Numerical Optimization

Quantitative Macroeconomics [Econ 5725]

Raül Santaeulàlia-Llopis

Washington University in St. Louis

Spring 2016

1. Introduction

2. Direct Search Methods
   - Golden Search Method
   - Nelder-Mead Search Method
   - Multidirectional Search Method, Torczon (1989)

3. Newton-Raphson Method

4. Quasi-Newton Methods
   - Method of Steepest Ascent
   - Davidson-Fletcher-Powell (DFP) method
   - Broyden-Fletcher-Goldfarb-Shano (BFGS)

- We want to solve finite-dimensional optimization problems. [1]

- Our goal:

$$\max_{x \in X \subseteq \Re^n} \; f(x)$$

  where f is the objective function, X is the feasible set, and x^*, if it exists, a maximum.

[1] These slides borrow from **?**, **?** and **?**.

- Remark 1: the first order conditions of an unconstrained problem pose a rootfinding problem. That is, we can solve our maximization with the rootfinding algorithms we have already discussed.

- Remark 2: the Karush-Kuhn-Tucker first-order necessary conditions of a constrained optimization problem pose a complementarity problem.

- **Weiertrass theorem**: if f is continuous and X is nonempty, closed and bounded, then f. has a maximum on X.

  - x^* is a **local maximum** of f if there is an ϵ-neighborhood N of x^* such that $f(x^*) \geq f(x)\ \forall x \in N \cap X$.

  - x^* is a **strict local maximum** of f if, additionally, $f(x^*) > f(x)$ $\forall x \neq x^* \in N \cap X$.

  - If x^* is a local maximum of f in the interior of X and f is 2ce differentiable there, then $f'(x^*) = 0$ and $f''(x^*)$ is negative semidefinite.

  - Conversely, if $f'(x^*) = 0$ and $f''(x^*)$ is negative semidefinite in an ϵ-neighborhood of x^* contained in X, then x^* is a local maximum; if, additionally, $f''(x^*)$ is negative definite, then x^* is a strict local maximum.

  - If f is concave, X is convex, and x^* is a local maximum of f, then x^* is a global maximum of f on X.

Direct Search Methods

- Direct search methods are derivative-free methods useful if f is rough or has expensive (to compute) derivatives.
    - They are definitely slow
    - Convergence not guaranteed

Golden Search Method

- Assume a univariate maximization problem bounded in $[a, b]$.
    - Pick $x_1 < x_2$ in $[a, b]$ and evaluate f at x_1 and x_2
    - Replace the original interval with $[x_1, b]$ if $f(x_1) < f(x_2)$
    - Replace the original interval with $[a, x_2]$ if $f(x_1) \geq f(x_2)$
- A local maximum must be contained in the new interval because the endpoints of the new interval have smaller function values than a point on the interval's interior.
- We can repeat this procedure, producing a sequence of progressively smaller intervals that are **guaranteed** to contain a **local maximum**.
- The golden search method is **guaranteed** to find the **global maximum** when the function is **concave**.

- How do we choose the interior evaluation points x_1 and x_2?

- Two criteria:

    - The length of the new interval should be independent of whether the upper or lower bound is replaced

    - On successive iterations, one should be able to reuse an interior point from the previous interaction so that only one new function evaluation is preformed per iteration.

- These conditions are uniquely satisfied by selecting:

$$x_i = a + \alpha_i(b - a),$$

where

$$\alpha_1 = \frac{3 - 5^{.5}}{2} \quad \text{and} \quad \alpha_2 = \frac{5^{.5} - 1}{2}$$

The value α_2 is known as the golden ratio, an irrational constant (fascinating for many), defined as the positive root of $\frac{a+b}{a} = \frac{a}{b} =$ Golden ratio.

Nelder-Mead Search Method

- This is the most famous simplex based direct search method.
- The simplex is so-named because it represents the simplest possible polytope in any given space:[2] a simplex in 1D is a line segment (1-simplex), a simplex in 2D is a triangle (2-simplex), a simplex in 3D is a tetrahedro (3-simplex) n, simplex in 4D pentachoron (4-simplex), etc.
- Specifically, an n-simplex is an n-dimensional polytope with n+1 vertices of which the simplex is the convex hull.

