2025

Dispense Corso Fondamenti E Metodi Per La Progettazione

UNIVERSITA' DEGLI STUDI DI TRIESTE
VALENTINO PEDIRODA

1 Introduzione

Durante il corso di **Fondamenti Numerici della Progettazione** verranno approfondite in modo sistematico le metodologie di progettazione avanzata basate sull'**ottimizzazione numerica**. L'attenzione sarà rivolta non soltanto agli aspetti teorici, ma anche alle modalità pratiche con cui tali metodi possono essere applicati per affrontare problemi ingegneristici complessi. L'obiettivo centrale è quello di individuare, tra le molte possibili configurazioni progettuali, il sistema che meglio soddisfa un insieme di criteri prestabiliti, che possono riguardare sia le prestazioni funzionali, sia l'efficienza economica e l'impiego ottimale delle risorse disponibili.

Questo approccio consente di affrontare la progettazione in modo più rigoroso e consapevole: attraverso la definizione di **funzioni obiettivo** e di **vincoli progettuali**, si costruisce un modello matematico del problema che può essere risolto mediante strumenti di calcolo avanzati. L'ottimizzazione numerica, in questo contesto, diventa quindi una guida per il progettista, permettendo non solo di **migliorare le prestazioni globali** di un sistema, ma anche di **contenere i costi**, ridurre l'impatto ambientale e aumentare l'affidabilità e la robustezza del progetto.

Inoltre, il corso offrirà una panoramica delle principali tecniche di ottimizzazione, dai metodi deterministici a quelli stocastici ed evolutivi, mettendo in evidenza vantaggi e limiti di ciascun approccio. Particolare enfasi sarà posta sull'applicazione a casi concreti, in modo da mostrare come i principi teorici possano tradursi in **strumenti operativi** capaci di supportare le decisioni progettuali.

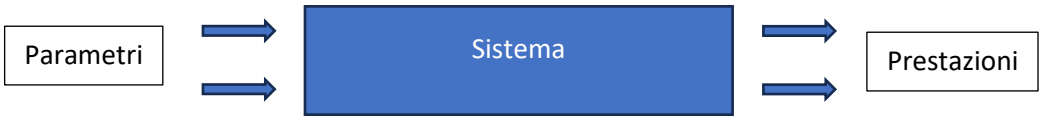
1.1 Concetto di Ottimizzazione Numerica nella Progettazione

Quando si affronta un problema di progettazione ingegneristica, il compito principale consiste nel determinare una configurazione del sistema capace di soddisfare requisiti tecnici e vincoli progettuali in modo ottimale. In altre parole, non si tratta semplicemente di trovare una soluzione valida, ma di individuare la **migliore soluzione possibile** tra un ampio insieme di alternative. È qui che entra in gioco l'**ottimizzazione numerica**, la quale permette di analizzare lo spazio delle possibili configurazioni e di selezionare quella che garantisce le prestazioni più elevate rispetto a criteri come **efficienza, costo, affidabilità e sostenibilità**.

Per rendere più concreti questi concetti, si possono considerare alcuni esempi. Nella **progettazione di un aeromobile**, il progettista deve bilanciare diversi aspetti: il numero massimo di passeggeri trasportabili, il consumo di carburante su una determinata tratta, la capacità di carico, i limiti strutturali e le normative di sicurezza. La ricerca di una soluzione ottimale non consiste solo nel massimizzare un singolo parametro, ma nel trovare un compromesso efficace che tenga conto di obiettivi potenzialmente in conflitto tra loro.

Un altro caso emblematico riguarda la **progettazione di un motore automobilistico**. Qui l'obiettivo può essere quello di massimizzare la potenza e le prestazioni dinamiche del veicolo, riducendo al contempo i consumi di carburante e le emissioni inquinanti. Anche in questo caso il problema si traduce in una sfida di **ottimizzazione multi-obiettivo**, in cui occorre individuare soluzioni che offrano il miglior equilibrio possibile tra desideri dell'utente finale, vincoli tecnologici e limiti ambientali.

Dal punto di vista operativo, tutti questi obiettivi vengono **formalizzati in termini matematici**, ossia tradotti in **funzioni obiettivo** da massimizzare o minimizzare, e in **vincoli** che delimitano lo spazio delle soluzioni ammissibili. L'ottimizzazione numerica consente quindi di esplorare questo spazio in maniera sistematica e rigorosa, offrendo al progettista un supporto essenziale per prendere decisioni consapevoli e basate su criteri quantitativi, piuttosto che affidarsi a tentativi empirici o scelte intuitive.

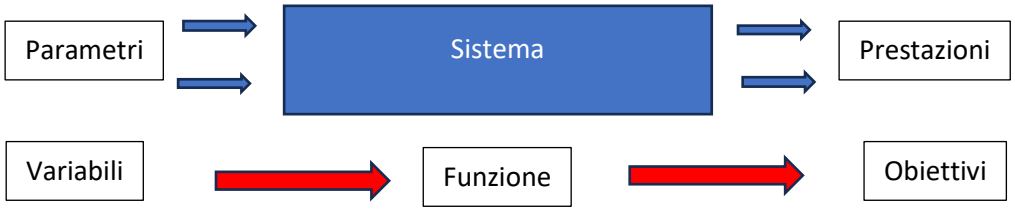


La progettazione ingegneristica numerica si può schematizzare come un problema di massimizzazione di una funzione obiettivo. Il sistema è definito da:

1. **Parametri di progetto:** variabili libere che definiscono la geometria e le caratteristiche del sistema.
2. **Prestazioni attese:** gli obiettivi che il sistema deve soddisfare.
3. **Vincoli progettuali:** limitazioni imposte dalle normative, dai materiali o da altre esigenze tecniche.
4. **Funzione obiettivo:** espressione matematica che rappresenta il criterio di ottimizzazione.

Modificando i parametri del sistema, si ottengono diverse configurazioni, e il compito dell'ottimizzazione numerica è identificare quella più performante.

Andare a progettare significa andare a determinare le variabili che massimizzano la mia funzione obiettivo.



$$Max\ F: \mathcal{R}^n \Longrightarrow \mathcal{R}^m$$

1.2 Concetto di Computer Aided Optimization (CEO)

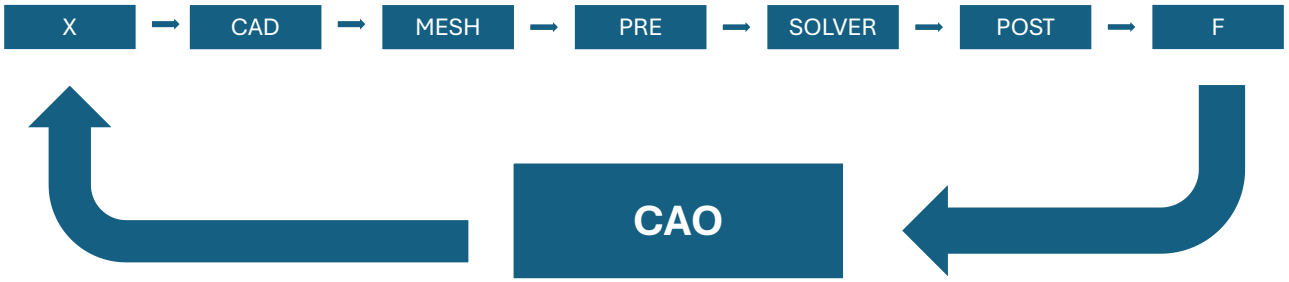
Oggi la progettazione ingegneristica non si limita più alla sola fase di concezione teorica o alla realizzazione di prototipi fisici: il cuore del processo è rappresentato dalla **simulazione numerica automatizzata**. Grazie a questo approccio, denominato **Computer Aided Optimization (CAO)**, è possibile integrare in un unico flusso strumenti di modellazione, analisi e ottimizzazione, con l'obiettivo di individuare in tempi ridotti le configurazioni progettuali più promettenti.

Il procedimento tipico parte dalla **modellazione geometrica** del sistema, realizzata con software di **Computer Aided Design (CAD)**, in cui vengono definiti i parametri fondamentali del progetto. Su questo modello si innestano poi i **solver numerici** – ad esempio per l'analisi strutturale, termica o fluidodinamica – che permettono di valutare in modo dettagliato il comportamento del sistema sotto differenti condizioni operative.

I risultati ottenuti dalle simulazioni non hanno soltanto un valore descrittivo, ma diventano essi stessi **input per il processo di ottimizzazione**: parametri come resistenza, deformazioni, efficienza energetica, costi di produzione o emissioni ambientali possono infatti essere trasformati in **funzioni obiettivo**. A questo punto entra in gioco l'elemento centrale del CAO: il **loop di ottimizzazione automatica**. Algoritmi dedicati – che spaziano dai metodi deterministici basati sui gradienti fino agli approcci stocastici ed evolutivi – modificano

iterativamente le variabili di progetto, analizzano i risultati delle nuove simulazioni e si muovono progressivamente verso il miglior compromesso possibile tra gli obiettivi prefissati.

Un aspetto cruciale di questo processo è l'**automazione completa del ciclo di simulazione–ottimizzazione**: il progettista non interviene più manualmente ad ogni passo, ma definisce inizialmente gli obiettivi, i vincoli e i parametri da esplorare, lasciando che il sistema informatico conduca migliaia di simulazioni in maniera autonoma. In questo modo è possibile ottenere una visione globale dello **spazio delle soluzioni**, individuando configurazioni che spesso sarebbero difficili da concepire con un approccio tradizionale basato su prove ed errori.



1.3 Prima parte: Variabili/Parametrizzazione/DOE/Analisi Statistica

Gli ingegneri moderni si trovano spesso a lavorare con **sistemi complessi**, in cui il numero di variabili in gioco può diventare estremamente elevato: parametri geometrici, condizioni al contorno, proprietà dei materiali, vincoli di funzionamento e requisiti normativi. Una tale abbondanza di fattori rende difficile sia l'analisi diretta del problema, sia l'applicazione immediata di algoritmi di ottimizzazione, poiché lo **spazio delle soluzioni** risulta troppo vasto per essere esplorato in modo esaustivo.

Per affrontare questa difficoltà, una strategia fondamentale è rappresentata dalle **tecniche di Design of Experiments (DOE)**. Questi metodi consentono di progettare in maniera razionale un numero ridotto ma rappresentativo di simulazioni o prove, con l'obiettivo di identificare quali variabili siano davvero influenti e in che misura. In pratica, il DOE permette di trasformare un problema inizialmente ad alta dimensionalità in un modello più compatto e gestibile, mantenendo però intatte le informazioni essenziali. Tecniche come i piani fattoriali, i piani frazionari o i metodi di campionamento statistico (ad esempio il Latin Hypercube Sampling) rendono possibile un'esplorazione efficiente dello spazio dei parametri, riducendo drasticamente i costi computazionali e sperimentali.

Parallelamente, un altro strumento cruciale è la **parametrizzazione geometrica**. Invece di descrivere la geometria di un sistema attraverso una moltitudine di punti o coordinate, essa viene ricondotta a un **insieme limitato di parametri significativi** (ad esempio raggi di curvatura, angoli, spessori, lunghezze caratteristiche). Questo approccio consente di rappresentare variazioni geometriche anche complesse in maniera compatta, semplificando il compito degli algoritmi di ottimizzazione e permettendo di esplorare configurazioni diverse senza dover ridefinire da zero l'intero modello.

L'unione di DOE e parametrizzazione geometrica costituisce quindi un passaggio chiave nel moderno processo di **ottimizzazione ingegneristica**: da un lato riduce la complessità del problema, dall'altro lo rende più interpretabile e controllabile, aprendo la strada a una ricerca sistematica ed efficiente della soluzione ottimale.

1.4 Seconda parte: algoritmi di ottimizzazione/multiobiettivo/Supporto alle decisioni

Gli **algoritmi di ottimizzazione** rappresentano il cuore del processo di ricerca della configurazione progettuale più adatta. La loro funzione è quella di guidare l'esplorazione dello spazio delle possibili soluzioni, individuando progressivamente quelle che soddisfano al meglio gli obiettivi definiti e i vincoli imposti. Esistono diverse famiglie di algoritmi, ciascuna con peculiarità e campi di applicazione specifici.

Gli **algoritmi deterministici**, come il metodo del gradiente o le tecniche basate sulla programmazione lineare e non lineare, utilizzano le informazioni locali della funzione obiettivo (ad esempio derivate e curvature) per muoversi in direzione della soluzione ottimale. Essi risultano molto efficienti in termini di rapidità di convergenza, soprattutto quando il problema è ben condizionato e la funzione obiettivo presenta un unico minimo. Tuttavia, possono incontrare difficoltà in presenza di spazi di ricerca complessi, con molteplici minimi locali.

Per affrontare scenari di questo tipo si ricorre spesso a **algoritmi stocastici ed evolutivi**, ispirati ai meccanismi della selezione naturale e dell'evoluzione biologica. Tecniche come gli **algoritmi genetici**, le **strategie evolutive** consentono di esplorare vaste regioni dello spazio delle soluzioni, evitando di rimanere intrappolati in ottimi locali. Pur richiedendo un numero maggiore di valutazioni rispetto ai metodi deterministici, essi risultano particolarmente adatti a problemi di grande complessità e non lineari, tipici dell'ingegneria moderna.

In molti casi la progettazione non si riduce a un singolo obiettivo, ma richiede di considerare **più criteri contemporaneamente**, spesso tra loro contrastanti: è il campo dell'**ottimizzazione multi-obiettivo**. Alcuni esempi classici chiariscono questa esigenza:

- nella definizione di un **profilo aerodinamico**, occorre bilanciare la **portanza**, che garantisce la capacità di volo, e la **resistenza**, che influisce sui consumi;
- nella progettazione di un **motore**, si cerca di massimizzare la **potenza erogata**, minimizzando allo stesso tempo il **consumo di carburante** e le **emissioni inquinanti**;
- per una **struttura ingegneristica**, l'obiettivo è massimizzare la **resistenza meccanica**, riducendo però al minimo il **peso complessivo** e i costi di produzione.

In questo contesto non esiste una singola soluzione "migliore" in senso assoluto, ma un insieme di soluzioni ottime dette **soluzioni di Pareto**, ognuna delle quali rappresenta un diverso compromesso tra gli obiettivi in conflitto. La scelta finale ricade dunque sul progettista o sul decisore, che deve valutare quale equilibrio risponde meglio alle priorità del problema specifico.

Per supportare questa fase si adottano metodologie di **Multi-Criteria Decision Making (MCDM)**, strumenti che consentono di analizzare e confrontare le diverse soluzioni ottime sulla base di preferenze, pesi o indicatori di prestazione. Tecniche come l'**Analytic Hierarchy Process (AHP)**, il **metodo TOPSIS** permettono di strutturare il processo decisionale in modo trasparente e razionale, riducendo la soggettività e facilitando la selezione della configurazione più adatta.

1.5 Terza parte: superfici di risposta/Intelligenza artificiale

Le simulazioni numeriche possono essere molto costose in termini di tempo di calcolo. Per accelerare il processo di ottimizzazione si adottano tre strategie principali:

1. **Uso di calcolo parallelo e hardware avanzato**, per velocizzare le simulazioni.
2. **Selezione di algoritmi efficienti**, per migliorare la convergenza dell'ottimizzazione.
3. **Utilizzo di superfici di risposta**, che creano modelli approssimati delle prestazioni del sistema basati su dati interpolati.

Le **simulazioni numeriche** costituiscono oggi lo strumento centrale della progettazione ingegneristica avanzata, ma presentano un limite significativo: possono risultare estremamente **onerose in termini di tempo di calcolo** e di risorse computazionali. Un singolo ciclo di simulazione complessa – ad esempio un'analisi fluidodinamica tridimensionale o uno studio termomeccanico accoppiato – può richiedere ore, se non giorni, di elaborazione anche su hardware dedicato. Questo rallenta sensibilmente l'intero processo di ottimizzazione, che richiede centinaia o migliaia di valutazioni per esplorare lo spazio delle soluzioni.

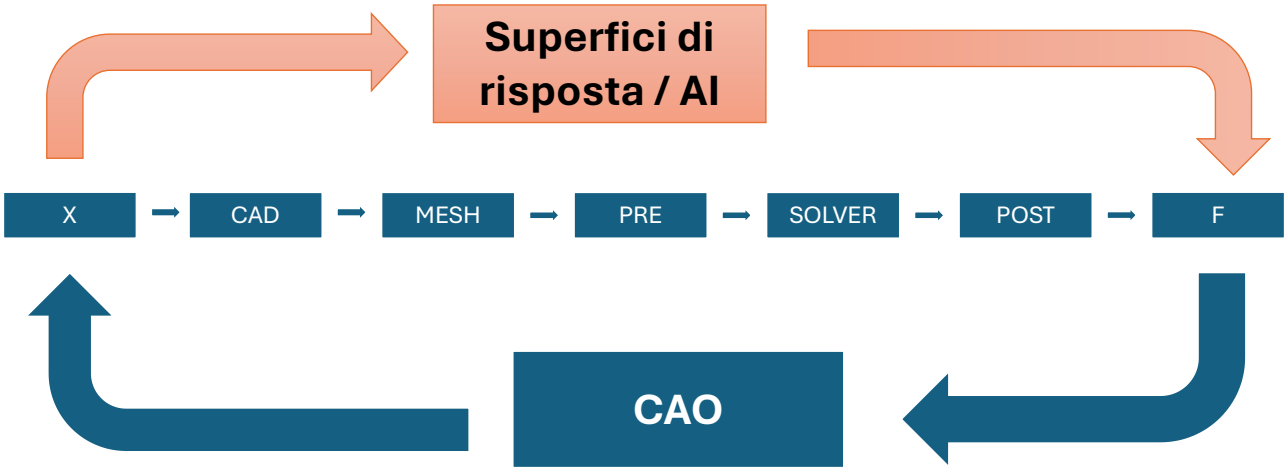
Per affrontare tale sfida si adottano tre strategie principali:

1. **Calcolo parallelo e hardware avanzato**: grazie ai supercalcolatori, alle GPU e alle architetture di calcolo distribuito, è possibile suddividere il carico computazionale e svolgere in parallelo molteplici simulazioni, riducendo drasticamente i tempi di attesa.
2. **Algoritmi efficienti di ottimizzazione**: la scelta dell'algoritmo gioca un ruolo cruciale. Metodi numerici in grado di convergere più rapidamente verso la soluzione ottimale, riducendo il numero di valutazioni necessarie, permettono di contenere i costi computazionali e aumentare l'affidabilità del processo.
3. **Superfici di risposta (Response Surfaces)**: attraverso tecniche di interpolazione e approssimazione, si costruiscono **modelli surrogati** che descrivono in maniera semplificata il comportamento del sistema. In questo modo non è necessario eseguire una simulazione complessa a ogni iterazione, poiché le prestazioni vengono stimate rapidamente dal modello approssimato. Tecniche come il Kriging, le regressioni polinomiali o i modelli basati su reti neurali permettono di generare superfici di risposta accurate, che costituiscono un compromesso ideale tra precisione e rapidità di calcolo.

A queste strategie consolidate si affianca oggi una rivoluzione legata all'uso dell'**Intelligenza Artificiale (IA)** e, in particolare, del **Machine Learning**. L'IA consente di:

- **Predire il comportamento del sistema** senza ricorrere ogni volta a simulazioni dettagliate, grazie a modelli predittivi addestrati su grandi quantità di dati;
- **Ottimizzare in modo adattivo**, con algoritmi capaci di apprendere dalle iterazioni precedenti e di concentrare la ricerca nelle regioni più promettenti dello spazio delle soluzioni;
- **Riconoscere pattern e correlazioni nascoste** nei dati di simulazione, migliorando così sia l'efficienza sia la robustezza dell'intero processo di ottimizzazione.

Grazie a queste innovazioni, l'**ottimizzazione numerica** si conferma come una disciplina essenziale dell'ingegneria moderna. Essa non solo consente di progettare sistemi più performanti, ma riduce significativamente tempi e costi di sviluppo, rendendo possibili analisi e soluzioni che fino a pochi anni fa sarebbero state impraticabili.



Il corso fornirà agli studenti gli strumenti teorici e pratici per affrontare problemi ingegneristici complessi, guidandoli all'uso di **metodologie avanzate di simulazione, ottimizzazione e intelligenza artificiale**.
L'obiettivo sarà quello di sviluppare la capacità di integrare modelli numerici, algoritmi di calcolo e tecniche innovative in un approccio coerente e moderno alla progettazione.

2 Definizioni

2.1 Variabili Di Progetto

Le **variabili di progetto** rappresentano un insieme di grandezze di natura quantitativa che subiscono modifiche durante il processo di progettazione e che, al contempo, contribuiscono alla definizione e caratterizzazione dei sistemi oggetto di studio.

Nell'ambito dell'analisi delle variabili di progetto, una distinzione essenziale è quella tra:

- **Variabili misurabili**: comprendono tutte le grandezze che possono essere espresse e ordinate secondo un criterio numerico. Esempi tipici di tali variabili includono lunghezza, angolo, peso, e molte altre proprietà fisiche. Le variabili misurabili si suddividono ulteriormente in:
    - **Variabili continue**: assumono un numero infinito di valori all'interno di un intervallo definito (ad esempio, la lunghezza di un oggetto può teoricamente assumere qualsiasi valore reale all'interno di un certo intervallo).
    - **Variabili discrete**: possono assumere solo determinati valori numerici distinti e separati tra loro (ad esempio, il numero di cilindri in un motore è un valore intero e non frazionabile).
- **Variabili di categoria**: si riferiscono a quelle grandezze per le quali non è possibile stabilire un criterio di ordinamento numerico. Un esempio tipico riguarda materiali quali vetro e legno: non esiste una relazione quantitativa che permetta di classificare il vetro rispetto al legno in termini numerici.

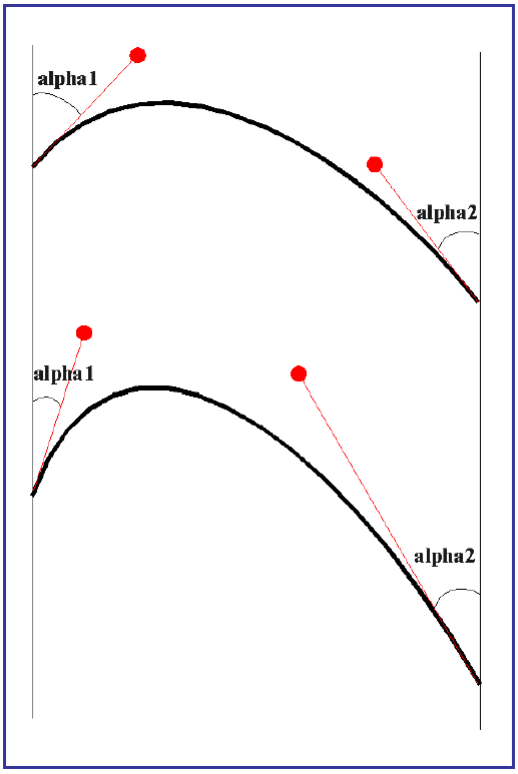
Questa distinzione è di fondamentale importanza, poiché l'idoneità di determinati algoritmi dipende dalla natura delle variabili considerate. Alcuni algoritmi risultano applicabili esclusivamente alle variabili misurabili continue, altri sono progettati per operare unicamente su variabili misurabili discrete, mentre alcuni non sono compatibili con variabili di categoria. Tuttavia, esistono anche algoritmi in grado di gestire simultaneamente tutte le tipologie di variabili.

2.2 Parametrizzazione

Il concetto di **parametrizzazione** risulta strettamente connesso alle **variabili di progetto** e si riferisce al processo mediante il quale un sistema oggetto di studio viene modificato attraverso l'assegnazione e la variazione controllata di specifici parametri. Tale operazione consente di definire, adattare e ottimizzare le caratteristiche del sistema in funzione degli obiettivi progettuali.

La parametrizzazione può riguardare diversi aspetti del sistema, tra cui la sua configurazione geometrica, le proprietà fisiche, i vincoli strutturali e altri fattori determinanti. Ad esempio, nell'ambito dell'ingegneria meccanica o strutturale, la parametrizzazione geometrica consente di variare dimensioni, forme e proporzioni degli elementi costitutivi, permettendo così di valutare l'impatto di tali modifiche sulle prestazioni globali del sistema. Analogamente, in altri contesti disciplinari, la parametrizzazione può essere applicata per analizzare e ottimizzare il comportamento di sistemi complessi attraverso la regolazione dei parametri di input.

La parametrizzazione costituisce un principio fondamentale della progettazione ingegneristica e scientifica, poiché consente di esplorare diverse configurazioni di un sistema e di selezionare la soluzione ottimale in base a criteri prestazionali, economici o funzionali.



2.3 Funzione Obiettivo

Le **funzioni obiettivo** rappresentano le funzioni matematiche che si desidera ottimizzare nell'ambito di un processo progettuale. L'ottimizzazione in ingegneria consiste nella ricerca della configurazione ottimale di un sistema che soddisfi determinate caratteristiche desiderate. In altre parole, progettare significa definire con precisione le caratteristiche richieste e individuare la configurazione ingegneristica che meglio le soddisfa.

Affinché sia possibile confrontare le diverse soluzioni progettuali, è necessario disporre di un criterio oggettivo di **classificazione**, ossia un parametro che consenta di valutare il grado di adeguatezza di ciascun sistema rispetto agli obiettivi di progetto. La formalizzazione numerica di tale criterio è definita come **funzione obiettivo**. Quest'ultima rappresenta quindi il parametro quantitativo che misura quanto una determinata soluzione sia vicina all'ottimizzazione desiderata.

Esistono numerosi tipi di funzioni obiettivo, a seconda del contesto applicativo. Ad esempio:

- **Rendimento termodinamico**: nell'ottimizzazione di una turbina, si può voler massimizzare il rendimento termodinamico. In questo caso, il rendimento diventa la funzione obiettivo, in quanto consente di confrontare e classificare i diversi sistemi sulla base dell'efficienza energetica.
- **Minimizzazione del peso**: nella progettazione di una struttura, può essere richiesto di ridurre il peso complessivo del sistema. In tale scenario, il peso diventa la funzione obiettivo poiché rappresenta il criterio di distinzione tra le varie configurazioni strutturali.
- **Ottimizzazione della geometria della biella**: nella progettazione di una biella con tensioni interne minime, l'obiettivo è determinare la configurazione geometrica ideale che minimizzi le sollecitazioni interne. Si tratta di un processo di **parametrizzazione geometrica**, in cui il criterio discriminante è il valore delle tensioni interne, che pertanto diventa la funzione obiettivo.
- **Minimizzazione della resistenza aerodinamica di un'ala**: nel caso dell'ottimizzazione aerodinamica di un'ala, si desidera ridurre al minimo la resistenza aerodinamica. Anche in questo caso, si opera

attraverso una parametrizzazione geometrica, e il criterio di classificazione delle varie configurazioni è il valore del coefficiente di resistenza aerodinamica **Cd**, che costituisce la funzione obiettivo.

In sintesi, la funzione obiettivo rappresenta un concetto fondamentale nei processi di progettazione e ottimizzazione ingegneristica, in quanto consente di valutare quantitativamente l'efficacia di una soluzione progettuale e di selezionare la configurazione più adatta in base agli obiettivi prestabiliti.

2.3.1 Definizione ottimizzazione numerica mono obiettivo

$$\text{Trovare } X = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} \text{ che massimizza } f(X)$$

con i vincoli

$$g_j(X) < 0,\ j = 1,\ldots,m$$

$$l_j(X) = 0,\ j = 1,\ldots,p$$

2.3.2 Definizione ottimizzazione numerica multi obiettivo

$$\text{Trovare } X = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix}$$

che massimizza $f_j(X)\ j = 1,\ldots,q$

con i vincoli

$$g_j(X) < 0,\ j = 1,\ldots,m$$

$$l_j(X) = 0,\ j = 1,\ldots,p$$

2.4 Massimizzazione E Minimizzazione Di Una Funzione Obiettivo

Nell'ambito dell'ottimizzazione matematica e ingegneristica, i concetti di **massimizzazione** e **minimizzazione** di una funzione obiettivo sono strettamente correlati e, dal punto di vista formale, del tutto equivalenti. La distinzione tra le due operazioni dipende unicamente dalla direzione dell'ottimizzazione desiderata: nel caso della massimizzazione, si ricerca la configurazione del sistema che produce il valore massimo della funzione obiettivo, mentre nella minimizzazione si cerca la configurazione che restituisce il valore minimo.

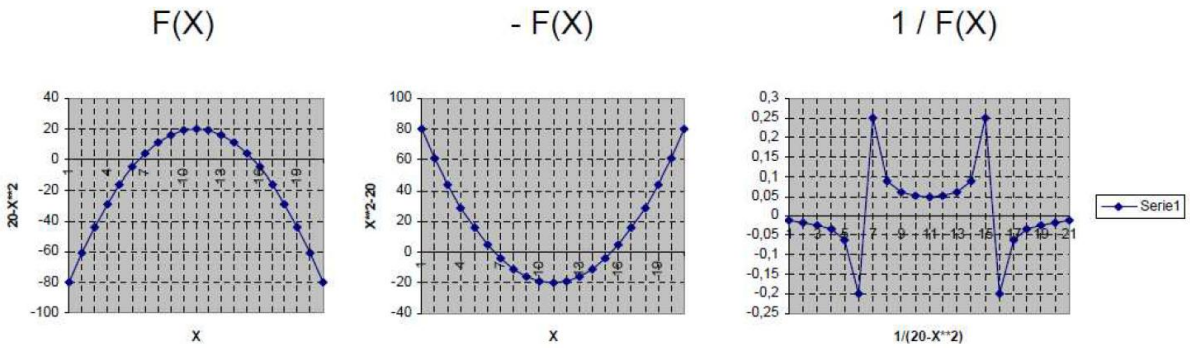
Quando si dispone di un algoritmo progettato per **massimizzare** una funzione obiettivo, è possibile utilizzarlo anche per **minimizzarla**, applicando una semplice trasformazione alla funzione stessa. Più precisamente, data

una funzione obiettivo F(x), si può convertire un problema di minimizzazione in uno di massimizzazione (o viceversa) moltiplicando la funzione per −1, ottenendo così:

Fnew(x) = -F(x)

Questa operazione inverte il criterio di ottimizzazione: il massimo della funzione F(x) diventa il minimo della funzione Fnew(x), e viceversa.

Questa proprietà è particolarmente utile nell'implementazione di algoritmi di ottimizzazione numerica, poiché consente di utilizzare lo stesso metodo computazionale indipendentemente dalla natura del problema. Ad esempio, un algoritmo progettato per trovare il massimo di una funzione può essere impiegato per determinare il minimo di un'altra funzione semplicemente invertendone il segno.



Da notare come nel terzo esempio 1/F(x) i risultati della trasformazione non sarebbero coerenti.

2.5 La Definizione dei Vincoli nell'Ottimizzazione Ingegneristica

Da un punto di vista ingegneristico, la definizione dei **vincoli** è spesso considerata più cruciale rispetto alla definizione degli obiettivi. Questo concetto è particolarmente rilevante quando si affrontano problemi di **ottimizzazione mono obiettivo vincolata**, dove l'ottimizzazione avviene sotto l'imposizione di vincoli che limitano le possibili soluzioni.

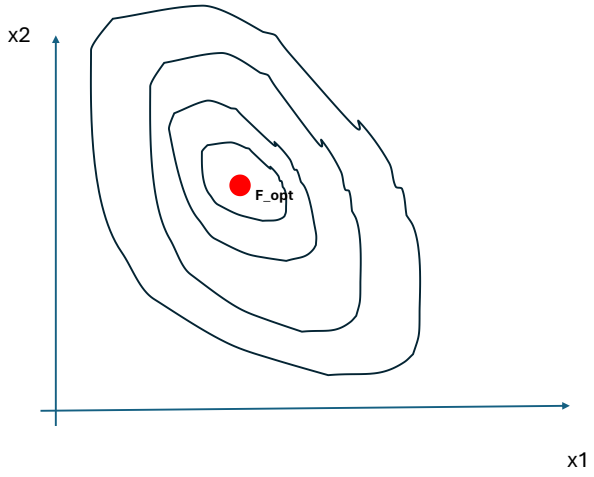
Un esempio pratico per chiarire questo concetto è il seguente. Immaginiamo di voler progettare un sistema in cui una determinata variabile t deve essere maggiore di un valore prefissato $t*$. In questo caso, la condizione di vincolo è rappresentata da $t>t*$, dove si esprime chiaramente un limite inferiore su t, escludendo tutte le soluzioni che non soddisfano tale condizione. La definizione del vincolo, in questo caso, è determinante, in quanto stabilisce una restrizione che le soluzioni ottimali devono rispettare.

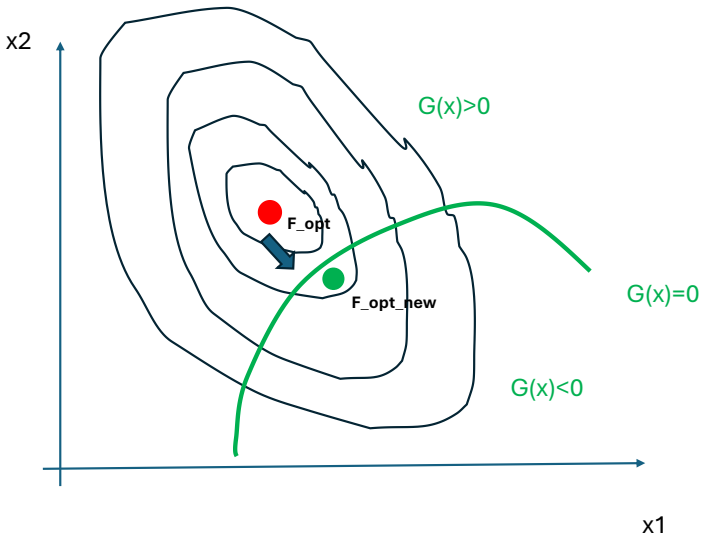
Da un punto di vista ingegneristico, spesso è più importante definire chiaramente ciò che non si vuole nel sistema, piuttosto che fissare direttamente gli obiettivi. In altre parole, il "non volere" è il principale strumento di selezione delle soluzioni, poiché i vincoli descrivono le limitazioni strutturali, fisiche o prestazionali che il sistema non deve superare. Ad esempio, un vincolo potrebbe essere rappresentato dalla necessità che la temperatura di un componente non superi un certo valore, mentre l'obiettivo potrebbe essere ottimizzare altre caratteristiche, come la massimizzazione dell'efficienza. La definizione rigorosa dei vincoli permette di focalizzarsi su soluzioni che rispettano questi limiti, evitando configurazioni irrealizzabili o pericolose.

