Appunti del corso di:

Computational Mathematics for Learning and Data Analysis: Ottimizzazione

incompleti
parti mancanti: mathematical background dalle funzioni in poi, ultimo capitolo sugli algoritmi di ottimizzazione vincolata, alcuni paragrafi nel mezzo (Accelerated Gradients, Deflected subgradient, ...).

Federica Di Pasquale
e-mail: federica.dipasquale@hotmail.com

Contents

Part I

Ottimizzazione

Chapter 1

Mathematical Background

1.1 Introduzione

L'obiettivo di questa parte del corso è risolvere dei problemi di ottimizzazione; questi possono essere scritti come:

$$(P) \qquad f_* = \min\{f(x) : x \in X\} \tag{1.1}$$

dove X è detta *regione ammissibile*, f *funzione obiettivo* ed f_* *valore ottimo*. Risolvere un problema di ottimizzazione significa dunque trovare una qualunque soluzione ottima $x_* \in X$ tale che $f(x_*) = f_*$. Tipicamente f rappresenta l'errore che vogliamo minimizzare ed assumeremo che sia una funzione a valori nei reali, cioè $f : X \to \mathbb{R}$; inoltre, dal momento che non possiamo raggiungere una precisione infinita, nella pratica cercheremo di risolvere questi problemi ammettendo una certa tolleranza ϵ, ovvero cercheremo un'approssimazione $\bar{x}$ della soluzione ottima tale per cui:

$$f(\bar{x}) - f_* \leq \epsilon \qquad \textit{oppure} \qquad \frac{f(\bar{x}) - f_*}{|f_*|} \leq \epsilon \tag{1.2}$$

dove la tolleranza ϵ può essere assoluta o relativa.

Notiamo inoltre che un problema di minimo può essere equivalentemente riscritto come un problema di massimo cambiando il segno della funzione obiettivo:

$$\min\{f(x) : x \in X\} \equiv \max\{-f(x) : x \in X\} \tag{1.3}$$

quindi non perdiamo di generalità se ci limitiamo a studiare un problema di minimizzazione.

La regione ammissibile X è determinata dai vincoli del problema ed in generale è solo una parte di un insieme più grande F, dunque se x è una soluzione candidata, si ha che questa è ammissibile solo se appartiene ad X, cioè:

$$X \subset F \qquad \begin{cases} x \in X & \text{soluzione ammissibile} \\ x \notin X & \text{soluzione non ammissibile} \end{cases} \tag{1.4}$$

Inoltre il valore ottimo deve essere tale per cui:

$$f_* \leq f(x), \forall x \in X, \forall v > f_* \, \exists x \in X : f(x) < v \tag{1.5}$$

Prima di cercare una soluzione ottima, tuttavia, è necessario accertarsi che questa esista, potrebbero infatti verificarsi i seguenti casi:

- $X = \emptyset$, quindi non esistono soluzioni ammissibili;
- f potrebbe non essere limitata inferiormente, cioè: $\forall M$, $\exists x_M : f(x_M) \leq M$.

Inoltre potrebbero esserci ulteriori problematiche a causa delle proprietà di f ed X, come avviene nei seguenti esempi:

$$\min\{x : x \in \mathbb{R} \wedge x > 0\} \qquad \text{"bad" } X \tag{1.6}$$

$$\min\{1/x : x \in \mathbb{R} \wedge x > 0\} \qquad \text{"bad" } f \text{ and } X \tag{1.7}$$

$$\min\{f(x) = \begin{cases} x & \text{if } x > 0 \\ 1 & \text{if } x = 0 \end{cases} : x \in [0,1]\} \qquad \text{"bad" } f \tag{1.8}$$

Dunque è necessario prima studiare gli oggetti matematici in gioco, ovvero la funzione da minimizzare e la regione ammissibile, e solo alla fine costruire un algoritmo per trovare x_* e studiarne l'efficienza e la convergenza.

Facciamo adesso un ripasso di alcune nozioni di base che serviranno per formalizzare i concetti usati negli algoritmi che risolvono questi problemi.

1.2 Insiemi e Successioni

Stiamo considerando funzioni del tipo $f : X \to \mathbb{R}$; dal momento che $\mathbb{R}$ è un insieme totalmente ordinato si ha:

$$\forall x, y \in X \qquad f(x) \leq f(y) \text{ oppure } f(x) \geq f(y) \tag{1.9}$$

Dato un sottoinsieme $S \subseteq \mathbb{R}$ sono definiti l'*inf* e il *sup* come:

$$\underline{s} = \inf S \iff \underline{s} \leq s \quad \forall s \in S \wedge \forall t > \underline{s}\, \exists s \in S \text{ s.t. } s \leq t$$
$$\bar{s} = \sup S \iff \bar{s} \geq s \quad \forall s \in S \wedge \forall t < \bar{s}\, \exists s \in S \text{ s.t. } s \geq t$$

inf e *sup* possono non esistere in $\mathbb{R}$, quindi si introducono i **reali estesi**:

$$\overline{\mathbb{R}} = \{-\infty\} \cup \mathbb{R} \cup \{+\infty\} \tag{1.10}$$

1.2.1 Successioni in $\mathbb{R}$

Gli algoritmi risolutivi che studieremo restituiscono ad ogni iterazione un punto $x_i \in X \subseteq \mathbb{R}^n$ ed il relativo valore della funzione obiettivo in quel punto $f(x_i)$.

E' necessario che la successione dei valori $f(x_i)$, che indichiamo con $v_1, v_2, \ldots$, converga al valore ottimo f_*. In generale non tutte le successioni convergono, ma quelle **monotone** sì, quindi è sufficiente costruire un algoritmo monotono.

Un modo ovvio per rendere una successione $\{v_i\}$ monotona, è costruire a partire da questa la successione:

$$v_i^* = \min\{v_h : h \leq i\} \tag{1.11}$$

ovvero il valore al passo i-esimo è il minimo fino a quel momento; in questo modo si ottiene una successione decrescente che asintoticamente fornisce una stima in eccesso del valore ottimo:

$$v_1^* \geq v_2^* \geq \cdots \geq v_\infty^* \geq f_* \text{ (stima asintotica)} \tag{1.12}$$

ma questo non è sufficiente per garantire che l'intera successione converga. Estraiamo invece le due successioni:

$$\underline{v}_i = \inf\{v_h : h \geq i\} \qquad \bar{v}_i = \sup\{v_h : h \geq i\} \tag{1.13}$$

in questo modo le due successioni così costruire sono rispettivamente crescenti e decrescenti, quindi ammettono entrambe un limite; se i due limiti sono uguali allora diciamo che la successione ha un limite:

$$\lim_{i\to\infty} v_i = v \iff \lim_{i\to\infty} \underline{v}_i = \lim_{i\to\infty} \bar{v}_i = v \tag{1.14}$$

Usiamo la seguente notazione:

$$\liminf_{i\to\infty} v_i = \lim_{i\to\infty} \underline{v}_i = \sup_i \underline{v}_i \tag{1.15}$$

$$\limsup_{i\to\infty} v_i = \lim_{i\to\infty} \bar{v}_i = \inf_i \bar{v}_i \tag{1.16}$$

ed in generale si ha:

$$\bar{v}_i \geq \underline{v}_i \implies \limsup_{i\to\infty} v_i \geq \liminf_{i\to\infty} v_i \tag{1.17}$$

inoltre:

$$\liminf_{i\to\infty} v_i = f_* \implies \{v_i\} \text{ successione minimizzante} \tag{1.18}$$

A questo punto però notiamo che non è sufficiente che la successione $\{f(x_i)\}$ prodotta dall'algoritmo converga al valore ottimo, ma è necessario che anche la successione dei punti x_i di $\mathbb{R}^n$ converga ad x_*, cioè vogliamo che:

$$\{f(x_i)\} \to f_* \implies \{x_i\} \to x_* \tag{1.19}$$

Dobbiamo estendere dunque questi concetti in $\mathbb{R}^n$.

1.2.2 Spazi vettoriali e topologia

Lo spazio euclideo $\mathbb{R}^n$ è definito come:

$$\mathbb{R}^n = \underbrace{\mathbb{R} \times \dots \mathbb{R}}_{n} = \{[x_1, \dots, x_n] : x_i \in \mathbb{R}, i = 1, \dots n\} \tag{1.20}$$

ed è chiuso rispetto alla somma e al prodotto per scalari, ma *non* è un insieme totalmente ordinato. Per estendere la definizione di limite di una successione in $\mathbb{R}^n$, è necessario prima definire *prodotti scalari*, *norme* e *distanze*.

Prodotto scalare

Siano x ed y due vettori di $\mathbb{R}^n$, il loro *prodotto scalare* è definito come:

$$\langle x, y\rangle := y^T x = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \dots + x_n y_n \tag{1.21}$$

e gode delle seguenti proprietà:

- $\langle x, y\rangle = \langle y, x\rangle$
- $\langle x, x\rangle \geq 0 \quad \forall x \in \mathbb{R}^n, \quad \langle x, x\rangle = 0 \iff x = 0$
- $\langle \alpha x, y\rangle = \alpha \langle x, y\rangle \qquad \forall x \in \mathbb{R}^n, \alpha \in \mathbb{R}$
- $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle \qquad \forall x, y, z \in \mathbb{R}^n$

Interpretazione geometrica: $\langle x, y\rangle = \|x\|\|y\| \cos\theta$

- $x \perp y \iff \langle x, y\rangle = 0$
- $\langle x, y\rangle > 0 \equiv$ "x e y puntano nella stessa direzione"

Norma euclidea

La norma indotta dal prodotto scalare $\langle \cdot, \cdot \rangle$ è definita come:

$$\|x\| := \sqrt{\langle x, x \rangle} = \sqrt{x_1^2 + \cdots + x_n^2} \tag{1.22}$$

Proprietà ≡ **definizione di una norma**:

- $\|x\| \geq 0 \qquad \forall x \in \mathbb{R}^n, \|x\| = 0 \iff x = 0.$
- $\|\alpha x\| = |\alpha| \|x\| \qquad \forall x \in \mathbb{R}^n, \alpha \in \mathbb{R}.$
- $\|x + y\| \leq \|x\| + \|y\| \qquad \forall x, y \in \mathbb{R}^n$ (*disuguaglianza triangolare*).

Valgono inoltre queste ulteriori proprietà:

- $|\langle x, y \rangle|^2 \leq \|x\| \|y\| \qquad \forall x, y \in \mathbb{R}^n$ (*disuguaglianza di Cauchy-Schwarz*).
- $\|x + y\|^2 = \|x\|^2 + \|y\|^2 + 2\langle x, y \rangle.$
- $2\|x\|^2 + 2\|y\|^2 = \|x + y\|^2 + \|x - y\|^2$ (*legge del parallelogramma*).

Useremo quasi sempre la norma euclidea, tuttavia esistono altre norme come ad esempio:

- $\|x\|_1 := \sum_{i=1}^n |x_i|.$
- $\|x\|_\infty := \max\{|x_i| : i = 1, \ldots, n\}.$
- $\|x\|_0 := |\{i : |x_i| > 0\}|.$

Molte (non tutte) derivano dalla **p-norm**:

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \tag{1.23}$$

che è *convessa* per $p \geq 1$. Vale la *disuguaglianza di Holder*:

$$\langle x, y \rangle^2 \leq \|x\|_p \|y\|_q \qquad \frac{1}{p} + \frac{1}{q} = 1 \tag{1.24}$$

Definizione di distanza (euclidea)

$$d(x, y) := \|x - y\| = \sqrt{(x_1 - x_2)^2 + \cdots + (x_n - y_n)^2} \tag{1.25}$$

Proprietà ≡ **definizione di distanza**:

- $d(x, y) \geq 0 \qquad \forall x, y \in \mathbb{R}^n, d(x, y) = 0 \iff x = y.$
- $d(\alpha x, 0) = |\alpha| d(x, 0) \qquad \forall x \in \mathbb{R}^n, \alpha \in \mathbb{R}.$
- $d(x, y) \leq d(x, z) + d(z, y) \qquad \forall x, y, z \in \mathbb{R}^n$ (*disuguaglianza triangolare*).

Definiamo inoltre la **palla** di centro $x \in \mathbb{R}^n$ e raggio $r > 0$:

$$\mathcal{B}(x, r) := \{y \in \mathbb{R}^n : \|y - x\| \leq r\} \tag{1.26}$$

cioè l'insieme di tutti i punti vicini ad x nella norma scelta.

La distanza/norma definisce una topologia ma, in uno spazio vettoriale di dimensione finita come $\mathbb{R}^n$, si ha che tutte le norme sono equivalenti, infatti siano $\|\cdot\|'$ e $\|\cdot\|$ due norme, allora per ogni x esistono due coefficienti α, β, con $0 < \alpha < \beta$, tali che:

$$\alpha \|x\|' \leq \|x\| \leq \beta \|x\|' \tag{1.27}$$

1.2.3 Successioni in $\mathbb{R}^n$

Possiamo adesso definire il limite di una successione $\{x_i\} \subset \mathbb{R}^n$ come:

$$\lim_{i\to\infty} x_i = x \equiv \{x_i\} \to x \tag{1.28}$$

$$\iff \forall \epsilon > 0 \quad \exists h \text{ s.t. } d(x_i, x) \leq \epsilon \quad \forall i \geq h \tag{1.29}$$

$$\iff \forall \epsilon > 0 \quad \exists h \text{ s.t. } x_i \in B(x, \epsilon) \quad \forall i \geq h \tag{1.30}$$

$$\iff \lim_{i\to\infty} d(x_i, x) = 0 \tag{1.31}$$

1.2.4 Insiemi

Insiemi aperti e chiusi

Insiemi compatti e Teorema di Bolzano-Weierstrass

1.3 Funzioni

Chapter 2

Ottimizzazione non vincolata e convessità

Per il momento vogliamo risolvere problemi di ottimizzazione per i quali la regione ammissibile è tutto R^n, ovvero problemi non vincolati.

$$(P) \qquad f_* = min\{f(x) : x \in \mathbb{R}^n\}$$

Dal momento che R^n è un insieme non limitato, potrebbe non esistere un minimo (non possiamo usare Weierstrass), ma anche se esistesse dobbiamo capire come trovarlo e come riconoscerlo.

In generale trovare il minimo globale di una funzione è molto complicato, a meno che la funzione non sia particolarmente semplice. Ci limitiamo dunque a cercare un **minimo locale**, ovvero un punto x_* che risolve:

$$min\{f(x) : x \in \mathcal{B}(x_*\epsilon)\} \qquad \text{per un qualche} \quad \epsilon > 0$$

2.1 Condizioni di Ottimalità

2.1.1 Primo ordine

Supponiamo di avere una funzione $f : \mathbb{R}^n \to \mathbb{R}$ che sia di classe C^1 in un intorno $\mathcal{B}(x, \delta)$ di un punto x. Una condizione necessaria ma *non sufficiente* affiché x sia un **minimo locale** di f, è che il gradiente della funzione in x sia nullo, ovvero:

Theorem 1.

$$x \ minimo \ locale \Rightarrow \nabla f(x) = 0$$

Proof. Assumiamo per assurdo che x sia un minimo locale ma $\nabla f(x) \neq 0$. Allora posso trovare un punto x' tale che $f(x') < f(x)$. Consideriamo l'insieme dei punti lungo la direzione dell'**anti-gradiente** passante per x:

$$x(\alpha) = x - \alpha \nabla f(x)$$

e sia $\phi(\alpha)$ la restrizione di f su questo insieme:

$$\phi(\alpha) = f(x(\alpha))$$

allora basta dimostrare che per ogni intorno di x $\exists \bar{\alpha} > 0 : \forall \alpha < \bar{\alpha} \quad \phi(\alpha) < f(x)$.

Dal momento che la funzione è **differenziabile**, possiamo scrivere lo sviluppo di Taylor di f al primo ordine come:

$$f(y) = \underbrace{f(x) + \langle \nabla f(x), y - x) \rangle}_{\text{first-order model}} + \underbrace{R(y-x)}_{reminder} \qquad dove \quad \lim_{y \to x} \frac{R(y-x)}{\|y-x\|} = 0$$

Nota: il resto R rappresenta l'errore che commettiamo approssimando f con il suo modello al primo ordine $L_x(y)$, e va a 0 più velocemente di $\|y - x\|$.

Sia adesso $y = x - \alpha \nabla f(x)$, riscriviamo allora $\phi(\alpha)$ come:

$$\begin{aligned} \phi(\alpha) &= f(x - \alpha \nabla f(x)) = f(x) + \langle \nabla f(x), -\alpha \nabla f(x) \rangle + R(-\alpha \nabla f(x)) \\ &= f(x) - \alpha \|\nabla f(x)\|^2 + R(-\alpha \nabla f(x)) \end{aligned}$$

a questo punto sappiamo che per $\alpha \to 0$:

$$\lim_{\alpha \to 0} \frac{R(-\alpha \nabla f(x))}{\|\alpha \nabla f(x)\|} = 0 \equiv \forall \epsilon > 0 \quad \exists \bar{\alpha} > 0 \quad t.c. \quad \frac{R(-\alpha \nabla f(x))}{\|\alpha \nabla f(x)\|} < \epsilon \quad \forall 0 \leq \alpha < \bar{\alpha}$$

Prendiamo allora $\epsilon < \|\nabla f(x)\|$, si ha dunque:

$$R(-\alpha \nabla f(x)) < \alpha \|\nabla f(x)\|^2 \implies \phi(\alpha) = f(x) - \alpha \|\nabla f(x)\|^2 + R(-\alpha \nabla f(x)) < f(x)$$

che vale $\forall \alpha < \bar{\alpha}$, dunque x non può essere un minimo locale. □

L'importanza della dimostrazione di questo teorema sta nel fatto che fornisce un'indicazione sulla procedura da seguire per trovare un minimo locale: dato un punto in cui il gradiente è diverso da 0 basta spostarsi lungo la direzione dell'antigradiente di uno step sufficientemente piccolo per trovare un punto *migliore*.

Infine notiamo che l'implicazione opposta non vale, infatti anche i massimi e i punti di sella hanno gradiente nullo. Dunque è necessario studiare le derivate di ordine successivo.

2.1.2 Secondo ordine

Abbiamo visto dunque che un minimo locale è un punto stazionario (gradiente nullo), ma:

$$punto\ stazionario \not\Rightarrow minimo\ locale$$

Prima di vedere qual è la condizione di ottimalità, notiamo la seguente: se x è un *punto stazionario* di una funzione $f \in C^2$, allora il gradiente del suo modello al secondo ordine in $\mathbf{x}$ è nullo, infatti:

$$Q_x(y) = L_x(y) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x)$$

è il modello al secondo ordine, il suo gradiente è:

$$\nabla Q_x(y) = \nabla f(x) + \nabla^2 f(x)(y-x)$$

che in $y = x$ diventa:

$$\nabla Q_x(x) = 0 + \nabla^2 f(x) \cdot 0 = 0$$

Vogliamo adesso trovare una condizione di ottimalità andando a studiare le derivate di ordine successivo. Se la funzione in questione è quadratica allora sappiamo già la risposta: $Q = \nabla^2 f(x) \succeq 0$. Dimostriamo adesso che questo vale anche per una funzione f generica (che sia almeno di classe C^2), ovvero:

Theorem 2.

$$f \in C^2 : x \text{ minimo locale} \Rightarrow \nabla^2 f(x) \succeq 0$$

Proof. Assumiamo per assurdo che x sia un *minimo locale* ma $\nabla^2 f(x) \not\succeq 0$.

Sviluppiamo f al secondo ordine:

$$f(y) = L_x(y) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x) + R(y-x) \qquad \text{dove} \quad \lim_{y \to x} \frac{R(y-x)}{\|y-x\|^2} = 0$$

Se $\nabla^2 f(x) \not\succeq 0$, allora esiste almeno un autovettore relativo ad un autovalore negativo. Sia d questo autovettore, allora:

$$\nabla^2 f(x) d = \lambda d \quad \text{con} \quad \lambda < 0$$

Dunque si ha:

$$d^T \nabla^2 f(x) d = \lambda \underbrace{\|d\|^2}_{=1} = \lambda < 0.$$

In pratica d è una direzione di *curvatura negativa*, dunque facciamo vedere che se ci spostiamo lungo questa direzione, arriviamo ad un punto x' per cui $f(x') < f(x)$.