[2] Recall that polytopes are geometric objects with flat sides (e.g. polytopes of two dimensions are polygones, and in three dimensions polyhedrons).

- A simplex based method constructs an evolving pattern of $n+1$ points in $\Re^n$ that are viewed as the vertices of a simplex.

- The iterative scheme forms a new simplex at each iteration by reflecting away from the vertex with the smallest value of f, or by contracting toward the vertex with he largest value of f. This way, the angles of every simplex remain the same throughout, even though the simplex may grow or decrease in size.

- At each iteration of the Nelder Mead algorithm, we have a current simplex defined by its $n+1$ vertices, each a point in $\Re^n$, along with the corresponding values of f.

- Iteration k begins by ordering and labeling the current set of vertices as

$$x_1^k, ..., x_{n+1}^k$$

such that

$$f_1^k \leq f_2^k \leq ... \leq f_{n+1}^k$$

where f_i^k denotes $f(x_i^k)$.

- Because we seek to minimize f we refer to x_1^k as the *best* point and to x_{n+1}^k as the *worst* point.

- After 'calculating one or more trial points' and evaluating f at these points, the kth iteration generates a set of $n+1$ vertices that define a different simplex for the next iteration.

- There are four possible operations that define those calculations:

  - These operations are: *reflection*, *expansion*, *contraction* and *shrinkage*, each associated with a scalar parameter.

  - The coefficients (scalar parameters) of *reflection*, *expansion*, *contraction* and *shrinkage* are respectively denoted by ρ, χ, γ and σ and they satisfy $\rho > 0, \chi > 1, 0 < \gamma < 1$ and $0 < \sigma < 1$.

  - The standard, nearly universal, choices for these parameters are:

    $$\rho = 1, \quad \chi = 2, \quad \gamma = .5, \text{ and } \sigma = .5$$

    The simplex shape undergoes already a noticeable change during an expansion or contraction with these standard coefficients.

- The Nelder-Mead iteration has 2 possible outcomes:

  1. A single new vertex, the accepted point that replaces x_{n+1} (the *worst* point) in the set of vertices for the next iteration; or,

  2. If a shrink is performed, a set of n new points that, together with x_1, form the simplex at the next iteration.

- A kind of 'search direction' is defined by x_{n+1} and x, the centroid of all vertices except x_{n+1}.

- The Nelder-Mead algorithm:

  Order Order the $n+1$ vertices to satisfy $f_1^k \leq f_2^k \leq ... \leq f_{n+1}^k$ using some consistent tie-breaking rule.

  Reflection Compute the *reflection point* x_r from

  $$x_r = x + \rho\ (x - x_{n+1})$$

  where x is the centroid of the n best vertices (all except x_{n+1}), i.e., $x = \sum_{i=1}^{n} \frac{x_i}{n}$. Evaluate $f_r = f(x_r)$. If $f_1 \leq f_r \leq f_n$, accept the reflected point x_r and terminate the iteration. Otherwise, if $f_r < f_1$ **expand** and if $f_r \geq f_n$ **contract**.

- (Continued)

  Expand If $f_r < f_1$, calculate the *expansion point* x_e from

  $$x_e = x + \chi\ (x_r - x)$$

  and evaluate $f_e = f(x_e)$. If $f_e < f_r$, accept x_e and terminate the iteration; otherwise, if $f_e \geq f_r$, accept x_r and terminate the iteration.

- (Continued)

  Contract If $f_r \geq f_n$, perform a *contraction* between x and the better of x_{n+1} and x_r.