2.5.1 La Relazione tra Funzione Obiettivo e Vincoli nell'Ottimizzazione

Supponiamo di avere un problema di ottimizzazione non vincolata, nel quale si vuole massimizzare una funzione f(x1,x2), dove x1e x2 sono le variabili di progettazione. Tracciando le **isovalue** o **isolivello** della funzione f(x1,x2), è possibile identificare il punto in cui la funzione raggiunge il massimo, che si trova all'interno di un dominio di ricerca che non è limitato da alcuna restrizione. Supponiamo che il massimo della funzione sia localizzato in un punto specifico, come ad esempio in un quadrato delimitato.

Tuttavia, nel caso di un **problema vincolato**, l'introduzione di vincoli cambia completamente la configurazione del problema. Immaginiamo ora di aggiungere due vincoli alla nostra funzione obiettivo, ad esempio g(x1,x2)≤0 e h(x1,x2)=0, dove g(x1,x2)e h(x1,x2) sono delle funzioni che impongono delle restrizioni sulle variabili di progettazione. A questo punto, le **isovalue** della funzione obiettivo dovranno essere sovrapposte con le **isovalue** dei vincoli, focalizzandosi solamente sulle aree dove i vincoli sono soddisfatti, ovvero dove g(x1,x2)=0 e h(x1,x2)=0h. Questo porterà a un cambiamento nella soluzione ottimale, poiché il dominio delle soluzioni ammissibili si riduce, e di conseguenza il massimo della funzione obiettivo potrebbe non coincidere più con il massimo inizialmente identificato in un'ottimizzazione non vincolata. L'introduzione dei vincoli non solo modifica la soluzione, ma spesso diventa il fattore che "domina" l'intero processo di ottimizzazione. In altre parole, i vincoli definiscono le limitazioni all'interno delle quali è possibile trovare la soluzione ottimale, riducendo il numero di possibili soluzioni accettabili.





2.5.2 La Fase Esplorativa e il Cambiamento degli Obiettivi e Vincoli

Nel processo di ottimizzazione, generalmente si procede in fasi successive. All'inizio, si esegue una **fase esplorativa**, in cui si cerca di identificare un ampio insieme di soluzioni potenziali, senza focalizzarsi su una singola configurazione. Successivamente, nelle fasi successive, la ricerca della soluzione ottimale viene affinata, rivedendo costantemente le scelte fatte inizialmente. Questo significa che, mentre si evolve nel processo di ottimizzazione, potrebbero esserci modifiche nei vincoli e negli obiettivi. Ciò accade perché, con l'acquisizione di nuove informazioni e una comprensione sempre più dettagliata del comportamento del sistema, alcune variabili che inizialmente erano trattate come obiettivi possono essere trasformate in vincoli, e viceversa.

Un esempio potrebbe essere una situazione in cui un obiettivo inizialmente definito (come il massimo rendimento di una macchina) venga trasformato in un vincolo quando si scopre che, per motivi di sicurezza o efficienza operativa, il rendimento massimo non può superare una certa soglia. Al contrario, un vincolo inizialmente imposto, come una restrizione sulle dimensioni di un componente, potrebbe essere modificato e trasformato in un obiettivo più flessibile se, nel corso dell'ottimizzazione, emerge che una certa tolleranza è ammissibile senza compromettere le performance del sistema.

Questo continuo adattamento degli obiettivi e dei vincoli durante l'ottimizzazione evidenzia l'importanza di disporre di una **piattaforma di ottimizzazione informatica** avanzata, che consenta di modificare dinamicamente variabili e obiettivi nelle diverse fasi del processo, per riflettere in tempo reale le scoperte e le necessità emergenti.

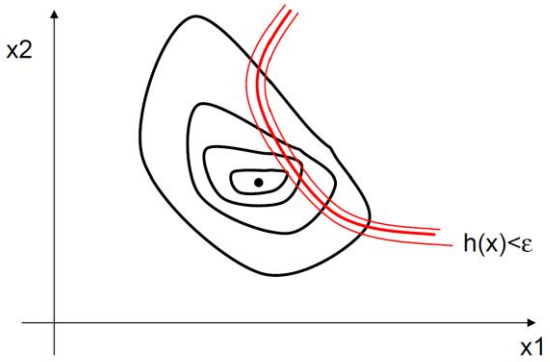
2.5.3 Vincoli di Uguaglianza e Diseguaglianza

Dal punto di vista numerico e computazionale, i vincoli possono essere suddivisi in **vincoli di uguaglianza** e **vincoli di diseguaglianza**. Un vincolo di uguaglianza impone che una determinata funzione $h(x)=0$, mentre un vincolo di diseguaglianza richiede che una funzione $g(x)$ sia inferiore o uguale a zero, ossia $g(x)\leq 0$.

Da un punto di vista ingegneristico, tuttavia, i **vincoli di uguaglianza** sono raramente applicabili in modo diretto. Questo perché le soluzioni esatte che soddisfano un vincolo di uguaglianza sono difficili da ottenere in un contesto numerico, dato che in pratica non esistono mai soluzioni perfette che soddisfano una condizione di uguaglianza esatta. Per esempio, non è realistico impostare una condizione come $CL=0.12$, poiché i metodi numerici non possono garantire che il valore di una funzione sia esattamente uguale a un numero fisso.

Per ovviare a questo problema, i vincoli di uguaglianza vengono comunemente trasformati in **vincoli di diseguaglianza** con una **tolleranza ammissibile**. In altre parole, un vincolo inizialmente espresso come $h(x)=0$ viene trasformato in $h(x)\leq\varepsilon$, dove ε rappresenta una piccola tolleranza accettabile che consente di risolvere numericamente il problema senza cercare una soluzione esatta ma in modo sufficientemente preciso. Questo approccio consente di trattare i vincoli di uguaglianza in modo pratico e numericamente stabile, pur mantenendo il rigore ingegneristico nelle restrizioni imposte al sistema.

In sintesi, la definizione e la gestione dei vincoli nell'ottimizzazione ingegneristica sono fondamentali non solo per la ricerca di soluzioni ottimali, ma anche per l'adattamento delle soluzioni alle reali capacità e limitazioni del sistema. Il trattamento dei vincoli, sia di uguaglianza che di diseguaglianza, è un aspetto critico per l'efficacia dell'ottimizzazione in contesti pratici e applicati.

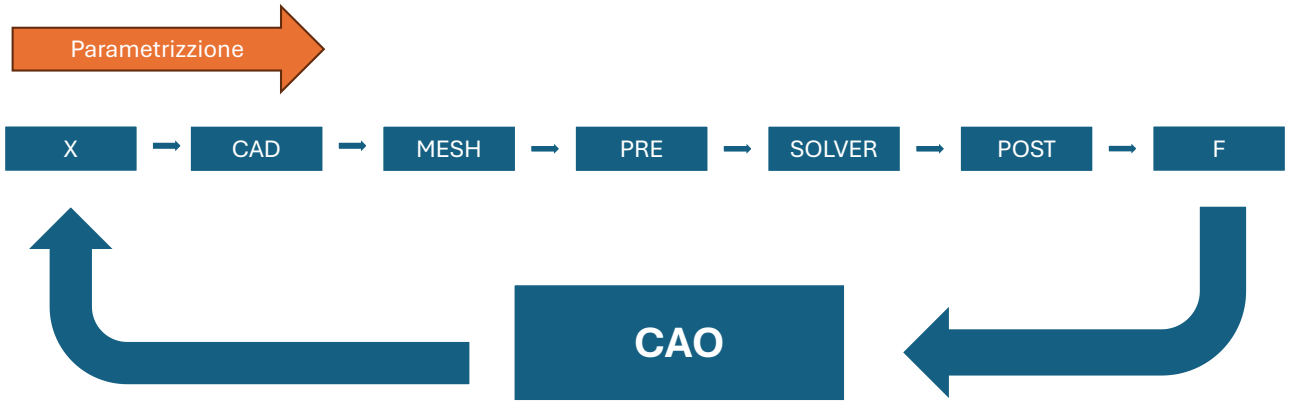

x2
h(x)<ε
x1

3 Parametrizzazione geometrica

Una **geometria parametrica** è una rappresentazione geometrica che dipende da un insieme di parametri variabili; modificando questi parametri, la forma e le dimensioni della geometria si aggiornano automaticamente, consentendo di individuare una configurazione ottimale per il sistema in studio.

Negli anni passati, la definizione di geometrie parametriche veniva effettuata attraverso la programmazione, richiedendo agli ingegneri di scrivere codice in linguaggi come Fortran o C per descrivere matematicamente la geometria del sistema in esame.

Attualmente, la creazione e la gestione delle geometrie parametriche non vengono più realizzate attraverso codici sviluppati autonomamente, ma sono integrate all'interno di software commerciali di **Computer-Aided Design (CAD) parametrico**. In questi strumenti, l'utente definisce un insieme di parametri iniziali e, a ogni loro variazione, il software aggiorna automaticamente l'intera geometria, garantendo un processo più intuitivo ed efficiente rispetto alla programmazione tradizionale.



Di conseguenza, il flusso di lavoro è passato da un approccio basato sulla scrittura di codice manuale a un'interazione diretta con interfacce grafiche avanzate, che facilitano la definizione e la modifica delle geometrie parametriche.

Questo capitolo esamina la teoria alla base della parametrizzazione geometrica, con un focus specifico sulla rappresentazione parametrica delle curve, al fine di fornire una comprensione più approfondita dei sistemi geometrici in esame.

3.1 Curva di Bezier

Il principio fondamentale della parametrizzazione geometrica risiede nell'utilizzo dei **punti di controllo**, ovvero punti geometrici specifici che determinano la forma della struttura modellata. La modifica della posizione o delle caratteristiche di questi punti consente di alterare l'intera geometria in modo controllato e coerente, rendendo la modellazione più flessibile e adattabile alle esigenze progettuali.

La curva più comunemente utilizzata nella parametrizzazione geometrica è una **curva parametrica** definita a partire da un insieme di **punti di controllo**. La variazione della posizione di questi punti determina la modifica dell'intera curva, consentendo un controllo preciso della sua forma.

Uno degli strumenti più noti per la rappresentazione di curve parametriche è la **curva di Bézier**, la quale è ampiamente impiegata in numerosi ambiti dell'ingegneria, della grafica computazionale e della

modellazione geometrica. Supponiamo di considerare una curva di Bézier di grado due, definita da tre punti di controllo b0, b1, b2. Una proprietà fondamentale di questa curva è che essa passa esattamente per il primo e l'ultimo punto di controllo b0 e b2, mentre il punto intermedio b1 influenza la forma della curva senza necessariamente appartenervi. La variazione della posizione di b1 determina un cambiamento nella configurazione della curva, alterandone la curvatura e l'orientazione complessiva.

Attraverso la parametrizzazione geometrica, è possibile associare ai punti di controllo b0,b1 e b2 delle variabili regolabili, trasformandoli in parametri del sistema. La modifica di uno qualsiasi di questi parametri comporta una variazione dell'intera geometria della curva, offrendo così un metodo efficace per l'adattamento delle forme in base a specifiche esigenze progettuali.

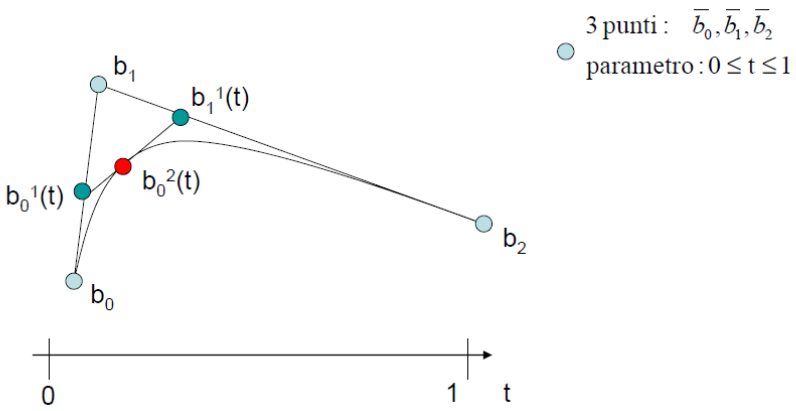
Un ulteriore passo nell'ottimizzazione della geometria parametrica consiste nell'impiego di **algoritmi di ottimizzazione** per determinare la configurazione ottimale dei parametri della curva. L'obiettivo di tali algoritmi è individuare le posizioni ideali di b0, b1 e b2 affinché il sistema soddisfi determinate prestazioni desiderate, come ad esempio minimizzare un errore di approssimazione, garantire la massima aerodinamicità o migliorare l'estetica del design. Questo approccio è ampiamente utilizzato in applicazioni ingegneristiche e di design computazionale, dove la flessibilità e il controllo sulla geometria risultano cruciali.

3.1.1 Algoritmo De Casteljan

L'**algoritmo di De Casteljau** è uno strumento fondamentale per comprendere il processo di generazione di una curva di Bézier. Sebbene non venga direttamente utilizzato per la definizione analitica della curva, esso rappresenta un metodo ricorsivo per la sua costruzione e fornisce un'interpretazione geometrica intuitiva del suo tracciamento.

Un aspetto fondamentale delle curve di Bézier è la relazione tra il numero di punti di controllo e il grado della curva. Formalmente, data una serie di **n punti di controllo**, il grado della corrispondente curva di Bézier è **n−1**. Questo significa, ad esempio, che una curva di Bézier definita da tre punti di controllo ha grado 2 (ossia è una parabola), mentre una curva con quattro punti di controllo ha grado 3 (una cubica).

Questa proprietà è cruciale nella modellazione parametrica, poiché il grado della curva determina la sua flessibilità e la capacità di rappresentare forme più complesse. Curve di grado superiore consentono di ottenere geometrie più articolate, ma al costo di un maggiore sforzo computazionale e di un controllo più difficile. Per questo motivo, nelle applicazioni pratiche, si preferisce spesso lavorare con curve di grado relativamente basso, combinandole opportunamente per ottenere superfici e forme più complesse attraverso tecniche di interpolazione e fusione di curve.



Algoritmo De Casteljan (formula ricorsiva)

$$\bar{b}_0^1(t) = (1-t)\bar{b}_0 + t\bar{b}_1$$

$$\bar{b}_1^1(t) = (1-t)\bar{b}_1 + t\bar{b}_2$$

$$\bar{b}_0^2(t) = (1-t)\bar{b}_0^1(t) + t\bar{b}_1^1(t) =$$

$$= (1-t)^2\bar{b}_0 + 2t(1-t)\bar{b}_1 + t^2\bar{b}_2 \quad \text{(forma quadratica)}$$

L'**algoritmo di De Casteljau** è un metodo ricorsivo utilizzato per la costruzione delle curve di Bézier. Il suo principio fondamentale consiste nel suddividere iterativamente i segmenti definiti dai punti di controllo fino a determinare un punto appartenente alla curva per un dato valore del parametro. Questa costruzione avviene attraverso interpolazioni lineari successive tra i punti di controllo, generando nuovi punti intermedi fino a convergere su un singolo punto della curva.

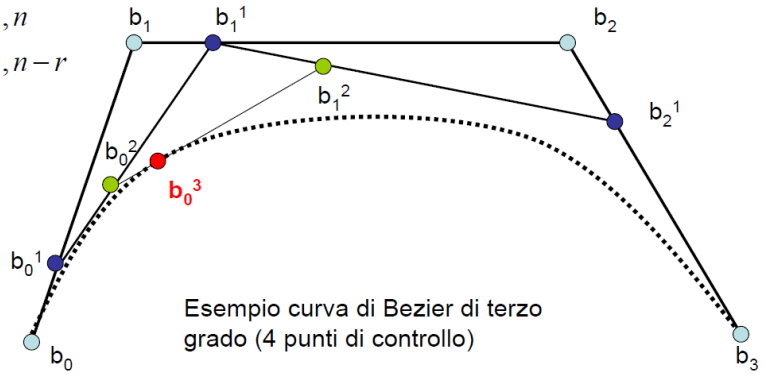
Dal punto di vista computazionale, l'algoritmo presenta un'elevata complessità per curve di **grado elevato**. Se una curva di Bézier è definita da **n punti di controllo**, il numero di passaggi richiesti dall'algoritmo per determinare un punto sulla curva cresce in modo proporzionale al grado della curva. Più precisamente, essendo basato su interpolazioni successive, il numero di operazioni necessarie è dell'ordine di **O(n^2)**, rendendo il metodo poco efficiente per curve di alto grado.

Per questa ragione, l'algoritmo di De Casteljau è particolarmente utile da un punto di vista **grafico**, in quanto consente di visualizzare il processo costruttivo della curva e fornisce un'interpretazione geometrica chiara della sua generazione. Tuttavia, dal punto di vista **analitico e computazionale**, il suo utilizzo risulta meno conveniente, specialmente per curve di ordine elevato, dove metodi più efficienti, come la rappresentazione polinomiale mediante la base di Bernstein o l'approccio con spline, sono preferibili per ridurre il costo computazionale e migliorare la stabilità numerica.

La precedente costruzione di una parabola può essere generalizzata per costruire una curva polinomiale di grado **n**.

$b_0, b_1, b_2, \ldots, b_n$ (grado curva n ⇒ n+1 punti di controllo)

$$\bar{b}_i^r(t) = (1-t)\bar{b}_i^{r-1}(t) \quad 0 \le t \le 1$$
$$\bar{b}_i^0(t) = \bar{b}_i \quad \text{punto di controllo}$$
$$r = 1,\ldots,n$$
$$i = 0,\ldots,n-r$$



3.1.2 Polinomi di Bernstein

Per realizzare un'**implementazione efficiente** dal punto di vista computazionale delle curve di Bézier, è necessario esprimerle mediante una formulazione basata sui **polinomi di Bernstein**. Questi polinomi costituiscono la base della rappresentazione analitica delle curve di Bézier e consentono una gestione più efficiente del calcolo, riducendo la complessità computazionale rispetto all'approccio ricorsivo dell'algoritmo di De Casteljan.

3.1.2.1 Definizione dei polinomi di Bernstein

I polinomi di Bernstein di grado n sono definiti come segue:

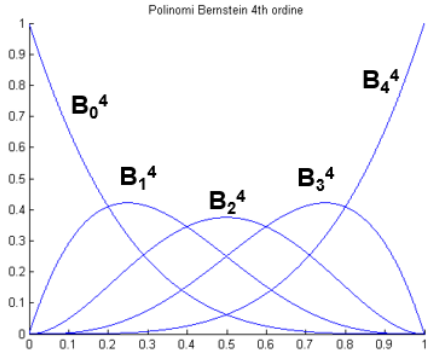
$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad \text{con } i = 0,1,\ldots,n$$

dove:

- $\binom{n}{i}$ è il coefficiente binomiale, definito come

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$$

- t è il parametro della curva, variabile nell'intervallo $[0,1]$;
- $(1-t)$ rappresenta il complemento di t, garantendo che la somma pesata dei termini mantenga le proprietà desiderate di interpolazione.



3.1.3 Applicazione alla curva di Bézier

Le curve di Bézier si basano su funzioni di forma, ovvero **funzioni peso** che determinano il contributo di ciascun punto di controllo nella costruzione della curva. Queste funzioni soddisfano una proprietà fondamentale di **partizionamento dell'unità**, espressa dalla seguente relazione:

$$\sum_{j=0}^{n} B_j^n(t) = 1$$

Questo implica che i **polinomi di Bernstein** agiscono come funzioni di distribuzione del peso, garantendo che la somma delle loro influenze sia sempre pari a uno per ogni valore del parametro t nell'intervallo $[0,1]$.

Proprietà Ricorsiva dei Polinomi di Bernstein

Un'importante proprietà dei polinomi di Bernstein è la loro definizione ricorsiva, che permette di esprimerli in termini di polinomi di grado inferiore:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$$

Questa relazione evidenzia il legame tra i polinomi di Bernstein e l'**algoritmo di De Casteljau**. Infatti, l'algoritmo di De Casteljau costruisce la curva attraverso interpolazioni lineari successive tra coppie di punti di controllo, eseguendo un procedimento iterativo fino a determinare un punto della curva per un dato valore di t.

Poiché la relazione ricorsiva dei polinomi di Bernstein segue lo stesso principio dell'algoritmo di De Casteljau, è possibile descrivere quest'ultimo attraverso le proprietà dei polinomi di Bernstein. Questo porta a una **forma compatta** per la definizione delle curve di Bézier, che può essere espressa come:

$$B^n(t) = \sum_{i=0}^{n} B_i^n(t)\, \bar{b}_i$$

dove:

- $\bar{b}_i$ rappresentano i **punti di controllo**, con $i = 0,1,\dots,n$;
- t è il parametro della curva, compreso nell'intervallo $0 \le t \le 1$.

Questa formulazione permette di calcolare direttamente i punti della curva di Bézier senza necessità di interpolazioni successive, garantendo una maggiore efficienza computazionale rispetto all'approccio ricorsivo di De Casteljau.

Una curva di Bézier di grado n, definita da $n+1$ punti di controllo $b_0, b_1, \dots, b_n$, può essere espressa nella forma parametrica utilizzando i polinomi di Bernstein:

$$P(t) = \sum_{i=0}^{n} B_{i,n}(t) \cdot b_i$$

Questa formulazione permette di calcolare direttamente la posizione di un punto sulla curva senza dover eseguire interpolazioni successive, come avviene nell'algoritmo di De Casteljau.

3.1.4 Vantaggi dell'uso dei polinomi di Bernstein

L'adozione della rappresentazione basata sui polinomi di Bernstein offre numerosi vantaggi:

- **Efficienza computazionale** – La curva può essere calcolata direttamente con una complessità computazionale di $O(n)$, molto inferiore rispetto al metodo ricorsivo di De Casteljau che ha complessità $O(n^2)$.
- **Stabilità numerica** – La formulazione mediante polinomi di Bernstein riduce il rischio di instabilità numerica dovuta a operazioni iterative ripetute.
- **Facilità di implementazione** – La funzione polinomiale è più semplice da valutare e ottimizzare rispetto a una procedura ricorsiva.
- **Adattabilità a trasformazioni geometriche** – La rappresentazione polinomiale permette di integrare la curva in sistemi di rendering grafico e progettazione assistita in modo più diretto.

Grazie a queste proprietà, i polinomi di Bernstein rappresentano la base della modellazione parametrica in **CAD parametrico, grafica computazionale, progettazione aerodinamica e animazione digitale**, consentendo la definizione e la manipolazione efficiente delle curve di Bézier all'interno di ambienti software avanzati.

Ecco un riassunto dettagliato e scientifico delle proprietà delle curve di Bézier, basato sull'immagine che hai fornito:

3.1.5 Proprietà delle Curve di Bézier

Le curve di Bézier sono ampiamente utilizzate nella parametrizzazione geometrica e nella rappresentazione grafica per tre principali proprietà:

1. **Passaggio per il primo punto di controllo**: La curva di Bézier passa sempre per il primo punto di controllo, garantendo un inizio preciso della curva.
2. **Controllo sulla tangenza iniziale e finale**: La tangente iniziale della curva è determinata dal vettore b*0* - *b*1, mentre la tangente finale è data dal vettore b*n* - *b*{n-1}. Questo permette un controllo accurato sulla curvatura iniziale e finale della curva.
3. **Accumulo di punti dove la curvatura è maggiore**: Le curve di Bézier tendono ad accumulare punti nelle zone dove la curvatura è più pronunciata, migliorando la rappresentazione grafica.

Vantaggi nella Rappresentazione Grafica

La tendenza ad accumulare punti nelle zone di maggiore curvatura rende le curve di Bézier particolarmente adatte alla rappresentazione grafica. Utilizzando una parametrizzazione quadratica o cubica, si ottiene una distribuzione dei punti più densa nelle zone di maggiore curvatura, migliorando il controllo sulla forma della curva e l'accuratezza della rappresentazione.

3.2 Parametrizzazione Geometrica di un Profilo Aerodinamico mediante Curve di Bézier

La parametrizzazione geometrica di un profilo aerodinamico rappresenta una fase cruciale nel processo di progettazione, poiché consente di manipolare la forma del profilo in modo da ottimizzare le sue caratteristiche aerodinamiche. In particolare, l'obiettivo principale della parametrizzazione è influenzare i parametri ingegneristici che determinano il comportamento del flusso d'aria, migliorando l'efficienza complessiva del sistema studiato. Un profilo aerodinamico bidimensionale, come nel nostro caso, è descritto da diverse variabili geometriche che impattano direttamente sulle performance del flusso.

3.2.1 Parametri Fondamentali di un Profilo Aerodinamico

Per ottenere una parametrizzazione efficace, è fondamentale prendere in considerazione gli aspetti principali che definiscono un profilo aerodinamico, ovvero:

- **Curvatura del profilo**: Determina la forma globale della superficie aerodinamica e influisce sul comportamento del flusso d'aria che la attraversa. La curvatura gioca un ruolo cruciale nell'ottimizzazione della resistenza e della portanza.
- **Distribuzione dello spessore**: La distribuzione dello spessore lungo il profilo determina la quantità di materiale presente a ogni punto e, di conseguenza, influisce sulla resistenza aerodinamica e sulla stabilità del profilo stesso.
- **Tangenza in ingresso**: La direzione del flusso d'aria all'ingresso del profilo è essenziale per evitare separazioni di flusso che potrebbero compromettere l'efficienza aerodinamica.
- **Tangenza in uscita**: Analogamente, la tangenza in uscita controlla come il flusso si adatta alla fine del profilo e deve essere progettata in modo da garantire una transizione fluida e senza turbolenze.

3.2.2 Strategia di Parametrizzazione del Profilo

Per parametrizzare completamente un profilo aerodinamico, si può suddividere il processo in tre principali aree di intervento:

- **Parametrizzazione della linea media**: Rappresenta la forma centrale del profilo e determina la distribuzione generale della curvatura. La tangenza in ingresso e in uscita della linea media deve essere accuratamente controllata per evitare discontinuità nel flusso.
- **Parametrizzazione della distribuzione dello spessore nell'estradosso**: Lo spessore del profilo nella parte superiore, che influisce sulla velocità del flusso e sulla distribuzione della pressione, deve essere definito con precisione per ottimizzare la portanza e la resistenza.
- **Parametrizzazione della distribuzione dello spessore nell'intradosso**: La parte inferiore del profilo, sebbene simile all'estradosso, richiede una diversa distribuzione di spessore per garantire la stabilità aerodinamica.

3.2.2.1 Parametrizzazione della Linea Media

Nel caso della parametrizzazione della **linea media**, i parametri critici da controllare sono le **tangenze in ingresso** e **in uscita**. Per ottenere un controllo preciso su questi parametri, è possibile utilizzare le **curve di Bézier**, che sono strumenti matematici ideali per rappresentare curve con un numero ridotto di parametri, mantenendo un alto grado di controllo sulla geometria.

Le curve di Bézier di grado 3 sono comunemente utilizzate per la parametrizzazione della linea media. Queste curve sono definite da tre punti di controllo, che determinano la forma complessiva della curva e consentono di controllare le tangenze in ingresso e in uscita. In particolare:

- **Tangenza in ingresso**: La tangenza in ingresso è data dal segmento che unisce il primo e il secondo punto di controllo. Questo segmento determina l'orientamento del flusso d'aria all'inizio del profilo. Nel caso di un profilo aerodinamico, questa tangenza deve essere **normale alla linea media**.

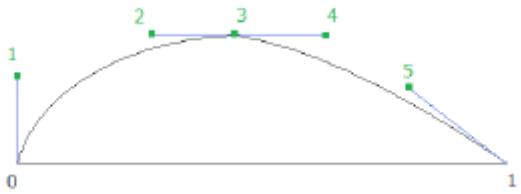
- **Tangenza in uscita**: La tangenza in uscita è determinata dal segmento che collega il penultimo e l'ultimo punto di controllo, definendo la direzione del flusso d'aria alla fine del profilo.

Le variabili di progettazione che definiscono la linea media sono quindi:

- Le **coordinate (x, y)** del secondo punto.
- Le **coordinate (x, y)** del terzo punto.

3.2.2.2 Parametrizzazione dello Spessore nell'Estradosso

La parametrizzazione dello spessore nell'**estradosso** (la parte superiore del profilo) è un altro passaggio fondamentale per definire le caratteristiche aerodinamiche del profilo. Un parametro cruciale è la posizione del massimo dello spessore, che rappresenta il punto in cui si verifica la maggiore espansione del profilo. In questo punto, la tangenza deve essere **orizzontale**.



Nel contesto della parametrizzazione dell'estradosso, i punti di controllo principali sono:

- **Punto 3**: Controlla la quantità di spessore (tipicamente tra il 10% e il 15%) e la posizione del massimo spessore lungo il profilo. Inoltre, definisce la tangenza orizzontale in corrispondenza del massimo spessore.
- **Punto 1**: Controlla la dimensione del "naso" del profilo, influenzando la forma dell'ingresso.
- **Punto 2**: Definisce l'arco della distribuzione dello spessore, determinando quanto sia "curvato" il profilo nell'estradosso.

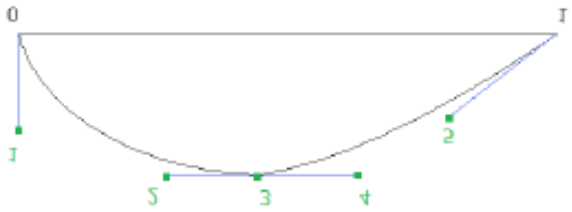
La parte finale della curva dello spessore viene fissata tramite i seguenti punti:

- **Punto 4**: Viene automaticamente determinato in modo tale da evitare discontinuità nel punto di massimo spessore, garantendo che la tangente sinistra sia uguale alla tangente destra.
- **Punto 5**: Controlla il grado di rastremazione della coda del profilo, determinando l'andamento della parte finale del profilo.

Le variabili di progetto per la parametrizzazione dello spessore nell'estradosso sono:

- La **y** del punto 1.
- La **x** del punto 2.
- Le **coordinate (x, y)** del punto 3.
- La **x** del punto 4.
- Le **coordinate (x, y)** del punto 5.

3.2.2.3 Parametrizzazione dello Spessore nell'Intradosso



La parametrizzazione dello spessore nell'**intradosso** segue un processo simile a quello dell'estradosso, ma con il segno delle ordinate invertito, poiché si riferisce alla parte inferiore del profilo. La gestione della geometria dell'intradosso è fondamentale per la stabilità del flusso e per garantire un'efficiente portanza.

Modificando le variabili precedentemente descritte, è possibile ottenere una vasta gamma di geometrie aerodinamiche, inclusi profili simmetrici o con una linea media orizzontale. Questo approccio di parametrizzazione consente un controllo dettagliato su ogni aspetto del profilo, facilitando la progettazione di geometrie ottimizzate per specifiche applicazioni aerodinamiche.

La parametrizzazione di un profilo aerodinamico mediante curve di Bézier offre un metodo flessibile e preciso per progettare geometrie avanzate, in cui ogni aspetto della forma può essere regolato per ottimizzare le performance aerodinamiche. Utilizzando curve di Bézier di grado 3 per la parametrizzazione della linea media e dei profili di spessore, si può controllare con grande dettaglio la curvatura, la distribuzione dello spessore e le transizioni del flusso d'aria, permettendo di esplorare una vasta gamma di soluzioni progettuali e ottimizzare le prestazioni in funzione delle esigenze specifiche del sistema.

Nel caso specifico in tutto ci saranno 18 variabili (19 se introduciamo anche l'angolo di attacco)

3.2.3 Parametrizzazione per la Modifica di Geometrie Esistenti

Il concetto di parametrizzazione geometrica assume un'importanza cruciale in vari contesti ingegneristici, in particolare quando si tratta di progettare e ottimizzare forme aerodinamiche. Una delle decisioni principali che un progettista deve affrontare è se partire da una geometria già esistente e migliorarla, o se esplorare e creare una geometria completamente nuova. La scelta tra questi due approcci ha implicazioni significative per il processo di parametrizzazione e, di conseguenza, per le tecniche di ottimizzazione da adottare.

3.2.3.1 Modifica di una Geometria Esistente: Un Approccio Incrementale

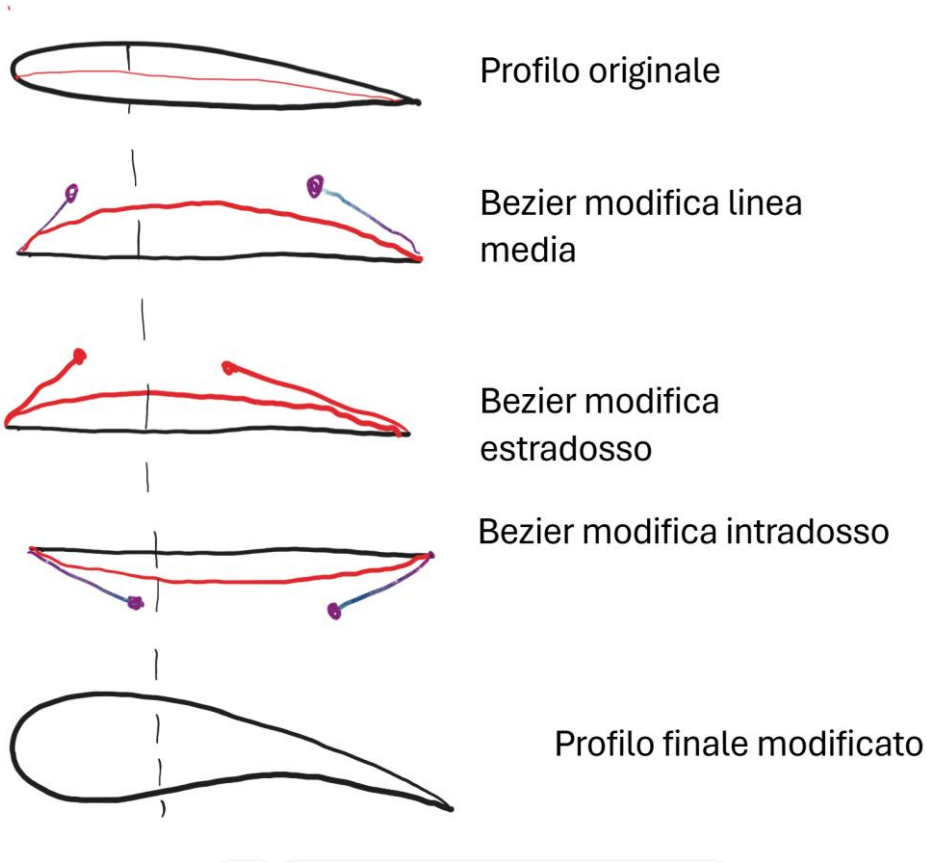
Quando si desidera modificare una geometria già esistente, come nel caso di un **profilo aerodinamico** che ha già mostrato buone performance, l'obiettivo non è partire da zero, ma perfezionare una forma preesistente. Questo approccio di parametrizzazione è intrinsecamente più semplice rispetto alla creazione di una nuova geometria, in quanto si lavora su un punto di partenza già definito e ottimizzato in parte.

