SECTION C: CONTINUOUS OPTIMISATION
LECTURE 1: INTRODUCTION

HONOUR SCHOOL OF MATHEMATICS, OXFORD UNIVERSITY
HILARY TERM 2006, DR RAPHAEL HAUSER
WITH A FEW ADDITIONS FROM DR. NICK GOULD

1. The Central Subject of this Course. The engineer who designs an aircraft with minimal drag given the required lift force, the manager who maximises profit within constraints imposed by the available resources, the bicycle courier who seeks the shortest path between two points in a city, and your cup of tea that cools down to maximise the entropy in the universe all solve optimisation problems! The world is full of them.

Mathematically, we can formulate an important class of such problems as follows:

$$\begin{aligned} \text{(P)} \qquad & \min_{x\in\mathbb{R}^n} f(x) \\ & \text{s.t.} \quad g_i(x) \geq 0, \quad (i = 1, \ldots, p), \\ & \qquad h_j(x) = 0, \quad (j = 1, \ldots, q), \end{aligned}$$

where f, g_i and h_j are sufficiently smooth functions: typically we require them to be twice continuously differentiable.

The function f represents an objective (such as energy, cost etc.) that has to be minimised under side constraints defined by the functions g_i and h_j. We therefore call f the *objective function*, the functions g_i the *equality constraint functions*, and the functions h_j the *inequality constraint* functions of (P). Note that by replacing f by $-f$ we can of course treat *maximisation problems* in the same framework.

EXAMPLE 1.1 (Linear Programming). *The* transshipment problem *occurs when the cheapest way of shipping prescribed amounts of a commodity across a transportation network has to be determined. This can be a network of oil pipe lines, a computer network, a network of shipping lanes, a road network etc..*

A network of gas pipelines is given in Figure 1.1 An arrow from node i to node j

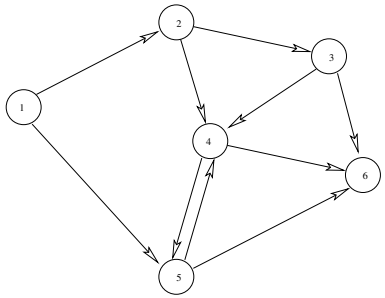

FIG. 1.1. *Gas pipeline network*

represents a pipe with transport capacity c_{ij} in the given direction. Transporting one unit of gas along the edge (ij) costs d_{ij}. The amount of gas produced at node i is p_i, and the amount of gas consumed is q_i. We assume that the total amount consumed equals the total amount of gas produced (if this assumption were not true, we could construct an equivalent transshipment problem that has this property). How do the quantities x_{ij} of gas shipped along the edges (ij) to be chosen so as to satisfy all the demands and to minimise costs?

We set $c_{ij} = 0$ (and d_{ij} arbitrary numbers) for all edges (ij) that do not exist. Doing so, we can assume that the network is a complete graph. The problem we have to solve is the following:

$$\begin{aligned} & \min_x \sum_{i,j=1}^{6} d_{ij} x_{ij} \\ & \text{s.t.} \quad \sum_{k=1}^{6} x_{ki} + p_i = \sum_{j=1}^{6} x_{ij} + q_i, \qquad (i = 1, \ldots, 6), \\ & \qquad 0 \leq x_{ij} \leq c_{ij}, \qquad (i, j = 1, \ldots, 6). \end{aligned}$$

This is an example of a *linear programming* problem, as the objective function and all the constraint functions are linear.

Note that it is not a priori clear that this problem has feasible solutions. One is therefore interested in algorithms that not only find optimal LP solutions when these exist but also detect when a problem instance is infeasible!

EXAMPLE 1.2 (Quadratic Programming). *In the portfolio optimisation problem, an investor considers a fixed time interval and wishes to decide which fraction of the capital he/she wants to invest in each of n different given assets when the expected return of asset i is μ_i and the covariance between assets i and j is σ_{ij}. The vector $\mu = [\mu_i]$ and the matrix $\sigma = [\sigma_{ij}]$ are assumed to be known and the investor aims at a total return of at least b. Subject to this constraint, he/she aims to minimise the risk as quantified by the variance of the overall portfolio.*

This problem can be modelled as

$$\begin{aligned} & \min_{x\in\mathbb{R}^n} \sum_{i=1}^{n}\sum_{j=1}^{n} \sigma_{ij} x_i x_j \\ & \text{s.t.} \quad \sum_{i=1}^{n} \mu_i x_i \geq b, \\ & \qquad \sum_{i=1}^{n} x_i = 1, \\ & \qquad x_i \geq 0 \quad (i = 1, \ldots, n). \end{aligned}$$

The constraint $\sum_{i=1}^{n} x_i = 1$ expresses the requirement that 100% of the initial capital has to be invested.