  - **(i) Outside**. If $f_n \leq f_r < f_{n+1}$ (i.e., x_r is strictly better than x_{n+1}), perform an *outside contraction*, that is,

    $$x_c = x + \gamma\,(x_r - x)$$

    and evaluate $f_c = f(x_c)$. If $f_c \leq f_r$, accept x_c and terminate the iteration; otherwise, go to step 5 (perform a shrink).
  - **(ii) Inside**. If $f_r \geq f_{n+1}$ (i.e., x_{n+1} is strictly better than x_r), perform an *inside contraction*, that is,

    $$x_c' = x - \gamma\,(x - x_{n+1})$$

    and evaluate $f_c' = f'(x_c)$. If $f_c' \leq f_{n+1}$, accept x_c' and terminate the iteration; otherwise, go to step 5 (perform a shrink).

- (Continued)

  Shrinking Perform a shrink step. Define n new vertices from

  $$v_i = x_1 + \sigma\,(x_i - x_1), \quad i = 2, ..., n+1$$

  and evaluate f at these points. The vertices of the simplex at the next iteration consist of $x_1, v_2, ..., v_{n+1}$.

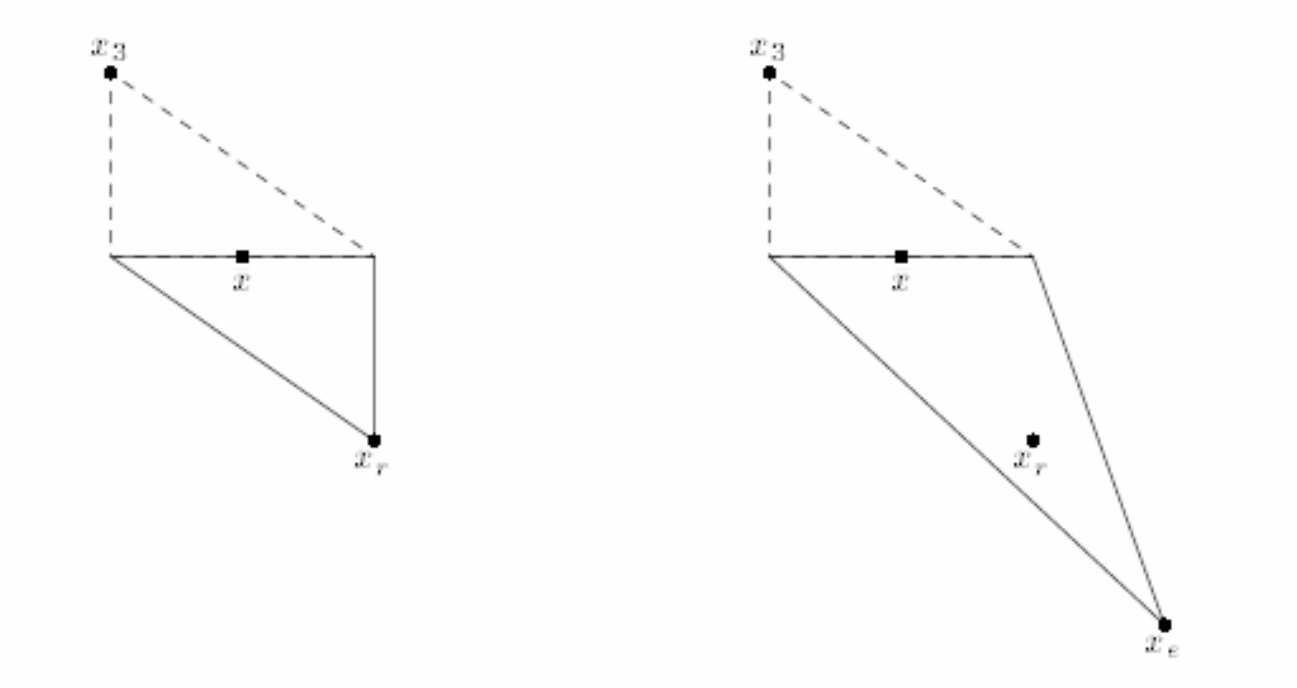

Figure: Nelder-Mead 2D simplices after a reflection and an expansion step. The original simplex is shown with a dashed line