In questo scenario, la parametrizzazione può essere concepita come un **aggiustamento fine** della geometria esistente, in cui si interviene su variabili specifiche per ottenere un miglioramento progressivo. Per esempio, la **linea media** di un profilo aerodinamico può essere modificata sommando un **offset** punto per punto alla geometria originaria. Ogni punto della curva originale viene traslato lungo una direzione specifica, in modo da modificarne la forma senza stravolgerla completamente. Questa trasformazione permette di curvare il profilo in maniera controllata e precisa.

La **parametrizzazione lineare**, dove i punti vengono sommati, è un metodo che mantiene la continuità della geometria iniziale, facendo sì che il profilo risultante conservi le sue caratteristiche principali, ma con una

curvatura o distribuzione dello spessore migliorata. Se tutte le variabili di modifica sono fissate a zero, si ottiene il profilo di partenza, garantendo che non vi siano alterazioni indesiderate.

In questo contesto, la parametrizzazione agisce come una **perturbazione controllata** della geometria esistente, permettendo di esplorare variazioni localizzate, ma con il vantaggio di non perdere le qualità fondamentali del profilo originario.



3.2.3.2 Creazione di una Nuova Geometria: Esplorazione Completa del Design

D'altro canto, se l'obiettivo è quello di esplorare una **nuova geometria**, la parametrizzazione prende una direzione completamente diversa. Qui non si parte da una forma predefinita, ma si costruisce una geometria da zero, basata su una serie di parametri indipendenti. Questo approccio è spesso più complesso, poiché implica la definizione di un insieme di variabili e la loro interazione in modo da generare una forma ottimizzata per le condizioni desiderate. L'intento di esplorare nuove geometrie può essere dettato dalla necessità di scoprire soluzioni innovative che superino le limitazioni delle geometrie esistenti, ad esempio aumentando l'efficienza aerodinamica o riducendo la resistenza.

In questo caso, la **parametrizzazione** non è una semplice somma di modifiche incrementali, ma una vera e propria **costruzione parametrica**, dove ogni parametro definisce un aspetto significativo della geometria, come la posizione dei punti di controllo, la curvatura, la distribuzione dello spessore, e altre caratteristiche critiche. Questo approccio richiede una maggiore attenzione nell'analisi dei parametri, poiché anche piccoli cambiamenti in questi ultimi possono comportare modifiche significative nel comportamento aerodinamico del profilo.

3.2.4 Differenze nella Scelta dell'Algoritmo di Ottimizzazione

La scelta di partire da una geometria esistente o di crearne una nuova ha anche un impatto significativo sul tipo di **algoritmo di ottimizzazione** da utilizzare. Quando si modifica una geometria esistente, l'ottimizzazione tende a concentrarsi su **piccole modifiche locali**, che agiscono sulla curvatura, sulla

distribuzione dello spessore e su altri parametri senza stravolgere la geometria di base. Algoritmi di ottimizzazione come il **gradiente discendente** o i **metodi di ricerca locale** sono spesso sufficienti, poiché l'algoritmo esplora modifiche limitate a un intorno della geometria di partenza, ottimizzando la forma in base a una funzione di costo (come la resistenza aerodinamica o la portanza).

Al contrario, quando si crea una nuova geometria da zero, l'ottimizzazione può dover affrontare un **spazio di soluzioni molto ampio e complesso**, richiedendo quindi tecniche di ottimizzazione più sofisticate, come gli **algoritmi genetici**, i **metodi evolutivi** o gli **algoritmi di ricerca globale**. Questi metodi esplorano un ampio spazio di design, cercando di identificare soluzioni ottimali in assenza di una geometria iniziale da cui partire.

3.2.4.1 Parametrizzazione e Flessibilità del Design

L'approccio di **modifica di geometrie esistenti** offre una **maggiore flessibilità** rispetto alla creazione di una nuova geometria, poiché permette di risparmiare tempo e risorse. In molte applicazioni ingegneristiche, i profili aerodinamici già esistenti sono il risultato di anni di ottimizzazione e sperimentazione. Pertanto, partire da una geometria ben consolidata e migliorarla incrementando piccole modifiche consente di ottenere soluzioni ottimali con uno sforzo relativamente ridotto. Inoltre, la modifica di una geometria esistente permette di mantenere le caratteristiche base del profilo, come la stabilità e la resistenza, minimizzando il rischio di soluzioni non praticabili.

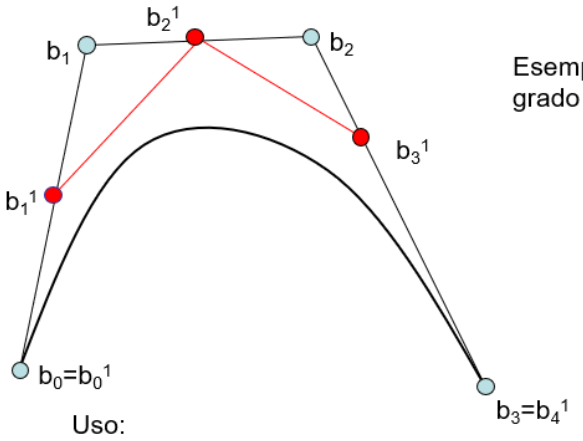
D'altro canto, la **creazione di geometrie nuove** è essenziale quando si vogliono esplorare completamente nuove soluzioni, senza vincoli preesistenti. Questo approccio è particolarmente utile quando si affrontano sfide progettuali complesse, dove le geometrie esistenti potrebbero non essere sufficienti a garantire i miglioramenti desiderati in termini di performance aerodinamica.

In sintesi, la **parametrizzazione che modifica geometrie esistenti** offre vantaggi in termini di semplicità, efficienza e controllo. In particolare, quando l'obiettivo è migliorare un profilo aerodinamico esistente, le modifiche incrementali basate su una parametrizzazione semplice come la somma dei punti rappresentano un approccio diretto e facilmente gestibile. Tuttavia, la scelta di esplorare nuove geometrie comporta una maggiore complessità, richiedendo algoritmi di ottimizzazione avanzati e una parametrizzazione più articolata. Entrambi gli approcci hanno il loro posto nella progettazione ingegneristica, e la decisione su quale percorso seguire dipenderà dalle specifiche esigenze del progetto e dai risultati attesi.

3.3 Degree Elevation: Aumento del grado di una curva di Bézier

È possibile aumentare il grado di una curva di Bézier, ovvero incrementare il numero di punti di controllo, senza perdere in accuratezza. Data una curva di grado (n), è possibile definire la stessa curva di grado (n+1). Il punto iniziale e il punto finale rimangono invariati poiché la curva non subisce modifiche, ma aumenta solo il grado. Inoltre, il primo punto di controllo e l'ultimo punto di controllo della nuova curva si trovano sul primo e sull'ultimo segmento della curva precedente, poiché la tangenza deve rimanere la stessa.

La formula analitica per determinare i nuovi punti di controllo è l'interpolazione lineare tra il punto (i)-esimo e il punto (i-1)-esimo:



Esempio: passaggio da curva di grado 3 a curva di grado 4

$$B_i^{n+1} = (1 - \frac{i}{n+1} B_i^n) + \frac{i}{n+1} B_{i-1}^n$$

Aumentare il grado della curva significa aumentare il numero di variabili. Questo è un aspetto positivo, poiché, pur avendo meno variabili possibili, non si può perdere in accuratezza. Con la curva di Bézier è molto semplice incrementare il numero di variabili, poiché si può raggiungere una geometria ottimale con la parametrizzazione e poi aumentare le variabili (i punti) senza perdere in accuratezza, ottenendo così un controllo migliore.

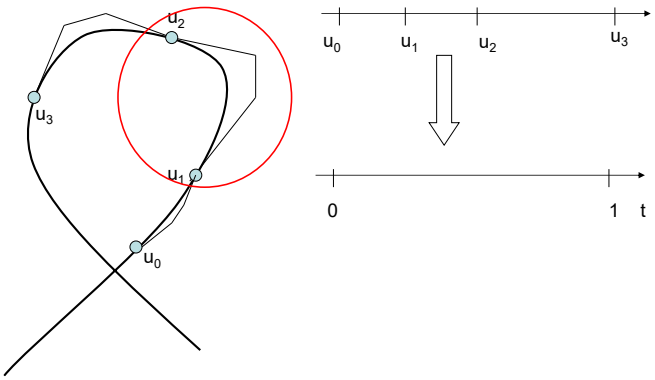
3.3.1 Esempio pratico

Consideriamo il profilo di un timone di una barca a vela da competizione. Si tratta di un profilo simmetrico. Attraverso l'ottimizzazione si ottengono risultati con miglioramenti del 5% quando ci si aspettavano miglioramenti del 15%. Il problema è che questo profilo è un profilo super laminare, ovvero la parte laminare è molto importante. Pertanto, per migliorare questo profilo, è fondamentale la parte iniziale. Una curva di grado 3 è troppo limitata per riuscire a gestire le micro curvature. Aumentando il grado a 4, non si perde nulla dal punto di vista geometrico, poiché il passaggio è analitico, ma si aumentano le variabili. Con una variabile in più, l'algoritmo è in grado di effettuare quelle micro modifiche che migliorano le prestazioni.

3.4 Curve di Bézier Tradizionali e i Loro Limiti

Le curve di Bézier tradizionali sono curve polinomiali che dipendono da un insieme di punti di controllo. Per una curva di Bézier di grado **n**, si utilizzano **n+1** punti di controllo. La forma della curva è fortemente influenzata dai punti di controllo, e un singolo punto di controllo può alterare significativamente la geometria della curva. Se si aumenta il grado della curva, si aumenta anche il numero di punti di controllo, ma questo rende difficile mantenere un controllo locale su ogni punto della geometria. Se il grado diventa troppo elevato (ad esempio, n=8 con 9 punti di controllo), la modifica di un punto di controllo può cambiare drasticamente tutta la curva, rendendo complicata la manipolazione della geometria.

Inoltre, con curve di alto grado, diventa difficile ottenere un buon compromesso tra flessibilità e controllo. Aggiungendo più punti di controllo, la curva può acquisire maggiore complessità, ma perde in termini di prevedibilità e controllo diretto.



3.5 B-spline: Parametrizzazione con Controllo Locale

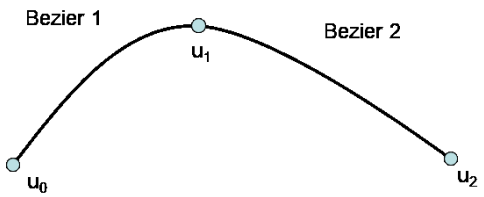
Le **B-spline** (Basis Splines) risolvono questo problema introducendo una rappresentazione più flessibile che conserva il controllo locale anche in geometrie complesse. Una B-spline è una combinazione di più curve di Bézier, ognuna delle quali ha il proprio dominio di definizione, e queste curve sono collegate in modo continuo attraverso una serie di nodi, definiti come $U_0, U_1, \ldots, U_n$. La connessione tra le curve di Bézier in una B-spline è tale che le proprietà di continuità della curva sono definite dai nodi, e la geometria complessiva è influenzata da modifiche locali.

3.5.1 Nodi e Continuità

Un aspetto fondamentale delle B-spline è il modo in cui le curve sono collegate attraverso i **nodi**. I nodi sono valori di parametri che definiscono dove una curva di Bézier termina e un'altra inizia. La continuità tra le curve è determinata dal grado di continuità specificato nei nodi. Ci sono vari livelli di continuità:

- **C0 (continuità di posizione):** La curva è continua nella posizione, ma non necessariamente nella direzione o nella curvatura.
- **C1 (continuità di tangente):** La curva è continua sia nella posizione che nella direzione (tangente), ma la curvatura potrebbe cambiare bruscamente.
- **C2 (continuità di curvatura):** La curva è continua nella posizione, direzione e curvatura, senza cambiamenti improvvisi in nessuna di queste proprietà.

Ogni curva di Bézier in una B-spline può essere progettata in modo da garantire una continuità adeguata, che è essenziale per ottenere una transizione liscia tra le segmentazioni della curva. La scelta del grado di continuità dipende dal tipo di applicazione: per esempio, per una superficie liscia, potrebbe essere necessaria la continuità C2.



3.5.2 Vantaggi delle B-spline

1. **Controllo Locale:** Modificare un singolo punto di controllo influenzerà solo la parte della curva che corrisponde a quel segmento, mantenendo il controllo locale. Questo è particolarmente utile per

modellare geometrie complesse, in quanto consente di manipolare singoli segmenti senza compromettere l'intera forma.

2. **Flessibilità:** Le B-spline possono rappresentare una vasta gamma di forme, anche molto complesse, con una quantità limitata di punti di controllo.

3. **Modularità:** Una B-spline è costruita da una serie di curve Bézier collegate, il che consente di adattare la forma della curva facilmente mediante l'aggiunta o la modifica dei nodi e dei punti di controllo, senza dover ricalcolare l'intera curva.

4. **Continuità Personalizzata:** È possibile scegliere il grado di continuità desiderato tra i segmenti della B-spline, ottenendo una geometria che soddisfi le esigenze specifiche del progetto.

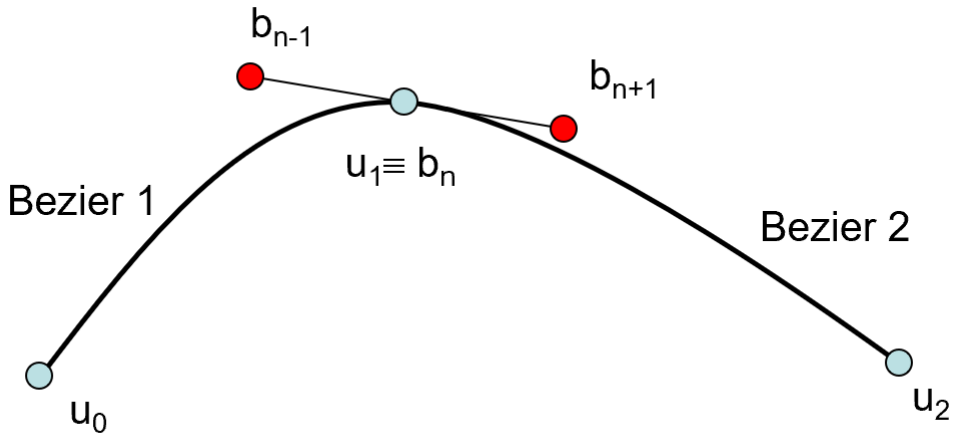
3.6 Parametrizzazione delle Curve B-spline

La parametrizzazione delle curve B-spline è definita dal **dominio dei nodi**, che suddivide l'intervallo in cui la curva è definita. Ogni nodo in una B-spline è associato a un intervallo di parametri e ciascun segmento di curva Bézier è valido in un intervallo di parametri. La posizione di ogni punto lungo la curva è determinata dalla combinazione lineare dei punti di controllo e dalle funzioni di base, che sono definite dai nodi. Le funzioni di base di una B-spline sono scelte in modo che ogni punto di controllo influenzi solo una porzione della curva, garantendo così il controllo locale.

Le B-spline offrono una soluzione potente per rappresentare curve e superfici complesse, superando i limiti delle curve di Bézier tradizionali. La possibilità di controllare localmente la geometria, mantenendo allo stesso tempo una buona continuità tra i segmenti, le rende particolarmente utili in ambito ingegneristico e nelle applicazioni di modellazione 3D. La gestione dei nodi e il grado di continuità permettono di ottenere geometrie lisce e controllabili, fondamentali per il design avanzato e la simulazione.

3.6.1 Continuità C^1 nelle B-Spline

La continuità C^1 in una curva spline indica che la funzione che definisce la curva è continua e che la sua derivata prima è anch'essa continua in ogni punto della curva, inclusi i nodi di giunzione tra segmenti di curva.



3.6.2 Condizione di continuità C^1 nel nodo U_i

Nel nodo U_i, affinché la curva sia C^1-continua, la derivata sinistra deve essere uguale alla derivata destra, ovvero:

$$\lim_{t \to U_i^-} \frac{dC(t)}{dt} = \lim_{t \to U_i^+} \frac{dC(t)}{dt}$$

dove $C(t)$ è la funzione che descrive la curva.

Questo implica che i vettori tangenti alle due sezioni di curva che si incontrano in U_i devono essere collineari. Consideriamo due curve di Bézier consecutive $C_1(t)$ e $C_2(t)$, definite rispettivamente dai punti di controllo $\{b_{n-2}, b_{n-1}, b_n\}$ e $\{b_n, b_{n+1}, b_{n+2}\}$.

Affinché la continuità C^1 sia garantita, è necessario che il segmento finale della prima curva abbia la stessa direzione del segmento iniziale della seconda curva, ovvero:

b_{n-1}, b_n, b_{n+1} sono allineati.

Questo si spiega perché, nelle curve di Bézier, le tangenti ai punti estremi della curva sono controllate direttamente dai punti di controllo adiacenti. Quindi, per garantire che la derivata prima sia continua in U_i, è necessario che il segmento $b_n - b_{n-1}$ sia parallelo a $b_{n+1} - b_n$.

3.6.3 Dipendenza tra i punti di controllo

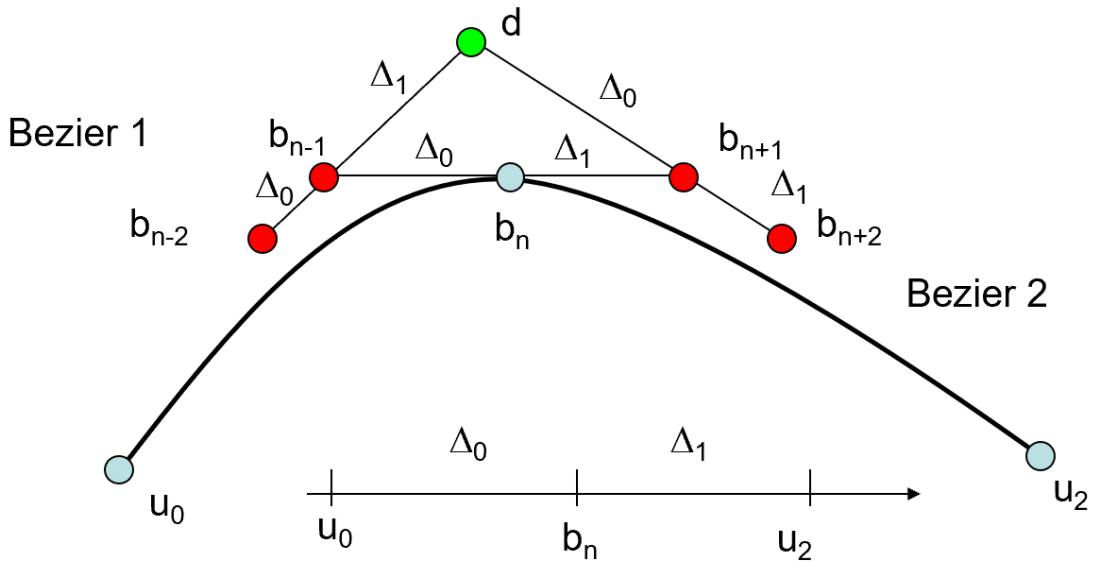
Nelle B-Spline di grado 3 (cubic B-Spline), la continuità C^1 implica che, se si modifica b_{n-1}, è necessario aggiornare b_{n+1} in modo da mantenere l'allineamento con b_n. Questo perché i segmenti tangenti in b_n sono determinati dai punti di controllo adiacenti.

In altre parole, b_{n+1} non può essere scelto arbitrariamente ma deve essere vincolato dalla posizione di b_{n-1} affinché la derivata prima rimanga continua. Questo vincolo assicura che la curva non presenti discontinuità angolari e che la transizione tra i segmenti sia fluida.

La condizione di continuità C^1 per una B-Spline impone che tre punti di controllo consecutivi siano allineati nei nodi di giunzione. Questo significa che la modifica di un punto di controllo precedente (b_{n-1}) deve essere compensata da un adeguato spostamento di quello successivo (b_{n+1}) per garantire una transizione liscia tra i segmenti della spline.

3.6.4 Continuità C^2 nelle B-Spline

La continuità C^2 in una curva spline implica che sia la funzione che la definisce sia la sua derivata prima e seconda siano continue in ogni punto della curva, inclusi i nodi di giunzione tra segmenti di curva.



3.6.4.1 Condizione di continuità C^2 nel nodo U_i

Per garantire la continuità C^2 in un nodo U_i, devono essere soddisfatte le seguenti condizioni:

Continuità C^0 (continuità della curva stessa):

$$\lim_{t \to U_i^-} C(t) = \lim_{t \to U_i^+} C(t)$$

Ovvero, la curva è continua senza discontinuità o buchi.

Continuità C^1 (continuità della derivata prima, ovvero della tangente):

$$\lim_{t \to U_i^-} \frac{dC(t)}{dt} = \lim_{t \to U_i^+} \frac{dC(t)}{dt}$$

Questo significa che i segmenti b_{n-1}, b_n, b_{n+1} devono essere allineati affinché la tangente sia continua.

Continuità C^2 (continuità della derivata seconda, ovvero della curvatura):

$$\lim_{t \to U_i^-} \frac{d^2C(t)}{dt^2} = \lim_{t \to U_i^+} \frac{d^2C(t)}{dt^2}$$

Questo implica che la variazione della tangente sia uniforme, quindi anche i punti b_{n-2} e b_{n+2} devono soddisfare una specifica relazione geometrica.

3.6.4.2 Interpretazione Geometrica della Continuità C^2

Nel caso di una curva B-Spline di grado $k = 3$ (cioè una cubic B-Spline), la continuità C^2 implica che:

- **Allineamento dei punti di controllo per la derivata prima continua:**
  Affinché la tangente sia continua, i punti b_{n-1}, b_n, b_{n+1} devono essere allineati.
- **Vincolo sui punti di controllo per la derivata seconda continua:**
  Affinché la curvatura sia continua, anche b_{n-2} e b_{n+2} devono soddisfare una determinata relazione geometrica. In particolare, la variazione della tangente tra segmenti consecutivi deve essere costante. Questo significa che la distribuzione dei punti di controllo segue un andamento regolare, evitando brusche variazioni nella curvatura della curva risultante.

Controllo Locale della Curva

Una proprietà importante delle B-Spline C^2 è che garantiscono un **controllo locale**, il che significa che la modifica di un singolo punto di controllo influenza la curva solo in un intervallo limitato. Questo è dovuto al fatto che il contributo di ciascun punto di controllo è confinato all'interno di un intervallo determinato dalla funzione base della B-Spline.

In sintesi, la continuità C^2 nelle B-Spline impone vincoli più stringenti rispetto alla continuità C^1, richiedendo che i punti di controllo siano distribuiti in modo da garantire non solo la continuità della tangente ma anche quella della curvatura, ottenendo così una transizione ancora più fluida tra i segmenti della curva.

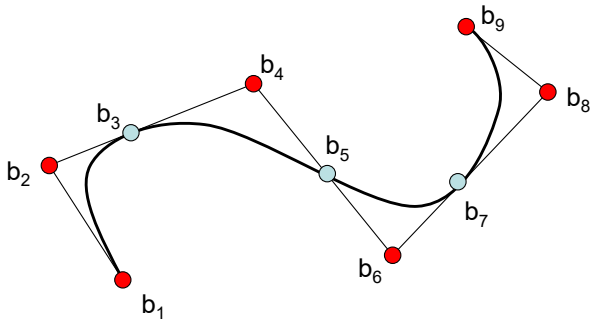
3.6.5 B-SPLINE QUADRATICHE C1

Le B-Spline possono essere:

- quadratiche : i vari segmenti di Bezier devono essere di ordine 2

- cubiche : i vari segmenti di Bezier devono essere di ordine 3.

Le B-Spline più utilizzate sono quelle quadratiche di grado C1.



Quando si parla di curve di Bézier **quadratiche**, ci si riferisce a curve il cui ordine è 2. Questo significa che ogni segmento della curva è descritto da una curva di Bézier di grado 2, ovvero definita da **tre punti di controllo**.

3.6.6 Proprietà della continuità C^1

Affinché la curva sia C^1**-continua**, ovvero abbia continuità di prima derivata, i segmenti devono connettersi in modo che la direzione della tangente sia coerente tra di loro nei punti di giunzione. Per ottenere questa proprietà nel caso di curve di Bézier quadratiche spezzate, si impone la condizione che:

- Ogni punto di giunzione sia il punto medio tra il punto di controllo precedente e il successivo.

Consideriamo i punti dati:

- I segmenti sono definiti da tre punti ciascuno:
  - b_2, b_3, b_4
  - b_4, b_5, b_6
  - b_6, b_7, b_8
- Per garantire la continuità C^1, i punti di connessione b_3, b_5, b_7 devono rispettare la condizione:

$$b_3 = \frac{b_2 + b_4}{2}, \quad b_5 = \frac{b_4 + b_6}{2}, \quad b_7 = \frac{b_6 + b_8}{2}$$

Questo significa che i punti di giunzione b_3, b_5, b_7 **non sono liberi** ma dipendono dai punti di controllo precedenti e successivi.

3.6.6.1 Grado di libertà e controllo locale

Il sistema ha come **variabili libere** i punti di controllo:

$$b_1, b_2, b_4, b_6, b_8, b_9$$

Poiché b_3, b_5, b_7 sono determinati in base ai punti adiacenti, il numero effettivo di gradi di libertà si riduce. Questo comporta che:

- La modifica di un punto di controllo **influenza solo localmente** la curva, perché i punti di connessione mantengono la continuità senza coinvolgere l'intera struttura.
- Non tutti i punti della curva si muovono simultaneamente quando si modifica un singolo punto di controllo, il che permette un maggiore **controllo locale** della forma della curva.

Questa costruzione di curve di Bézier quadratiche spezzate con continuità C^1 è molto utilizzata in grafica e modellazione geometrica, perché consente di ottenere transizioni fluide tra segmenti senza richiedere un controllo globale sull'intera curva.

3.7 Curve NURBS: curve parametriche

Una curva di Bézier è definita come:

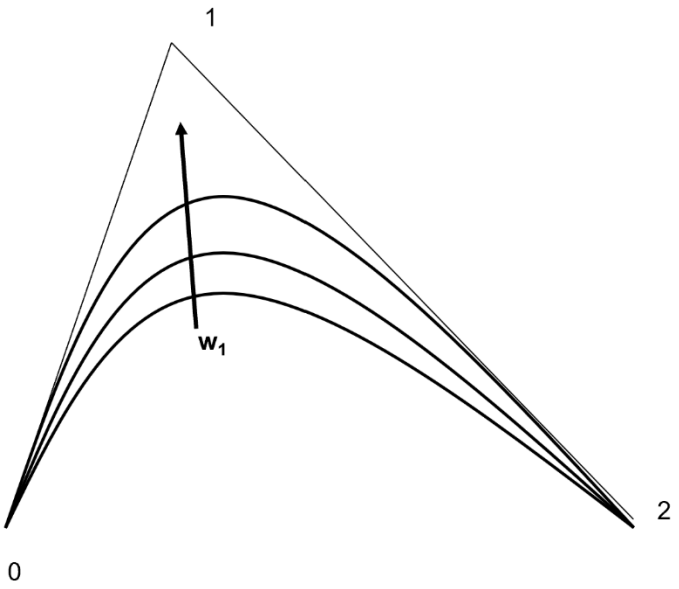
$$\mathbf{P}(u) = \sum_{i=0}^{n} \mathbf{P}_i \, B_i^n(u)$$

dove B_i^n sono i polinomi di Bernstein e $\mathbf{P}_i$ sono i punti di controllo.

Le curve NURBS estendono le curve di Bézier introducendo dei pesi w_i per ogni punto di controllo. La formula per una curva NURBS è:

$$\mathbf{T}(u) = \frac{\sum_{i=0}^{n} w_i \, \mathbf{P}_i B_i^n(u)}{\sum_{i=0}^{n} w_i \, B_i^n(u)}$$

In questa formula, i pesi w_i influenzano l'importanza relativa dei punti di controllo. Un punto con un peso maggiore avrà un'influenza maggiore sulla forma della curva.



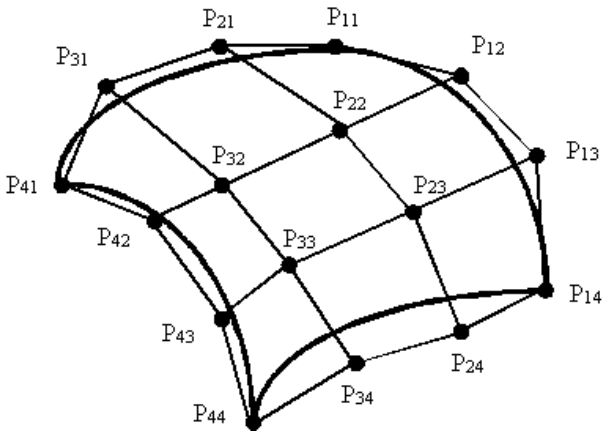
Consideriamo una curva di Bézier di grado 3 con punti di controllo $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ e pesi associati w_0, w_1, w_2, w_3. Se uno dei pesi, ad esempio w_2, è molto maggiore degli altri, la curva si avvicinerà di più al punto $\mathbf{P}_2$.

Nelle applicazioni CAD (Catia), i pesi possono essere modificati per controllare la "tensione" della curva, permettendo una modellazione più precisa e flessibile delle forme.

3.8 Superfici di Bézier: Estensione delle Curve di Bézier in 2D

Le curve di Bézier sono ampiamente utilizzate nella modellazione geometrica e nella computer grafica per rappresentare curve parametriche lisce mediante un insieme di punti di controllo. Le **superfici di Bézier** rappresentano la naturale estensione bidimensionale di questo concetto, consentendo di modellare superfici nello spazio tridimensionale.

Mentre una curva di Bézier è una funzione che dipende da un singolo parametro t, una superficie di Bézier dipende da **due parametri** u e v, che controllano rispettivamente la direzione orizzontale e verticale della superficie.



Definizione Matematica

Se una curva di Bézier di grado n è definita da:

$$\mathbf{x}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{b}_i$$

dove $B_i^n(t)$ sono le **basi di Bernstein** e $\mathbf{b}_i$ sono i **punti di controllo**, la superficie di Bézier diventa:

$$\mathbf{x}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)\mathbf{b}_{i,j}$$

dove:

- $B_i^n(u)$ e $B_j^m(v)$ sono i **polinomi di Bernstein** nelle direzioni u e v,
- $\mathbf{b}_{i,j}$ sono i **punti di controllo**, disposti in una **griglia bidimensionale**.

Polinomi di Bernstein in 2D

I polinomi di Bernstein sono definiti come:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Nel caso delle superfici, questi polinomi vengono moltiplicati nei due parametri u e v, permettendo di interpolare la superficie in entrambe le direzioni.

Punti di Controllo e Matrice Associata

I punti di controllo di una superficie di Bézier sono organizzati in una **matrice** di dimensioni $(n+1) \times (m+1)$, corrispondente ai gradi della superficie in ciascuna direzione.

Ad esempio, per una superficie quadratica ($n = 2, m = 2$), la matrice dei punti di controllo è:

$$\begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,1} & \mathbf{b}_{0,2} \\ \mathbf{b}_{1,0} & \mathbf{b}_{1,1} & \mathbf{b}_{1,2} \\ \mathbf{b}_{2,0} & \mathbf{b}_{2,1} & \mathbf{b}_{2,2} \end{bmatrix}$$

La superficie risultante sarà una combinazione pesata di questi punti di controllo.

3.8.1 Proprietà delle Superfici di Bézier

Interpolazione dei Vertici

- La superficie passa esattamente per i **quattro vertici estremi** della griglia di punti di controllo (per superfici di Bézier rettangolari).

Effetto Locale

- Modificare un punto di controllo influisce **solo su una porzione locale** della superficie, rendendo l'editing intuitivo.

Continuità C^0, C^1, C^2

- La continuità geometrica tra più superfici di Bézier può essere regolata scegliendo opportunamente i punti di controllo:
  - C^0: Le superfici si toccano nei bordi.
  - C^1: La transizione è liscia, con continuità della prima derivata.
  - C^2: Anche la curvatura è continua.

Elevazione del Grado (Degree Elevation)

- È possibile aumentare il grado della superficie senza modificarne la forma, semplicemente ridistribuendo i punti di controllo in modo appropriato.

Riduzione del Grado (Degree Reduction)

- Si possono approssimare superfici di Bézier di grado alto con superfici di grado inferiore, riducendo la complessità computazionale.

Superfici Composte (Patch Joining)

- Superfici più complesse possono essere costruite affiancando più superfici di Bézier con continuità appropriata.

Le superfici di Bézier sono uno strumento potente per la modellazione geometrica e trovano applicazione in **grafica computerizzata, CAD, animazione e modellazione industriale**. Sebbene la maggior parte dei

software CAD si occupi automaticamente della gestione delle superfici, comprendere la matematica sottostante è essenziale per ottimizzare il controllo e la qualità della modellazione.

