Despina Zoras 083 209 0596
despinazoras@gmail.com

Nonlinear unconstrained optimization

School of Computational and Applied Mathematics,
University of the Witwatersrand, Johannesburg

Lecturer : Dr M Ali

1 Introduction

Nonlinear Programming (NLP) is the broad area of applied mathematics that addresses optimization problems when nonlinearity in the functions are involved. In this section we introduce the problem, and making reference to the smooth case, we review the standard optimality conditions that are at the basis of most algorithms for its solutions. Then, we give basic notions concerning the performance of algorithms, in terms of convergence and rate of convergence.

1.1 Problem definition

We consider the problem of determining the value of a vector of *decision variables* $x \in R^n$ that minimizes an *objective function* $f : R^n \to R$, when x is required to belong to a *feasible set* $S \subseteq R^n$; that is we consider the problem:

$$\min_{x \in S} f(x). \tag{1.1}$$

Two cases are of main interest:
- the feasible set S is R^n, so that Problem (1) becomes:

$$\min_{x \in R^n} f(x); \tag{1.2}$$

in this case we say that Problem (1.1) is *unconstrained*. More in general, Problem (1) is unconstrained if S is an open set. The optimality conditions for unconstrained problems stated in 1.2 hold also in the general case. Here for simplicity we assume that $S = R^n$.
- the feasible set is described by *inequality* and/or *equality constraints* on the decision variables:

$$S = \{x \in R^n : g_i(x) \leq 0,\ i = 1, \ldots, p; \quad h_j(x) = 0,\ j = 1, \ldots, m\};$$

then Problem (1.1) becomes:

$$\min_{x \in R^n} \quad f(x)$$

subject to

$$g(x) \leq 0, \qquad h(x) = 0, \tag{1.3}$$

with $h : R^n \to R^m$ and $g : R^n \to R^p$. In this case we say that Problem (1.1) is *constrained.*

Problem(1.1) is a *Nonlinear Programming* (NLP) problem when at least one, among the problem functions f, $g_i, i = 1, \ldots, p$; $h_j = 1, \ldots, m$, is nonlinear in its argument x.

Usually it is assumed that in Problem (1.3) the number m of equality constraints is not larger than the number n of decision variables. Otherwise the feasible set could be empty, unless there is some dependency among the constraints. If only equality (inequality) constraints are present, Problem (1.3) is called an *equality (inequality)* constrained NLP problem.

In the following we assume that the problem functions f, g, h are at least *continuously differentiable* (hence smoothness is assumed) in R^n.

When f is a convex function and S is a convex set, Problem (1.1) is a *convex* NLP problem. In particular, S is convex if the equality constraint functions h_j are affine and the inequality constraint functions g_i are convex. Convexity adds a lot of structure to the structure to the NLP problem, and can be exploited widely both from the theoretical and the computational point of view. If f is convex quadratic and h, g are affine, we have a *Quadratic Programming* problem. This is a case of special interest. Here we will confine ourselves to general NLP problems, without convexity assumptions.

A point $x^* \in S$ is a *global solution* of Problem (1.1) if $f(x^*) \leq f(x)$, for all $x \in S$; it is a *strict* global solution if $f(x^*) < f(x)$, for all $x \in S$, $x \neq x^*$. A main *existence result* for a constrained problem is that a global solution exists if S is compact (Weierstrass Theorem). An easy consequence for unconstrained problems is that a global solution exists if the *level set* $\mathcal{L}_\alpha = \{x \in R^n : f(x) \leq \alpha\}$ is compact for some finite α.

A point $x^* \in S$ is a *local solution* of Problem (1.1) if there exists an open neighborhood $\mathcal{B}_{x^*}$ of x^* such that $f(x^*) \leq f(x)$, for all $x \in S \cap \mathcal{B}_{x^*}$; it is a *strict* local solution of $f(x^*) < f(x)$, for all $x \in S \cap \mathcal{B}_{x^*}, x \neq x^*$. Of course, a global solution is also a local solution.

To determine a global solution of a NLP problem is in general a very difficult task. Usually, NLP algorithms are able to determine only local solutions. Nevertheless, in practical applications, also to get a local solution can be of great worth.

1.2 Optimality conditions

Local solutions must satisfy *necessary optimality conditions* (NOC). For the unconstrained Problem (1.2) we have the well know result of classical calculus:

Proposition 1.1 *Let x^* be a local solution of Problem (1.2), then*

$$\nabla f(x^*) = 0; \tag{1.4}$$

moreover, if f if twice continuously differentiable, then

$$d^T \nabla^2 f(x^*) d \geq 0, \quad \forall d \in R^n. \tag{1.5}$$

For the constrained problem (1.3), most of the NOC commonly used in the development of algorithms assume that at a local solution the constraints satisfy some qualification condition to prevent the occurrence of degenerate cases. These conditions are usually called *constraints qualifications* and among them the *linear independence* constraints qualification (LICQ) is the simplest and by far the most invoked.

Let $\hat{x} \in S$. We say that the inequality constraint g_i is *active* at $\hat{x}$ if $g_i(\hat{x}) = 0$. We denote by $I_a(\hat{x})$ the index set of inequality constraints active at $\hat{x}$:

$$I_a(\hat{x}) = \{i \in \{1, \ldots, p\} : g_i(\hat{x}) = 0\}. \tag{1.6}$$

Of course, any equality constraint h_j, is active at $\hat{x}$. LICQ is satisfied at $\hat{x}$ if the gradients of the active constraints, $\nabla g_{I_a}(\hat{x})$, are linearly independent.

Under LICQ for Problem (1.3) are stated making use of the (generalized) *Lagrangian function:*

$$L(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x), \tag{1.7}$$

where $\lambda \in R^p$, $\mu \in R^m$ are called (generalized Lagrange) *multipliers*, or *dual variables.*

The so called *Karush-Kuhn-Tucker* (KKT) NOC are stated as follows:

Proposition 1.2 *Assume that x^* is a local solution of Problem (1.3) and that LICQ holds at x^*; then multipliers $\lambda^* \geq 0$, μ^* exist such that:*

$$\nabla L(x^*, \lambda^*, \mu^*) = 0,$$

$$\lambda^{*T} g(x^*) = 0; \tag{1.8}$$

moreover, if f, g, h are twice continuously differentiable, then:

$$d^T \nabla_x^2 L(x^*, \lambda^*, \mu^*) d \geq 0, \ \forall y \in \mathcal{N}(x^*),$$

where:

$$\mathcal{N}(x^*) = \{d \in R^n : \nabla g_{I_a}(x^*)^T d = 0;\ \nabla h(x^*)^T d = 0\}.$$

A point x^* satisfying the NOC conditions (1.8) together with some multipliers λ^*, μ^* is called a KKT point.

If a point $x^* \in S$ satisfies a *sufficient optimality condition*, then it is a local solution of Problem (1.1). For general NLP problems, sufficient optimality conditions can be stated under the assumption that the problem functions are twice continuously differentiable, so that we have *second order sufficiency conditions* (SOSC). For the unconstrained Problem (1.2) we have the SOSC:

Proposition 1.3 *Assume that x^* satisfies the NOC of Proposition (1.1). Assume further that*

$$d^T \nabla^2 f(x^*) d > 0,\ \forall d \in R^n,\ d \neq 0, \tag{1.9}$$

that is that assume that $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local solution of Problem (1.2).

For the constrained Problem (1.3) similar SOSC exists but we will not discuss this here. However, we will discuss the first order condition later.

Remark 1.1: A main feature of convex problems is that a (strict) local solution is also a (strict) global solution. Moreover, when f is (strictly) convex and, if present, g_i are convex and h_i are affine, the NOC given in terms of first order derivatives are also sufficient for a point x^* to be a (strict) global solution.

Optimality conditions are fundamental in the solution of NLP problems. It is known that a global solution exists, the most straightforward method to employ them is as follows: find all points satisfying the first order necessary conditions, and declare as global solution the point with the smallest value of the objective function. If the problem functions are twice differentiable, we can also check the second order necessary condition, filtering out those points that do not satisfy it; for the remaining candidates, we can check a second order sufficient condition to find local minima.

It is important to realize, that except for very simple cases, using optimality conditions as described above *does not work*. The reason is that, even for an unconstrained problem, to find a solution of the system of equations $\nabla f(x) = 0$ is nontrivial; algorithmically, it is usually as difficult as solving the original minimization problem.

The principle context in which optimality conditions become useful is the development and analysis of algorithms. An algorithm for the solution of Problem (1.1) produces a sequence $\{x^k\}$, $k = 0, 1, \ldots$, of tentative solutions, and terminates when a stopping criterion is satisfied. Usually the stopping criterion is based on

satisfaction of necessary optimality conditions within a prefixed tolerance; moreover, necessary optimality conditions often suggest how to improve the current tentative solution x^k in order to get the next one x^{k+1}, closer to the optimal solution. Thus, necessary optimality conditions provide the basis for the convergence analysis of algorithms. On the other hand, sufficient optimality conditions play a key role in the analysis of the rate of convergence.

1.3 Convergence and rate of convergence

Let $\Omega \subset S$ be the subset of points that satisfy the first order NOC for problem (1.1). From a theoretical point of view, an optimization algorithm stops when a point $x^* \in \Omega$ is reached. From this point of view the set Ω is called the target set. Convergence properties are stated with reference to the target set Ω. In the unconstrained case, a possible target set is $\Omega = \{x \in R^n : \nabla f(x) = 0\}$ whereas in the constrained case Ω may be the set of KKT points satisfying (1.8).

Let $\{x^k\}, k = 1, 2, \ldots$ be the sequence of points produced by an algorithm. Then, the algorithm is globally convergent if a point x^* of $\{x^k\}$ exists such that $x^* \in \Omega$ for any starting point $x^0 \in R^n$; it is locally convergent if the existence of the limit point $x^* \in \Omega$ can be established only if the starting point x^0 belongs to some neighbourhood of Ω.

The notion of convergence stated before is the weakest that ensures that a point x^k arbitrarily close to Ω can be obtained for k large enough. In the unconstrained case this implies that

$$\lim_{k \to \infty} \inf \|\nabla f(x^k)\| = 0. \tag{1.10}$$

Nevertheless, stronger convergence properties can often be established. For instance that any sequence of $\{x^k\}$ possess a limit point, and any limit point of $\{x^k\}$ belongs to Ω; for the unconstrained case, this means that

$$\lim_{k \to \infty} \|\nabla f(x^k)\| = 0.$$

The strongest convergence requirement is that the whole sequence $\{x^k\}$ convergences to a point $x^* \in \Omega$. One of the key measures of performance of an algorithm is its rate of convergence. The most widely employed notion of rate of convergence is the Q-rate of convergence, that considers the quotient between two successive iterates given by

$$\frac{\|\underline{x}^{(k+1)} - \underline{x}^*\|}{\|\underline{x}^{(k)} - \underline{x}^*\|} \tag{1.11}$$

Definition 1.1

If $\{\underline{x}^{(k)}\} \in R^n$ suppose that $\{\underline{x}^{(k)}\} \rightarrow \underline{x}^*$ as $k \rightarrow \infty$. Then if r is the largest real number for which

$$\lim_{k\to\infty} \frac{\|\underline{x}^{(k+1)} - \underline{x}^*\|}{\|\underline{x}^{(k)} - \underline{x}^*\|^r} = \gamma, \quad \text{for} \quad 0 \leq \gamma < \infty, \tag{1.12}$$

the sequence is said to have *asymptotic rate of convergence* r, with *asymptotic error constant* γ.