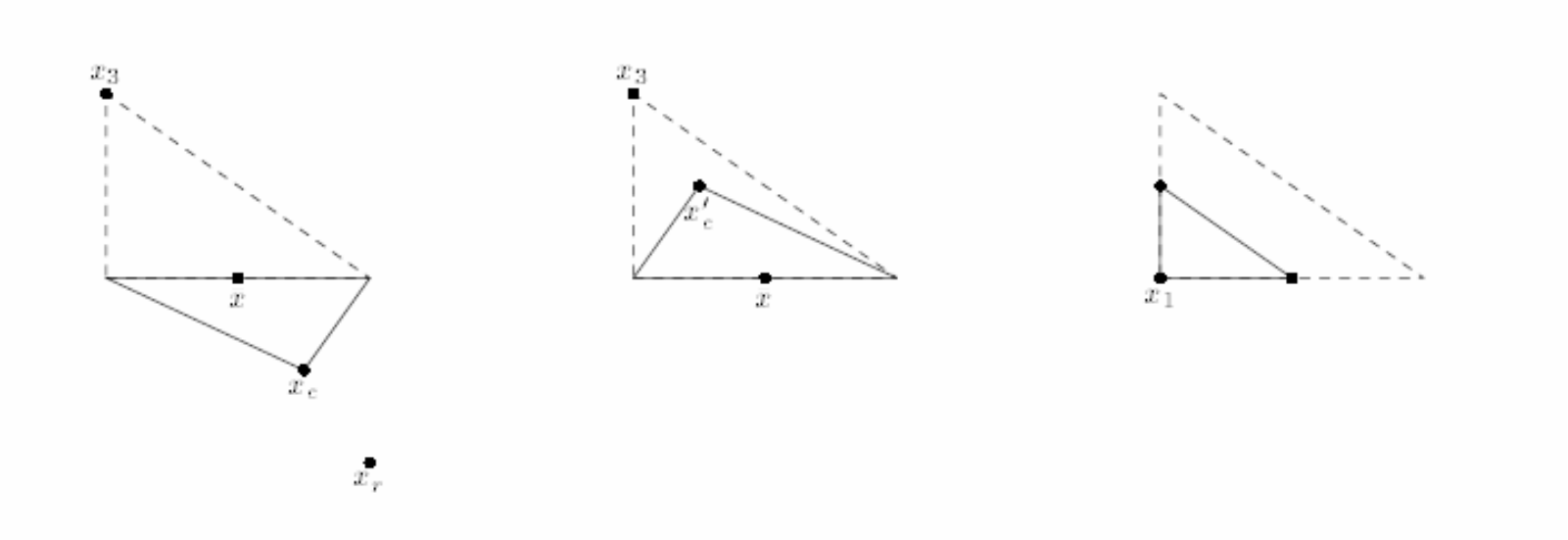

Figure: Nelder-Mead 2D simplices after an outside contraction, an inside contraction and a shrink

- The Nelder-Mead method has several interesting properties:

  - A successful non-shrink iteration produces a new vertex whose function value is strictly less than f_{n+1}. This *simple decrease* requirement is **much weaker** than those usually imposed in optimization convergence theory
  - It is particularly **parsimonious** in function evaluations per iteration (compared to other direct search methods): it requires one evaluation of f in step 2, 3 evaluations when termination occurs in step 3 or 4, and $n + 2$ evaluations if a shrink step occurs.
  - The next simplex is determined by the coordinates of the simplex vertices and the order information about the vertices, **not numerical function values**.
  - In the expand step, the method in the original Nelder-Mead paper accepts x_e if $f(x_e) < f_1$ and accepts otherwise. Standard practice today, as stated above, accepts the better of x_r and x_e if both give a strict improvement over x_1.

- To completely specify the Nelder-Mead algorithm, we need to define an **initial simplex** and **termination criteria**.

Initial Simplex A successful non-shrink iteration produces a new vertex whose function value is strictly less than f_{n+1}. This *simple decrease* requirement is **much weaker** than those usually imposed in optimization convergence theory

- If we knew well the function being optimized, we can specify $n + 1$ suitable starting vertices.
- Otherwise, it is customary to specify a starting point in $\Re^n$ that is taken as one of the initial simplex vertices, then, the other n vertices are generated by: either perturbing the starting point by a specified step along the n coordinate directions; or, creating a regular simplex with specified edge length and orientation.

