Ricerca Operativa e Pianificazione delle Risorse

Fabio Ferrario
@fefabo

Sara Angeretti
@Sara1798

2022/2023

Indice

Capitolo 1

Introduzione

1.1 Il corso

Il corso di ROPR verrà svolto da:

- Fabio Antonio Stella
- Guglielmo Lulli

Il testo di riferimento è *Frederick S. Hiller and Gerald J. Lieberman, Ricerca Operativa, McGraw-Hill, nona edizione, 2010.*

Il programma

Il Syllabus indica:

- Introduzione: storia-motivazione-esempi
- Ottimizzazione non lineare.
    1. Ottimizzazione di funzioni non lineari ad una variabile: ricerca dicotomia-metodo Bisezione- metoto Newton.
    2. Ottimizzazione di funzioni non lineari multivariate: metodo Gradiente - metodo Newton
    3. Ottimizzazione non lineare vincolate: Condizioni di Karush-Kuhn-Tucker.
- Ottimizzazione lineare e intera.
    1. Introduzione alla programmazione linare (PL): Proprietà dei problemi di PL, strategie di modellizzazione.

- Soft Computing per l'ottimizzazione.
    1. Algoritmi evolutivi
    2. Reti neurali ed SVM

Gli argomenti in specifico

- Modelli nella Ricerca Operativa
- Introduzione alla Programmazione Lineare
- Metodo del Simplesso
- Teoria del Simplesso
- Teoria della Dualità e Analisi di Sensitività
- Programmazione Lineare intera
    - Introduzione alla PLI.
    - Branch and Bound Binario
    - Branch and Bound PLI
- Programmazione Non Lineare

1.2 Modalità d'esame

Due Modalità:

- Modalità con Parziali
- Modalità con Scritto unico

Parziali

Primo parziale sarà durante la settimana di sospensione didattica, il secondo a fine del corso a ridosso del primo appello completo. Entrambe le prove hanno voto massimo 14 e soglia di superamento 6.
É possibile *recuperare solo una delle due prove parziali*, a ridosso degli appelli completi.

Primo Parziale Il primo parziale del 17 novembre prevede:

- 1 Esercizio Numerico su *Simplesso* (6 Punti)
- 1 esercizio Numerico su *Dualità* e/o *Sensitività* (4 Punti)
- 1 Domanda a risposta *Aperta* (3 Punti)
- 1 Domanda a risposta *Chiusa* (1 Punto)

La prova avrà una durata di 90 minuti.

Assignments Durante l'insegnamento verranno assegnati 4 Assignments da risolvere a casa e consegnare entro date stabilite, Ad ogni assignment può andare un voto minimo di 0 a un massimo di 1.

Orale non obbligatorio, consiste in 3 domande (aperte/esercizi) su tutto il programma. Si accede all'orale se si hanno già raggiunti i 18 punti. L'orale ha un voto di ± 3.

Scritto unico

Scritto (voto max 28) con domande/esercizi su tutto il programma. Orale faoltativo si accede con un voto minimo di 15.

1.3 Prerequisiti

Per il corso di ROPR sono necessari alcuni prerequisiti, che verranno ripassati a lezione ma è bene conoscere già.
I prerequisiti indicati a lezione sono:

- Familiarità con le **Funzioni**.
  - Definizione di Funzione, dominio, codominio
  - Definizione di F Suriettiva, iniettiva e biiettiva
  - Definizione di F lineare/non lineare
  - Definizione di F concava/convessa
- Familiarità con gli **Spazi Vettoriali**.
  - Definizione di Vettore, Spazio Vettoriale, Base di uno S. Vettoriale
  - Definizione di vettori linearmente dipendenti/indipendenti

- Familiarità con le **Matrici e Sistemi Lineari**.
  - Definizione di matrice, rango e determinante
  - Calcolo di una matrice inversa
  - Definizione e risoluzione di un sistema lineare
- Basi di **Topologia**.
- Basi di **Teoria dei Grafi**.
- Familiarità con il **Calcolo Differenziale**.

Un ripasso sui principali requisiti può essere trovato su E-Learning

Capitolo 2

Ripasso di Algebra

2.1 Operazioni tra matrici

DEFINIZIONE

Somma di Matrici
Siano $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ due matrici *Dello stesso ordine* $2x3$ allora

$$A + B = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

DEFINIZIONE

Prodotto di Matrici
Siano $A = n\text{x}p, B = p\text{x}m$ allora $\exists A \cdot B$ e $\nexists B \cdot A$

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -1 \end{pmatrix}, B = \begin{pmatrix} 4 & 1 \\ -2 & 2 \\ 0 & 3 \end{pmatrix} \rightarrow AB = \begin{pmatrix} 4 & 7 \\ -6 & 3 \end{pmatrix}$$

in cui $ab_{1,1} = (a_{1,1} \cdot b_{1,1}) + (a_{1,2} \cdot b_{2,1}) + (a_{1,3} \cdot b_{3,1})$

2.2 Determinante di una matrice

Il determinante di una radice quadrata si calcola come di seguito:

Determinante 2x2

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \implies det(A) = |A| = a \cdot d - b \cdot c$$

Determinante 3x3

1. Scelgo un elemento a_{ij} della matrice.
2. Prendo la sua riga (o colonna)
3. Sommo, per ogni elemento a_{ij} della riga (colonna)

   $$(-1)^{1+j} \cdot a_{ij} \cdot det(B)$$

   in cui B é la matrice quadrata che ottengo rimuovendo da A sia la riga che la colonna di a_{ij}

2.2.1 Gli Autovalori di una Matrice

L'Autovalore di una matrice quadrata A é un valore scalare λ t.c.:

$$\exists v \neq 0 \text{ t.c. } Av = \lambda v$$

v é un vettore colonna non nullo, ed é l'autovettore per la matrice A relativo a λ t.c.

$$Av - \lambda v = (A \cdot \lambda I) \cdot v$$

Con I Matrice Identitá di A.

Capitolo 3

Modelli nella Ricerca Operativa

3.1 Problemi di Ottimizzazione

La Ricerca Operativa si occupa di Problemi di Ottimizzazione:

DEFINIZIONE

Data la funzione $f : \mathbb{R}^n \to \mathbb{R}$, un **problema di ottimizzazione** è formulabile come segue:

$$\text{opt } f(x) \text{ s.a. } x \in X, X \subseteq \mathbb{R}^n$$

Dove $opt \in \{min, max\}$ e $s.a.$ = "soggetto a".

I problemi di ottimizzazione possono essere:

- Minimizzazione $minf(x)$
- Massimizzazione $maxf(x)$

DEFINIZIONE

In un problema di ottimizzazione la *funzione* $f : R^n ...$ è detta **funzione obiettivo**, X è la **regione ammissibile** e $x \in X$ sono le **variabili decisionali**.

Quindi un problema di ottimizzazione consiste nel determinare (se esistono) uno o più punti di min/max x^* (x^* è una particolare assegnazione che gode di una certa proprietà).

Ottimizzazione Vincolata/non Vincolata Esistono due tipi principali di Ottimizzazione, determinate dall'esistenza o meno di delle *regioni di vincolo*:

- **Non Vincolata** $X = \mathbb{R}^n$: la ricerca dell'ottimo viene effettuata in tutto R^n.
- **Vincolata** $X \subset \mathbb{R}^n$: La ricerca dell'ottimo è soggetta al muoversi all'interno di una certa regione ($x \in [a, +\infty)$ per esempio).

Nel caso dell'ottimizzazione vincolata semplicemente si considera la regione dello spazio definita da un determinato intervallo, quindi le soluzioni valide di solito cambiano.

Ottimizzazione Intera e Binaria Non esiste solo l'ottimizzazione in cui le variabili assumono valori nello spazio reale, ma esistono anche ottimizzazioni con delle caratteristiche particolari.
Il vincolo $X \in Z^n$ richiede che le mie soluzioni *siano valori interi*, quindi **ottimizzazione intera**.
Un altro vincolo molto "reale" è $x \in \{0, 1\}^n$, in questo caso si chiama **ottimizzazione binaria** (vero falso, spento acceso,...). Entrambe queste ottimizzazioni formano le ottimizzazioni a numeri interi. Esiste anche l'ottimizzazione mista, in cui alcune variabili sono intere e altre binarie.

3.2 Programmazione Matematica

DEFINIZIONE

Quando l'insieme delle soluzioni ammissibili di un problema di ottimizzazione viene espresso attraverso un sistema di equazioni e disequazioni, esso prende il nome di *Programmazione Matematica*.

Ovvero, se i vincoli di un problema di ottimizzazione vengono espressi tramite delle equazioni/disequazioni, allora questo problema diventa un problema di programmazione matematica.

