Optimization

Ottimizzazione quadratica convessa su dominio vincolato con metodo Frank Wolfe

Pierfrancesco Benincasa

June 2024

1 Introduzione

Il problema che verrà analizzato in questo report riguarda l'ottimizzazione di una funzione qudratica convessa con dominio vincolato attraverso l'utilizzo del metodo Frank-Wolfe. La funzione quadratica $f : D \subseteq R^n$ di forma:

$$f(x) = x^T Q x + qx$$

dove $Q \in R^{nxn}$ ed è una matrice semi-definita positiva mentre $q \in R^n$ un vettore. La funzione, come indicato dalla traccia, è vincolato su un set di K simplessi disgiunti, riportando quanto segue :

$$min\{x^T Q x + qx : \sum_{i \in I^k} x_i = 1 \ , k \in K \ , x \geq 0\}$$

dove $x \in R^n$,e il set degli indici I^k forma una partizione di un set {1,..,n} , ad esempio , $\cup_{k \in K} I^k = \{1, ..., n\}$ *e* $I^h \cap I^k = \emptyset$ per ogni h e k. Il dominio sottoposto a vincolo è definito come segue:

$$D = \{x \in R^n \mid x_i \geq 0, \ \sum_{i \in I^k} x_i = 1, \forall k \in \{1, ..., K\}\}$$

2 Proprietà e Condizioni

2.1 Dominio

Come definito dalla traccia esiste un set di indici I^k, tale per cui, si formano una partizione di un set, nel quale ogni I^k definisce l'insieme degli indici che andranno a formare un politopo P^k in R^n formato dall'insieme di vertici V^k. Ognuno dei Politopi è un insieme compatto e convesso, in quanto è la forma convessa dei suoi vertici V^k. Date queste condizioni possiamo affermare che il dominio D è la somma dei Politopi corrispondente alla 'Somma di Minkowski':

$$D = P^1 + P^2 + ... + P^K$$

Anche il dominio mantiene la proprietà di convessità essendo la somma di politopi convessi [3].

2.2 Convessità

Fondamentale nello studio della convessità sono l'insieme di tutte le matrici simmetriche semidefinite positive, è un insieme convesso (in generale, $\mathbb{S}^n \subseteq \mathbb{R}^{n \times n}$ denota l'insieme delle matrici simmetriche $n \times n$). Ricordiamo che una matrice $Q \in \mathbb{R}^{n \times n}$ è simmetrica semidefinita positiva se e solo se $Q = Q^T$ e per tutti $x \in \mathbb{R}^n$, $x^T Q x \geq 0$. Consideriamo ora due matrici simmetriche semidefinite positive $Q, B \in \mathbb{S}^n_+$ e $0 \leq \theta \leq 1$. Allora per ogni $x \in \mathbb{R}^n$ [2],

$$x^T(\theta Q + (1-\theta)B)x = \theta x^T Q x + (1-\theta) x^T B x \geq 0. \quad (1)$$

Considerata una funzione $f : D \to R$ diciamo che f è convessa se per ogni $x_1,\ x_2$ *punti di*D vale la seguente disequazione:

$$f((1-\lambda)x_1 + \lambda x_2) \leq (1-\lambda)f(x_1) + \lambda f(x_2) \tag{2}$$

Diversamente dalla classica definizione di funzione convessa, è sicuramente più utile, anche per il nostro tipo di problema, utilizzare condizioni e proposizioni che riguardano le funzioni quadratiche in R^n che affermano quanto segue:

Condizione del primo ordine
Supponiamo che una funzione $f : R^n \to R$ sia differenziabile (cioè, il gradiente $\nabla_x f(x)$ esiste per tutti i punti x nel dominio di f). Allora f è convessa se e solo se $\mathcal{D}(f)$ è un insieme convesso e per tutti $x, y \in \mathcal{D}(f)$,

$$f(y) \geq f(x) + \nabla_x f(x)^T (y - x). \tag{3}$$

La funzione $f(x) + \nabla_x f(x)^T (y - x)$ è definita *approssimazione del primo ordine* della funzione f nel punto x. Intuitivamente, questa può essere pensata come l'approssimazione di f con la sua retta tangente nel punto x.

La condizione di primo ordine per la convessità afferma che f è convessa se e solo se la retta tangente è un sotto-stimatore globale della funzione f. In altre parole, se prendiamo la nostra funzione e tracciamo una tangente in qualsiasi punto, allora ogni punto su questa linea sarà al di sotto del corrispondente punto su f.

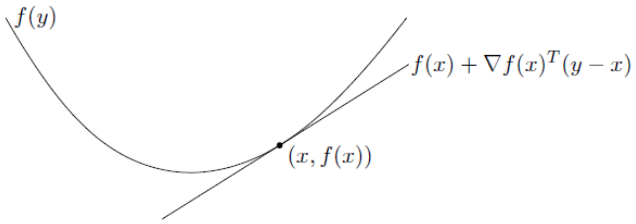


Figure 1: Condizione del primo ordine

Condizione del secondo ordine
Supponiamo una funzione $f : R^n \to R$ due volte differenziabile (cioè, l'Hessiana $\nabla_x^2 f(x)$ è definita per tutti i punti x nel dominio di f). Allora f è convessa se e solo se $D(f)$ è un insieme convesso e la sua Hessiana è semidefinita positiva: cioè, per ogni $x \in D(f)$,

$$\nabla_x^2 f(x) \succeq 0.$$

Qui, la notazione '$\succeq$' quando usata in congiunzione con matrici si riferisce alla semidefinità positiva, piuttosto che all'ineguaglianza componente per componente. In una dimensione, questo è equivalente alla condizione che la seconda

derivata $f''(x)$ sia sempre positiva (cioè, la funzione ha sempre curvatura positiva). Ancora, analogo sia alla definizione che alle condizioni di primo ordine per la convessità, f è strettamente convessa se la sua Hessiana è definita positiva, concava se la Hessiana è semidefinita negativa, e strettamente concava se la Hessiana è definita negativa.

2.3 Proprietà della funzione Lipschitz

Nel nostro caso specifico, con Q matrice simmetrica e semidefinita positiva e i vincoli denotati da vertici che formano simplessi anch'essi convessi, ci permetterà di dire che il punto minimo trovato con l'algoritmo sarà un minimo globale. La funzione è quadratica con Q semidefinita positiva e risulta quindi differenziabile con derivata seconda non negativa nel dominio. Per poter successivamente discutere alcuni possibili risultati di convergenza dell'algoritmo Frank-Wolfe, è necessario notare che la funzione è L-smooth, ovvero 'e differenziabile e il suo gradiente è L-continuo con costante di Lipschitz positiva, questo ci permette di affermare che la funzione non presenta variazioni estremamente ripide o instabilità locali rendendo i metodi di ottimizzazioni più stabili e affidabili.

Una funzione f è detta avere una proprietà di Lipschitz se esiste una costante $L \geq 0$ tale che per ogni coppia di punti x,y nel dominio di f [6] :

$$||f(x) - f(y)|| \leq L||x - y|| \tag{4}$$

Quando questa proprietà è riferita al gradiente della funzione, si parla di gradiente Lipschitziano. Una funzione f ha un gradiente Lipschitziano se esiste una costante L tale che per ogni x, y:

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \tag{5}$$

Per funzioni due volte differenziabili, questa condizione si riduce a dire che la norma operatore della matrice Hessiana (H) è limitata superiormente da L. Poichè la norma operatore di una matrice simmetrica è uguale al suo massimo autovalore

$$L = 2||Q|| = 2\lambda_{max} \tag{6}$$

3 Algoritmo di Frank Wolfe

3.1 Formulazione matematica e problema di ottimizzazione

L'algoritmo di Frank-Wolfe è un algoritmo iterativo di ottimizzazione del primo ordine per l'ottimizzazione convessa vincolata.il metodo è stato originariamente proposto da Marguerite Frank e Philip Wolfe nel 1956 è conosciuto anche con diverse denominazioni come: metodo del gradiente condizionato, algoritmo del gradiente ridotto e algoritmo della combinazione convessa. In ogni iterazione, l'algoritmo Frank-Wolfe considera un'approssimazione lineare della funzione obiettivo e si muove verso un minimizzatore di questa funzione lineare (presa sullo

stesso dominio). Essendo un problema di ottimizzazione lineare su un dominio convesso, la soluzione ottima si trova sempre su un vertice del dominio (in questo caso caratterizzato da simplessi). Questo è un aspetto fondamentale in questo problema di ottimizzazione poichè caratterizza la nostra funzione e i suoi vincoli e di conseguenza rende ottimale il metodo di Frank-Wolfe per la sua risoluzione.
Supponiamo che D sia un insieme convesso compatto in uno spazio vettoriale e $f : D \to R$ è una funzione reale convessa e differenziabile. L'algoritmo FW risolve il problema di ottimizzazione

$$Minimize\ f(x)\ subject\ to\ x \in D$$

Possiamo suddividere il cycle time dell'algoritmo nei seguenti 5 step:
1. Si inizializza scegliendo un punto $x_0 \in D$ e si pone k = 0
2. Si calcola un

$$s_k = min\ \ < s, \nabla f(x_k) > \tag{7}$$

3. Si determina un $\gamma < [0, 1]$ *(Nelle prossime sezioni saranno presentate diverse tipologia di approccio al calcolo dello step size.)*
4. Si pone $x_{k+1} = x_k + \gamma(s_k - x_k)$ *(la differenza $s_k - x_k$ indica la direzione verso cui muoversi)*
5. Si pone $k = k + 1$ e si ritorna al passo 2
6. Viene definito un criterio di arresto (scegliendo un ϵ per bloccare l'esecuzione)