If $r = 1$ and $\gamma = 0$ for sufficiently large k then the convergences is said to be Q-superlinear. If $r = 1$ and $\gamma < 1$ for sufficiently large k then the convergence is said to be Q-linear. If $r = 2$ for sufficiently large k then the convergences is said to be Q-quadratic, where γ is a positive constant, not necessarily less than 1. For in general, the rate of convergence is of Q-order r if there exists a positive constant γ such that (1.12) holds for all sufficiently large k. However, a Q-order larger than 2 is very seldom achieved. Algorithms of common use are either superlinear or quadratically convergent.

Examples 1.1

If $x^{(k)} = a^{2^k}$ where $a \in [0, 1)$ is a constant, i.e. $\{x^{(k)}\} = (a, a^2, a^4, a^8, \ldots\ldots)$, then it is clearly seen that as $k \rightarrow \infty, x^{(k)} \rightarrow 0$.
When $r = 2$, $\lim_{k\to\infty} \dfrac{\|\underline{x}^{(k+1)} - x^*\|}{\|\underline{x}^{(k)} - x^*\|^2} = \dfrac{a^{2^{k+1}}}{(a^{2^k})^2} = \dfrac{a^{2^{k+1}}}{a^{2^{k+1}}} = 1.$
Thus the convergence is quadratic with asymptotic error constant $\gamma = 1$. (Check that with $r = 2 + \epsilon$, $\lim_{k\to\infty} \dfrac{|x^{(k+1)}|}{|x^{(k)}|} = \infty$).

If $x^{(k)} = a^{2^{-k}}$ where $a < 0$ is a constant, i.e. $\{x^{(k)}\} = \left(a, a^{\frac{1}{2}}, a^{\frac{1}{4}}, a^{\frac{1}{8}}, \ldots\ldots\right)$, then it is clearly seen that as $k \rightarrow \infty, x^{(k)} \rightarrow 1$. (Repeatedly press the square root sign on a calculator.) To determine the rate of convergence choose $r = 1$ (this will turn out to be a good guess)

$$\lim_{k\to\infty} \frac{\|\underline{x}^{(k+1)} - x^*\|}{\|\underline{x}^{(k)} - x^*\|^2} = \frac{a^{2-(k+1)} - 1}{a^{2-k} - 1} = \frac{|a^{2-(k+1)} - 1| \cdot |a^{2-(k+1)} + 1|}{|a^{2-k} - 1| \cdot |a^{2-(k+1)} + 1|}$$

$$= \lim_{k\to\infty} \frac{a^{2-k} - 1}{|a^{2-k} - 1| \cdot |a^{2-(k+1)} + 1|} = \lim_{k\to\infty} \frac{1}{|a^{2-(k+1)} + 1|} = \frac{1}{2}.$$

If $r = 1 + \epsilon$ the limit is not defined. The sequence has linear convergence with asymptotic error constant $\gamma = \frac{1}{2}$.

Theorem 1.1: Theorem on superlinear convergence

If $\underline{x}^{(k)} \to \underline{x}^*$ superlinearly then

$$\lim_{k\to\infty} \frac{\|\underline{x}^{(k+1)} - \underline{x}^*\|}{\|\underline{x}^{(k)} - \underline{x}^*\|} = 1.$$

The converse is not true in general.

Proof

$$\frac{\|\underline{x}^{(k+1)} - \underline{x}^*\|}{\|\underline{x}^{(k)} - \underline{x}^*\|} = \frac{\|\underline{x}^{(k+1)} - \underline{x}^{(k)} + \underline{x}^{(k)} - \underline{x}^*\|}{\|\underline{x}^{(k)} - \underline{x}^*\|} \geq |\frac{\|\underline{x}^{(k+1)} - \underline{x}^{(k)}\|}{\|\underline{x}^{(k)} - \underline{x}^*\|} - 1|.$$

Let $k \to \infty$, the left hand side $\to 0$ by superlinearity and the result follows. Thus $\|\underline{x}^{(k+1)} - \underline{x}^{(k)}\| \to \|\underline{x}^{(k+1)} - \underline{x}^*\|$ as $k \to \infty$ and we obtain an estimate of the error from successive approximations.

2 Unconstrained optimization

In this chapter we consider algorithms for solving the *unconstrained* Nonlinear Programming problem (1.2)

$$\min_{x\in R^n} f(x),$$

where $x \in R^n$ is the vector of decision variables and $f : R^n \to R$ is the objective function. The algorithms described in this section can be easily adapted to solve the constrained Problem $\min_{x\in S} f(x)$ when S is an open set, provided that a feasible point $x^0 \in S$ is available.

In the following we assume for simplicity that the problem function f is twice continuously differentiable in R^n even if in many cases only once continuously differentiability is needed.

We also assume that the standard assumption ensuring the existence of a solution of Problem (1.2) holds, namely that:

Assumption 2.1 *The level set* $\mathcal{L}^0 = \{x \in R^n : f(x) \leq f(x^0)\}$ *is compact, for some* $x^0 \in R^n$.

The algorithm models treated in this section generate a sequence $\{x^k\}$, starting from x^0, by the following iteration

$$x^{k+1} = x^k + \alpha^k\, d^k, \tag{2.1}$$

where d^k is a *search direction* and α^k is a *stepsize* along d^k. The basic structure of the kth iteration of a minimization algorithm that determines α^k is given below:

(i) Determine a direction of search $\underline{d}^k$

(ii) Find α^k to minimize $f(\underline{x}^k + \alpha \underline{d}^k)$ with respect to α.

(iii) Set $\underline{x}^{k+1} = \underline{x}^k + \alpha^k \underline{d}^k$.

Different minimization methods select $\underline{d}^k$ in different ways in (i). Step (ii) is the one dimensional *sub-problem* carried out along the line $\underline{x}^{k+1} = \underline{x}^k + \alpha \underline{d}^k$. Therefore, methods differentiate in the way the direction and the stepsize are chosen. Of course different choices of d^k and α^k yield different convergence properties. Roughly speaking, the search direction affects the local behaviour of an algorithm and its rate of convergence whereas global convergence is often tied to the choice of the stepsize α^k.

Many methods are classified according to the information they use regarding the smoothness of the function. In particular we will briefly describe quasi-Newton methods, conjugate gradient methods and derivative free methods (direct search or zero the order methods).

Most methods determine α^k by means of a line search technique. Hence we start with a short review of the most used techniques and more recent developments in line search algorithms.

2.1 Line search algorithms

Line Search Algorithms (LSA) determine the stepsize α^k along the direction d^k. The aim of different choices of α^k is mainly to ensure that the algorithm defined by (2.1) results to be globally convergent, possibly without deteriorating the rate of convergence. A first possibility is to set $\alpha^k = \alpha^*$ with

$$\alpha^* = \arg \min_{\alpha} f(x^k + \alpha d^k), \tag{2.2}$$

that is α^k is the value that minimizes the function f along the direction d^k. However, unless the f has some special structure (being quadratic), an exact line search (2.2) is usually quite computationally costly and its use may not be worthwhile. Hence approximate methods (inexact line search) are used. In the cases when ∇f can be computed, we assume that the condition

$$\nabla f(x^k)^T d^k < 0, \quad \text{for all} \quad k \tag{2.3}$$

holds, namely we assume that d^k is a *descent direction* for f at x^k. Among the simplest LSAs is *Armijo's method*, reported in Algorithm 1.

Armijo's LSA finds a stepsize α^k that satisfies a condition of sufficient decrease of the objective function and implicitly a condition of sufficient displacement from the current point x^k.

In many cases it can be necessary to impose stronger conditions on the stepsize α^k. In particular *Wolfe's conditions* are widely employed. In this case, Armijo's condition

$$f(x^k + \alpha d^k) \leq f(x^k) + \alpha\rho\nabla f(x^k)^T d^k \tag{2.4}$$

is coupled either with Wolfe's condition

$$\nabla f(x^k + \alpha d^k)^T d^k > \sigma\nabla f(x^k)^T d^k, \tag{2.5}$$

Algorithm 1: Backtracking Line Search

Step (i) Choose $0 < \epsilon_1 < \epsilon_2 < 1$ and $\rho \in (0, 0.5)$; set $\alpha = 1$;

Step (ii) While $f(\underline{x}^k + \alpha\underline{d}^k) > f(\underline{x}^k) + \rho\alpha\nabla f(\underline{x}^k)^T\underline{d}^k$, do

$\alpha = \epsilon * \alpha$, for some $\epsilon \in (\epsilon_1, \epsilon_2) = (0.1, 0.5)$
(ϵ is chosen anew each time by the line search)

Step (iii) Terminate with $\alpha^k = \alpha$, set $x^{k+1} = x^k + \alpha^k d^k$.

or with the stronger one (Wolfe's condition)

$$\left|\nabla f\left(x^k + \alpha d^k\right)^T d^k\right| \leq \sigma\left|\nabla f\left(x^k\right)^T d^k\right|, \tag{2.6}$$

where $\sigma \in (\rho, 1)$, being ρ the value used in (2.4).

Conditions (2.4) was suggested by Armijo while

$$f(x^k + \alpha d^k) \leq f(x^k) + \alpha\rho\nabla f(x^k)^T d^k \tag{2.7}$$

and

$$f(x^k + \alpha d^k) \geq f(x^k) + \alpha(1-\rho)\nabla f(x^k)^T d^k \tag{2.8}$$

are known to have been suggested by Goldstein. The latter condition is used to exclude the left-hand extreme, where $\rho \in (0, \frac{1}{2})$ is fixed parameter. The condition $\sigma > \rho$ guarantees that (2.4) and (2.5) can be satisfied simultaneously. In practice ρ is chosen to be very small e.g., $\rho = 10^{-4}$. The condition (2.4) gives a simplest but efficient form of line search provided the step length are chosen appropriately using a so-called *backtracking* approach as described in Algorithm 1.

The initial step length α is chosen to be 1 in Newton and quasi-Newton methods, but can have different values for other methods such as the conjugate gradient method to seek appropriate $\alpha \in (0, \bar{\alpha})$. For example, the strategy is to interpolate a quadratic to the data f^{k-1}, f^k and $f'(0)$ and to define $\bar{\alpha}$ to be its minimizer. This strategy yields

$$\bar{\alpha} = \frac{2(f^k - f^{k-1})}{f'(0)}. \tag{2.9}$$

On the first iteration of the minimization method $f^{k-1} - f^k$ must be user supplied.

We now explain how the backtracking line search is implemented. If we take the initial step $\alpha^k = 1$, and then, if $x^k + d^k$ is not acceptable, 'backtrack' (reduce α^k) until an acceptable $x^k + \alpha^k d^k$ is found. Notice that the condition (2.5) is not implemented because the the backtracking strategy avoids excessively small steps.

The typical behaviour of an algorithm is that it repeatedly generates points $\underline{x}^k$ such that as k increases $\underline{x}^k$ moves close to $\underline{x}^*$. Features of a minimization algorithm is that $f(\underline{x}^k)$ is always reduced on each iteration, which imply that the stationary point turns out to be a local minimizer.

There are two fundamental strategies for moving from the current point $\underline{x}^k$ to a new iterate $\underline{x}^{k+1}$. These are (a) the line search and (b) the trust region strategy.

We will now describe a number of minimization methods that use line search algorithm.

3 Quasi-Newton algorithm

The main disadvantage of Newton's method, even when modified (the introduction of line search in Newton method and the calculation of search directions thereof) to ensure convergence, is that the calculation of the second derivative matrix G^k at each iteration. Therefore, a computational drawback of Newton method is the need to evaluate the inverse Hessian. This is avoided using quasi-Newton method. Quasi-Newton methods use an approximation in place of the true inverse.