Come si definiscono i vincoli

I vincoli vengono espressi con delle espressioni del tipo $g_i(x) \begin{cases} \geq \\ = \\ \leq \end{cases} 0$ in cui $g_i : X \to \mathbb{R}$ è una funzione generica che lega tra loro le variabili decisionali.

Nota che i vincoli dipendo dalle *stesse variabili da cui dipende la funzione obiettivo.*
L'insieme (somma) di tutti i vincoli definisce la *regione ammissibile*, se $x \in X$ allora è una soluzone ammissibile.
In un problema di ottimizzazione abbiamo quindi m vincoli e n variabili.

Le possibilità Quando risolviamo un problema di PM abbiamo quindi le seguenti possibilità:

- Problema non ammissibile, $X = \emptyset$ (problema mal posto con regione ammissibile vuota)
- Problema illimitato, ovvero per ogni soluzione che trovo ne posso trovare un'altra che è MIGLIORE di quella che ho trovato, (può essere illim superiormente o infer)
- Problema con soluzione ottima unica
- Problema con più soluzioni ottime (anche infinite), tutte le ottime hanno lo stesso valore della funzione obiettivo.

Ottimi Globali e Locali I punti di ottimo possono essere Locali o Globali (se la funzione è convessa Locale=Globale). NB un minimo globale è anche locale, un minimo locale non necessariamente è globale. Ad oggi *non* sappiamo dire se una soluzione sia Globale, sappiamo solo con certezza che sia locale (a meno di usare brute force).
La risoluzione di un problema di programmazione matematica consiste nel trovare una soluzione ammissibile che sia ottimo GLOBALE. Alla fine della fiera la definizione di globale e locale corrisponde a quella di Analisi Matematica. I punti di ottimo globali possono essere multipli.

Capitolo 4

Metodo del Simplesso

Il metodo del simplesso è una pocedura algebrica che si basa su dei *concetti geometrici*.

Alcuni Termini Questo metodo si basa su alcune terminologie che ci servono per capire come funziona. Definiamo quindi:

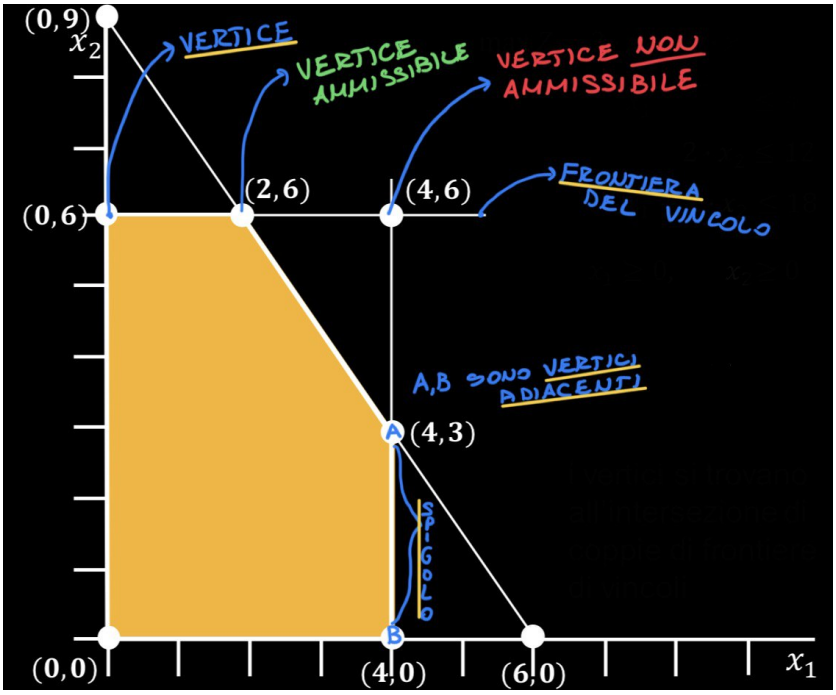
- **Frontiera del vincolo**: è la "linea" di demarcazione di un vincolo.
- **Vertice**: un vertice è un punto di intersezione di coppie di frontiere di vincoli.
  - **Vertici Adiacenti**: Due vertici si dicono adiacenti se condividono $n-1$ frontiere di vincoli.
  - **Spigolo**: segmento che collega due vertici adiacenti
- **Vertice Ammissibile**: Un vertice è detto ammissibile se *fa parte* della regione ammissibile
- **Vertice *non* ammissibile**: Un vertice è detto *non* ammissibile se *non* fa parte della regione ammissibile.

Test di Ottimalità L'interesse per i *vertici adiacenti* sta nella seguente proprietà di cui godono:

DEFINIZIONE

Si consideri ogni problema di PL tale da ammettere almeno una soluzione ottimale:
Se una soluzione vertice non ammette soluzioni vertice a egli adiacen-

ti con valore della funzione obiettivo migliore, allora la **soluzione** in questione è **ottimale**.



4.1 La Procedura Algebrica

Il metodo del simplesso è utilizzabile sia in forma algebrica che geometrica. Siccome questo algoritmo è usualmente eseguito su un calcolatore, che è in grado di interpretare solo istruzioni algebriche, è necessario tradurre la procedura geometrica in una procedura algebrica, che si basa sulla risoluzione di un sistema di equazioni lineari.

Forma Aumentata Per usare il metodo del simplesso bisogna trasformare il modello dalla *forma standard* alla *forma aumentata*:
Per fare ciò bisogna fare le seguenti operazioni:

1. Trasformo tutti i vincoli (tranne quelli di non negatività) in vincoli di *eguaglianza*: $x_1 + x_2 \leq 5 \rightarrow x_1 + x_2 = 5$

2. Per ogni vincolo aggiungo una **Variabile di Slack/Surplus**
   Ogni vincolo riceve una variabile di slack/surplus unica, se il vincolo é

di $\leq$ avrá una variabile di slack, se é di $\geq$ avrá una variabile di surplus. quindi se abbiamo:

$$\begin{cases} x_1 \leq 3 \\ x_2 \leq 2 \\ x_1 + x_2 \leq 5 \end{cases} \rightarrow \begin{cases} x_1 + x_3 = 3 \\ x_2 + x_4 = 2 \\ x_1 + x_2 + x_5 = 5 \end{cases}$$

x_3, x_4, x_5 sono le *variabili di slack*. Assumendo che $x_1 = 0$, la variabile di slack x_3 è la *quantità che manca al termine sinistro della diseguaglianza, affinchè questa sia verificata con il segno di uguaglianza.*

Le variabili di slack hanno alcune proprietà che sono utili da tenere a mente: Per un qualunque valore delle variabili decisionali:

- Se $slack = 0 \implies$ La soluzione corrispondente giace sulla frontiera del vincolo della forma originale.
- Se $slack > 0 \implies$ La soluzione corrispondente giace sul lato ammissibile della frontiera del vincolo della forma originale.
- Se $slack < 0 \implies$ La soluzione corrispondente giace sul lato NON ammissibile della frontiera del vincolo.

Si definisce **Soluzione Aumentata** una soluzione del modello in forma originale che viene *aumentata* tramite i corrispondenti valori delle variabili di slack.

Soluzione di Base Si definisce soluzione di base, un vertice del modello in forma aumentata.
Se abbiamo come variabili di base $x_1 = 4, x_2 = 6$ e come variabili di slack $x_2 = 0, x_3 = 0, x_4 = -6$ allora la soluzione di base (non ammissibile) associata sarà $(4, 6, 0, 0, -6)$.

4.1.1 Variabili di Base e Non

Un modello in forma aumentata consiste di d variabili decisionali e s variabili di slack. Abbiamo quindi a disposizione d gradi di libertà per risolvere il sistema lineare. In altre parole, quando risolviamo un sistema lineare (ponendo a zero il valore di x variabili scelte arbitrariamente) abbiamo tante variabili azzerabili quante sono le variabili decisionali.

Proprietà delle Soluzioni di Base Una **soluzione di base** gode delle seguenti proprietà:

1. Una variabile può essere *Di base* o *NON di base*
2. Il numero delle variabili di base *eguaglia il numero dei vincoli funzionali*, Di conseguenza il numero delle variabili non di base eguaglia il numero totale delle variabili, meno il numero dei vincoli funzionali
3. Le variabili NON di base *vengono poste a zero*
4. I valori delle variabili di base sono ottenuti come risoluzione simultanea del sistema di equazioni lineari (in forma aumentata).
5. Se le variabili di base soddisfano i vincoli di non negatività, la soluzione di base è una soluzione ammissibile di base.