4 Design Of Experiments (DOE) e Analisi Statistica Dei Dati

Il **Design of Experiments (DOE)** e l'**analisi statistica dei dati** sono metodologie fondamentali per la progettazione ingegneristica e l'ottimizzazione di sistemi complessi. Il loro obiettivo principale è **identificare e quantificare l'influenza delle variabili di controllo su un processo** in modo efficiente, riducendo il numero di esperimenti necessari e massimizzando l'informazione acquisita.

In ambito ingegneristico e scientifico, il DOE è utilizzato per **pianificare gli esperimenti** in maniera sistematica, evitando approcci empirici basati su tentativi ed errori. L'analisi statistica dei dati consente poi di **estrarre informazioni significative** per la successiva ottimizzazione.

4.1 Le Tre Domande Fondamentali del DOE

Il **Design of Experiments** (DoE), grazie alla pianificazione degli esperimenti (che nella metodologia CAO sono ovviamente di tipo numerico), ha l'obiettivo di comprendere nel modo più efficace possibile il funzionamento del sistema in esame.

Per raggiungere questo scopo è necessario innanzitutto riuscire a rispondere a tre aspetti fondamentali:

- individuare le variabili più importanti del sistema;
- determinarne il campo di variazione;
- definire, se possibile, le relazioni principali tra variabili e prestazioni.

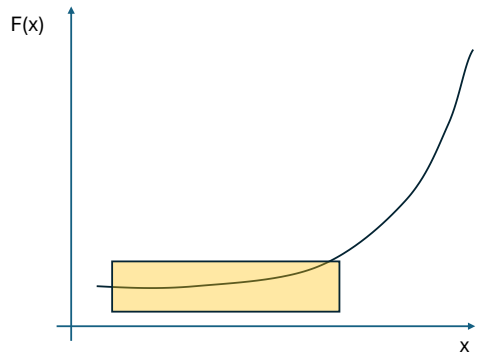
4.1.1 Determinazione delle variabili significative

Nel contesto dell'ottimizzazione, la scelta delle variabili di controllo è essenziale. Dato un sistema con più parametri x1,x2,…,xn, si vuole identificare quali sono le variabili che influenzano significativamente la funzione obiettivo.

Ridurre il numero di variabili consente di semplificare il problema, ridurre i costi computazionali e migliorare l'efficacia dell'ottimizzazione.

4.1.2 Determinazione del campo di variazione delle variabili

Stabilire intervalli troppo ampi per le variabili di input può portare a simulazioni inutili o poco significative. Al contrario, un range troppo ristretto potrebbe escludere configurazioni ottimali.

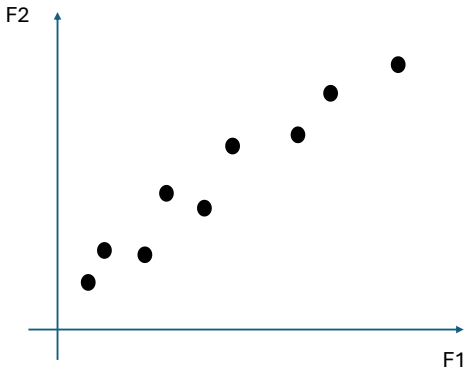


Esempio: se una funzione di costo cresce rapidamente oltre un certo valore di x1, non ha senso analizzare regioni dove il valore è chiaramente inferiore.

4.1.3 Determinazione delle relazioni tra variabili e obiettivi

L'ottimizzazione multi obiettivo è una caratteristica comune dei problemi ingegneristici, dove si cerca di massimizzare o minimizzare più funzioni contemporaneamente (es. minimizzare il costo e massimizzare la resistenza di un materiale).

Se due funzioni obiettivo sono **fortemente correlate** (es. f1 e f2), potrebbe non essere necessario ottimizzarle separatamente: migliorando una, migliora automaticamente anche l'altra.



Design of Experiments (DOE)
Il *Design of Experiments* è un insieme di metodologie numeriche che hanno lo scopo di pianificare in maniera ottimale una campagna sperimentale o computazionale. L'obiettivo principale è individuare, in modo sistematico e rigoroso, quali configurazioni o combinazioni di parametri devono essere analizzate per ottenere il massimo delle informazioni possibili con il minimo sforzo computazionale o sperimentale.

Queste metodologie non si riducono a una singola tecnica, ma comprendono un insieme variegato di approcci che possono essere scelti e adattati a seconda della natura del problema da affrontare. In pratica, il DOE guida nella selezione delle condizioni sperimentali più significative, riducendo il numero totale di

prove necessarie, eliminando configurazioni inutili o ridondanti e garantendo al contempo un'elevata efficienza informativa.

I principali vantaggi di un approccio basato sul DOE sono:

- **Riduzione del numero di configurazioni da testare**: attraverso un'attenta selezione, si limita il numero di simulazioni o esperimenti da eseguire, ottimizzando così tempi e risorse;
- **Eliminazione delle configurazioni ridondanti**: si evitano prove inutili che non apportano nuove informazioni rilevanti;
- **Massimizzazione delle informazioni**: ogni configurazione scelta contribuisce in modo significativo all'analisi, permettendo di estrarre il massimo contenuto informativo dall'intero processo sperimentale.

Analisi statistica dei dati

Una volta raccolti i dati attraverso le configurazioni pianificate, entra in gioco l'analisi statistica. Questa fase è fondamentale perché permette di elaborare i risultati ottenuti, identificare relazioni tra variabili, valutare la significatività dei fattori in gioco e, soprattutto, rispondere in modo quantitativo e oggettivo alle principali domande che si pongono in fase progettuale o di ottimizzazione.

In sintesi, il DOE, abbinato a una corretta analisi statistica, rappresenta uno strumento potentissimo per condurre esperimenti in modo strutturato, efficiente e scientificamente fondato, specialmente in contesti in cui le risorse o il tempo sono limitati.

Normalmente il DOE precede la fase di progettazione che prevede:

1. Parametrizzazione
2. DOE
3. Analisi statistica dei dati
4. Ottimizzazione vera e propria.

In realtà, le diverse fasi del processo non sono sempre rigidamente sequenziali, ma possono subire modifiche ed essere iterate in base ai risultati ottenuti. Ad esempio, può accadere che, una volta completata un'ottimizzazione e analizzati i risultati tramite il *post-processing*, emerga la necessità di migliorare ulteriormente alcuni aspetti. In tal caso, si torna a una nuova fase di *Design of Experiments* (DOE) per ridefinire le configurazioni da analizzare, avviando così un nuovo ciclo di ottimizzazione. In questo scenario, il DOE può trovarsi sia prima che dopo l'ottimizzazione, svolgendo un ruolo centrale nel processo iterativo.

Un altro possibile flusso di lavoro prevede una sequenza ancora più articolata: si parte con la parametrizzazione del problema, si esegue il DOE, si analizzano statisticamente i dati ottenuti, quindi si procede con una prima fase di ottimizzazione. Se necessario, si ritorna al DOE per affinare ulteriormente la selezione delle configurazioni. A questo punto, possono essere impiegate le superfici di risposta — metodi numerici che forniscono un'approssimazione della funzione obiettivo — per modellare il comportamento del sistema in modo più efficiente. Infine, si esegue una nuova ottimizzazione basata su queste superfici, rendendo l'intero processo ciclico e progressivamente più preciso.

4.2 METODOLOGIE DOE: RANDOM E SOBOL

I metodi *DOE Random* e *Sobol* sono tra i primi approcci utilizzati nell'ambito del *Design of Experiments*. Entrambi si basano sulla generazione casuale di numeri, e di conseguenza sulla generazione casuale di variabili e configurazioni. Questo consente di esplorare lo spazio delle variabili in modo stocastico.

Metodo Random

Il metodo *Random* si fonda sulla generazione di numeri casuali compresi tra 0 e 1, mediante algoritmi che sfruttano l'orologio interno del calcolatore. Questi algoritmi restituiscono un numero casuale R tale che:

$$0 \leq R \leq 1$$

Supponiamo di voler valutare una funzione obiettivo $f(x_1, x_2)$, dove:

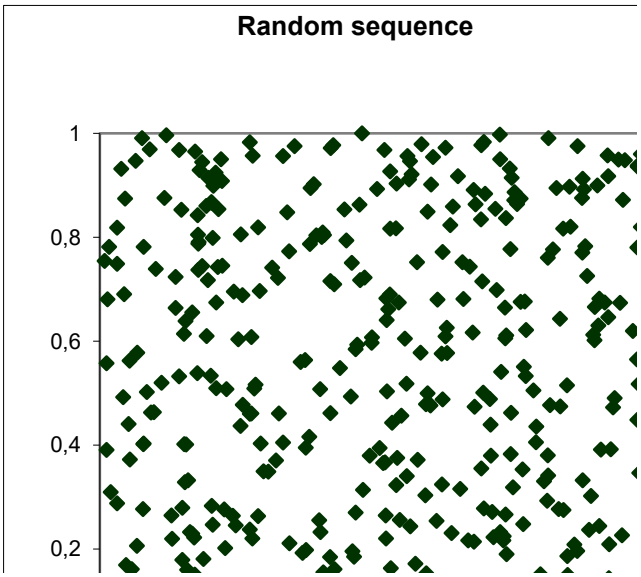
$$x_1 \in [x_1^{\min}, x_1^{\max}], \quad x_2 \in [x_2^{\min}, x_2^{\max}]$$

Poiché R varia tra 0 e 1, possiamo generare configurazioni casuali $x_1^{(i)}$ e $x_2^{(i)}$ come segue:

$$x_1^{(i)} = x_1^{\min} + R \cdot \left(x_1^{\max} - x_1^{\min}\right)$$

$$x_2^{(i)} = x_2^{\min} + R \cdot \left(x_2^{\max} - x_2^{\min}\right)$$

In questo modo, otteniamo punti casuali $X^{(i)} = \left(x_1^{(i)}, x_2^{(i)}\right)$ che riempiono lo spazio delle variabili. Ogni configurazione rappresenta un punto in cui valutare la funzione obiettivo. Ripetendo il processo per un numero sufficiente di volte, è possibile esplorare l'intero dominio di interesse in maniera casuale.



4.2.1 Metodo Sobol

Il metodo *Sobol* appartiene alla categoria dei metodi *quasi-random* (o *low-discrepancy sequences*) utilizzati nel *Design of Experiments* per la generazione di punti nello spazio delle variabili in modo sistematico e uniforme. A differenza del metodo *Random*, che si basa su una generazione puramente casuale e non controllata, il metodo *Sobol* genera una sequenza di numeri che, pur mantenendo una base casuale, segue una logica deterministica finalizzata a coprire lo spazio di definizione delle variabili in maniera ottimale.

In particolare, la sequenza di Sobol è costruita per minimizzare la **discrepanza**, cioè la misura della non uniformità della distribuzione dei punti all'interno dello spazio. Questo significa che i punti generati dalla sequenza tendono a riempire in modo più omogeneo il dominio delle variabili rispetto ai punti casuali ottenuti da un generatore random classico.

L'intuizione alla base è quella di campionare progressivamente il dominio delle variabili coprendo inizialmente le **direzioni principali** (come le diagonali dello spazio multidimensionale), per poi andare a colmare sistematicamente gli **spazi vuoti residui**. Questo approccio assicura che già con un numero relativamente basso di punti si ottenga una buona copertura del dominio, riducendo la possibilità di zone scarsamente campionate.

In termini pratici, per ogni punto della sequenza si ottiene un vettore di dimensione pari al numero di variabili del problema, in cui ciascun componente assume un valore compreso tra 0 e 1. Tali valori vengono poi scalati rispetto ai limiti inferiori e superiori di ciascuna variabile:

$$x_j^{(i)} = x_j^{\min} + S_j^{(i)} \cdot \left(x_j^{\max} - x_j^{\min}\right) \quad \text{per } j = 1, \dots, n$$

dove:

- $x_j^{(i)}$ è il valore della j-esima variabile nella i-esima configurazione,
- $S_j^{(i)}$ è l'elemento della sequenza di Sobol per la j-esima variabile al passo i,
- $x_j^{\min}$, $x_j^{\max}$ sono i limiti del dominio della variabile x_j.

Grazie alla sua struttura ordinata ma non prevedibile, la sequenza di Sobol è particolarmente efficace nei problemi di ottimizzazione e simulazione ad alta dimensione, dove è essenziale una copertura omogenea dello spazio senza dover aumentare eccessivamente il numero di esperimenti.

La distinzione tra i metodi *Random* e *Sobol* ha implicazioni pratiche importanti sull'uso dell'uno o dell'altro in funzione del numero di variabili che definiscono il problema.

In generale:

- **Metodo Random** → più adatto per **funzioni con molte variabili**
- **Metodo Sobol** → più efficace per **funzioni con poche variabili**

Questo perché, nella pratica, il numero di configurazioni da calcolare è solitamente **fissato in anticipo**, sulla base dei tempi di calcolo disponibili o delle risorse computazionali. Quando si utilizza il metodo **Sobol**, le prime configurazioni generate tendono a coprire in maniera ordinata le **direzioni principali** del dominio (ad esempio, le diagonali), cercando di esplorare sistematicamente lo spazio.

Tuttavia, in uno spazio ad alta dimensione, le diagonali principali sono numerose, e per riuscire a coprirle in modo significativo è necessario generare un grande numero di configurazioni. Se il numero massimo di punti calcolabili è limitato, può succedere che tutte le configurazioni si distribuiscano solo lungo queste diagonali. In tal caso, le informazioni ottenute rischiano di non essere **sufficientemente indipendenti**, rendendo l'analisi meno efficace.

Al contrario, il metodo **Random**, generando punti in modo casuale e non strutturato, tende a distribuire le configurazioni in maniera più dispersa all'interno dell'intero dominio, anche in presenza di molte variabili. Questo riduce la probabilità che due configurazioni siano **linearmente dipendenti** e migliora la varietà delle informazioni ottenute.

D’altra parte, quando il problema coinvolge **poche variabili**, il metodo **Sobol** diventa molto vantaggioso. In questi casi, è in grado di coprire il dominio in modo molto più uniforme rispetto a un approccio casuale, garantendo una maggiore efficienza informativa anche con un numero limitato di configurazioni.

Per comprendere meglio la differenza tra i metodi *Random* e *Sobol*, consideriamo un esempio pratico: la risoluzione di un integrale tridimensionale (cioè su tre variabili) tramite l’algoritmo di **Monte Carlo**. Per semplicità, iniziamo considerando un caso bidimensionale, ovvero il calcolo dell'integrale:

$$\int \int f(x_1, x_2)\, dx_1 dx_2 \approx \sum_{i=1}^{n} f\,(x_i)$$

Per calcolare i punti x_i in cui valutare la funzione, possiamo utilizzare algoritmi di generazione casuale come **Random** o **Sobol**. In entrambi i casi, l'obiettivo è stimare il valore dell'integrale con un certo errore ε, definito come:

$$\varepsilon = | \hat{I} - I |$$

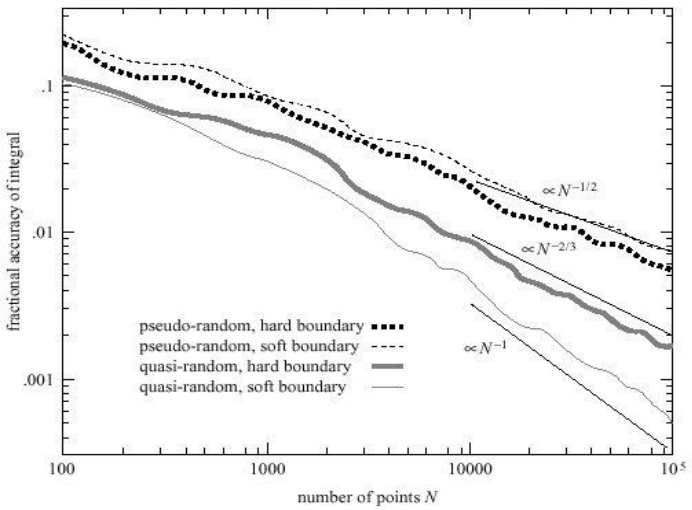
dove:

- $\hat{I}$ è il valore dell’integrale calcolato numericamente;
- I è il valore esatto (analitico) dell’integrale.

L’andamento dell’errore in funzione del numero di configurazioni calcolate (n) mostra una chiara differenza tra i due approcci:

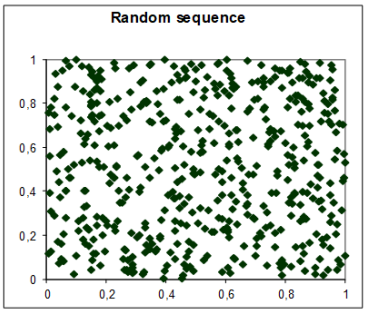
- Con il metodo **Sobol**, l’errore decresce più rapidamente all’aumentare di n;
- Con il metodo **Random**, l’errore decresce più lentamente, a causa della distribuzione meno uniforme dei punti.

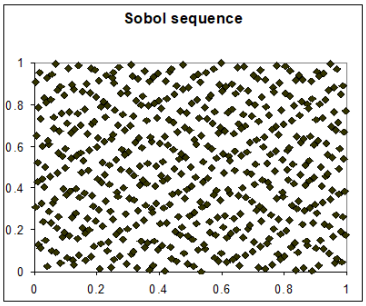
Questo comportamento è illustrato nel grafico che mostra la curva dell’errore decrescente rispetto al numero di punti campionati: la curva di **Sobol** si abbassa più velocemente rispetto a quella di **Random**, indicando una maggiore efficienza nella copertura del dominio.



Nelle immagini a confronto:

- La distribuzione dei punti generata con **Random** appare più irregolare, con zone densamente popolate e altre meno esplorate.
- La sequenza **Sobol**, invece, distribuisce i punti in modo più uniforme su tutto il dominio, evitando accumuli e spazi vuoti.



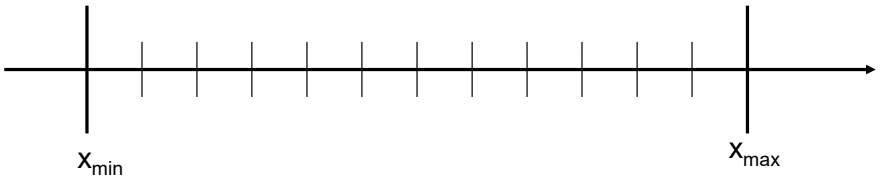


Questa maggiore uniformità nella copertura del dominio rende l'algoritmo Sobol particolarmente efficace per problemi a bassa dimensionalità. Tuttavia, quando il numero di variabili diventa elevato, la logica deterministica del metodo può generare configurazioni **linearmente dipendenti**, soprattutto se il numero di punti calcolabili è limitato. In tali casi, il metodo **Random** può risultare preferibile, poiché, pur essendo meno strutturato, mantiene una maggiore diversità tra le configurazioni generate.

4.3 DOE: Full Factorial e Reduced Factorial

4.3.1 DEFINIZIONE DI BASE

Abbiamo una variabile i che normalmente, dal punto di vista analitico, è una variabile continua. Da un punto di vista dell'implementazione non ho mai variabili continue ma le divido discretamente:

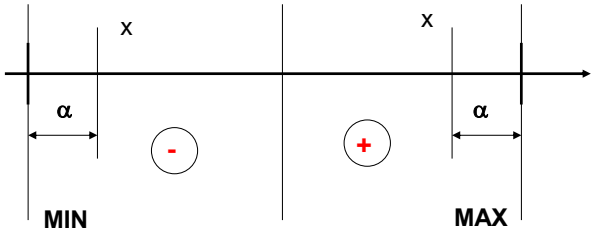


Il numero m (di divisioni della mia variabile) è chiamata base. Per esempio una variabile analogica può variare tra 0° e 100°. In tal caso se divido a metà posso realizzare 1,001° ma posso passare solo da 1° a 1,5° ... Divido la mia base della metà della tolleranza.

Per il DOE dividiamo la variabile a metà:

E definiamo come:

- X_i: il valore mediano della parte inferiore
- X_i': il valore mediano della parte superiore



Possiamo introdurre anche un parametro a, che determina quanto in profondità vogliamo posizionare il punto all'interno dell'intervallo. Questo parametro può essere interpretato così:

- Se $a = 0$, il punto si trova esattamente sul bordo inferiore dell'intervallo.
- Se $a = 1$, il punto si sposta verso il valore mediano della variabile. Tuttavia, non raggiunge mai esattamente la mediana, altrimenti coinciderebbe con il punto "+" (che si assume si trovi al centro del campo). Di norma, quindi, si ferma intorno al 75% dell'intervallo di variazione.

4.3.2 FULL FACTORIAL

Il metodo **Full Factorial** consiste nel considerare tutte le possibili configurazioni tra tutte le basi delle variabili. Questo significa che, se abbiamo n variabili e ciascuna variabile può assumere m valori distinti, il numero totale di configurazioni sarà m^n. Matematicamente, possiamo rappresentarlo come:

$$f: R^n \quad n^{conf} = m^n$$

4.3.2.1 Applicabilità Ingegneristica

Dal punto di vista ingegneristico, il metodo Full Factorial è spesso impraticabile. Questo perché il numero di configurazioni necessarie per esplorare tutte le combinazioni possibili può diventare estremamente elevato, rendendo il processo di analisi e sperimentazione molto costoso e dispendioso in termini di tempo.

4.3.2.2 Strategie di Riduzione

Per ovviare a questo problema, si adottano due principali strategie:

- **Riduzione della Base**: Si cerca di ridurre il numero di valori che ciascuna variabile può assumere. Questo diminuisce il numero totale di configurazioni possibili.
- **Metodi Fattoriali Ridotti**: Si passa da un metodo fattoriale completo a un metodo fattoriale ridotto. Questo implica selezionare un sottoinsieme delle configurazioni totali che rappresentano in modo efficace l'intero spazio delle variabili. Alcuni esempi di metodi fattoriali ridotti includono il **Reduced Factorial Design**.

4.3.2.3 Ridurre la Base

Quando si utilizza un metodo **Full Factorial**, una strategia comune per semplificare il numero di configurazioni è ridurre la base delle variabili. Questo significa limitare il numero di livelli di variazione che ciascuna variabile può assumere. Normalmente, si utilizza una base $m = 2$, il che implica che ogni variabile può assumere solo due livelli distinti: uno negativo (-) e uno positivo (+).

4.3.2.4 Esempio di Riduzione della Base

Consideriamo un esempio pratico con 3 variabili ($n = 3$) e una base $m = 2$. In questo caso, ogni variabile può assumere due livelli:

- Parte inferiore (valore -)
- Parte superiore (valore +)

Quindi, il numero totale di configurazioni da calcolare sarà:

$$f: \mathbb{R}^3 \quad m = 2 \quad 8 \text{ configurazioni da calcolare}$$

Matematicamente, possiamo rappresentarlo come:

$$n^{conf} = m^n = 2^3 = 8$$

4.3.2.5 Vantaggi della Riduzione della Base

1. **Semplificazione**: Ridurre la base semplifica il processo di sperimentazione, diminuendo il numero di configurazioni da analizzare.
2. **Efficienza**: Con meno configurazioni, è possibile eseguire esperimenti in modo più rapido ed economico.
3. **Focalizzazione**: Concentrarsi su due livelli distinti permette di identificare rapidamente gli effetti principali delle variabili.

4.3.2.6 Tabella DOE Full Factorial 3 variabili base 2

	x_1	x_2	x_3
1	+	+	+
2	+	+	-
3	+	-	+
4	+	-	-
5	-	+	+
6	-	+	-
7	-	-	+
8	-	-	-

4.3.3 Reduced Factorial

Il metodo **Reduced Factorial** è una tecnica utilizzata per semplificare il numero di configurazioni necessarie in un esperimento fattoriale completo. Invece di considerare tutte le variabili, si concentra solo sulle variabili che l'ingegnere reputa più importanti. Le altre variabili vengono trattate mediante combinazioni lineari delle variabili principali.

4.3.3.1 *Esempio Pratico*

Consideriamo una funzione definita su 4 variabili:

$$f: x^4$$

Le variabili sono x_1, x_2, x_3, x_4, e ciascuna variabile può assumere due livelli ($m = 2$), cioè $-$ e $+$.

Se volessimo fare un **Full Factorial**:

$$n_f^{conf} = 2^4 = 16$$

Questo numero di configurazioni potrebbe essere troppo elevato rispetto al tempo di calcolo disponibile. Per questo motivo, si passa al **Reduced Factorial**.

4.3.3.2 *Riduzione delle Variabili*

Supponiamo che le variabili più importanti per il nostro sistema siano 3 ($P = 3$). Quindi, le variabili principali sono x_1, x_2, x_3.

Il numero di configurazioni nel **Reduced Factorial** sarà:

$$n_r^{conf} = 2^P = 8$$

La variabile x_4 viene trattata come una combinazione lineare delle variabili principali.

4.3.3.3 *Tabella delle Configurazioni*

Ecco un esempio di tabella delle configurazioni per il **Reduced Factorial**:

n. design	x1	x2	x3	x4 (=x1*x2)
1	+	+	+	+
2	+	+	-	+
3	+	-	+	-
4	+	-	-	-
5	-	+	+	-
6	-	+	-	-
7	-	-	+	+
8	-	-	-	+

4.3.4 Vantaggi del Reduced Factorial

- **Efficienza**: Riduce significativamente il numero di configurazioni da calcolare, rendendo il processo più rapido ed economico.
- **Focalizzazione**: Permette di concentrarsi sulle variabili che hanno un impatto maggiore sul sistema, migliorando la qualità dell'analisi.
- **Semplificazione**: Facilita la gestione e l'interpretazione dei dati sperimentali.

Decido che le variabili x_4 sia espressa come combinazione lineare delle variabili precedenti. Il vantaggio di questa metodologia è che consente una riduzione intelligente del numero di variabili, mantenendo comunque un approccio *full factorial* sulle variabili più significative.

Alla fine, ottengo una tabella DOE (Design of Experiments) ordinata, in cui ogni variabile presenta lo stesso numero di valori sia nella metà superiore che in quella inferiore del proprio intervallo. Questo è un aspetto molto utile per l'analisi statistica successiva, in quanto garantisce una buona distribuzione dei dati.

4.3.4.1 Limiti del metodo

Il principale svantaggio è che non è possibile esplorare tutte le configurazioni possibili tra tutte le variabili, quindi si perde parte dell'informazione. In altre parole, il metodo è completo (come un full factorial) solo per le variabili considerate più importanti; per le altre, la copertura dello spazio di variazione non è esaustiva.

Un ulteriore limite riguarda la scelta del numero di variabili principali p: non posso ridurlo liberamente. Ad esempio, supponiamo di avere 6 variabili:

$$x_1, x_2, x_3, x_4, x_5, x_6$$

e di scegliere $p = 3$, ovvero di eseguire un full factorial sulle prime tre variabili.

In questo caso:

- x_4 sarà una combinazione lineare di x_1 e x_2
- x_5 sarà una combinazione di x_2 e x_3
- x_6 sarà una combinazione di x_1 e x_3

In questo modo, le configurazioni di x_4, x_5, x_6 risultano non correlate tra loro.

Supponiamo ora di voler aggiungere una settima variabile x_7, mantenendo $p = 3$. Dovrei definire anche x_7 come combinazione lineare delle prime tre variabili, ma a questo punto le variabili derivate diventerebbero tra loro correlate, causando una perdita eccessiva di informazione. La soluzione sarebbe aumentare il numero di variabili principali a:

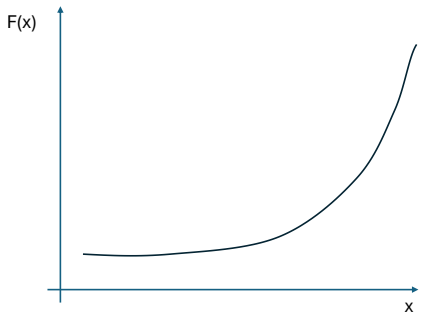
$$p = 4$$

Il problema, quindi, sta nell'equilibrio tra la riduzione della dimensionalità e la qualità dell'informazione ottenuta.

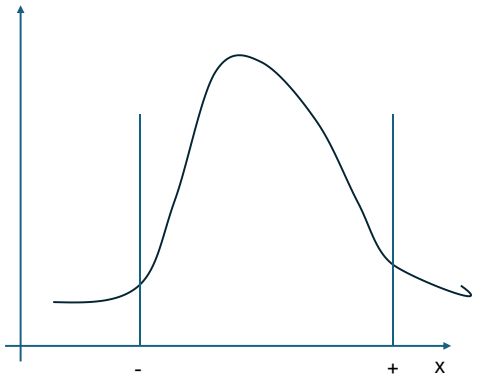
4.3.5 Problema degli algoritmi fattoriali

Il principale limite degli algoritmi fattoriali è la necessità di mantenere una base ridotta, tipicamente al massimo $m = 3$. Tuttavia, in ambito ingegneristico, questa restrizione spesso non è accettabile, soprattutto quando ci troviamo di fronte a comportamenti fortemente non lineari.

Lavorare con una base $m = 2$ può andare bene solo in casi in cui la funzione associata a una variabile X_i sia monotona. Ad esempio, se la funzione $f(x)$ mostra un andamento regolare e prevedibile, anche con soli due livelli (ad esempio basso e alto), è comunque possibile coglierne le tendenze principali.



Tuttavia, se la funzione $f(x)$ ha un andamento complesso o presenta comportamenti non monotoni, una base $m = 2$ non è sufficiente a catturarne le variazioni significative. In questi casi, il risultato dell'esperimento risulterebbe poco rappresentativo e non affidabile.



In ingegneria, questo è un problema concreto: limitare eccessivamente la base può compromettere l'efficacia dell'analisi. Per ottenere una rappresentazione accurata del sistema, è spesso necessario aumentare il numero di livelli per ciascuna variabile.

Tuttavia, c'è un problema pratico: se manteniamo una base elevata per tutte le variabili e applichiamo un metodo fattoriale classico, il numero totale di configurazioni cresce esponenzialmente, rendendo il problema computazionalmente insostenibile.

In sintesi:

- Una base bassa (es. $m = 2$) non è sufficiente per sistemi con alta non linearità.
- Una base alta è spesso necessaria in ingegneria, ma rende inapplicabili i metodi fattoriali completi a causa della crescita combinatoria delle configurazioni.

4.3.6 DOE Latin Square

Nel contesto della progettazione sperimentale, quando si lavora con **metodi factoriali completi**, uno dei principali problemi è la **crescita esponenziale del numero di configurazioni sperimentali** al crescere della base m, ovvero del numero di livelli (o modalità) che ogni fattore può assumere.

Nel caso di un **full factorial design**, il numero totale di prove richieste è:

$$N = m^k$$

dove:

- m è la base (cioè il numero di livelli per ciascun fattore),
- k è il numero di fattori (variabili indipendenti) coinvolti.

Questa crescita esponenziale rende rapidamente **impraticabile** l'esecuzione di tutti gli esperimenti, soprattutto in applicazioni ingegneristiche o industriali dove $m > 2$ o k è elevato.

Per superare questa difficoltà, è stata sviluppata la metodologia del **Latin Square Design**, che consente di ridurre drasticamente il numero di esperimenti **senza compromettere la validità statistica dell'analisi** dei risultati. È particolarmente utile quando si vogliono studiare gli effetti di un **fattore principale** (ad esempio

un trattamento), ma si sa che due **fattori di disturbo (blocking factors)** possono influenzare la risposta del sistema.

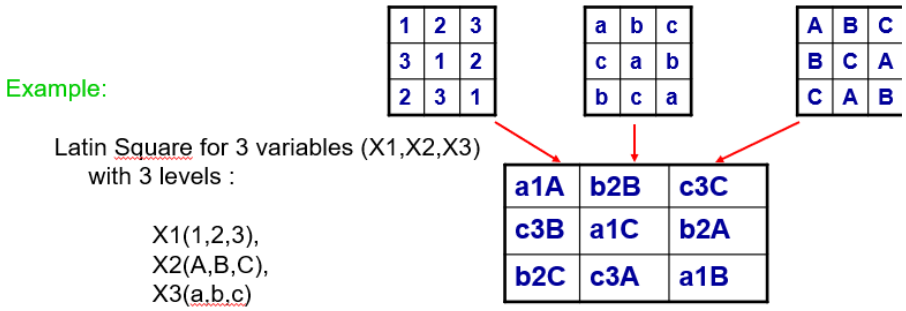
Nel **Latin Square Design**, il numero totale di configurazioni richieste è:

$$N = m^2$$

dove:

- m è il numero di livelli del trattamento (e anche dei due blocchi),
- i blocchi sono rappresentati dalle righe e dalle colonne della matrice.

Questo approccio è estremamente più efficiente rispetto al factorial completo, soprattutto quando m è grande.



Supponiamo di avere:

- 4 trattamenti da testare (es. 4 materiali diversi),
- 2 fonti di disturbo (es. operatore e turno di lavoro),
- e 4 livelli per ciascuno.

Con un **Latin Square** abbiamo bisogno di soli:

$$4^2 = 16 \text{ esperimenti}$$

In confronto, un full factorial con 3 fattori a 4 livelli richiederebbe:

$$4^3 = 64 \text{ esperimenti}$$

Quindi il Latin Square riduce il numero di prove del **75%**, mantenendo comunque il controllo su due fonti di variabilità.

- Il Latin Square **non permette di stimare interazioni** tra i fattori.
- È necessario che tutti e tre i fattori (trattamento + 2 blocchi) abbiano lo **stesso numero di livelli**.
- È più adatto per situazioni in cui **l'effetto dei blocchi è sistematico e rilevante**, e si vuole isolarlo dall'analisi del trattamento principale.

Il Latin Square rappresenta un **compromesso ottimale** tra **riduzione della complessità sperimentale** e **robustezza dell'analisi statistica**, rendendolo una scelta efficace in ambiti dove le risorse (tempo, costi, materiali) sono limitate, ma si richiede comunque un disegno ben bilanciato.