- (continued)

Termination Criteria For any non-derivative method, the issue of termination is problematical as well as highly sensitive to problem scaling.

- Since gradient information is unavailable, it is probably impossible to verify closeness to optimality simply by sampling f at a finite number of points.
- Most implementation of direct search methods terminate based on two criteria that reflect the progress of the algorithm: either the function values at the vertices are close, or the simplex has become very small. For example, Woods and Torczon suggest termination when the current vertices $x_1, ..., x_{n+1}$ satisfy

$$\max_{2 \leq i \leq n+1} ||x_i - x_1|| \leq \epsilon \ \max(1, ||x_1||)$$

where ϵ is a tolerance.

External Figure 1: Himmelblau's function [3D with contours], 4 identical local minima

External Figure 2: Nelder-Mead algorithm to the Himmelblau's function [2D with contours]

Multidirectional Search Method, Torczon (1989)

- Each iteration is associated with a current simplex whose best (with lowest function value) vertex is so labeled. Reflection, expansion and contraction are defined as in the Nelder-Mead method, but these now involve the n edges of the simplex emanating from the best vertex, so that the entire simplex is reflected, expanded and contracted (Torczon (1989)).

- The iteration scheme succeeds when it finds a point of strict improvement over the best vertex, in contrast to the much weaker condition in a Nelder-Mead iteration of finding a strict improvement compared to the worst point.

- Useful for efficiency in a parallel environment.