La scelta del parametro ϵ è cruciale per garantire un equilibrio tra l'accuratezza della soluzione ottenuta e l'efficienza computazionale dell'algoritmo di Frank-Wolfe. Un valore di ϵ troppo grande può portare a soluzioni subottimali, mentre un valore troppo piccolo può aumentare il numero di iterazioni senza un miglioramento significativo della soluzione. Pertanto, ϵ è scelto in base alla tolleranza agli errori specifica del problema in esame e alla disponibilità di risorse computazionali. Analiticamente quello che si vuol cercare assegnando un ϵ piccolo a piacere, è di convergere uniformemente verso una soluzione ottimale contenuta in un intorno del punto x di ampiezza 2 ϵ, $x \in [x+\epsilon, x-\epsilon]$.

Il criterio di arresto utilizzato

$$(x_{k-1} - s_k)^T \nabla f(x_{k-1}) \leq \epsilon \tag{8}$$

il termine $x_{k-1} - s_k$ rappresenta la direzione del passo all'iterazione corrente k, mentre il prodotto scalare con il gradiente $\nabla f(x_{k-1})$ misura quanto questo passo contribuisce alla riduzione della funzione obiettivo, quando esso assumerà valori molto piccoli, possiamo affermare che i miglioramenti apportati da ulteriori iterazioni diventano trascurabili. In questo modo si cerca di garantire l'efficienza computazionale senza compromettere l'accuratezza della soluzione. Questo approccio consente di ottenere soluzioni approssimate di alta qualità in tempi computazionali ragionevoli, particolarmente importante in applicazioni di ottimizzazione su larga scala.

Algorithm 1 Algoritmo di Frank-Wolfe

procedure FRANKWOLFE
Require: Funzione obiettivo f e regione ammissibile D
Require: Un punto iniziale ammissibile $x_0 \in D$
Require: Sequenza dei passi γ_k e livello di tolleranza $\epsilon > 0$
    for $k = 0, 1, 2, \ldots$ **do**
        Calcola $s_k = \arg\min_{s \in \Omega} \nabla f(x_{k-1})^T s$
        if $(x_{k-1} - s_k)^T \nabla f(x_{k-1}) \leq \epsilon$ **then**
            ferma
            return x_k
        end if
        Aggiorna $x_k = (1 - \gamma_k) x_{k-1} + \gamma_k s_k$
    end for
    return L'ultimo risultato x dell'iterazione k
end procedure

3.2 Ottimizzazione con vincolo

Come detto in precedenza, ad ogni iterazione l'algoritmo considera un'approssimazione lineare che si muove in direzione di un punto che minimizza la seguente approssimazione lineare, che indicheremo con $g_x(s)$. Dato $x \in D$, $s_k := \arg\min_{s \in D} s^T \nabla f(x)$ minimizza $L_x(s)$ in D e il vettore $d = s_k - x$ è una direzione di discesa.

$$\min_{s \in D} L_x(s) = f(x) + \min_{s \in D} s^T \nabla f(x) - x^T \nabla f(x) = L_x(s_k) \tag{9}$$

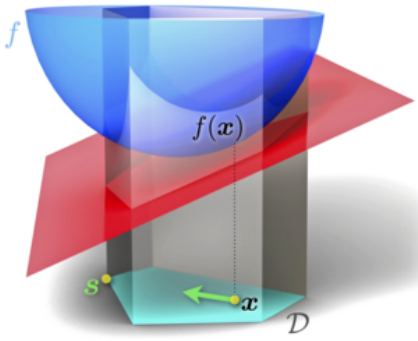


Figure 2: Rappresentazione della funzione con dominio vincolato [4]

Per trovare il vettore s_k, sfruttiamo la struttura del dominio. L'idea intuitiva è di prendere gli indici di ogni I^k e usarli per selezionare l'indice della componente minima del gradiente limitato a quegli indici. Dopo aver proceduto in questo modo per ogni I^k, otteniamo un insieme di indici $U = \{j_1, j_2, \ldots, j_K\}$. Infine, per minimizzare il prodotto scalare, scegliamo s tale che $s_i = 1$ se $i \in U$ e $s_i = 0$ per tutti gli altri i,

quindi andare a porre 0 negli indici in cui il gandiente non ha il minimo con riferimento il simplesso, vincolo del dominio.

Algorithm 2 Min Gradient Vertex

```
1: procedure MIN_GRADIENT_VERTEX(f, x, partition)
2:     grad ← ∇f(x_k)
3:     s ← 0 inizializziamo un vettore tutto nullo
4:     for I^k in partition do
5:         j ← arg min grad[I^k] scelta dell'argomento minimo per partizione
6:         s[I^k[j]] ← 1
7:     end for
8: end procedure
```

Algorithm 3 Algoritmo di Frank-Wolfe con vincoli

```
1: procedure FW(f, x, partition, ϵ)
2:     repeat
3:         s̄ ← Min Gradient Vertex(f, x, partition)
4:         d ← s̄ − x
5:         Select γ ∈ [0, 1]
6:         x ← x + γd
7:     until −d^T ∇f(x) < ϵ
8: end procedure
```

Il problema risolto dall'Algorithm 2 può essere formulato come la minimizzazione dell'approssimazione lineare della funzione su un dominio vincolato definito da simplessi. L'approssimazione lineare, espressa precedentemente dall'espressione (9), minimizza la funzione rispetto a s soggetto al Dominio D definito da simplessi, il che significa trovare il vertice del simplesso che minimizza il prodotto scalare $\nabla f(x)^T s$ (essendo la funzione approssimata linearmente, il minimo ricadrà su uno dei suoi vertici). Ogni partizione I^k nel dominio vincolato corrisponde a un simplesso, e l'Algorithm 2 seleziona il vertice del simplesso che minimizza il gradiente $\nabla f(x)$ in quella partizione. La scelta del vertice j come j = $arg\ min \nabla f(x)[I^k]$ garantisce che si scelga il punto che riduce maggiormente la funzione lineare approssimata, dato che:

$$min \nabla f(x)^T s = \nabla f(x)^T s_j \text{ dove } j \text{ è il vertice selezionato.}$$

Poiché s è scelto in ogni partizione come il vertice che minimizza il prodotto scalare con il gradiente, e poiché il gradiente indica la direzione di maggiore incremento della funzione, scegliere il minimo in ogni simplesso porta a una riduzione locale ottimale. Essendo il Dominio D un politopo, somma di politopi, dalla programmazione lineare ci tornano utili alcuni dei teoremi fondamentali per dimostrare che il punto di ottimo è individuabile in un vertice, definito anch'esso ottimo.

Definizione combinazione convessa Dati k punti $x^1, x^2, ..., x^k \in R^n$ il punto $z \in R^n$ è combinazione convessa di $x^1, x^2, ..., x^k$ se esistono k scalari non negativi

$\lambda_1, \lambda_2, ..., \lambda_k \geq 0$ tali che $\sum_{i=1}^{k} \lambda_i = 1$ e $z = \sum_{i=1}^{k} \lambda_i x^i$

Teorema Rappresentazione dei poliedri Dato un poliedro limitato $P \subseteq \mathbb{R}^n$, e indicando con $v^1, v^2, \ldots, v^k$ i vertici di P, ogni punto $x \in P$ può essere ottenuto come combinazione convessa dei suoi vertici:

$$x = \sum_{i=1}^{k} \lambda_i v^i \quad \text{con} \quad \lambda_i \geq 0, \quad \sum_{i=1}^{k} \lambda_i = 1.$$

In altre parole, ogni punto di P si può esprimere come combinazione convessa dei suoi vertici.

Teorema Vertice Ottimo Dato un problema di PL $\min\{c^T x : x \in P\}$, se P è non vuoto e limitato, allora il problema ammette soluzione ottima e questa si trova in almeno uno dei vertici di P.