EXAMPLE 1.3 (Semidefinite Programming). *In optimal control, variables $y_1, \ldots, y_m$ have to be chosen so as to design a system that is driven by the linear ODE*

$$\dot{u} = M(y)u,$$

where $M(y) = \sum_{i=1}^m y_i A_i + A_0$ is an affine combination of given symmetric matrices A_i $(i = 0, \ldots, m)$. To stabilise the system, one would like to choose y so as to minimise the largest eigenvalue of $M(y)$.

Note that $\lambda_1(M) \leq \eta$ if and only if $\eta \mathrm{I} - M \succeq 0$ (is positive semidefinite). Therefore, the problem we need to solve is

$$\begin{aligned} \max_{\eta, y} \; & -\eta \\ \text{s.t.} \quad & \eta \mathrm{I} - A_0 - \sum_{i=1}^m y_i A_i \succeq 0. \end{aligned}$$

EXAMPLE 1.4. *An engineer designs a system determined by two design variables x and y which are dependent on each other via the relation $xy = 1$. The energy consumed by the system is given by $E(x, y) = x^2 + y^2 - 4$. Furthermore, the physical properties of materials used impose the constraints $x \in [0.5, 3]$. The engineer wishes to design a system that consumes the smallest amount of energy among all admissible systems.*

This problem can be formulated as

$$\begin{aligned} \text{(P)} \qquad \min_{x,y} \; & x^2 + y^2 - 4 \\ \text{s.t.} \quad & x - 0.5 \geq 0, \\ & -x + 3 \geq 0, \\ & x^{-1} - y = 0. \end{aligned}$$

The objective function is $f(x, y) = x^2 + y^2 - 4$, the inequality constraint functions are $g_1(x, y) = x - 0.5$ and $g_2(x, y) = -x + 3$, and the equality constraint function is $h(x, y) = x^{-1} - y$.

1.1. Learning Goals. The aim of this course is to teach you how to solve such problems numerically on a computer. But rather than programming efficient code (which is challenging in its own right), we concentrate on the theoretical properties of prototype algorithms. Moreover, in order to derive the mathematical building blocks of algorithms, we will have to derive mathematical conditions that characterise the points x^* at which (P) is minimised (the so-called *minimisers* or *argmins* of the problem).

For those of you who have never taken a course on numerical analysis, arguing about algorithms will at first have a strange new flavour: in principle, algorithms are mathematical functions that relate turn input values into an output. However, these functions cannot in general be written down in terms of a closed-form formula. Indeed, the computation path might be different for different input values (there might be conditional statements in the algorithm), and the computation may contain loops (iterations) the number of which may again depend on the input values and not be known a priori. Therefore, proving theorems about algorithms is much like proving theorems about mathematical formulas, but typically messier because many different scenarios might have to be analysed exhaustively.

Among the properties of algorithms that are most often analysed in theorems we can single out three important groups:

Correctness: Does the algorithm compute the claimed input-output relation for all input values? This is like proving that a mathematical equation or inequality holds true, where one side of the relation can be seen as the function of interest and the other as the computational rule or algorithm.

Complexity: How many computer operations will running the algorithm require as a function of the input data? Since this cannot usually be determined exactly for all input values, one is often interested either in the worst case that can occur or in an average case under some probability distribution over the input values. Moreover, when an algorithm can be run on input data of various dimensions, one quantifies the worst-case complexity as a function of the problem dimension or another measure of input size. Finally, when a numerical algorithm proceeds by iteratively improving approximations to the true (theoretical) solution of a problem, the complexity is usually analysed in terms of the number of computer operations per iteration and the convergence speed or *convergence rate* of the algorithm.

Reliability: Is there a guarantee of how accurately the final result of the algorithm approximates the true solution of the problem? Does this guarantee hold for a large set of input data, or are there domains in the input space for which the algorithm struggles to compute an accurate solution? How do rounding errors affect the computation?

2. Prerequisite Knowledge. Only linear algebra and multivariate calculus are required to understand this course. A course in numerical linear algebra or in numerical analysis helps in understanding some of the deeper issues but is not absolutely essential. Everything else will be developed from first principles. We will spend the remainder of this first lecture to discuss some important preliminary concepts. Other notions will be introduced if an when we need them.