4.4 Analisi statistica dei dati post-DOE

Una volta completata la valutazione delle configurazioni sperimentali previste dal piano **DOE (Design of Experiments)**, la fase successiva consiste in un'**analisi statistica strutturata dei dati** raccolti. Questo processo è fondamentale per estrarre informazioni utili sul comportamento del sistema sotto indagine e per supportare decisioni ingegneristiche o progettuali più consapevoli.

L'obiettivo di questa fase è duplice:

1. **Comprendere le relazioni tra le variabili di input e la risposta del sistema**.
2. **Ottimizzare l'utilizzo delle risorse, riducendo la complessità del modello** senza compromettere l'accuratezza delle previsioni.

4.4.1 Obiettivi principali dell'analisi

L'analisi statistica post-DOE permette di rispondere a una serie di domande chiave:

1. Quali sono le variabili più influenti?

Attraverso l'applicazione dell' analisi statistica è possibile identificare i fattori che esercitano l'influenza maggiore sulla variabile di risposta. Questo consente di concentrare l'attenzione sui parametri realmente determinanti, trascurando quelli trascurabili.

2. È possibile ridurre la dimensionalità dello spazio delle variabili?

Si può valutare se alcune variabili possono essere eliminate o combinate linearmente senza perdita significativa di informazione. Questo passaggio è cruciale per semplificare il modello e ridurre i tempi computazionali nelle fasi successive.

3. Qual è la regione più efficiente dello spazio delle variabili?

L'analisi dei dati sperimentali consente di:

- individuare **regioni ottimali** all'interno dello spazio di definizione,
- delimitare aree in cui il sistema soddisfa in modo più efficiente i requisiti di progetto,
- costruire **superfici di risposta (Response Surface Models, RSM)** utili per l'ottimizzazione multi-obiettivo.

4. Il numero di obiettivi e vincoli è adeguato?

L'analisi permette inoltre di verificare se:

- il numero di **funzioni obiettivo** è coerente con la complessità del problema,
- esistono **vincoli ridondanti o inconsistenti**,
- sono necessarie **riformulazioni del problema di ottimizzazione**, ad esempio introducendo obiettivi ponderati, vincoli soft o funzioni di penalizzazione.

L'analisi statistica dei dati DOE non rappresenta un semplice passaggio accessorio, ma costituisce una **fase centrale del processo di modellazione e ottimizzazione**, in quanto permette di:

- quantificare l'incertezza e la variabilità del sistema,
- orientare le scelte progettuali verso soluzioni più robuste ed efficienti,
- facilitare l'eventuale integrazione con modelli predittivi o algoritmi di ottimizzazione.

In sintesi, un'analisi rigorosa consente di trasformare i dati grezzi in **conoscenza utile per la progettazione avanzata**, secondo un approccio ingegneristico data-driven.

4.4.2 Analisi Statistica Semplificata

L'analisi statistica condotta può essere considerata semplificata in quanto si limita a valutare esclusivamente la differenza media dei valori assunti dalla funzione all'interno di un intervallo definito, generalmente centrato attorno a un punto di interesse (ad esempio, un valore atteso o osservato), e simmetrico rispetto a tale punto (cioè ±Δ). In questo contesto, non si tiene conto di altri indicatori statistici quali la varianza, la deviazione standard, la distribuzione dei dati o eventuali correlazioni. Tale approccio, sebbene utile per ottenere una valutazione preliminare dell'andamento medio della funzione, non fornisce informazioni complete sulla variabilità o sulla significatività statistica delle osservazioni.

Diff=Tot+- Tot-
Effect=Diff / 4

	A	B	C	D	OBJ
1	-	-	-	-	65,6
2	-	-	+	+	79,3
3	-	+	-	+	51,3
4	-	+	+	-	69,6
5	+	-	-	+	59,8
6	+	-	+	-	77,7
7	+	+	-	-	74,2
8	+	+	+	+	87,9

	A	B	C	D
Tot +	300	283	315	278
Tot -	266	282	251	287
Diff	34	0,6	64	-9
Effect	8,5	0,2	16	-2

AB
307
258,4
48,6
12,5

AC
231,2
282,9
-51,7
-12,925

AD
282,9
282,5
0,4
0,1

Interpretazione scientifica semplificata dell'analisi di sensitività tramite differenze medie e implicazioni per l'ottimizzazione:

L'analisi basata sulla differenza media dei valori della funzione obiettivo, calcolata facendo variare ciascuna variabile indipendente all'interno di un intervallo predefinito (±Δ), evidenzia in modo chiaro l'importanza relativa delle variabili. In particolare, la variabile **X_3** risulta essere quella con l'impatto maggiore sulla funzione: la differenza tra il valore della funzione quando X_3 assume il suo valore minimo e quello massimo all'interno del range considerato è significativamente superiore rispetto alle altre variabili. Questo comportamento suggerisce che **X_3 è la variabile più influente** nel sistema.

Alla luce di questa informazione, si può semplificare la futura fase di **ottimizzazione** limitandola a un sottogruppo di variabili, ad esempio **X_1 e X_3**, escludendo quelle con impatto trascurabile. Questa riduzione del numero di variabili in gioco consente **una diminuzione del tempo computazionale necessario** e una maggiore efficienza nella ricerca del massimo della funzione.

Inoltre, si osserva che per alcune variabili, come X_1 e X_3, i valori più elevati all'interno del range inizialmente definito sono associati a valori più alti della funzione obiettivo. Di conseguenza, ha senso **rivedere i limiti dell'intervallo di esplorazione**: il valore inferiore **I** può essere aggiornato a un nuovo limite **I'**, più prossimo alla zona in cui la funzione cresce, mentre il valore superiore può eventualmente essere **esteso**. Questo tipo di modifica permette di **concentrare la ricerca nelle regioni dello spazio delle variabili più promettenti**, riducendo lo spreco di risorse su regioni subottimali.

In sintesi, mediante un **Design of Experiments (DOE)** semplice e l'analisi della **media dei valori della funzione** nei punti ±Δ, si ottiene un'indicazione qualitativa ma efficace sull'**andamento globale della funzione**. Ciò consente, già in fase preliminare, di prendere decisioni informate per ottimizzare il sistema in modo mirato e con maggiore efficienza.

4.4.3 Parametro di t-student

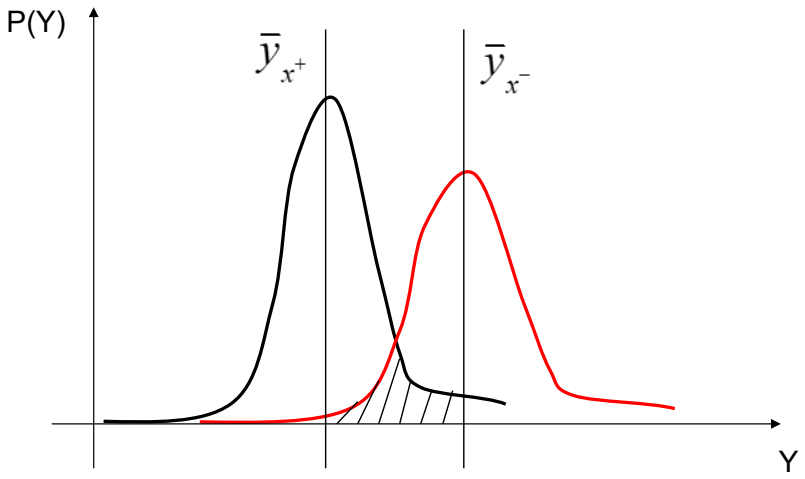
Viene definita **Analisi Statistica Semplificata** quella metodologia in cui si valutano le variazioni della funzione obiettivo considerando esclusivamente la **differenza tra i valori medi** assunti dalla funzione stessa quando una variabile indipendente, indicata genericamente come x_i, viene fatta variare tra due livelli (ad esempio, un livello basso e uno alto). In pratica, per ogni variabile x_i, si calcola la media dei valori della funzione ottenuti quando x_i è a livello basso e la media quando x_i è a livello alto, mantenendo costanti o variando in modo controllato le altre variabili secondo uno schema DOE (Design of Experiments). La **differenza tra queste due medie** viene quindi interpretata come indicatore dell'influenza di quella variabile sul comportamento del sistema.

Tuttavia, questa metodologia **non tiene conto della distribuzione statistica dei dati** ottenuti per ciascun livello della variabile, e in particolare **ignora informazioni fondamentali come la varianza o la forma della distribuzione** (ad esempio, se è simmetrica, stretta, larga o presenta outlier). Si assume implicitamente che una maggiore distanza tra le medie implichi una maggiore importanza della variabile, ma ciò può portare a valutazioni fuorvianti.

Ad esempio, prendendo in esame la variabile x_i, se la differenza tra i valori medi della funzione è elevata, si potrebbe concludere che questa variabile ha un impatto significativo sul sistema. Tuttavia, senza considerare la **distribuzione della funzione** nei due gruppi (ad esempio con un'analisi della varianza o un test t di Student), non si ha alcuna misura della **significatività statistica** di tale differenza. Una distribuzione molto ampia o rumorosa potrebbe infatti rendere tale differenza non statisticamente rilevante.

Per questo motivo, la semplice osservazione delle medie, pur utile in fase esplorativa, può risultare **limitativa**: non è in grado di distinguere tra una differenza reale (cioè sistematica) e una differenza dovuta al rumore o alla variabilità interna del sistema. Per superare questo limite, si ricorre in genere a

metodologie più robuste, come l'**analisi della varianza (ANOVA)** o il **test t di Student**, che consentono di confrontare le medie tenendo conto della **variabilità interna ai gruppi**.



L'approccio basato sull'**Analisi Statistica Semplificata** presenta alcune limitazioni strutturali, in quanto si fonda esclusivamente sulla differenza tra i **valori medi** della funzione obiettivo, trascurando completamente la **distribuzione statistica** dei dati associati a ciascun livello delle variabili. Questo può condurre a interpretazioni errate sull'importanza relativa delle variabili in un modello multivariato.

Supponiamo di confrontare due variabili indipendenti, x_i e x_j. Secondo l'analisi semplificata, la variabile x_i risulterebbe più rilevante semplicemente perché la differenza tra le medie della funzione valutata a x_i^- e x_i^+ è maggiore rispetto a quella osservata per x_j. Tuttavia, **questa valutazione ignora completamente la variabilità interna** delle osservazioni. Potrebbe infatti accadere che la funzione valutata in corrispondenza di x_j presenti una **distribuzione molto stretta** e ben definita (ad esempio una distribuzione gaussiana con deviazione standard ridotta). In questo caso, pur osservando una differenza media più contenuta, il passaggio da x_j^- a x_j^+ comporterebbe un **cambiamento sistematico e statisticamente significativo**, ovvero un vero e proprio "gradino" nella funzione, altamente prevedibile e affidabile.

Di conseguenza, per avere una misura più completa dell'effetto di una variabile, è necessario **integrare la valutazione delle medie con quella delle varianze**. Questo si ottiene attraverso l'utilizzo del **parametro t di Student**, definito come:

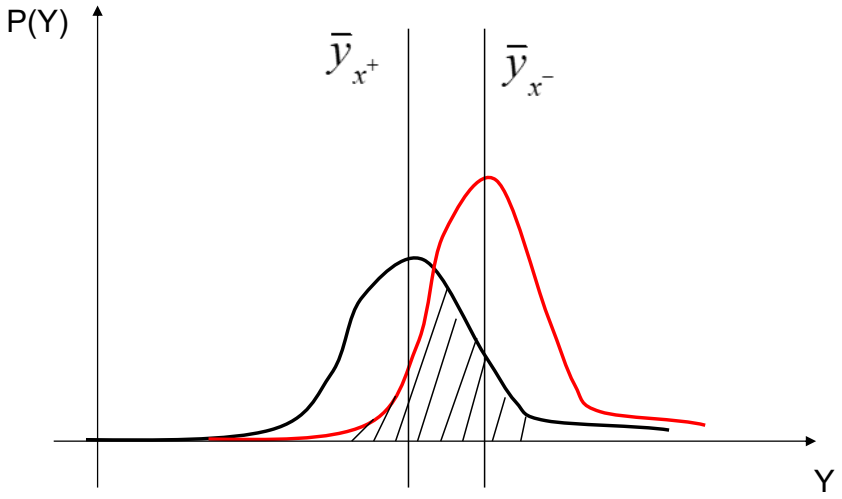
$$t = \frac{\bar{y}^+ - \bar{y}^-}{s_x}$$

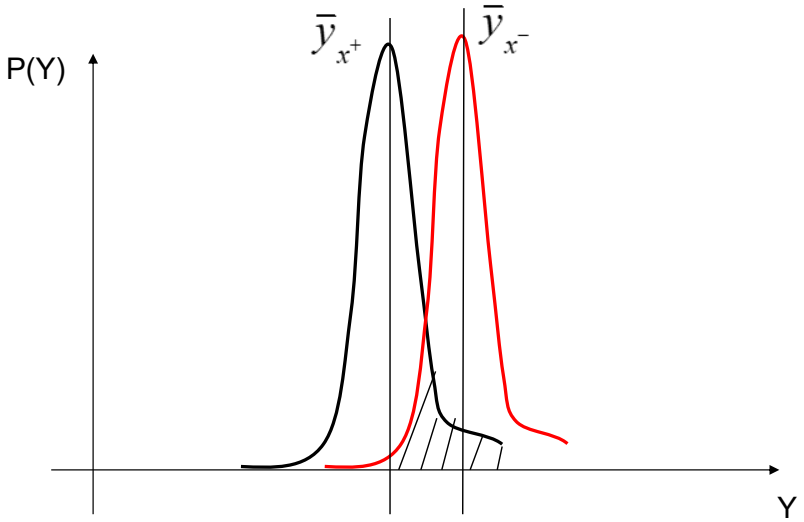
$$s_{x_1} = \sqrt{\frac{\sum_{i=1}^{n_{1,x_1^+}} \left(\bar{y}_{x_1^+} - y_{i,x_1^+}\right)^2 + \sum_{i=1}^{n_{1,x_1^-}} \left(\bar{y}_{x_1^-} - y_{i,x_1^-}\right)^2}{\left(n_{1,x_1^+} + n_{1,x_1^-} - 2\right)\left(n_{1,x_1^+} \cdot n_{1,x_1^-}\right)} \left(n_{1,x_1^+} + n_{1,x_1^-}\right)}$$

dove $\bar{y}^+$ e $\bar{y}^-$ rappresentano le medie della funzione ai due livelli della variabile, mentre s_x è la deviazione standard combinata delle due distribuzioni (o una stima ponderata, in caso di varianze diverse tra i due gruppi). Questo parametro esprime una **differenza normalizzata tra medie**, ponderata in base alla

dispersione dei dati: più è grande il valore assoluto di t, più **le due distribuzioni sono ben separate**, e quindi più è **probabile che il cambiamento osservato nella funzione sia significativo** e non dovuto al caso.

In termini grafici, si può dire che il parametro t di Student **misura l'area di sovrapposizione tra le due curve di distribuzione** associate ai livelli alto e basso della variabile. Un **valore elevato di t** corrisponde a un'area di ricoprimento molto piccola, ovvero a una netta distinzione tra i due stati del sistema. Questo tipo di analisi offre **una maggiore robustezza e affidabilità** rispetto alla semplice analisi delle medie, soprattutto quando si opera in contesti affetti da rumore o incertezza.





Tuttavia, il principale **svantaggio** dell'utilizzo del parametro di t di Student è che **richiede un campionamento più ampio**: non è sufficiente valutare la funzione in pochi punti, ma è necessario disporre di un numero adeguato di osservazioni per stimare in modo affidabile le distribuzioni di probabilità. Questo comporta un **maggiore costo computazionale** e tempi più lunghi.

In sintesi, l'analisi semplificata è utile per una **valutazione preliminare** e rapida, mentre l'utilizzo del **parametro t di Student** consente una comprensione **più profonda e statistica** del sistema. Quando il valore di t è particolarmente elevato, si può concludere che la variabile in questione ha un impatto così netto che

diventa sensato **escludere porzioni del dominio della funzione** meno significative e **concentrare l'analisi nelle aree di maggiore interesse**, ottimizzando così le risorse analitiche.

4.4.4 Criterio di Chauvenet: rilevazione degli outlier nella catena numerica CAO

Il **criterio di Chauvenet** è una metodologia statistica impiegata per identificare **valori anomali (outlier)** in un insieme di dati sperimentali o numerici. A differenza di strumenti come l'analisi della varianza o il t-test di Student, questo criterio **non valuta l'influenza delle variabili di input né la bontà del campo di variazione scelto**, bensì è orientato a **diagnosticare malfunzionamenti o anomalie nel processo numerico** che collega la parametrizzazione geometrica al risultato finale della simulazione.

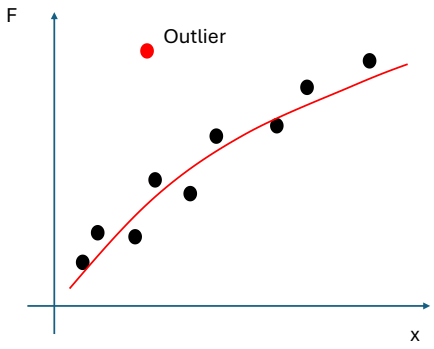
Contesto applicativo

Supponiamo di avere un tipico flusso di simulazione numerica in ingegneria:

- Definizione dei parametri geometrici $(x_1, \dots, x_m)$ →
- Modellazione CAD →
- Generazione della mesh →
- Simulazione (solver) →
- Post-processing dei risultati →
- Output della funzione obiettivo f

Effettuando un **DOE (Design of Experiments)**, si ottiene un insieme di punti nel dominio $x_1 - f(x)$, che può essere rappresentato con una curva di regressione. Tuttavia, può capitare che una o più configurazioni generino un risultato che **si discosta visibilmente dalla tendenza generale**, come illustrato nel grafico: un punto fuori linea rispetto alla curva di regressione.

A questo punto sorge una domanda cruciale: **quel punto è veramente anomalo, oppure rappresenta una naturale variabilità del sistema?**



4.4.4.1 Formalizzazione del criterio

Per rispondere a questa domanda, si applica il **criterio di Chauvenet**, che si basa sulla seguente analisi:

5. **Si costruisce una curva di regressione** sulla base dei dati raccolti.
6. **Si calcolano le deviazioni** tra i valori osservati f_i e quelli stimati dalla regressione $\hat{f}_i$, definite come:

$$\Delta_i = f_i - \hat{f}_i$$

3. **Si valuta la deviazione standard** delle Δ_i:

$$\sigma_\Delta = \sqrt{\frac{\sum\left(\bar{\Delta} - \Delta_i\right)^2}{n-1}}$$

4. Un valore è considerato **statisticamente anomalo** se:

$$| \Delta_i | > \varepsilon_{\text{max}} \cdot \sigma_\Delta$$

Dove:

- ε_{max} è un valore tabulato in funzione della numerosità del campione n;
- σ_Δ è la deviazione standard delle differenze tra i dati e la curva.

Interpretazione tecnica

Se una configurazione viola il criterio di Chauvenet, significa che **il risultato ottenuto non è rappresentativo del comportamento previsto del sistema**, e ciò può dipendere da due categorie di cause:

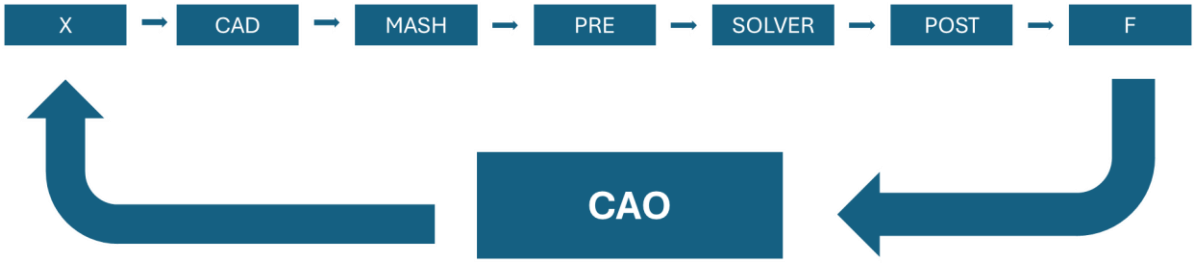
7. **Cause numeriche**:
    - Errori nella generazione del modello (mesh troppo grossolana, errori di interpolazione);
    - Errori di convergenza del solver;
    - Malfunzionamenti nella pipeline CAD–CAE.
8. **Cause fisiche**:
    - Presenza di fenomeni locali non lineari o critici (es. **stallo aerodinamico**, cavitazione, cambi di regime);
    - Comportamenti di soglia che non emergono nella maggior parte delle simulazioni.

Vantaggio rispetto ai modelli parametrici

L'applicazione del criterio di Chauvenet fornisce **un livello superiore di diagnostica** rispetto alla semplice modellazione parametrica: consente di **identificare configurazioni critiche**, isolare eventuali malfunzionamenti nel processo numerico, e migliorare la robustezza dell'analisi successiva (come l'ottimizzazione).

5 Algoritmi di ottimizzazione

Gli algoritmi di ottimizzazione rappresentano il nucleo tecnologico fondamentale all'interno del concetto di Computer-Aided Optimization (CAO). Questi algoritmi costituiscono infatti la metodologia principale utilizzata per individuare la configurazione ottimale di un sistema ingegneristico, con l'obiettivo di massimizzare una o più prestazioni desiderate.



Nel contesto della progettazione avanzata, gli algoritmi di ottimizzazione vengono impiegati per esplorare spazi di progetto spesso complessi e multidimensionali, alla ricerca delle combinazioni di variabili e parametri che conducono al comportamento più efficiente o performante del sistema. Questo processo comporta naturalmente la gestione di compromessi, vincoli e obiettivi multipli, richiedendo strategie matematiche sofisticate.

Esiste una vasta gamma di algoritmi di ottimizzazione, ciascuno con caratteristiche specifiche in termini di logica operativa, tempi di convergenza e idoneità a particolari tipologie di problemi. Si va da metodi basati sul gradiente a tecniche evolutive, fino ad approcci più recenti come quelli basati su modelli surrogati o algoritmi ispirati all'intelligenza artificiale. Questa varietà riflette la diversità delle sfide ingegneristiche affrontate nel mondo reale.

Non ha quindi senso affermare che un algoritmo sia intrinsecamente migliore di un altro: l'efficacia di un algoritmo dipende strettamente dal problema specifico che si intende risolvere. La scelta dell'algoritmo più adatto deve essere guidata dalla comprensione approfondita delle caratteristiche del problema di progetto.

In particolare, due sono i criteri fondamentali da considerare nella selezione dell'algoritmo:

1. **Massimizzazione delle prestazioni**: l'algoritmo deve permettere l'individuazione di una configurazione di sistema che presenti le migliori caratteristiche possibili secondo gli obiettivi prefissati (ad esempio efficienza, robustezza, leggerezza, riduzione dei costi, ecc.);
2. **Efficienza computazionale**: il processo di ottimizzazione deve avvenire nel minor tempo possibile, garantendo al contempo risultati affidabili e accurati.

5.1 Caratteristiche principali degli algoritmi di ottimizzazione

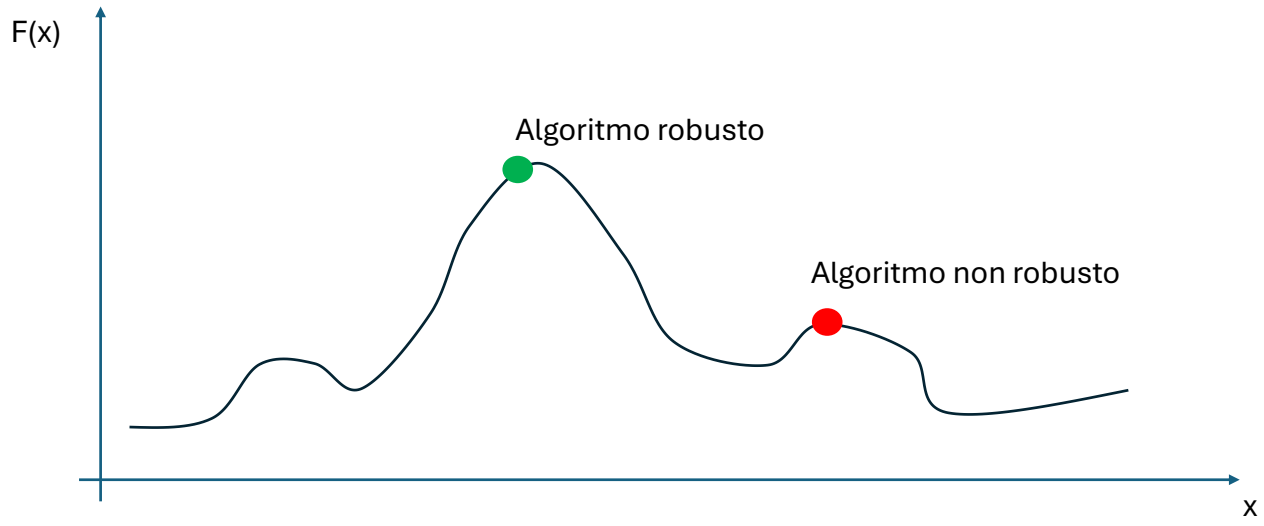
Nella selezione dell'algoritmo di ottimizzazione più adatto a un determinato problema di progettazione, è essenziale valutare attentamente tre proprietà fondamentali:

1. **Robustezza**
2. **Accuratezza**
3. **Velocità di convergenza**

5.1.1 Robustezza

La **robustezza** è una delle caratteristiche più rilevanti per la valutazione della qualità di un algoritmo di ottimizzazione. In termini generali, essa indica la capacità dell'algoritmo di identificare la soluzione globale ottimale—cioè il massimo (o minimo) **assoluto** della funzione obiettivo—indipendentemente dalla configurazione iniziale da cui parte la ricerca.

Molti problemi di ottimizzazione presentano **funzioni obiettivo non convex**, caratterizzate dalla presenza di numerosi **massimi e minimi locali**. In tali casi, un algoritmo può facilmente convergere verso un **ottimo locale** se la sua strategia di esplorazione dello spazio delle soluzioni è limitata o troppo condizionata dal punto iniziale.



Gli algoritmi che tendono a convergere verso l'ottimo più vicino alla configurazione iniziale sono detti **algoritmi locali**. Al contrario, gli algoritmi **globali** sono progettati per esplorare in modo più ampio lo spazio delle soluzioni, aumentando la probabilità di individuare l'ottimo assoluto.

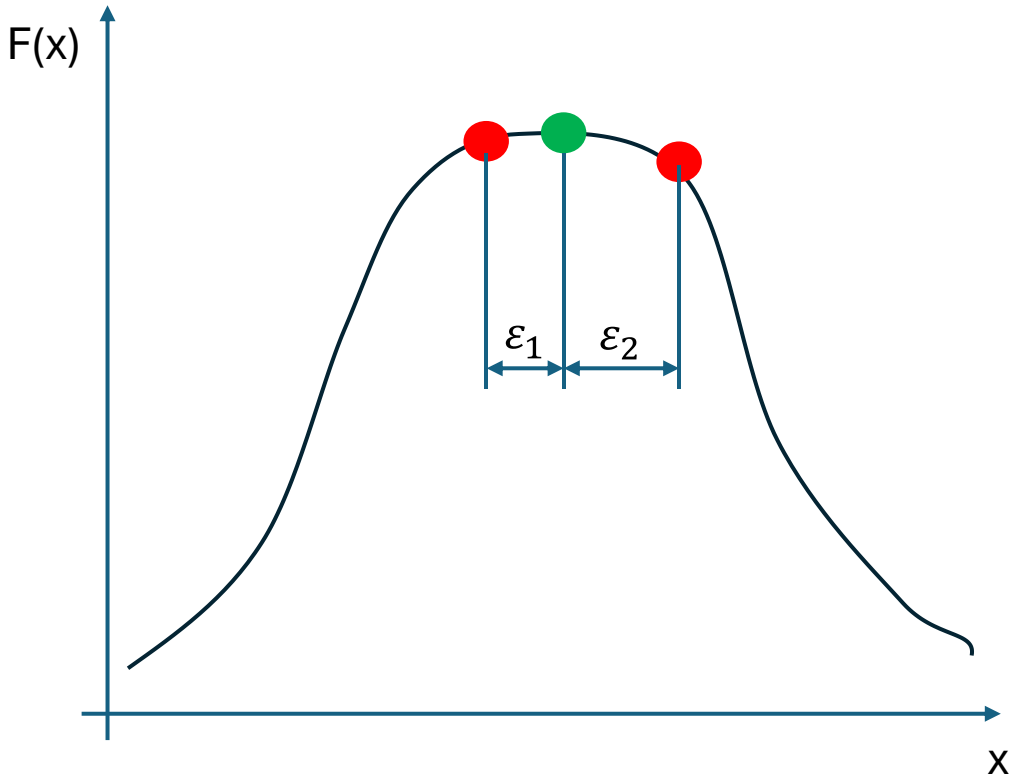
Un indice elevato di robustezza implica:

- **Indipendenza dalla configurazione iniziale**: la soluzione finale trovata non dipende in maniera determinante dal punto di partenza.
- **Capacità di evitare estremi locali**: l'algoritmo riesce a superare barriere numeriche o topologiche che delimitano regioni subottimali dello spazio delle soluzioni.
- **Stabilità del risultato**: piccole variazioni nei dati iniziali o nei parametri non influenzano in modo significativo la qualità della soluzione trovata.

Alcuni algoritmi, come quelli basati su popolazioni (ad esempio, algoritmi genetici), utilizzano **insiemi multipli di configurazioni iniziali**. Questo approccio aumenta la probabilità di coprire una porzione più ampia dello spazio di ricerca e di convergere verso la soluzione globale ottima.

5.1.2 Accuratezza

L'**accuratezza** di un algoritmo di ottimizzazione misura la sua capacità di determinare una soluzione numerica che si avvicini il più possibile al **valore teorico** del massimo (o minimo) assoluto della funzione obiettivo.



Nel contesto dell'ottimizzazione numerica, la funzione obiettivo non è trattata nella sua forma analitica continua, ma viene campionata **punto per punto**, ovvero si valuta il valore della funzione f(x) dato un certo input x. Questo approccio implica inevitabilmente un certo grado di **approssimazione**: l'algoritmo opera infatti con una rappresentazione discreta e finita dello spazio delle soluzioni.

Supponendo che la funzione obiettivo ammetta un **massimo assoluto analitico** ben definito, l'algoritmo non sarà generalmente in grado di determinare esattamente quel valore, ma individuerà un punto all'interno del suo **intorno numerico**, ossia una regione vicina in cui la funzione assume valori molto prossimi a quello teorico massimo. L'**accuratezza** misura quanto il valore trovato si avvicini a tale massimo teorico.

In altri termini:

- Un algoritmo **più accurato** è in grado di individuare un valore di f(x∗) molto prossimo al massimo teorico.
- Un algoritmo **meno accurato** fornisce soluzioni che, pur trovandosi nella stessa regione, si discostano in modo più significativo dal valore massimo effettivo.

5.1.3 Accuratezza vs. Robustezza

Esiste una **correlazione naturale** tra accuratezza e robustezza, che spesso impedisce di massimizzare entrambe le proprietà con un singolo algoritmo:

- Gli **algoritmi robusti**, progettati per esplorare ampie porzioni dello spazio delle soluzioni e per evitare trappole locali, tendono a sacrificare la precisione con cui individuano il massimo nella regione trovata. Sono quindi utili per localizzare **dove** si trova il massimo globale, ma non eccellono nel determinare con precisione **quanto** vale.
- Gli **algoritmi accurati**, al contrario, sono estremamente efficaci nell'identificare il massimo **locale** con grande precisione, ma partendo da una configurazione iniziale sbagliata rischiano di convergere verso soluzioni subottimali (ovvero massimi locali anziché assoluti).

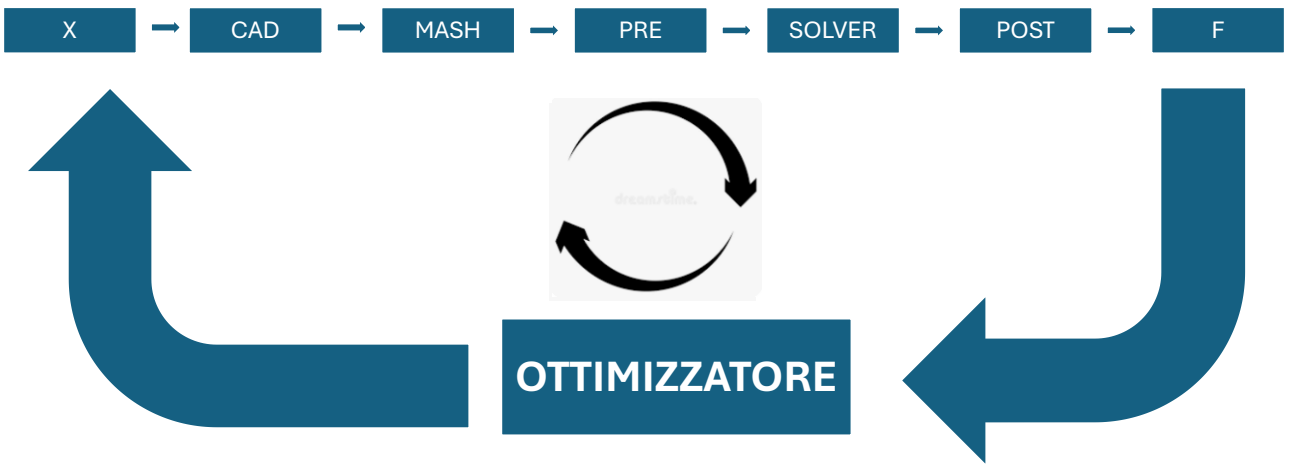
Questa dicotomia ha portato allo sviluppo e all'adozione di **tecnologie ibride**, che combinano più algoritmi in una sequenza strategica, sfruttando i punti di forza di ciascuno. Un approccio tipico e molto efficace prevede:

1. **Fase di esplorazione globale**: utilizzo di un **algoritmo robusto** (es. algoritmi genetici, simulated annealing, particle swarm) per identificare l'**intorno** del massimo assoluto, indipendentemente dalla configurazione iniziale;
2. **Fase di raffinamento locale**: utilizzo di un **algoritmo accurato** (es. metodi basati sul gradiente, quasi-Newton) per ottimizzare con elevata precisione all'interno della regione già individuata.