Dimostrazione: Consideriamo il minimo valore assunto dalla funzione obiettivo sui vertici e sia v^* il vertice (o uno dei vertici) in cui la funzione obiettivo assume questo valore minimo:

$$v^* = \arg\min\{c^T v : v \in V\}.$$

Per un generico punto del poliedro $x \in P$, possiamo scrivere:

$$c^T x = \sum_{i=1}^{k} \lambda_i c^T v^i \geq \sum_{i=1}^{k} \lambda_i c^T v^* = c^T v^* \sum_{i=1}^{k} \lambda_i = c^T v^*.$$

In sintesi, $\forall x \in P$, $c^T v^* \leq c^T x$, cioè v^* è una soluzione ottima corrispondente ad un vertice del poliedro P [1].

3.3 Calcolo dello Step Size

3.3.1 Introduzione

La scelta della stepsize (o passo di aggiornamento) γ_k è cruciale per la performance dell'algoritmo di Frank-Wolfe. Essa determina quanto ci si sposta verso la nuova direzione calcolata ad ogni iterazione. Esistono vari metodi per scegliere la stepsize, che possono essere classificati in metodi a stepsize fissa, dinamica o adattiva. In questa sezione, analizzeremo diverse strategie per determinare γ_k.

3.3.2 Stepsize Fisso

Uno dei modi più semplici per scegliere la stepsize è utilizzare un valore fisso:

$$\gamma_k = \gamma \textbf{ con } 0 < \gamma \leq 1$$

I Vantaggi di questo approccio riguardano sicuramente la semplicità di implementazione, dovendo scegliere unicamente il valore da assegnare a γ come abbiamo visto negli pseudo-codici precedenti, e in aggiunta, il basso costo computazionale per iterazione, evitando così ulteriori problemi di ottimizzazione. Gli svantaggi di questo approccio riguardano la convergenza che potrebbe risultare lenta, se il valore di γ è troppo piccolo, invece, se γ è troppo grande, può causare oscillazioni troppo elevate e non garantire la convergenza.

3.3.3 Step Size Lineare

Un altro approccio ampiamente utilizzato per la sua semplicità nel calcolo della step size è quello della step size lineare, dove:

$$\gamma_k = \frac{2}{k+2}$$

in cui k indica l'indice dell'iterazione corrente. Uno dei vanataggi di questo metodo riguarda la facilità di calcolo, la formula è semplice e non richiede calcoli complessi o informazioni aggiuntive sulla funzione obiettivo, inoltre, la convergenza è garantita. Uno dei problemi in cui si potrebbe incorrere è la lenta convergenza, problematica probabile poiché la step size diminuisce lentamente, l'algoritmo può convergere più lentamente rispetto ad altri metodi, specialmente nelle prime iterazioni. Un altro punto a sfavore da considerare è che non è adattivo, ovvero non tiene conto delle caratteristiche locali della funzione quadratica o dei vincoli che caratterizzano il dominio.

3.3.4 Metodo della Ricerca Lineare (Line Search)

Il metodo della ricerca lineare mira a trovare la step size ottimale γ_k che minimizza la funzione obiettivo lungo la direzione di discesa individuata ad ogni iterazione. Formalmente, per una direzione di discesa $\mathbf{d}_k = \mathbf{s}_k - \mathbf{x}_k$, il problema è:

$$\gamma_k = \arg\min_{\gamma \in [0,1]} f(\mathbf{x}_k + \gamma \mathbf{d}_k)$$

I passaggi da dover affrontare per arrivare alla soluzione di questo problema sono:

1. Definizione della direzione di discesa:

$$\mathbf{d}_k = \mathbf{s}_k - \mathbf{x}_k$$

2. Consideriamo la funzione obiettivo lungo la linea che passa per il punto corrente $\mathbf{x}_k$ nella direzione di discesa $\mathbf{d}_k$:

$$\phi(\gamma) = f(\mathbf{x}_k + \gamma \mathbf{d}_k)$$

3. Minimizzazione della funzione: Trova γ_k che minimizza $\phi(\gamma)$ nell'intervallo $[0, 1]$.

In pratica, il problema di minimizzazione cerca il valore ottimo di γ, tale che la funzione $\phi(\gamma)$ raggiunge il suo punto minimo.

Esempio Se f è una funzione quadratica $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x}$, la funzione $\phi(\gamma)$ diventa:

$$\phi(\gamma) = f(\mathbf{x}_k + \gamma \mathbf{d}_k) = \frac{1}{2}(\mathbf{x}_k + \gamma \mathbf{d}_k)^T Q(\mathbf{x}_k + \gamma \mathbf{d}_k) + \mathbf{c}^T(\mathbf{x}_k + \gamma \mathbf{d}_k)$$

Questa è una funzione quadratica in γ, che può essere minimizzata derivando rispetto a γ e risolvendo l'equazione risultante:

$$\frac{d\phi(\gamma)}{d\gamma} = (\mathbf{d}_k^T Q(\mathbf{x}_k + \gamma \mathbf{d}_k)) + \mathbf{c}^T\mathbf{d}_k = 0$$

Risolviamo per γ:

$$\gamma_k = -\frac{\mathbf{d}_k^T Q\mathbf{x}_k + \mathbf{c}^T\mathbf{d}_k}{\mathbf{d}_k^T Q\mathbf{d}_k}$$

Dimostrazioni di Convergenza per line search In questa sezione, viene fornita l'intuizione generale per la dimostrazione della convergenza lineare per l'algoritmo FW con l'utilizzo della line search, estratto da un lavoro di Martin Jaggi e Simon Lacoste Julienne [5]. Supponiamo che la funzione obiettivo f sia 'smooth' su un insieme compatto $\mathcal{D}$, cioè che il suo gradiente sia lipschitziano continuo con costante L. Definiamo anche $D := \text{diam}(\mathcal{D})$. Sia d_t la direzione in cui viene eseguita la ricerca lineare dall'algoritmo, abbiamo:

$$f(x^{(t+1)}) \leq f(x^{(t)}) + \gamma \langle \nabla f(x^{(t)}), d_t \rangle + \frac{\gamma^2}{2} L \|d_t\|^2 \quad \forall \gamma \in [0, \gamma_{\max}]. \tag{10}$$

Definiamo $r_t := -\nabla f(x^{(t)})$ e $h_t := f(x^{(t)}) - f(x^*)$ come l'errore di subottimalità. Supponendo per ora che $\gamma_{\max} \geq \gamma_t^* := \langle r_t, d_t \rangle / ([L\|d_t\|^2])$. Possiamo impostare $\gamma = \gamma_t^*$ per minimizzare il termine destro dell'equazione (10), sottrarre $f(x^*)$ da entrambi i lati e riorganizzare per ottenere un limite inferiore sul progresso:

$$h_t - h_{t+1} \geq \frac{\langle r_t, d_t \rangle^2}{2L\|d_t\|^2} = \frac{1}{2L} \langle r_t, \hat{d}_t \rangle^2, \tag{11}$$

dove utilizziamo la notazione 'hat' per denotare vettori normalizzati: $\hat{d}_t := d_t / \|d_t\|$. Sia $e_t := x^* - x^{(t)}$ il vettore errore. Per la μ-forte convessità di f, abbiamo:

$$f(x^{(t)} + \gamma e_t) \leq f(x^{(t)}) + \gamma \left\langle \nabla f(x^{(t)}), e_t \right\rangle + \frac{\gamma^2}{2} \mu \|e_t\|^2 \quad \forall \gamma \in [0, 1]. \tag{12}$$

Il lato destro è limitato inferiormente dal suo minimo come funzione di γ (non vincolata), ottenuto utilizzando $\gamma := \langle r_t, e_t \rangle / (\mu \|e_t\|^2)$. Siamo quindi liberi di utilizzare qualsiasi valore di γ sul lato sinistro e mantenere un limite valido. In particolare, usiamo $\gamma = 1$ per ottenere $f(x^*)$. Riarrangiando nuovamente, otteniamo:

$$h_t \leq \frac{\langle r_t, \hat{e}_t \rangle^2}{2\mu}, \tag{13}$$

e combinando con (11), otteniamo:

$$h_t - h_{t+1} \geq \frac{\mu}{L} \frac{\langle r_t, \hat{d}_t \rangle^2}{\langle r_t, \hat{e}_t \rangle^2}. \tag{14}$$

L'ineguaglianza (14) è piuttosto generale e valida per qualsiasi metodo di ricerca lineare nella direzione d_t. Per ottenere un tasso di convergenza lineare, dobbiamo limitare inferiormente (con una costante positiva) il termine davanti a h_t nel RHS, che dipende dall'angolo tra la direzione di aggiornamento d_t e il gradiente negativo r_t. Se assumiamo che la soluzione x^* si trovi nell'interno relativo di $\mathcal{M}$ a una distanza di almeno $\delta > 0$ dal bordo, allora $\langle r_t, d_t \rangle \geq \delta \|r_t\|$ per la direzione FW d_t^{FW}, e combinando con $\|d_t\| \leq M$, otteniamo un tasso lineare con costante $1 - \frac{\mu}{L} \left(\frac{\delta}{M}\right)^2$. D'altra parte, se x^* si trova sul bordo, allora $\langle r_t, \hat{d}_t \rangle$ diventa arbitrariamente vicino a zero per il FW standard (il fenomeno di zig-zag) e la convergenza è sublineare.