Soluzioni di Base Adiacenti Due soluzioni di base ammissibili sono *adiacenti* se sono caratterizzate dal **condividere le stesse variabili non di base eccetto una**.
Di conseguenza, muoversi da una soluzione di base ammissibile ad una soluzione ad essa adiacente *Implica che una variabile non di base divenga di base* e che *una variabile di base divenga non di base*. Il che richiede di aggiustare i valori delle variabili di base per garantire che il sistema di equazioni sia ancora soddisfatto.

4.2 I Tableau: un Cheatsheet

Ricordiamo il problema in forma standard:

$$\begin{gathered} \max \underline{c}^T \ \underline{x} \\ \underline{\underline{\mathrm{A}}} \ \underline{x} \leq \underline{b} \\ \underline{x} \geq 0 \end{gathered}$$

Tableau Il metodo del simplesso è facilmente risolvibile tramite la *Forma Tabellare*.

1 - La forma aumentata Per portare il problema in Forma Aumentata:

- Vincoli
  - **MinoreUguale** $\leq \rightarrow = \wedge$ + SLACK

  - **Uguale** $=\rightarrow$ INVARIATO
  - **MaggioreUguale** $\geq \rightarrow = \wedge$ - SURPLUS

- Se ho una variabile non positiva: $x_i \leq 0 \rightarrow x_i' = -x_i$, con $x_i' \geq 0$

2 - Porto in forma Tabellare Il problema in forma aumentata viene riscritto in forma tabellare.
ATTENZIONE PERÒ: La funzione obiettivo passa da $z = \underline{c}^T \underline{x}$ diventa $z - (\underline{c}^T \underline{x}) = 0$ (Quindi la funzione obiettivo diventa Z meno i coefficenti delle variabili).

V. BASE	Eq	Z	x_1	x_2	...	x_n	T. Noto
Z	R_0	1	c_1	c_2	...	c_n	0
x_1	R_1	0	a_{11}	a_{12}	...	a_{1n}	b_1
x_m	R_n	0	a_{m1}	a_{m2}	...	a_{mn}	b_m

4.3 Il Metodo delle Due Fasi

In alcuni problemi di PL il punto $(0, 0)$, ovvero l'origine, non è una soluzione ammissibile, quindi non può essere usata come *Soluzione Ammissibile di Base* per il Metodo Del Simplesso.
Quando ci troviamo di fronte a questa situazione abbiamo due possibilità: usare il metodo Big M (che non vedremo) oppure usare il *Metodo delle due Fasi*.

Come funziona il metodo delle due Fasi? Con questo metodo si risolve il problema di PL aggiungendo alla forma aumentata una variabile *artificiale* che va ad "annullare" la variabile di slack (prima fase). La seconda fase consiste nel prendere la soluzione di base generata dalla prima fase e eseguire il metodo del simplesso partendo da lì.

Capitolo 5

Dualità

Ad ogni problema di programmazione lineare, che d'ora in poi chiameremo problema **Primale**, è associato un problema chiamato **Duale**. La soluzione ottimale del problema Duale fornisce i **prezzi ombra** del problema Primale.

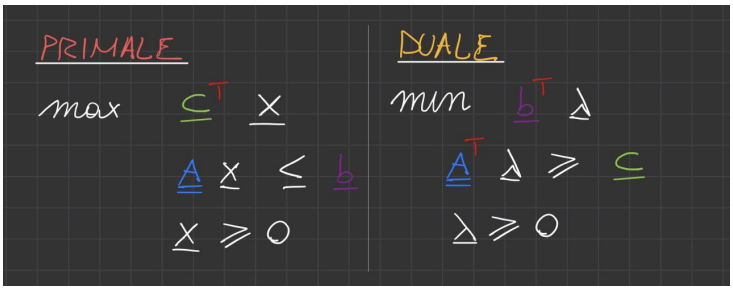
Il Prezzo Ombra Il *prezzo ombra* per la risorsa $i-$esima (y_i^*) misura il valore marginale della risorsa, cioè, il tasso al quale la funzione obiettivo z potrebbe essere incrementata aumentato (leggermente) la quantità di risorsa b_i disponibile. Il metodo del simplesso identifica il prezzo ombra y_i^* come segue:

y_i^* = coefficente della $i-$esima variabile slack nella riga 0 del tableau finale.

Analisi di Sensitività La teoria della Dualità permette l'implementazione e l'interpretazione dell'analisi di sensitività.
L'Analisi di Sensitività è una tecnica che permette di decidere i valori di alcuni parametri di un problema di PL.

5.1 Da Primale a Duale

Un problema duale è strettamente legato al suo primale, infatti per passare da un problema al suo duale:



Più in specifico, le regole per Primale→Duale sono:

Funzione Obiettivo:	max $c_i^T x$	min $b^T \lambda$
Vincoli $\leq$ (primale)	$a_i^T x \leq b_i$	$\lambda_i \geq 0$
Vincoli $=$ (primale)	$a_i^T x = b_i$	λ_i Libera
Vincoli $\geq$ (primale)	$a_i^T x \geq b_i$	$\lambda_i \leq 0$
Vincoli Non negatività	$x_j \geq 0$	$A_j^T \geq c_j$
Variabile Libera	x_j Libera	$A_j^T = c_j$
Vincoli Non positività	$x_j \leq 0$	$A_j^T \leq c_j$

Quindi dato un qualunque problema di PL:

- un problema di MAX diventa un problema di MIN.
- i coefficenti del primale diventano i termini noti del duale
- i termini noti del primale diventano coefficenti del duale.
- i coefficienti di variabile rimangono invariati

Si può notare che il Duale di un Duale è il suo Primale, quindi un problema di minimizzazione diventa un problema di massimizzazione.

5.2 Teorema della dualità

Riassumendo le relazioni tra primale e duale, il teorema di dualità dice:

1. Se un problema ha soluzioni ammissibili e funzione obiettivo limitata (quindi ha soluzione ottimale), allora la stessa cosa vale per l'altro problema, per cui le proprietà di dualità devole e forte sono entrambe applicabili.

2. Se un problema ha soluzioni ammissibili e funzione obiettivo Illimitata (non ha ottimo), allora l'altro problema non ha soluzioni ammissibili.

3. Se un problema ha soluzioni ammissibili, allora l'altro problema o non ha soluzioni ammissibili o ha una funzione obiettivo Illimitata

O, in parole povere: Se un problema ha:

- Sol. Ammissibili, e Ottimo $\implies$ Idem per l'altro (Dualità forte e debole applicabili)
- Sol. Ammissibili, e NON Ottimo (obb. illimitata) $\implies$ L'altro NON ha soluzioni ammissibili
- No Sol. Ammissibili $\implies$ L'altro o non ha sol. amm. o ha f.obb Illimitata

5.3 Gli Scarti Complementari

Se ho una soluzione del problema primale e voglio controllare se essa è **ottima**, posso farlo senza risolvere il problema usando le *Condizioni degli Scarti Complementari*.
Le condizioni degli scarti complementari mi permettono, oltre che a controllare se la soluzione del primale in esame sia ottima, anche a **trovare la soluzione del duale** senza dover usare il metodo del simplesso su di esso.

5.3.1 Come funziona

Se ho una soluzione del primale, controllo che sia ottima e trovo la corrispondente soluzione ottima del duale:

1. Controllo che la soluzione in esame sia **ammissibile per il primale**
2. applico le **condizioni degli scarti complementari**:
   - Se una variabile della soluzione è *diversa da 0*, allora il corrispondente vincolo del duale attivo.
   - Se un vincole del primale è attivo, allora la corrispondente variabile del duale è uguale a zero.
3. Pongo a sistema le equazioni che ho trovato, se il risultato è **ammissibile per il duale**, allora esso è l'ottimo del duale e il risultato in esame è l'ottimo del primale.

I vincoli attivi Un vincolo è attivo per una soluzione se applicando i valori delle variabili esso risulta *rispettato all'uguaglianza*.
Per esempio, se ho una soluzione $(3, 0, 2)$, i vincoli:

- $x_1 + 2x_2 + 3x_3 \leq 9$ è attivo: $9 \leq 9$
- $x_1 + 2x_2 + x_3 \leq 7$ NON è attivo: $7 \leq 9$

Capitolo 6

Modelli PLI

6.1 Introduzione

Fino ad ora abbiamo visto esempi di programmazione lineare che utlizzavano variabili di decisione nel dominio **reale**.

Esistono però molti problemi che richiedono che richiedono che le variabili di decisione siano *intere o binarie*. In questi casi parliamo di **Programmazione Lineare Intera** o PLI.

