=0$. The constraint is added to the original objective function to form the Lagrange function:

$$L(\mathbf{x},\lambda) = F(\mathbf{x}) + \lambda \cdot f(\mathbf{x}) \tag{16}$$

Why is this new objective function useful? First, as usual, set the derivative of the function equal to zero as an optimality condition:

$$\nabla L(\mathbf{x},\lambda) = 0 \tag{17}$$

The first part of this derivative is:

$$\nabla_{\mathbf{x}} L(\mathbf{x},\lambda) = \nabla F(\mathbf{x}) + \lambda \cdot \nabla f(\mathbf{x}) = 0 \tag{18}$$

and ensures stationarity: Study a figure and show that this is exactly what identifies an optimum point. Then realize that the last derivative enforces the constraint(s):

$$\nabla_{\lambda} L(\mathbf{x},\lambda) = \lambda \cdot \nabla f(\mathbf{x}) = 0 \tag{19}$$

Multi-constrained problems are handled by adding Lagrange multipliers:

$$L(\mathbf{x},\lambda) = F(\mathbf{x}) + \sum_{j=1}^{M} \lambda_j \cdot f_j(\mathbf{x}) \tag{20}$$

where M is the number of constraints.

Augmented Lagrangian Method

The penalty method minimizes:

$$L(\mathbf{x},\mu) = F(\mathbf{x}) + \mu \cdot \sum_{j=1}^{M} f_j(\mathbf{x})^2 \tag{21}$$

where the penalty value μ gets larger and larger in the iterations. The augmented Lagrangian method uses this objective function:

$$L(\mathbf{x},\mu,\boldsymbol{\lambda}) = F(\mathbf{x}) + \mu \cdot \sum_{j=1}^{M} f_j(\mathbf{x})^2 - \sum_{j=1}^{M} \lambda_j \cdot f_j(\mathbf{x})^2 \tag{22}$$

where the value of each constraint-specific λ in the next iteration is $\lambda_j - \mu \cdot f_j(\mathbf{x}_i)$. In fact, λ is an increasingly accurate estimate of the Lagrange multiplier. Unlike the penalty method, the augmented Lagrangian method does not require $\mu \to \infty$.

Optimality Conditions

Optimality conditions are useful in convergence checks and to develop optimization algorithms. For unconstrained problems, the first-order necessary optimality condition is that the gradient of the objective function is zero. The second-order necessary optimality condition is that the Hessian matrix is positive semi-definite. The second-order sufficient optimality condition is that the Hessian matrix is positive definite. With reference to variational calculus and energy methods, a point that satisfies the necessary conditions is a stationary point. For constrained problems, the Karush-Kuhn-Tucker (KKT) necessary conditions are developed. Before proceeding, it is assumed that the optimum point is regular, i.e., that the gradient vectors of the active constraints are linearly independent. The number of linearly independent vectors in the space of design variables cannot exceed the number of design variables. Under this assumption, the Lagrange function is established:

$$L(\mathbf{x},\boldsymbol{\lambda}) = F(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}) \tag{23}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers associated with each constraint. Setting the derivatives of the Lagrange function equal to zero yields the first-order necessary optimality conditions:

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x},\boldsymbol{\lambda}) = 0 \quad &\Rightarrow \quad \nabla F(\mathbf{x}) + \boldsymbol{\lambda}^T \nabla \mathbf{f}(\mathbf{x}) = 0 \\ \nabla_{\boldsymbol{\lambda}} L(\mathbf{x},\boldsymbol{\lambda}) = 0 \quad &\Rightarrow \quad \mathbf{f}(\mathbf{x}) = 0 \end{aligned} \tag{24}$$

The KKT conditions in Eq. (24) form an (N+M)-dimensional system of usually nonlinear equations in the N+M unknowns. The second-order sufficient optimality condition is that the Hessian of the Lagrange function is positive definite. The optimality conditions identify local optima. In general, it cannot be guaranteed that the point is a global optimum unless the objective function is convex. In fact, if the problem is convex then the KKT first-order conditions are both necessary and sufficient. In practice, this is addressed by trying different starting points in the space of design variables.