3.4 Studio della convergenza

3.4.1 Introduzione alla convergenza

Lo studio della convergenza dell'algoritmo di Frank-Wolfe è un argomento cruciale per garantire che l'algoritmo raggiunga effettivamente una soluzione ottimale. La convergenza dell'algoritmo può essere analizzata da diverse prospettive, considerando vari

aspetti come la velocità di convergenza e le condizioni necessarie per la convergenza. L'algoritmo di Frank-Wolfe converge ai valori ottimali in modo sublineare, ovvero, la differenza tra il valore della funzione obiettivo $f(x_t)$ alla t-esima iterazione e il valore ottimale $f(x^*)$ diminuisce con una velocità che non è esponenziale. Precisamente, se f è una funzione convessa differenziabile su un dominio compatto convesso D, la convergenza è garantita con una velocità $O(1/t)$.

Le condizioni da rispettare per le quali l'algoritmo di Frank-Wolfe convergerà a una soluzione ottimale riguardano convessità e differenziabilità, in particolare:

- **Funzione Convessa e Differenziabile**: La funzione obiettivo $f(x)$ deve essere convessa e differenziabile.
- **Insieme Convesso**: L'insieme dei vincoli D deve essere un insieme convesso compatto.

L'algoritmo di Frank-Wolfe ha una velocità di convergenza sublineare. Più precisamente, la distanza dalla soluzione ottimale diminuisce a un ritmo $O\left(\frac{1}{k}\right)$, dove k è il numero di iterazioni. Questo significa che l'errore obiettivo

$$f(x_k) - f(x^*)$$

decresce ad una data velocità pari a $O\left(\frac{1}{k}\right)$.

3.4.2 Duality Gap

Il duality gap è un concetto fondamentale per l'analisi della convergenza dell'algoritmo di Frank-Wolfe. Alla t-esima iterazione, il duality gap $g(x_t)$ è definito come:

$$g(x_t) = \max_{s \in D} \nabla f(x_t)^T (x_t - s_k). \tag{15}$$

Il duality gap misura la distanza tra la soluzione corrente x_t e la soluzione ottimale in termini di ottimalità duale. È un indicatore diretto della qualità della soluzione corrente: un duality gap pari a zero indica che x_t è una soluzione ottimale. L'algoritmo di Frank-Wolfe garantisce che il duality gap diminuisce ad ogni iterazione. Il duality gap può essere utilizzato per diverse finalità nell'algoritmo di Frank-Wolfe:

- **Condizione di Arresto**: L'algoritmo di Frank-Wolfe può utilizzare il duality gap come criterio di arresto. L'algoritmo si arresta quando il duality gap scende al di sotto di una soglia predefinita ϵ:

$$g(x_k) = \nabla f(x_k)^T (x_k - s_k) \leq \epsilon$$

- **Misura di Accuratezza**: Il duality gap fornisce una misura di quanto la soluzione corrente è vicina alla soluzione ottimale. Un duality gap piccolo indica che x_k è vicino all'ottimo globale.
- **Monitoraggio della Convergenza**: Il duality gap può essere monitorato durante le iterazioni per valutare la velocità di convergenza dell'algoritmo e per identificare eventuali problemi di convergenza.

Se x_t è la soluzione alla t-esima iterazione dell'algoritmo di Frank-Wolfe e x^* è una soluzione ottimale, allora

$$f(x_t) - f(x^*) \leq g(x_t). \tag{16}$$

Proof. Poiché x^* è ottimale, abbiamo

$$\nabla f(x_t)^T(x^* - x_t) \geq 0. \tag{17}$$

D'altra parte, per la definizione di $g(x_t)$,

$$g(x_t) = \max_{s \in D} \nabla f(x_t)^T(x_t - s) \geq \nabla f(x_t)^T(x_t - x^*). \tag{18}$$

Pertanto,

$$f(x_t) - f(x^*) \leq \nabla f(x_t)^T(x_t - x^*) \leq g(x_t). \tag{19}$$

□

3.5 Convergenza Sublineare

Possiamo dimostrare che il duality gap $g(x_t)$ diminuisce con una velocità di $O(1/t)$. Questo risultato è importante perché implica che l'algoritmo di Frank-Wolfe raggiunge una soluzione approssimativa con una qualità garantita dopo un numero finito di iterazioni.

Se x_t è la soluzione alla t-esima iterazione dell'algoritmo di Frank-Wolfe, allora

$$g(x_t) \leq \frac{2L \cdot (\text{diam}(D))^2}{t+2}, \tag{20}$$

dove L è la costante di Lipschitz del gradiente di f e $\text{diam}(D)$ è il diametro del dominio D. Il diametro di un dominio D è una misura della massima distanza tra due punti all'interno del dominio. Formalmente, se D è un insieme nel quale si sta operando (ad esempio, un insieme convesso, come in questo caso), il diametro $\text{diam}(D)$ è definito come:

$$\text{diam}(D) = \sup_{x,y \in D} \|x - y\|$$

dove $\|\cdot\|$ denota una norma appropriata (spesso la norma euclidea) e sup rappresenta il supremo (o il massimo) tra tutte le distanze possibili tra coppie di punti x e y appartenenti al dominio D. La dimostrazione si basa sulla proprietà di Lipschitz del gradiente e sull'aggiornamento iterativo dell'algoritmo di Frank-Wolfe. Senza perdita di generalità, consideriamo le iterazioni t e $t+1$. Per la proprietà di Lipschitz del gradiente [2],

$$\|\nabla f(x_t) - \nabla f(x_{t+1})\| \leq L\|x_t - x_{t+1}\|. \tag{21}$$

Dato che x_{t+1} è una combinazione convessa di x_t e di un punto $s_t \in D$,

$$x_{t+1} = (1 - \gamma_t)x_t + \gamma_t s_t, \tag{22}$$

dove γ_t è un passo scalare. Utilizzando la disuguaglianza triangolare e il fatto che $s_t \in D$,

$$\|x_t - x_{t+1}\| = \gamma_t\|x_t - s_t\| \leq \gamma_t \cdot \text{diam}(D). \tag{23}$$

Quindi,

$$\|\nabla f(x_t) - \nabla f(x_{t+1})\| \leq L\gamma_t \cdot \text{diam}(D). \tag{24}$$

Dato che $\gamma_t = \frac{2}{t+2}$ per l'algoritmo di Frank-Wolfe,

$$g(x_t) \leq \frac{2L \cdot (\text{diam}(D))^2}{t+2}. \tag{25}$$

Per il calcolo delle costanti nella disequazione (20) possiamo rifarci alla formula (6) (il massimo autovalore), per il calcolo della costante L, mentre per calcolare il diametro del dominio D è possibile trovare la massima distanza dei vertici che vincolano il dominio, anche se computazionalmente costoso.

$$\mathrm{diam}(D) = \max_{v_i \in S_a, v_j \in S_b} \left(\sqrt{\sum_{k=1}^{d} (v_i^{(k)} - v_j^{(k)})^2} \right)$$

4 Parte Sperimentale

Oggetto di studio nella convergenza dell'algoritmo sono stati i metodi adottati nel calcolo dello stepsize, principalmente tre, quelli presi in considerazione, tra cui:
- stepsize fisso
- linear stepsize
- exact line search

spiegati in modo dettagliato nel capitolo precedente. Nella parte sperimentale, tanta attenzione è stata posta anche nella generazione dei dati, definendo comunque un 'seed' a cui affidare la randomicità dei dati', assicurando che i vincoli del dominio (quindi la non negatività e la somma degli elementi delle partizioni sia = 1) e le proprietà della matrice Q fossero mantenute (convessa e semidefinita positiva). Per questo motivo tra gli input iniziali, per la generazione dei dati, troviamo quello che è la dimensione del kernel della matrice, fissato al 10% della dimensionsionalità del problema, spectral radius (maggiore autovalore) , dimensionalità della matrice, il numero di simplessi che condizionano il dominio, la soglia di tolleranza (stop condition $\epsilon < 1e^{-6}$) e il numero di iterazioni massime. I risultati riportati nelle Tabelle sono il risultato della media di 3 simulazioni per ogni terna di parametri provata (n,k e spectral radius), con la generazione dei dati affidata ad ogni simulazione ad un 'seed' diverso. Di seguito sono riportati lo pseudocodice utilizzato nella generazione dei dati e i risultati ottenuti delle simulazioni che mostrano il comportamento dell'algoritmo di Frank-Wolfe, aggiungendo anche alcune dimostrazioni grafiche di quanto riportato nelle tabelle. Essendo l'obiettivo dello studio mostrare la differenza nei risultati, nei tempi e nelle performance tra i diversi algoritmi di calcolo dello stepsize per l'algoritmo FW, ogni terna di parametri è stata passata e valutata per ognuno degli algoritmo prima citato. I risultati ottenuti con stepsize fisso non sono stati riportati poichè non accennavano a raggiungimento di convergenza, fino al raggiungimento del numero di iterazioni massime, bloccando cosi l'escuzione dell'algoritmo senza accenni di convergenza.