Quasi-Newton methods, therefore, were introduced with the aim of defining efficient methods that do not require the evaluation of second derivatives. They are obtained by setting d^k as the solution of

$$B^k d^k = -\nabla f(x^k), \tag{3.1}$$

where B^k is the $n \times n$ symmetric and positive definite matrix, taken as an approximation to G^k, which is adjusted iteratively in such a way that the direction d^k tends to approximate the Newton direction. Formula (3.1) is known as the direct quasi-Newton formula; in turn the inverse quasi-Newton formula is

$$d^k = -H^k \nabla f(x^k), \tag{3.2}$$

where $H^k = B^{k^{-1}}$. The main idea at the basis of quasi-Newton methods is that of obtaining the curvature information not from the Hessian but only from the values of the function and its gradient.

Introduction of the quasi-Newton method largely increased the range of problems which could be solved. *This type of method is like Newton method with line search, except that $G^{k^{-1}}$ at each iteration is approximated by a symmetric positive definite matrix H^k, which is updated from iteration to iteration.* Thus the kth iteration has the basic structure.

(a) Set $\underline{d}^k = -H^k \underline{g}^k$

(b) Line search along $\underline{d}^k$ giving $\underline{x}^{k+1} = \underline{x}^k + \alpha^k \underline{d}^k$

(c) Update H^k giving H^{k+1}

The initial positive definite matrix is chosen as $H^0 = I$. Potential advantages of the method (as against Newton's method) are :

- Only first derivative required (Second derivative required in Newton method)
- H^k positive definite implies the descent property (G^k may be indefinite in Newton method)

Much of the interest lies in the updating formula which enables H^{k+1} to be calculated from H^k.

We know that for any quadratic function

$$q(\underline{x}) = \frac{1}{2}\underline{x}^T G \underline{x} + \underline{b}^T \underline{x} + c$$

where G, $\underline{b}$ and c are constant and G is symmetric, the Hessian maps differences in position into differences in gradient,.i.e.,

$$\underline{g}^{k+1} - \underline{g}^k = G(\underline{x}^{k+1} - \underline{x}^k). \tag{3.3}$$

The above property says that changes in gradient $\underline{g}$ ($=\nabla f(x)$) provide information about the second derivative of $q(\underline{x})$ along $(\underline{x}^{k+1} - \underline{x}^k)$. In the quasi-Newton methods at $\underline{x}^k$ we have the information about the direction $\underline{d}^k$, H^k and the gradient $\underline{g}^k$. We can use these information to perform line search to obtain $\underline{x}^{k+1}$ and $\underline{g}^{k+1}$. We now need to calculate H^{k+1} (the approximate inverse of G^{k+1}) using the above information. At this point we impose the condition (3.3) for the non-quadratic function f. In other words, we impose that changes in the gradient provide information about the second derivative of f along the search direction $\underline{d}^k$.. Hence, we have

$$G^{(k+1)^{-1}}(\underline{g}^{k+1} - \underline{g}^k) = (\underline{x}^{k+1} - \underline{x}^k) \tag{3.4}$$

Taking into account that, in the case of quadratic function, the so called quasi-Newton equation

$$\nabla^2 f(x^{k+1})\delta^k = \gamma^k \tag{3.5}$$

holds, the correction ΔB^k is chosen such that $(B^k+\Delta B^k)\delta^k = \gamma^k$ i.e., $B^{k+1^{-1}}\delta^k = \gamma^k$. Therefore, we would like have $H^{k+1} = H^k + \Delta H^k$ such that

$$H^{k+1}\gamma^k = \delta^k, \tag{3.6}$$

where $\delta^k = (\underline{x}^{k+1} - \underline{x}^k)$ and $\gamma^k = (\underline{g}^{k+1} - \underline{g}^k)$. This is known as the *quasi-Newton condition* and for the quasi-Newton algorithm the update H^{k+1} from H^k must satisfy (3.6).

Methods differ in the way they update the matrix H^k. Essentially they are classified according to a rank one and rank two updating formula.

Algorithm 2: BFGS inverse quasi-Newton method

Step 1. Choose $x^0 \in R^n$. Set $H^0 = I$ and $k = 0$.

Step 2. If $\nabla f(x^k) = 0$ stop.

Step 3. Set the direction $d^k = -H^k \nabla f(x^k)$;

Step 4. Find α^k by means of a LSA that satisfies the strong Armijo's condition (2.4) and Wolf condition (2.6).

Step 5. Set $x^{k+1} = x^k + \alpha^k d^k$, $H^{k+1} = H^k + \Delta H^k$ with ΔH^k is given by (3.7). Set $k = k + 1$ and go to Step 2.

The most used class of quasi-Newton method is the rank two Brayden class. The updating rule of the matrix H^k is given by the following correction terms

$$\Delta H^k = \frac{\delta^k \delta^{k^T}}{\delta^{k^T}\gamma^k} - \frac{H^k\gamma^k(H^k\gamma^k)^T}{\gamma^{k^T}H^k\gamma^k} + c\gamma^{k^T}H^k\gamma^k v^k v^{k^T}, \tag{3.7}$$

where

$$v^k = \frac{\delta^k}{\delta^{k^T}\gamma^k} - \frac{H^k\gamma^k}{\gamma^{k^T}H^k\gamma^k},$$

and c is a nonnegative scalar. For $c = 0$ we have the Davidon-Fletcher-Powell (DFP) formula whereas for $c = 1$ we have the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula. An important property of the methods in the Broyden class is that H^0 is positive definite and for each k it results $\delta^{k^T}\gamma^k > 0$, then the positive definiteness is maintained for all the matrices H^k. Algorithm 2 describe the BFGS algorithm.

4 Conjugate gradient methods

The conjugate gradient method is very popular due to its simplicity and low computational requirements. The methods was originally introduced for solving the minimization of a strictly convex quadratic function with the aim of accelerating the gradient method without requiring the overhead needed by the Newton's

method. Indeed it requires only first order derivatives and no matrix storage and operations are needed.

The basic idea is that the minimization on R^n of the strictly convex quadratic function

$$f(x) = \frac{1}{2}x^T Gx + b^T x + c \tag{4.1}$$

with G symmetric positive definite, can be split into n minimizations over R. This is done by means of n directions $d^0, \ldots, d^{n-1}$ *conjugate* with respect to the positive definite Hessian G, that is directions are such that $(d^j)^T G d^i = 0$ for $j \neq i$. The vectors $\underline{d}^i$, $i = 0, 1, 2, \cdots, n-1$ are linearly independent. Along each direction d^j an exact line search is performed in closed form. This corresponds to the *conjugate directions algorithm* (CDA) that finds the global minimizer of a strictly convex quadratic function in at most n iterations and that is reported in Algorithm 3. The value of α^k at Step 4 is the exact minimizer of the quadratic function $f(x^k + \alpha d^k)$. Note that CDA can be seen equivalently as an algorithm for $\nabla f(x) = 0$, that is for solving the linear system $Gx = -b$.

Algorithm 3: CDA for quadratic functions

Step 1.. Calculate G-conjugate directions $d^0, \ldots, d^{n-1}$,

Step 2. Choose $x^0 \in R^n$ and set $k = 0$.

Step 3. If $\nabla f(x^k) = 0$ stop.

Step 4. Set

$$\alpha^k = -\frac{\nabla f(x^k)^T d^k}{(d^k)^T G d^k}.$$

Step 5. Set

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

$k = k + 1$ and go to Step 2.

Remark 4.1: A conjugate direction method is one which generates conjugate directions when applied to a quadratic function with Hessian G. Therefore, an n-dimensional quadratic function can be minimized in at most n steps given n mutually conjugate directions with respect to the Hessian matrix. One of the remarkable properties of the conjugate direction method is its ability to generate, in a very economic fashion, a set of vectors $\underline{d}^i$, $i = 1, 2, \cdots, n$ with a property known as conjugacy. It is easy to show that these vectors are linearly independent.

Theorem 4.1: Conjugate direction theorem

For the quadratic function $f(\underline{x}) = \frac{1}{2}\underline{x}^T G\underline{x} + \underline{x}^T \underline{b} + c$ with $\underline{g} = G\underline{x} + \underline{b}$,

$$\underline{g}^{(k+1)T}\underline{d}^{(i)} = 0, \text{for all} \quad k,\ 0 \leq k \leq n-1 \text{ and } 0 \leq i \leq k, \tag{4.2}$$

where the $\underline{d}^{(i)}$ are the conjugate directions defined above. The fact that the current residual $g^{(k+1)}$ is orthogonal to to all previous search directions, as expressed in this theorem, is a property that will be used extensively in this section.

In the CDA, the conjugate directions are given, whereas in the *conjugate gradient algorithm* (CGA) the n conjugate directions are generated iteratively according to the rule

$$d^k = \begin{cases} -\nabla f(x^k) & k = 0 \\ -\nabla f(x^k) + \beta^{k-1} d^{k-1} & k \geq 1. \end{cases}$$

This may be seen as a modification to the steepest descent algorithm, made by forming a new search direction, by adding a proportion of the previous search direction to the new gradient.

4.1 Properties of conjugate gradient method for quadratic function

The conjugate gradient method is a conjugate direction method with a very special property : In generating its set of conjugate vectors, it can compute a new vector $\underline{d}^{k+1}$ by using only the previous vector $\underline{d}^k$, it does not need to know all the previous $\underline{d}^0, \underline{d}^1, \ldots\ldots, \underline{d}^{(k-1)}$ of the conjugate set; $\underline{d}^{(k+1)}$ is automatically conjugate to these vectors. This remarkable property implies that the method requires little storage and computation.

Now for the details of the conjugate gradient method. Each direction $\underline{d}^{(k+1)}$ is chosen to be a linear combination of the steepest descent direction $g^{(k+1)}$ and the previous direction $\underline{d}^k$, we write

$$\underline{d}^{(k+1)} = -g^{(k+1)} + \beta^k \underline{d}^k, \tag{4.3}$$

where the scalar β^k must be determined by the requirement that $\underline{d}^{(k+1)}$ and $\underline{d}^k$ must be conjugate with respect to the matrix G. By premultiplying (4.3) by $\underline{d}^{(k)T} G$ and imposing the condition

$$\underline{d}^{(k)T} G \underline{d}^{(k+1)} = 0,$$

we find that

$$\beta^k = \frac{g^{(k+1)T} G \underline{d}^k}{\underline{d}^{(k)T} G \underline{d}^k}. \tag{4.4}$$

As in the general conjugate gradient, we perform successive one-dimensional minimizations along each of the search directions. We have thus specified a complete algorithm, which we express next.

4.2 Conjugate gradient algorithm for quadratic functions

In the conjugate direction algorithm the conjugate directions are set at the beginning by the matrix G. In the conjugate gradient algorithm these direc-

tions are calculated as we proceed. Again consider a quadratic function $f(\underline{x}) = \frac{1}{2}\underline{x}^T G\underline{x} + \underline{x}^T\underline{b} + c$.
Set $\underline{d}^0 = -\underline{g}^{(0)}$ and $\underline{x}^{(1)} = \underline{x}^{(0)} + \alpha^0\underline{d}^0$,
where α^0 is chosen to minimize $f(\underline{x}^{(0)} + \alpha\underline{d}^0)$, i.e. $\alpha^0 = -\frac{\underline{g}^{(0)T}\underline{d}^0}{\underline{d}^{(0)T}G\underline{d}^0}$ (Prove this!).