I tipi di PLI Esistono 3 tipi di PLI:

- **Intera** $\rightarrow x \in \mathbb{Z}^n$ (PLI)
- **Binaria** $\rightarrow x \in \{0, 1\}$ (PB)
- **Mista** $\rightarrow x \in \mathbb{R}^p \times \mathbb{Z}^q$, con $p + q = n$ e $p > 0, q > 0$ (MIP)

6.2 Le Condizioni Logiche

Come abbiamo detto, nella PLI possiamo introdurre anche delle **Variabili Binarie**. In generale, le variabili binarie sono utilizzate nella RO per modellare decisioni del tipo *Si/No*, e possono essere usate per introdurre delle *Condizioni Logiche*.

Vincoli "*either-or*"

Se consideriamo il caso in cui dati due vincoli almeno uno di questi deve essere soddisfatto (per esempio perché potrebbe esserci una scelta su risorse

alternative da utilizzare per un determinato scopo, in modo che sia necessario solo uno dei vincoli). Prendiamo come esempio:

$$3x_1 + 2x_2 \leq 18 \vee x_1 + 4x_2 \leq 16$$

Questa diventa equivalente a:

$$\begin{cases} 3x_1 + 2x_2 \leq 18 + My \\ x_1 + 4x_{\leq} 16 + M(1-y) \end{cases}$$

in cui M è un *numero molto grande* e $y \in \{0, 1\}$ quindi se:

- $y = 0 \implies$ il secondo vincolo è sempre soddisfatto, quindi si può eliminare.
- $y = 1 \implies$ il secondo vincolo è sempre soddisfatto, quindi si può eliminare.

M molto grande Questo elemento si trova molto spesso nelle condizioni logiche, e di solito è moltiplicato a y e $(y-1)$, e significa:

- $y = 0 \implies My = 0 \wedge M(1-y) = M$
- $y = 1 \implies My = M \wedge M(1-y) = 0$

Quindi, al variare di y uno dei due vincoli dovrà essere $\leq M$, ed essendo "M molto grande" questo è sempre verificato, di fatto disattivando il vincolo.

N Vicoli, K devono essere soddisfatti

Consideriamo il caso in cui in un modello includa N vincoli tali che solo K di essi devono essere soddisfatti ($K < N$).

$$\begin{gathered} f_1(x_1, x_2, x_n) \leq d_1 \\ f_2(x_1, x_2, x_n) \leq d_2 \\ \ldots \\ f_N(x_1, x_2, x_n) \leq d_N \end{gathered}$$

Con N variabili binarie $y_i \in \{0, 1\}$ per $i = 1, ..., N$ ed un numero M positivo molto grande. Abbiamo la seguente formulazione equivalente al requisito che solo k di questi N vincoli devono essere soddisfatti:

$$\begin{cases} f_1(x_1, x_2, x_n) \leq d_1 + My_1 \\ f_2(x_1, x_2, x_n) \leq d_2 + My_2 \\ \ldots \\ f_N(x_1, x_2, x_n) \leq d_N + My_N \\ \sum_{i=1}^{N} y_i = N - K \end{cases}$$

In questo caso solo $N-K$ delle nostre y può essere uguale a 1, quindi $N-K$ vincoli (quelli con $y=1$) saranno disattivati (rilasciati).

La funzione assume solo N possibili valori

Si consideri la situazione in cui una data funzione assuma uno fra N possibili valori:

$$f(x_1, x_2, ..., x_n) = d_1 \vee d_2, ..., \vee d_n$$

consideriamo N variabili binarie $y_i \in \{0,1\}$ per $i = 1, ..., N$ abbiamo la seguente formulaizone equivalente al requisito che una data funzione assuma uno fra N possibili valori:

$$\begin{cases} f(x_1, x_2, ..., x_n) = \sum_{i=1}^{N} d_i y_i \\ \sum_{i=1}^{N} y_i = 1 \end{cases}$$

Per cui uno solo dei termini noti sarà attivo.

6.3 Risoluzione dei PLI

Nonostante lo spazio delle variabili sia "solo" $\mathbb{N}$, anche in questo caso una risoluzione con forza bruta non è considerabile.

6.3.1 Il Rilassamento Lineare

Per un qualsiasi problema PLI è possibile formulare il corrispettivo problema PL, ovvero lo stesso problema senza i vincoli di interezza. Tale problema prende il nome di **rilassamento lineare**.

In sostanza il rilassamento lineare consiste nel togliere i vincoli di Interezza ($x \in \mathbb{Z}$) e sostituirli con dei vincoli di non negatività ($x \geq 0$)

Questo può in molti casi essere utile, perché il rilassamento lineare può essere risolto con il metodo del simplesso.

Risolvere un rilassamento lineare, per esempio tramite il metodo del simplesso, ci dà due possibilità:

- Soluzione **Intera** $\implies$ è anche la soluzione del PLI.
- Soluzione **Frazionaria** $\implies$ ho trovato un **Upper (Lower) Bound** per il PLI di **Max (Min)**.

Upper/Lower Bound Quando risolvo il rilassamento lineare di un PLI di Max, se la soluzione che ho trovato NON è intera allora significa che ho trovato un Upper Bound, ovvero un valore che l'ottimo del PLI sicuramente non supererà. Viceversa vale per i problemi di minimo e il rispettivo Lower Bound.

I Bound (Upper e Lower) sono utili ad alcuni algoritmi di risoluzione dei PLI.

Attenzione però, in generale la soluzione ottima di un PLI può non corrispondere a una delle sue soluzioni intere ottenute arrotondando le variabili non intere, quindi non posso risolvere un PLI arrotondando il risultato di un rilassamento.

Capitolo 7

Branch and Bound

Il Branch and Bound è una tecnica per la risoluzione di problemi di ottimizzazione intera.
Questa è una tecnica di *enumerazione implicita*, ovvero:

- "Valuta" le soluzioni possibili fino a trovare quella ottima.
- Scarta alcune di queste soluzioni a priori dimostrando la loro non ottimalità.
- Si basa sul concetto di Dividi et Impera.

7.1 Come funziona

Supponiamo di avere un problema di PLI che chiamiamo *problema completo*. Un problema di PLI rappresenta un **sotto-problema** del problema completo se presenta la medesima funzione obiettivo, ma ha un sottoinsieme proprio di X come regione ammissibile.

Sia $z* = f(x*)$ la soluzione del problema completo e $\tilde{z} = f(\tilde{x})$ la soluzione ottima di un sotto-problema. Allora si ha che $f(\tilde{x}) \leq f(x*)$.
Basta solo notare che tra le soluzioni ammissibili del problema completo vi è anche x (non dovrebbe essere $\tilde{x}$?) e che per definizione di ottimo $f(\tilde{x}) \leq f(x*)$.

7.2 Le tecniche di risoluzione

Il Branch and Bound fa uso delle tre seguenti tecniche per risolvere un generico problema di PLI:

- La partizione rispetto al valore delle variabili, o **Branching**.
- La determinazione di un limite superiore, o **Bounding**.
- L'eliminazione di sottoproblemi, o **Fathoming**.

La procedura del Branch and Bound per i problemi di PB e quelli di PIM è molto simile, la soluzione dei problemi rilassati è ancora alla base delle fasi di *bounding* e *fathoming* ma occorre apportare alcune modifiche (che riporto nelle sezioni adeguate).

7.2.1 Branching

Il Branching implica la selezione di un sotto-problema da analizzare e la divisione di questo in sotto-problemi più piccoli. Vi sono diverse regole di selezione e divisione possibili. Posso scegliere:

- ***Il sotto-problema creato più di recente***: tale regola di selezione è efficiente per ripartire con l'ottimizzazione del rilassamento LP dal precedente problema (devo solo aggiungere un vincolo).
  È anche nota come *Depth First*, perché equivale a scegliere il nodo di livello maggiore, ovvero più profondo.
  ◦ *Pro*: è semplice da implementare, permette di ottenere presto delle soluzioni ammissibili (ci si avvicina più rapidamente alle foglie) e limita la memoria necessaria per memorizzare l'albero delle soluzioni, visto che si tendono a chiudere molti nodi per inammissibilità e rimangono pochi nodi contemporaneamente aperti.
  ◦ *Cons*: presenta il rischio di esplorare completamente sotto-alberi con soluzioni scadenti.