2.1. Local versus Global Optimality. Let us first explain what we mean by "optimal solution" to the problem (P). A point $x \in \mathbb{R}^n$ is *feasible* for the optimisation problem (P) if $g_i(x) \geq 0 \; \forall i$ and $h_j(x) = 0 \; \forall j$, that is, if x satisfies all the constraints of the problem. The set $\mathcal{F}$ of feasible points is called the *domain of feasibility* of (P). A feasible point x^* is a *local minimiser* if there exists a ball $B_\epsilon(x^*)$ around x^* such that

$$f(x^*) \leq f(x) \qquad \forall x \in B_\epsilon(x^*) \cap \mathcal{F},$$

that is, x^* is a minimiser amongst all the feasible points in a neighbourhood of x^*, but there might be feasible points further away from x^* where the objective function takes an even smaller value. A feasible point x^* is a *global minimiser* if

$$f(x^*) \leq f(x) \qquad \forall x \in \mathcal{F},$$

that is, x^* minimises the objective function amongst all feasible points of the problem, although there might exist several of these points.

EXAMPLE 2.1. *The problem*

$$\begin{aligned} (P) \qquad \min_{x \in \mathbb{R}} \; & f(x) = x^3 + 9x^2 \\ \text{s.t.} \quad & -10 \leq x \leq 2 \end{aligned}$$

has a local minimiser at $x = 0$, and a global minimiser at $x^* = -10$, see Figure 2.1.

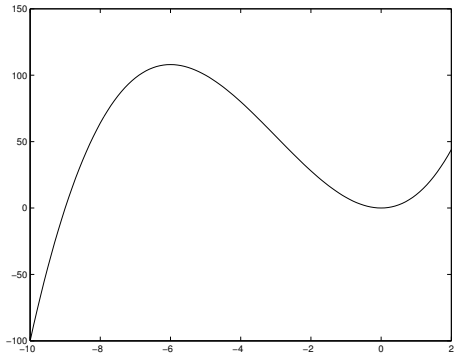

FIG. 2.1. *Objective function of Example 2.1*

In the general framework, efficient algorithms can only be devised for the problem of finding a *local* minimiser. The problem of finding a global minimiser is extremely important in practise, but its solution is typically based on heuristics that rely on local minimisation as a subproblem. We therefore restrict the material of this course to local minimisation.

A slightly confusing terminology is the following: an iterative algorithm for solving (P) converges *globally* if the output sequence $(x_k)_{\mathbb{N}}$ converges to a *local minimiser* x^* for all starting points $x_0 \in \mathcal{F}$. On the other hand, an iterative algorithm is called *locally convergent* if the output sequence $(x_k)_{\mathbb{N}}$ converges to a *local minimiser* x^* for all feasible starting points x_0 close enough to x^*, that is, for all $x_0 \in B_r(x^*) \cap \mathcal{F}$ for some $r > 0$.

EXAMPLE 2.2. *Let us go back to the problem of Example 2.1 and consider the following algorithm:*

S0 *Choose x_0. Set $\alpha = 1$, $k = 0$.*
S1 *$x = x_k + \alpha f'(x_k)$.*
S2 *If x is feasible then goto* **S3**, *else $\alpha \leftarrow \alpha/2$ and goto* **S1**.
S3 *Set $x_{k+1} = x$, $k \leftarrow k + 1$, $\alpha = 1$, and goto* **S1**.

This algorithm converges to the local minimiser $x^* = 0$ for all starting points $x_0 \in (-6, 2]$, and to the global minimiser $x^* = -10$ for $x_0 \in [-10, -6)$. For $x_0 = -6$ it remains stuck. If we exclude x_0 as a starting point, then this algorithm is globally convergent, even though it only converges to local minimisers! The focus here is that the algorithm converges no matter what the starting point is.

On the other hand, if we omit the judicious choice of α, we obtain the following algorithm:

S0 Choose x_0. Set $k = 0$.
S1 Set $x_{k+1} = x_k + f'(x_k)$, $k \leftarrow k + 1$, and goto **S1**.

This algorithm converges locally to the local minimiser $x^* = 0$, but for values $x_0 < -6$ the algorithm diverges to $-\infty$ and becomes infeasible instead of finding the local (and global) minimiser $x^* = -10$.

2.2. Convergence Rates. Since much of numerical analysis is devoted to the construction of algorithms that converge quickly, we should be able to quantify convergence speed.