Now choose $\underline{d}^{k+1}$ as a linear combination of $\underline{g}^{(k+1)}$ and $\underline{d}^k$,

$$\underline{d}^{k+1} = -\underline{g}^{(k+1)} + \beta^k\underline{d}^k, \text{ with } \beta^k = \frac{\underline{g}^{(k+1)T}G\underline{d}^k}{\underline{d}^{(k)T}G\underline{d}^k}.$$

Algorithm 4: Conjugate gradient for quadratic function

Step 1 Set $k = 0$ and choose $\underline{x}^0$.

Step 2 Calculate $\underline{g}^{(0)} = \nabla f(\underline{x}^{(0)})$. If $\underline{g}^{(0)} = 0$ stop else $\underline{d}^0 = -\underline{g}^{(0)}$.

Step 3 Set $\alpha^k = -\frac{\underline{g}^{(k)T}\underline{d}^k}{\underline{d}^{(k)T}G\underline{d}^k}$.

Step 4 Calculate $\underline{x}^{(k+1)} = \underline{x}^{(k)} + \alpha^k\underline{d}^k$.

Step 5 Calculate $\underline{g}^{(k+1)} = \nabla f(\underline{x}^{(k+1)})$. If $\underline{g}^{(k+1)} = 0$ stop.

Step 6 Set $\beta^k = \frac{\underline{g}^{(k+1)T}G\underline{d}^k}{\underline{d}^{(k)T}G\underline{d}^k}$

Step 7 Calculate $\underline{d}^{k+1} = -\underline{g}^{(k+1)} + \beta^k\underline{d}^k$.

Step 8 Set $k = k + 1$ go to 3.

One can see the only difference between Algorithm 3 and Algorithm 4 is that in former the directions are pre-calculated while in latter the directions are calculated sequentially

4.3 Conjugate gradient methods for non-quadratic function

In the previous sections, we have described the conjugate gradient algorithm for quadratic function. It is natural to ask whether we can adapt to minimize the general nonlinear functions f. Indeed, conjugate gradient methods can be used to estimate a minimizer $\underline{x}^*$ of a non-quadratic objective function f without having to compute the second partial derivatives of f. The CGA can be extended to general nonlinear functions by interpreting the quadratic function (4.1) as a second order Taylor series approximation of the non-quadratic objection function. Near the minimum such functions behave approximately as quadratics. For a quadratic the Hessian G is constant. However, for a general nonlinear function the Hessian

is a matrix that has to be re-evaluated at each iteration of the algorithm. This can be computationally very expensive. Thus, an efficient implementation of the CGA that eliminates the Hessian evaluation at each step is desirable. We observe that G appears only in the calculation of α^k and β^k. Because

$$\alpha^k = \arg \min_{\alpha \geq 0} f(x^k + \alpha d^k), \tag{4.5}$$

the close form formula for α^k in Algorithm 3 can be replaced by a numerical line search procedure. Since we are dealing with a non-quadratic function f, and α^k is determined from a one-dimensional optimization that minimizes $f(\underline{x}^{(k)} + \alpha \underline{d}^k)$ we can do this numerically using line search. It is now appropriate to consider how the line search method with $\underline{d}^k$ can be applied to the minimization of non-quadratic functions. An immediate question is what sort of line search to use, and how accurate it should be. In this regards the inexact line search with Armijo's (2.4) and Wolf's (2.6) conditions are recommended.

Next we only need to concern ourself with the formula (4.4) of β^k. Fortunately, eliminating G from the formula is possible and results in algorithms that depend only on function and gradient values at each iteration. To calculate β^k we use the following arguments. Here we interpret $f(\underline{x}) = \frac{1}{2}\underline{x}^T G \underline{x} + \underline{b}^T \underline{x} + c$ as the second order Taylor Series approximation of the non-quadratic objective function with G the approximation of the Hessian and β^k determined from G. With $\beta^k = \frac{\underline{g}^{(k+1)T} G \underline{d}^k}{\underline{d}^{(k)T} G \underline{d}^k}$ and $\alpha^k G \underline{d}^k = g^{k+1} - g^k$ for the quadratic function we replace the calculation for β^k by

$$\beta^k = \frac{\underline{g}^{(k+1)T}(\underline{g}^{(k+1)} - \underline{g}^{(k)})}{\underline{d}^{(k)T}(\underline{g}^{(k+1)} - \underline{g}^{(k)})} \tag{4.6}$$

4.4 The Polak Ribiere method

In this case the denominator in (4.6) formula is multiplied out.

$\underline{d}^{(k)T}\underline{g}^{(k+1)} - \underline{d}^{(k)T}\underline{g}^{(k)} = \underline{d}^{(k)T}\underline{g}^{(k)}$, since $\underline{d}^{(k)T}\underline{g}^{(k+1)} = 0$ from Theorem 4.1.

Also

$\underline{d}^{(k)T}\underline{g}^{(k)} = (-\underline{g}^{(k)T} + \beta^{k-1}\underline{d}^{(k-1)T})\underline{g}^{(k)} = -\underline{g}^{(k)T}\underline{g}^{(k)}$ also from Theorem 4.1.

Thus

$$\beta^k = \frac{\underline{g}^{(k+1)T}(\underline{g}^{(k+1)} - \underline{g}^{(k)})}{\underline{g}^{(k)T}\underline{g}^{(k)}} \tag{4.7}$$

4.5 The Fletcher Reeves method

Multiply out the numerator giving a term $\underline{g}^{(k+1)T}\underline{g}^{(k)}$.
Now $0 = \underline{g}^{(k+1)T}\underline{d}^k = \underline{g}^{(k+1)T}(-\underline{g}^{(k)} + \beta^{k-1}\underline{d}^{k-1})$ and $\underline{g}^{(k+1)T}\underline{d}^{k-1} = 0$ from Theorem 4.1. Thus $\underline{g}^{(k+1)T}\underline{g}^{(k)} = 0$ and

$$\beta^k = \frac{\underline{g}^{(k+1)T}\underline{g}^{(k+1)}}{\underline{g}^{(k)T}\underline{g}^{(k)}}. \tag{4.8}$$

All of these methods requires only a small modifications to the steepest descent algorithm.

The aim of the conjugate gradient method is to associate conjugacy properties with the steepest descent method in an attempt to achieve both efficiency and reliability.

The two formulas (4.7) and (4.8) are equivalent in the case of quadratic objective function. The formulas give us conjugate gradient algorithms that do not require explicit knowledge of the Hessian G. All we need are the objective function and gradient values at each iteration.

When f if not a quadratic function the two formulas (4.7) and (4.8) give different values for β^k and an inexact line search is performed to determine the stepsize α^k. Usually a LSA that enforces the strong Wolfe's conditions (2.4) and (2.6) is used. A scheme of CGA is reported

Algorithm 5: CGA for nonlinear functions

Step 1. Choose $x^0 \in R^n$ and set $k = 0$.

Step 2. If $\nabla f(x^k) = 0$ stop.

Step 3. Compute β^{k-1} by either (4.7) or (4.8) and set the direction

$$d^k = \begin{cases} -\nabla f(x^k) & k = 0 \\ -\nabla f(x^k) + \beta^{k-1}d^{k-1} & k \geq 1; \end{cases}$$

Step 4. Find α^k by means of a LSA that satisfies the strong Wolfe's conditions (2.4) and (2.6).

Step 5. Set

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

$k = k + 1$ and go to Step 2.

We just present the Fletcher Reeves algorithm the other are very similar.

Algorithm 6: The Fletcher Reeves algorithm for non-quadratic function

Step 1. Select and arbitrary starting point $\underline{x}^0$. Set $k = 0$ and compute $\underline{g}^k = g(\underline{x}^k)$.

Step 2. Compute $\underline{d}^0$ as $\underline{d}^0 = -\underline{g}^0$

Step 3 Compute $\alpha = \alpha^k$ such that $f(\underline{x}^k + \alpha^k \underline{d}^k) < f(\underline{x}^k)$, set $\underline{x}^{k+1} = \underline{x}^k + \alpha^k \underline{d}^k$

Step 4. Compute $\underline{g}^{k+1}$ such that $\underline{g}^{k+1} = g(\underline{x}^{k+1})$; if $\|g^{k+1}\| \leq \epsilon$ (ϵ is a user supplied small number) then go to Step 7.

Step 5. Compute the new search direction

$$\underline{d}^{k+1} = -\underline{g}^{k+1} + \beta^k \underline{d}^k, \quad k > 1$$

$$\beta^k = ||\underline{g}^{k+1}||^2 / ||\underline{g}^k||^2$$

Step 6. Set $k = k + 1$ and go to Step 3

Step 7. Set $\underline{x}^* = \underline{x}^{k+1}$, stop.

4.6 Descent property in the case of exact line search

It is obvious that for any conjugate gradient algorithm when $\underline{d}^{0^T} \underline{g}^0 = -\underline{g}^{0^T} \underline{g}^0 < 0$, the descent property or condition holds on the first iteration, moreover when the line search is exact we have:

$$\underline{g}^{k+1^T} \underline{d}^k = 0 \quad \text{for} \quad k \geq 1. \tag{4.9}$$

Hence from (4.2) it follows:

$$\begin{aligned} \underline{g}^{k+1^T} \underline{d}^{k+1} &= \underline{g}^{k+1^T} (-\underline{g}^{k+1} + \beta^k \underline{d}^k) \\ &= -||\underline{g}^{k+1}||^2 < 0 \end{aligned} \tag{4.10}$$

This shows that a descent property holds on all iterations for any conjugate gradient formula and in particular for Fletcher-Reeves.

4.7 Re-starting of the conjugate gradient methods for non-quadratic function

For nonquadratic functions, the CGA will not usually converge in n steps, and as the algorithm progresses, the conjugacy of the directions will tend to deteriorate.

Thus a common practice is to re-initialize the direction vector to the negative gradient after every few iterations (e.g. n or $n+1$) and continue until the algorithm satisfies the stopping condition. This periodic *restart* along the steepest descent direction guarantee global convergence properties of CG methods. Restarting also essential to deal with the loss of conjugacy due to the presence of nonquadratic terms in the objective function that may cause the method to generate inefficient or even meaningless directions. Usually the restart procedure occurs every n steps or if a test on the loss of conjugacy such as

$$\left|\nabla f(x^k)^T \nabla f(x^{k-1})\right| > \delta \|\nabla f(x^{k-1})\|^2,$$

with $0 < \delta < 1$, is satisfied. Therefore, due to reset (or restart) a cycle type method is obtained. After each n iterations the Algorithm is started again with $\underline{x}^0 = \underline{x}^n$ and $\underline{d}^0 = -\underline{g}^n$. This strategy is called restarting because we reset the values of $\underline{x}^0$ and $\underline{g}^0$ before restarting the Algorithm again.

Remark 4.2 : In general the quasi-Newton methods are faster and efficient than the conjugate gradient methods. However, the conjugate gradient methods may be the only methods which are applicable to large problems, that is problems with hundreds or thousands of variables.

Remark 4.3 : Nonlinear conjugate gradient algorithms make up another popular class of algorithms for large-scale optimization. These algorithms can be derived as extensions of the conjugate gradient algorithm for quadratic functions. Given an iterate x^k and a direction d^k, a line search is performed along d^k to produce $x^{k+1} = x^k + \alpha^k d^k$. The β^k for the Fletcher-Reeves variant of the nonlinear conjugate method and for the Polak-Ribiere variant of conjugate gradient method is the same when f is quadratic, but not otherwise. Numerical testing suggests that the Polak-Ribiere method tends to be more efficient than the Fletcher-Reeves method.

5 Trust region algorithm for nonlinear optimization

The fundamental problem with Newton's method for unconstrained minimization of a function $f(\underline{x})$ is that it can fail to converge to a local minimum point (due to non-positive definite matrices in Newton method). We know that if the method is started close enough to a local minimum point it will converge, but there may be (and often are in practice) starting points from which the method fails to converge. The central challenge in practical implementations of Newton's method is 'globalization' (global convergence). That is, modifying the algorithm so that it will always converge to a local (not necessarily global) minimum point.