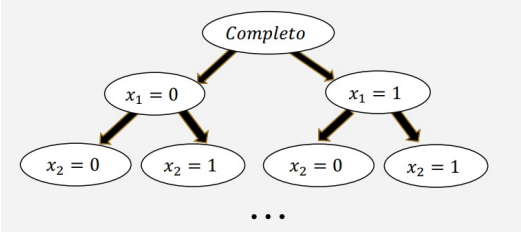
- ***Il sotto-problema con il miglior bound***: in genere conduce a soluzioni incombenti più velocemente e quindi permette di effettuare un *fathoming* più efficiente.
  È anche nota come *Best Bound First*: si sceglie il nodo *più promettente* ossia il nodo con il *bound migliore* (**lower bound** più **basso**, per problemi di *minimo*, o **upper bound** più **alto**, per problemi di *massimo*).
  ◦ *Pro*: permette di limitare il numero di nodi visitati esplicitamente e tende pertanto a essere più efficiente.
  ◦ *Cons*: l'esplorazione tende a rimanere a livelli poco profondi, dove i problemi sono meno vincolati e i bound sono più promettenti ma difficilmente si ottengono presto soluzioni ammissibili che migliorino

la soluzione incombente corrente. Questo impedisce di applicare efficacemente le regole di *fathoming*, pertanto è maggiore la richiesta di memoria per i nodi aperti contemporaneamente.

- Si può anche usare una combinazione delle due regole: i nodi vengono scelti cioè alternando i diversi criteri, per evitare gli svantaggi di uno o dell'altro.

Branching per problemi di PB (variabili binarie)

Nel caso di problemi a variabili binarie (PB), il modo più semplice per dividere l'insieme delle soluzioni ammissibili in sotto-insiemi è fissare il valore di una delle variabili, per esempio $x_1 = 0$, per un sottoinsieme e a $x_1 = 1$ per l'altro sottoinsieme.



La variabile utilizzata ad ogni iterazione per eseguire questa suddivisione è chiamata **variabile di branching**. Nel diagramma qua sopra, al primo livello la variabile di branching sarà x_1, a quello successivo x_2 etc... Questo perché per selezionare ad ogni iterazione la variabile di branching si può utilizzare l'ordinamento naturale delle variabili (x_1, x_2, ...).
N.B.: nel caso binario, ad ogni branching generiamo due nuovi sotto-problemi, creando un albero binario.

Branching per problemi di PIM

Differisce dal *Branching* per problemi di PB precedentemente visto per poche cose. Il **primo** cambiamento riguarda la scelta della variabile di *Branching*:

- Nel B&B per PB la scelta è stata effettuata seguendo l'indice delle variabili: ora non è più possibile perché sulle variabili continue non può essere effettuato il *Branching*.

- La regola deve essere modificata scegliendo la prima variabile non continua seguendo l'ordine naturale degli indici (o regole più sofisticate).

Il **secondo** cambiamento riguarda i valori assegnati alla variabile per generare i sotto-problemi:

- Nel caso PB alla variabile di branching veniva assegnato il valore 0 e 1 rispettivamente per generare i due nuovi sotto-problemi.
- Nel caso generale, assegnare tutti i possibili valori interi sarebbe impossibile quindi si generano due sotto-problemi specificando per la variabile due insiemi di valori

  $$x_j \leq \lfloor x_j^* \rfloor \quad e \quad x_j \geq \lfloor x_j^* \rfloor + 1$$

  dove $\lfloor x_j^* \rfloor$ è il massimo intero per cui $\lfloor x_j^* \rfloor \leq x_j^*$.

N.B.: può verificarsi che una stessa variabile venga selezionata più volte durante la procedura.
Il **terzo** cambiamento riguarda la fase di *Bounding*.
Il **quarto** (e ultimo) cambiamento riguarda la fase di *Fanthoming*.

7.2.2 Bounding

Bounding per problemi di PB (variabili binarie)

Per ognuno dei sotto-problemi dobbiamo ricavare un limite superiore (perché è un problema di massimo, altrimenti un limite inferiore).
Per ottenere questo limite si risolve il *problema rilassato* (ovvero senza considerare i vincoli di interezza ma, essendo il problema binario, limitando le variabili tra 0 e 1).
Quindi risolvo i miei sotto-problemi sostituendo l'ultima riga con i vincoli $x_j \geq 0 \wedge x_j \leq 1$ (ovvero $0 \leq x_j \leq 1$).
Come prima cosa risolviamo la versione rilassata del problema completo (per esempio con il metodo del simplesso) e otteniamo un valore per la nostra Z, per esempio $Z = 12,345$. Essendo nell'ambito binario possiamo (come anche nell'ambito intero ovviamente) arrotondare, in questo caso all'intero precedente: se i coefficienti della funzione obiettivo del problema sono tutti interi e le soluzioni del problema devono essere binarie (o intere), sicuramente con le mie variabili non potrei mai raggiungere il valore decimale.

Bounding per problemi di PIM

- Nel caso PB, se i coefficienti della funzione obiettivo erano tutti interi, il valore della soluzione del problema rilassato Z poteva essere arrotondato all'intero inferiore (se *Max*, viceversa se *Min*).
- Nel caso generale questo non è più possibile.

7.2.3 Fathoming

Fathoming per problemi di PB (variabili binarie)

Un sotto-problema può essere "eliminato" dalla lista dei problemi da considerare per tre possibili ragioni:

1. La soluzione ottenuta soddisfa i vincoli di interezza (come la soluzione del sotto-problema 1. In particolare, questa soluzione può essere considerata come la soluzione intera migliore trovata fino a questo momento (soluzione incombente Z*). Quindi il valore della soluzione incombente a questo punto è Z*=9).
   La soluzione incombente cambierà ogni volta che un sotto-problema otterrà una soluzione intera migliore della soluzione incombente corrente.

2. La soluzione incombente può essere utilizzata per un'altra possibilità di fathoming: se il bound di un sotto-problema (che rappresenta la soluzione ottima migliore che si potrà trovare continuando a considerare il sotto-problema) è peggiore della soluzione incombente (migliore soluzione ammissibile intera già trovata) allora non vale la pena continuare a considerare il sotto-problema che quindi può essere chiuso (*fathomed*).

3. Il sotto-problema non ammette soluzioni ammissibili.

Quindi verranno ulteriormente risolti ed analizzati solo quei problemi che non hanno una soluzione intera ma il loro bound è migliore della soluzione incombente.

Fanthoming per problemi di PIM

- Nel caso PB, potevamo fermarci se la soluzione del problema era *intera*.
- Nel caso generale basta che la soluzione soddisfi la condizione di interezza per le variabili che non possono essere continue.

7.3 La divisione di una variabile

Può essere effettuata in più modi:

- si sceglie una variabile e si assegnano valori diversi (es.: programmazione binaria);
- si assegna alla variabile un insieme di valori (es.: problemi di PLI).

Branching

La scelta della variabile su cui effettuare il **branching** va effettuata tra eventuali possibili scelte con l'obiettivo di arrivare più velocemente ad una possibilità di *fathoming*.

Bounding

Il **bounding** è generalmente eseguito risolvendo un rilassamento del problema (noi ne abbiamo visto un tipo, ma ne esistono varie tipologie).

7.4 Il rilassamento Lagrangiano

L'intero insieme di vincoli funzionali $\mathbf{Ax} \leq \mathbf{b}$ viene *cancellato* (o *quasi interamente cancellato*) e la funzione obiettivo

$$Max\ Z = \mathbf{cx}$$

viene sostituita da

$$Max\ Z = \mathbf{cx} - \lambda(\mathbf{Ax} - \mathbf{b})$$

Il problema

$$Max\ Z = \mathbf{cx} - \lambda(\mathbf{Ax} - \mathbf{b})$$

$$s.t.\ \geq 0$$

è detto ***problema Lagrangiano*** e ha la proprietà secondo la quale $\forall\, \lambda \geq 0$ fissato la sua soluzione ottima costituisce un **upper bound** sull'ottimo del problema originario di *Max* (**lower bound** se il problema è di *Min*).
Il bound ottenibile con il *rilassamento Lagrangiano* è quindi almeno tanto buono quanto quello ottenibile con il *rilassamento lineare*.

Pro

L'efficacia pratica del rilassamento Lagrangiano è legata all'efficienza con cui si riesce a risolvere il problema duale del problema Lagrangiano.

Cons

Non è però efficiente come il rilassamento LP nel calcolare il *primo* e il *terzo* dei test di *fathoming*.

I criteri di Fanthoming

Ricordando che i criteri di *fathoming* derivanti dall'analisi di un problema rilassato:

- *Criterio 1*: È stata trovata una soluzione ottima intera del sotto-problema.
- *Criterio 2*: La soluzione ottima Z del sotto-problema deve essere peggiore della soluzione incombente Z^* ($Z \leq Z^*$ per un problema di *Max* e *viceversa* per un problema di *Min*).
- *Criterio 3*: Il sotto-problema non presenta soluzioni ammissibili.