4.1 Generazione dei Dati

La generazione della matrice Q, come indicato nello pseudocodice si basa su una scelta accurata degli autovalori, determinati dai parametri $\lambda_{\max}$, $\lambda_{\min}$, del kernel dim_Ker, oltre che alla dimensionalità . Gli autovalori di una matrice sono cruciali nel determinare le sue proprietà, come la positività definita. Il parametro $\lambda_{\max}$ rappresenta il raggio spettrale, ovvero il valore massimo degli autovalori non nulli di Q. Questo parametro è importante poiché controlla la massima variazione lungo una direzione della matrice. Il valore minimo degli autovalori non nulli è fissato da $\lambda_{\min}$, che assicura che non vi siano autovalori troppo piccoli, evitando una condizionabilità eccessiva della matrice. La dimensione del kernel, dim_Ker, gioca un ruolo fondamentale nella determinazione del rango di Q.

- **Kernel massimo**: Se dim_Ker $= n$, tutti gli autovalori sono nulli, quindi Q è la matrice nulla.
- **Kernel di dimensione** $n-1$: Se dim_Ker $= n-1$, Q ha un solo autovalore non nullo, pari a $\lambda_{\max}$, rendendo Q una matrice di rango 1. Questo caso rappresenta una situazione in cui esiste una sola direzione di variazione significativa.
- **Kernel ridotto**: Quando dim_Ker $< n-1$, la matrice ha diversi autovalori non nulli, distribuiti tra $\lambda_{\min}$ e $\lambda_{\max}$.

La gestione degli autovalori , nel caso di $\lambda_{\max}$,permette di sapere quale sia e quanto vale la direzione di massima crescita della forma quadratica e di capire questo parametro quanto possa incidere sulla convergenza del problema. La scelta di mantenere fisso il valore di $\lambda_{\min} = 1$ viene dal fatto che data la presenza di autovalori uguali a zero pari alle dimensioni del kernel, che indicano quante direzioni 'piatte' sono presenti, per evitare numeri positivi ma ben prossimi allo zero, viene imposta la condizione che il più piccolo autovalore diverso da zero sia pari a 1.

Controllo della Densità della Matrice

Oltre alla scelta degli autovalori, un altro aspetto fondamentale nella generazione della matrice Q è il controllo della sua *densità*, specificato dal parametro 'density'. La densità di una matrice si riferisce alla frazione di elementi non nulli rispetto al totale degli elementi. Il parametro density consente di variare la struttura della matrice Q, rendendola più o meno "sparse", ovvero con più zeri:

- **Densità piena (density=1)**: Quando density=1, la matrice Q è completamente densa e non contiene elementi nulli, ad eccezione di quelli che risultano direttamente dalle scelte sugli autovalori. In questo caso, Q rappresenta una situazione standard per problemi di ottimizzazione.
- **Densità parziale ($0 \leq$ density ≤ 1)**: Se la densità è impostata su un valore inferiore a 1, la matrice Q diventa *sparsa*, con molti elementi uguali a zero. In questo scenario, la matrice descrive un sistema in cui solo alcune variabili sono tra loro correlate, rendendo più semplice ed efficiente l'analisi e l'ottimizzazione.
- **Densità nulla (density=0)**: Quando density=0, tutti gli elementi di Q sono nulli, ad eccezione di quelli eventualmente derivanti dal kernel definito. Questo può essere utile per testare il comportamento degli algoritmi in condizioni limite, in cui la matrice non rappresenta nessuna interazione significativa tra le variabili.

In conclusione, la generazione della matrice Q avviene non solo selezionando accuratamente gli autovalori per assicurare le proprietà richieste (ad esempio, semidefinitività positiva e vincoli), ma anche controllando la densità della matrice per modellare la complessità del problema. Queste scelte permettono di testare il comportamento degli algoritmi in diversi scenari, da quello denso e pienamente correlato, a quello sparso con poche interazioni significative tra le variabili.

Algorithm 4 Generazione dell'istanza dei dati (`generate_instance`)

Input: n: dimensione della matrice, K, `dim_Ker`, `spectral_radius`, $\lambda_{\min}$, `density`, `seed`
Output: Q_{nxn}, q_{1xn} P_{Kxn}, `Ik`:

Validazione degli input
if $K < 1$ **or** $K > n$ **then**
    Errore: "Il numero di simplessi K deve essere tra 1 e n"
end if
if `dim_Ker` < 0 **or** `dim_Ker` $> n$ **then**
    Errore: "La dimensione del kernel deve essere tra 0 e n"
end if
if `dim_Ker` $< n$ **and** `spectral_radius` ≤ 0 **then**
    Errore: "Il raggio spettrale deve essere maggiore di 0"
end if
if $\lambda_{\min} \leq 0$ **or** $\lambda_{\min} >$ `spectral_radius` **then**
    Errore: "Il valore minimo degli autovalori deve essere tra 0 e `spectral_radius`"
end if
if `density` < 0 **or** `density` > 1 **then**
    Errore: "La densità deve essere tra 0 e 1"
end if

Generazione della matrice Q
Inizializza il seme del generatore casuale con `seed`
if `dim_Ker` $= n$ **then**
    `rc` $\leftarrow$ `zeros`(n) ▷ Kernel massimo
else if `dim_Ker` $= n - 1$ **then**
    `rc` $\leftarrow$ [`spectral_radius`] $+ [0] \times$ (`dim_Ker`)
else
    `rc` $\leftarrow [\lambda_{\min} + ($`spectral_radius` $- \lambda_{\min}) \cdot x \; \forall \; x \in [1] +$ `rand`$(n -$ `dim_Ker` $- 2) + [0]]$
    `rc` $\leftarrow$ `rc` $+ [0] \times$ `dim_Ker`
end if
if `density` $= 1$ **then**
    `S` $\leftarrow$ `diag`(`rc`)
    `U` $\leftarrow$ `orth`(`rand`(n, n))
    $Q \leftarrow$ `U` $\cdot$ `S` $\cdot$ `U`^{T}
    $Q \leftarrow (Q + Q^T)/2$ ▷ Assicura la simmetria numerica
else
    $Q \leftarrow$ `sparse_random`$(n, n,$ `density=density`$)$
end if

Generazione della matrice dei vincoli P e delle partizioni degli indici
P, `indices` $\leftarrow$ `generate_constraints`$(n, K,$ `seed`$)$

Generazione del vettore q
$q \leftarrow$ `rand`(n)

Ritorna $Q, q, P,$ `Ik`

Algorithm 5 Generazione dei vincoli (`generate_constraints`)

Input: n, K, `seed`
Output: P: matrice dei vincoli, I_k: partizione degli indici
`indices` $\leftarrow$ `np.arange`(n)
`np.random.shuffle(indices)` ▷ Rimescola casualmente gli indici
$I_k \leftarrow$ `np.array_split`(`indices`, K) ▷ Divide gli indici in K gruppi

Costruzione della matrice P
$P \leftarrow$ `zeros`(K, n) ▷ Matrice con K righe e n colonne
for $j \leftarrow 0$ **to** $K - 1$ **do**
    `pos` $\leftarrow I_k[j]$
    $P[j,$ `pos`$] \leftarrow 1$
end for

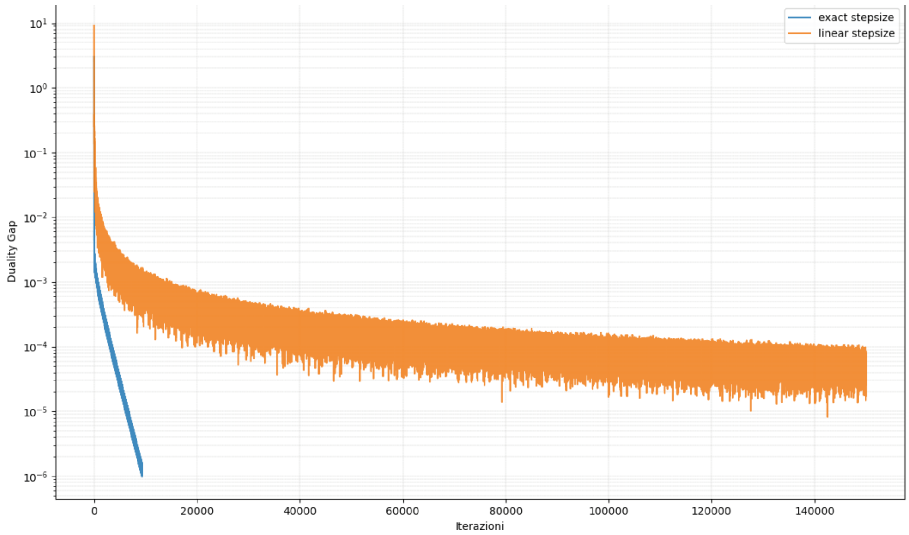
Ritorna P, I_k

4.2 Risultati Sperimentali

I valori 'Iterazioni medie', 'Tempo medio' e 'Gap di Dualità medio' sono ottenuti dalla media di 3 simulazioni con 3 diversi seed per la generazione dei dati.