Historically, Newton's method was 'patched up' by adding a line search to the algorithm so that instead of moving from a point $\underline{x}$ to a point $\underline{x}+\underline{d}$, the algorithm picks an α between 0 and 1 so as to minimize $f(\underline{x}+\alpha * \underline{d})$. In practice, adding the line search makes Newton's method vastly more robust. In fact, under quite reasonable assumptions, and with an accurate enough line search, you can prove that Newton's method with line search will converge to a local minimum point. For example, the Armijo condition (2.4) and Wolfe condition (2.6) conditions on the accuracy of the line search will ensure convergence.

Trust region methods are an alternative approach to globalizing Newton's method (hence the method is known as trust region Newton method so long as it uses the Hessian matrix in the third term of the model (5.1) below). In this approach we construct an approximation of the objective function f near a point x^k by using the first three terms of a Taylor's series.

$$m_k(\underline{x}) = f(\underline{x}^k) + \nabla f(\underline{x}^k)^T(\underline{x} - \underline{x}^k) + (1/2)(\underline{x} - \underline{x}^k)^T\nabla^2 f(\underline{x}^k)(\underline{x} - \underline{x}^k) \qquad (5.1)$$

This approximation will only be valid for points near $\underline{x}^k$, so we set a bound Δ^k on how large we will let $x - x^k$ get. We then minimize $m_k(\underline{x})$ over this set:

$$\min \qquad m_k(\underline{x}) \text{ such that } \|\underline{x} - \underline{x}^k\| \leq \Delta^k. \qquad (5.2)$$

This subproblem is called the 'trust region subproblem'.

Notice that if Δ^k is large enough, then the solution to this minimization problem is exactly the same point that we would get by performing one step of Newton's method without a line search. Once we have solved the trust region subproblem, we move to the point, adjust Δ^k and repeat. With an appropriate strategy for adjusting the tolerance Δ^k, you can construct an algorithm with the same global convergence properties as Newton's method with line search. Here by the global convergence we mean the method will converge to the minimum point when starting from any point in the region of attraction of the minimizer.

The method discussed so far is known as the trust region Newton method since the matrix used in the model is Hessian matrix. In fact, this may not be the case. Note that (5.1) and (5.2) can be written as follows:

$$m_k(d) = f(\underline{x}^k) + \nabla f(\underline{x}^k)^T\underline{d} + (1/2)\underline{d}^T G^k \underline{d} \qquad (5.3)$$

$$\min \qquad m_k(d) \text{ such that } \|\underline{d}\| \leq \Delta^k, \qquad (5.4)$$

where $\Delta^k > 0$ is the trust region radius varies from iteration to iteration. Therefore, $\underline{d}^* = \underline{d}^k$ minimizes of (5.3) and (5.4) in the ball of radius Δ^k. If the matrix G^k at iteration k is Hessian at x^k the method will be trust region Newton method, if it is an approximation to Hessian such as the one used for the Quasi-Newton algorithm then method will be called the trust region Quasi-Newton method. Notice that $m_k(d)$ in (5.3) is a quadratic function and so is the constraint in (5.4) a quadratic constraint. Since we can write the constraint in (5.4)

as $\underline{d}^t\underline{d} \leq (\Delta^k)^2$. When G^k is positive definite and $\|G^{k^{-1}}\nabla f\| \leq \Delta^k$ then the minimizer $\underline{d}^k = -\nabla f/G^k$ of m_k. In this case we call $\underline{d}^k$ the full step. The solution to (5.3) and (5.4) is not so obvious in other cases. However, in any case we only need an approximate solution.

5.1 Outline of a trust region algorithm

Let us first rewrite the problem with a general matrix B^k, where B^k can be Hessian at x^k or the quasi-Newton matrix or the finite difference approximation of the Hessian etc. The model m_k can be written as

$$m_k(d) = f(\underline{x}^k) + \nabla f(\underline{x}^k)^T\underline{d} + (1/2)\underline{d}^T B^k\underline{d} \tag{5.5}$$

$$\min \qquad m_k(d) \text{ such that } \|\underline{d}\| \leq \Delta^k, \tag{5.6}$$

The trust-region approach can be motivated by noting that the quadratic model $m_k(\underline{x})$ is a useful model of the function $f(\underline{d})$ only near $\underline{x}^k$. When the Hessian matrix is indefinite, the quadratic function $m_k(\underline{d})$ is unbounded below, so is obviously a poor model of $f(\underline{x})$ when $\underline{d}$ is large. Therefore, it is reasonable to select the step by solving the subproblem (5.4). The trust-region parameter, Δ^k, is adjusted between iterations according to the agreement between predicted and actual reduction in the function as measured by the ratio

$$\rho^k = \frac{f(\underline{x}^k) - f(\underline{x}^{k+1})}{f(\underline{x}^k) - m_k(\underline{d})} = \frac{f(\underline{x}^k) - f(\underline{x}^k + \underline{d})}{m_k(0) - m_k(\underline{d})} \tag{5.7}$$

Here the numerator is called the actual reduction, and the denominator is the predicted reduction. If there is good agreement, that is, ρ^k is close to 1, then Δ^k is increased. If the agreement is poor; i.e., ρ^k is small or negative, then Δ^k is decreased. The decision to accept the step $\underline{d}^k$ is also based on ρ^k. Usually, $\underline{x}^{k+1} = \underline{x}^k + \underline{d}^k$ if $\rho^k > \sigma^0$, where σ^0 is small (typically 10^{-4}), otherwise $\underline{x}^{k+1} = \underline{x}^k$. The following algorithm describe the process.

Algorithm 7: Trust region algorithm

Step 1 Choose $\Delta > 0$, $\Delta_0 \in (0, \Delta)$, and $\eta \in [0, \frac{1}{4})$.

Step 2 Obtain d^k by approximating solving (5.6). Evaluate ρ^k from (5.7). If $f(x^k) = m_k(d^k)$ stop.

Step 3 If $\rho^k > \frac{1}{4}$ then $\Delta_k = 0.25\|d^k\|$
else if $\rho^k > \frac{3}{4}$ and $\|d^k\| = \Delta_k$ then $\Delta_{k+1} = \min(2\Delta_k, \Delta)$
else $\Delta_{k+1} = \Delta_k$

Step 4 if $\rho^k > \eta$ then $x^{k+1} = x^k + d^k$ go to Step 2 else $x^{k+1} = x^k$ go to Step 2.

Here Δ is an overall bound on the step lengths. Notice that the radius is increased only if $\|d^k\|$ actually reaches the boundary of the trust region. If the step stays strictly inside the region, we infer that the current value of Δ^k is not interfering with the progress of the algorithm, so we leave its value unchanged for the next iteration.

The Step 2 of the Algorithm 7 needs to solve (5.6) approximately. Here will describe a method, called the *dogleg method*, which is appropriate when B^k is positive definite. The solution obtained by this method is at least as much reduction in m_k as the reduction obtained by the so-called *Cauchy point*. This point is simply the minimizer of m_k along the steepest descent direction $-\nabla f(x^k)$, subject to the trust region bound.

5.2 The Cauchy point

It is enough for the global convergence purposes to find an approximate solution d^k of (5.6) that lies within the trust region and gives a *sufficient reduction* in the model. The sufficient reduction can be quantified in terms of the *Cauchy point*, which we denote by d_c^k. The *Cauchy point* calculation involves the following two steps:

- Find the vector d_s^k that solves a linear version of (5.6), that is,

$$d_s^k = \arg\min_{d \in R^n} f(x^k) + \nabla f(x^k)^T d, \qquad \|d\| \leq \Delta^k \tag{5.8}$$

- Calculate the scalar $\tau_k > 0$ that minimizes $m_k(\tau d_s^k)$ subject to satisfying the trust region bound, that is

$$\tau_k = \arg\min_{\tau > 0} m_k(\tau d_s^k) \qquad \|\tau d_s^k\| \leq \Delta^k \tag{5.9}$$

The *Cauchy point* is given by $d_c^k = \tau_k d_s^k$. The *Cauchy point* can be obtained in closed-form. The solution of (5.8) is simply

$$d_s^k = -\frac{\Delta^k}{\|\nabla f(x^k)\|} \nabla f(x^k).$$

To obtain τ_k explicitly, we consider the cases of $\nabla f(x^k)^T B^k \nabla f(x^k) \leq 0$ and $\nabla f(x^k)^T B^k \nabla f(x^k) > 0$ separately. For the former case, the function $m_k(\tau d_s^k)$decreases monotonically with τ whenever $\nabla f(x^k) \neq 0$, so τ_k is simply the largest value that satisfies the trust region bound, namely, $\tau_k = 1$. For the case $\nabla f(x^k)^T B^k \nabla f(x^k) > 0$, $m_k(\tau d_s^k)$ is a convex quadratic in τ, so τ_k is either the unconstrained minimizer

of this quadratic, $\|\nabla f(x^k)\|^3/(\Delta^k \nabla f(x^k)^T B^k \nabla f(x^k))$, or the boundary value 1, whichever comes first. In summary, we have

$$d_c^k = -\tau_k \frac{\Delta^k}{\|\nabla f(x^k)\|} \nabla f(x^k),$$

where

$$\tau_k = \begin{cases} 1 & \nabla f(x^k)^T B^k \nabla f(x^k) \leq 0, \\ \|\nabla f(x^k)\|^3/(\Delta^k \nabla f(x^k)^T B^k \nabla f(x^k)) & \text{otherwise.} \end{cases}$$

One of the advantages of the *Cauchy point* is that d_c^k is in-expensive to calculate. The *Cauchy point* is also important in that it is used in deciding if an approximate solution of the trust region subproblem is acceptable, i.e., if $(m_k(0) - m_k(d^k)) \geq c(m_k(0) - m_k(d_c^k))$, where $c \in (0, 1)$. Specifically, a trust region method will be globally convergent if its steps d^k attain sufficient decrease in m_k; that is, they give a reduction in the model m_k that is at least some fixed multiple of the decrease attained by the *Cauchy point*.

The main dis-advantage of the *Cauchy point* is that if we implement always the *Cauchy point* as our step, we are simply implementing the steepest descent method with a particular choice of step length. We know that steepest descent performs poorly in optimizing functions. Next we discuss a strategy that improve on the Cauchy point.

5.3 The dogleg method

When the matrix B^k is positive definite and the solution d^k remains within the bounds of the subproblem then the step taken is called the full step and it is denoted by $d_b^k = -B^{k^{-1}} \nabla f(x^k)$ where $\|d_b^k\| \leq \Delta^k$. When Δ^k is tiny, then the restriction $\|d^k\| \leq \Delta^k$ ensures that the quadratic term in m_k has little effect on the solution of (5.6). It is therefore sensible to minimize the linear function $f(x^k) + \nabla f(x^k)^T d^k$ over $\|d^k\| \leq \Delta^k$, that is, $d^k = -\Delta^k \frac{\nabla f(x^k)}{\|\nabla f(x^k)\|}$. For intermediate values of Δ^k, the dogleg method finds an approximate solution d_d^k using two line segments. The first line segment runs from the origin the the unconstrained minimizer along the steepest descent direction defined by

$$d_u^k = -\frac{\nabla f(x^k)^T \nabla f(x^k)}{\nabla f(x^k)^T B^k \nabla f(x^k)} \nabla f(x^k),$$

while the second line segment runs from the d_u^k to d_b^k. Hence

$$d^k(\tau) = \begin{cases} \tau d_u^k, & 0 \leq \tau \leq 1, \\ d_u^k + (\tau - 1)(d_b^k - d_u^k), & 1 \leq \tau \leq 2 \end{cases}$$

The the dogleg method minimize the model m_k along this path, subject to the trust region bound. Since m_k is decreasing function along the above path, the chosen d_d^k will be d_b^k when $\|d_b^k\| \leq \Delta^k$, otherwise at the point of intersection of the dogleg and the trust region boundary. In the latter case, the value of τ is calculated from

$$\|d_u^k + (\tau - 1)(d_b^k - d_u^k)\|^2 = (\Delta^k)^2$$

Despina Zoras (539077)

Theory of constrainted optimization-the optimality conditions

Professor M Ali

29 February, 2010

We consider to minimize functions subject to constraints on the variables. A general formulation for these problems is

$$\min_{x\in\Re^n} f(x) \quad \text{subject to} \begin{cases} c_i(x) = 0, i \in E, \\ c_i(x) \geq 0, i \in I; \end{cases} \tag{1}$$

where f and the functions c_i are all smooth, real-valued functions on a subset of $\Re^n$, and I and E are two finite sets of indices.