7.5 Problemi con soluzioni ottime alternative

Potrebbe essere importante poter ottenere una lista delle soluzioni ottime, in modo che la scelta della soluzione migliore possa essere poi effettuata sulla base di considerazioni che potrebbero non essere state incluse nel modello matematico.
Per trovarle basta fare alcune piccole modifiche:

- Il test del Criterio 2 deve essere fatto sulla base della disugualianza stretta ($Z < Z^*$ per un problema di *Max* e *viceversa* per un problema di *Min*).
  In questo modo il *Fanthoming* non avverrà se $Z = Z^*$.
- Se la soluzione di un sotto-problema soddisfa il criterio 1 con $Z = Z^*$ allora anche questa soluzione deve essere memorizzata come incombente, e occorre controllare se il sotto-problema ammette soluzioni multiple. Nel qual caso occorre calcolare e memorizzare anche le soluzioni alternative come incombenti.

Alla fine della procedura tutte le soluzioni correnti incombenti (tutte *equivalenti*) saranno considerate **ottime**.

7.6 Soluzioni "quasi-ottime"

Il Branch and Bound può essere utilizzato anche per trovare soluzioni "buone" (quasi-ottime) anche se non necessariamente ottime, che in genere richiedono uno sforzo computazionale molto minore.

Una soluzione è considerata quasi-ottima quando la sua soluzione Z è "abbastanza" vicina al valore di una soluzione ottima Z^{**}, ovvero

$$Z^{**} - K \leq Z$$

o equivalentemente

$$(1 - \alpha)Z^{**} \leq Z$$

dove K e α sono costanti date.
N.B.: α mi rappresenta l'approssimazione (percentuale) della soluzione quasi-ottima trovata.
Quindi se una soluzione incombente Z^* soddisfa

$$Z^{**} - K \leq Z^*$$

oppure

$$(1 - \alpha)Z^{**} \leq Z^*$$

La procedura può terminare fornendo una soluzione Z^* quasi-ottima.

Ma come si può identificare quel valore Z^{**}?
Se esistesse una soluzione ammissibile con valore Z^{**} allora si avrebbe $Z^{**} \leq Bound$ dove $Bound$ è la soluzione migliore tra quelle dei problemi rilassati ancora aperti.
Quindi possiamo sostituire $Bound$ a Z^{**} nelle formule precedenti ottenendo:

$$Bound - K \leq Z^*$$

oppure

$$(1 - \alpha)Bound \leq Z^*$$

Anche se la soluzione corrispondente a $Bound$ non è ammissibile (intera) costituisce comunque un limite superiore valido.
Per problemi di grosse dimensioni considerare le soluzioni quasi-ottime può evitare tempi computazionali proibitivi.

7.7 Criteri d'arresto

Il metodo del Branch-and-Bound si arresta quando tutti i nodi sono dichiarati *fathomed*. In questo caso, la soluzione ammissibile corrente corrisponde ad una soluzione ottima.

Si possono anche usare soluzioni quasi-ottime e fermarci non appena troviamo una soluzione quasi ottima.
Oppure ancora possono anche essere adottati criteri relativi a limiti computazionali, come ad esempio raggiunti limiti di tempo di calcolo o di memoria, ma non è garantito che l'eventuale soluzione ammissibile corrente sia ottima (posso comunque calcolare quanto distante è dal *Bound*).

7.8 Considerazioni finali sul B&B

Valutazione di soluzioni ammissibili

Per applicare efficacemente le regole di *Fathoming*, è necessario disporre di soluzioni ammissibili di buona qualità. Bisogna quindi stabilire come e quando calcolare soluzioni ammissibili.
Abbiamo varie possibilità:

- Aspettare che l'enumerazione generi un nodo foglia ammissibile;
- Implementare un algoritmo che valuti una buona soluzione prima dell'esplorazione;
- Sfruttare, con frequenza da valutare, l'informazione raccolta durante l'esplorazione dell'albero per costruire soluzioni ammissibili sempre migliori;
- In ogni caso, bisogna **sempre** valutare il compromesso tra la qualità della soluzione ammissibile corrente e lo sforzo computazionale per ottenerla.

Capitolo 8

Introduzione alla Programmazione non Lineare

Un **problema di programmazione non lineare (PNL)** può essere formulato come:

$$\text{opt}\, f(\mathbf{x})$$

soggetto ai seguenti vincoli:

$$g_j(\mathbf{x}) \leq 0 \quad \text{con} \quad j = 1, ..., m$$

$$x_i \geq 0 \quad \text{con} \quad i = 1, ..., n$$

dove opt $= \begin{cases} min \\ max \end{cases}$
in cui $f\colon \mathbb{R}^n \to \mathbb{R}$ e $g_j : \mathbb{R}^n \to \mathbb{R}$ sono funzioni note di $x \in \mathbb{R}^N$.

Tipologie di Programmazione non-lineare

A seconda delle caratteristiche del problema abbiamo diversi tipi di programmazione non lineare.

Ottimizzazione non vincolata

I problemi di ottimizzazione non vincolata non hanno vincoli sulla regione ammissibile. Quindi l'obiettivo è semplicemente

$$\max f(\mathbf{x}) \quad o \quad \min f(\mathbf{x}) \quad x \in \mathbb{R}^n$$

.

Ottimizzazione con vincoli lineari

I problemi di ottimizzazione con vincoli lineari sono caratterizzati da tutte le funzioni $g_i(\mathbf{x})$ lineari, ma dalla funzione obiettivo non-lineare. Sono problemi che vanno a dare origine ad una *regione ammissibile convessa*. Un caso particolare è la *programmazione quadratica*, in cui la funzione obiettivo è una funzione quadratica.

Ottimizzazione convessa

Problemi in cui $f(\mathbf{x})$ è una funzione **concava** o **convessa** e ogni funzione $g_i(\mathbf{x})$ è **convessa**.

Ottimizzazione non convessa

I problemi di programmazione non convessa comprendono tutti i problemi che non soddisfano le ipotesi di convessità. Sono più difficili da risolvere perché possono presentare diversi punti di minimo/massimo.

Capitolo 9

PNL Monovariata

9.1 Risoluzione Analitica

Consideriamo un problema di massimizzazione con una funzione obiettivo $f(x)$ concava da massimizzare. Essendo questa funzione **concava**, sappiamo che una condizione sufficiente affinchè x^* sia **punto di massimo** è che:

$\frac{d}{dx}f(x^*) = 0$, ovvero il punto di massimo è dove la derivata è stazionaria

Capiamo quindi che se *un'equazione può essere risolta analiticamente* allora il procedimento per trovare l'ottimo termina.
Un discorso equivalente si può fare per problemi di minimizzazione di funzioni convesse.

9.2 Algoritmi Generici per la Risoluzione Numerica

E se non posso risolverla analiticamente?
In mancanza di una risoluzione analitica (per esempio per una funzione troppo complicata) esistono degli **algoritmi per la risoluzione numerica** che si basano sul concetto di sequenza.

Gli algoritmi della sequenza Si costruisce una sequenza di punti $\{x_k\}$ t.c.: (nei casi di minimizzazione)

$$\lim_{x \to +\infty} x_k = x^* \text{ e } f(x_{k+1}) \leq f(x_k)$$

Ad ogni iterazione k, partendo da x_k si esegue una ricerca sistematica per identificare un punto migliore x_k.

I criteri di Arresto A differenza dell'algoritmo del simplesso, in questo caso la sequenza di punti $\{x_k\}$ non è detto che converga alla soluzione ottima del problema in un numero finito di iterazioni.
Quindi quando fermo la sequenza di punti? Mi fermo se:

- La soluzione è sufficientemente accurata, ovvero $\frac{df(x_k)}{dx} \simeq 0$
- Quando si è raggiunto un numero massimo di iterazioni N o un tempo computazionale massimo.
- I progressi sono lenti, ovvero $|x_{k+1} - x_k| < \epsilon_x$ o $|f(x_{k+1}) - f(x_k)| < \epsilon_f$.
- La soluzione diverge.
- Si verificano cicli.

Gli algoritmi

Esistono due tipi di algoritmi che si basano su questo funzionamento:

Dicotomici: Algoritmi di ricerca per individuare un determinato valore (per il quale la funzione derivata si annulla) all'interno di un intervallo che ad ogni iterazione viene ridotto.

Di Approssimazione: Algoritmi che utilizzando approssimazioni locali della funzione.

Per l'ottimizzazione di funzioni in una variabile tratteremo il *Metodo di Bisezione*, che è di tipo Dicotomico, e il *Metodo di Newton*, che è un algoritmo di Approssimazione.