Siano $x(\alpha)$ l'insieme dei punti lungo la direzione parallela a d passante per x, e $\phi(\alpha)$ la restrizione di f su questo insieme:

$$x(\alpha) = x + \alpha d, \qquad \phi(\alpha) = f(x(\alpha))$$

Sviluppiamo $\phi(\alpha)$ attorno ad x (che abbiamo assunto essere un minimo dunque $\nabla f(x) = 0$) al secondo ordine:

$$\phi(\alpha) = f(x) + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x) d + R(\alpha d)$$

A questo punto sappiamo che:

$$\lim_{\alpha \to 0} \frac{R(\alpha d)}{\alpha^2} = 0 \equiv \forall \epsilon > 0 \quad \exists \bar{\alpha} > 0 \text{ t.c. } R(\alpha d) \leq \epsilon \alpha^2, \quad \forall 0 \leq \alpha \leq \bar{\alpha}$$

Prendiamo allora:

$$\epsilon < -\frac{1}{2} d^T \nabla^2 f(x) d \implies R(\alpha d) < -\frac{1}{2}\alpha^2 d^T \nabla^2 f(x) d$$

si ha infine:

$$\implies \phi(\alpha) = f(x) + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x) d + R(\alpha d) < f(x), \quad \forall 0 \leq \alpha \leq \bar{\alpha}$$

□

Quindi x non è un minimo locale. Notiamo che anche in questo caso l'implicazione opposta non vale.

Theorem 3. *Una condizione necessaria e sufficiente affinché x sia un minimo locale è:*

$$f \in C^2 : \nabla f(x) = 0 \text{ e } \nabla^2 f(x) \succ 0 \implies x \text{ minimo locale } (\textit{stretto})$$

Proof. Sia x un punto in cui $\nabla f(x) = 0$ e $\nabla^2 f(x) \succ 0$. Sviluppiamo f al secondo ordine:

$$f(x+d) = f(x) + \frac{1}{2} d^T \nabla^2 f(x) d + R(d)$$

si ha che:

$$\lim_{\|d\| \to 0} \frac{R(d)}{\|d\|^2} = 0 \equiv \forall \epsilon > 0 \quad \exists \delta > 0 \text{ t.c. } R(d) \geq -\epsilon \|d\|^2 \quad \forall d \text{ t.c. } \|d\|^2 < \delta$$

Adesso indichiamo con $\lambda_{min} > 0$ il minimo autovalore di $\nabla^2 f(x)$, si ha dunque:

$$d^T \nabla^2 f(x) d \geq \lambda_{min} \|d\|^2$$

Prendiamo allora $\epsilon < \lambda_{min}/2$. Otteniamo che $\forall d$ t.c. $\|d\|^2 < \delta$:

$$f(x+d) = f(x) + \frac{1}{2} d^T \nabla^2 f(x) d + R(d) > f(x) + \underbrace{\frac{\lambda_{min}}{2} \|d\|^2 - \epsilon \|d\|^2}_{positive}$$

In altre parole se mi sposto a partire da x (localmente) lungo una qualunque direzione d, il valore della funzione cresce (strettamente), quindi x è un *minimo locale* stretto. □

Nella pratica raramente andremo a studiare derivate di ordine successivo al primo e, soprattutto, non ci preoccuperemo di trovare il minimo globale. Tuttavia ci sono dei casi semplici in cui le proprietà della funzione ci garantiscono che, se riusciamo a trovare un minimo, allora questo è un minimo globale. L'intuizione è: se ho più di un minimo, allora tra due minimi deve esistere necessariamente un massimo, dunque se la funzione è tale per cui non esistono massimi, allora un minimo è anche globale. Una funzione non ammette massimi se la sua matrice Hessiana non è mai semidefinita negativa, dunque una condizione *sufficiente* è:

$$\nabla^2 f(x) \succeq 0, \quad \forall x \in \mathbb{R}^n$$

una funzione f che gode di questa proprietà è una *funzione convessa*.

2.2 Convessità

2.2.1 Insiemi convessi

Siano x e y due punti di $\mathbb{R}^n$. Indichiamo con $conv(x, y)$ l'insieme (detto *inviluppo convesso o convex hull*):

$$conv(x, y) = \{z = \alpha x + (1 - \alpha) y : \alpha \in [0, 1]\}$$

ovvero tutti i punti che appartengono al segmento che congiunge x e y.

Un insieme $C \subset \mathbb{R}^n$ è detto **convesso** se:

$$\forall x, y \in C \quad conv(x, y) \subseteq C$$

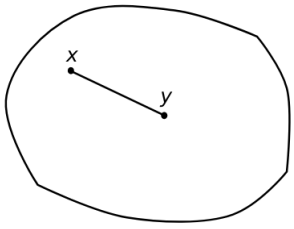


Figure 2.1: Insieme convesso

La maggior parte degli insiemi non sono convessi, tuttavia esiste un risultato molto importante: ogni insieme non convesso S può essere "completato" nel seguente modo:

$$\begin{aligned} conv(S) &= \bigcup \{conv(x, y) : x, y \in S\} \\ &= \bigcap \{C : C \text{ is convex } \wedge C \supseteq S\} \end{aligned}$$

Il completamento $conv(S)$ è detto **Convex hull of S** ed è un insieme convesso. Può essere visto come l'insieme che si ottiene aggiungendo all'insieme di partenza S le convex hull di tutte le coppie $x, y \in S$, oppure come il più piccolo insieme convesso che contiene S.

In realtà vale:

$$C \text{ è convesso} \iff C = conv(C)$$

Verificare se un insieme è convesso può essere complicato, tuttavia ci sono casi in cui siamo noi a dover costruire un insieme su cui lavorare; è utile dunque sapere che esistono dei prototipi di insiemi convessi e delle operazioni insiemistiche che preservano la convessità (vedi slides).

2.2.2 Funzioni convesse

Una funzione f è detta convessa se:

$$f \text{ convessa} \equiv epi(f) \text{ convesso}$$

o equivalentemente si ha:

$$\forall x, y \in dom(f), \alpha \in [0,1] \qquad \alpha f(x) + (1-\alpha) f(y) \geq f(\alpha x (1-\alpha) y)$$

ovvero per ogni coppia (x, y), tutti i punti lungo il segmento che congiunge x ed y si trovano al di sopra del valore della funzione. Se nella seconda definizione sostituiamo $\geq$ con $>$, allora f è detta *strettamente* convessa. Le funzioni per cui vale l'uguaglianza sono quelle lineari. Graficamente si ha:

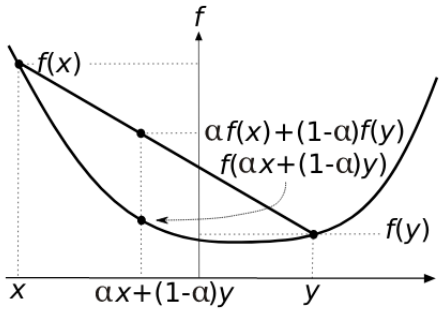


Figure 2.2: Convex Function

Un risultato importante è il seguente:

$$f \text{ convessa} \implies S(f, v) \text{ convesso } \forall v \in \mathbb{R}$$

cioè tutti i sublevel set sono convessi. Anche in questo caso esistono prototipi di funzioni convesse e operazioni su di esse che preservano la convessità (vedi slides).

Per capire invece se, data una funzione, questa è convessa, possiamo innanzitutto controllare se soddisfa alcune proprietà. In particolare, poiché siamo interessati a problemi di minimizzazione, le funzioni che ci interessano devono essere almeno *lower-semicontinuous* (l.s.c.), dunque è importante il seguente risultato:

$$f \text{ closed convex} := \text{f l.s.c.} \iff epi(f) \text{ closed} \iff S(f, v) \text{ closed } \forall v$$

Inoltre una funzione convessa è sempre *Lipshitziana* in ogni insieme compatto contenuto nella parte interna del suo dominio:

$$f \text{convex} \implies f \text{ Lipshitz continuos } \forall \text{ bounded convex } S \subseteq \text{int } dom(f)$$

mentre al contorno $\partial dom f$ non ci sono limitazioni (es: funzione indicatrice). Se $intdom(f)$ è vuota, allora le proprietà viste valgono per la parte interna relativa del dominio di f.

Convessità ed informazioni al primo ordine: valgono le seguenti proprietà:

$$f \text{ convex} \implies \nabla f \text{ "exists almost everywhere"}$$

$$f \in C^1 \text{ convex on } C \text{ convex} \iff f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle \quad \forall x, y \in C$$

ovvero una funzione f convessa è differenziabile *quasi ovunque* ed in ogni punto il suo grafico sta "sopra" al suo modello al primo ordine.

In particolare l'ultima proprietà ci garantisce che: in presenza di un punto stazionario, in cui il modello al primo ordine è semplicemente una retta orizzontale, tutto il grafico di f si trova al di sopra di questa retta, dunque il punto in questione è un *minimo globale*.

Convessità ed informazioni al secondo ordine: dal momento che verificare le proprietà precedenti risulta troppo costoso, può essere utile estrarre informazioni dalle derivate di ordine successivo (se la funzione è almeno C^2). In particolare si ha:

$$f \text{ convex} \equiv \nabla^2 f(x) \succeq 0$$

[strettamente/fortemente] convessa se [$\succ 0 / \succeq \tau I$].

Chapter 3

Gradient-type methods

3.1 Ottimizzazione non vincolata

Come abbiamo già detto, un algoritmo di ottimizzazione parte da un punto iniziale x^0 e, tramite un processo, calcola i punti successivi $x^i \to x^{i+1}$ in modo che la *successione* così costruita converga ad una soluzione ottima. L'obiettivo è adesso studiare la *convergenza* e l' *efficienza* di tali algoritmi, assicurandoci che funzionino "bene" qualunque sia la scelta del punto iniziale.

Nella pratica gli algoritmi che studieremo si limitano a cercare un punto stazionario e, solamente nel caso in cui la funzione da minimizzare è convessa, si ha la certezza che il risultato ottenuto corrisponde alla soluzione ottima.

Possiamo avere tre tipi di *convergenza*:

- (strong): $\{x^i\} \to x_*$: l'intera successione converge alla soluzione ottima;
- (weaker): tutti i punti di accumulazione di $\{x^i\}$ sono soluzioni ottime;
- (weakest): almeno un punto di accumulazione di $\{x^i\}$ è una soluzione ottima.

Esistono due approcci generali per costruire il processo:

- **Line search:** prima si sceglie la *direzione* $d^i \in \mathbb{R}^n$ in cui muoversi e poi di quanto (step-size) $\alpha^i \in \mathbb{R}$, in modo che: $x^{i+1} = x^i + \alpha^i d^i$;
- **Trust region:** prima si sceglie la dimensione dello step α^i e poi la direzione d^i.

Useremo quasi sempre il primo approccio.

In entrambi gli approcci è fondamentale stabilire il modo (non solo l'ordine) in cui si scelgono direzione e step-size, e questo dipende dal modello che scegliamo per approssimare f, scelta che dunque diventa cruciale. Qual è il modello più semplice che possiamo scegliere? il modello al primo ordine; i metodi che utilizzano il modello al primo ordine vengono detti *metodi del gradiente*.

3.2 Metodo del gradiente

Stiamo assumendo che la funzione sia differenziabile e che siamo in grado di calcolare facilmente il gradiente di f.

Ad ogni iterazione i possiamo scrivere il modello al primo ordine:

$$L^i(x) = L_{x^i}(x) = f(x^i) + \nabla f(x^i)(x - x^i)$$

Potremmo pensare di minimizzare questo modello invece della funzione di partenza, tuttavia ci sono due problemi:

- Si tratta di una funzione lineare, dunque *non è limitata inferiormente* su $\mathbb{R}^n$.
- Il modello al primo ordine approssima f solo *localmente*.

Possiamo però utilizzare il modello al primo ordine per scegliere la direzione in cui muoverci. Sappiamo infatti che localmente $L^i(x)$ approssima bene f, dunque è ragionevole scegliere la direzione lungo la quale il modello decresce più velocemente.

Consideriamo allora una direzione generica d (di norma 1) e scriviamo il modello come:

$$L^i(x^i + \epsilon d) = f(x^i) + \underbrace{\nabla f(x^i)\epsilon d}_{\text{da minimizzare}}$$

la direzione che cerchiamo è quella per cui il termine $\nabla f(x^i)\epsilon d$ è minimo, ovvero:

$$d = -\nabla f(x^i) \qquad \textbf{anti-gradiente} \tag{3.1}$$

A questo punto, una volta scelta la direzione, il problema è scegliere la dimensione dello step α. Proprio perché il modello funziona bene solo localmente, se scegliamo un α troppo grande, allora potremmo arrivare ad un punto peggiore di quello di partenza; d'altra parte se α è troppo piccolo la funzione descresce troppo lentamente.

Quello che si fa allora è la cosidetta **exact line search**, ovvero si considera la restrizione di f lungo la direzione scelta e si risolve un ulteriore problema di ottimizzazione:

$$\alpha^i \in argmin\{f(x^i + \alpha d^i) : \alpha \geq 0\} \tag{3.2}$$

che sicuramente è più semplice di quello di partenza in quanto dobbiamo minimizzare una funzione di una sola variabile.

Questa è l'idea, adesso vediamo come metterla in pratica prima nel caso semplice di una funzione quadratica e poi in un caso più generale.

3.2.1 Metodo del gradiente per funzioni quadratiche

Sappiamo che una funzione quadratica si scrive come:

$$f(x) = \frac{1}{2}x^T Qx + qx \qquad Q \succeq 0 \tag{3.3}$$

ed affinché ammetta un minimo la matrice Q deve essere semi-definita positiva. A questo punto l'obiettivo è trovare un punto stazionario, dunque risolvere $\nabla f(x) = 0$, dove in questo caso è facile calcolare il gradiente: $\nabla f(x) = Qx + q$. Possiamo procere in due modi:

- Risolviamo direttamente il sistema lineare:

  $$\text{troviamo } x_* \text{ che risolve } Qx = -q \tag{3.4}$$

  ma il costo sarà $O(n^3)$;
- Utilizziamo il *Metodo del gradiente*: costo $O(n^2)$.

Metodo del gradiente: dobbiamo scegliere d^i ed α^i. Abbiamo detto che la direzione sarà quella dell'anti-gradiente, dunque:

$$d^i = -\nabla f(x^i) = -Qx^i - q \tag{3.5}$$

Mentre α^i sarà la soluzione di $\alpha^i \in argmin\{f(x^i + \alpha d^i) : \alpha \geq 0\}$, ovvero:

$$f(x^i + \alpha d^i) = \frac{1}{2}(x^i + \alpha x)^T Q(x^i + \alpha d^i) + q^T(x^i + \alpha d^i) \tag{3.6}$$

calcoliamo la derivata rispetto ad α e poniamola uguale a 0:

$$\frac{\partial}{\partial\alpha}f(x+\alpha d) = \frac{1}{2}x^TQd + \frac{1}{2}d^TQx + q^Td + \alpha d^TQd = 0 \tag{3.7}$$

Nota che x^TQd è un numero, dunque è uguale al suo trasposto: $(x^TQd)^T = d^TQ^Tx$. Inoltre $Q^T = Q$ in quanto una matrice associata ad una forma quadratica è sempre simmetrica. Dunque:

$$\alpha = -\frac{1}{d^TQd}\left[\frac{1}{2}d^TQx + \frac{1}{2}d^TQx + (q^Td)^T\right] \tag{3.8}$$

$$= -\frac{1}{d^TQd}\left[d^TQx + d^Tq\right] \tag{3.9}$$

$$= -\frac{1}{d^TQd}d^T\underbrace{(Qx+q)}_{=-d} \tag{3.10}$$

$$= \frac{\|d\|^2}{d^TQd} \tag{3.11}$$

A questo punto abbiamo sia d che α, dunque possiamo scrivere la procedura.

Algoritmo: Steepest Descent for Quadratic functions

E' necessario avere in input la funzione (cioè Q e q), un punto iniziale x ed infine anche un parametro relativo alla tolleranza ϵ, in quanto richiedere che il gradiente sia nullo equivale nella pratica a richiedere che sia sufficientemente piccolo (quantificare questa cosa può essere complicato).

Algorithm 1: Steepest Descent for Quadratic functions

procedure: $x = \mathbf{SDQ}(Q, q, x, \epsilon)$
while $\|\nabla f(x)\|^2 > \epsilon$ **do**
  $d \leftarrow -\nabla f(x)$;
  $\alpha \leftarrow \frac{\|d\|^2}{d^TQd}$;
  $x \leftarrow x + \alpha d$;
end

Questo algoritmo è monotono, quindi se nell'implementazione il valore della funzione non decresce allora ci deve essere un errore. Invece non è detto che la norma del gradiente abbia un andamento monotono. Dimostriamo adesso che l'algoritmo converge ad un punto stazionario.

Convergenza

La successione di punti $\{x^i\}$ prodotta dall'algoritmo gode della seguente proprietà:

$$\left\langle \nabla f(x^i), \nabla f(x^{i+1}) \right\rangle = 0 \tag{3.12}$$

Proof. Ad ogni iterazione eseguiamo una *exact-line search*, dunque il punto successivo è scelto proprio perché la sua derivata direzionale lungo $d^i = -\nabla f(x^i)$ è nulla, ovvero il gradiente in x^{i+1} è perpendicolare al gradiente in x^i (Matlab: si vede chiaramente che ci si sposta sempre di 90 gradi). □

Questa proprietà ci garantisce che *se* la successione $\{x^i\} \to x$ converge, allora converge ad un punto stazionario, infatti:

Proof.

$$\lim_{i\to\infty} \left\langle \nabla f(x^i), \nabla f(x^{i+1}) \right\rangle = \langle \nabla f(x), \nabla f(x) \rangle = \|\nabla f(x)\|^2 = 0 \tag{3.13}$$

□

Il problema sarebbe dimostrare che effettivamente la successione converge; se fossimo in un insieme compatto sarebbe semplice, ma non è questo il caso, tuttavia se la funzione è convessa allora basta dimostrare che una volta all'interno di un ellissoide (sublevel set) non si esce più da quell'ellissoide, e questo è un insieme compatto.

Notiamo infine che poiché non richiediamo che il gradiente vada a 0 ma solo che sia sufficientemente piccolo ($\epsilon > 0$), allora l'algoritmo termina in un tempo finito.

Efficienza

Di solito l'efficienza di un algoritmo si misura contando il numero di iterazioni in funzione della dimensione dell'input. In un algoritmo di ottimizzazione è più complicato in quanto potenzialmente potrebbe girare all'infinito a meno che non stabiliamo una certa tolleranza (ϵ).

Assumendo che la soluzione ottima esista, il modo naturale per misurare l'efficienza sarebbe vedere quanto velocemente decresce la successione $\left\|x^i - x_*\right\|$, dove x^i è il punto all'iterazione i ed x_* è la soluzione ottima. Tuttavia questo approccio presenta due problemi: è complicato tecnicamente (perché sono punti di $\mathbb{R}^n$) e richiede che l'intera successione converga alla soluzione ottima.

Quello che si fa allora è vedere quanto velocemente decresce $f(x^i) - f_*$. Definiamo il **rate/order of convergence** come:

$$\lim_{i\to\infty} \frac{f(x^{i+1}) - f_*}{(f(x^i) - f_*)^p} = R \tag{3.14}$$

e classifichiamo i tipi di convergenza come:

Convergeza	**p**	**R**	**Error**
Sublinear	1	1	$1/i$, $1/i^2$
Linear	1	< 1	γ^i, $\gamma < 1$
Superlinear	1	0	γ^{i^2}, $\gamma < 1$
Quadratic	2	> 0	γ^{2^i}, $\gamma < 1$

Table 3.1: Tipi di convergenza

Per avere un'idea: la convergenza quadratica è estremamente veloce e significa che il numero di cifre in accordo raddoppia ad ogni iterazione. Il metodo del gradiente invece ha una convergenza lineare (il grafico iterazioni vs. log(errore) è una retta con pendenza negativa che dipende dal valore di R).