1 Examples

To introduce the basic principles behind the characterization of solutions of constrained optimization problems, we work through three simple examples. The ideas discussed here will be made rigorous in the sections that follow. We start by noting one item of terminology that recurs throughout this document: At a feasible point x, the inequality constraint $i \in I$ is said to be active if $c_i(x) = 0$ and inactive if the strict inequality $c_i(x) > 0$ is satisfied.

Example 1. *(A single equality constraint)*
Our first example is a two-variable problem with a single equality constraint:

$$\min\ x_1 + x_2 \qquad \text{s.t. } x_1^2 + x_2^2 - 2 = 0. \tag{2}$$

In the language of (1), we have $f(x) = x1+x_2$, $I = \emptyset$, $E = \{1\}$, and $c_1(x) = x_1^2+x_2^2-2$. We can see by inspection that the feasible set for this problem is the circle of radius $\sqrt{2}$ centered at the origin—just the boundary of this circle, not its interior. The solution x^* is obviously $(-1,-1)^T$. From any other point on the circle, it is easy to find a way to move that stays feasible (that is, remains on the circle) while decreasing f. For instance, from the point $x = (\sqrt{2}, 0)^T$ any move in the clockwise direction around the circle has the desired effect. We also see that at the solution x^*, the constraint normal $\nabla c_1(x^*)$ is parallel to $\nabla f(x^*)$. That is, there is a scalar λ_1^* such that

$$\nabla f(x^*) = \lambda_1^* \nabla c_1(x^*). \tag{3}$$

(In this particular case, we have $\lambda_1^* = -\frac{1}{2}$.) ■

We can derive (3) by examining first-order Taylor series approximations to the objective and constraint functions. To retain feasibility with respect to the function $c_1(x) = 0$, we require that $c_1(x + d) = 0$; that is,

$$0 = c_1(x + d) \approx c_1(x) + \nabla c_1(x)^T d = \nabla c_1(x)^T d. \tag{4}$$

Hence, the direction d retains feasibility with respect to c_1, to first order, when it satisfies

$$\nabla c_1(x)^T d = 0. \tag{5}$$

Similarly, a direction of improvement must produce a decrease in f, so that

$$0 > f(x+d) - f(x) \approx \nabla f(x)^T d.$$

or, to first order,

$$0 > \nabla f(x)^T d. \tag{6}$$

If there exists a direction d that satisfies both (5) and (6), we conclude that improvement on our current point x is possible. It follows that a necessary condition for optimality for the problem (2) is that there exist no direction d satisfying both (5) and (6). By drawing a picture, we can check that the only way that such a direction cannot exist is if $\nabla f(x)$ and $\nabla c_1(x)$ are parallel, that is, if the condition $\nabla f(x) = \lambda_1 \nabla c_1(x)$ holds at x, for some scalar λ_1. If this condition is not satisfied, the direction defined by

$$d = -\left(I - \frac{\nabla c_1(x) \nabla c_1(x)^T}{\|\nabla c_1(x)\|^2}\right) \nabla f(x) \tag{7}$$

satisfies both conditions (5) and (6).

By introducing the Lagrangian function

$$L(x, \lambda_1) = f(x) - \lambda_1 c_1(x). \tag{8}$$

and noting that $\nabla_x L(x, \lambda_1) = \nabla f(x) - \lambda_1 \nabla c_1(x)$, we can state the condition (3) equivalently as follows: At the solution x^*, there is a scalar λ_1 such that

$$\nabla_x L(x^*, \lambda_1^*) = 0. \tag{9}$$

This observation suggests that we can search for solutions of the equality-constrained problem (2) by searching for stationary points of the Lagrangian function. The scalar quantity λ_1 in (8) is called a Lagrange multiplier for the constraint $c_1(x) = 0$.

Though the condition (3) (equivalently, (9)) appears to be necessary for an optimal solution of the problem (2), it is clearly not sufficient. For instance, in Example 1, (3) is satisfied at the point $x = (1, 1)$ (with $\lambda_1 = \frac{1}{2}$), but this point is obviously not a solution– in fact, it maximizes the function f on the circle. Moreover, in the case of equality-constrained problems, we cannot turn the condition (3) into a sufficient condition simply by placing some restriction on the sign of λ_1. To see this, consider replacing the constraint $x_1^2 + x_2^2 - 2 = 0$ by its negative $2 - x_1^2 - x_2^2 = 0$ in Example 1. The solution of the problem is not affected, but the value of λ_1 that satisfies the condition (3) changes from $\lambda_1 = -\frac{1}{2}$ to $\lambda_1 = \frac{1}{2}$.

Example 2. *(A single inequality constraint)*
Our second example is a two-variable problem with a single inequality constraint: This is a slight modification of Example 1, in which the equality constraint is replaced by an inequality. Consider

$$\min \; x_1 + x_2 \qquad \text{s.t. } 2 - x_1^2 - x_2^2 \geq 0. \tag{10}$$

for which the feasible region consists of the circle of problem (2) and its interior.

Note that the constraint normal ∇c_1 points toward the interior of the feasible region at each point on the boundary of the circle. By inspection,we see that the solution is still $(-1,-1)$ and that the condition (3) holds for the value $\lambda_1^* = \frac{1}{2}$. However, this inequality constrained problem differs from the equality-constrained problem (2) of Example 1 in that the sign of the Lagrange multiplier plays a significant role, as we now argue.

As before, we conjecture that a given feasible point x is not optimal if we can find a step d that both retains feasibility and decreases the objective function f to first order. The main difference between problems (2) and (10) comes in the handling of the feasibility condition. As in (6), the direction d improves the objective function, to first order, if $\nabla f(x)^T d < 0$. Meanwhile, the direction d retains feasibility if

$$0 \leq c_1(x+d) \approx c_1(x) + \nabla c_1(x)^T d,$$

so, to first order, feasibility is retained if

$$c_1(x) + \nabla c_1(x)^T d \geq 0. \tag{11}$$

In determining whether a direction d exists that satisfies both (6) and (11), we consider the following two cases.

Case I: Consider first the case in which x lies strictly inside the circle, so that the strict inequality $c_1(x) > 0$ holds. In this case, any vector d satisfies the condition (11), provided only that its length is sufficiently small. In particular, whenever $\nabla f(x^*) \neq 0$, we can obtain a direction d that satisfies both (6) and (11) by setting

$$d = -c_1(x)\frac{\nabla f(x)}{\|\nabla f(x)\|}.$$

The only situation in which such a direction fails to exist is when

$$\nabla f(x) = 0. \tag{12}$$

CaseII: Consider now the case in which x lies on the boundary of the circle, so that $c_1(x) = 0$. The conditions (6) and (11) therefore become

$$\nabla f(x)^T d < 0, \nabla c_1(x)^T d \geq 0.$$

The first of these conditions defines an open half-space, while the second defines a closed half-space. We can easily see by drawing a picture that the two regions fail to intersect only when $\nabla f(x)$ and $\nabla c_1(x)$ point in the same direction, that is, when

$$\nabla f(x) = \lambda_1 \nabla c_1(x), \quad \text{for some } \lambda_1 \geq 0. \tag{13}$$

Note that the sign of the multiplier is significant here. If (3) were satisfied with a negative value of λ_1, then $\nabla f(x)$ and $\nabla c_1(x)$ would point in opposite directions, and we see that the set of directions that satisfy both (6) and (11) would make up an entire open half-plane.

The optimality conditions for both cases I and II can again be summarized neatly with reference to the Lagrangian function. When no first-order feasible descent direction exists at some point x^*, we have that

$$\nabla_x L(x^*, \lambda_1^*) = 0, \quad \text{for some } \lambda_1^* \geq 0, \tag{14}$$

where we also require that

$$\lambda_1^* c_1(x^*) = 0. \tag{15}$$

This condition is known as a complementarity condition; it implies that the Lagrange multiplier λ_1 can be strictly positive only when the corresponding constraint c_1 is active. Conditions of this type play a central role in constrained optimization, as we see in the sections that follow. In case I, we have that $c_1(x^*) > 0$, so (15) requires that $\lambda_1^* = 0$. Hence, (14) reduces to $\nabla f(x^*) = 0$, as required by (12). In case II, (15) allows λ_1^* to take on a nonnegative value, so (14) becomes equivalent to (13). ■

2 First-order optimality conditions

2.1 Statement of first-order necessary conditions

The three examples above suggest that a number of conditions are important in the characterization of solutions for (1). These include the relation $\nabla_x L(x, \lambda) = 0$, the nonnegativity of λ_i for all inequality constraints $c_i(x)$, and the complementarity condition $\lambda_i c_i(x) = 0$ that is required for all the inequality constraints.We now generalize the observations made in these examples and state the first-order optimality conditions in a rigorous fashion.

In general, the Lagrangian for the constrained optimization problem (1) is defined as

$$L(x, \lambda) = f(x) - \sum_{i \in E \cap I} \lambda_i c_i(x). \tag{16}$$

The active set $A(x)$ at any feasible x is the union of the set E with the indices of the active inequality constraints; that is,

$$A(x) = E \cup \{i \in I | c_i(x) = 0\}. \tag{17}$$

Next, we need to give more attention to the properties of the constraint gradients. The vector $\nabla c_i(x)$ is often called the normal to the constraint c_i at the point x, because it is usually a vector that is perpendicular to the contours of the constraint c_i at x, and in the case of an inequality constraint, it points toward the feasible side of this constraint. It is possible, however, that $\nabla c_i(x)$ vanishes due to the algebraic representation of c_i , so that the term $\lambda_i \nabla c_i(x)$ vanishes for all values of λ_i and does not play a role in the Lagrangian gradient $\nabla_x L$. For instance, if we replaced the constraint in (2) by the equivalent condition

$$c_1(x) = (x_1^2 + x_2^2 - 2)^2 = 0,$$

we would have that $\nabla c_1(x) = 0$ for all feasible points x, and in particular that the condition $\nabla f(x) = \lambda_1 \nabla c_1(x)$ no longer holds at the optimal point $(-1, -1)^T$. We usually make an assumption called a constraint qualification to ensure that such degenerate behavior does not occur at the value of x in question. One such constraint qualification (probably the one most often used in the design of algorithms) is the one defined as follows:

Definition 1. *(LICQ)*
Given the point x^* and the active set $A(x^*)$ defined by (17), we say that the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\nabla c_i(x^*), i \in A(x^*)$ is linearly independent. ■

Note that if this condition holds, none of the active constraint gradients can be zero.