9.3 Metodo di Bisezione

Il metodo di Bisezione è una procedura intuitiva e semplice che può essere applicata ad un problema PNL non vincolato quando $f(x)$ è concava/convessa, continua e derivabile.
L'idea di questo algoritmo è la seguente:
Se $f(x)$ è *continua e concava* in un intervallo chiuso $[a, b]$ allora, considerando **un generico punto** x_k, se:

- $\frac{d}{dx}f(x_k) < 0 \implies$ l'ottimo x^* si trova a **sinistra** di x_k.
- $\frac{d}{dx}f(x_k) > 0 \implies$ l'ottimo x^* si trova a **destra** di x_k.
- $\frac{d}{dx}f(x_k) \simeq 0 \implies x_k \simeq x^*$, ovvero è circa l'ottimo.

9.3.1 Funzionamento

Il metodo di bisezione è una procedura abbastanza intuitiva.
Guardando la *pendenza della derivata* per un punto x_k in una funzione *concava e derivabile*:

- Se $f'(x_k) > 0$:
  - Il nuovo punto x_{k+1} lo ottengo spostandomi verso Destra
  - x_k rappresenta un *Estremo Inferiore* $\underline{x}$ per il punto x^*
- Se $f'(x_k) < 0$:
  - Il nuovo punto x_{k+1} lo ottengo spostandomi verso Sinistra
  - x_k rappresenta un *Estremo Superiore* $\bar{x}$ per il punto x^*

Ad ogni iterazione k posso identificare un sotto-intervallo di ricerca $[a_k, b_k] \subset [a,\, b]$ per ridurre lo spazio di ricerca in modo da identificare un'iterazione k per cui $|b_k - a_k| < 2\epsilon$ con $|a_k - x^*| < \epsilon$ e $|x^* - b_k| < \epsilon$

9.3.2 Algoritmo

Inizializzazione $k = 0$ e ϵ piccolo a piacere.
Si determinano i valori $\underline{x}$ e $\bar{x}$ per cui la derivata sia rispettivamente positiva e negativa.
Si seleziona il punto iniziale $x_0 = \frac{\underline{x}+\bar{x}}{2}$

Iterazione del metodo di bisezione bisogna:

1. Calcolare $f'(x_k)$
2. if $f'(x_k) = 0$
   then $x_k = x^*$
   else
   if $f'(x_k) < 0$
   $\bar{x} = x_k$
   else
   $\underline{x} = x_k$
3. Pongo $x_{k+1} = \frac{\underline{x}+\bar{x}}{2}$ e $k = k + 1$

Criterio di arresto: se $\underline{x} - \bar{x} \leq 2\epsilon$ allora il punto di ottimo x^* avrà una distanza minore di ϵ da uno dei due.

9.4 Metodo di Newton

Il metodo di newton è molto simile a quello di bisezione ma considera anche la derivata seconda e non solo l'informazione relativa alla derivata prima.

9.4.1 Funzionamento

il Metodo di newton ottiene un'approssimazione quadratica (in x^2) tramite la formula di Taylor centrata in x_k

$$f(x_k) \sim f(x_k) + f'(x_k)(x_{k+1} - x_k) + \frac{1}{2}f''(x_k)(x_{k+1} - x_k)^2$$

L'approsimazione quadratica è in funzione solo di x_{k+1}! Calcoliamo la derivata dell'approssimazione quadratica

$$f'(x_k) + f''(x_k)(x_{k+1} - x_k) = 0 \rightarrow x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Quindi, l'idea dell'algoritmo di Newton è quella di usare l'ottimo dell'approsimazione quadratica di $f(x)$ dato da:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Questa formula viene usata ad ogni generiaca iterazione k per calcolare il punto successivo x_{k+1}

Osservazione Se $f(x)$ è concava allora x_k converge verso un punto di massimo, perchè?

- x_k è a Sinistra del punto di massimo
  $\implies f'(x_k) > 0 \implies -\frac{f'(x_k)}{f''(x_k)} > 0 \implies x_{k+1} > x_k$
- x_k è a Destra del punto di massimo
  $\implies f'(x_k) < 0 \implies -\frac{f'(x_k)}{f''(x_k)} < 0 \implies x_{k+1} < x_k$

Ovvero L'algoritmo di Newton "decide" matematicamente se spostarsi a sinistra o destra.

9.4.2 Algoritmo

- Inizializzazione
  - si fissa ϵ e $k = 0$

- Iterazione del metodo di newton
  - Calcola $f'(x_k)$ e $f''(x)$
  - Si pone $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$

Criterio di arresto: se $|x_{k+1} - x_k| \leq \epsilon$ allora $x_{k+1} = x^*$ è la soluzione ottima, altrimenti eseguo una nuova iterazione del metodo di Newton e pongo $k = k + 1$

Capitolo 10

PNL Multivariata

Nella PNL Monovariata utilizziamo le informazioni di Derivata Prima e Seconda per trovare l'ottimo. Anche nel caso della PNL Multivariata usiamo gli stessi concetti, bisogna peró estenderli agli spazi multidimensionali.
Vogliamo estendere i concetti di derivata (prima e seconda) visti per funzioni in una variabile, al caso di funzioni in più variabili.
In particolare, consideriamo uno spazio $\mathbb{R}^n$, ovvero uno spazio vettoriale a n dimensioni. Se nel caso $\mathbb{R}^1$ avevamo Derivata I e II, nel caso n-dimensionale:

- $f'(x) \rightarrow$ Gradiente.
- $f''(x) \rightarrow$ Hessiana.

10.1 Derivate Parziali

Per conoscere Gradiente ed Hessiana dobbiamo introdurre il concetto di Derivata Parziale.

DEFINIZIONE

Le **Derivate Parziali** (anche dette derivate direzionali) sono le derivate in un iperspazio per una determinata direzione v.

Quindi se ho $f(x, y, z)$ posso ottenere

$$\frac{df(x,y,z)}{dx}, \frac{df(x,y,z)}{dy}, \frac{df(x,y,z)}{dz}$$

Ovvero le derivate parziali di $f(x, y, z)$ in ognuna delle sue variabili. ***N.B.*** Se derivo $f(x, y, z)$ in x, allora y e z vengono trattate come costanti nel processo di derivazione.

10.2 Gradiente ed Hessiana

Ora possiamo quindi introdurre Gradiente ed Hessiana:

10.2.1 Gradiente

Il gradiente di $f(x, y)$ è per definizione il vettore $\mathrm{D}f$ le cui componenti sono le **derivate parziali** di f, ovvero

$$\nabla f(x,\, y) = \left[\frac{\partial f(x,\, y)}{\partial x},\, \frac{\partial f(x,\, y)}{\partial y}\right]^T$$

Esistono varie notazioni per indicare il gradiente:

$$\nabla f, \quad \mathrm{D}f(x,\, y), \quad \nabla f(x,\, y), \quad \mathrm{grad} f$$

10.2.2 Hessiana

L'insieme delle derivate seconde costituisce una matrice quadrata chiamata matrice Hessiana, e si denota con il simbolo $\mathrm{D}^2 f$ o H_f.

$$H_f f(x,\, y) = \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x \partial x} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial y \partial x} & \frac{\partial^2 f(x,y)}{\partial y \partial y} \end{bmatrix}$$

In cui ogni riga della matrice Hessiana "corrisponde" ad un elemento del Gradiente, mentre ogni colonna corrisponde alla derivata parziale in ognuna delle variabili di f.

Derivate Pure Gli elementi sulla diagonale principale vengono chiamati **derivate pure**, per distinguerli dagli altri elementi chiamati **derivate miste**.

10.3 Metodo Analitico per l'Ottimo

Il gradiente riassume le informazioni di crescita della funzione lungo tutte le direzioni. I punti per cui il gradiente si annulla ($\nabla f = 0$), prendono il nome di **punti stazionari**.

10.3.1 Punti stazionari n-dimensionali

DEFINIZIONE

Se abbiamo un punto (x_0, y_0) t.c.

$$\nabla f(x_0, y_0) = \left[\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right]^T = [0, 0]^T$$

Allora questo é un **punto stazionario**, e potrebbe essere un punto di massimo, minimo o di sella.

Punto di Sella Se nel caso 1D ogni punto in cui la derivata si annulla é un punto di Max o Min, nel caso n-dimensionale puó anche essere un *punto di sella*, ovvero un punto che é massimo in una direzione (variabile) e minimo nell'altra.