Questa combinazione consente di superare i limiti intrinseci dei singoli approcci, aumentando sia la probabilità di trovare l'ottimo globale sia la qualità numerica della soluzione finale.

5.1.4 Velocità di convergenza

La **velocità di convergenza** è una misura fondamentale dell'efficienza computazionale di un algoritmo di ottimizzazione. Essa quantifica il **numero di iterazioni** (o, più precisamente, il numero di **valutazioni della funzione obiettivo**) necessarie affinché l'algoritmo individui una soluzione ottimale accettabile, ossia sufficientemente vicina al massimo (o minimo) della funzione obiettivo.



Nel contesto operativo, ogni iterazione dell'algoritmo comporta una **chiamata al solver**, ovvero una richiesta di valutazione della funzione obiettivo o di calcolo di eventuali derivate (nel caso di algoritmi

basati su metodi deterministici o su informazioni di gradiente). Poiché queste valutazioni costituiscono spesso la parte più onerosa dal punto di vista computazionale—soprattutto nei problemi ingegneristici reali, dove la funzione obiettivo può corrispondere a una simulazione numerica complessa—è desiderabile che il **numero totale di chiamate al solver** sia il più contenuto possibile.

Formalmente, maggiore è la **velocità di convergenza**, minore sarà il numero di iterazioni necessarie per raggiungere un punto x^* tale che $f(x^*) \approx \max f(x)$. In simboli:

$$\text{Velocità di convergenza} \propto \frac{1}{N_{\text{solver calls}}}$$

dove $N_{\text{solver calls}}$ è il numero totale di valutazioni richieste per la convergenza.

Il compromesso con robustezza e accuratezza

In generale, esiste una **relazione inversa** tra la velocità di convergenza e le altre due caratteristiche desiderabili—**robustezza** e **accuratezza**:

- Un algoritmo **robusto**, per esplorare ampiamente lo spazio delle soluzioni e ridurre la dipendenza dalla configurazione iniziale, tende a richiedere **un numero elevato di valutazioni**, penalizzando la velocità.
- Un algoritmo **accurato**, che effettua una ricerca locale fine per avvicinarsi il più possibile all'estremo analitico della funzione, spesso necessita di **passi incrementali molto piccoli** e di numerosi cicli di raffinamento, rallentando la convergenza.

Di conseguenza, ottenere **robustezza**, **accuratezza** e **velocità di convergenza** in un unico algoritmo è raramente possibile. Nella pratica ingegneristica, la progettazione del processo di ottimizzazione richiede un compromesso intelligente tra queste tre proprietà, oppure l'adozione di **strategie ibride**.

Un approccio efficace consiste nell'utilizzare:

- **Algoritmi globali robusti**, anche se lenti, per esplorare l'intero dominio e individuare le regioni promettenti;
- **Algoritmi locali rapidi e accurati**, per rifinire la soluzione a partire da una configurazione già vicina all'ottimo.

In questo modo, si riduce il numero totale di valutazioni della funzione obiettivo pur mantenendo elevati livelli di affidabilità e precisione.

5.2 Algoritmi di Ottimizzazione Basati sul Gradiente

Nel contesto dell'ottimizzazione numerica, gli algoritmi si possono suddividere in due grandi categorie a seconda del tipo di informazione utilizzata per guidare la ricerca del minimo (o massimo) di una funzione obiettivo:

Algoritmi che utilizzano il gradiente: Utilizzano la derivata prima (gradiente) della funzione obiettivo, ovvero:

$$\nabla f(x_k)$$

Esempi: *Gradient Descent*, *Newton's Method*, *Conjugate Gradient*, *BFGS*.

Algoritmi che non utilizzano il gradiente (spesso detti *stocastici* o *zero-order methods*): Utilizzano solo il valore della funzione:

$$f(x_0)$$

Esempi: *Simulated Annealing*, *Genetic Algorithms*, *Particle Swarm Optimization*, *Random Search*.

5.2.1 Confronto tra le due classi di algoritmi

Caratteristica	**Algoritmi che usano il gradiente**	**Algoritmi che non usano il gradiente**
Robustezza	Bassa	Alta
Accuratezza	Alta	Bassa
Velocità di convergenza	Alta	Bassa

5.2.2 Analisi delle Proprietà

5.2.2.1 Robustezza

La **robustezza** è definita come la capacità dell'algoritmo di trovare il **massimo (o minimo) globale** della funzione obiettivo, anche in presenza di più ottimi locali. Gli algoritmi che utilizzano il gradiente sono generalmente **sensibili alla scelta del punto iniziale**, in quanto si basano su informazioni locali (il gradiente è una derivata prima locale), e quindi possono facilmente rimanere intrappolati in un minimo locale. Questo compromette la loro robustezza.

Al contrario, gli algoritmi che non usano il gradiente esplorano lo spazio delle soluzioni in modo più globale, talvolta tramite meccanismi di ricerca casuale o euristiche evolutive, permettendo una maggiore capacità di sfuggire agli ottimi locali.

5.2.2.2 Accuratezza

L'utilizzo del gradiente consente di seguire con precisione la direzione di massimo incremento (o decremento) della funzione. Per questo, una volta prossimi a un minimo, questi algoritmi sono in grado di **convergere con elevata accuratezza** verso una soluzione ottimale locale.

Gli algoritmi stocastici, non avendo accesso alla derivata, adottano strategie meno precise e spesso necessitano di più iterazioni per affinare la soluzione, con una precisione generalmente inferiore.

5.2.2.3 Velocità di Convergenza

Gli algoritmi basati sul gradiente, specialmente quelli che usano anche l'informazione di secondo ordine (come il metodo di Newton), tendono a **convergere molto rapidamente** verso un punto stazionario. Tuttavia, tale efficienza è spesso limitata a funzioni ben comportate (lisce, differenziabili e con pochi minimi locali). Gli algoritmi non derivativi, invece, procedono tramite esplorazione casuale o euristica e richiedono **molte più valutazioni della funzione**, risultando in una convergenza tipicamente più lenta.

La principale limitazione degli algoritmi basati sul gradiente è che il **gradiente rappresenta un'informazione locale**, quindi l'algoritmo può dirigersi verso un **ottimo locale** e non globale. L'effetto è illustrato nel diagramma incluso, dove a partire da diversi punti iniziali, il gradiente conduce a minimi diversi (ottimi locali). Questa limitazione si traduce in **bassa robustezza**: se il punto iniziale è mal scelto, il risultato sarà subottimale, sebbene accurato.

Al contrario, gli algoritmi che non utilizzano il gradiente esplorano lo spazio delle soluzioni in modo più **globale**, rendendoli adatti a problemi complessi, discontinui, non derivabili o con molti ottimi locali, pur a scapito della precisione e dell'efficienza.

Gli algoritmi di ottimizzazione che **non fanno uso del gradiente** si basano su un approccio radicalmente diverso rispetto ai metodi derivativi. Invece di partire da un singolo punto iniziale e di esplorare il dominio della funzione basandosi sull'informazione locale (cioè il gradiente), questi algoritmi utilizzano un **insieme di configurazioni iniziali**, distribuite nello spazio delle soluzioni. Questo insieme di punti è generalmente definito come un **Design of Experiments (DOE)**.

L'idea alla base del DOE è quella di **esplorare il dominio della funzione su una griglia di punti scelti in modo da coprire lo spazio delle variabili** il più uniformemente possibile. L'algoritmo valuta la funzione obiettivo su ognuna di queste configurazioni iniziali, e in base ai risultati ottenuti cerca di **guidare l'esplorazione futura** verso le regioni più promettenti.

Questa strategia porta a una **maggiore probabilità di individuare l'ottimo globale**, soprattutto in presenza di funzioni obiettivo **non convexe** o altamente **multimodali**, cioè con molti minimi o massimi locali. La robustezza di tali algoritmi è dovuta al fatto che non si affidano a informazioni locali, ma **esplorano lo spazio in modo globale**, anche se a scapito della velocità e precisione.

5.2.3 Velocità di Convergenza e Calcolo del Gradiente

Gli algoritmi derivativi (che usano il gradiente) sono notoriamente più **rapidi in termini di iterazioni**, ma il loro utilizzo è condizionato dalla necessità di calcolare il gradiente, il che introduce delle complessità computazionali importanti, soprattutto in assenza di espressioni analitiche della funzione.

Infatti, se si ha una funzione:

$$f: \mathbb{R}^n \to \mathbb{R}$$

il **gradiente** è un vettore colonna delle derivate parziali:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Quando **la funzione obiettivo non è differenziabile analiticamente**, il gradiente deve essere approssimato **numericamente**. Il metodo più semplice per farlo è quello delle **differenze finite** (di primo ordine), in cui ciascuna derivata parziale viene stimata come:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_0 + \Delta e_i) - f(x_0)}{\Delta}$$

dove e_i è il vettore unitario nella direzione della i-esima variabile e Δ è un piccolo incremento (passo finito).

Costo computazionale del calcolo numerico del gradiente

Se lo spazio delle variabili è $\mathbb{R}^n$, allora per stimare il gradiente di primo ordine servono **n valutazioni della funzione obiettivo** (una per ogni variabile). Per un **gradiente di secondo ordine** (cioè l'Hessiana), si richiedono fino a $\mathcal{O}(n^2)$ valutazioni. Questo comporta un **costo computazionale elevato**, specialmente quando:

- La funzione è costosa da valutare (es. simulazioni ingegneristiche o CFD).

- Lo spazio delle variabili è ad alta dimensionalità (grande n).

Problemi ingegneristici nel calcolo del gradiente

Il calcolo del gradiente può non solo rallentare la convergenza in termini di tempo reale (anche se richiede meno iterazioni), ma presenta **gravi limitazioni nella pratica ingegneristica**, tra cui:

Problemi nella parametrizzazione

Il gradiente assume implicitamente che **le variabili siano continue, differenziabili e misurabili** in uno spazio dotato di metrica (cioè dove abbia senso parlare di "distanza"). Tuttavia, in ingegneria:

- **Variabili categoriche**: Molti problemi includono variabili categoriche (es. tipo di materiale, forma geometrica, tipo di attuatore) che **non hanno derivata**, né è possibile definire una distanza tra le categorie. Il gradiente quindi **non può essere usato** in modo coerente.

Variabili discrete

Quando una variabile può assumere **solo valori discreti** (es. numero di pale, livelli di potenza, scelte intere), il concetto di variazione infinitesima (su cui si basa il gradiente) **non è più applicabile**. Non esiste un "piccolo incremento" Δ continuo da usare nelle differenze finite.

Scelta del passo Δ

Nel calcolo delle differenze finite, la precisione dell'approssimazione del gradiente dipende dalla **scelta del valore** Δ:

- Se Δ è troppo grande, si perde l'informazione locale e l'approssimazione è imprecisa.
- Se Δ è troppo piccolo, l'errore numerico (dovuto alla sottrazione di numeri molto vicini) può dominare, portando a instabilità.

Pertanto, la scelta ottimale di Δ è delicata e problema-dipendente, con un impatto diretto sulla qualità dell'ottimizzazione.

5.2.4 Calcolo numerico del gradiente e problematiche legate al rumore numerico

Il calcolo del **gradiente** in un contesto numerico si basa sull'approssimazione del limite del rapporto incrementale:

$$\frac{\partial f}{\partial x} \approx \frac{f(x_0 + \Delta) - f(x_0)}{\Delta}$$

dove Δ è un piccolo incremento introdotto nella variabile x, e x_0 è il punto attorno al quale si calcola il gradiente. L'accuratezza di questa stima dipende fortemente dalla scelta di Δ e dalla natura della funzione. In contesti ingegneristici reali, tuttavia, **le funzioni obiettivo sono spesso affette da rumore numerico**, che può derivare da errori di discretizzazione, da perturbazioni geometriche minime o da instabilità del metodo di calcolo.

5.2.5 Effetti del rumore numerico nella valutazione delle derivate

Consideriamo il caso in cui la funzione obiettivo dipenda da una **variabile geometrica continua**, ad esempio una coordinata o una deformazione della superficie di un profilo alare. In uno scenario di progettazione

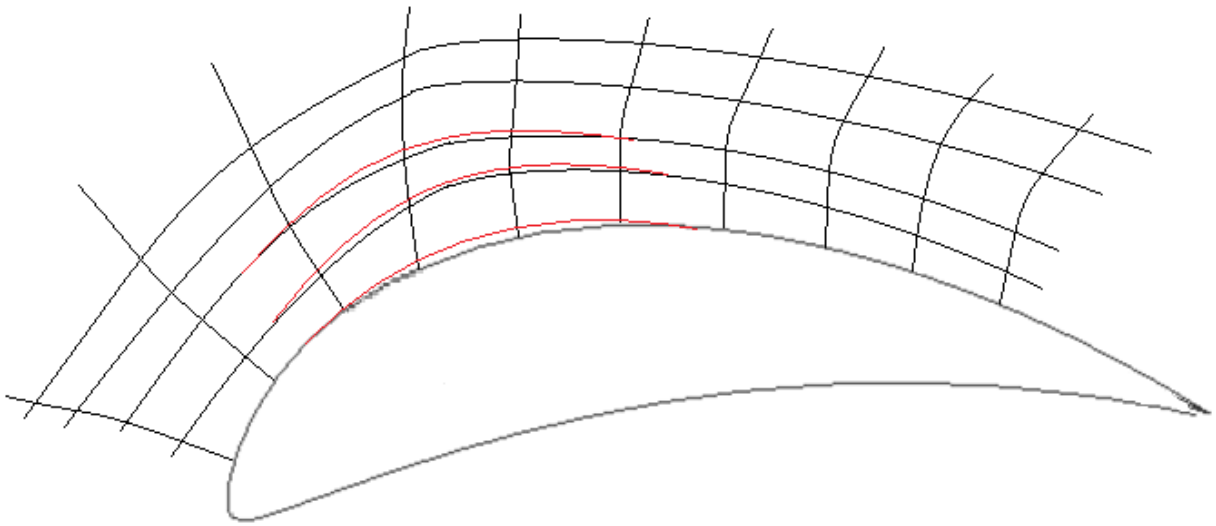
numerica, tale variabile geometrica influisce sulla **mesh di calcolo**, cioè sulla discretizzazione della geometria usata dal solutore per eseguire la simulazione.

Quando si introduce una **variazione infinitesima nella variabile geometrica**, ci si aspetta una variazione infinitesima nella funzione obiettivo, se questa fosse analitica. Tuttavia, nella realtà numerica accade che:

- La mesh viene rigenerata o adattata alla nuova geometria.
- La variazione locale nella geometria può **non riflettersi** in maniera proporzionale sul campo di calcolo.
- **L'errore di discretizzazione** e la **non linearità delle perturbazioni** causano un **rumore numerico** che può sovrastare l'effetto fisico atteso.

5.2.6 Esempio geometrico: perturbazione di una mesh aerodinamica

L'esempio dell'immagine riguarda una **piccola perturbazione di una geometria alare**, discretizzata con una mesh. Una modifica minima della forma (ad esempio uno spostamento di un nodo di controllo) porta a un cambiamento altrettanto minimo della mesh. Tuttavia, il **solutore CFD** può generare un risultato significativamente diverso in termini di coefficienti aerodinamici, ad esempio il **coefficiente di portanza** C_L.



Pertanto:

$$C_{L'} \neq C_L$$

anche se il **risultato geometrico** (cioè la forma effettiva) è praticamente invariato.

Questo è un chiaro esempio di come un **piccolo Δ numerico** (in input) non si traduca necessariamente in un piccolo cambiamento fisico (in output), violando implicitamente l'ipotesi di derivabilità e linearità locale su cui si basa il calcolo del gradiente.

5.2.7 Implicazioni sul calcolo del gradiente numerico

Questo fenomeno impone alcune considerazioni importanti:

- **Δ non può essere scelto arbitrariamente piccolo**: se è troppo piccolo, il rumore numerico maschera completamente la variazione reale della funzione, rendendo il gradiente instabile e non significativo.
- **Δ deve essere "fisico"**: cioè, deve essere scelto in base alla **scala del fenomeno fisico modellato**, non solo su criteri numerici. Ad esempio, se la perturbazione geometrica è dell'ordine della precisione numerica della mesh, è probabile che il gradiente calcolato sia dominato dal rumore.
- **Il gradiente perde significato ingegneristico** in presenza di rumore numerico elevato, ed è necessario introdurre **strategie di regolarizzazione o di media statistica** (es. medie su più simulazioni, uso di surrogate models) per ottenere stime affidabili.

Il calcolo del gradiente numerico in ambito ingegneristico è fortemente condizionato:

- dalla **discretizzazione del dominio** (es. mesh),
- dalla **scelta del passo Δ**, e
- dalla **presenza di rumore numerico**.

In problemi di ottimizzazione numerica, è quindi fondamentale **commisurare il passo Δ alla scala fisica del problema**, e **valutare la sensibilità del sistema** al fine di evitare instabilità nel calcolo delle derivate. In alternativa, in presenza di forti effetti numerici, può essere preferibile ricorrere ad **algoritmi non derivativi**, più robusti in presenza di rumore.

5.3 Moltiplicatori di Lagrange

L'obiettivo di questa trattazione è determinare le condizioni necessarie e sufficienti affinché una funzione ammetta un punto di estremo, mediante l'utilizzo del gradiente della funzione obiettivo. Sia data una funzione:

$$f: \mathbb{R}^n \to \mathbb{R}$$

Affinché un punto $x^* \in \mathbb{R}^n$ sia un punto di estremo (minimo o massimo locale), è necessario che il gradiente della funzione calcolato in x^* sia nullo, ovvero:

$$\nabla f(x^*) = 0$$

Questa rappresenta la **condizione necessaria** per l'esistenza di un punto stazionario.

Condizione sufficiente

Affinché il punto x^* sia effettivamente un estremo (e non semplicemente un punto stazionario), è necessario considerare la **matrice Hessiana** della funzione, ovvero la matrice delle derivate seconde parziali miste, che si esprime come:

$$[H]_{x^*} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}_{x^*}$$

A seconda della natura della matrice Hessiana in x^*, si possono distinguere i seguenti casi:

- Se la matrice Hessiana è **definita positiva**, ovvero tutti gli autovalori sono strettamente maggiori di zero, allora x^* è un **punto di minimo locale**.
- Se la matrice Hessiana è **definita negativa**, ovvero tutti gli autovalori sono strettamente minori di zero, allora x^* è un **punto di massimo locale**.
- Se la matrice Hessiana è **indefinita**, ovvero possiede sia autovalori positivi che negativi, allora x^* è un **punto di sella** (cioè non rappresenta né un minimo né un massimo locale).

5.3.1 Considerazioni sull'ottimalità assoluta o relativa

Per determinare se un punto critico (stazionario) rappresenta un massimo o un minimo **assoluto** e non semplicemente locale, è necessario confrontare i valori della funzione in tutti i punti x^* candidati: il punto che restituisce il valore massimo della funzione sarà un massimo assoluto, e analogamente per il minimo.

Tuttavia, dal punto di vista analitico, questa verifica non è sempre praticabile, specialmente in ambiti ingegneristici dove spesso non è possibile esplorare l'intero dominio della funzione. Per questo motivo, si fa ricorso a **metodi numerici** e **algoritmi di ottimizzazione**, che consentono di individuare i punti nei quali il gradiente della funzione si annulla, fornendo così potenziali estremi da analizzare.

Nel caso in cui la funzione obiettivo sia soggetta a dei **vincoli**, le condizioni di ottimalità precedentemente discusse devono essere modificate. In presenza di vincoli di uguaglianza, è necessario ricorrere all'utilizzo dei **moltiplicatori di Lagrange**.

L'obiettivo è determinare il **minimo** di una funzione scalare $f(\vec{x})$ definita su uno spazio euclideo $\mathbb{R}^n$, soggetta a **m vincoli di uguaglianza** rappresentati da funzioni $g_j(\vec{x}) = 0$, per $j = 1, \dots, m$. Formalmente, il problema può essere espresso come:

$$\min_{\vec{x}} f(\vec{x}) \quad \text{con} \quad \vec{x} \in \mathbb{R}^n$$

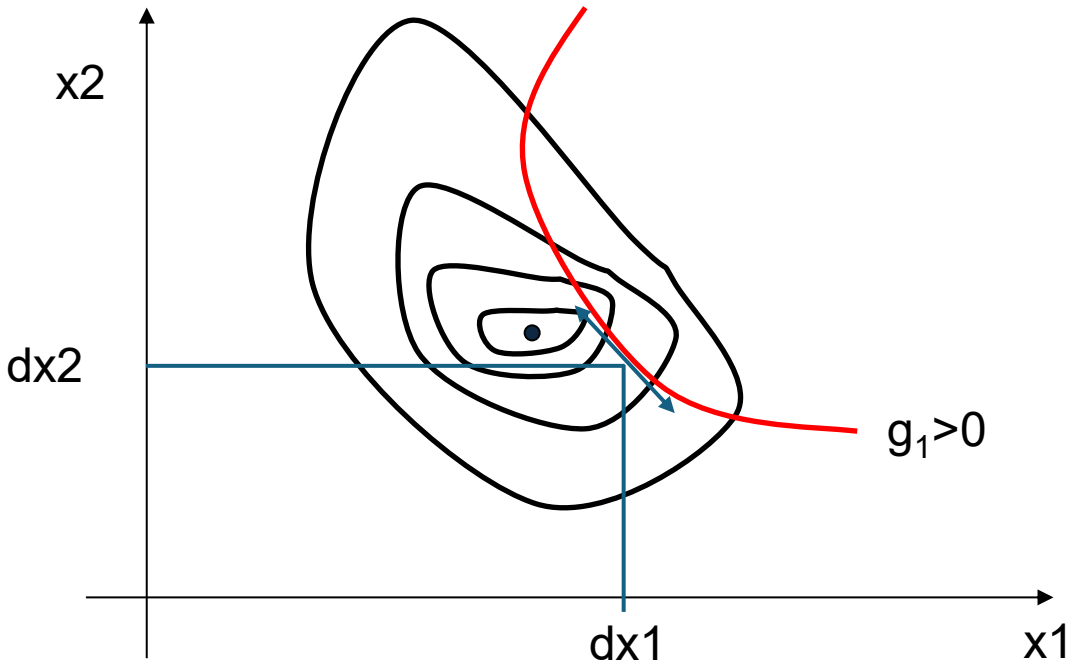
$$\text{s.t.} \quad g_j(\vec{x}) = 0 \quad \text{per } j = 1, \dots, m$$

Per garantire la **compatibilità** del problema (cioè l'esistenza di una soluzione non banale), è necessario che il numero di vincoli sia strettamente inferiore al numero di variabili decisionali, cioè:

$$m < n$$

Se questa condizione non fosse rispettata, si avrebbe un **problema sovravincolato**, ovvero con più vincoli di quanti siano i gradi di libertà, rendendo difficile o impossibile soddisfare simultaneamente tutte le condizioni imposte.

Caso esemplificativo: m = 1, n = 2



Nel caso specifico in cui si abbia **un solo vincolo** (m = 1) e **due variabili** (n = 2), il problema può essere visualizzato geometricamente. L'immagine mostra una serie di **linee di livello** (contour lines) della funzione $f(\vec{x})$, corrispondenti ai valori costanti della funzione in due dimensioni. La **curva rossa** rappresenta il vincolo di uguaglianza:

$$g(\vec{x}) = 0$$

Questa curva vincolata rappresenta l'insieme dei punti che soddisfano la condizione imposta.

Cercare il minimo della funzione f lungo tale curva implica che il punto $\vec{x}^*$ ottimale non possa variare liberamente nello spazio $\mathbb{R}^2$, ma solamente lungo le **direzioni ammissibili**, ovvero tangenti alla curva del vincolo (evidenziate in blu nella figura).

In particolare, se si considera un piccolo spostamento dx_1 lungo l'asse x_1, il corrispondente spostamento dx_2 lungo l'asse x_2 non può essere scelto arbitrariamente: esso **deve rispettare il vincolo**. Questo vincolo impone quindi una **restrizione sulle direzioni possibili** lungo cui cercare l'ottimo.

Questa limitazione introduce la necessità di studiare il problema attraverso il concetto di **moltiplicatori di Lagrange**, che consentono di trasformare un problema vincolato in un problema equivalente non vincolato, mediante la costruzione di una funzione ausiliaria (la funzione di Lagrange) che incorpora i vincoli come penalità nel calcolo dell'ottimo.

Si consideri una funzione $f(x_1, x_2)$ soggetta a un vincolo di uguaglianza $g(x_1, x_2) = 0$. La curva definita da $g(x) = 0$ rappresenta il vincolo, e il problema consiste nel trovare un punto x^* che minimizzi f lungo tale curva. Questo implica che x^* può muoversi solo lungo direzioni **ammissibili**, ovvero tangenti alla curva del vincolo.

Affinché f abbia un estremo in $x^* = (x_1, x_2)$, il differenziale totale di f deve annullarsi:

$$df = \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 = 0$$

Poiché il punto è vincolato, deve valere anche:

$$g(x_1 + dx_1, x_2 + dx_2) = 0$$

Espandendo con una serie di Taylor di primo ordine:

$$g(x_1, x_2) + \frac{\partial g}{\partial x_1} dx_1 + \frac{\partial g}{\partial x_2} dx_2 = 0$$

Dato che $g(x_1, x_2) = 0$, si ottiene:

$$\frac{\partial g}{\partial x_1} dx_1 + \frac{\partial g}{\partial x_2} dx_2 = 0$$

Da cui si ricava la relazione tra le variazioni:

$$dx_2 = -\frac{\frac{\partial g}{\partial x_1}}{\frac{\partial g}{\partial x_2}} dx_1$$

Sostituendo questa relazione nel differenziale di f:

$$df = \left(\frac{\partial f}{\partial x_1} + \frac{\partial f}{\partial x_2} \cdot \frac{dx_2}{dx_1}\right) dx_1 = 0$$

Sostituendo dx_2:

$$\frac{\partial f}{\partial x_1} - \frac{\partial f}{\partial x_2} \cdot \frac{\frac{\partial g}{\partial x_1}}{\frac{\partial g}{\partial x_2}} = 0$$

Da cui si ottiene la **condizione necessaria**:

$$\frac{\partial f}{\partial x_1} \cdot \frac{\partial g}{\partial x_2} = \frac{\partial f}{\partial x_2} \cdot \frac{\partial g}{\partial x_1}$$

Si introduce il **moltiplicatore di Lagrange** λ e si definisce la funzione lagrangiana:

$$\mathcal{L}(x_1, x_2, \lambda) = f(x_1, x_2) + \lambda g(x_1, x_2)$$

Le condizioni necessarie per un estremo vincolato sono:

$$\frac{\partial \mathcal{L}}{\partial x_1} = \frac{\partial f}{\partial x_1} + \lambda \frac{\partial g}{\partial x_1} = 0$$

$$\frac{\partial \mathcal{L}}{\partial x_2} = \frac{\partial f}{\partial x_2} + \lambda \frac{\partial g}{\partial x_2} = 0$$

$$g(x_1, x_2) = 0$$

Per una funzione $f: \mathbb{R}^n \to \mathbb{R}$ soggetta a p vincoli di uguaglianza $g_j(x) = 0$, con $j = 1, \dots, p$, si definisce:

$$\mathcal{L}(x, \lambda_1, \dots, \lambda_p) = f(x) + \sum_{j=1}^{p} \lambda_j \, g_j(x)$$

Le condizioni necessarie per un estremo sono:

$$\nabla_x \mathcal{L} = \nabla f(x) + \sum_{j=1}^{p} \lambda_j \, \nabla g_j(x) = 0$$

$$g_j(x) = 0 \quad \text{per } j = 1, \dots, p$$

Dal punto di vista pratico, la risoluzione del sistema lagrangiano può essere complessa per due motivi:

- **Non linearità**: né la funzione obiettivo f né i vincoli g_j sono necessariamente lineari.
- **Informazione discreta**: spesso f e g_j non sono note in forma analitica, ma solo in forma discreta (per punti).

5.3.2 Interpretazione e significato dei moltiplicatori di Lagrange

Nel contesto dell'ingegneria applicata, l'utilizzo diretto dei moltiplicatori di Lagrange risulta spesso impraticabile. Questo è dovuto alla complessità computazionale e alla difficoltà di ottenere espressioni analitiche per le funzioni obiettivo e i vincoli. Tuttavia, esistono algoritmi numerici, come l'**SQP (Sequential Quadratic Programming)**, che impiegano i moltiplicatori di Lagrange in modo efficace per risolvere problemi di ottimizzazione vincolata.

Consideriamo un problema di ottimizzazione con vincolo di uguaglianza:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{soggetto a} \quad g(x) = b$$

Questo vincolo può essere riscritto come:

$$\tilde{g}(x) = g(x) - b = 0$$

Applichiamo il metodo dei moltiplicatori di Lagrange, definendo la funzione lagrangiana:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda(g(x) - b)$$

Le condizioni di stazionarietà (necessarie per un estremo) sono:

$$\nabla_x \mathcal{L} = \nabla f(x) + \lambda \nabla g(x) = 0$$

$$g(x) = b$$

Ora, supponiamo di perturbare leggermente il vincolo, cioè di variare il valore della costante b.
Differenziando il vincolo otteniamo:

$$db = \nabla g(x)^T dx$$

Poiché $\nabla f(x) + \lambda \nabla g(x) = 0$, possiamo scrivere:

$$df = -\lambda \nabla g(x)^T dx = -\lambda db$$

Da cui segue:

$$\frac{df}{db} = \lambda$$

Questa relazione mostra che il **moltiplicatore di Lagrange** λ rappresenta la **sensibilità** del valore ottimo della funzione obiettivo rispetto a una variazione del vincolo. In altre parole, λ quantifica quanto cambia $f(x)$ al variare di b.

Esaminiamo tre casi distinti:

- $\lambda > 0$: un aumento del vincolo b comporta un aumento del valore minimo di $f(x)$. Il vincolo è attivo e penalizzante.
- $\lambda < 0$: un aumento del vincolo b comporta una diminuzione del valore minimo di $f(x)$. Il vincolo è attivo ma favorisce l'ottimizzazione.
- $\lambda = 0$: il vincolo non influisce sul valore ottimo della funzione. In questo caso, il vincolo è **non attivo** o **non vincolante**.

Questa analisi permette di comprendere il ruolo di ciascun vincolo in un problema con più vincoli: il valore assoluto di λ indica **quanto fortemente il vincolo influenza la soluzione ottima**.

Interessante sono le considerazioni ingegneristiche che si possono fare vedendo per esempio la funzione da minimizzare come la resistenza di un profilo aerodinamico con un vincolo di uguaglianza sulla portanza.

5.3.3 Estensione ai vincoli di disuguaglianza

Nella pratica ingegneristica, i vincoli di disuguaglianza sono molto comuni. Il problema assume la forma:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{soggetto a} \quad g_j(x) \leq 0, \quad j = 1, \dots, m$$

In questo contesto, i moltiplicatori di Lagrange sono ancora utilizzabili, ma con una condizione aggiuntiva: **i moltiplicatori associati ai vincoli di disuguaglianza devono essere non negativi** e **attivi solo quando il vincolo è attivo** (cioè quando $g_j(x) = 0$).

Nel caso di vincoli di disuguaglianza, è possibile trasformare il problema in uno con vincoli di uguaglianza introducendo una variabile ausiliaria y, tale che:

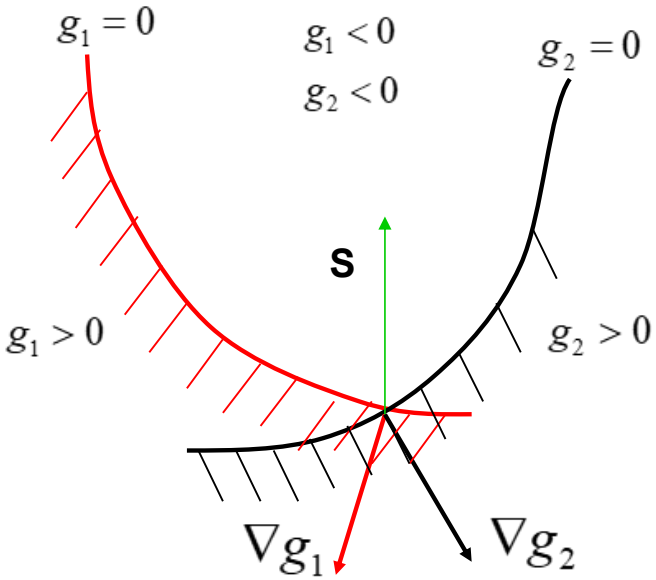
$$G(x, y) = g(x) + y = 0$$

In questo modo, il vincolo originario $g(x) \leq 0$ viene riformulato come un vincolo di uguaglianza, dove $y \geq 0$ garantisce la validità della disuguaglianza. Si definisce quindi la funzione lagrangiana:

$$\mathcal{L}(x, \lambda, y) = f(x) + \lambda G(x, y)$$

La variabile y è ora parte integrante del problema, e deve essere determinata in modo tale da annullare il vincolo. Si procede quindi come nel caso classico dei vincoli di uguaglianza, imponendo le condizioni di stazionarietà:

$$\nabla_x \mathcal{L} = 0, \quad \nabla_y \mathcal{L} = 0$$



Supponiamo di avere due vincoli di disuguaglianza:

$$g_1(x) \leq 0, \quad g_2(x) \leq 0$$

e una funzione obiettivo $f: \mathbb{R}^2 \to \mathbb{R}$. I gradienti ∇g_1 e ∇g_2 sono perpendicolari alle curve di livello dei rispettivi vincoli, mentre ∇f è perpendicolare alle curve di livello della funzione obiettivo.