This condition allows us to state the following optimality conditions for a general nonlinear programming problem (1). These conditions provide the foundation for many of the algorithms designed for (1). They are called first-order conditions because they concern themselves with properties of the gradients (first-derivative vectors) of the objective and constraint functions.

Theorem 1. *(First-Order Necessary Conditions under LICQ)*
Suppose that x^* is a local solution of (1) and that the LICQ holds at x^*. Then there is a Lagrange multiplier vector λ^* with components λ_i^*, $i \in E \cup I$, such that the following conditions are satisfied at (x^*, λ^*)

$$\nabla_x L(x^*, \lambda^*) = 0, \tag{18}$$
$$c_i(x^*) = 0, \quad \text{for all } i \in E, \tag{19}$$
$$c_i(x^*) \geq 0, \quad \text{for all } i \in I, \tag{20}$$
$$\lambda_i \geq 0, \quad \text{for all } i \in I, \tag{21}$$
$$\lambda_i c_i(x^*) = 0, \quad \text{for all } i \in E \cup I. \tag{22}$$

■

The conditions (18)-(22) are often known as the Karush-Kuhn-Tucker conditions, or KKT conditions for short. Because the complementarity condition implies that the Lagrange multipliers corresponding to inactive inequality constraints are zero, we can omit the terms for indices $i \notin A(x^*)$ from (18) and rewrite this condition as

$$0 = \nabla_x L(x^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in A(x^*)} \lambda_i \nabla c_i(x^*). \tag{23}$$

A special case of complementarity is important and deserves its own definition:

Definition 2. *(Strict Complementarity)*
Given a local solution x^* of (1) and a vector λ^* satisfying (18)-(22), we say that the strict complementarity condition holds if exactly one of λ_i and $c_i(x^*)$ is zero for each index $i \in I$. In other words, we have that $\lambda_i > 0$ for each $i \in I \cap A(x^*)$. ■

For a given problem (1) and solution point x^*, there may be many vectors λ^* for which the conditions (18)-(22) are satisfied. When the LICQ holds, however, the optimal λ^* is unique.

Another useful generalization of the LICQ is the Mangasarian-Fromovitz constraint qualification (MFCQ), which we define formally as follows.

Definition 3. *(MFCQ)*
Given the point x^* and the active set $A(x^*)$ defined by (17), we say that the Mangasarian-Fromovitz constraint qualification (MFCQ) holds if there exists a vector $w \in \Re^n$ such that

$$\begin{aligned} &\nabla c_i(x^*)^T w > 0, \quad \text{for all } i \in A(x^*) \cap I; \\ &\nabla c_i(x^*)^T w = 0, \quad \text{for all } i \in E, \end{aligned} \tag{24}$$

and the set of equality constraint gradients $\{\nabla C_i(x^*), i \in E\}$ is linearly independent. ■

Note the strict inequality involving the active inequality constraints. The MFCQ is a weaker condition than LICQ. If LICQ is satisfied, then the system of equalities defined by

$$\begin{aligned} \nabla c_i(x^*)^T w &= 1, \quad \text{for all } i \in A(x^*) \cap I, \\ \nabla c_i(x^*)^T w &= 0, \quad \text{for all } i \in E, \end{aligned} \tag{25}$$

has a solution w, by full rank of the active constraint gradients. Hence, we can choose the w of Definition 3 to be precisely this vector. On the other hand, it is easy to construct examples in which the MFCQ is satisfied but the LICQ is not.

It is possible to prove a version of the first-order necessary condition result, Theorem 1, in which MFCQ replaces LICQ in the assumptions.

Theorem 2. *(First-Order Necessary Conditions under MFCQ)*
Suppose that x^* is a local solution of (1) and that the MFCQ holds at x^*. Then there is a Lagrange multiplier vector λ^* with components λ_i^*, $i \in E \cup I$, such that the following conditions are satisfied at (x^*, λ^*)

$$\nabla_x L(x^*, \lambda^*) = 0, \tag{26}$$
$$c_i(x^*) = 0, \quad \text{for all } i \in E, \tag{27}$$
$$c_i(x^*) \geq 0, \quad \text{for all } i \in I, \tag{28}$$
$$\lambda_i \geq 0, \quad \text{for all } i \in I, \tag{29}$$
$$\lambda_i c_i(x^*) = 0, \quad \text{for all } i \in E \cup I. \tag{30}$$

■

Remark 1. *MFCQ has the particularly nice property that it is equivalent to boundedness of the set of Lagrange multiplier vectors λ^* for which the KKT conditions (18)–(22) are satisfied. (In the case of LICQ, this set consists of a unique vector λ^*, and so is trivially bounded.)* ■

3 Second-order conditions

So far, we have described the first-order conditions–the KKT conditions–which tell us how the first derivatives of f and the active constraints c_i are related at x^*. When these conditions are satisfied, a move along any vector w from F_1 (see Definition 4) either increases the first-order approximation to the objective function (that is, $w^T \nabla f(x^*) > 0$), or else keeps this value the same (that is, $w^T \nabla f(x^*) = 0$).

What implications does optimality have for the second derivatives of f and the constraints c_i? We see in this section that these derivatives play a 'tiebreaking' role. For the directions $w \in F_1$ for which $w^T \nabla f(x^*) = 0$, we cannot determine from first derivative information alone whether a move along this direction will increase or decrease the objective function f. Second-order conditions examine the second derivative terms in the Taylor series expansions of f and c_i, to see whether this extra information resolves the issue of increase or decrease in f. Essentially, the second-order conditions concern the curvature of the Lagrangian function in the 'undecided' directions-the directions $w \in F_1$ for which $w^T \nabla f(x^*) = 0$.

Since we are discussing second derivatives, stronger smoothness assumptions are needed here than in the previous sections. For the purpose of this section, f and c_i, $i \in E \cup I$, are all assumed to be twice continuously differentiable.

Definition 4. *Given a point x^* and the active constraint set $A(x^*)$ defined by (17), the set F_1 is defined by*

$$F_1 = \left\{ \alpha d \;\middle|\; \alpha > 0, \; \begin{array}{ll} d^T \nabla c_i(x^*) = 0, & \text{for all } i \in E, \\ d^T \nabla c_i(x^*) \geq 0, & \text{for all } i \in A(x^*) \cap I \end{array} \right\}$$

■

Note that F_1 is a cone. In fact, when a constraint qualification is satisfied, F_1 is the tangent cone to the feasible set at x^*. Given F_1 from Definition 4 and some Lagrange multiplier vector λ^* satisfying the KKT conditions (18)-(22), we define a subset $F_2(\lambda^*)$ of F_1 by

$$F_2(\lambda^*) = \{w \in F_1 | \nabla c_i(x^*)^T w = 0, \text{ all } i \in A(x^*) \cap I \text{ with } \lambda_i > 0\}.$$

Equivalently,

$$w \in F_2(\lambda^*) \Longrightarrow \left\{ \begin{array}{ll} \nabla c_i(x^*)^T w = 0, & \text{for all } i \in E; \\ \nabla c_i(x^*)^T w = 0, & \text{for all } i \in A(x^*) \cap I \text{ with } \lambda_i > 0; \\ \nabla c_i(x^*)^T w \geq 0, & \text{for all } i \in A(x^*) \cap I \text{ with } \lambda_i = 0. \end{array} \right. \tag{31}$$

The subset $F_2(\lambda^*)$ contains the directions w that tend to 'adhere' to the active inequality constraints for which the Lagrange multiplier component λ_i^* is positive, as well as to the equality constraints. From the definition (31) and the fact that $\lambda_i^* = 0$ for all inactive components $i \in I \setminus A(x^*)$, it follows immediately that

$$w \in F_2(\lambda^*) \Longrightarrow \lambda_i^* \nabla c_i(x^*)^T w = 0 \text{ for all } i \in E \cup I. \tag{32}$$

Hence, from the first KKT condition (18) and the definition (16) of the Lagrangian function, we have that

$$w \in F_2(\lambda^*) \Longrightarrow w^T \nabla f(x^*) = \sum_{i \in E \cup I} \lambda_i^* \nabla c_i(x^*)^T w = 0. \tag{33}$$

Hence the set $F_2(\lambda^*)$ contains directions from F_1 for which it is not clear from first derivative information alone whether f will increase or decrease. The first theorem defines a necessary condition involving the second derivatives: If x^* is a local solution, then the curvature of the Lagrangian along directions in $F_2(\lambda^*)$ must be nonnegative.

Theorem 3. *(Second-Order Necessary Conditions under LICQ)*
Suppose that x^* is a local solution of (1) and that the LICQ condition is satisfied. Let λ^* be a Lagrange multiplier vector such that the KKT conditions (18)–(22) are satisfied, and let $F_2(\lambda^*)$ be defined as above. Then

$$w^T \nabla_{xx} L(x^*, \lambda^*) w \geq 0, \text{ for all } w \in F_2(\lambda^*). \tag{34}$$

■

Theorem 4. *(Second-Order Necessary Conditions under MFCQ)*
Suppose that x^* is a local solution of (1) and that the MFCQ condition is satisfied. Let λ^* be a Lagrange multiplier vector such that the KKT conditions (18)–(22) are satisfied, and let $F_2(\lambda^*)$ be defined as above. Then

$$w^T \nabla_{xx} L(x^*, \lambda^*) w \geq 0, \text{ for all } w \in F_2(\lambda^*). \tag{35}$$

■

Sufficient conditions are conditions on f and c_i, $i \in E \cup I$, that ensure that x^* is a local solution of the problem (1). (They take the opposite tack to necessary conditions, which assume that x^* is a local solution and deduce properties of f and c_i.) The second order sufficient condition stated in the next theorem looks very much like the necessary condition just discussed, but it differs in that the constraint qualification is not required, and the inequality in (35) is replaced by a strict inequality.

Theorem 5. *(Second-Order Sufficient Conditions)*
Suppose that for some feasible point $x^* \in \Re^n$ there is a Lagrange multiplier vector λ^* such that the KKT conditions (18)–(22) are satisfied. Suppose also that

$$w^T \nabla_{xx} L(x^*, \lambda^*) w > 0, \quad \text{for all } w \in F_2(\lambda^*), \quad w \neq 0. \tag{36}$$

Then x^* is a strict local solution for (1). ■

Example 3. *For an example in which the issues are more complex, consider the problem*

$$\min -0.1(x_1 - 4)^2 + x_2^2 \qquad \text{s. t. } x_1^2 + x_2^2 - 1 \geq 0, \tag{37}$$

in which we seek to minimize a non-convex function over the exterior of the unit circle. Obviously, the objective function is not bounded below on the feasible region, since we can take the feasible sequence

$$\begin{bmatrix} 10 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 20 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 30 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 40 \\ 0 \end{bmatrix}$$

and note that $f(x)$ approaches $-\infty$ along this sequence. Therefore, no global solution exists, but it may still be possible to identify a strict local solution on the boundary of the constraint. We search for such a solution by using the KKT conditions (18)–(22) and the second-order conditions of Theorem 5.