Convergenza Lineare significa che ad un certo punto si ha:

$$f(x^{i+1}) - f(x_*) \approx R(f(x^i) - f(x_*)) \tag{3.15}$$

quindi possiamo scrivere l'errore all'iterazione i-esima in funzione dell'errore alla prima iterazione come:

$$f(x^i) - f_* \leq (f(x^1) - f_*)R^i \tag{3.16}$$

dunque decresce esponenzialmente ($R < 1$) ed il punto di partenza è importante. A questo punto possiamo scrivere il numero di iterazioni necessarie per ottenere la soluzione con una certa tolleranza ϵ come:

$$f(x^i) - f_* \leq \epsilon \implies i \geq \log\left(\frac{f(x^1) - f_*}{\epsilon}\right) \frac{1}{\log \frac{1}{R}} \tag{3.17}$$

quindi l'errore va come $O(\log 1/\epsilon)$ che è un buon risultato, però la costante moltiplicativa può andare ad infinito se $R \to 1$

Analisi dell'efficienza: caso quadratico

Per fare un'analisi dell'efficienza, è necessario stimare quanto velocemente convergiamo al valore ottimo, ma quest'ultimo in generale non è noto; tuttavia, nel caso di una funzione quadratica conosciamo già la soluzione ottima: x_* è la soluzione di $\nabla f(x) = Qx + q = 0$, dunque:

$$x_* = -Q^{-1}q \implies f_* = -\frac{1}{2}q^T Q^{-1} q \tag{3.18}$$

Questo ci permette di fare un'analisi più dettagliata dell'efficienza. Per prima cosa facciamo vedere che l'errore $f_*(x) = f(x) - f_*$ può essere scritto come:

$$\begin{aligned} f_*(x) &= \frac{1}{2}(x - x_*)^T Q (x - x_*) && (3.19)\\ &= \frac{1}{2}x^T Q x + \frac{1}{2}x_*^T Q x_* - x_*^T Q x && (3.20)\\ &= \frac{1}{2}x^T Q x + \frac{1}{2}(Q^{-1}q)^T Q Q^{-1} q + (Q^{-1}q)^T Q x && (3.21)\\ &= \frac{1}{2}x^T Q x + q^T x + \frac{1}{2}q^T Q^{-1} q && (3.22)\\ &= f(x) - f_* && (3.23) \end{aligned}$$

ovvero l'errore in x è la distanza da x_* in norma $\|\cdot\|_Q$. Oppure, in altre parole, il centro dell'ellissoide è il punto di minimo della funzione.

Si può dimostrare che se $Q \succ 0$ allora:

$$f_*(x^{i+1}) = \left(1 - \frac{\left\|d^i\right\|^4}{((d^i)^T Q d^i)((d^i)^T Q^{-1} d^i)}\right) f_*(x^i) \tag{3.24}$$

dove il termine tra parentesi corrisponde al **rate di convergenza**, che notiamo essere minore di 1. Tuttavia vogliamo cercare di riscriverlo in maniera che non dipenda dall'iterazione i che stiamo considerando, ovvero vogliamo trovare un limite superiore in modo da poter fare delle analisi qualitative sul rate di convergenza a partire dalla forma della funzione quadratica.

Indichiamo con $\lambda^1 \geq \cdots \geq \lambda^n > 0$ gli autovalori di Q ordinati in senso decrescente. Dunque gli autovalori di Q^{-1} saranno $1/\lambda^n \geq \cdots \geq 1/\lambda^1 > 0$. In generale valgono le seguenti:

$$x^T Q x \leq \lambda^1 \|x\|^2 \qquad x^T Q^{-1} x \leq \frac{1}{\lambda^n}\|x\|^2 \qquad \forall x \in \mathbb{R}^n \tag{3.25}$$

dunque:

$$\frac{\|x\|^2}{x^T Q^{-1} x} \geq \lambda^n \qquad \frac{\|x\|^2}{x^T Q x} \geq \frac{1}{\lambda^1} \tag{3.26}$$

da cui ricaviamo:

$$\forall x \in \mathbb{R}^n \quad \frac{\|x\|^4}{((x)^T Q x)((x)^T Q^{-1} x)} \geq \frac{\lambda^n}{\lambda^1} \implies R \leq \left(1 - \frac{\lambda^n}{\lambda^1}\right) \tag{3.27}$$

Da questo risultato si spiega subito l'andamento già visto (Matlab): si ha una convergenza più veloce quando il rapporto tra il più grande ed il più piccolo autovalore di Q è vicino a 1. Quando gli ellissoidi invece erano molto "allungati" la convergenza era più lenta.

Una stima migliore, che però non dimostriamo, è la seguente:

$$\forall x \in \mathbb{R}^n \quad \frac{\|x\|^4}{((x)^T Q x)((x)^T Q^{-1} x)} \geq \frac{4\lambda^1 \lambda^n}{(\lambda^1 + \lambda^n)^2} \tag{3.28}$$

da cui si ottiene infine:

$$R \leq \left(\frac{\lambda^1 - \lambda^n}{\lambda^1 + \lambda^n}\right)^2 \implies f(x^{i+1}) - f_* \leq \left(\frac{\lambda^1 - \lambda^n}{\lambda^1 + \lambda^n}\right)^2 (f(x^i) - f_*) \tag{3.29}$$

L'aspetto positivo di questo risultato è che *non dipende dalla dimensione n* (quindi è scalabile), quello negativo è che invece dipende fortemente dalla forma della matrice Q.

Prima di passare al caso di una funzione generica, discutiamo più nel dettaglio il criterio che abbiamo utilizzato per stoppare le iterazioni.

Stopping Criterion

Come scegliamo ϵ? Innanzitutto distinguiamo tra errore assoluto e relativo:

$$\epsilon_A = f(x^i) - f_* \leq \epsilon \qquad \epsilon_R = \frac{f(x^i) - f_*}{|f_*|} = \frac{\epsilon_A}{|f_*|} \leq \epsilon \tag{3.30}$$

$$\text{Absolute error} \qquad \text{Relative error} \tag{3.31}$$

chiaramente è meglio lavorare con errori relativi, anche se è necessario accertarsi che il denominatore non sia nullo (in quel caso si può fare una traslazione). Tuttavia, anche in questo caso, il problema è che per utilizzare un criterio del genere è necessario conoscere il anticipo il valore ottimo f_*. L'unica cosa che possiamo fare allora è cercare di stimare f_* ed in particolare cercare di trovare un limite inferiore $\underline{f} \leq f_*$.

Altrimenti possiamo trovare un'alternativa: ad esempio nell'algoritmo appena studiato abbiamo utilizzato come criterio la norma del gradiente:

$$\left\|\nabla f(x^i)\right\| \leq \epsilon \qquad \frac{\left\|\nabla f(x^i)\right\|}{\|\nabla f(x^0)\|} \leq \epsilon \tag{3.32}$$

$$\text{(Assoluto)} \qquad \text{(Relativo)} \tag{3.33}$$

tuttavia la relazione fra questo ϵ e quello precedente non è banale. Dunque ϵ è un parametro algoritmico che deve essere scelto con cura. (Da fare nel progetto: cercare di stimare il valore ottimo e confrontare i due ϵ)

Notiamo infine: se f è convessa ed il minimo è contenuto in un insieme compatto, allora un limite inferiore di f_* si ottiene calcolando il minimo del modello al primo ordine all'interno dell'insieme compatto (infatti la funzione sta sempre sopra perché è convessa).

3.2.2 Metodo del gradiente per funzioni generiche

Nel caso di una funzione non quadratica, l'algoritmo rimane quasi identico:

Convergenza

La dimostrazione di *convergenza* già vista non sfruttava le proprietà delle funzioni quadratiche, dunque è valida in generale. Quello che cambia è che in questo caso non è semplice fare *l'exact-line search.*

Tuttavia, notiamo che: la proprietà fondamentale che ci ha permesso di dimostrare la convergenza è $\left\langle \nabla f(x^i), \nabla f(x^{i+1}) \right\rangle = 0$; ma questa è valida non solo per i minimi locali, ma anche per i massimi e i punti di sella; dunque non dobbiamo per forza trovare un minimo

Algorithm 2: Steepest Descent for general functions

procedure: $x = \mathbf{SDQ}(f, x, \epsilon)$
while $\|\nabla f(\mathbf{x})\|^2 > \epsilon$ **do**
  $d \leftarrow -\nabla f(x)$;
  $\alpha \leftarrow argmin\{f(x + \alpha d)\}$;
  $x \leftarrow x + \alpha d$;
end

quando facciamo la *line-search* ma basta che troviamo un punto stazionario (in cui il valore della funzione è comunque più piccolo di quello precedente) e la convergenza è assicurata lo stesso. Questo problema prima non si presentava in quanto le funzioni quadratiche che ammettono un minimo devono essere convesse, in generale invece possiamo avere anche massimi e punti di sella.

Efficienza

L'efficienza "nella coda" è praticamente la stessa. Non lo dimostriamo ma vale il seguente risultato:

Sia $f \in C^2$ ed x_* un minimo locale tale che $\nabla^2 f(x_*) \succ 0$, allora

$$\{x^i\} \to x_* \implies \{f(x^i)\} \to f(x_*) \quad \text{Linearly} \tag{3.34}$$

cioè abbiamo lo stesso tipo di convergenza (lineare) con la stessa costante R che dipende dagli autovalori λ_1 e λ_n della matrice Hessiana $\nabla^2 f(x_*)$. Intuitivamente: più ci avviciniamo al minimo locale (= "nella coda") e più il modello al secondo ordine, che è una funzione quadratica, approssima bene f. Per questo motivo possiamo distinguere due fasi della convergenza: una globale, di cui non sappiamo molto, ed una locale in cui invece abbiamo questo andamento.

Line-search

Il problema fondamentale su cui ci concentriamo adesso è trovare un minimo locale di:

$$\phi(\alpha) = f(x + \alpha d) \tag{3.35}$$

ovvero trovare le radici di $\phi'(\alpha)$. In generale è molto complicato, quindi spesso troveremo solo delle soluzioni approssimate, cioè tali per cui:

$$|\phi'(\alpha)| \leq \epsilon' \tag{3.36}$$

A questo punto però la convergenza dipende non solo da ϵ ma anche da ϵ', infatti, se $\phi'(\alpha)$ non è esattamente 0, allora non vale più la proprietà $\left\langle \nabla f(x^i), \nabla f(x^{i+1}) \right\rangle = 0$. Dunque, fissato ϵ, dobbiamo trovare una relazione tra ϵ ed ϵ' che ci garantisca ancora la convergenza. Possiamo scrivere allora:

$$|\phi'(\alpha)| = |\left\langle d^i, \nabla f(x^{i+1}) \right\rangle| = \left| \left\langle \frac{\nabla f(x^i)}{\|\nabla f(x^i)\|}, \nabla f(x^{i+1}) \right\rangle \right| \tag{3.37}$$

dove abbiamo normalizzato la direzione d^i (il problema della divisione per 0 non si pone perché dividiamo per la norma del gradiente). Da questa relazione, se $\{x^i\} \to x$, otteniamo:

$$\lim_{i \to \infty} \left| \left\langle \frac{\nabla f(x^i)}{\|\nabla f(x^i)\|}, \nabla f(x^{i+1}) \right\rangle \right| = \left| \left\langle \frac{\nabla f(x)}{\|\nabla f(x)\|}, \nabla f(x) \right\rangle \right| = \|\nabla f(x)\| \leq \epsilon' = \epsilon \tag{3.38}$$

dunque l'algoritmo converge lo stesso con $\boxed{\epsilon' = \epsilon}$.

Se non avessimo normalizzato d^i, avremmo ottenuto:

$$\epsilon' = \epsilon \left\| \nabla f(x^i) \right\| \tag{3.39}$$

da cui è più evidente il seguente problema: la *line search* deve essere sempre più accurata ad ogni iterazione (perché la norma va a 0) e questa richiesta si traduce in un numero più elevato di iterazioni; questo è il motivo per cui nella pratica spesso si utilizzano degli approcci non esatti (*inexact line search*) ma, per capirli, è necessario prima studiare bene come funziona l' *exact line search*.

3.3 "Exact" Line Search

3.3.1 Approcci al primo ordine

L'obiettivo è trovare α' tale che:

$$\phi(\alpha') \approx 0 \tag{3.40}$$

Non è detto che esista un punto α' in cui ϕ' si annulla ma, assumendo che f sia di classe C^1, possiamo sfruttare il teorema del valore intermedio per ottenere una condizione di esistenza: infatti ∇f continuo $\implies$ ϕ' continua e $\phi(\alpha = 0) < 0$, dunque se riusciamo a trovare un punto $\bar{\alpha}$ in cui $\phi'(\bar{\alpha}) > 0$, allora la continuità di ϕ' ci garantisce l'esistenza di α'.

Quindi per prima cosa cerchiamo $\bar{\alpha}$; una soluzione ovvia è:

Algorithm 3: Find $\bar{\alpha}$

```
α' ← 1; // or whatever value > 0
while φ'(ᾱ) ≤ −ϵ do
    ᾱ ← 2ᾱ; // or whatever factor > 1
end
```

questa procedura funziona *quasi* sempre; in particolare funziona *sempre* se la funzione è *coerciva*: $\lim_{\alpha \to \infty} \phi(\alpha) = \infty$.

Una volta trovata $\bar{\alpha}$ abbiamo ristretto l'intervallo in cui cercare α' a $[0, \bar{\alpha}]$. Due modi possibili per procedere sono: *metodo della bisezione* e *interpolazione*.

Bisection Method

Algorithm 4: $\alpha = \textbf{LSBM}(\phi', \bar{\alpha}, \epsilon)$

```
α₋ ← 0; α ← α₊ ← ᾱ;
while |φ'(α)| > ϵ do
    α ← (α₊ + α₋)/2;
    if φ'(α) < 0 then
        α₋ ← α;
    else
        α₊ ← α;
    end
end
```

In pratica dato un intervallo, dove per costruzione agli estremi ϕ' assume segno opposto, si considera il punto di mezzo e si controlla il segno di ϕ'; successivamente si procede ricorsivamente sulla porzione di intervallo ai cui estremi ϕ' ha segno opposto, finchè non ci si avvicina abbastanza allo 0.

Consideriamo adesso le due successioni relative agli estremi dell'intervallo. Possiamo scrivere la larghezza dell'intervallo all'iterazione k come:

$$\alpha_+^k - \alpha_-^k = \bar{\alpha} 2^{-k} \tag{3.41}$$

in quanto inizialmente la dimensione è $\bar{\alpha}$ e ad ogni iterazione viene dimezzata. Dunque se $k \to \infty$ la dimensione tende a 0 e le due successioni convergono (esponenzialmente) ad uno stesso punto α_*. La cosa importante è che anche la successione $\{\phi'(\alpha^k)\}$ converge a $\phi'(\alpha_*) = 0$; inoltre se ϕ' è localmente Lipshitziana, allora la convergenza è lineare.

Vediamo adesso un altro modo più efficiente per risolvere lo stesso problema.

Interpolation

L'idea è: conosciamo il valore di ϕ e ϕ' agli estremi dell'intervallo (4 valori in totale), quindi possiamo cercare di approssimare meglio ϕ con una funzione quadratica. Tuttavia una funzione quadratica ha 3 parametri da fissare, mentre noi abbiamo 4 condizioni. Dunque ci restringiamo ad approssimare ϕ con una funzione che soddisfa solo le condizioni sulle derivate. Cioè, data una generica funzione quadratica $a\alpha^2 + b\alpha + c$, imponiamo:

$$2a\alpha_+ + b = \phi'(\alpha_+) \qquad e \qquad 2a\alpha_- + b = \phi'(\alpha_-) \tag{3.42}$$

risolvendo $2a\alpha + b = 0$ otteniamo:

$$\alpha = \frac{\alpha_- \phi'(\alpha_+) - \alpha_+ \phi'(\alpha_-)}{\phi'(\alpha_+) - \phi'(\alpha_-)} \tag{3.43}$$

nota: è una combinazione convessa di α_+ e α_-. Sostanzialmente si tratta del metodo delle secanti, cioè si considera un segmento che congiunge $\phi'(\alpha_-)$ e $\phi'(\alpha_+)$ e si calcola il punto α in cui si annulla. Successivamente si procede come il metodo della bisezione utilizzando però l'α calcolato in questo modo.

Tipicamente funziona meglio rispetto al considerare semplicemente il punto di mezzo dell'intervallo, infatti la convergenza dell'*Interpolation* è *Superlinear*, mentre per il metodo delle *Bisezione* la convergenza era *Lineare*. Si potrebbe fare anche una *cubic interpolation* e si avrebbe una convergenza quadratica, ma è complicato e non è necessario in quanto si può ottenere una convergenza quadratica anche utilizzando degli approcci al secondo ordine.

3.3.2 Approcci al secondo ordine

Per utilizzare le informazioni al secondo ordine è ovviamente necessario che la funzione sia di classe C^2; questo implica che $\phi''(\alpha) = d^T \nabla^2 f(x + \alpha d) d$ esiste ed è continua.

Newton's method

Costruiamo il modello al primo ordine di ϕ' ($\equiv$ modello al secondo ordine di f):

$$\phi'(\alpha) \approx \phi'(\alpha^k) + \phi''(\alpha^k)(\alpha - \alpha^k) \tag{3.44}$$

ponendolo uguale a 0 otteniamo:

$$\phi'(\alpha) = 0 \implies \alpha = \alpha^k - \frac{\phi'(\alpha^k)}{\phi''(\alpha^k)} \tag{3.45}$$

Numericamente dobbiamo stare attenti al denominatore ϕ'' che può annullarsi.

Questo metodo funziona molto bene anche in più dimensioni; inoltre la convergenza è quadratica ma solo se il punto iniziale è "abbastanza buono", altrimenti potrebbe anche non convergere.

Scriviamo la procedura come:

Algorithm 5: Newton's Method

$\alpha = \textbf{LSNM}(\phi', \phi'', \alpha, \epsilon)$
while $|\phi'(\alpha)| > \epsilon$ **do**
  $\alpha \leftarrow \alpha - \frac{\phi'(\alpha)}{\phi''(\alpha)}$;
end

Dimostriamo adesso la convergenza del metodo di Newton:

Theorem 4. *Sia $\phi \in C^3$, $\phi'(\alpha_*) = 0$ e $\phi''(\alpha_*) \neq 0$, allora $\exists \delta > 0$ tale che:*

$$\alpha^0 \in [\alpha_* - \delta, \alpha_* + \delta] \implies \{\alpha^k\} \to \alpha_* \quad con \quad p = 2 \tag{3.46}$$

cioè esiste un intervallo all'interno del quale la convergenza è quadratica.