Per trovare un punto ottimo, si impone:

$$\nabla f + \lambda_1 \nabla g_1 + \lambda_2 \nabla g_2 = 0$$

Moltiplicando scalarmene entrambi i membri per una direzione arbitraria $\mathbf{S}$, si ottiene:

$$-\mathbf{S}^T \nabla f = \lambda_1 \mathbf{S}^T \nabla g_1 + \lambda_2 \mathbf{S}^T \nabla g_2$$

Questa relazione mostra come i moltiplicatori di Lagrange influenzano la direzione di incremento della funzione. Se $\lambda_i > 0$, il vincolo g_i è attivo e limita la direzione di miglioramento. Se $\lambda_i = 0$, il vincolo è non attivo e non influisce sull'ottimizzazione.

5.3.4 Condizioni di Kuhn-Tucker

Nel caso generale di un problema con vincoli di uguaglianza e disuguaglianza:

$$\min f(x) \quad \text{soggetto a} \quad h_j(x) = 0 \ (j = 1, \dots, m), \quad g_i(x) \leq 0 \ (i = 1, \dots, p)$$

le **condizioni di Karush-Kuhn-Tucker (KKT)** forniscono un insieme di condizioni necessarie per l'ottimalità:

$$\nabla f(x) + \sum_{i=1}^{p} \lambda_i \nabla g_i(x) + \sum_{j=1}^{m} \mu_j \nabla h_j(x) = 0$$

$$g_i(x) \leq 0, \quad \lambda_i \geq 0, \quad \lambda_i g_i(x) = 0 \quad \text{(condizioni di complementarità)}$$

$$h_j(x) = 0$$

Dove:

- λ_i sono i moltiplicatori associati ai vincoli di disuguaglianza,
- μ_j sono i moltiplicatori associati ai vincoli di uguaglianza.

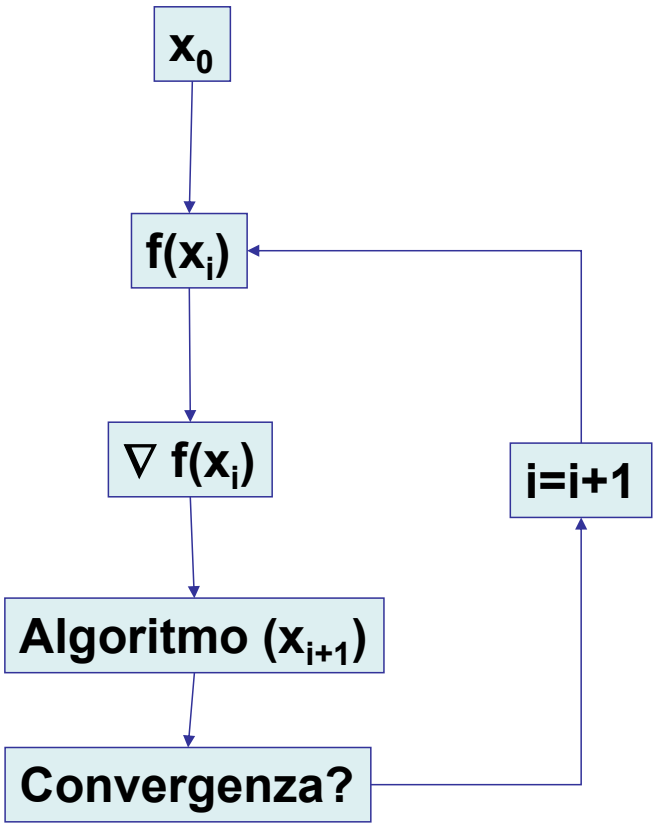
In ambito ingegneristico, la risoluzione analitica del sistema KKT è spesso impraticabile a causa della non linearità delle funzioni coinvolte e dell'impossibilità di calcolare esattamente i gradienti. Per questo motivo, si ricorre ad **algoritmi numerici di ottimizzazione**, che procedono iterativamente verso il minimo o massimo della funzione obiettivo.

5.4 Algoritmi di Ottimizzazione basati sul Gradiente: implementazione

L'idea fondamentale è sfruttare il **gradiente della funzione obiettivo**, che indica la direzione di massimo incremento. In un algoritmo di ottimizzazione, si parte da un punto iniziale e si aggiorna la soluzione lungo la direzione del gradiente (o del gradiente negativo, se si cerca un minimo):

$$x_{k+1} = x_k + \alpha_k \nabla f(x_k)$$

dove α_k è il passo di aggiornamento. Questo approccio è alla base di metodi come il **gradient descent** e le sue varianti.



L'utilizzo del gradiente nella risoluzione di problemi di ottimizzazione è uno degli approcci più diffusi, in particolare per funzioni differenziabili. L'idea alla base è sfruttare l'informazione direzionale fornita dal gradiente della funzione obiettivo $f(\mathbf{x})$, il quale indica la direzione di massimo incremento locale della funzione in un dato punto.

Formalmente, se $f: \mathbb{R}^n \to \mathbb{R}$ è una funzione continua e differenziabile, il gradiente $\nabla f(\mathbf{x})$ rappresenta il vettore delle derivate parziali:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right]^T$$

In un problema di **massimizzazione**, ci si muove nella direzione del gradiente, mentre nei problemi di **minimizzazione** si segue la direzione opposta (cioè il **gradiente negativo**).

L'algoritmo di ottimizzazione iterativa si basa sui seguenti passi:

- **Inizializzazione**: si parte da una configurazione iniziale $\mathbf{x}_0$, scelta sulla base di una conoscenza a priori del problema o arbitrariamente.
- **Valutazione**: si calcola il valore della funzione obiettivo $f(\mathbf{x}_i)$ nel punto corrente.
- **Calcolo del gradiente**: si determina $\nabla f(\mathbf{x}_i)$, che indica la direzione di incremento della funzione.
- **Aggiornamento**: si aggiorna la configurazione secondo la regola:

  $$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha \nabla f(\mathbf{x}_i)$$

  dove $\alpha > 0$ è il passo di apprendimento (learning rate), che determina l'ampiezza del salto nella direzione del gradiente.
- **Verifica della convergenza**: se il miglioramento è inferiore a una certa soglia prefissata (criteri di convergenza), allora l'algoritmo si arresta. Altrimenti, si ripete il ciclo.

Vantaggi dell'approccio

- **Efficienza computazionale**: l'utilizzo del gradiente consente aggiornamenti direzionali rapidi, evitando la necessità di esplorazioni casuali o esaustive dello spazio delle soluzioni.
- **Semplicità di implementazione**: l'algoritmo è facilmente codificabile e applicabile a una vasta gamma di problemi.
- **Flessibilità**: può essere esteso a varianti come il metodo del gradiente con momenti, il metodo del gradiente stocastico (SGD), l'uso di Hessiane approssimate (quasi-Newton) o strategie di adattamento del passo α.

Limitazioni

- **Sensibilità al punto iniziale**: la convergenza è locale e l'algoritmo può convergere verso un massimo locale, non necessariamente globale.
- **Scelta del passo** α: un valore troppo piccolo rallenta la convergenza, uno troppo grande può causare divergenza.
- **Funzioni non lisce o rumorose**: la presenza di discontinuità o rumore può compromettere l'accuratezza del gradiente numerico.

5.4.1 Criteri di Convergenza negli Algoritmi basati sul Gradiente

Nei metodi numerici di ottimizzazione che si basano sull'uso del gradiente della funzione obiettivo, la definizione di criteri di convergenza è fondamentale per garantire l'arresto dell'algoritmo in condizioni accettabili di accuratezza e costo computazionale. I principali criteri di convergenza utilizzati sono i seguenti:

1) Numero massimo di iterazioni

$$n_{\text{iterazioni}} < n_{\text{max iterazioni}} \quad \text{(criterio assolutamente necessario)}$$

Questo criterio impone un limite superiore al numero di iterazioni che l'algoritmo può eseguire. Tale limite viene stabilito a priori in funzione del tempo computazionale disponibile o della complessità del problema. Il suo utilizzo è imprescindibile per evitare cicli infiniti e per garantire la terminazione del processo anche in assenza di miglioramenti significativi.

2) Normale condizione di stazionarietà del gradiente

$$\frac{\partial f}{\partial x_i} = 0$$

In teoria, il punto di minimo (o massimo) di una funzione differenziabile si verifica quando il gradiente della funzione si annulla. Tuttavia, nei calcoli numerici, l'uguaglianza esatta a zero non è mai raggiunta a causa di limiti di precisione numerica. Pertanto, si adotta una forma approssimata:

$$\max_i \left|\frac{\partial f}{\partial x_i}\right| < \varepsilon_1$$

Dove ε_1 rappresenta una soglia di tolleranza prefissata. Il criterio impone che il massimo valore assoluto tra le derivate parziali sia inferiore a ε_1, indicando così una condizione prossima alla stazionarietà.

3) Convergenza assoluta del valore della funzione obiettivo

$$|f(x_{k+1}) - f(x_k)| < \varepsilon_2$$

Questo criterio valuta la variazione assoluta del valore della funzione obiettivo tra due iterazioni successive. Se tale variazione è inferiore alla soglia ε_2, si considera che l'algoritmo non stia apportando miglioramenti significativi e si può quindi arrestare il processo.

4) Convergenza relativa del valore della funzione obiettivo

$$\frac{|f(x_{k+1}) - f(x_k)|}{|f(x_k)|} < \varepsilon_3$$

Il criterio relativo confronta l'incremento della funzione obiettivo rispetto al suo valore corrente. È particolarmente utile quando si lavora con funzioni i cui valori assoluti possono variare su diversi ordini di grandezza. Ad esempio, un miglioramento relativo minore dell'1% ($\varepsilon_3 = 0.01$) può essere considerato trascurabile. Questo criterio è molto importante soprattutto nella fase di assestamento della funzione, in prossimità del minimo locale, dove le oscillazioni possono essere causate dal rumore numerico o sperimentale (come illustrato nel grafico a fianco).

5) Convergenza delle variabili

$$\| x_{k+1} - x_k \| < \varepsilon_4$$

Questo criterio si basa sulla distanza tra due iterazioni successive del vettore delle variabili. Se tale distanza è inferiore a una soglia ε_4, il cambiamento delle variabili è ritenuto irrilevante dal punto di vista

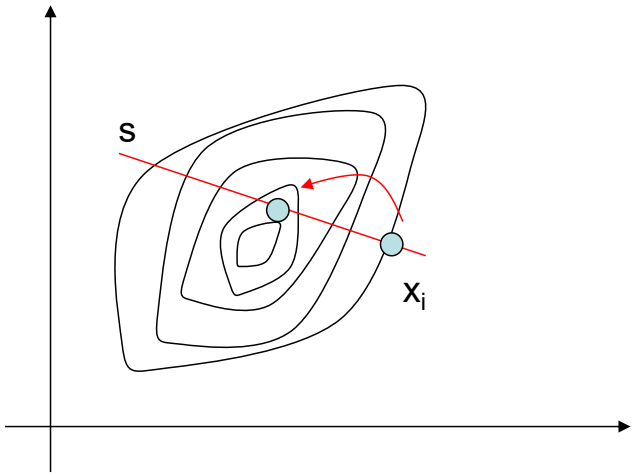
ingegneristico o fisico. Ad esempio, in un problema meccanico, una variazione inferiore a 0.1 mm potrebbe non avere impatto significativo.

Osservazioni finali

Tra tutti i criteri esposti, **il primo** (numero massimo di iterazioni) è **sempre attivo** per motivi di sicurezza computazionale. Gli altri criteri (dal 2 al 5) possono essere attivati o disattivati in base alle esigenze del progettista, alla sensibilità del problema e alla natura della funzione obiettivo. È buona prassi utilizzare almeno **due o tre criteri congiuntamente**, in modo da garantire una convergenza robusta e significativa dal punto di vista fisico e computazionale.

5.4.2 Ricerca dello Step Ottimale lungo una Direzione

Il **gradiente** della funzione obiettivo ci serve per individuare una direzione lungo la quale possiamo ottimizzare la funzione stessa. Una volta nota questa direzione, il problema diventa quello di determinare **di quanto** ci si deve muovere lungo di essa.



Supponiamo di partire da un punto $\mathbf{x}_k$. L'algoritmo ci fornisce una direzione $\mathbf{d}$ lungo la quale dobbiamo muoverci. Per trovare il punto ottimale lungo questa direzione, dobbiamo ottimizzare la funzione:

$$f(\mathbf{x}_k + \lambda \mathbf{d})$$

In questo modo, il problema originariamente bidimensionale (con variabili x_1 e x_2) si riduce a un problema **monodimensionale**, in cui l'unica variabile è λ. La nuova funzione obiettivo diventa:

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

La condizione necessaria per avere un minimo è che il gradiente della nuova funzione sia nullo:

$$\frac{df}{d\lambda} = 0$$

Espandendo la derivata tramite la regola della catena:

$$\frac{df}{d\lambda} = \frac{\partial f}{\partial x_1} \cdot \frac{\partial x_1}{\partial \lambda} + \frac{\partial f}{\partial x_2} \cdot \frac{\partial x_2}{\partial \lambda}$$

Poiché:

$$x_1 = x_1^k + \lambda d_1^k \quad \text{e} \quad x_2 = x_2^k + \lambda d_2^k$$

si ha:

$$\frac{dx_i}{d\lambda} = d_i \quad \text{per } i = 1{,}2$$

Sostituendo:

$$\frac{df}{d\lambda} = \sum_i \frac{\partial f}{\partial x_i} \cdot d_i = \nabla f(\mathbf{x}) \cdot \mathbf{d}$$

Imponendo la condizione di ottimalità:

$$\langle \nabla f(\mathbf{x}), \mathbf{d} \rangle = 0$$

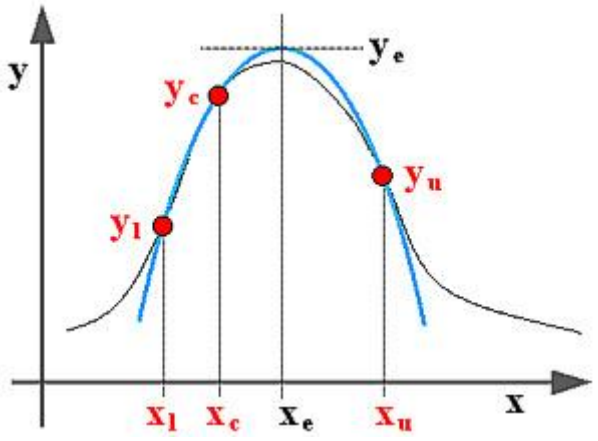
5.4.3 Metodo delle Parabole per la Ricerca del Minimo di una Funzione Univariata

Il **metodo delle parabole**, o **metodo di interpolazione quadratica**, è una tecnica numerica per la determinazione del **minimo (o massimo)** di una funzione reale $f: \mathbb{R} \to \mathbb{R}$, continua e sufficientemente liscia (tipicamente almeno due volte derivabile). Questo metodo si basa sull'assunzione che, localmente, la funzione possa essere approssimata da una parabola.

È un metodo deterministico, non derivativo, e si colloca tra i metodi di ottimizzazione univariata più efficienti quando la funzione obiettivo è ben comportata.

Data una funzione $f(x)$, si scelgono inizialmente **tre punti distinti** $x_1 < x_2 < x_3$, tali che $f(x_2) < \min\{f(x_1), f(x_3)\}$, ovvero x_2 rappresenta il punto centrale con valore di funzione inferiore rispetto agli estremi.

Si interpola quindi una **parabola** $p(x)$ passante per i tre punti $(x_1, f(x_1))$, $(x_2, f(x_2))$, $(x_3, f(x_3))$, e si determina il minimo della parabola x_{min}. Tale punto viene poi utilizzato per aggiornare la terna di punti, ripetendo il processo.



Formula esplicita del minimo della parabola

La parabola che interpola i tre punti ha l'equazione generale:

$$p(x) = ax^2 + bx + c$$

Tuttavia, in pratica si utilizza una formula chiusa per il punto di minimo della parabola x_{min}, calcolata come:

$$x_{\text{min}} = x_2 - \frac{1}{2} \cdot \frac{(x_2 - x_1)^2[f(x_2) - f(x_3)] - (x_2 - x_3)^2[f(x_2) - f(x_1)]}{(x_2 - x_1)[f(x_2) - f(x_3)] - (x_2 - x_3)[f(x_2) - f(x_1)]}$$

Questa espressione deriva dal calcolo del vertice della parabola interpolante usando i tre punti dati.

- Scegli tre punti iniziali x_1, x_2, x_3 tali che $x_1 < x_2 < x_3$ e $f(x_2) < f(x_1), f(x_3)$.
- Costruisci la parabola che interpola i punti.
- Calcola il punto x_{min} che minimizza la parabola.
- Sostituisci uno dei tre punti originali con x_{min}, mantenendo l'ordine dei punti e assicurandoti che $f(x_2)$ resti il valore minimo tra i tre.
- Ripeti fino a che la differenza tra i punti consecutivi o il cambiamento nei valori di $f(x)$ non è inferiore a una tolleranza prefissata ε.

Criteri di arresto

L'algoritmo si considera convergente quando:

- $|x_{\text{min}}^{(k)} - x_{\text{min}}^{(k-1)}| < \varepsilon$, oppure
- $|f(x_{\text{min}}^{(k)}) - f(x_{\text{min}}^{(k-1)})| < \varepsilon$

dove k è l'indice dell'iterazione corrente.

Vantaggi

- Non richiede il calcolo esplicito delle derivate.
- Può essere più efficiente rispetto alla bisezione o al metodo della sezione aurea in prossimità del minimo.
- Buona accuratezza locale se la funzione è ben approssimata da una parabola.

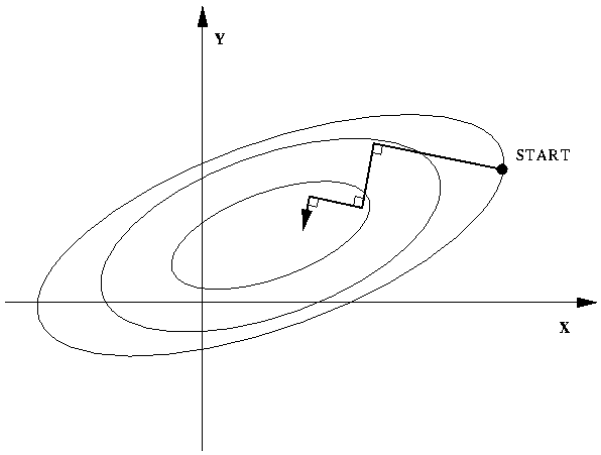
Svantaggi

- L'accuratezza può degradare se la funzione non è sufficientemente regolare.
- Richiede la valutazione della funzione in tre punti a ogni iterazione.
- Può convergere lentamente o instabilmente se i tre punti sono mal scelti o la parabola è poco curvata (quasi lineare).

Il metodo delle parabole rappresenta una tecnica efficace e intuitiva per l'ottimizzazione univariata, specialmente quando la funzione è smooth e ben modellabile localmente da un polinomio di secondo grado. Tuttavia, la sua stabilità e precisione dipendono dalla scelta iniziale dei punti e dalla natura della funzione.

5.5 Algoritmo di Cauchy

L'**algoritmo di Cauchy**, anche noto come **steepest descent** o **metodo del gradiente discendente**, è uno dei metodi iterativi più semplici per la minimizzazione di funzioni differenziabili. È particolarmente utile per problemi non vincolati di ottimizzazione in cui si desidera trovare il minimo locale di una funzione obiettivo $f(\vec{x})$, dove $\vec{x} \in \mathbb{R}^n$.



Idea Fondamentale

L'algoritmo si basa sull'intuizione geometrica che, in un intorno locale, la direzione di massima discesa della funzione coincide con la **direzione opposta al gradiente**. Quindi, dato un punto $\vec{x}_k$, la direzione di discesa più rapida è:

$$\vec{s}_k = -\nabla f(\vec{x}_k)$$

La nuova iterazione è ottenuta aggiornando il punto corrente nella direzione $\vec{s}_k$ con un passo λ_k, detto **step size** o **learning rate**, che può essere determinato con una **line search**:

$$\vec{x}_{k+1} = \vec{x}_k + \lambda_k \vec{s}_k$$

Schema dell'algoritmo

- **Inizializzazione**: si sceglie un punto iniziale $\vec{x}_0$.
- **Calcolo del gradiente**: si valuta $\nabla f(\vec{x}_k)$.
- **Direzione di discesa**: si imposta $\vec{s}_k = -\nabla f(\vec{x}_k)$.
- **Determinazione dello step** λ_k: si risolve un problema di minimizzazione monodimensionale lungo la direzione $\vec{s}_k$, ovvero:
- $\lambda_k = \underset{\lambda}{\operatorname{argmin}} f(\vec{x}_k + \lambda \vec{s}_k)$
- **Aggiornamento del punto**: $\vec{x}_{k+1} = \vec{x}_k + \lambda_k \vec{s}_k$

- **Verifica di convergenza**: se $\| \nabla f(\vec{x}_{k+1}) \|$ è sufficientemente piccolo, il processo si arresta; altrimenti si ripete dal passo 2.

5.5.1 Caso analitico: esempio svolto

Minimizzare la funzione:

$f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$

Punto iniziale $x^{(0)} = [0, 0]$

Gradiente

Poiché la funzione è analitica, possiamo calcolare il gradiente direttamente:

$\nabla f(x) = [\partial f/\partial x_1, \partial f/\partial x_2] = [1 + 4x_1 + 2x_2, -1 + 2x_1 + 2x_2]$

Iterazione 1

Calcolo del gradiente in $x^{(0)}$:

$\nabla f(x^{(0)}) = [1, -1]$

Direzione di discesa:

$S^{(0)} = -\nabla f(x^{(0)}) = [-1, 1]$

Minimizzazione lungo la direzione:

$f(x^{(0)} + \alpha S^{(0)}) = f(-\alpha, \alpha) = \alpha^2 - 2\alpha$

Step ottimale:

$\alpha = 1$

Nuovo punto:

$x^{(1)} = x^{(0)} + \alpha S^{(0)} = [-1, 1]$

Iterazione 2

Calcolo del gradiente in $x^{(1)}$:

$\nabla f(x^{(1)}) = [1 + 4(-1) + 2(1), -1 + 2(-1) + 2(1)] = [-1, -1]$

Direzione di discesa:

$S^{(1)} = -\nabla f(x^{(1)}) = [1, 1]$

Minimizzazione lungo la direzione:

$f(x^{(1)} + \alpha S^{(1)}) = f(-1 + \alpha, 1 + \alpha) = 5\alpha^2 - 2\alpha - 1$

Step ottimale:

$\alpha = 1/5$

Nuovo punto:

$x^{(2)} = x^{(1)} + \alpha S^{(1)} = [-0.8, 1.2]$

Considerazioni teoriche

- Il metodo **converge globalmente** per funzioni convessa.
- La **velocità di convergenza** è **lineare** in generale, ma può peggiorare sensibilmente in presenza di condizionamento elevato (funzioni con livelli di curvatura molto diversi lungo assi ortogonali).
- La **direzione del gradiente** può portare a **"zig-zag"** nel caso di valli strette, rendendo il metodo inefficiente.

Pro e contro del metodo

Vantaggi:

- Semplice da implementare.
- Richiede solo il calcolo del gradiente (no derivata seconda o hessiana).
- Può essere efficace in prossimità del minimo se ben condizionato.

Svantaggi:

- **Convergenza lenta**: particolarmente inefficiente per funzioni mal condizionate.
- **Step numerico problematico**: nella pratica ingegneristica non si ha sempre accesso al gradiente analitico.
- Può fermarsi in **punti stazionari non minimi** se non si applicano opportune verifiche.

Applicazioni ingegneristiche e problemi numerici

Nel contesto dell'ingegneria, l'algoritmo di Cauchy mostra i seguenti limiti:

- **Calcolo del gradiente**: spesso è necessario ricorrere a differenze finite, con perdita di precisione e incremento del costo computazionale.
- **Approssimazioni numeriche**: gli errori di arrotondamento possono compromettere la direzione del gradiente e quindi la discesa.
- In **vicinanza del minimo**, l'approccio può diventare instabile o lento, specialmente se i contorni della funzione sono ellittici ma molto allungati.
- La **"sporcatura" della direzione** dopo più step numerici può portare fuori dalla traiettoria ideale.

5.6 Algoritmo del Gradiente coniugato

Vogliamo minimizzare una funzione quadratica:

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x} + c$$

dove:

- $A \in \mathbb{R}^{n\times n}$ è una matrice **simmetrica e definita positiva**;
- $\vec{b} \in \mathbb{R}^n$, $c \in \mathbb{R}$.

L'algoritmo del gradiente coniugato ricerca una successione di punti $\vec{x}_k$ tale che $f(\vec{x}_k) \to \min$, lungo direzioni $\vec{d}_k$ **coniugate rispetto ad** A.

Si parte da un punto iniziale $\vec{x}_0$. Il **gradiente** della funzione è:

$$\nabla f(\vec{x}) = A\vec{x} - \vec{b}$$

La **prima direzione** $\vec{d}_0$ è sempre scelta come il **gradiente negativo**:

$$\vec{d}_0 = -\nabla f(\vec{x}_0)$$

Dato $\vec{x}_0$ e $\vec{d}_0$, cerchiamo il minimo della funzione lungo la direzione $\vec{d}_0$, cioè troviamo il **passo ottimale** α_0:

$$\vec{x}_1 = \vec{x}_0 + \alpha_0\vec{d}_0$$

Dobbiamo determinare α_0 in modo che $f(\vec{x}_1)$ sia minimo:

$$\alpha_0 = \underset{\alpha}{\operatorname{argmin}} f(\vec{x}_0 + \alpha\vec{d}_0)$$

Sviluppiamo:

$$f(\vec{x}_0 + \alpha\vec{d}_0) = \frac{1}{2}(\vec{x}_0 + \alpha\vec{d}_0)^T A(\vec{x}_0 + \alpha\vec{d}_0) - \vec{b}^T(\vec{x}_0 + \alpha\vec{d}_0) + c$$

Per trovare α_0 ottimale, imponiamo:

$$\frac{d}{d\alpha} f(\vec{x}_0 + \alpha\vec{d}_0) = 0$$

Calcoliamo la derivata:

$$\frac{d}{d\alpha} f(\vec{x}_0 + \alpha\vec{d}_0) = \vec{d}_0^T A(\vec{x}_0 + \alpha\vec{d}_0) - \vec{b}^T\vec{d}_0$$

Sostituendo $\nabla f(\vec{x}_0) = A\vec{x}_0 - \vec{b}$:

$$= \vec{d}_0^T(A\vec{x}_0 - \vec{b}) + \alpha\vec{d}_0^T A\vec{d}_0 = \vec{d}_0^T\nabla f(\vec{x}_0) + \alpha\vec{d}_0^T A\vec{d}_0$$

Ponendo la derivata uguale a zero:

$$\vec{d}_0^T\nabla f(\vec{x}_0) + \alpha_0\vec{d}_0^T A\vec{d}_0 = 0 \Rightarrow \alpha_0 = -\frac{\vec{d}_0^T\nabla f(\vec{x}_0)}{\vec{d}_0^T A\vec{d}_0}$$

Per efficienza, non vogliamo semplicemente ripetere il metodo del gradiente (che converge lentamente), ma usiamo **direzioni coniugate rispetto ad** A, cioè:

$$\vec{d}_k^T A \vec{d}_j = 0 \quad \text{per } k \neq j$$

Si impone quindi una **ricorrenza** per le direzioni:

$$\vec{d}_{k+1} = -\nabla f(\vec{x}_{k+1}) + \beta_k \vec{d}_k$$

dove il parametro β_k è scelto in modo che $\vec{d}_{k+1}$ sia coniugata rispetto a $\vec{d}_k$.

Prendiamo la condizione di coniugatezza:

$$\vec{d}_{k+1}^T A \vec{d}_k = 0$$

Sostituiamo la definizione:

$$(-\nabla f(\vec{x}_{k+1}) + \beta_k \vec{d}_k)^T A \vec{d}_k = 0$$

$$-\nabla f(\vec{x}_{k+1})^T A \vec{d}_k + \beta_k \vec{d}_k^T A \vec{d}_k = 0 \Rightarrow \beta_k = \frac{\nabla f(\vec{x}_{k+1})^T A \vec{d}_k}{\vec{d}_k^T A \vec{d}_k}$$

In alternativa, usando la proprietà:

$$\nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k) = A(\vec{x}_{k+1} - \vec{x}_k) = \alpha_k A \vec{d}_k$$

otteniamo:

$$\vec{g}_{k+1} = \vec{g}_k + \alpha_k A \vec{d}_k \Rightarrow A \vec{d}_k = \frac{\vec{g}_{k+1} - \vec{g}_k}{\alpha_k}$$

Quindi:

$$\beta_k = \frac{\vec{g}_{k+1}^T (\vec{g}_{k+1} - \vec{g}_k)}{\alpha_k \vec{d}_k^T (\vec{g}_{k+1} - \vec{g}_k)}$$

Ma si può semplificare in:

$$\beta_k = \frac{\vec{g}_{k+1}^T \vec{g}_{k+1}}{\vec{g}_k^T \vec{g}_k}$$

Questa è la forma **più comune** usata per il calcolo di β_k nell'algoritmo del gradiente coniugato.

1. Scelta del punto iniziale $\mathbf{X_1}$

2. Calcolo della prima direzione $\mathbf{S_1}$

$$\bar{S}_1 = -\nabla f(\bar{x}_1) = -\nabla f_1$$

3. Calcolo del punto $\mathbf{X_2}$:

$$\bar{x}_2 = \bar{x}_1 + \lambda_1^* \bar{S}_1$$

4. Calcolo della direzione $\mathbf{S_i}$

$$\bar{S}_i = -\nabla f_i + \frac{|\nabla f_i|^2}{|\nabla f_{i-1}|^2} \bar{S}_{i-1}$$

5. Calcolo del nuovo punti $\mathbf{X_{i+1}}$

$$\bar{x}_{i+1} = \bar{x}_i + \lambda_i^* \bar{S}_i$$

6. Convergenza?

5.6.1 Esempio — Metodo del Gradiente Coniugato

Funzione da minimizzare:

$$f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$$

Punto iniziale:

$$x_0 = (0,0)$$

1. Calcolo del gradiente in x_0

$$\nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 + 4x_1 + 2x_2 \\ -1 + 2x_1 + 2x_2 \end{bmatrix}$$

In $x_0 = (0,0)$:

$$\nabla f(x_0) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

2. Prima direzione di discesa

$$d_0 = -\nabla f(x_0) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

3. Calcolo di α_0

Seguiamo la definizione:

$$\alpha_0 = \underset{\alpha}{\operatorname{argmin}} f(x_0 + \alpha d_0)$$

Posto:

$$x(\alpha) = x_0 + \alpha d_0 = (\alpha(-1), \alpha(1)) = (-\alpha, \alpha)$$

Sostituiamo nella funzione:

$$f(-\alpha, \alpha) = (-\alpha) - \alpha + 2(-\alpha)^2 + 2(-\alpha)(\alpha) + (\alpha)^2$$

Sviluppiamo:

$$f(-\alpha, \alpha) = -2\alpha + 2\alpha^2 - 2\alpha^2 + \alpha^2 = -2\alpha + \alpha^2$$

Troviamo il minimo derivando rispetto ad α:

$$\frac{d}{d\alpha} f = -2 + 2\alpha \Rightarrow 0 = -2 + 2\alpha \Rightarrow \alpha_0 = 1$$

4. Aggiorna il punto:

$$x_1 = x_0 + \alpha_0 d_0 = (0,0) + 1 \cdot (-1,1) = (-1,1)$$

5. Calcolo del gradiente in x_1:

$$\nabla f(-1,1) = \begin{bmatrix} 1 + 4(-1) + 2(1) \\ -1 + 2(-1) + 2(1) \end{bmatrix} = \begin{bmatrix} 1 - 4 + 2 \\ -1 - 2 + 2 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

6. Calcolo di β_0

$$\beta_0 = \frac{\| \nabla f(x_1) \|^2}{\| \nabla f(x_0) \|^2} = \frac{(-1)^2 + (-1)^2}{1^2 + (-1)^2} = \frac{2}{2} = 1$$

7. Nuova direzione

$$d_1 = -\nabla f(x_1) + \beta_0 d_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

8. Calcolo di α_1

Poniamo:

$$x(\alpha) = x_1 + \alpha d_1 = (-1,1) + \alpha(0,2) = (-1, 1 + 2\alpha)$$

Sostituiamo nella funzione:

$$f(-1, 1 + 2\alpha) = -1 - (1 + 2\alpha) + 2(-1)^2 + 2(-1)(1 + 2\alpha) + (1 + 2\alpha)^2$$

Sviluppiamo:

- $-1 - 1 - 2\alpha = -2 - 2\alpha$
- $2(-1)^2 = 2$
- $2(-1)(1 + 2\alpha) = -2(1 + 2\alpha) = -2 - 4\alpha$
- $(1 + 2\alpha)^2 = 1 + 4\alpha + 4\alpha^2$

Sommiamo tutto:

$$f = -2 - 2\alpha + 2 - 2 - 4\alpha + 1 + 4\alpha + 4\alpha^2 = (-2\alpha - 4\alpha + 4\alpha) + (-2 + 2 - 2 + 1) + 4\alpha^2$$
$$= -2\alpha + (-1) + 4\alpha^2 = 4\alpha^2 - 2\alpha - 1$$

Deriviamo:

$$\frac{d}{d\alpha} f = 8\alpha - 2 \Rightarrow 0 = 8\alpha - 2 \Rightarrow \alpha_1 = \frac{1}{4}$$

9. Aggiorna il punto

$$x_2 = x_1 + \alpha_1 d_1 = (-1,1) + \frac{1}{4}(0,2) = (-1, 1 + \frac{1}{2}) = (-1, \frac{3}{2})$$

Il **minimo** della funzione si raggiunge in **due iterazioni**:

$$x^* = (-1, \frac{3}{2})$$

Conferma che il metodo del gradiente coniugato ha **convergenza in al più n passi** in $\mathbb{R}^n$, qui $n = 2$.