By defining the Lagrangian for (37) in the usual way, it is easy to verify that

$$\nabla_x L(x, \lambda) = \begin{bmatrix} -0.2(x_1 - 4) - 2\lambda x_1 \\ 2x_2 - 2\lambda x_2 \end{bmatrix} \tag{38}$$

$$\nabla_{xx} L(x, \lambda) = \begin{bmatrix} -0.2 - 2\lambda & 0 \\ 0 & 2 - 2\lambda \end{bmatrix} \tag{39}$$

The point $x^* = (1, 0)^T$ satisfies the KKT conditions with $\lambda_1 = 0.3$ and the active set $A(x^*) = \{1\}$. To check that the second-order sufficient conditions are satisfied at this point, we note that

$$\nabla c_1(x^*) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

so that the space F_2 defined in (31) is simply

$$F_2(\lambda^*) = \{w | w_1 = 0\} = \{(0, w2)^T | w_2 \in \Re\}. \tag{40}$$

Now, by substituting x^* and λ^*. into (39), we have for any $w \in F_2$ with $w \neq 0$ that

$$w^T \nabla_{xx} L(x^*, \lambda^*) w = \begin{bmatrix} 0 \\ w_2 \end{bmatrix}^T \begin{bmatrix} -0.4 & 0 \\ 0 & 1.4 \end{bmatrix} \begin{bmatrix} 0 \\ w_2 \end{bmatrix} = 14 w_2^2 > 0.$$

Hence, the second-order sufficient conditions are satisfied, and we conclude from Theorem 5 that $(1, 0)^T$ is a strict local solution for (37). ■

Despina Zoras (539077)

Algorithms for Nonlinear Constrained Optimization

We consider the methods for solving the nonlinear programing problem (1):

$$\min_{x\in\Re^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, i \in E, \\ h_j(x) \geq 0, j \in I, \end{cases} \tag{1}$$

where f and the functions c_i, h_i are all smooth, real-valued functions on a subset of $\Re^n$, and I and E are two finite sets of indices. We write

$$c = (c_1, c_2, \ldots, c_i \ldots, c_m)^T; \qquad h = (h_1, h_2, \ldots, h_j, \ldots, h_k)^T, m \leq n$$

The methods for solving (1) can be classified as: unconstrained sequential methods (USM), unconstrained exact penalty methods (UEPM) and sequential quadratic programming (SQP)

- USM: There are three types of methods in this class: (i) The quadratic penalty method (ii) The logarithmic barrier method (iii) The augmented Lagragian method

(i) The quadratic penalty method (QPM):

$$\min_{x\in\Re^n} f(x) + \frac{1}{\varepsilon}\{||\max\{-h(x), 0\}||^2 + ||c(x)||^2\}$$

$$\min_{x\in\Re^n} f(x) + \frac{1}{\varepsilon}\sum_{i\in E} c_i^2(x) + \frac{1}{\varepsilon}\sum_{j\in I}\left([h_j(x)]^-\right)^2,$$

where $[h_j(x)]^-$ denotes $\max(-h_j(x), 0)$.

(ii) The Logarithmic barrier method (LBM):
LBM is applied when only in-equality constraints are present, i.e.

$$\min_{x\in\Re^n} f(x) - \varepsilon\sum_{i=1}^{k}\ln(h_i(x))$$

(iii) The augmented Lagrangian method (ALM):
ALM is applied when only equality constraints are present. The lagrangian function is given by

$$L(x, \lambda) = f(x) - \lambda^T c(x),$$

and the augmented Lagrangian function is given by:

$$L_a(x, \lambda; \varepsilon) = L(x, \lambda) + \tfrac{1}{\varepsilon}||c(x)||^2$$

$$\min_{x\in\Re^n} f(x) - \lambda^T c(x) + \tfrac{1}{\varepsilon}||c(x)||^2$$

- UEPM: There are several methods in this class. The methods in this class minimizes the so called ℓ_1 exact penalty function:

$$\min_{x\in\Re^n} f(x) + \frac{1}{\varepsilon}\left\{\sum_{i=1}^{k}\max(-h_i(x), 0) + \sum_{j=1}^{m}|c_j(x)|\right\}$$

Finally, the sequential Quadratic Programmin g (SQP) methods are widely used methods and the description of SQP is given below.

Sequential Quadratic Programming Methods

The sequential quadratic programming (SQP) approach can be viewed as a generalization to constrained optimization of Newton's method for unconstrained optimization: in fact it finds a step away from the current point by minimizing a quadratic model of the problem. For simplicity, let us consider the equality constrained Problem (2). In quadratic programming problem a quadratic function is minimized subject to linear constraints. Suppose we have only equality constrained problem i.e.

$$\min f(x) \quad \text{subject to} \quad c(x) = 0 \tag{2}$$

A penalty function method is an indirect way of attempting to (2). A more direct and efficient approach is to iterate on the basis of certain approximation to $f(x)$ and linear approximation to $c(x)$ then at each iteration one need to solve a quadratic programming problem. Hence the name 'sequentiat quadratic programming'. Although, we consider the approximation of $f(x)$ and $c(x)$ separately, there is a connection between the quadratic programming problem, resulting from the approximation of $f(x)$ and $c(x)$, and the Newton iteration of the Lagrangian function

$$L(x, \lambda) = f(x) - \lambda^T c(x). \tag{3}$$

The KKT first order NOC of $L(x, \lambda)$ is

$$\nabla L(x, \lambda) = \begin{pmatrix} \nabla_x L \\ \nabla_\lambda L \end{pmatrix} = \begin{pmatrix} \nabla f(x) - \nabla c\lambda \\ c(x) \end{pmatrix} = 0, \tag{4}$$

where $\nabla c(x) = (\nabla c_1, \nabla c_2 \cdots \nabla c_m) = Jc^T$ and $Jc = \begin{pmatrix} \nabla c_1^T \\ \vdots \\ \nabla c_m^T \end{pmatrix}$ is the Jacobian matrix of the vector function $c(x)$. The Newton iteration for the solution of (4) is

$$\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix} + \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} \tag{5}$$

where the Newton step $d = \begin{pmatrix} d_x \\ d_\lambda \end{pmatrix} = \begin{pmatrix} x - k^k \\ \lambda - \lambda^k \end{pmatrix}$ solves

$$H(x, \lambda) = H\left(x^k, \lambda^k\right) + JHd = 0, \quad d = \begin{pmatrix} x - x^k \\ \lambda - \lambda^k \end{pmatrix}, \tag{6}$$

where $H(x, \lambda) = \nabla L(x, \lambda)$, $\quad \nabla L(x, \lambda) = \begin{pmatrix} \nabla_x L(x, \lambda) \\ \nabla_\lambda L(x, \lambda) \end{pmatrix} = 0$; JH is the Jacobian of H.
It is clear from (6) that

$$JH \begin{pmatrix} x - x^k \\ \lambda - \lambda^k \end{pmatrix} = -H\left(x^k, \lambda^k\right) \Rightarrow \nabla H^T d = -H\left(x^k, \lambda^k\right)$$

$$\Rightarrow J(\nabla L)\begin{pmatrix} x-x^k \\ \lambda-\lambda^k \end{pmatrix} = -\nabla L\left(x^k,\lambda^k\right)$$

$$J\begin{pmatrix} \nabla_x L\left(x^k,\lambda^k\right) \\ \nabla_\lambda L\left(x^k,\lambda^k\right) \end{pmatrix}\begin{pmatrix} x-x^k \\ \lambda-\lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x f(x^k) - \nabla_x c(x^k)\lambda^k \\ c(x^k) \end{pmatrix}$$

$$\begin{pmatrix} \nabla\left(\nabla_x L\left(x^k,\lambda^k\right)\right)^T \\ \nabla\left(\nabla_\lambda L\left(x^k,\lambda^k\right)\right)^T \end{pmatrix}\begin{pmatrix} x-x^k \\ \lambda-\lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x f\left(x^k\right) - \nabla_x c(x^k)\lambda^k \\ c(x^k) \end{pmatrix}$$

$$\begin{pmatrix} \nabla_x\left(\nabla_x L\right) & \nabla_\lambda\left(\nabla_x L\right)^T \\ \nabla_x\left(\nabla_\lambda L\right)^T & \nabla_\lambda\left(\nabla_\lambda L\right) \end{pmatrix}\begin{pmatrix} x- x^k \\ \lambda- \lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x f(x^k) - \nabla_x c(x^k)\lambda^k \\ c(x^k) \end{pmatrix}$$

$$\begin{pmatrix} \nabla^2_{xx} L & \nabla^2_{x\lambda} L^T \\ \nabla^2_{\lambda x} L^T & \nabla^2_{\lambda\lambda} L \end{pmatrix}\begin{pmatrix} x-x^k \\ \lambda-\lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x f(x^k) - \nabla_x c(x^k)\lambda^k \\ c(x^k) \end{pmatrix}$$

$$\begin{pmatrix} \nabla^2_{xx} L & -\nabla_x c(x^k) \\ \nabla_x c(x^k)^T & 0 \end{pmatrix}\begin{pmatrix} x-x^k \\ \lambda-\lambda^k \end{pmatrix} = -\nabla L\left(x^k,\lambda^k\right) \tag{7}$$

where

$$\nabla^2_{x\lambda} L = \nabla_\lambda\left(\nabla_x L\right) = \nabla_\lambda\left(\nabla_x f(x^k) - \nabla_x c\left(x^k\right)\lambda^k\right)$$

$$= -\nabla_x c(x^k)^T$$

$$\nabla^2_{\lambda x} L = \nabla_x\left(\nabla_\lambda L\right) = \nabla_x\left(c(x^k)\right) = \nabla_x c(x^k)$$

Now subtracting $\nabla_x c(x^k)\lambda^k$ both sides in the first equation of (7) we get

$$\begin{pmatrix} \nabla^2_{xx} L & -\nabla_x c(x^k) \\ \nabla_x c(x^k)^T & 0 \end{pmatrix}\begin{pmatrix} x-x^k \\ \lambda \end{pmatrix} = -\begin{pmatrix} \nabla_x f(x^k) \\ c(x^k) \end{pmatrix}$$

$$\begin{pmatrix} \nabla^2_{xx} L & -\nabla_x c(x^k) \\ \nabla_x c(x^k)^T & 0 \end{pmatrix}\begin{pmatrix} d^k_x \\ d^k_\lambda + \lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x f(x^k) \\ c(x^k) \end{pmatrix} \tag{8}$$

Now consider the quadratic programming problem

$$\left.\begin{array}{c} \min_d \frac{1}{2} d^T \nabla^2_{xx} L\left(x^k,\lambda^k\right) d + \nabla_x f(x^k)^T d \quad \text{such that} \\ Jc(x^k)d + c(x^k) = 0 \quad \text{where} \\ c(x^k+d) = c(x^k) + Jc(x^k)d = 0 \quad \text{is the first} \end{array}\right\}$$

order approximation of $c(x) = 0$ close to x^k. The Lagrangian of the above QP is given by:

$$F(d,\mu) = \frac{1}{2} d^T \nabla^2_{xx} L(x^k,\lambda^k) d + \nabla_x f(x^k)^T d - \mu^T\left[Jc(x^k)d + c(x^k)\right]$$

$$\left\{ \begin{array}{lll} \nabla_d F(d,\mu) = 0 & \Rightarrow & \nabla^2_{xx} L\left(x^k,\, \lambda^k\right) d + \nabla_x f(x^k) - Jc(x^k)^T \mu = 0 \\ \nabla_\mu F(d,\mu) = 0 & \Rightarrow & Jc(x^k) d + c(x^k) \qquad = 0 \end{array} \right\}$$

$$\Rightarrow \begin{pmatrix} \nabla^2_{xx} L\left(x^k, \lambda^k\right) & -\nabla_x c(x^k) \\ \nabla_x c(x^k)^T & 0 \end{pmatrix} \begin{pmatrix} d \\ \mu \end{pmatrix} = - \begin{pmatrix} \nabla_x f\left(x^k\right) \\ c(x^k) \end{pmatrix} \tag{9}$$

By comparing (9) and (8) we see that $\begin{pmatrix} d^k \\ \mu^k \end{pmatrix} = \begin{pmatrix} d_x^k \\ \lambda^{k+1} \end{pmatrix}$. Hence we solve (2) we use (9) instead of (8).