Proof. Scriviamo la differenza tra il valore all'iterazione $k+1$ e il valore cercato α_* come:

$$\alpha^{k+1} - \alpha_* = \alpha^k - \frac{\phi'(\alpha^k)}{\phi''(\alpha^k)} - \alpha_* - \underbrace{\phi'(\alpha_*)}_{=0} \tag{3.47}$$

$$= \frac{\phi'(\alpha^k) + \phi''(\alpha^k)(\alpha^k - \alpha_*) - \phi'(\alpha_*)}{\phi''(\alpha^k)} \tag{3.48}$$

a questo punto utilizziamo la secondo forma della formula di Taylor al secondo ordine (dove inglobiamo il residuo nel parametro $\beta \in [\alpha^k, \alpha_*]$) per riscrivere $\phi'(\alpha_*)$ come:

$$\phi'(\alpha_*) = \phi'(\alpha^k) + \phi''(\alpha^k)(\alpha^k - \alpha_*) + \frac{\phi'''(\beta)(\alpha^k - \alpha_*)^2}{2} \tag{3.49}$$

sostituendo otteniamo:

$$\alpha^{k+1} - \alpha_* = \left[-\frac{\phi'''(\beta)}{2\phi''(\alpha^k)} \right] (\alpha^k - \alpha_*)^2 \tag{3.50}$$

che corrisponde ad una convergenza quadratica se riusciamo a trovare un upper-bound al termine tra parentesi quadre. Per farlo sfruttiamo la continuità di ϕ'' e ϕ''': poiché sono continue in un intervallo limitato, allora saranno limitate sia superiormente che inferiormente, dunque fissate due costanti k_1 e k_2 allora $\exists \delta > 0$ tale che per ogni $\alpha, \beta \in [\alpha_* - \delta, \alpha_* + \delta]$ si ha:

$$\phi''(\alpha) \geq k_2 > 0 \qquad e \qquad |\phi'''(\beta)| \leq k_1 < \infty \tag{3.51}$$

da cui otteniamo:

$$|\alpha^{k+1} - \alpha_*| \leq \left[\frac{k_1}{2k_2} \right] (\alpha^k - \alpha_*)^2 \tag{3.52}$$

per la convergenza quadratica non è necessario che la costante sia minore di 1 ma è sufficiente che esista. La cosa importante invece è che per avere la convergenza tutto il termine al secondo membro deve essere minore di 1, cioè:

$$\frac{k_1(\alpha^k - \alpha_*)}{2k_2} \leq 1 \implies |\alpha^{k+1} - \alpha_*| \leq |\alpha^k - \alpha_*| \implies \{\alpha^k\} \to \alpha_* \tag{3.53}$$

quindi la convergenza si ha solo se all'inizio $|\alpha^0 - \alpha_*|$ è piccolo abbastanza (minore di 1) ed in quel caso la convergenza è quadratica (in pratica non è ovvio). □

Riassumendo: il metodo di Newton converge quadraticamente e funziona molto bene anche nel caso multidimensionale, ma il punto iniziale deve essere sufficientemente vicino alla soluzione cercata.

3.3.3 Approcci all'ordine zero

In certi casi il calcolo del gradiente (e a maggior ragione dell'Hessiana) può essere troppo costoso, quindi vediamo un metodo che non utilizza queste informazioni ma calcola solo il valore della funzione. Notiamo che più informazioni abbiamo dalle derivate e meno sono i punti necessari in cui calcolare la funzione.

Il calcolo della derivata è comunque necessario per trovare la direzione, ma adesso vogliamo risolvere solo l'ultimo step della line-search: trovare α_* tale che $\phi'(\alpha_*) = 0$ nell'intervallo $[0, \bar{\alpha}]$. Il procedimento è il seguente: si individuano due punti $\alpha_1, \alpha_2 \in [0, \bar{\alpha}]$ e si calcola il valore della funzione agli estremi dell'intervallo ed in questi punti. L'intervallo iniziale risulta diviso in 3 parti e ad ogni iterazione, a seconda dei valori calcolari della funzione, si elimina o la parte destra o quella sinistra e si ripete il procedimento.

La cosa importante è che i punti scelti devono creare degli intervalli abbastanza bilanciati, in modo da garantire una convergenza efficiente: per farlo si scelgono i due punti ad una distanza dagli estremi pari a $(1 - r)$ dove r è la sezione aurea; in questo modo la larghezza dell'intervallo decresce come $(1 - r)^k \approx 0.4^k$ (k è il numero di iterazioni).

Si potrebbe fare anche meglio usando la sequenza di Fibonacci, ma non entriamo nel dettaglio in quanto questo approccio è molto poco usato. Passiamo invece a quello che viene fatto davvero in pratica, ovvero l'*Inexact Line Search*.

3.4 Inexact Line Search: Armijo-Wolfe

Il problema dell'*Exact line search* è che l'accuratezza richiesta aumenta ad ogni iterazione, infatti avevamo visto che:

$$\epsilon' = \epsilon \left\| \nabla f(x^i) \right\| \tag{3.54}$$

In realtà, intuitivamente, non è davvero necessario trovare un minimo locale, l'importante è che la funzione decresca abbastanza. Le **Condizioni di Armijo** definiscono formalmente questo concetto:

$$(A) \quad \phi(\alpha) \leq \phi(0) + m_1 \alpha \phi'(0) \qquad 0 < m_1 < 1 \tag{3.55}$$

L'idea è che più siamo vicini ad un minimo locale e più il valore della derivata si avvicina a 0; dunque la condizione che stabilisce se si ha un decremento sufficiente deve dipendere dal valore della derivita nel punto iniziale. Si considera allora il modello al primo ordine in α e si "appiattisce" di un fattore m_1 in modo che i punti che soddisfano questa condizione siano tutti quelli che si trovano al di sotto di questa retta. La scelta di m_1 non è banale, ma in pratica funziona quasi sempre $m_1 = 0.0001$. Il risultato è mostrato in figura:

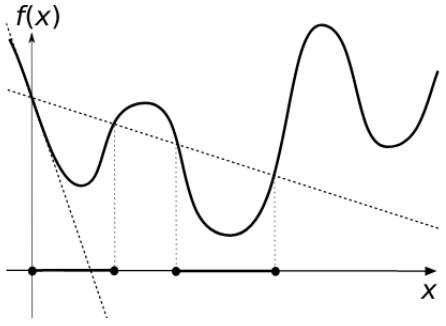


Figure 3.1: Armijo Condition

tuttavia, in questo modo, anche i punti arbitrariamente vicini a $\phi(0)$ soddisfano questa condizione, quindi è necessario aggiungerne un'altra; una possibilità è la **Goldstein condition**:

$$(G) \quad \phi(\alpha) \geq \phi(0) + m_2 \alpha \phi'(0) \qquad m_1 < m_2 < 0 \tag{3.56}$$

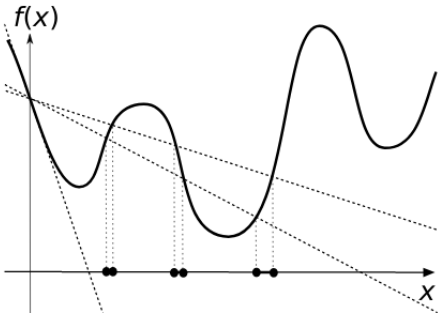


Figure 3.2: Armijo + Goldstein Conditions

ovvero graficamente stiamo considerando tutti i punti tra le due rette così determinate. Il problema è che in questo modo probabilmente escludiamo tutti i minimi locali.

Un'altra condizione possibile allora è la **Wolfe Condition**:

$$(W) \quad \phi'(\alpha) \geq m_3\phi'(0) \qquad m_1 < m_3 < 1 \tag{3.57}$$

cioè richiediamo che in α la derivata sia più vicina allo 0 di un fattore m_3 rispetto alla derivata nel punto di partenza. Dimostreremo alcuni risultati utilizzando (W) ma in realtà si usa la versione **Strong**:

$$(W') \quad |\phi'(\alpha)| \leq m_3|\phi'(0)| \qquad m_1 < m_3 < 1 \tag{3.58}$$

in modo da evitare che il valore sia $\gg 0$, ed il risultato è il seguente:

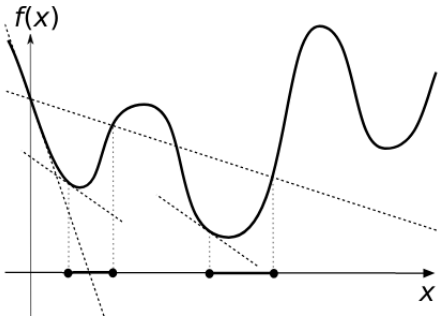


Figure 3.3: Armijo + Wolfe (Strong) Conditions

Quindi il modo di procedere sarà trovare dei punti che soddisfano le **Armijo-Wolfe Conditions**; in particolare quello che spesso si fa è iniziare con un approccio *Exact* e non aspettare che questo converga ad un minimo locale, ma fermarsi quando le *Armijo-Wolfe Conditions* sono soddisfatte. A questo punto è dunque necessario chiedersi:

- Se esistono dei punti che soddisfano queste condizioni;
- Come trovarli;
- Queste condizioni mi garantiscono la convergenza?

infatti la dimostrazione di convergenza precedente si basava proprio sul fatto che, grazie all'exact-line search, si ottenevano ad ogni iterazione dei punti stazionari, condizione che adesso non è più garantita. Vediamo nel dettaglio allora l'esistenza, la procedura per trovare tali punti e la dimostrazione di convergenza.

Esistenza

Theorem 5. *Sia $\phi(\alpha) \in C^1$, limitata inferiormente per $\alpha \geq 0$, allora $\exists \alpha$ tale che siano soddisfatte le **Armijo-Wolfe Conditions**:* $(A) \cap (W')$

Proof. Indichiamo con $l(\alpha)$ la retta determinata dalla condizione di *Armijo*, ovvero:

$$l(\alpha) = \phi(0) + m_1 \alpha \phi'(0) \tag{3.59}$$

e sia invece $d(\alpha)$ la differenza tra $l(\alpha)$ e la funzione $\phi(\alpha)$, quindi se $d(\alpha) > 0$ allora la funzione sta sotto $l(\alpha)$ ed è soddisfatta la condizione di *Armijo*. Notiamo allora che in 0 si ha:

$$d(\alpha) = l(\alpha) - \phi(\alpha) \implies \begin{cases} d(0) = 0 \\ d'(0) = (m_1 - 1)\phi'(0) > 0 \end{cases} \tag{3.60}$$

e, dal momento che abbiamo assunto $\phi(\alpha)$ limitata inferiormente, allora deve esistere almeno un punto in cui d si annulla di nuovo, altrimenti significa che la funzione va $-\infty$ come $l(\alpha)$. Indichiamo con $\bar{\alpha}$ il più piccolo tra questi, allora si ha:

$$d(\bar{\alpha}) = 0 \implies d(\alpha) > 0 \quad \forall \alpha \in (0, \bar{\alpha}] \tag{3.61}$$

cioè la condizione di *Armijo* è soddisfatta per ogni punto in questo intervallo.

Vediamo adesso se all'interno di questo intervallo esiste almeno un punto che soddisfa anche la condizione di *Wolfe*: si ha che 0 e $\bar{\alpha}$ sono due radici consecutive di $d(\alpha)$, dunque per il teorema di *Rolle*, in questo intervallo la derivata cambia segno un numero dispari di volte; quindi dato che $d'(0) > 0$, si ha che $d'(\bar{\alpha}) < 0$ e questo implica che:

$$d'(\bar{\alpha}) < 0 \implies \phi'(\alpha) > m_1 \phi'(0) > m_3 \phi'(0) \tag{3.62}$$

quindi la condizione di *Wolfe* è soddisfatta. Inoltre è soddisfatta anche la *Strong Wolfe Condition*, infatti per il teorema del valore intermedio su ϕ' si ha che:

$$\exists \alpha' \in (0, \bar{\alpha}) : \phi'(\alpha') = m_3 \phi'(0) \tag{3.63}$$

□

Come troviamo i punti che soddisfano le Armijo-Wolfe Conditions? In teoria dovremmo garantire che la scelta del parametro m_1 non escluda tutti i minimi locali, e successivamente riuscire a calcolare un intervallo $[\underline{\alpha}, \bar{\alpha}]$ all'interno del quale sono soddisfatte le *Armijo-Wolfe Conditions*; infine effettuare una *exact line search* in questo intervallo.

In pratica invece si prende un intervallo ragionevole, si setta $m_1 = 0.0001$ e si effettua una *exact line search*; ad ogni iterazione si controllano le *Armijo-Wolfe Conditions* e ci si ferma quando queste sono soddisfatte anche se non si tratta di un minimo locale.

Si può utilizzare una versione ancora più semplice, detta **Backtracking Line Search**, che parte da uno step α arbitrario e controlla solo le condizioni di *Armijo*; se sono soddisfatte bene, altrimenti ricalcola α:

Algorithm 6: Backtracking Line Search

$\alpha = \mathbf{BLS}(\phi, \phi', \alpha, m_1, \tau)$
while $\phi(\alpha) > \phi(0) + m_1 \alpha \phi'(0)$ **do**
  $\alpha \leftarrow \tau\alpha$; // $\tau < 1$
end

Convergenza

Le condizioni di *Armijo-Wolfe* sono sufficienti a garantire la convergenza? Facendo ulteriori assunzioni sì; in particolare è necessario che ∇f sia **Lipshitziana**.

L'idea è quella di dimostrare che, utilizzando queste condizioni, lo step α^i che si sceglie è sufficientemente grande; ma quest'ultimo dipende da qual è il primo punto che soddisfa la condizione di *Wolfe*: se ϕ' cambia troppo velocemente allora questo punto potrebbe essere molto vicino a quello di partenza, ma se assumiamo che ϕ' sia Lipshitziana, allora non si può avere una decrescita troppo veloce e quindi la dimensione dello step sarà maggiore. Vediamolo più formalmente:

Proof. Abbiamo dimostrato che esiste $0 < \alpha' < \bar{\alpha}$ tale che:

$$\phi'(\alpha') = m_3\phi'(0) \tag{3.64}$$

e si ha ∇f Lipshitziana $\implies$ ϕ' Lipshitziana, dunque:

$$L(\alpha' - 0) \geq \phi'(\alpha') - \phi'(0) > (1 - m_1)(-\phi'(0)) \implies \tag{3.65}$$

$$\alpha' > \frac{(1 - m_1)\left\|\nabla f(x^i)\right\|}{L} \tag{3.66}$$

dove abbiamo utilizzato $\phi'(0) = -\left\|\nabla f(x^i)\right\|$ e abbiamo sfruttato la condizione di *Armijo*. Dimostriamo allora che se ad ogni step i del nostro algoritmo valgono $(A) \cap (W)$, allora o $\{f(x^i)\} \to -\infty$ oppure $\{\left\|\nabla f(x^i)\right\|\} \to 0$.

Assumiamo $-\phi'(0) = \left\|\nabla f(x^i)\right\| \geq \epsilon > 0 \, \forall i$, allora:

$$(W) \implies \alpha^i \geq \alpha' > \frac{(1 - m_1)\left\|\nabla f(x^i)\right\|}{L} \geq \delta := \frac{(1 - m_1)\epsilon}{L} > 0 \tag{3.67}$$

$$(A) \implies f(x^{i+1}) \leq f(x^i) - m_1\alpha^i\left\|\nabla f(x^i)\right\| \leq f(x^1) - m_1\delta\epsilon \tag{3.68}$$

$$\implies \{f(x^i)\} \to -\infty \qquad (or \quad \{\left\|\nabla f(x^i)\right\|\} \to 0) \tag{3.69}$$

cioè abbiamo dimostrato che se ad ogni iterazione scegliamo uno step α in modo da soddisfare le condizioni di *Armijo-Wolfe*, allora o la funzione non è limitata, oppure (*se* la successione converge) l'algoritmo converge ad un punto stazionario. □

Questa dimostrazione ci garantisce anche che l'algoritmo termina qualunque sia il parametro ϵ che scegliamo (la norma va a 0).

Vediamo adesso che, assumendo sempre ∇f Lipshitziana, anche la **Backtracking Line Search** converge.

Proof. Assumiamo per semplicità di avere in input $\alpha = 1$

$$\left\|\nabla f(x^i)\right\| > \epsilon \quad \forall i \implies \alpha' > \delta > 0 \quad \forall i \tag{3.70}$$

$$h = \min\{k : \tau^k \leq \delta\} \implies \alpha^i \geq \tau^h > 0 \quad \forall i \tag{3.71}$$

$$\implies f(x^{i+1}) \leq f(x^i) - m_1\tau^h\epsilon \tag{3.72}$$

$$\implies \{f(x^i)\} \to -\infty \qquad (or \quad \{\left\|\nabla f(x^i)\right\|\} \to 0) \tag{3.73}$$

□

quindi *Backtracking* funziona ed è più semplice; il problema è che si deve scegliere bene l'α iniziale; se si hanno dunque delle informazioni sull'α da scegliere, può essere una buona alternativa.

3.5 "Very Inexact" Line Search: fixed stepsize

L'idea è che si può convergere anche senza effettuare la *Line-Search*, cioè fissando in anticipo il valore di α. Vediamolo più nel dettaglio: è necessario sempre assumere che ∇f sia Lipshitziana; questo ci permette di scrivere:

$$f(y) \leq Q_{x(y)} = f(x) + \nabla f(x)(y-x) + \frac{L}{2}\|y-x\|^2 \tag{3.74}$$

intuitivamente: avere ∇f Lipshitziana significa che l'Hessiana è limitata, quindi possiamo usare la costante di Lipshitz per trovare un upper-bound all'Hessiana ($L/2$). Ritornando all'algoritmo le variabili corrispondono a:

$$y := x^{i+1} \quad x := x^i \quad (y-x) := -\alpha \nabla f(x^i) \tag{3.75}$$

quindi una stima della decrescita che si ha ad ogni iterazione è:

$$f(x^{i+1}) - f(x^i) \leq \left(\frac{L\alpha^2}{2} - \alpha\right)\left\|\nabla f(x^i)\right\|^2 \tag{3.76}$$

a questo punto possiamo scegliere il valore di α minimizzando questa stima, cioè:

$$\frac{\partial}{\partial \alpha}\left(\frac{L\alpha^2}{2} - \alpha\right)\left\|\nabla f(x^i)\right\|^2 = 0 \implies \alpha_* = \frac{1}{L} \tag{3.77}$$

Andando a sostituire α_* alla fine si ottiene:

$$f(x^{i+1}) - f(x^i) \leq -\frac{\left\|\nabla f(x^i)\right\|^2}{2L} \tag{3.78}$$

cioè l'algoritmo converge in quanto la decrescita dipende dalla norma del gradiente che va 0. Quindi anche se non stiamo effettuando una *line-search*, la convergenza è comunque garantita. Tuttavia la stima che abbiamo fatto è troppo pessimistica ed in pratica, quindi, scegliere α in questo modo è molto poco efficiente (si ha una convergenza *sublinear*).

Efficienza

Dal momento che α ha un valore fissato, è più semplice fare un'analisi dell'efficienza. In breve (vedi slides per i passaggi) quello che si ottiene è:

- L'errore decresce come $O(1/i) \implies O(1/\epsilon)$ iterazioni;
- Se sfruttiamo anche la *Strong Convexity* allora si ottiene $O(1/\sqrt{\epsilon})$;
- Se invece la funzione non è *Strong-Convex* allora il meglio che si può ottenere è $O(1/\sqrt{\epsilon})$

$O(1/\epsilon)$ significa avere una convergenza *sublinear*, mentre effettuando la *line-search* avevamo ottenuto una convergenza lineare ($O(1/\log\epsilon)$). Tutte le stime sono asintotiche, quindi in pratica si deve tener conto anche delle costanti di proporzionalità che possono fare la differenza. In generale comunque è un metodo che funziona bene se non si è interessati ad avere un'elevata precisione.

Chapter 4

More than gradient methods

4.1 General descent methods

Finora abbiamo sempre assunto come direzione di decrescita quella dell'antigradiente, ovvero ad ogni iterazione i scegliamo:

$$d^i = -\frac{\nabla f(x^i)}{\|\nabla f(x^i)\|} \tag{4.1}$$

Notiamo adesso che non è davvero necessario scegliere esattamente questa direzione, infatti, immaginiamo ad esempio di fare una rotazione di $\pi/4$ rispetto alla direzione dell'antigradiente, allora l'effetto complessivo è semplicemente una moltiplicazione per una costante ($\cos \pi/4$) e, finché la decrescita va come una frazione della norma del gradiente, la convergenza è comunque garantita. Quindi diciamo che l'importante è scegliere una **direzione di decrescita**, dove quest'ultima è una qualunque direzione tale per cui:

$$\textbf{descent direction} \equiv \frac{\partial f(x^i)}{\partial d^i} < 0 \equiv \left\langle d^i, \nabla f(x^i) \right\rangle < 0 \equiv \cos\theta^i > 0 \tag{4.2}$$

dove θ^i è l'angolo formato tra d^i e $-\nabla f(x^i)$.