Raggio Spettrale	Metodo	K	Iterazioni Medie	Tempo Medio (s)	Gap di Dualità Medio
10	Exact	1	12 126	3,1564	$1,322 \times 10^{-7}$
	Linear		150 000	5,6417	$4,217 \times 10^{-5}$
100	Exact	1	102 265	4,6823	$1,603 \times 10^{-5}$
	Linear		150 000	5,8727	$3,530 \times 10^{-4}$
1000	Exact	1	105 258	4,9157	$1,301 \times 10^{-4}$
	Linear		150 000	5,6108	$4,294 \times 10^{-3}$

Table 1: Risultati medi per diversi metodi e raggi spettrali con $n = 10$, $K = 1$, densità pari a 1



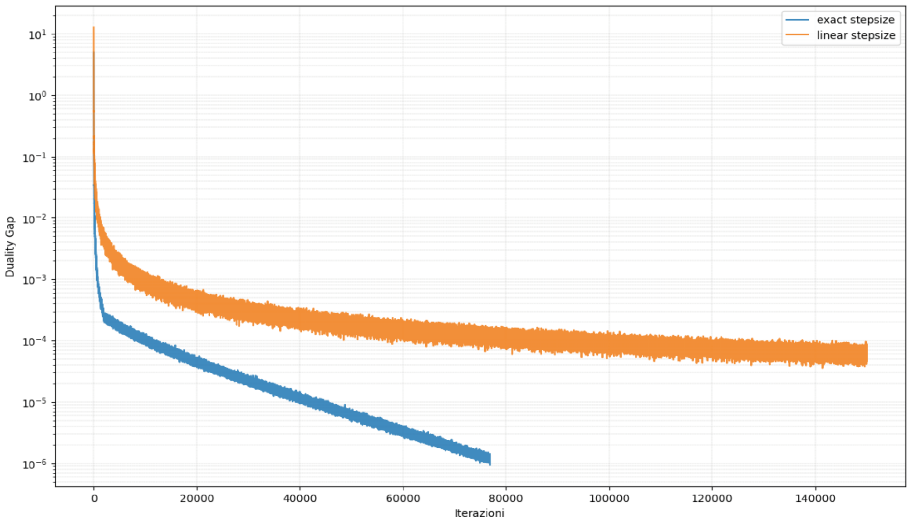


Figure 3:
Dall alto troviamo i risultati di Convergenza con i seguenti parametri
n = 10, k = 1, sp = 10
n = 100, k = 10, sp = 10

Metodo	Raggio Spettrale	K	Iterazioni Medie	Tempo Medio (s)	Gap di Dualità Medio
exact	10	1	125 970	6.1871	3.592455e-06
linear		1	150 000	6.5083	5.535713e-05
exact		10	150 000	11.8894	5.402134e-04
linear		10	150 000	11.1181	5.496540e-04
exact	100	1	150 000	7.6514	5.162826e-04
linear		1	150 000	6.7806	5.778944e-04
exact		10	150 000	12.3182	5.432619e-03
linear		10	150 000	11.4704	5.742796e-03
exact	1000	1	150 000	12.0800	5.662196e-02
linear		1	150 000	11.4209	5.415083e-02
exact		10	150 000	12.7129	5.621530e-02
linear		10	150 000	11.2991	5.497317e-02

Table 2: Average results for different methods, spectral radii, and values of K

Metodo	Raggio Spettrale	K	Tempo Medio (s)	Iterazioni	Gap di Dualità Medio
exact	10	1	327.5	50 000	0.000175
linear		1	308	50 000	0.000215
exact		10	354	50 000	0.001847
linear		10	322.4	50 000	0.002038
exact		100	896.26	50 000	0.017411
linear		100	902.62	50 000	0.017024
exact	100	1	311.6	50 000	0.000888
linear		1	597	50 000	0.001916
exact		10	315.2	50 000	0.016304
linear		10	303.1	50 000	0.018238
exact		100	925.70	50 000	0.167633
linear		100	896.37	50 000	0.166024
exact	1000	1	793.2	50 000	0.012636
linear		1	769.5	50 000	0.017788
exact		10	311.8	50 000	0.149701
linear		10	774.7	50 000	0.173816
exact		100	780.13	50 000	1.602874
linear		100	331	50 000	1.619630

Table 3: risutati per n = 1000 e diversi valori di K e raggio spettrale

Per l'analisi della dimensionalità n = 1000 non è stata riportata la media di 3 simulazioni e il numero di iterazioni massime è stato diminuito a 50 000 per questioni di potenza computazionale e tempi di iterazione. A parità di iterazioni rispetto ai precedenti esperimenti riportati si presume un duality gap sicuramente minore per tutte le simulazioni e quindi una precisione maggiore ma tempi molto più elevati per la singola simulazione come è possibile visionare dalla seguente immagine che indica

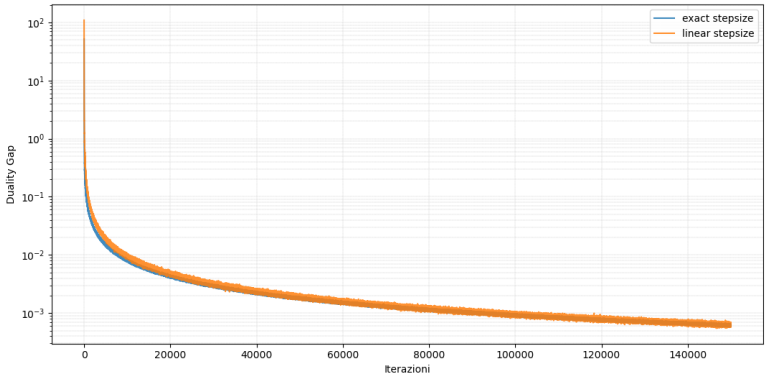


Figure 4: Risultati Convergenza per n = 1000, k = 10 , sp = 10

un Duality Gap = 0.000598 raggiunto a parità di iterazioni.

4.3 Confronto delle Strategie di Convergenza

Per analizzare le prestazioni delle due strategie, abbiamo condotto una serie di esperimenti variando la dimensione del problema n, il parametro K e il raggio spettrale della matrice Q associata alla funzione obiettivo quadratica:

$$f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x} + \mathbf{q}^T\mathbf{x}. \tag{27}$$

Dalle simulazioni effettuate con $n = 10$, 100 e 1000, emergono interessanti osservazioni sul comportamento dell'algoritmo.

4.3.1 Impatto della dimensionalità $n = 10, n = 100, n = 1000$

Per problemi di piccola dimensione, l'algoritmo di Frank-Wolfe con ricerca lineare esatta mostra una convergenza più rapida rispetto alla regola del passo lineare. In particolare, per $K = 1$ e raggio spettrale basso, il numero di iterazioni necessarie per raggiungere un *gap duale* soddisfacente è significativamente inferiore quando si utilizza la ricerca lineare esatta. Questo risultato è attribuibile al fatto che, in dimensioni ridotte, il costo computazionale aggiuntivo della ricerca lineare esatta è trascurabile rispetto al beneficio ottenuto in termini di convergenza. Per problemi di dimensione media e grande, il costo computazionale della ricerca lineare esatta diventa più rilevante. Nonostante ciò, l'algoritmo con ricerca lineare esatta continua a offrire vantaggi in termini di precisione della soluzione, anche se all'aumentare dei vincoli e della dimensionalità c'è da valutare un trade off tra costo computazionale e performance. In particolare, per $n = 1000$, l'algoritmo raggiunge il numero massimo di iterazioni consentito senza convergere completamente. Questo comportamento suggerisce che, in problemi di grandi dimensioni, potrebbe essere necessario adottare strategie ibride o modificare la tolleranza sul gap duale per ottenere risultati più soddisfacenti.