A converging sequence $(x_k)_{\mathbb{N}} \to x^*$ has Q-convergence rate $r \geq 1$ (Q stands for "quotient") if $\exists \rho > 0$ and $k_0 \in \mathbb{N}$ such that

$$\|x_{k+1} - x^*\| \leq \rho \|x_k - x^*\|^r, \quad \forall k \geq k_0.$$

Note that when $r = 1$, only bounds with $\rho < 1$ are useful. If $r = 1$ or $r = 2$ we speak of Q-linear and Q-quadratic convergence respectively. Finally, $(x_k)_{\mathbb{N}}$ converges Q-superlinearly if

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0.$$

EXAMPLE 2.3. *Let $z \in (0, 1)$ and consider the sequence $(x_k)_{\mathbb{N}}$ defined by*

$$x_k := \sum_{n=0}^{k} z^n.$$

Then $(x_k)_{\mathbb{N}}$ converges Q-linearly but not Q-superlinearly to $x^* = (1 - z)^{-1}$. Indeed, we have $|x_k - x^*| = \sum_{n=k+1}^{\infty} z^n$. Therefore,

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|} = z < 1,$$

which shows the claim.

We say that an iterative algorithm has Q-convergence rate r if every output sequence produced by it converges at least at the Q-convergence rate r.

The practical significance of Q-linear convergence is that asymptotically the point x_{k+1} approximates x^* with $\log_{10} \rho$ more correct digits than x_k. This means that the number of correct digits grows linearly in the number of iterations taken. For Q-convergence of order r on the other hand, the number of additional correct digits asymptotically grows by a factor of r, that is, the number of correct digits is exponential in the number of iterations taken and the convergence is super fast.

In practice, Q-convergence of any order $r > 1$ is qualitatively similar to convergence of any other order because applying an order r algorithm j steps at a time yields an order r^j algorithm.

2.3. Convex Sets. The notion of convexity plays a central role in optimisation. A set $C \subseteq \mathbb{R}^n$ is *convex* if

$$x, y \in C \Rightarrow \lambda x + (1 - \lambda) y \in C \quad \forall \lambda \in [0, 1],$$

that is, if the straight line segment joining any two elements of C lies in C. The empty set, half spaces $\{x : a^{\mathrm{T}} x \geq 0\}$, polyhedra $\{x : Ax \geq b\}$, open balls $B_\rho(\bar{x}) =$

$\{x : \|x - \bar{x}\| < \rho\}$, ellipsoids $\{x : x^{\mathrm{T}}Bx \le r\}$ (with B a positive definite matrix) and affine subspaces $\{x : a^{\mathrm{T}}x = b\}$ are all examples of convex sets.

If $C, D \subseteq \mathbb{R}^n$ are convex sets, $\lambda \in \mathbb{R}$ and $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ is a linear map, then $C + D := \{x + y : x \in C, y \in D\}$, $\lambda C := \{\lambda x : x \in C\}$, $\varphi(C) := \{\varphi(x) : x \in C\}$ and $C \cap D$ are convex sets.

2.4. Convex Functions. Functions $f : \mathbb{R}^n \to (-\infty, +\infty]$ into the real line extended by $+\infty$ are called *proper*. A proper function is *convex* if its epigraph

$$\operatorname{epi}(f) := \big\{(x, z) \in \mathbb{R}^{n+1} : f(x) \le z\big\}$$

is a convex set in $\mathbb{R}^{n+1}$.

A proper function f (assumed to be defined on all of $\mathbb{R}^n$) is convex if and only if

$$f\big(\lambda x + (1-\lambda)y\big) \le \lambda f(x) + (1-\lambda) f(y) \tag{2.1}$$

for all $x, y \in \mathbb{R}^n$, $\lambda \in [0, 1]$. If this becomes a strict inequality $<$ for all $\lambda \in (0, 1)$ we say that f is *strictly convex*.

If f is convex then its *effective domain* $\operatorname{dom}(f) := \{x : f(x) < +\infty\}$ is a convex set in $\mathbb{R}^n$. On the other hand, we call any function $f : C \to \mathbb{R}$ which is defined on a convex set C and satisfies (2.1) for all $x, y \in C$ and $\lambda \in [0, 1]$ convex, and any such function can be extended to a convex proper function by setting $f(x) := +\infty$ for all $x \notin C$.