Convergenza

Vale il seguente risultato generale:

Theorem 6. ***(Zoutendijk)*** *Sia $f \in C^1$, ∇f Lipshitziana ed f limitata inferiormente. Allora le condizioni di Armijo-Wolfe implicano la convergenza della seguente serie:*

$$(A) + (W') \implies \sum_{i=1}^{\infty} \cos^2\theta^i \left\|\nabla f(x^i)\right\|^2 < \infty \tag{4.3}$$

quindi, a patto che la direzione sia scelta in modo che il coseno sia strettamente positivo, la norma del gradiente va a 0:

$$\cos\theta^i \geq \epsilon > 0 \implies \|\nabla f(x^i)\| \to 0 \tag{4.4}$$

In altre parole basta prendere d^i *non* ortogonale al gradiente (e ovviamente nello stesso semispazio dell'antigradiente altrimenti il coseno non è positivo). Tecnicamente anche $\cos\theta^i$ potrebbe andare a 0, l'importante è che ci vada più lentamente di $\|\nabla f(x^i)\|$.

Vediamo adesso alcuni metodi che utilizzano delle direzioni di decrescita diverse da quella dell'antigradiente.

4.2 Newton's method

Abbiamo già visto il metodo di Newton in una dimensione quando abbiamo discusso del problema della *line-search*; vediamolo adesso in generale. L'idea è quella di approssimare la funzione con il modello al secondo ordine e minimizzare quest'ultimo, quindi:

$$Q_x = f(x) + \nabla f(x)(y-x) + (y-x)^T \nabla^2 f(x)(y-x) \tag{4.5}$$
$$\nabla Q_x = \nabla f(x) + \nabla^2 f(x)(y-x) = 0 \tag{4.6}$$
$$\implies y = x - [\nabla^2 f(x)]^{-1} \nabla f(x) \tag{4.7}$$

dove abbiamo implicitamente assunto $\nabla^2 f(x) \succ 0$. Quindi la direzione d^i è:

$$d^i = -[\nabla^2 f(x^i)]^{-1} \nabla f(x^i) \tag{4.8}$$

e lo step α è uguale a $\alpha = 1$, quindi non abbiamo nemmeno bisogno di effettuare la *line-search*. La direzione così scelta è una direzione di decrescita, infatti:

$$\left\langle \nabla f(x^i), d^i \right\rangle = -[\nabla f(x^i)]^T [\nabla^2 f(x^i)]^{-1} \nabla f(x^i) < 0 \tag{4.9}$$

perché l'Hessiana è definita positiva (strettamente).

Abbiamo già visto che, assumendo $f \in C^3$, $\nabla f(x_*) = 0$ e $\nabla^2 f(x_*) \succ 0$ allora:

$$\implies \exists \mathcal{B}(x_*, r) \text{ s.t. } x^1 \in \mathcal{B} \implies \{x^i\} \to x_* \text{ quadratically} \tag{4.10}$$

ovvero la **convergenza** è **quadratica**, ma **locale**, cioè solo se il punto di partenza è abbastanza vicino alla soluzione ottima x_*. Dunque è necessario "globalizzare" il metodo, cioè vedere sotto quali condizioni converge anche se il punto iniziale non è sufficientemente vicino alla soluzione. Prima di questo però vediamo un'interpretazione geometrica del metodo di Newton.

4.2.1 Interpretazione geometrica del Metodo di Newton

Il metodo di Newton può essere visto come il metodo del gradiente ma in uno "spazio diverso", ovvero dopo aver applicato una trasformazione alle variabili. Per vederlo meglio facciamo l'esempio per una funzione quadratica:

$$f(x) = \frac{1}{2} x^T Q x + qx \tag{4.11}$$

Abbiamo visto che l'efficienza del metodo del *gradiente* per una funzione quadratica dipende da quanto differiscono il minimo ed il massimo autovalore dell'Hessiana; se questi sono uguali (level sets tondi) allora l'algoritmo converge in una sola iterazione. Il metodo di Newton, invece, termina *sempre* in una sola iterazione, infatti lo step di Newton è:

$$d_x = -[\nabla^2 f(x)]^{-1} \nabla f(x) \tag{4.12}$$
$$= -Q^{-1}(Qx + q) = -x - Q^{-1} q \tag{4.13}$$

quindi il punto successivo:

$$x' = x + d_x = x - x - Q^{-1} q = -Q^{-1} q \tag{4.14}$$
$$\implies \nabla f(x') = -QQ^{-1} q + q = 0 \tag{4.15}$$

è esattamente un punto stazionario. Quello che si può fare è allora applicare una trasformazione ad x in modo da ottenere una nuova funzione quadratica la cui Hessiana abbia massimo e minimo autovalore uguali, in modo che anche il metodo del gradiente termini in

una sola iterazione. Si può dimostrare che per una funzione quadratica questa trasformazione è proprio la radice quadrata di Q:

$$Q \succeq 0 \implies Q = RR, R = Q^{1/2} \tag{4.16}$$

che è simmetrica e si può scrivere come:

$$Q = H\Lambda H^T \implies R = H\sqrt{\Lambda}H^T \tag{4.17}$$

Quindi applicando R ad x otteniamo $y = Rx \equiv x = R^{-1}y$. Riscriviamo f in funzione di y:

$$f(y) = \frac{1}{2}(R^{-1}y)^T Q(R^{-1}y) + q(R^{-1}y) \tag{4.18}$$

$$= \frac{1}{2}y^T R^{-1} Q R^{-1} y + qR^{-1}y \tag{4.19}$$

$$= \frac{1}{2}y^T R^{-1} RRR^{-1} y + qR^{-1}y \tag{4.20}$$

$$= \frac{1}{2}y^T Iy + qR^{-1}y \tag{4.21}$$

la direzione calcolata dal metodo del gradiente è allora:

$$d_y = -\nabla f(y) = -y - R^{-1}q \tag{4.22}$$

dunque il punto calcolato all'iterazione successiva è:

$$y' = y + d_y = -R^{-1}q \implies \nabla f(y') = 0 \tag{4.23}$$

che è un punto stazionario, quindi il metodo del gradiente termina in una sola iterazione. Quindi i due metodi sono in questo senso equivalenti ed un'ulteriore conferma si ha riapplicando la trasformazione inversa alla direzione d_y appena calcolata:

$$R^{-1}d_y = R^{-1}(-y - R^{-1}y) = -x - Q^{-1}q = d_x \tag{4.24}$$

ottenendo così proprio lo step di Newton. Possiamo sintetizzare tutto questo discorso dicendo: *Newton = Gradient in uno spazio in cui l'Hessiana della funziona è la matrice identità.* Il problema è che se la funzione non è quadratica, trovare la trasformazione giusta non è banale, dunque gli algoritmi che vedremo adesso si sviluppano su questa idea ma si limitano a cercare trasformazioni che rendano l'Hessiana il più "simile" possibile all'identità, ottenendo in ogni caso una convergenza più veloce del metodo del gradiente.

4.2.2 Convergenza globale del metodo di Newton

Abbiamo detto che una condizione sufficiente per la convergenza è:

$$\cos\theta^i = \frac{d^i \nabla f(x^i)}{\|d^i\|\|\nabla f(x^i)\|} \leq \delta < 0 \tag{4.25}$$

Assumendo che il gradiente sia Lipshitziano e la funzione si strettamente convessa, cioè:

$$uI \preceq \nabla^2 f \preceq LI \tag{4.26}$$

allora il metodo di Newton converge, infatti:

$$\nabla^2 f(x^i)d^i = -\nabla f(x^i) \implies \tag{4.27}$$

$$d^i \nabla f(x^i) = -(d^i)^T \nabla^2 f(x^i) d^i \leq -\lambda^n \|d^i\|^2 \tag{4.28}$$

$$\|\nabla f(x^i)\| = \|\nabla^2 f(x^i) d^i\| \leq \|\nabla^2 f(x^i)\|\|d^i\| = \lambda^1 \|d^i\| \tag{4.29}$$

$$\implies cos(\theta^i) \leq -\frac{\lambda^n}{\lambda^1} \leq -\frac{u}{L} \implies \textit{global convergence} \tag{4.30}$$

Inoltre la convergenza è veloce (si può dimostrare che è *superlinear*), tuttavia l'assunzione che abbiamo fatto è forte (soprattutto il fatto che la funzione è convessa). Tutte queste dimostrazioni richiedono che:

$$m_1 \leq \frac{1}{2} \tag{4.31}$$

(quando $m_1 > 1/2$ i minimi di una funzione quandratica potrebbero essere "tagliati"). In realtà possiamo dimostrare non solo che converge ma anche che da una certa iterazione in poi, lo step unitario soddisfa sempre le *Armijo condition*, infatti si ha:

$$f(x^i + d^i) = f(x^i) + d^i \nabla f(x^i) + \frac{1}{2} d^i \nabla^2 f(x^i) d^i + R(\|d^i\|) \tag{4.32}$$

$$= f(x^i) - \frac{1}{2} d^i \nabla^2 f(x^i) d^i + + R(\|d^i\|) \tag{4.33}$$

$$= \underbrace{f(x^i) + \frac{1}{2} \left\langle \nabla f(x^i), d^i \right\rangle}_{\text{Armijo with } m_1 = 1/2} + R(\|d^i\|) \tag{4.34}$$

quando il gradiente tende a 0 allora anche d^i e quindi R tendono a 0 e si ha che le condizioni di *Armijo* sono soddisfatte. Le condizioni di *Wolfe* invece non è detto che siano soddisfatte, per questo motivo spesso si usa la *Backtrackin line search* che richiede solo *Armijo*, e si ha che con $\bar{\alpha} = 1$ alla fine si avrà una convergenza quadratica.

4.2.3 Metodo di Newton: nonconvex case

Tutte le dimostrazioni valgono se l'Hessiana è convessa, tuttavia non è necessario avere esattamente l'Hessiana, l'importante è avere una funzione che abbia un andamento sufficientemente vicino a quello dell'Hessiana, in questo modo lo step di Newton sarà:

$$d^i = -H^i f(x^i) + (A) \cap (W') \implies \text{global convergence if} \quad uI \preceq H^i \preceq LI \tag{4.35}$$

$$H^i \approx \nabla^2 f(x^i) \implies \text{local quadratic and global superlinear} \tag{4.36}$$

se prendiamo $H = I$ allora è precisamente il metodo del gradiente (che non è veloce), ma se invece cerchiamo di prendere $H \approx \nabla^2 f$ allora si ottiene alla fine una convergenza quadratica.

Tutto questo è utile soprattutto quando l'Hessiana non è convessa, in questo caso possiamo prendere H^i come:

$$H^i = \nabla^2 f(x^i) + \epsilon^i I \succ 0 \tag{4.37}$$

dove $\epsilon' > -\lambda^n$ (λ^n è il più piccolo autovalore minore di 0). In questo modo, sommando un multiplo dell'identità all'Hessiana, aggiungiamo un fattore ϵ^i a tutti i suoi autovalori. In realtà vogliamo che l'Hessiana non sia solo convessa ma deve essere maggiore di 0 di un certo fattore δ, quindi spesso si sceglie ϵ come:

$$\epsilon = \max\{0, \delta - \lambda^n\} \tag{4.38}$$

in questo modo il minimo autovalore di H^i sarà δ.

Sostanzialmente dobbiamo risolvere il seguente problema di ottimizzazione vincolato:

$$\min\{\|H - \nabla^2 f(x^i)\|, H \succeq \delta I\} \tag{4.39}$$

ma è semplice perché abbiamo visto che basta calcolare il minimo autovalore dell'Hessiana e sommare un multiplo dell'identità. Nel caso in cui la norma non sia quella derivata da quella euclidea ma ad esempio sia $\|\cdot\|_F$, allora si può risolvere calcolando la decomposizione spettrale della matrice e modificando solo gli autovalori negativi. In entrambi i casi la cosa importante

è che questo processo in automatico non comporta modificazioni se l'Hessiana è già definita positiva (semplicemente l'ϵ così calcolato si annulla), quindi si ha una convergenza quadratica alla fine (diventa metodo di Newton).

Qualunque procedimento si scelga, il costo alla fine è sempre $O(n^3)$ (calcolo λ^n+ Cholesky factorization), quindi va bene solo se n è piccolo.

Invece di modificare l'Hessiana potremmo usare un approccio diverso:

Trust Region se l'Hessiana non è definita positiva, allora esiste una direzione di curvatura negativa lungo la quale la funzione decresce; allora perché non sceglierla? Il problema è che il modello al secondo ordine non è limitato lungo questa direzione, ma possiamo restringerci ad un insieme *compatto* e minimazzarlo su quello. Quindi sia $Q_{x^i}(y)$ il modello al secondo ordine e $\mathcal{T}^i$ un insieme compatto, allora possiamo scegliere il punto dell'iterazione successiva come:

$$x^{i+1} \in argmin\{Q_{x^i}(y) : y \in \mathcal{T}^i\} \tag{4.40}$$

cioè risolvere un altro problema di ottimizzazione vincolato. Questo problema è in generale *NP-Hard*, ma si può risolvere efficientemente se si sceglie $\mathcal{T} = \{x \in \mathbb{R}^n : \|x - x^i\|_2 \leq r\}$ (in pratica procedere in questo modo è quasi equivalente a quello che abbiamo fatto prima→vedi slides).

4.3 Quasi-Newton methods

Vediamo adesso metodi simili al metodo di Newton senza però il problema di dover invertire ad ogni iterazione una matrice molto grande, motivo per cui Newton è troppo costoso.

Nei Quasi-Newton methods l'idea è quella di procedere come nel metodo di Newton classico, ma senza usare le informazioni al secondo ordine, quindi l'hessiana. Dunque ad ogni iterazione, si costruisce una sorta di modello al secondo ordine:

$$m^i(x) = \nabla f(x^i)(x - x^i) + \frac{1}{2}(x - x^i)^T H^i (x - x^i) \tag{4.41}$$

$$x^{i+1} = x^i + \alpha^i d^i \tag{4.42}$$

dove appunto al posto dell'Hessiana compare una sua approssimazione H. All'iterazione successiva analogamente si ha:

$$m^{i+1}(x) = \nabla f(x^{i+1})(x - x^{i+1}) + \frac{1}{2}(x - x^{i+1})^T H^{i+1} (x - x^{i+1}) \tag{4.43}$$

Deve essere possibile scrivere la matrice H^{i+1} a partire solo dalle informazioni al primo ordine. Ma prima di vedere come costruire H^{i+1}, discutiamo delle proprietà che vogliamo che abbia:

- $H^{i+1} \succ 0$
- $\nabla m^{i+1}(x^i) = \nabla f(x^i)$
- $\|H^{i+1} - H^i\|$ "small"

La seconda proprietà ci dice che il modello che stiamo costruendo non è completamente "senza memoria", nel senso che non solo è corretto all'iterazione corrente, ma mantiene anche delle informazioni sull'iterazione precedente; questa proprietà può essere riscritta come:

$$(S) \quad H^{i+1}(x^{i+1} - x^i) = \nabla f(x^{i+1}) - \nabla f(x^i) \tag{4.44}$$

che prende il nome di *secant equation* (S).

Usiamo la seguente notazione per semplificare le formule:

$$s^i = x^{i+1} - x^i = \alpha^i d^i \qquad y^i = \nabla f(x^{i+1}) - \nabla f(x^i) \tag{4.45}$$

Quindi la secant equation si riscrive come:

$$H^{i+1}s^i = y^i \implies s^i y^i = (s^i)H^{i+1}s^i \tag{4.46}$$

Dal momento che richiediamo che la matrice H sia definita positiva, l'ultimo membro a destra è sicuramente positivo; la secant equation implica che questo sia uguale a $s^i y^i$ quindi si deve avere:

$$s^i y^i > 0 \qquad \text{Curvature Condition (C)} \tag{4.47}$$

tuttavia non è scontato che $s^i y^i$ sia positivo, perché il suo valore dipende dai dati; la notizia positiva è che s^i dipende da come scegliamo α^i quindi una scelta appropriata potrebbe far funzionare tutto, ed in effetti siamo fortunati perché si ha che $(W) \implies (C)$:

$$\phi'(\alpha) = \nabla f(x^{i+1})d^i \geq m_3\phi'(0) = m_3\nabla f(x^i)d^i \implies \tag{4.48}$$

$$(\nabla f(x^{i+1}) - \nabla f(x^i))d^i \geq (m_3 - 1)\phi'(0) > 0 \tag{4.49}$$

Quindi se facciamo una line-search con le condizioni di Armijo-Wolfe allora (C) può essere sempre soddisfatta.

Veniamo adesso alla terza proprietà che vogliamo che abbia H, ovvero $\|H^{i+1} - H^i\|$ "small"; questa equivale a risolvere un ulteriore problema di ottimizzazione:

$$H^{i+1} = argmin\{\|H - H^i\| : (S), H \succeq 0\} \tag{4.50}$$

con una scelta appropriata della norma matriciale. Indiachiamo con $\rho^i = 1/y^i s^i > 0$, allora esiste una formula chiusa per costruire H^{i+1} a partire da H^i che prende il nome di **Davidon-Fletcher-Powell** formula:

$$(DFP) \quad H^{i+1} = (I - \rho^i y^i (s^i)^T)H^i(I - \rho^i s^i (y^i)^T) + \rho^i y^i (y^i)^T \tag{4.51}$$

Ma in realtà a noi serve la matrice inversa ad ogni step, quindi invece di ricavare H^{i+1} e poi invertirla, vogliamo ricavare direttamente $(H^{i+1})^{-1}$ (che indichiamo nel seguito con B^{i+1}) a partire da $(H^i)^{-1}$; questo si può fare utilizzando la **Sherman-Morrison-Woodbury** formula:

$$(SMW) \quad [A + ab^T]^{-1} = A^{-1} - \frac{A^{-1}ab^TA^{-1}}{1 - b^TA^{-1}a} \tag{4.52}$$

$$\implies (DFP^{-1}) \quad B^{i+1} = B^i + \frac{\rho^i s^i (s^i)^T - B^i y^i (y^i)^T B^i}{(y^i)^T B^i y^i} \tag{4.53}$$

In questo modo il costo è solo quello di un prodotto matrice-vettore $O(n^2)$ e non è necessario invertire nessuna matrice.

BFGS formula

Esiste un altro modo di procedere andando a scrivere la secant equation direttamente per la matrice inversa: si ottiene così la **BFGS** formula:

$$(S) \text{ for } B^{i+1} : s^i = B^{i+1}y^i \implies B^{i+1} = argmin\{\|B - B^i\| : \dots\} \tag{4.54}$$

tutto il ragionamento è simmetrico scambiando $B \leftrightarrow H$ e $s \leftrightarrow y$. Si ottiengono quindi le **Broyden-Fletcher-Goldfarb-Shanno** formule:

$$(BFGS) \quad H^{i+1} = H^i + \rho^i y^i (y^i)^T - \frac{H^i s^i (s^i)^T H^i}{(s^i)^T H^i s^i} \tag{4.55}$$

$$(BFGS) \quad B^{i+1} = (I - \rho^i s^i (y^i)^T) B^i (I - \rho^i y^i (s^i)^T) + \rho^i s^i (s^i)^T \tag{4.56}$$

$$= B^i + \rho^i [(1 + \rho^i (y^i)^T B^i y^i) s^i (s^i) T - (B^i y^i (s^i)^T + s^i (y^i)^T B^i)] \tag{4.57}$$

In pratica si usa questa perché è numericamente più stabile della (DFP). (In realtà si può usare una qualunque combinazione convessa di (DFP) e (BFGS) e si possono ottenere anche risultati migliori, ma bisogna scegliere per bene i parametri della combinazione e diventa tutto più complicato).