10.3.2 Trovare Massimo e Minimo

Una volta trovato un punto stazionario (x, y), voglio capire se si tratta di un punto di Massimo o di Minimo:

DEFINIZIONE

Condizione **sufficiente** affinché (x, y) sia un punto di **Minimo** per f è che:

- $\nabla f(x, y) = 0$ (derivate parziali tutte nulle)
- $H_f(x, y)$ sia **definita positiva**

Condizione **sufficiente** affinché (x, y) sia un punto di **Massimo** per f è che:

- $\nabla f(x, y) = 0$ (derivate parziali tutte nulle)
- $H_f(x, y)$ sia **definita negativa**

10.3.3 Matrice Definita Positiva/Negativa

Il Concetto di Matrice Definita Positiva/Negativa ci serve per distinguere i punti di Massimo, Minimo e di Sella.

DEFINIZIONE

Una Matrice Quadrata si Dice:

- Definita Positiva se tutti gli Autovalori sono > 0
- Semi Definita Positiva se tutti gli Autovalori sono ≥ 0
- Definita Negativa se tutti gli Autovalori sono < 0
- Semi Definita Negativa se tutti gli Autovalori sono ≤ 0

Osservazione Sia $f(x, y)$ una funzione derivabile due volte, allora:

- f è convessa nell'insieme A se e solo se $H_f(x, y)$ è semi definita positiva.
- f è concava nell'insieme A se e solo se $H_f(x, y)$ è semi definita negativa.

10.4 Algoritmi Numerici per i PNL Multivariati

Come per la PNL Monovariata, anche per la Multivaariata esistono degli algoritmi numerici per trovare l'ottimo qual'ora non fosse possibile trovarlo analiticamente.
Gli algoritmi che affronteremo sono:

- Metodo del Gradiente (Steepest Descent)
- Metodo di Newton

Entrambi gli algoritmi utilizzano strategie di tipo **Line Search**.

10.4.1 Gli algorimti Line Search

Questo tipo di algorimto genera una succesione di punti $x_k \in \mathbb{R}^n$ in modo che il punto x_{k+1} é ottenuto a partire dal punto x_k muovendosi lungo una **Direzione di Salita (Massimo) o di Discesa (Minimo)**.

DEFINIZIONE

Dati $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$ con f Derivabile in x.
Definiamo:

- Direzione di Discesa un generico vettore $v \in \mathbb{R}^n$ tale che $\langle v, f(x) \rangle <$

0

- Direzione di Salita un generico vettore $v \in \mathbb{R}^n$ tale che $\langle v, f(x) \rangle > 0$

Dove $\langle\rangle$ indica il Prodotto scalare tra vettori.

10.4.2 Schema generico per il Line Search

Dati $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$ con f Derivabile in x.
Se vogliamo cercare un generico punto di minimo di una funzione, possiamo usare la seguente strategia:

1. Poniamo $k = 0$ e consideriamo un generico punto x^k
2. Determiniamo una Direzione di Discesa d^k
3. Cerchiamo un nuovo punto x^{k+1} lungo la direzione d^k:

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

dove

- k é la generica iterazione
- α^k é una quantitá scalare> 0 chiamata step-size.

Step-Size Lo Step-Size é il valore che determina di quanto ci andremo a spostare nella direzione d_k. Esso é inversamente proporzionale al numero di iterazioni.
Lo step-size α^k corrisponde a:

$$\max/\min\ f(x^k \pm \alpha^k \cdot d^k)$$

e uso un metodo di Max/Min per trovare α^k ottimale, per esempio con una soluzione analitica:

$$\frac{\delta g(\alpha^k)}{\delta \alpha} = 0 \text{ dove } g(\alpha) = f(x^k \pm \alpha^k \cdot d^k)$$

Ovvero Trovato un x^{k+1} che ha α come incognita, lo uso per trovare lo step-size ponendo $g(\alpha) = f(x^{k+1})$ e $\alpha^* = \max\ g(\alpha)$.
$maxg(\alpha)$ lo posso spesso risolvere analiticamente, ponendo $g'(\alpha) = 0$.

10.5 Metodo del Gradiente

Il metodo del gradiente é un algoritmo Line Search e come direzione di crescita d^k usa:

- $d^k = \nabla f(x^k)$ per i problemi di Massimo
- $d^k = -\nabla f(x^k)$ per i problemi di Minimo

E come formula generale per il punto x^{k+1}:

$$x^{k+1} = x^k \pm \alpha^k \cdot \nabla f(x^k)$$

N.B. $\pm\nabla f(x^k)$ é la direzione di crescita, quindi il $\pm$ diventa $+$ se é un problema di Massimo, e $-$ se é un problema di minimo

10.5.1 L'algoritmo

1. Pongo $k = 0$ e considero un generico x^k
2. Calcolo $\nabla f(x^k)$ e pongo la direzione di crescita $d^k = \pm\nabla f(x^k)$
3. Cerco un nuovo punto x^{k+1} lungo la direzione d^k:

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

4. Calcolo α^k come la soluzione di $\frac{df(x^k \pm \alpha^k \cdot d^k)}{d\alpha^k} = 0$

Creiteri di Arresto

- $|f(x^{k+1} - f(x^k)| < \epsilon_1)$ Mi fermo
- $||f(x^{k+1}|| < \epsilon_2)$ Mi fermo, dove $||f||$ é la norma di f.

Osservazioni . Noto che l'algoritmo del gradiente si differenzia da un generico Line Search solo dalla determinazione della direzione di Crescita. Inoltre noto che se $\nexists \alpha^k > 0 \implies \nexists$ una direzione di Salita.

10.6 Metodo di Newton

A differenza del metodo del Gradiente, il metodo di Newton (N-dimensionale) utilizza un'approssimazione quadratica di $f(x)$ per ottenere un nuovo punto.

Sviluppo di Taylor per funzioni a piú variabili:

$$f(x^k + \Delta x) = f(x^k) + \nabla f(x^k) \cdot \Delta x + \frac{1}{2}\Delta x \cdot Hf(x^k)\Delta x$$

In questo caso $x^k + \Delta x = x^{k+1}$

Direzione di Miglioramento Essendo un algoritmo Line Search, Newton segue una direzione di miglioramento per il nuovo punto $(x^k + \Delta x)$ cosí definita:

$$\frac{\delta f(x^k + \Delta x)}{\delta \Delta x} = 0 \rightarrow H(x^k)\Delta x = -\nabla f(x^k)$$

Δx é il vettore spostamento, e in questo caso é l'incognita.

Osservazione Newton si muove ad un punto stazionario del II ordine dello sviluppo in serie di Taylor della funzione.

10.6.1 l'Algoritmo

Tenendo peresente che $\Delta x = x^{k+1} - x^k$

1. Pongo $k = 0$ e considero un generico x^k
2. Calcolo $\nabla f(x^k)$ e $H(x^k)^{-1}$
3. Cerco un nuovo punto x^{k+1}:

$$x^{k+1} = x^k - H(x^k)^{-1} \cdot \nabla f(x^k)$$

Criteri di Arresto Sono gli stessi del metodo del gradiente.

- $|f(x^{k+1} - f(x^k)| < \epsilon_1)$ Mi fermo
- $||f(x^{k+1}|| < \epsilon_2)$ Mi fermo, dove $||f||$ é la norma di f.

10.6.2 Evitare il calcolo della Matrice Inversa

Nell'alogirmto di Newton il calcolo del nuovo punto ci chiede di calcolare $-H_f(x_k, y_k)^{-1}\nabla f(x_k, y_k)$ il che ci richiederebbe di calcolare la matrice Hessiana e Gradiente nel punto corrente, ma anche di calcolare la *Matrice inversa dell'hessiana* nel punto.

Calcolare una matrice inversa é **molto oneroso dal punto di vista computazionale**, esiste peró un modo per evitare di calcolarlo.

Considerazione A noi non interessa sapere il valore di $H_f(x_k, y_k)^{-1}$, bensí ci interessa $-H_f(x_k, y_k)^{-1}\nabla f(x_k, y_k) = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$, ovvero il *vettore spostamento* Δx per aggiornare il punto corrente.
Dalla formula della direzione di miglioramento sappiamo che il vettore Δx puó essere calcolato come soluzione del seguente sistema:

$$H_f(x_k, y_k) \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = -\nabla f(x_k, y_k)$$

In questo modo dobbiamo soltanto risolvere un sistema di equazioni invece di calcolare una matrice inversa, il che é molto meno oneroso computazionalmente.

10.6.3 Newton VS Gradiente

- Newton utilizza sia Gradiente che Hessiana ad ogni iterazione
- Newton se converge, converge molto piú rapidamente del metodo del gradiente ma richiede uno sforzo computazionale molto maggiore.
- Se la funzione é quadratica, Newton converge ad una sola iterazione.
- Nel caso di funzioni piú complesse, la matrice Hessiana risulta piú complessa da calcolare.