4.3.2 Effetto del Raggio Spettrale

All'aumentare del raggio spettrale, la matrice Q diventa più condizionata, influenzando negativamente la convergenza dell'algoritmo. Sia la ricerca lineare esatta che la regola del passo lineare mostrano un incremento nel numero di iterazioni necessarie per raggiungere una soluzione di qualità accettabile. Tuttavia, la ricerca lineare esatta mantiene un vantaggio in termini di precisione della soluzione. Questo suggerisce che l'adattabilità della dimensione del passo alle caratteristiche locali di f è particolarmente benefica in presenza di problemi più condizionati. I risultati ottenuti e le valutazioni fatte considerando questo parametro singolarmente andrebbero prese valutando anche il contesto, e il numero di altri vincoli utilizzati. Una valutazione a se stante sul raggio spettrale non darebbe una visione globale di questo problema.

4.3.3 Influenza del Parametro K

Il parametro K, che può essere interpretato come un indicatore della complessità del vincolo o della dimensione dell'insieme ammissibile, ha un impatto significativo sulla convergenza. Per valori crescenti di K, il numero di iterazioni richieste dall'algoritmo aumenta per entrambe le strategie di calcolo dello stepsize. Anche in questo caso, la ricerca lineare esatta tende a fornire soluzioni con gap duale inferiore, evidenziando la sua efficacia nel trattare problemi più complessi.

4.4 Considerazioni

Lo studio condotto e i risultati ottenuti evidenziano come la scelta della dimensione del passo nell'algoritmo di Frank-Wolfe influenzi in modo significativo la convergenza e l'efficienza computazionale. La ricerca lineare esatta, sebbene più costosa per iterazione, offre vantaggi in termini di convergenza e accuratezza della soluzione, specialmente in problemi di piccole e medie dimensioni.

La regola del passo lineare, con $\gamma_t = \dfrac{2}{t+2}$, rappresenta un'alternativa più semplice e meno onerosa dal punto di vista computazionale, ma può richiedere un numero maggiore di iterazioni per raggiungere una soluzione di qualità comparabile.

In pratica, la scelta tra le due strategie dovrebbe considerare il bilanciamento tra il costo computazionale per iterazione e il numero totale di iterazioni necessarie. Per problemi di piccola dimensione o quando è richiesta un'elevata precisione, la ricerca lineare esatta può essere preferibile. Al contrario, per problemi di grandi dimensioni o quando le risorse computazionali sono limitate, la regola del passo lineare potrebbe risultare più efficiente.

È importante sottolineare che ulteriori miglioramenti possono essere ottenuti attraverso l'uso di varianti avanzate dell'algoritmo di Frank-Wolfe, come l'*Away-Step Frank-Wolfe* o l'*Accelerated Frank-Wolfe*, che introducono meccanismi per accelerare la convergenza e gestire in modo più efficace le peculiarità del problema.

4.5 Impatto della densità sulla convergenza

Per questi risultati è stato utilizzato lo stesso metodo dell'analisi precedente, calcolare la media delle iterazioni, del tempo di iterazione e del duality gap per ogni settaggio dei parametri

Metodo	**Raggio Spettrale**	**K**	**Iterazioni Medie**	**Tempo Medio (s)**	**Gap di Dualità**
Exact	1	10	55 993	3.0900	9.98e-07
Linear		10	150 000	7.6945	1.40e-03
Exact		100	150 000	8.9428	2.08e-03
Linear		100	150 000	8.1576	1.84e-03
Exact		1000	150 000	9.1164	4.44e-03
Linear		1000	150 000	7.9632	4.17e-03
Exact	10	10	8 515	0.8846	9.92e-07
Linear		10	150 000	14.6228	1.17e-02
Exact		100	150 000	15.9841	1.74e-02
Linear		100	150 000	14.6763	1.66e-02
Exact		1000	150 000	16.6758	4.12e-02
Linear		1000	150 000	16.8795	4.12e-02

Table 4: Risultati calcolati per n = 100 con densità = 0.1.

Metodo	**Raggio Spettrale**	**K**	**Iterazioni Medie**	**Tempo Medio (s)**	**Gap di Dualità**
Exact	1	10	119 702	16.28	9.96e-07
Linear		10	150 000	18.41	1.29e-03
Exact		100	150 000	23.41	1.77e-03
Linear		100	150 000	23.40	1.79e-03
Exact		1000	150 000	28.73	5.71e-03
Linear		1000	150 000	25.04	5.45e-03
Exact	10	10	15 179	20.22	1.17e-06
Linear		10	150 000	47.09	1.30e-02
Exact		100	150 000	45.43	1.37e-02
Linear		100	150 000	44.91	1.46e-02
Exact		1000	150 000	37.47	2.84e-02
Linear		1000	150 000	35.51	2.41e-02

Table 5: Risultati calcolati per n = 100 con densità = 0.5.

Per questa analisi, sono state eseguite simulazioni utilizzando l'algoritmo Frank-Wolfe su un problema di dimensione $n = 100$ e un kernel di dimensione $K = 10$, variando la densità ρ del problema. Sono stati considerati due valori per la densità: $\rho = 0.1$ e $\rho = 0.5$.

I parametri utilizzati nelle simulazioni sono il raggio spettrale (*raggio*) e il numero di iterazioni massime (K).

4.5.1 Convergenza per Densità $\rho = 0.1$

I risultati delle simulazioni con densità $\rho = 0.1$ sono riportati nella Tabella 4. È evidente che, sia per il metodo *exact line search* che per la *linear search*, l'aumento del raggio spettrale porta a un incremento del numero di iterazioni e del tempo di esecuzione. Tuttavia, osserviamo che il metodo *exact line search* risulta più efficiente in termini di qualità della soluzione (gap di dualità), a fronte di tempi di esecuzione più lunghi. Il metodo *linear search*, invece, pur risultando più rapido, converge con un gap di dualità leggermente superiore.

Un esempio evidente è il caso con raggio spettrale pari a 1000, dove il metodo *exact line search* raggiunge un gap di dualità di $5.71e-03$, mentre il metodo *linear* si attesta su $4.12e-02$, evidenziando una differenza significativa nella qualità della soluzione.

4.5.2 Convergenza per Densità $\rho = 0.5$

Passando a una densità più elevata ($\rho = 0.5$, Tabella 5), i risultati mostrano che, nonostante il numero di iterazioni rimanga invariato, il tempo di esecuzione cresce in maniera significativa. La maggiore densità del problema implica una maggiore complessità computazionale, il che si riflette nei tempi medi di esecuzione, che raddoppiano o triplicano rispetto a quelli ottenuti con $\rho = 0.1$.

Il gap di dualità risulta comunque migliore nel metodo *exact line search*, ma la differenza con il metodo *linear search* diventa meno pronunciata. Ad esempio, per un raggio spettrale pari a 100, il metodo *exact* ottiene un gap di dualità di $1.37e-02$ contro $1.46e-02$ del metodo *linear*.

4.5.3 Impatto della Densità sulla Convergenza

L'influenza della densità ρ sul comportamento dell'algoritmo è un aspetto cruciale da analizzare. Con densità inferiori a 1, come nel caso di $\rho = 0.1$, il problema è meno complesso, permettendo una convergenza più rapida e tempi di esecuzione inferiori, almeno per il caso in cui k = 10, i tempi medi per iterazioni sono più bassi. Tuttavia, l'incremento della densità, come osservato per $\rho = 0.5$, comporta un aumento sostanziale del tempo necessario per l'algoritmo a convergere. Il numero di iterazioni rimane pressoché costante, suggerendo che la complessità del problema agisce principalmente sui calcoli interni piuttosto che sulla struttura generale del problema. Il metodo *exact line search* risulta essere sempre più accurato in termini di gap di dualità, ma a fronte di tempi di esecuzione superiori, specialmente per densità più elevate. D'altra parte, il metodo *linear search* mostra di essere più veloce, ma con una precisione leggermente inferiore, sebbene questa differenza tenda a ridursi all'aumentare della densità.

Dal confronto tra il metodo *exact line search* e la *linear search*, possiamo trarre le seguenti conclusioni:

- Il metodo *exact line search* garantisce un gap di dualità inferiore, ossia soluzioni più accurate, ma richiede tempi di esecuzione più elevati, specialmente per problemi con alta densità.
- Il metodo *linear search* offre un compromesso tra velocità e accuratezza. Sebbene tenda a convergere più velocemente, le soluzioni finali possono presentare un gap di dualità leggermente superiore.

- L'aumento della densità del problema ha un impatto diretto sul tempo di esecuzione per entrambi i metodi, ma non altera significativamente il numero di iterazioni necessarie alla convergenza.