A questo punto l'ultima cosa che rimane da fare è scegliere B^1 per la prima iterazione, in che modo? ci sono diverse alternative tra cui iniziare con la matrice identità I, oppure con un multiplo di questa δI, oppure ancora cercando di approssimare per differenze finite $[\nabla^2 f(x^1)]^{-1}$.

Limited-memory BFGS

Quando le dimensioni del problema sono molto elevate, ci potrebbero essere problemi di memoria, infatti la matrice B^{i+1} contiene le informazioni di tutte le iterazioni precedenti. Si può allora settare un parametro k e costruire B^{i+1} considerando solo le ultime k iterazioni; esiste una formula chiusa per farlo (vedi slides) ed il costo per iterazione diventa $O(kn)$ (k small $\approx$ gradient, k large $\approx$ Newton).

4.4 Metodi del Gradiente Coniugato

Abbiamo visto che *Quasi-Newton* utilizza una matrice $n \times n$, quindi, in termini di memoria, è come se usasse l'informazione di n gradienti; dunque se le iterazioni sono meno di n allora stiamo usando più memoria di quanta sia davvero necessaria. Nella versione limited memory invece ci si limita a considerare gli ultimi k gradienti; ci chiediamo adesso cosa succede se usiamo le informazioni di solo due gradienti. Potremmo settare $k = 2$ nella limited-memory *BFGS* oppure usare il *metodo del gradiente coniugato*.

La classe degli algoritmi *conjugate gradient* infatti usa solo due gradienti, quello corrente e quello dell'iterazione precedente. L'idea è di trovare ad ogni iterazione una direzione d^i che sia ortogonale non solo alla direzione d^{i-1} precedente (come avviene quando si fa una *exact line-search* nel metodo del gradiente), ma anche a tutte le d^j con $j < i$. Equivalentemente possiamo dire di voler trovare ad ogni iterazione il minimo della funzione nel sottospazio generato dalle direzioni precedenti $\{d^1, d^2, \dots d^i\}$. Vediamo adesso prima il caso quadratico e poi quello generale.

4.4.1 Gradiente Coniugato per funzioni quadratiche

Innanzitutto ricordiamo che due direzioni sono dette *Q-Conjugate* se $(d^i)^T Q d^j = 0$. Non lo dimostriamo ma, nel caso di una funzione quadratica, si ha che per trovare una direzione ortogonale a tutte quelle precedenti, è sufficiente calcolarla in questo modo:

$$d^i = -\nabla f(x^i) + \beta^i d^{i-1} \tag{4.58}$$

ovvero prendiamo la direzione dell'antigradiente "deflessa" di $\beta^i d^{i-1}$; esiste una formula chiusa per β che si calcola in $O(n)$:

$$\beta^i = \frac{\nabla f(x^i)^T Q d^{i-1}}{(d^{i-1})^T Q d^{i-1}} \tag{4.59}$$

Successivamente lo step α_i viene calcolato esattamente come per il metodo del gradiente per funzioni quadratiche, cioè minimizzando la funzione lungo una direzione scelta. La formula chiusa per α^i è sempre:

$$\alpha^i = \frac{\|\nabla f(x^i)\|^2}{(d^i)^T Q d^i} \tag{4.60}$$

A questo punto possiamo scrivere la procedura per le funzioni quadratiche:

Algorithm 7: $x = CGQ(Q, q, x, \epsilon)$

$d^- \leftarrow 0$;
while $\|\nabla f(x)\| > \epsilon$ **do**
  if $d^- = 0$ **then** $d \leftarrow -\nabla f(x)$;
  else $\beta = \frac{\nabla f(x)^T Q d^-}{(d^-)^T Q d^-}$; $d \leftarrow -\nabla f(x) + \beta d^-$;
  $\alpha = -\frac{\nabla f(x)^T d}{(d)^T Q d}$;
  $x \leftarrow x + \alpha d$; $d^- \leftarrow d$;
end

Andando a sostituire la forma esplicita del gradiente per una funzione quadratica, otteniamo:

$$\beta^i = \frac{\|\nabla f(x^i)\|^2}{\|\nabla f(x^{i-1})\|^2}, \quad \alpha^i = \frac{\|\nabla f(x^i)\|^2}{(d^i)^T Q d^i} \tag{4.61}$$

Inoltre il problema si risolve in al massimo n iterazioni (in aritmetica esatta) infatti intuitivamente ad ogni step aggiungiamo una dimensione al sottospazio rispetto a cui abbiamo trovato il minimo, quindi arrivati ad n abbiamo ottimizzato su tutto lo spazio.

Questo metodo si può migliorare molto andando a vedere come sono clusterizzati gli autovalori ecc..

4.4.2 Gradiente Coniugato per funzioni non quadratiche

Se la funzione non è quadratica allora tipicamente non riusciamo a fare un'*exact line-search*, quindi α_i deve essere calcolato con metodi inesatti, come ad esempio *Armijo-Wolfe*. La differenza sostaziale rispetto al caso quadratico è che adesso beta può essere calcolato in più modi, vediamone alcuni:

- **Flercher-Reeves**: $\beta = \frac{\|\nabla f(x^i)\|^2}{\|\nabla f(x^{i-1})\|^2}$
- **Polak-Ribiere**: $\beta = \frac{\|\nabla f(x^i)\|^T (\nabla f(x^i) - \nabla f(x^{i-1}))}{\|\nabla f(x^{i-1})\|^2}$
- **Hestenes-Stiefel**: $\beta = \frac{\|\nabla f(x^i)\|^T (\nabla f(x^i) - \nabla f(x^{i-1}))}{(\nabla f(x^i) - \nabla f(x^{i-1}))^T d^{i-1}}$
- **Dai-Yuan**: $\beta = \frac{\|\nabla f(x^i)\|^2}{(\nabla f(x^i) - \nabla f(x^{i-1}))^T d^{i-1}}$

Se la funzione è quadratica tutte queste formule ci forniscono lo stesso risultato.

Possiamo allora scrivere l'algoritmo nella versione *Flercher-Reeves* e con *Armijo-Wolfe line-search* come:

Convergenza

La dimostrazione di convergenza non è banale anche perchè dipende fortemente da beta. Mostriamo solo alcuni risultati senza dimostrazione:

- La versione *Fletcher-Reeves* richiede $m_1 < m_2 < 1/2$ affinche *Armijo Wolfe* funzioni;

Algorithm 8: Fletcher-Reeves version

procedure $x = CGQ(f, x, \epsilon)$
$d^- \leftarrow 0$;
$\nabla f^- = 0$;
while $\|\nabla f(x)\| > \epsilon$ **do**
  if $\nabla f^- = 0$ **then** $d \leftarrow -\nabla f(x)$;
  else $\beta = \frac{\|\nabla f(x)\|^2}{\|\nabla f^-\|^2}$; $d \leftarrow -\nabla f(x) + \beta d^-$;
  $\alpha \leftarrow AWLS(f(x + \alpha d))$;
  $x \leftarrow x + \alpha d$; $d^- \leftarrow d$; $\nabla f^- \leftarrow \nabla f(x)$;
end

- Nella versione *Polak-Ribiere* le condizioni di *Armijo Wolfe* non garantiscono che la direzione ottenuta sia di decrescita; in questo caso si potrebbero ottenere $\beta^i < 0$ ma se ad ogni iterazione di sceglie $\beta_{PR} = \max\{0, \beta^i\}$ allora si ha la convergenza. Sostanzialmente quando si ottiene un $\beta^i < 0$ allora si fa una sorta di *restart*, cioè per un'iterazione si procede come nel metodo del gradiente classico. Inoltre questa versione potrebbe non convergere per alcuni tipi di funzioni.
- Il *restart* certe volte è una buona idea anche nella versione *Fletcher-Reeves*, infatti spesso succede: $\|\nabla f(x^i)\| \ll \|d^i\| \iff \cos(\theta^i) \approx 0 \equiv \nabla f(x^i) \approx\perp d^i$ e questo implica $x^{i+1} \approx x^i$, ovvero si iniziano a fare degli step molto piccoli. Quindi ogni tot iterazioni potrebbe essere utile fare un *restart*.

Efficienza

L'efficienza è *n-step quadratica*, ovvero:

$$\|x^{i+n} - x_*\| \leq R\|x^i - x_*\|^2 \quad \equiv \text{n CG steps} \approx \text{1 Newton step} \tag{4.62}$$

Intuitivamente quando siamo vicini alla soluzione ottima x_* l'Hessiana è più o meno fissata perché $\nabla^2 f(x^i) \approx \nabla^2 f(x_*)$ e quindi in n steps CG risolve esattamente un sistema lineare. Se n è molto grande quindi l'efficienza non è buona.

Non sempre questi metodi funzionano meglio del metodo del gradiente classico, e soprattutto l'andamento dipende molto dalla funzione e dai parametri che si scelgono.

4.5 Deflected gradient methods

L'idea è la stessa dei metodi *Conjugate Gradient* ma ancora più semplificata. Quello che si fa è aggiungere un termine detto *momentum* che evita il classico andamento a *zig-zag* del metodo del gradiente:

$$x^{i+1} = x^i + \alpha^i \nabla f(x^i) + \beta^i (x^i - x^{i+1}) \tag{4.63}$$

La differenza rispetto al *Conjugate Gradient* è che in questo caso manteniamo comunque come direzione quella dell'antigradiente, aggiungendo però un termine direttamente nell'equazione che determina il punto successivo. La cosa importante da notare è che in questo modo non si ha un algoritmo monotono (*descent algorithm*), cioè non è detto che ad ogni iterazione il valore della funzione nel punto decresca. Questo implica che le dimostrazioni di convergenza per questa classe di algoritmi siano molto complicate e spesso non ci sono sufficienti risultati teorici.

Tuttavia nel caso in cui la funzione sia strettamente convessa ed il gradiente sia *Lipshitziano*, ovvero:

$$uI \preceq \nabla^2 f(x) \preceq LI \qquad u = \lambda^n, L = \lambda^1 \tag{4.64}$$

dove λ^n e λ^1 sono il minimo ed il massimo autovalore dell'Hessiana, allora è possibile stimare dei valori ottimi (nella peggiore delle condizioni) per α^i e β^i:

$$\alpha = \frac{4}{(\sqrt{\lambda^1} + \sqrt{\lambda^n})^2} \qquad \beta = \max\{|1 - \sqrt{\alpha\lambda^n}|, |1 - \sqrt{\alpha\lambda^1}|\}^2 \tag{4.65}$$

Con questa scelta si ottiene:

$$\|x^{i+1} - x_*\| \leq \left(\frac{\sqrt{\lambda^1} - \sqrt{\lambda^n}}{\sqrt{\lambda^1} + \sqrt{\lambda^n}}\right) \|x^i - x_*\| \tag{4.66}$$

che è un risultato simile a quello ottenuto per il metodo del gradiente classico, ma con la differenza sostanziale che gli autovalori sono presenti sotto radice, quindi si ha un'efficienza maggiore.

Notiamo che sono dei valori *fissati*, quindi non è necessario calcolarli ad ogni iterazione; il problema è che dipendono da λ^1 e λ^n che tipicamente non conosciamo. Nella pratica, quindi, quello che si fa è ad esempio trovare α^i tramite una *line-search* e trattare β^i come un parametro algoritmico; oppure trattare entrambi α^i e β^i come parametri algoritmici.

4.5.1 Accelerated Gradients

Dunque per i metodi *Deflected Gradients* è difficile ottenere dei risultati teorici forti; tuttavia esistono delle varianti per le quali invece questi risultati sono stati già dimostrati; una di queste varianti e quella degli *Accelerated Gradients Methods*.

Chapter 5

Less than gradient methods

Vediamo adesso degli approcci che non utilizzano le informazioni fornite dal gradiente, ad esempio perché la funzione non è differenziabile oppure perché calcolarlo costa troppo.

Motivazione: in ML tipicamente vogliamo minimizzare la *loss function*:

$$\min \left\{ \sum_{i \in I} I(y^i, \langle \phi(X^i), w \rangle) : w \in \mathbb{R}^n \right\} \tag{5.1}$$

dove le X^i sono le osservazioni, w i pesi e y^i gli output delle osservazioni. La loss function più semplice è ad esempio $I(y, Aw) = -(y - Aw)^2/2$. Il gradiente si calcola come:

$$\nabla f(w) = \sum_{i \in I} \nabla f^i(w) = \sum_{i \in I} -A^i(y^i - A^i w) \tag{5.2}$$

quindi è una somma di m termini dove m è il numero di osservazioni; ma questo numero in genere è estremamente elevato, quindi calcolare una somma così grande ad ogni iterazione diventa troppo costoso.

5.1 Incremental Gradient methods

Un'idea per superare questo problema è assumere che il processo sia stocastico e selezionare solo un piccolo campione per calcolare il gradiente, cioè selezioniamo un sottoinsieme K e

$$-d^i = \nabla f^K(w) = \sum_{i \in K} \nabla f^i(w) \tag{5.3}$$

Il gradiente calcolato in questo modo viene detto *incremental gradient*; notiamo che in questo modo non abbiamo la certezza di ottenere una direzione di decrescita e inoltre bisogna stabilire come scegliere K e $|K|$. Il modo migliore di procedere è scegliere K in modo random (*stochastic gradient*). Tipicamente sono necessarie molte più iterazioni rispetto a quelle effettuate dal metodo del gradiente, ma queste sono molto meno costose.

Questo è solo un esempio di una tipica sitauzione in cui non è possibile sfruttare l'informazione al primo ordine e quindi tutti gli algoritmi già visti non sono applicabili. Vediamo adesso più in particolare il caso in cui la funzione da minimizzare non sia differenziabile.

5.2 Subgradients and subdifferential

Se la funzione non è differenziabile allora il vero problema dell'utilizzo di algoritmi come lo *Steepest Descent* sta nella line-search; infatti nei punti in cui la funzione non è differenziabile non esiste un unico gradiente ma molti *subgradienti* e alcuni di questi non determinano

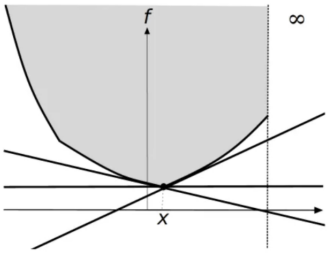


Figure 5.1: Subgradienti

direzioni di decrescita. Ciascuna delle rette in figura rappresenta un subgradiente; nei punti in cui la funzione è differenziabile invece si ha un solo subgradiente che corrisponde al gradiente classico. In particolare un problema che può capitare è che proprio il minimo si trovi in un punto in cui la funzione non è differenziabile; in questo caso uno dei subgradienti sarà nullo, ma l'algoritmo potrebbe fornirci un altro subgradiente e quindi non siamo in grado di riconoscere che il punto in questione è un minimo.

Chiamiamo *Subdifferenziale* $\partial f(x)$ l'insieme di tutti i subgradienti in un punto; si ha che

- $\partial f(x)$ è un insieme chiuso e convesso se $x \in intdom(f)$.
- Una direzione d è di decrescita se $\langle d, s\rangle < 0$ per ogni subgradiente $s \in \partial f(x)$.
- Il subgradiente di norma minima è la direzione di decrescita massima:

$$s_* = -argmin\{\|s\| : s \in \partial f(x)\} \implies \text{Steepest Descent direction} \tag{5.4}$$

In teoria si potrebbe in certi casi calcolare l'insieme completo dei subgradienti in modo poi da ottenere quello di norma minima, ma in pratica è molto complicato.

Abbiamo però già visto che ci sono metodi che convergono pur non scegliendo ad ogni iterazione direzioni di decrescita. Inoltre se la funzione è convessa allora tutti i subgradienti godono di una proprietà molto importante: se ci muoviamo lungo la direzione opposta ad uno dei subgradienti, anche se questa non è una direzione di decrescita, facendo un passo sufficientemente piccolo ci avviciniamo comunque ad x_*. Questa proprietà si vede chiaramente se facciamo un grafico delle curve di livello di una funzione convessa e non differenziabile:

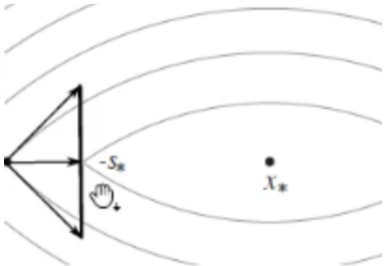


Figure 5.2: Curve di livello di una funzione non differenziabile
7

Tutte le direzioni intermedie riportate in figura rappresentano i $-s$ possibili, alcune di queste sono di decrescita, altre invece (quelle agli estremi) no; tuttavia anche nei casi in cui la direzione non è di decrescita, se il passo è abbastanza piccolo, si vede che ci muoviamo verso x_*.

5.3 Subgradient methods

Vediamo adesso algebricamente la proprietà detta sopra: dal momento che $f(x)$ è convessa si ha

$$f(x_*) \geq f(x) + \langle g, x_* - x \rangle \implies \langle g, x_* - x \rangle \leq f(x_*) - f(x) \leq 0 \tag{5.5}$$

Riscriviamo dunque la distanza tra il punto all'iterazione $i+1$ ed x_* come

$$\begin{aligned} \|x^{i+1} - x_*\|^2 &= \|x^i - \alpha^i g^i - x_*\|^2 && (5.6)\\ &= \|x^i - x_*\|^2 + 2\alpha^i \langle g^i, x_* - x^i \rangle + (\alpha^i)^2 \|g^i\|^2 && (5.7)\\ &\leq \|x^i - x_*\|^2 + 2\alpha(f(x_*) - f(x^i)) + (\alpha^i)^2 \|g^i\|^2 && (5.8) \end{aligned}$$

Il termine lineare in α è negativo mentre quello quadratico positivo, dunque per α sufficientemente piccolo il termine lineare domina e si ha che la distanza da x_* all'iterazione $i+1$ è minore di quella all'iterazione i. Riscriviamo adesso questa relazione considerando come direzione $d^i = -g^i/\|g^i\|$:

$$\begin{aligned} \|x^{i+1} - x_*\|^2 &\leq \|x^i - x_*\|^2 + 2\alpha \frac{f(x_*) - f(x^i)}{\|g^i\|^2} + (\alpha^i)^2 && (5.9)\\ &\leq \|x^i - x_*\|^2 + (\alpha^i)^2 && (5.10)\\ &\leq \|x^{i-1} - x_*\|^2 + (\alpha^i)^2 + (\alpha^{i-1})^2 && (5.11)\\ &\leq \|x^1 - x_*\|^2 + \sum_{k=1}^{i} (\alpha^k)^2 && (5.12) \end{aligned}$$

Se facciamo il limite per $i \to \infty$ vogliamo che la serie degli $(\alpha^i)^2$ converga ma non troppo velocemente perché altrimenti l'algoritmo non funziona bene. Notiamo inoltre che il fatto che questa serie converga implica che la successione $\{x^i\}$ è limitata. Una scelta possibile è allora una successione di α^i tale che la corrispendente serie diverga ma quella dei quadrati converga, cioè

$$\sum_{i=1}^{\infty} \alpha^i = \infty \quad \wedge \quad \sum_{i=1}^{\infty} (\alpha^i)^2 < \infty \tag{5.13}$$

Ad esempio $\alpha^i = 1/i$ ha esattamente questa proprietà (*diminishing-square-summable*). Se vale questa proprietà allora l'algoritmo converge, anche se la convergenza è debole cioè:

$$\liminf_{i \to \infty} f(x^i) = f(x_*) \tag{5.14}$$

Proof. Se per assurdo $f(x^i) - f(x_*) \geq \epsilon > 0$ per ogni i allora si avrebbe

$$\|x^{i+1} - x_*\| \leq \|x^1 - x_*\|^2 - \delta \sum_{k=1}^{i} \alpha^i + \sum_{k=1}^{i} (\alpha^i)^2 \tag{5.15}$$

dove $\delta > 0$ e non va a 0 in quanto $f(x^i) - f(x_*) \geq \epsilon$ e $\|g^i\|$ non può andare ad infinito perché una proprietà delle funzioni convesse è che se $\{x^i\}$ è limitata allora lo è anche $\{g^i\}$. Dunque se prendiamo il limite per $i \to \infty$ allora il membro a destra tenderebbe a $-\infty$. □

Quindi procedendo in questo modo la convergenza è garantita, tuttavia in pratica è un metodo molto poco efficiente.