If f and g are convex proper functions then so are $f + g$ and λf for any $\lambda \ge 0$. If $\mathcal{F}$ is a set of convex proper functions then the pointwise supremum

$$\sup_{\mathcal{F}} : x \mapsto \sup\{f(x) : f \in \mathcal{F}\}$$

is a convex proper function. In particular, the pointwise maximum of finitely many convex proper functions is convex.

If f is a convex proper function then all its level sets $\{x : f(x) \le z\}$ (where $z \in (-\infty, +\infty]$ is fixed) are convex. Any convex proper function f is continuous on the topological interior $\operatorname{intr}\big(\operatorname{dom}(f)\big)$ of its effective domain.

A proper function $g : \mathbb{R}^n \to [-\infty, +\infty)$ or a function $g : C \to [-\infty, +\infty)$ defined on a convex set is called *concave* if $-g$ is convex.

THEOREM 2.4 (First order differential properties of convex functions).
Let $f : D \to \mathbb{R}$ be a function defined on a convex open domain $D \subset \mathbb{R}^n$.

(i) If f is convex then x^* is a local minimiser if and only if it is a global minimiser.

(ii) If f is C^1 on D, then f is convex if and only if for all $x, y \in D$,

$$f(y) \ge f(x) + \nabla f(x) \cdot (y - x), \tag{2.2}$$

that is, the graph of the first order approximation of f at x lies below the graph of f.

(iii) If f is convex and $\nabla f(x^*) = 0$ then x^* is a global minimiser of f. If $D = \mathbb{R}^n$ then this condition is both sufficient and necessary.

(iv) f is both convex and concave if and only if f is an affine function.

Proof. Suppose $x^* \in D$ is a local but not a global minimiser. Then there exists a $y \in D$ such that $f(y) < f(x^*)$, and then $f(\lambda y + (1-\lambda)x^*) \le \lambda f(y) + (1-\lambda) f(x^*) <$

$f(x^*)$ for all $\lambda \in [0, 1)$ and x^* cannot be a local minimiser because λ can be chosen arbitrarily close to 0. On the other hand, every global minimiser is a local minimiser. This proves (i). Suppose now that f satisfies (2.2). Given $\lambda \in [0, 1]$ and $x, y \in D$, let $z = (1-\lambda)x + \lambda y$. (2.2) implies

$$\begin{aligned} f(x) &\ge f(z) + \nabla f(z) \cdot (x - z) \quad \text{and} \\ f(y) &\ge f(z) + \nabla f(z) \cdot (y - z). \end{aligned}$$

Multiplying the first inequality by $(1-\lambda)$ and the second by λ, and adding the two inequalities we get $f(z) \le (1-\lambda) f(x) + \lambda f(y)$. Hence, f is convex. Suppose on the other hand that f is convex. Then $f\big(x + \lambda(y - x)\big) \le f(x) + \lambda\big(f(y) - f(x)\big)$, and hence

$$\frac{f\big(x + \lambda(y - x)\big) - f(x)}{\lambda} \le f(y) - f(x)$$

Taking limits as $\lambda \to 0$ we get (2.2). This proves (ii). (iii) is a trivial consequence of (i) and (ii). If f is affine, then it is clearly both convex and concave. On the other hand, if f is both convex and concave, and if f is differentiable at least at one point x^* then it follows from (2.2) that $f(y) \ge f(x^*) + \nabla f(x^*) \cdot (y - x^*)$ and $-f(y) \ge -f(x^*) - \nabla f(x^*) \cdot (y - x^*)$ for all y. Hence, $f(y) \equiv f(x^*) + \nabla f(x^*) \cdot (y - x^*)$. The general case can be proved in a similar way using the notion of subdifferential. One can also prove that there are always points where f is differentiable, but this is technically more difficult. □

THEOREM 2.5 (Second order differential properties of convex functions).
Let $f : D \to \mathbb{R}$ be a function defined on a convex open domain $D \subset \mathbb{R}^n$.

(i) If f is convex, $x \in D$ and the Hessian $H(x) = f''(x)$ exists, then $H(x) \succeq 0$ (positive semidefinite, that is, $z^{\mathrm{T}} H(x) z \ge 0$ for all $z \in \mathbb{R}$).

(ii) If $H(x)$ exists for all $x \in D$ and $H(x) \succeq 0$ then f is convex.

(iii) If $H(x)$ exists for all $x \in D$ and $H(x) \succ 0$ (positive definite, that is, $z^{\mathrm{T}} H(x) z > 0$ for all $z \in \mathbb{R} \setminus \{0\}$) then f is strictly convex.

Proof. See homework assignments. □