Questo studio evidenzia come la scelta tra i due metodi dipenda dal *trade-off* tra accuratezza e tempo di esecuzione richiesto per risolvere problemi di ottimizzazione convessa con l'algoritmo Frank-Wolfe.

4.6 Confronto con Solutor-General Purpose

Per validare il lavoro svolto fin ora è stato preso in considerazione GUROBI, un 'Optimizer' e risolutore, poiché utilizza l'ottimizzazione matematica per calcolare la risposta a un problema. Di seguito diversi confronti, calcolando il punto ottimo utilizzando l'algoritmo Frank Wolfe con l'exact line search e la soluzione ottima del Solutor.

n	k	Raggio	**F(x*)**	**F(xFW)**	**Relative Gap**	**Tempo FW**	**Tempo Gurobi**
10	1	10	1.1837288224	1.1837288225	8.1774×10^{-11}	0.03s	0.01s
		100	8.1214194566	8.1214194568	2.411×10^{-11}	1.7s	0.01s
		1000	73.451298	73.451855	7.58×10^{-6}	8.2s*	0.01s
100	1	10	0.342479501	0.3424795121	3.222×10^{-8}	5.76s	0.04s
		100	1.40032995	1.40036995	2.849×10^{-5}	52s*	0.04s
		1000	10.269983	10.27163	$1,6 \times 10^{-4}$	55s*	0.05s
	10	10	14.355637	14.355744	1.174×10^{-5}	88s*	0.04s
		100	103.302790	103.30384	1.545×10^{-5}	93s*	0.03s
		1000	982.359052	982.36941	$1,59 \times 10^{-5}$	88s*	0.04s
1000	1	10	0.09325322	0.093284222	3.32×10^{-4}	861s	0.79s
		100	0.31490913	0.3150309	3.86×10^{-4}	826s	0.81s
		1000	1.369360	1.371863	$1,65 \times 10^{-3}$	988s	0.85s
	10	10	3.294029	3.294336	9.32×10^{-5}	774s	0.81s
		100	13.739673	13.742529	2.07×10^{-4}	760s	0.85s
		1000	103.793977	103.821412	$2,6 \times 10^{-4}$	825s	0.77s
	100	10	141.071802	141.07475	$2,09 \times 10^{-5}$	891s	0.81 s
		100	1042.544128	1042.572331	2.704×10^{-5}	922s	0.73s
		1000	9996.268833	9996.547393	2.786×10^{-5}	2203s	2.5s

Table 6: Confronto dei risultati tra Frank Wolfe e Gurobi
I tempi indicati con '' hanno avuto un maxiter = 250 000 per valutare miglioramenti in precisione*

La tabella riporta il confronto tra l'algoritmo Frank-Wolfe (indicato come x_{FW}) e un solutore *general purpose* (con soluzione ottimale indicata come x^*), su diversi scenari definiti dalla dimensione n, dal parametro κ e dal raggio spettrale della matrice Q. Le colonne della tabella mostrano i valori della funzione obiettivo $F(x^*)$ e $F(x_{\text{FW}})$, insieme al **Relative Gap** tra questi valori e i tempi di esecuzione per ciascun metodo.

4.6.1 Definizione del *Relative Gap*

Il **Relative Gap** è definito come:

$$\text{Relative Gap} = \frac{|F(x_{\text{FW}}) - F(x^*)|}{|F(x^*)|},$$

e rappresenta la differenza relativa tra il valore della funzione obiettivo ottenuto con l'algoritmo Frank-Wolfe e il valore ottimale $F(x^*)$ fornito dal solutore general purpose. Questo indicatore è fondamentale per valutare l'accuratezza dell'algoritmo rispetto all'ottimo globale. Un **Relative Gap** basso indica che l'algoritmo produce soluzioni molto vicine all'ottimo, garantendo affidabilità nei risultati. L'utilizzo del **Relative Gap** è quindi cruciale per comprendere se un algoritmo approssimato come Frank-Wolfe sia adeguato per risolvere un determinato problema, soprattutto quando si confronta con solutori esatti come Gurobi.

4.6.2 Analisi dei risultati

- **Accuratezza dell'algoritmo Frank-Wolfe**:

  L'algoritmo Frank-Wolfe mostra un'ottima precisione su tutti i casi analizzati, in particolare nei casi con basso condizionamento e numero di vincoli ridotto (valori bassi di κ e raggio spettrale). Il **Relative Gap**, che esprime la differenza relativa tra $F(x^*)$ e $F(x_{\text{FW}})$, è molto basso, con valori che variano da 10^{-11} a 10^{-4}. Questo indica che Frank-Wolfe è in grado di fornire una soluzione molto vicina all'ottimo globale.

  In particolare, per $n = 10$ e alcuni casi di $n = 100$, il **Relative Gap** rimane compreso tra 10^{-11} e 10^{-6}, evidenziando che l'algoritmo raggiunge ottime performance in questi casi. Tuttavia, per $n = 1000$ e con condizioni di vincolo più severe (valori elevati di κ e del raggio spettrale), il Relative Gap aumenta leggermente, pur mantenendosi entro valori accettabili (fino a 10^{-4}, in un caso 10^{-3}). Questo comportamento riflette un'accuratezza ridotta in scenari più complessi.

- **Influenza della dimensione** n:

  La dimensione del problema, n, ha un impatto significativo sui tempi di esecuzione. Per $n = 10$, l'algoritmo Frank-Wolfe converge in tempi molto ridotti, con esecuzioni che durano pochi secondi. Tuttavia, all'aumentare della dimensione, i tempi di esecuzione aumentano in modo non lineare. Per $n = 100$, i tempi di esecuzione si attestano tra 5.76 secondi e 88 secondi, mentre per $n = 1000$, il tempo di calcolo aumenta considerevolmente, raggiungendo un massimo di 2203 secondi.

  Nonostante l'aumento dei tempi, l'algoritmo continua a mantenere un Relative Gap accettabile anche su dimensionalità più grandi, garantendo una buona precisione. Al contrario, Gurobi mantiene tempi di esecuzione più contenuti rispetto a Frank-Wolfe, soprattutto per problemi di dimensioni elevate.

- **Effetto del raggio spettrale**:

  Il raggio spettrale della matrice Q, cioè il massimo autovalore $\lambda_{\max}(Q)$, ha un'influenza diretta sia sui valori della funzione obiettivo che sulla precisione della soluzione di Frank-Wolfe. Un raggio spettrale elevato, ad esempio $\lambda_{\max}(Q) = 1000$, comporta un aumento del Relative Gap, indicando che vincoli più stringenti e problemi più condizionati, l'algoritmo impiega più tempo a convergere a una soluzione ottimale, aumentando anche il tempo medio per singola iterazione.

  Questo risultato è in linea con le aspettative, poiché il raggio spettrale influenza il condizionamento del problema, rendendo più difficoltoso per Frank-Wolfe raggiungere rapidamente una soluzione molto accurata.

- **Impatto del parametro** κ:

  Il parametro κ, che rappresenta il numero di simplessi che vincolano il dominio del problema, ha un'influenza significativa sulle performance dell'algoritmo. Aumentando κ, si incrementa il numero di vincoli nel problema, il che rallenta la convergenza.

  Dalla tabella si osserva che i tempi di esecuzione di Frank-Wolfe aumentano considerevolmente rispetto ai casi con $\kappa = 1$. In questi scenari, il Relative Gap aumenta leggermente, ma rimane comunque accettabile. Questo conferma che un numero maggiore di vincoli riduce la velocità di convergenza e la precisione dell'algoritmo, soprattutto nei problemi di grandi dimensioni.

Dalla tabella emerge che l'algoritmo Frank-Wolfe, pur richiedendo tempi di esecuzione più lunghi rispetto a Gurobi nei casi più complessi, riesce a mantenere una buona precisione, come evidenziato dal **Relative Gap** molto basso anche nei casi con condizioni di vincolo severe. L'influenza dei parametri n, κ e del raggio spettrale $\lambda_{\max}(Q)$ dimostra che l'algoritmo è capace di gestire problemi complessi, sebbene a costo di un tempo di esecuzione crescente.

La velocità di Gurobi nel risolvere i problemi rimane costante indipendentemente dalla dimensionalità del problema e dai vincoli imposti, mantenendo tempi di esecuzione sotto il secondo, ad eccezione dell'ultimo caso (il più complesso), che ha richiesto 2.5 secondi ma 'bruciando' comunque sul tempo il Frank-Wolfe, superando anche in termini di efficienza e precisione.

References

[1] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Komei Fukuda. From the zonotope construction to the minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4), 2004.

[4] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International conference on machine learning*. PMLR, 2013.

[5] Simon Lacoste-Julien and Martin Jaggi. On the convergence of the frank-wolfe algorithm. *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015.

[6] Xin-She Yang. *Optimization techniques and applications with examples*. John Wiley & Sons, 2018.