Polyak step-size

Supponiamo adesso di conoscere il valore di $f(x_*)$; in questo caso possiamo calcolare esattamente l'α che minimizza il membro destro della (1.9), cioè $\alpha^i = \frac{f(x^i)-f(x_*)}{\|g^i\|}$. Si hanno prestazioni migliori se si aggiunge un parametro $\beta^i \in (0, 2)$:

$$\alpha^i = \beta^i \frac{f(x^i) - f(x_*)}{\|g^i\|} \tag{5.16}$$

il passo scelto in questo modo prende il nome di *Polyak step-size*. Il problema è che $f(x_*)$ di solito non è noto ed in ogni caso l'efficienza di questo metodo non è molto buona: la differenza tra il migliore valore all'iterazione k ed il valore ottimo va come $O(1/\epsilon^2)$. Tuttavia è stato dimostrato che per funzioni convesse non differenziabili questo risultato è "ottimo", nel senso che esistono funzioni per cui non è possibile trovare algoritmi più efficienti.

Target level step-size

La scelta di $\alpha^i = 1/i$ è immediatamente implementabile, mentre se si vuole usare il *Polyak step-size* è necessario trovare un modo per stimare $f(x_*)$. Questo approccio in cui si utilizzano dei parametri per stimare il valore ottimo prende il nome di *target level step-size*: si fissa un valore di riferimento che viene ricalcolato se l'algoritmo non procede come ci si aspetta.

5.3.1 Deflected subgradient

5.4 Smoothed gradient methods

Perché i metodi del subgradiente non sono così efficienti? perché già l'informazione al primo ordine è il minimo che possiamo avere, in più in questo caso è ancora più debole (potrebbe anche fornirci direzioni non di decrescita come abbiamo visto). Ci serve quindi un modello migliore di quello al primo ordine. In alcuni casi si può fare qualcosa di meglio, in particolare quando la funzione non differenziabile è del tipo:

$$f(x) = \max\{x^T Az : z \in Z\} \tag{5.17}$$

con Z un insieme convesso e compatto. Quello che possiamo fare è allora aggiungere un termine quadratico alla funzione (*smoothed function*):

$$f_\mu(x) = \max\{x^T Az - \mu\|z\|^2 : z \in Z\} \tag{5.18}$$

e risolvere quindi un problema diverso che però è differenziabile. Vale questo risultato:

$$f_\mu(x) \leq f(x) \leq f_\mu(x) + \mu D \tag{5.19}$$

dove $D = \{\|z\|^2/2 : z \in Z\} < \infty$ (è una costante finita perché Z è compatto), e quindi per $\mu \to 0$ si ha "$argmin\{f_\mu(x)\} \to x_*$". Dopo aver fatto questa trasformazione si può usare ad esempio *accelerated gradient* e per ottenere un'accuratezza ϵ è possibile scegliere $\mu = \epsilon/(2D)$ e saranno necessarie $O(1/\epsilon)$ iterazioni invece di $O(1/\epsilon^2)$. Visto che la velocità dipende dall'ϵ che scegliamo, si può "barare" scegliendo inizialmente un ϵ più piccolo e aggiustandolo mano a mano durante l'esecuzione dell'algoritmo. Inoltre è un algoritmo dove non ci sono parametri, a parte il fatto che è necessario stimare D.

5.5 Bundle methods

Il problema è dunque che l'informazione al primo ordine non è sufficiente, soprattutto quando la funzione non è differenziabile e otteniamo dei subgradienti invece che un gradiente unico. L'idea è però quella di "collezionare" più subgradienti possibili e di sfruttare il fatto che per una funzione convessa, questi ci forniscono non solo informazioni locali ma anche globali. Infatti tutti i subgradienti di qualunque punto saranno tali per cui il modello al primo ordine costruito in quel punto con il subgradiente trovato sarà sempre al di sotto della funzione. Quindi possiamo costruire un "modello al primo ordine" migliore sfruttando queste che sono informazioni non solo locali ma anche globali.

Quindi quello che si fa è che per ogni punto x^i visitato, memorizziamo le seguenti informazioni:

$$\mathcal{B} = \{x^i, f^i = f(x^i), g^i \in \partial f(x^i)\} \equiv \text{ Bundle} \tag{5.20}$$

e costruiamo il cosiddetto *Cutting-Plane model* di $f(x)$:

$$f_{\mathcal{B}}(x) = \max\{f^i + \langle g^i, x - x^i \rangle : (x^i, f^i, g^i) \in \mathcal{B}\} \tag{5.21}$$

Come costruiamo l'algortimo? a partire da un punto collezioniamo ad ogni iterazione le informazioni di quel punto nel *Bundle* e prendiamo come punto successivo quello relativo al minimo del modello $f_{\mathcal{B}}(x)$ (che è sempre minore della funzione $f(x)$).

Il problema è che prendere il minimo di $f_{\mathcal{B}}(x)$ significa risolvere un problema di ottimizzazione non differenziabile. Tuttavia è possibile riscriverlo come un problema di programmazione lineare semplicemente aggiungendo una variabile v:

$$\min\{f_{\mathcal{B}}(x)\} = \min\{v : v \geq f^i + \langle g^i, x - x^i \rangle, (x^i, f^i, g^i) \in \mathcal{B}\} \tag{5.22}$$

e quindi può essere risolto molto efficientemente.

Ci sono alcuni problemi quando si utilizza questo metodo: innanzitutto è necessario risolvere un problema di ottimizzazione ad ogni iterazione la cui dimensione (che è quella di $\mathcal{B}$) cresce ad ogni nuovo punto trovato; inoltre è un algoritmo poco "stabile" perché può fornire ad ogni iterazioni punti molto lontani tra di loro se il nostro modello è ancora poco accurato; infine non è detto che i punti presenti in $\mathcal{B}$ siano sufficienti a costruire un modello che non sia illimitato inferiormente.

Come possiamo risolvere alcuni di questi problemi? per evitare che l'algoritmo ci fornisca ad ogni iterazione punti molto lontani fra loro, possiamo scengliere un punto $\bar{x}$ (ad esempio quello che ci ha fornito la soluzione migliore fino a quel momento) che chiamiamo *stability center* e minimizziamo un problema leggermente diverso, cioè:

$$\min\left\{f_{\mathcal{B}}(x) + \mu\frac{\|x - \bar{x}\|^2}{2}\right\} \tag{5.23}$$

dove abbiamo "regolarizzato" il problema aggiungendo un termine che dipende da un *stability parameter* μ. Sostanzialmente questo termine aggiuntivo penalizza le soluzioni più lontane da $\bar{x}$ e ci garantisce anche che il problema non sia illimitato inferiormente perché la funzione complessiva è fortemente convessa. E' necessario tuttavia scegliere con cura μ e $\bar{x}$, in particolare ad ogni iterazione si controlla una condizione e si vede se cambiare $\bar{x}$ con il punto appena ottenuto (questa cosa si chiama *serious step*). Inoltre si dimostra che prima poi questo *serious step* viene fatto, oppure ci si ferma per la stopping condition.

Chapter 6

Ottimizzazione vincolata e Dualità: teoria

6.1 Ottimizzazione vincolata

Vediamo adesso come risolvere problemi di ottimizzazione con vincoli, quindi in generale:

$$(P) \qquad f_* = \min\{f(x) : x \in X\} \tag{6.1}$$

In questo caso non è sufficiente trovare dei punti stazionari ma è necessario accertarsi che le soluzioni che otteniamo siamo ammissibili, cioè che appartengano ad X.

A meno che la funzione non sia convessa, anche in questo caso non abbiamo la certezza di ottenere una minimo globale, quindi ci limitiamo sempre a cercare minimi locali, ovvero:

$$\min\{f(x) : x \in \mathcal{B}(x_*, \epsilon) \cap X\} \text{ for some } \epsilon > 0 \tag{6.2}$$

La differenza rispetto al caso non vincolato è che potrebbero esserci dei minimi locali che appartengono al bordo di X ma che non sono punti stazionari. Se invece il punto considerato appartiene alla parte interna di X allora non ci sono differenze ed un minimo locale sarà un punto stazionario.

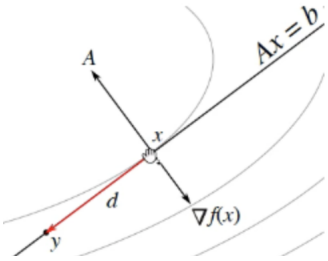
Quindi le condizioni di ottimalità (locali) sono diverse solo sul bordo ∂X.

6.2 Problemi con vincoli di uguaglianza lineari

Iniziamo a studiare il caso più semplice, ovvero problemi con soli vincoli di uguaglianza lineari:

$$(P) \qquad \min\{f(x) : Ax = b\} \tag{6.3}$$

In questo caso $x \in X \equiv x \in \partial X$, cioè tutte le soluzioni ammissibili si trovano sul bordo. Quando ci troviamo in un punto sul bordo ∂X l'insieme delle direzioni ammissibili $F = \{d \in \mathbb{R}^n : Ad = 0\}$ è vincolato; in particolare nel caso di un solo vincolo lineare $Ax = b$, le uniche direzioni ammissibili sono quelle perpendicolari all'unico vettore A:



Come si vede dalla figura si ha un minimo quando $\nabla f(x)||A$. Questa è la condizione di minimalità che adesso ricaviamo anche algebricamente.

Se siamo in uno spazio di dimensione n allora il numero di vincoli m è sicuramente minore di n, dunque $A \in \mathbb{R}^{m \times n}$ e assumiamo $rank(A) = m < n$, cioè tutte le righe di A sono linearmente indipendenti (se ci fossero delle righe linearmente dipendenti allora o sarebbero inconsistenti oppure ridondandi). Dunque possiamo scrivere A come:

$$A = [A_B, A_N] \quad con \quad detA_B \neq 0 \tag{6.4}$$

$$x = [x_B, x_N] \quad dunque \tag{6.5}$$

$$Ax = b \equiv A_B x_B + A_N x_N = b \tag{6.6}$$

ma sappiamo che A_B è invertibile quindi:

$$x_B = A_B^{-1}(b - A_N x_N) \tag{6.7}$$

cioè le variabili x_N sono quelle realmente indipendenti, e da queste possiamo ricavare le x_B. Tutto questo significa che in realtà possiamo ridurci a considerare il problema in un suo sottospazio perché ogni vincolo uccide un grado di libertà. Se consideriamo solo il sottospazio delle variabili x_N, allora questo è un problema non vincolato. Quindi scriviamo il problema non vincolato ridotto come:

$$(R) \quad \min\{r(w) = f(Dw + d) : w \in \mathbb{R}^{n-m}\} \tag{6.8}$$

con

$$D = \begin{bmatrix} -A_B^{-1} A_N \\ I \end{bmatrix} \quad d = \begin{bmatrix} A_B^{-1} b \\ 0 \end{bmatrix} \tag{6.9}$$

dove w sarebbe x_N. Adesso possiamo risolvere questo problema non vincolato, dunque calcoliamo il gradiente:

$$\nabla r(w) = D^T \nabla f(Dw + d) \tag{6.10}$$

vogliamo trovare w_* tale che $\nabla r(w_*) = 0$. Notiamo che $AD = 0$, quindi:

$$\forall \mu \in \mathbb{R}^m, z = \mu A \implies zD = 0 \equiv D^T z = 0 \tag{6.11}$$

quindi tutti i punti z scrivibili come combinazioni lineari delle righe di A, cioè come $z = \mu A$, sono punti che soddisfano $D^T z = 0$; questo significa che una condizione sufficiente per avere $\nabla r(w) = 0$ è che:

$$\exists \mu \in \mathbb{R}^m \, s.t. \, \mu A = \nabla f(x) \tag{6.12}$$

Da tutto questo ricaviamo una versione semplificata delle cosiddette **KKT conditions**, cioè un punto è un minimo (locale) se soddisfa:

$$Ax = b \qquad \wedge \qquad \exists \mu \in \mathbb{R}^m \, s.t. \, \mu A = \nabla f(x) \tag{6.13}$$

se la funzione è convessa allora otteniamo un minimo globale. Quindi possiamo dire che sul bordo non è necessario che $\nabla f(x) = 0$, ma l'importante è che:

$$\frac{\partial f}{\partial d}(x) = \langle \nabla f(x), d \rangle \geq 0 \, \forall d \in F \tag{6.14}$$

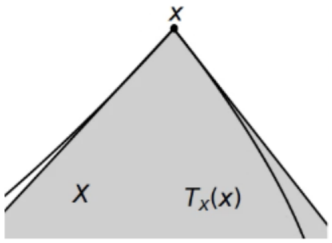
cioè se consideriamo l'insieme di tutte le direzioni ammissibili, allora si ha che nessuna di queste è una direzione di decrescita. Nell'esempio in figura di un solo vincolo lineare, se d è una direzione ammissibile allora anche $-d$ lo è, dunque la condizione diventa $\langle \nabla f(x), d \rangle = 0$. Vediamo adesso i casi più complicati.

6.3 Condizioni di ottimalità al primo ordine: versione geometrica

Per prima cosa definiamo il cosiddetto **Tangent Cone** di X in un punto x come:

$$T_X(x) = \{d \in \mathbb{R}^n : \exists\{z_i \in X\} \to x \wedge \{t_i \geq 0\} \to 0 \, s.t. \, d = \lim_{ß \to \infty} \frac{z_i - x}{t_i}\} \tag{6.15}$$

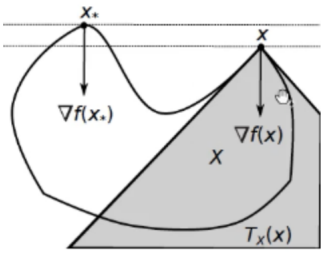
che definisce più formalmente l'insieme delle direzioni possibili a partire da x che mi permettono di rimanere all'interno della regione ammissibile (se lo step è sufficientemente piccolo). In pratica la definizione di **Tangent Cone** ci dice che una direzione d è ammissibile se esiste una successione di punti (z_i) appartenenti ad X che converge ad x e d è il limite di $\frac{z_i - x}{t_i}$. L'unico problema si ha per le direzioni "al limite", cioè quelle che possono essere tangenti alla regione ammissibile, graficamente:



Ma se trascuriamo questi casi limite (consideriamo solo l'interno del cono), allora la condizione di ottimalità per un punto x che si trova sul bordo ∂X è che:

$$\langle \nabla f(x), d \rangle \geq 0 \quad \forall d \in T_X(x) \tag{6.16}$$

Ovviamente è sempre una condizione che garantisce un'ottimalità locale e non globale, come si vede chiaramente in figura:



A meno che l'insieme non sia convesso, in quel caso è una condizione di ottimalità globale, infatti in questo caso si ha che $X \subseteq x + T_X(x)$, cioè la regione ammissibile è contenuta nel cono.

Dimostriamo adesso questa condizione di ottimalità.

Theorem 7. *x ottimo locale* $\implies \langle \nabla f(x), d \rangle \geq 0 \quad \forall d \in T_X(x)$

Proof. Assumiamo per assurdo che x sia un ottimo locale e che esista una direzione d che appartiene al suo cono tangente che abbia un prodotto scalare con il gradiente in x negativo, cioè:

$$\exists d \in T_X(x) \, s.t. \, \langle \nabla f(x), d \rangle < 0 \wedge x \text{ ottimo locale} \tag{6.17}$$

Dalla definizione di cono tangente, $\exists X \supset \{z_i\} \to x, \{t_i\} \to 0$, tale che:

$$d = \lim_{i \to \infty} \frac{z_i - x}{t_i} \tag{6.18}$$

A questo punto sviluppiamo $f(x)$ al primo ordine intorno ad x e otteniamo:

$$f(z_i) = f(x) + \langle \nabla f(x), z_i - x \rangle + R(z_i - x) \tag{6.19}$$

Dividiamo tutto per t_i e facciamo il limite per $i \to \infty$:

$$\lim_{i\to\infty} \frac{f(z_i) - f(x)}{t_i} = \lim_{i\to\infty} \langle \nabla f(x), (z_i - x)/t_i \rangle + \frac{R(z_i - x)}{t_i} \tag{6.20}$$

$$\lim_{i\to\infty} \frac{f(z_i) - f(x)}{t_i} = \lim_{i\to\infty} \langle \nabla f(x), d \rangle + \frac{R(z_i - x)}{t_i} \tag{6.21}$$

Ma per Taylor si ha $\lim_{i\to\infty} R(z_i - x)/t_i = 0$ che implica dunque:

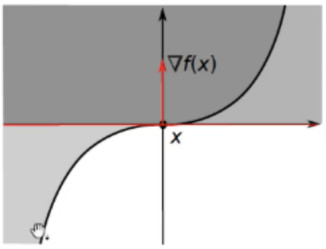
$$\lim_{i\to\infty} \frac{f(z_i) - f(x)}{t_i} = \lim_{i\to\infty} \langle \nabla f(x), d \rangle < 0 \tag{6.22}$$

ovvero:

$$\exists z_i \in X \cap \mathcal{B}(x_*, \epsilon) \, s.t. \, f(z_i) < f(x) \, \forall \epsilon \tag{6.23}$$

cioè abbiamo trovato dei punti z_i migliori di x contraddicendo l'ipotesi che x fosse un ottimo locale. □

Questa condizione è necessaria ma non sufficiente, vediamo un controesempio:



Il cono tangente è il semispazio positivo ed x soddisfa la condizione di ottimalità ma non è un minimo locale.

A questo punto il problema è che il tangent cone per come è definito, è complicato da calcolare. Vediamo allora una definizione alternativa del **cono delle direzioni ammissibili**:

$$F_X(x) = \{d \in \mathbb{R}^n : \exists \bar{\epsilon} > 0 \, s.t. \, x + \epsilon d \in X \, \forall \epsilon \in [0, \bar{\epsilon}]\} \tag{6.24}$$

si ha che $F_X \subseteq T_X$ perché le direzioni tangenti non sono contenute in F_X (perché muovendoci lungo queste direzioni potremmo ottenere dei punti non ammissibili). Si ha quindi che F_X in generale non è un insieme chiuso, mentre T_X lo era.

Se X è convesso si ha che sia T_X che F_X sono convessi e $clF_X = T_X$. Inoltre:

Theorem 8. *f ed X convessi* $\implies$

$$x \text{ ottimo globale} \iff \langle \nabla f(x), d \rangle \geq 0 \quad \forall d \in T_X(x) \tag{6.25}$$

Tuttavia vorremmo una condizione di ottimalità considerando F_X e non T_X, perché quest'ultimo è difficile da costruire.

6.4 Condizioni di ottimalità al primo ordine: versione algebrica

Come caratterizziamo T_X? dipende da come caratterizziamo X, ovvero da come descriviamo esplicitamente i vincoli del problema. Indichiamo con $\mathcal{I}$ l'insieme dei vincoli di disuguaglianza e con $\mathcal{J}$ l'insieme dei vincoli di uguaglianza, possiamo descrivere la regione ammissibile come:

$$X = \{x \in \mathbb{R}^n : g_i(x) \leq 0 \, i \in \mathcal{I}, h_j(x) = 0 \, j \in \mathcal{J}\} \tag{6.26}$$

Indicando con $G(x) = [g_i(x)]_{i\in\mathcal{I}}$ e con $H(x) = [h_j(x)]_{j\in\mathcal{J}}$, possiamo riscrivere X come:

$$X = \{x \in \mathbb{R}^n : G(x) \leq 0, H(x) = 0\} \tag{6.27}$$

dove stiamo assumendo $g_i(x)$ e $h_j(x)$ almeno differenziabili.