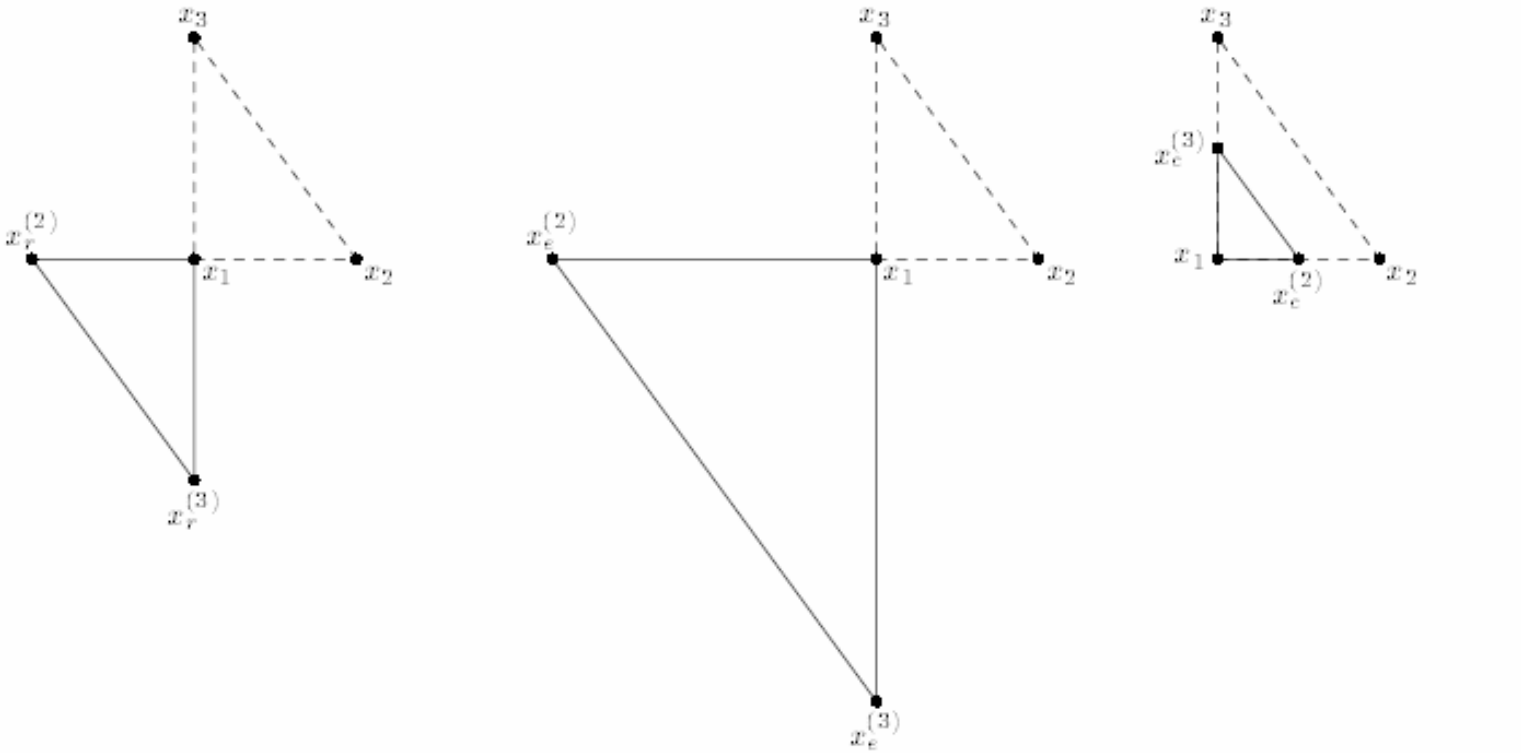

Figure: Multidirectional search reflection, expansion and contraction.

Newton-Raphson Method

- The Newton-Raphson method uses successive quadratic approximations to the objective in the hope that the maxima of the approximants will converge to the maximum of the objective.

- The Newton-Raphson method is identical to applying Newton's method to compute the root to the gradient of the objetive function.

- It begins with the analyst supplying a guess x^0 for the maximum of f. Given x^k, the subsequent iterate x^{k+1} is computed by maximizing the second-order Taylor approximation to f about x^k.

$$f(x) \approx f(x^k) + f'(x^k)(x - x^k) + \frac{1}{2}(x - x^k)^T f''(x^k)(x - x^k)$$

- Solving the first-order condition,

$$f'(x^k) + f''(x^k)(x - x^k) = 0$$

that yields the iteration rule,

$$x^{k+1} \leftarrow x^k - [f''(x^k)]^{-1} \, f'(x^k)$$

- Convergence,
    - The Newton-Raphson method converges if f is 2ce continuously differentiable and if the initial guess, x^0, is *'sufficiently'* close to a local maximum of f at which the Hessian, f'', is negative definite.
    - The Newton Rapshon method can be robust to the starting value if f is well behaved, for example, if f is globally concave.
    - The Newton Rapshon method, however, can be very sensitive if the function is not globally concave. The Hessian f'' must also be well behaved at the optimum.

- Remarks,

  - The Newton-Raphson method requires computation of both the first and second derivatives of the objective function.

  - The Newton Rapshon method offers no guarantee that the objective function value may be increased in the direction of the Newton step — only guaranteed if the Hessian $f''(x^k)$ is negative definite; otherwise, one may move toward a saddle point of f (if the Hessian is indefinite) or even a minimum (if the Hessian is positive definite).

Quasi-Newton Methods

- Similar to Newton-Raphson but replace the Hessian of the objective function (or its inverse) with a negative definite approximation, guaranteeing that the function value can be increased in the direction of the Newton step.

- This **approximation to the inverse Hessian** also eases the burden of implementation and the cost of manipulation by avoiding to perform a linear solve, and instead, employ updating rules that do not require second-derivative information.

- Hence, in quasi-Newton methods the **direction search** takes the form:

$$d^k = -B^k f'(x^k)$$

where B^k is an approximation to the inverse Hessian of f at the kth iterate x^k. The vector d^k is called the Newton or quasi-Newton step.

- The more robust quasi-Newton methods do **not** necessarily take the **full Newton step**, but shorten it or lengthen it in order to obtain improvement in the objective function.