Definiamo inoltre l'insieme dei vincoli *attivi* in x come

$$\mathcal{A}(x) = \{i \in \mathcal{I} : g_i(x) = 0\} \subseteq \mathcal{I} \tag{6.28}$$

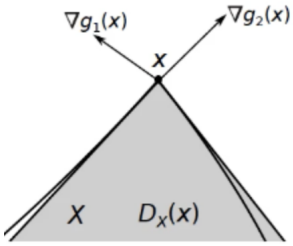
(i vincoli in $\mathcal{J}$ sono sempre attivi altrimenti la soluzione non è ammissibile). Dato un sottoinsieme $\mathcal{B} \subseteq \mathcal{I}$ indichiamo con $G_{\mathcal{B}} = [g_i(x)]_{i\in\mathcal{B}} : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{B}|}$.

A questo punto definiamo il **cono delle direzioni ammissibili al primo ordine** in $x \in X$ come:

$$D_X(x) = \{d \in \mathbb{R}^n : \langle \nabla g_i(x), d\rangle \leq 0 \, i \in \mathcal{A}(x), \langle \nabla h_j(x), d\rangle = 0 \, j \in \mathcal{J}\} \tag{6.29}$$

$$= \{d \in \mathbb{R}^n : (JG_{\mathcal{A}(x)})d \leq 0, (JH(x))d = 0\} \tag{6.30}$$

dove notiamo che il gradiente è calcolato solo per i vincoli attivi, in quanto questi sono gli unici che "contano" perché sono quelli che definiscono il bordo su cui si trova il punto x. Quindi in pratica vogliamo tutte le direzioni che abbiamo prodotto scalare negativo con tutti i gradienti dei vincoli attivi nel punto, graficamente



mentre per i vincoli di uguaglianza è necessario che il prodotto scalare sia nullo, in quanto ci possiamo spostare solo lungo quel vincolo.

Si può dimostrare che $T_X(x) \subseteq D_X(x)$, dunque se la condizione di ottimalità che avevamo scritto per $T_X(x)$ vale per $D_X(x)$ allora abbiamo automaticamente una condizione sufficiente per l'ottimalità locale ed il vantaggio è che $D_X(x)$ si costruisce facilmente perché basta calcolare i gradienti dei vincoli attivi. Tuttavia vorremmo che i due insiemi fossero uguali; ci sono tre condizioni dette **constraint qualifications** che ci garantiscono che $D_X(x) = T_X(x)$:

- *Affine constraints*: g_i e h_j lineari $\forall i \in \mathcal{I}$ e $j \in \mathcal{J}$;
- *Slater's condition*: g_i convesse $\forall i \in \mathcal{I}$, h_j lineari $\forall j \in \mathcal{J}$ ed $\exists \bar{x} \in X \, s.t. \, g_i(\bar{x}) < 0 \, \forall i \in \mathcal{I}$, cioè esiste almeno un punto interno.
- *Linear independence*: $\bar{x} \in X \wedge$ i vettori $\{\nabla g_i(\bar{x}) : i \in \mathcal{A}(\bar{x})\} \cup \{\nabla h_j(\bar{x}) : j \in \mathcal{J}\}$ linearmente indipendenti.

Ricapitolando se valgono queste tre condizioni allora:

$$x \text{ ottimo locale} \implies \langle \nabla f(x), d \rangle \geq 0 \quad \forall d \in D_X(x) \tag{6.31}$$

Questa condizione di ottimalità è sicuramente meglio della precedente in quanto è possibile calcolare facilmente D_X, tuttavia rimane il problema di dover calcolare il prodotto scalare tra il gradiente della funzione obiettivo e *tutte* le possibili direzioni d che sono infinite. Ma questo problema può essere facilmente risolto grazie al **Farkas' lemma**.

Farkas' Lemma

Si ha che D_X è un *cono poliedrale*, cioè può essere scritto come: $\mathcal{C} = \{d \in \mathbb{R}^n : Ad \leq 0\}$ per una qualche matrice $A \in \mathbb{R}^{k \times n}$. Indichiamo il *cono duale* con $\mathcal{C}^* = \{c = \sum_{i=1}^{k} \lambda_i A_i : \lambda \geq 0\}$, cioè il cono formato da tutti i punti che possono essere scritti come combinazioni lineari con coefficienti positivi a partire dalle A_i; in figura

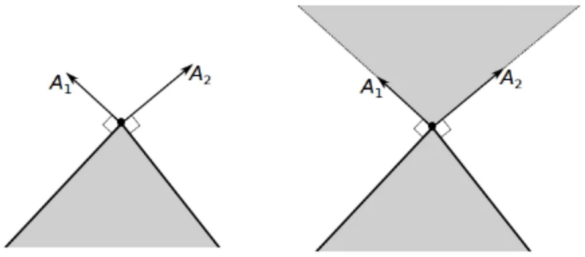


Figure 6.1: D_X a sinistra ed il cono *duale* a destra

Il *Farkas' Lemma* ci dice che per ogni punto $c \in \mathbb{R}^n$ o questo appartiene al cono duale $\mathcal{C}^*$, oppure esiste una direzione d che appartiene al cono $\mathcal{C}$ che ha prodotto scalare negativo con c. Le due situazioni sono mostrate in figura

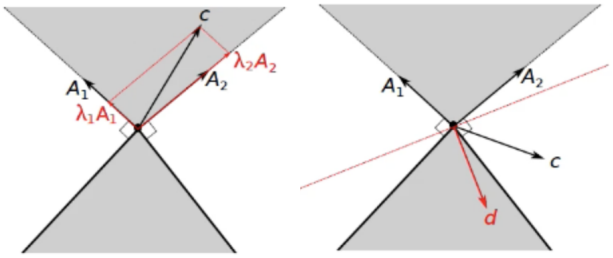


Figure 6.2: $c \in \mathcal{C}^*$ a sinistra e $c \notin \mathcal{C}^*$ a destra

Ritornando al nostro problema possiamo usare questo lemma per dire che: la condizione di ottimalità in un punto x_* che ci dice tutte le direzioni del cono devono avere un prodotto scalare positivo con il gradiente equivale a dire che $-\nabla f(x_*)$ deve appartenere al cono duale e quindi deve poter essere scritto come:

$$\exists \lambda \in \mathbb{R}_+^{|\mathcal{A}(x)|} \text{ e } \mu \in \mathbb{R}^{|\mathcal{J}|} \text{ s.t.} \tag{6.32}$$

$$-\nabla f(x_*) = \sum_{i \in \mathcal{A}(x)} \lambda_i \nabla g_i(x_*) + \sum_{j \in \mathcal{J}} \mu_j \nabla h_j(x_*) \tag{6.33}$$

dove appunto il cono è definito dai vincoli attivi e da quelli di uguaglianza e quindi possiamo scrivere $-\nabla f(x_*)$ come una combinazione lineare a coefficienti positivi dei vincoli attivi più una combinazione non vincolata in segno di quelli di uguaglianza. Il motivo per cui i coefficienti dei vincoli in $\mathcal{J}$ non sono vincolati in segno è che un vincolo di uguaglianza $h(x) = 0$ può

essere equivalentemente riscritto come $h(x) \leq 0 \wedge -h(x) \leq 0$, quindi avremo due coefficienti $\mu_+, \mu_- \geq 0$ di cui nello sviluppo finale comparirà la differenza $\mu = \mu_+ - \mu_-$ che non è vincolata in segno.

Questo risultato è molto importante perché se valgono le *constraint qualifications* e siamo in grado di trovare i moltiplicatori λ e μ allora abbiamo un condizione necessaria (anche sufficiente nel caso convesso) di ottimalità.

La forma canonica in cui vengono scritte queste condizioni è la seguente.

Karush-Kuhn-Tucker conditions: $\exists \lambda \in \mathbb{R}_+^{|\mathcal{A}(x)|}$ e $\mu \in \mathbb{R}^{|\mathcal{J}|}$ s.t.

$$\text{(KKT-F)} \quad g_i(x) \leq 0 \, i \in \mathcal{I} \quad h_j(x) = 0 \, j \in \mathcal{J} \tag{6.34}$$

$$\text{(KKT-G)} \quad \nabla f(x_*) + \sum_{i \in \mathcal{I}} \lambda_i \nabla g_i(x_*) + \sum_{j \in \mathcal{J}} \mu_j \nabla h_j(x_*) = 0 \tag{6.35}$$

$$\text{(KKT-CS)} \quad \sum_{i \in \mathcal{I}} \lambda_i g(x) = 0 \tag{6.36}$$

dove la prima garantisce l'ammissibilità della soluzione, mentre l'ultima è la cosiddetta *condizione degli scarti complementari*, in quanto le λ_i sono tutte positive, quindi la somma di tutti i termini sarà nulla se e solo se per ogni i o $\lambda_i = 0$ oppure $g_i(x) = 0$. Questa condizione deriva dal fatto che nella somma in (KKT-G) abbiamo sostituito l'insieme dei vincoli attivi con $\mathcal{I}$, quindi (KKT-CS) serve per annullare il contributo dei vincoli non attivi, per i quali $g_i(x) \neq 0$ e quindi il relativo coefficiente λ_i sarà nullo.

A questo punto concludiamo enunciando il seguente teorema.

Theorem 9. *$T_X(x) = D_X(x) \wedge x$ ottimo locale $\implies$ (KKT)*

Si ha un $\iff$ nel caso convesso.

Dunque tutti gli algoritmi che vedremo cercheranno dei punti che soddisfano (KKT).

6.5 Condizioni di ottimalità al secondo ordine

Abbiamo visto che le condizioni (KKT) sono necessarie ma non sufficienti, dunque è necessario trovare delle condizioni di ottimalità al secondo ordine. Nell'ottimizzazione non vincolata la condizione di ottimalità al secondo ordine era $\nabla^2 f(x_*) \succ 0$; adesso invece vediamo una generalizzazione. Si definisce **funzione Lagrangiana** la seguente

$$L(x; \lambda, \mu) = f(x) + \sum_{i \in \mathcal{I}} \lambda_i g_i(x) + \sum_{j \in \mathcal{J}} \mu_j h_j(x) \tag{6.37}$$

e notiamo che (KKT-G)$\equiv \nabla L(x; \lambda, \mu) = 0$, cioè la condizione di ottimalità al primo ordine equivale a trovare un punto stazionario della funzione lagrangiana. Quindi potremmo aspettarci che la condizione al secondo ordine sia $\nabla^2 L(x; \lambda, \mu) \succeq 0$, ma in realtà è un po' più complicato. Si definisce il cosiddetto *critical cone* $C(x, \lambda, \mu)$ (vedi slides per la definizione) e una condizione sufficiente di ottimalità locale è

$$\text{(KKT)} \wedge \nabla^2 L(x; \lambda, \mu) \succ 0 \text{ on } C(x, \lambda, \mu) \implies x \text{ local optimum} \tag{6.38}$$

mentre $\nabla^2 L(x; \lambda, \mu) \succeq 0$ è una condizione necessaria.

6.6 Dualità Lagrangiana

La funzione lagrangiana ricopre dunque un ruolo molto importante; vediamo adesso da dove viene fuori. Riscriviamo il problema di partenza

$$(P) \quad \min\{f(x) : G(x) \leq 0, H(x) = 0\} \tag{6.39}$$

La funzione lagrangiana invece è

$$L(x; \lambda, \mu) = f(x) + \lambda G(x) + \mu H(x) \tag{6.40}$$

consideriamo allora il seguente problema

$$\psi(\lambda, \mu) = \min_{x \in \mathbb{R}^n} \{f(x) + \lambda G(x) + \mu H(x)\} \tag{6.41}$$

Si ha che questo problema è un rilassamento del problema di partenza, cioè il valore ottimo è un lower bound per il problema (P), infatti nei punti appartenenti alla regione ammissibile di (P) i contributi di $\lambda G(x)$ e $\mu H(x)$ sono rispettivamente negativo e nullo ($\lambda \leq 0, G(x) \leq 0, H(x) = 0$). Una cosa importante è che questo secondo problema non è vincolato, quindi possiamo usare tutte le tecniche che abbiamo già visto. Quindi per qualunque scelta di $\lambda \geq 0$ e μ, fissata una qualunque soluzione ammissibile per (P) $\bar{x}$, si ha:

$$\psi(\lambda, \mu) \leq f(\bar{x}) \tag{6.42}$$

Questa proprietà viene chiamata *weak duality*. In particolare è valida anche per la soluzione ottima x_*.

Naturalmente il lower bound fornito da $\psi(\lambda, \mu)$, può essere più o meno significativo; cerchiamo il migliore cioè il risultato del seguente problema detto *duale*

$$(D) \quad \max\{\psi(\lambda, \mu) : \lambda \geq 0\} \tag{6.43}$$

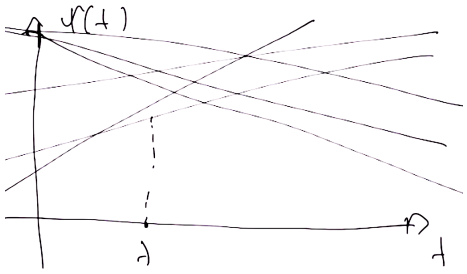
che è un problema vincolato (se avessimo solo vincoli di uguaglianza nel problema di partenza allora sarebbe un problema non vincolato), ma i vincoli sono pochi e semplici. Anche massimizzando ψ, in ogni caso si ha:

$$v(D) \leq v(P) \tag{6.44}$$

Ci chiediamo quando i due valori ottimi sono uguali: nel caso convesso.

Inoltre ψ è una funziona concava quindi possiamo ottenere un ottimo globale, tuttavia tipicamente non è differenziabile ed inoltre calcolare il suo valore significa ogni volta risolvere un problema di minimizzazione (anche se comunque non è vincolato quindi potrebbe non essere particolarmente difficile).

Se guardiamo L come una funzione di λ e μ, allora è una funzione lineare, e ψ non è che il minimo punto per punto di funzioni lineari, quindi è concava ed è fatta così



Anche se non è differenziabile si possono utilizzare i subgradienti per risolvere il duale lagrangiano. In ogni caso anche se non siamo nel caso convesso, risolvere il duale lagrangiano può essere utile per ottenere dei validi lower bound, tuttavia in questo caso il problema di minimizzazione che dobbiamo risolvere per calcolare ψ deve essere risolto all'ottimo, e non essendo una funzione convessa può essere complicato.

Si può inoltre dimostrare:

Strong duality

$$(P) \text{ convex, } x_* \text{ optimum, } T_X(x_*) = D_X(x_*) \implies v(D) = v(P) \tag{6.45}$$

Proof. Dal momento che x_* è una soluzione ottima e $T_X(x_*) = D_X(x_*)$, allora valgono le condizioni necessarie cioè le KKT, ovvero $\exists(\lambda_*, \mu_*)$ che soddisfano le KKT in x_*. Ma allora tra tutte le soluzioni ottime del problema del problema duale (D), ne esisterà una (λ^*, μ^*) per cui $v(D) = v(P)$, infatti:

$$v(D) \geq \psi(\lambda^*, \mu^*) = L(x_*, \lambda^*, \mu^*) = f(x_*) = v(P) \geq v(D) \tag{6.46}$$

□

Il problema è che in generale (D) ha più soluzioni ottime, e alcune di queste potrebbero non essere nemmeno ammissibili per il problema di partenza. Se la soluzione ottima è unica invece non ci sono problemi e tra l'altro in questo caso si ha che ψ è anche differenziabile.

6.7 Casi particolari del Duale

Vediamo alcuni casi particolari a cui applicare questi concetti e vediamo come si semplifica e come si può sfruttare il duale lagrangiano.

6.7.1 Programmazione Lineare

Nel caso di un problema lineare, il duale lagrangiano è semplicemente il duale, infatti

$$(P) \quad \min\{cx : Ax \geq b\} \tag{6.47}$$

La funzione lagrangiana è $L(x; \lambda) = cx + \lambda(b - Ax)$, e il duale

$$\psi(\lambda) = \min_{x \in \mathbb{R}^n} L(x; \lambda) = \begin{cases} -\infty & \text{if } c - \lambda A \neq 0 \\ \lambda b & \text{if } c - \lambda A = 0 \end{cases} \tag{6.48}$$

dunque si ha:

$$(D) \quad \max\{\lambda b : \lambda A = c, \lambda \geq 0\} \tag{6.49}$$

Strong duality $\equiv v(P) = v(D)$ vale quasi sempre.

6.7.2 Quadratic Programs

Consideriamo un esempio semplice

$$(P) \quad \min\left\{\frac{1}{2}\|x\|_2^2 : Ax = b\right\} \tag{6.50}$$

La funzione lagrangiana è $L(x; \lambda) = \frac{1}{2}\|x\|_2^2 + \mu(Ax - b)$. Minimizzare questa funzione significa minimizzare una funzione convessa senza vincoli, quindi possiamo semplicemente calcolare il gradiente e porlo uguale a 0. Dunque la funzione duale

$$\psi(\mu) = \min_{x \in \mathbb{R}^n} L(x; \mu) \tag{6.51}$$

$$\nabla L(x; \mu) = x + \mu A = 0 \iff x = -\mu A \implies \tag{6.52}$$

$$\psi(\mu) = -\frac{1}{2}\mu^T(AA^T)\mu - \mu b \tag{6.53}$$

Il duale lagrangiano è allora

$$(D) \quad \max\left\{-\frac{1}{2}\mu^T(AA^T)\mu - \mu b : \mu \in \mathbb{R}^m\right\} \tag{6.54}$$

cioè un altro problema di programmazione quadratica ma *non vincolato*.

Un esempio più complicato è il caso di un problema di programmazione quadratica strettamente convesso, cioè assumendo $Q \succ 0$

$$(P) \quad \left\{\frac{1}{2}x^TQx + qx : Ax \geq b\right\} \tag{6.55}$$

vale quasi sempre la proprietà di *strong duality*, cioè $v(P) = v(D)$ ed il problema duale è

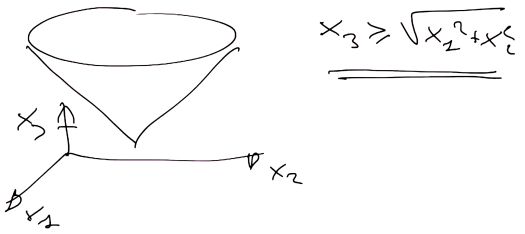
$$(D) \quad \max\left\{\lambda b - \frac{1}{2}v^TQ^{-1}v : \lambda A - v = q, \lambda \geq 0\right\} \tag{6.56}$$

6.7.3 Conic Programs

L'idea è di considerare problemi non lineari e trattarli come se fossero lineari.

$$(P) \quad \min\{cx : Ax \geq_K b\} \tag{6.57}$$

dove il maggiore o uguale è inteso rispetto "ad un cono", per esempio graficamente:



cioè quando scriviamo $Ax - b \geq_K 0$ in realtà intendiamo un vincolo non lineare come quello scritto in figura, ma lo scriviamo come se fosse lineare specificando rispetto a quale cono è scritto. Se il cono K è il quadrante positivo allora ci riduciamo al caso lineare classico. Inoltre il duale di un problema di programmazione conica è ancora un problema di programmazione conica. Un cono importante è ad esempio quello formato dalle matrici semidefinite positive.

Chapter 7

Ottimizzazione Vincolata

Ci restringiamo a considerare problemi in cui in vincoli sono dei vincoli di disuguaglianza lineari e la funzione obiettivo è lineare o quadratica. Inoltre ci concentriamo principalmente su vincoli della forma $l \leq x \leq u$ detti *boxed constraints*. Possiamo *shiftare* le x e senza perdere di generalità considerare vincoli del tipo $0 \leq x \leq u$.

7.1 Equality Constrained Quadratic Problems

7.2 Projected gradient method

7.3 Active-Set method

7.4 Frank-Wolfe method

7.5 Dual methods

7.6 Interior-Point method