- This can be done with a *line search* in which one seeks a step length $s > 0$ that (nearly) maximizes $f(x^k + s\, d^k)$. Given the computed step length s^k, one updates the iterate as follows:

$$x^{k+1} = x^k + s^k\, d^k$$

- *Small digression on Line search methods:*

  - *Golden Search* is reliable but computationally inefficient.

  - *Armijo approach*. The idea is to find the minimum power j such that

$$\frac{f(x+sd)-f(x)}{s} \geq \mu f'(x)^T d$$

    where $s = \rho^j$, $0 < \rho < 1$, and $0 < \mu < .5$.

    - The LHS is the slope of the line from the current iteration point to the candidate for the next iteration.
    - The RHS is the directional derivative at x in the search direction d, that is, the instantaneous slope at the current iteration point.
    - That is, this approach is to backtrack from a step size of 1 until the slope on the LHS is a given fraction μ of the slope on the RHS.
    - The *Armijo approach* is both a method for selecting candidate values of the step size s and a stopping rule.

- (continued)
  - *Goldstein search*. The idea is to find any value of s that satisfies
    - $$\mu_0 f'(x)^T d \leq \frac{f(x+sd) - f(x)}{s} \leq \mu_1 f'(x)^T d$$

      for some values of $0 < \mu_0 \leq .5 \leq \mu_1 < 1$.
    - The *Goldstein* criterion is simply a stopping rule.

- Quasi-Newton methods differ in how the inverse Hessian approximation B^k is constructed and updated:
  - *Method of the Steepest Ascent*, $B^k = -I$
  - Using some curvature information:
    - *Davidson-Fletcher-Powell (DFP) method*
    - *Broyden-Fletcher-Goldfarb-Shano (BFGS) method*

Method of Steepest Ascent

- *Method of Steepest Ascent*
  - Set the Hessian to the identity matrix, $B^k = -I$
  - This approach leads to a Newton step that is identical to the gradient of the objective function at the current iterate,

$$d^k = f'(x^k)$$

- (continued)

  - This choice of gradient as step is intuitively appealing because the gradient always points in the direction which, to a first order, promises the greatest increase in f. For this reason, this quasi-Newton method is called the *method of steepest ascent*.

  - This method is simple to implement, but it is numerically less efficient in practice than other quasi-Newton methods that incorporate information regarding the curvature of the objective function.

- The information about the curvature of f is used to produce a sequence of inverse Hessian estimates that satisfy two conditions:

  - First, given that, for the Newton step

    $$d^k \approx f''^{-1}(x^k)\left[f'(x^k + d^k) - f'(x^k)\right]$$

    the inverse Hessian estimate B^k is required to satisfy the so-called quasi-Newton condition:

    $$d^k = B^{k+1}\left[f'(x^k + d^k) - f'(x^k)\right]$$

  - Second, the inverse Hessian estimate is required to bet both symmetric and negative definite, as must be true of the inverse Hessian at a local maximum. The negative definiteness of the Hessian estimate assures that the objective function value can be increased in the direction of the Newton step.

Davidson-Fletcher-Powell (DFP) method

- *Davidson-Fletcher-Powell (DFP) method*
  - The *DFP* method uses the updating scheme

$$B \leftarrow B + \frac{dd^T}{d^T u} - \frac{Buu^T B^T}{u^T Bu}$$

where

$$d = x^{k+1} - x^k \quad \text{and} \quad u = f'(x^{k+1}) - f'(x^k)$$

Broyden-Fletcher-Goldfarb-Shano (BFGS)

- *Broyden-Fletcher-Goldfarb-Shano (BFGS) method*
  - The *BFGS* method uses the updating scheme

$$B \leftarrow B + \frac{1}{d^T u}\left(wd^T + dw^T - \frac{w^T u}{d^T u} dd^T\right)$$

where

$$w = d - Bu$$

- Quasi-Newton methods are susceptible to certain problems. Notice that in both update formulas we divide by $d^T\ u$.

- If this value becomes very small in absolute value, numerical instabilities will appear. A rule to monitor whether it becomes too small or not is,

$$|d^T u| < \epsilon ||d||\ ||u||$$

  where ϵ is the precision of the computer.