L'**algoritmo di Cauchy** si muove ad ogni passo lungo la direzione del gradiente negativo corrente. Questo può risultare inefficiente, soprattutto per funzioni con **contorni ellittici allungati**: in questi casi, l'algoritmo tende a **zigzagare**, richiedendo **molti passi** per convergere.
Al contrario, il **metodo del gradiente coniugato** costruisce ad ogni iterazione una direzione di discesa **coniugata rispetto alle precedenti** (cioè ortogonale rispetto al prodotto scalare indotto dalla matrice hessiana), il che garantisce che in **al più nnn** iterazioni (dove nnn è la dimensione del dominio), si giunga esattamente al minimo della funzione quadratica (in aritmetica esatta).
Quindi, **la velocità di convergenza** è **notevolmente superiore** rispetto al metodo di Cauchy, **senza compromessi sull'accuratezza**.

5.7 Metodo di Newton

Espansione di Taylor della funzione $f(\bar{x})$:

$$f(\bar{x}) = f(\bar{x}_i) + \nabla f_i^T(\bar{x} - \bar{x}_i) + \frac{1}{2}(\bar{x} - \bar{x}_i)^T[J](\bar{x} - \bar{x}_i)$$

Condizione di stazionarietà:

$$\nabla f = 0 \quad \Rightarrow \quad \frac{\partial f}{\partial x_j} = 0 \quad \text{per } j = 1, \dots, n$$

Gradiente approssimato:

$$\nabla f = \nabla f_i + [J](\bar{x} - \bar{x}_i) = \vec{0}$$

Formula del metodo di Newton:

$$x_{i+1} = x_i - [J]^{-1}\nabla f_i$$

oppure, con step ottimale λ_i^*:

$$x_{i+1} = x_i + \lambda_i^* S_i = x_i - \lambda_i^*[J]^{-1}\nabla f_i$$

Questo metodo converge su una quadratica di ordine n con 1 unica iterazione, quindi si dimostra molto più veloce rispetto ai metodi visti precedenti, però necessita la valutazione della matrice Hessiana con le derivate seconde che in termini di CAO è ovviamente impossibile da calcolare.

5.8 Metodo di Quasi-Newton

In molte applicazioni di ottimizzazione numerica, il calcolo diretto della matrice Hessiana – ovvero la matrice delle derivate seconde della funzione obiettivo – risulta computazionalmente oneroso oppure impraticabile, ad esempio a causa della complessità della funzione o della mancanza di espressioni analitiche delle derivate seconde. In tali casi, l'approccio tradizionale del metodo di Newton, che richiede esplicitamente la valutazione dell'Hessiana e la sua inversione, non è direttamente applicabile.
Il metodo di **quasi-Newton** rappresenta una valida alternativa, in quanto consente di mantenere le caratteristiche di rapidità di convergenza del metodo di Newton, evitando però il calcolo esplicito della

matrice Hessiana. In particolare, l'idea di base è quella di costruire iterativamente un'approssimazione dell'inversa dell'Hessiana (denotata con B) aggiornata a ogni passo dell'algoritmo sulla base delle informazioni di gradiente disponibili.

$$\lim_{i\to\infty}[H_i] = [J]^{-1}$$

La matrice $[B_k]$ è una matrice **simmetrica**, come la matrice Hessiana. Per trovare la matrice $[B_k]$, si parte dall'espansione di Taylor:

$$f(x) = f(x_0) + \nabla f^T(x_0)(x - x_0) + \frac{1}{2}(x - x_0)^T[H_0](x - x_0)$$

$$\nabla f(\bar{x}) \approx \nabla f(\bar{x}_0) + H_0$$

Gradiente al passo $i+1$:

$$\nabla f_{i+1} = \nabla f(\bar{x}_0) + A_i$$

Gradiente al passo i:

$$\nabla f_i = \nabla f(\bar{x}_0) + A_i$$

Facendo la differenza tra le due

$$[A_i]\vec{d}_i = \vec{g}_i$$

$$\vec{d}_i = \bar{x}_{i+1} - \bar{x}$$

$$\vec{g}_i = \nabla f_{i+1} - \nabla f$$

Invertendo

$$\vec{d} = [B]\vec{g}$$

$$[B] = [A]^{-1} = [J]$$

Aggiornamento della variabile x:

$$x_{i+1} = x_i - \lambda[B]\nabla(x_i)$$

Aggiornamento della matrice B (rango 1):

$$[B_{i+1}] = [B_i] + [\Delta B_i]$$
$$\Delta B_i = czz^T$$
$$[B_{i+1}] = [B_i] + cz\bar{z}^T$$

Condizione forzata:

$$d_i = [B_{i+1}]g_i$$

Sviluppando:

$$d_i = ([B_i] + cz\bar{z}^T)g_i = [B_i]g_i + cz(\bar{z}^T g_i)$$

Poiché $\bar{z}^T g_i$ è uno scalare, si ottiene:

$$cz = \frac{d_j - [B_j]\bar{g}_j}{\bar{z}^T g_j}$$

Separazione dei termini:

$$z = d_j - [B_j]\bar{g}_j$$
$$c = \frac{1}{\bar{z}^T g_j}$$

Formula finale dell'aggiornamento:

$$[B_{i+1}] = [B_j] + \frac{(d_j - [B_j]\bar{g}_j)(d_j - [B_j]\bar{g}_j)^T}{(d_j - [B_j]\bar{g}_j)^T g_i}$$

Uno degli aspetti critici nell'implementazione dei metodi di quasi-Newton, come ad esempio il BFGS, risiede nella stabilità numerica dell'algoritmo e nella conservazione delle proprietà strutturali desiderate della matrice B_k, che rappresenta un'approssimazione dell'inversa della matrice Hessiana.

In particolare, durante l'aggiornamento della matrice secondo le formule ricorsive (come quelle del metodo DFP o BFGS), si possono verificare **fenomeni di instabilità numerica**. Un caso tipico si manifesta quando il denominatore delle formule di aggiornamento, spesso rappresentato da un prodotto scalare del tipo $s_k^T y_k$, diventa **eccessivamente piccolo**. Questo può portare a una divisione numericamente instabile, causando la generazione di una matrice B_{k+1} mal condizionata o addirittura non numericamente significativa.

Un secondo problema riguarda la **positività definita** della matrice B_k. Anche qualora l'approssimazione corrente B_k sia simmetrica e definita positiva, **non è garantito** che l'aggiornamento successivo B_{k+1}, ottenuto applicando le formule standard, mantenga tale proprietà. La **definitezza positiva** è però fondamentale: essa assicura che la direzione di discesa calcolata all'iterazione corrente sia effettivamente una **direzione di discesa** (cioè, che il prodotto scalare $-\nabla f(x_k)^T p_k$ sia negativo), condizione necessaria per la convergenza dell'algoritmo verso un minimo locale.

Il venir meno della definitezza positiva può compromettere seriamente il comportamento dell'algoritmo, inducendo **direzioni errate** che non riducono il valore della funzione obiettivo, o che addirittura conducono l'iterazione lontano dal punto di minimo.

Per far fronte a tali problematiche, nella pratica si preferisce utilizzare **formule di aggiornamento di rango 2**, come nel caso del metodo BFGS, che è progettato per garantire – **sotto condizioni deboli** (in particolare, che $s_k^T y_k > 0$) – che l'approssimazione dell'inversa dell'Hessiana **resti definita positiva**. In particolare, l'aggiornamento BFGS costruisce una combinazione di due termini di rango 1 tale da preservare simmetria e positività definita, migliorando la robustezza numerica del metodo.

Infine, per assicurare ulteriormente la stabilità dell'algoritmo, in implementazioni pratiche si adottano spesso tecniche di **modifica del gradiente**, **damping** (smorzamento dell'aggiornamento) o **verifiche di curvatura** (cioè controlli sul valore di $s_k^T y_k$) al fine di prevenire aggiornamenti degeneri o instabili.

5.8.1 Aggiornamento di rango 2:

$$[\Delta B_i] = c_1 \bar{z}_1 \bar{z}_1^T + c_2 \bar{z}_2 \bar{z}_2^T$$

Espressione della direzione d_i:

$$d_i = [B_i]\bar{\bar{g}}_i + c_1 \bar{z}_1 (z_1^T g_i) + c_2 \bar{z}_2 (z_2^T g_i)$$

Definizioni ausiliarie:

$$\bar{\bar{z}}_1 = d_i$$

$$\bar{\bar{z}}_2 = [B_i] g_i$$

Coefficienti scalari:

$$c_1 = -\frac{1}{z_1^T g_i}$$

$$c_2 = -\frac{1}{z_2^T g_i}$$

Formula finale dell'aggiornamento della matrice B:

$$[B_{i+1}] = [B_i] + [\Delta B_i] \equiv [B_i] - \frac{\bar{\bar{d}}_i d_i^T}{\bar{\bar{d}}_i^T g_i} - \frac{([B]g)([B]g)^T}{([B]g)^T g}$$

Si giunge così all'agoritmo Broyden-Fletcher-Goldfarb-Shanno (BFGS)

1) Inizializzare $\bar{x}_1$ (soluzione di partenza)

Inizializzare $[\mathbf{B}_1] \equiv [\mathbf{I}]$

Calcolare $\nabla f_1 = \nabla f(\bar{x}_1)$

2) $\bar{S}_i = -[\mathbf{B}_i]\nabla f_i$

3) $\bar{x}_{i+1} = \bar{x}_i + \lambda_i^* \bar{S}_i$

4) se $\left\|\nabla f_{i+1}\right\| \le \varepsilon \quad \bar{x}^* = x_{i+1}$

5) $$[\mathbf{B}_{i+1}] = [\mathbf{B}_i] + \left(1 + \frac{\bar{g}_i^T [\mathbf{B}_i] \bar{g}_i}{\bar{d}_i^T \bar{g}_i}\right) \frac{\bar{d}_i \bar{d}_i^T}{\bar{d}_i^T \bar{g}_i} - \frac{\bar{d}_i \bar{g}_i^T [\mathbf{B}_i]}{\bar{d}_i^T \bar{g}_i} - \frac{[\mathbf{B}_i] \bar{g}_i \bar{d}_i^T}{\bar{d}_i^T \bar{g}_i}$$

dove

$\bar{d}_i = \bar{x}_{i+1} - \bar{x}_i = \lambda_i^* \bar{S}_i$

$\bar{g}_i = \nabla f_{i+1} - \nabla f_i$

5.8.2 Metodo BFGS: Efficienza, Stabilità e Ri-inizializzazione

Tra i metodi della famiglia quasi-Newton, il **metodo BFGS** (Broyden–Fletcher–Goldfarb–Shanno) è considerato il più efficiente e comunemente utilizzato, sia nella teoria dell'ottimizzazione numerica che nelle applicazioni pratiche. La sua popolarità è dovuta a un ottimo compromesso tra efficienza computazionale, stabilità numerica e rapidità di convergenza.

Approssimazione dell'inversa della matrice hessiana

Il metodo BFGS non si basa sul calcolo esplicito della matrice Hessiana $H(x)$, ma costruisce iterativamente un'approssimazione della **sua inversa**, denotata con $B_k \approx H(x_k)^{-1}$. Questa strategia consente di evitare i costi computazionali elevati associati all'inversione diretta di matrici di grandi dimensioni e di aggirare la necessità di conoscere le derivate seconde della funzione obiettivo.

L'aggiornamento della matrice B_k è progettato per garantire, sotto opportune condizioni, che B_k resti simmetrica e definita positiva. Una delle condizioni cruciali affinché la definitezza positiva sia preservata è che il prodotto scalare $s_k^T y_k$ sia strettamente positivo. Questo requisito è soddisfatto automaticamente **se lo step ottimale** λ lungo la direzione di discesa viene calcolato **esattamente** (ovvero mediante **line search analitica** che risolve il sottoproblema monodimensionale in modo accurato).

Impatto della line search in pratica

Tuttavia, nelle implementazioni pratiche dell'algoritmo, il passo λ_k non viene solitamente calcolato in maniera esatta, ma tramite una **line search approssimata**, ad esempio soddisfacendo solo condizioni di tipo Wolfe o Armijo. In questi casi, può accadere che il requisito $s_k^T y_k > 0$ non sia rispettato, con conseguente **perdita della definitezza positiva** della matrice B_k, e quindi della correttezza della direzione di discesa.

Ri-inizializzazione della matrice B_k

Per ovviare a tali situazioni, una pratica comune consiste nella **ri-inizializzazione** periodica della matrice B_k, sostituendola con una **matrice identità** (o una sua opportuna scalatura). Questa operazione agisce da "reset" numerico, ristabilendo le proprietà desiderate della matrice e migliorando la stabilità dell'algoritmo. In molti software di ottimizzazione, la ri-inizializzazione è automatica quando si rilevano condizioni di curvatura non soddisfatte o instabilità numeriche durante l'aggiornamento.

Tale strategia consente di proseguire l'ottimizzazione mantenendo le prestazioni elevate del metodo BFGS senza compromettere l'integrità numerica del processo.

5.9 Trattazione Dei Vincoli

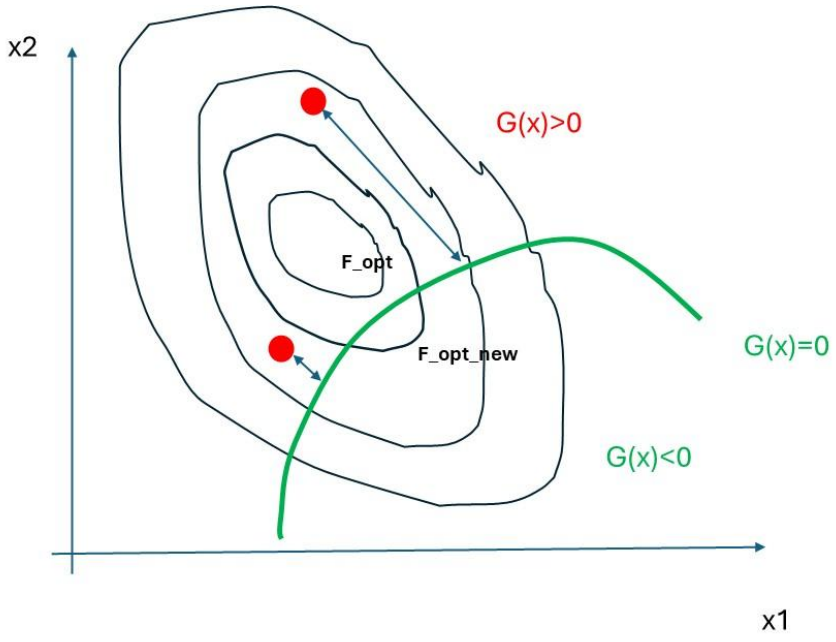
1. **Introduzione all'ottimizzazione vincolata**

Nel contesto dell'ottimizzazione, la **presenza di vincoli** (sia di uguaglianza sia di diseguaglianza) è cruciale per modellare problemi reali. Il problema generale si formula come:

$$\begin{aligned} \text{min/max} \quad & f(x), \quad x \in \mathbb{R}^n \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \dots, p \\ & h_j(x) = 0, \quad j = 1, \dots, q \end{aligned}$$

dove $f(x)$ è la funzione obiettivo, $g_i(x)$ i vincoli di diseguaglianza e $h_j(x)$ i vincoli di uguaglianza.

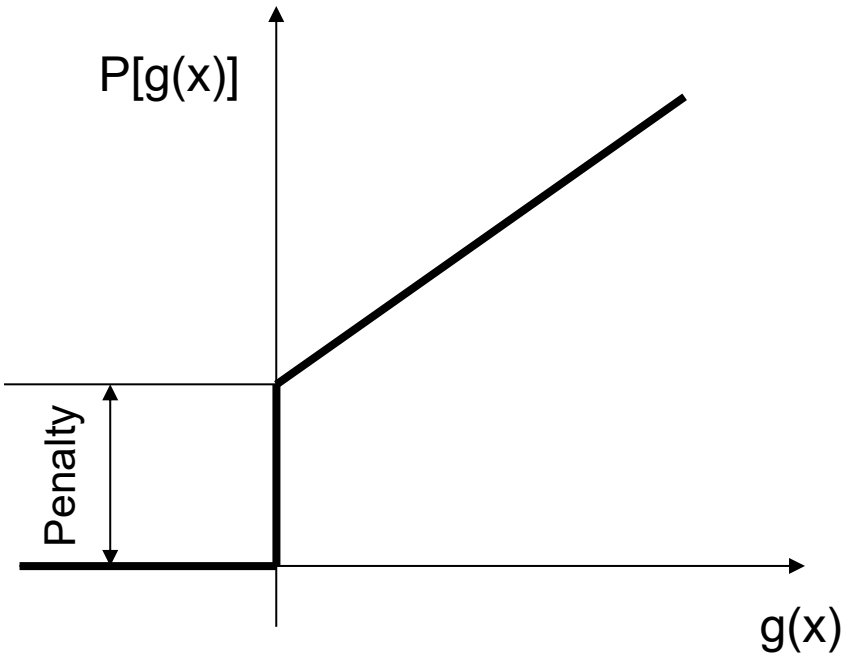
3. **Funzione di penalità (Penalty Function)**



Per trattare i vincoli in algoritmi che altrimenti risolverebbero problemi non vincolati, si introduce il concetto di **funzione di penalità**, che modifica la funzione obiettivo aggiungendo un termine penalizzante per le violazioni dei vincoli:

$$FIT(x) = f(x) - P(g(x))$$

dove $P(g(x))$ è una funzione che restituisce un valore positivo se $g(x) > 0$ (cioè il vincolo è violato) e zero altrimenti.



Esempio di penalità:

Si può utilizzare una penalità di tipo "gradino":

$$P(g(x)) = \begin{cases} 0, & \text{se } g(x) \leq 0 \\ \alpha \cdot g(x), & \text{se } g(x) > 0 \end{cases}$$

Il parametro α (detto *penalty*) ha un ruolo critico: se troppo piccolo, la penalizzazione sarà inefficace; se troppo grande, può sovrastimare l'importanza del vincolo rispetto alla funzione obiettivo. Una buona pratica consiste nel fissare α a circa **10 volte l'ordine di grandezza** di $f(x)$.

Nel caso di più vincoli, si costruisce una funzione di penalità aggregata:

$$FIT(x) = f(x) - \sum_{i=1}^{p} P\,(g_i(x))$$

Ogni vincolo contribuisce con la sua penalizzazione. Questo approccio è flessibile ma sensibile alla scelta dei pesi relativi dei vincoli.

In contesti pratici (per es. ingegneria strutturale o aerodinamica), si può introdurre una **soglia di tolleranza** ε per modellare le incertezze numeriche:

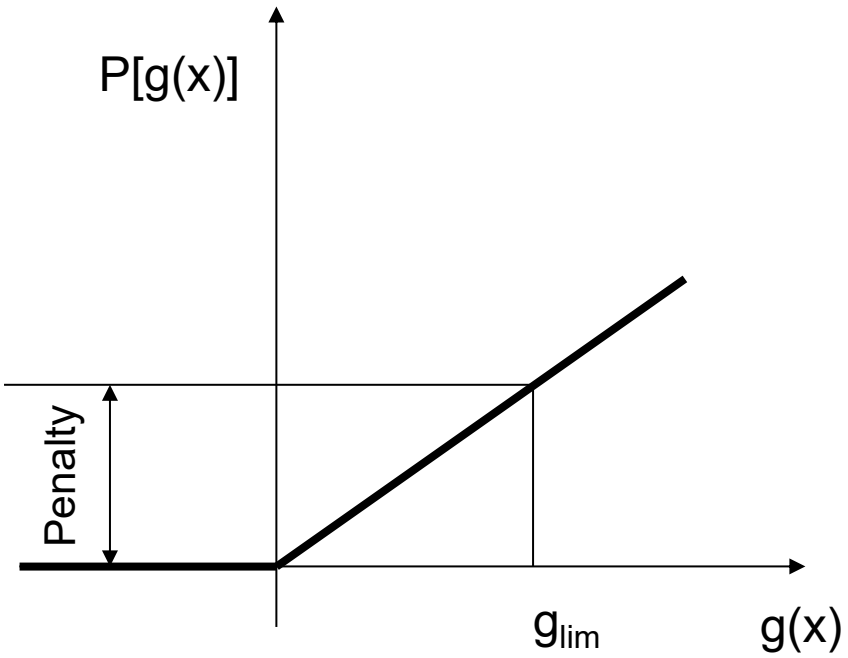
$$h(x) = 0 \quad \Rightarrow \quad |h(x)| \leq \varepsilon$$

così da evitare rigidità computazionali.

- **Penalità "hard"**: penalizzazione severa con uno scalino netto.
- **Penalità "soft"**: introduzione di una funzione continua (es. quadratica o esponenziale) che cresce con la distanza dal vincolo, permettendo una penalizzazione graduale:

$$P(g(x)) = \alpha \cdot \max(0, g(x))^2$$

Questa forma è spesso preferita per evitare discontinuità e instabilità numeriche.

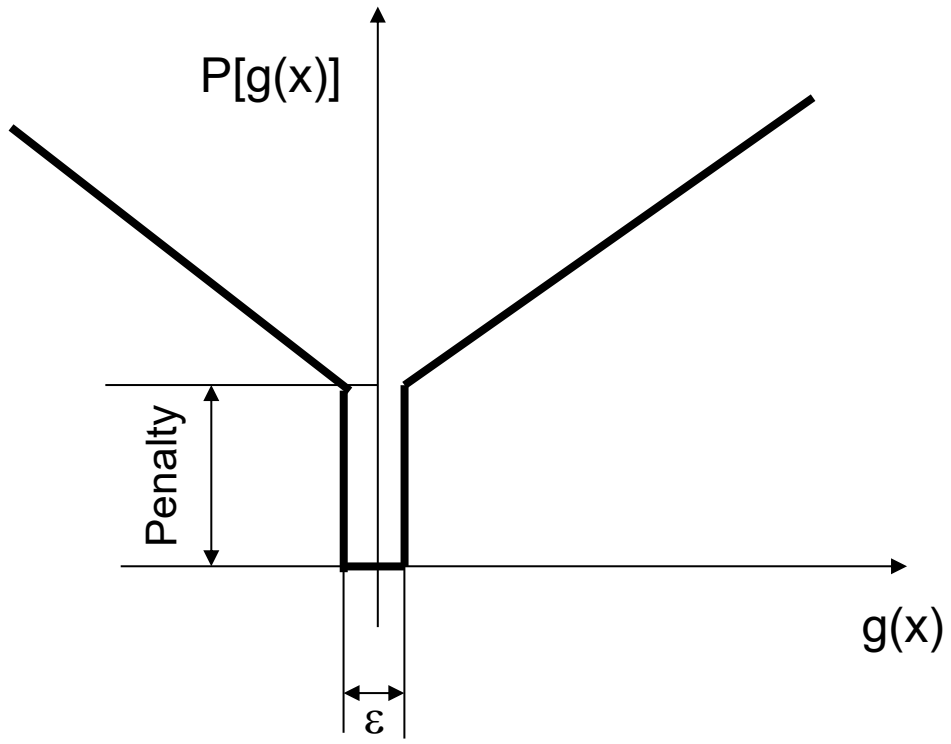


5.9.1 Vincoli di uguaglianza nella pratica ingegneristica

Dal punto di vista numerico, un vincolo come $h(x) = 0$ è irrealistico. Si usa invece:

$$|h(x)| \leq \varepsilon$$

con ε tipicamente pari a 10^{-4} o 10^{-6}, compatibile con la precisione numerica del solver.



Sebbene l'uso di penalità sia diffuso per la sua semplicità, presenta limiti:

- Richiede un'attenta taratura del coefficiente α
- Non garantisce sempre la convergenza a una soluzione ammissibile
- Può compromettere la qualità della soluzione se il gradiente è disturbato dalle penalizzazioni.

In alternativa, metodi più rigorosi come **SQP (Sequential Quadratic Programming)** utilizzano i **moltiplicatori di Lagrange** e forniscono trattamenti teorici più robusti, a scapito della complessità implementativa.

5.10 Algoritmo SQP (Sequential Quadratic Programming)

L'**algoritmo SQP** è uno dei metodi più efficaci per la risoluzione di problemi di **ottimizzazione non lineare vincolata**, in particolare quando i vincoli sono sia di uguaglianza che di disuguaglianza. La sua peculiarità risiede nell'impiego **diretto e sistematico dei moltiplicatori di Lagrange** all'interno della formulazione dell'algoritmo stesso: non si tratta di una strategia di penalizzazione o correzione, ma di una **integrazione esplicita della teoria dei vincoli nella costruzione dell'algoritmo**.

Vantaggi principali

- **Velocità di convergenza elevata**: grazie alla natura *second-order* dell'approccio, l'algoritmo SQP raggiunge la soluzione con un numero di iterazioni significativamente inferiore rispetto ad altri metodi, come quelli basati su penalità o metodi di gradiente.
- **Gestione analitica dei vincoli**: i vincoli (sia $h_i(x) = 0$ che $g_j(x) \leq 0$) vengono incorporati **analiticamente** tramite la costruzione del **Lagrangiano**:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{p} \lambda_i \, h_i(x) + \sum_{j=1}^{q} \mu_j \, g_j(x)$$

dove λ e μ sono i moltiplicatori di Lagrange per i vincoli di uguaglianza e disuguaglianza rispettivamente.

- **Flessibilità e potenza teorica**: l'algoritmo è basato su una formulazione solida dal punto di vista dell'analisi convessa e della teoria della dualità. Tuttavia, proprio per questa potenza, **l'implementazione è non banale**, e varia sensibilmente da autore ad autore, con differenze nei metodi di calcolo della Hessiana, strategie di aggiornamento, e tolleranze numeriche.

L'algoritmo SQP (Sequential Quadratic Programming) si basa sull'**approssimazione locale** del problema originario con un **problema quadratico vincolato**, che viene risolto iterativamente.

Si parte da:

$$\min f(x) : \mathbb{R}^n \to \mathbb{R}$$

$$h_k(x) = 0 \quad \text{per } k = 1, \dots, p$$

(è l'unico caso in cui nella teoria parto con il mettere un vincolo).

Per risolvere il problema utilizzo il Lagrangiano:

$$L(x, \lambda) = f(x) + \sum_{k=1}^{p} \lambda_k \, h_k(x)$$

Le condizioni Kuhn-Tuker per risolvere questo problema sono:

$$\begin{cases} \nabla f + \sum_{k=1}^{p} \lambda_k \, \nabla h_k(x) = 0 \\ h_k(x) = 0 \end{cases}$$

Avrò quindi un sistema fatto da $n + p$ equazioni. Posso scrivere questo sistema in modo matriciale:

$$\begin{bmatrix} \nabla f + H^T \lambda \\ h \end{bmatrix} = 0$$

Dove la matrice H contiene tutte le derivate prime di tutti i vincoli.

Posso allora riscrivere il sistema come una funzione:

$$F(Y) = 0 \quad \text{con} \quad F = \begin{Bmatrix} \nabla L \\ h \end{Bmatrix}, \quad Y = \begin{Bmatrix} x \\ \lambda \end{Bmatrix}$$

Da un punto di vista numerico posso utilizzare il metodo ricorsivo di Newton:

$$Y_{j+1} = Y_j + \Delta Y_j$$

$$\nabla F^T(Y_j)\Delta Y_j = -F(Y_j)$$

Scriviamo questa equazione ricordandoci quanto vale F:

$$x_{j+1} = x_j + \Delta x_j$$

$$\lambda_{j+1} = \lambda_j + \Delta\lambda_j$$

In questo sistema prendiamo il primo set di equazioni (prima moltiplicazione riga-colonna):

Otteniamo questa forma e sfruttiamo il concetto di quadratico. Creiamo la forma quadratica:

$$Q = \nabla f^T x + \frac{1}{2}\Delta x^T \nabla^2 L \Delta x$$

Con i vincoli:

$$h_k + \nabla h_k^T \Delta x = 0 \quad \text{per } k = 1, \dots, p$$

Ottimizziamo la forma quadratica. Essendo vincolata occorre usare nuovamente il Lagrangiano:

$$L^2 = \nabla f^T x + \frac{1}{2}\Delta x^T \nabla^2 L \Delta x + \sum_{k=1}^{p} \lambda_k \left(h_k + \nabla h_k^T \Delta x\right)$$

Nuovamente applico le condizioni di Kuhn-Tucker:

$$\begin{cases} \nabla f + \nabla^2 L \Delta x + H^T \lambda = 0 \\ h_k + \nabla h_k^T \Delta x = 0 \end{cases}$$

Questo è il sistema che consente di risolvere la forma quadratica costruita localmente. L'equazione risolutiva di questo problema quadratico coincide con quella del problema originario non lineare. In altre parole, abbiamo costruito una **forma duale** del problema iniziale, che condivide con esso la stessa soluzione. Pertanto, se ottimizzo la funzione quadratica Q, le variabili che la ottimizzano coincidono con quelle che ottimizzano la funzione f, soggetta ai vincoli.

Se prendiamo la forma quadratica (da minimizzare):

$$Q = \nabla f^T \Delta\bar{x} + 1/2\ \Delta\bar{x}^T\ [\nabla^2 L]\ \Delta\bar{x}$$

con i vincoli

$$h_k + \nabla h_k^T \Delta\bar{x} = 0 \qquad k = 1, \dots, p,$$

la forma lagrangiana diventa

$$\tilde{L} = \nabla f^T \Delta\bar{x} + 1/2\ \Delta\bar{x}^T\ [\nabla^2 L]\ \Delta\bar{x} + \sum_{k=1}^{p} \lambda_k \left(h_k + \nabla h_k^T \Delta\bar{x}\right).$$

Con le condizioni di Kuhn–Tucker:

$$\begin{cases} \nabla f + [\nabla^2 L]\ \Delta\bar{x} + [H]\ \lambda = 0 \\ h_k + \nabla h_k^T \Delta\bar{x} = 0, \quad k = 1, \dots, p \end{cases}$$

Come si nota, la minimizzazione della forma quadratica porta alle stesse equazioni della forma originale.

5.10.1 Vantaggi

- La funzione da ottimizzare è quadratica, quindi è possibile applicare un metodo di Newton, che garantisce una rapida convergenza locale (se le condizioni sono soddisfatte).

5.10.2 Problemi

- All'interno della funzione quadratica compaiono **derivate seconde miste**: questo implica che, nella pratica, **non verrà mai utilizzata la forma quadratica esatta**, ma una **forma approssimata**, con conseguente introduzione di un errore.
- In particolare, la matrice $\nabla^2 L$ (Hessiana del Lagrangiano) è approssimata numericamente.
- Per tale motivo **non è possibile utilizzare direttamente un algoritmo di Newton completo**, ma è necessario ricorrere a un **algoritmo di tipo quasi-Newton**, che approssima la Hessiana in modo efficiente ma non esatto.

5.11 Algoritmi basati sul gradiente nella progettazione ingegneristica: conclusioni

Gli **algoritmi basati sul gradiente della funzione obiettivo** continuano ad essere strumenti fondamentali nella **progettazione e ottimizzazione di sistemi ingegneristici complessi**. Sono utilizzati in settori come l'aerospazio, l'ingegneria meccanica, l'energia, l'automotive e la robotica, per l'ottimizzazione di prestazioni, costi, affidabilità o efficienza energetica.

Il principio di fondo è l'utilizzo delle **informazioni derivate** (cioè il gradiente $\nabla f(x)$) per guidare l'evoluzione della soluzione verso un estremo locale o globale della funzione obiettivo $f(x)$.

Vantaggi principali

Elevata accuratezza della soluzione

Poiché il gradiente fornisce una direzione **localmente ottimale** di discesa (o salita), questi algoritmi sono capaci di determinare con precisione il punto di minimo/massimo. Se la funzione è regolare (continua, differenziabile) e ben condizionata, l'accuratezza numerica può arrivare fino alla **precisione macchina** (es. 10^{-8} o inferiore).

2. Convergenza rapida (poche iterazioni)

In molte applicazioni, i metodi del gradiente richiedono **un numero ridotto di iterazioni** per raggiungere la convergenza. Questo è particolarmente vero per:

- funzioni **quasi-convesse** o **fortemente convesse**;
- quando si dispone di un **gradiente analitico**;

- in combinazione con metodi second-order (es. Newton, quasi-Newton).

Per esempio, il metodo di Newton converge **quadraticamente** vicino all'ottimo, mentre il metodo del gradiente classico converge **linearmente**, ma con costo per iterazione molto più basso.

Problematiche e limiti

Bassa robustezza (sensibilità al punto iniziale)

I metodi del gradiente **non esplorano lo spazio delle soluzioni**: si muovono solo in base a informazioni locali. Questo li rende **vulnerabili ai minimi/massimi locali**, in particolare in:

- spazi ad alta dimensionalità (10–100 variabili);
- funzioni non convesse o multimodali;
- superfici con molte discontinuità nel gradiente.

Una cattiva scelta del punto iniziale può portare alla **convergenza verso un estremo subottimale**.

Costo del calcolo delle derivate parziali

Il calcolo del gradiente può essere oneroso, specialmente quando:

- non è disponibile in forma **analitica**;
- si deve stimare numericamente tramite **differenze finite** o **automatic differentiation**;
- ogni valutazione della funzione è costosa (es. richiede una simulazione CFD o FEM).

In uno spazio a n dimensioni, una stima per differenze finite richiede **almeno** $n + 1$ valutazioni della funzione, moltiplicando drasticamente il costo computazionale.

3. Incompatibilità con variabili discrete o intere

I metodi del gradiente **richiedono variabili continue e differenziabili**. Quando si introducono **variabili intere o discrete** (es. numero di pale in una turbina, numero di supporti strutturali), il gradiente **non è definito**.

Questo comporta:

- Impossibilità di usare discesa del gradiente classica;
- Necessità di **rilassare il problema** (es. usare una variabile continua poi arrotondata);
- Oppure usare **algoritmi misti** (gradienti per variabili continue + euristiche o algoritmi evolutivi per quelle discrete).

Osservazione finale

Nonostante le limitazioni, i metodi basati sul gradiente restano la **scelta preferenziale** quando:

- La funzione obiettivo è regolare;
- I vincoli sono gestibili tramite Lagrangiano (es. con SQP);
- Le variabili sono tutte continue;
- Si cerca efficienza locale e precisione.

5.12 Algoritmi stocastici

A partire dagli anni ’90, sono stati sviluppati nuovi metodi di ottimizzazione per superare alcune criticità associate agli algoritmi classici basati sul gradiente della funzione obiettivo. In particolare, tali metodi mirano a risolvere due problemi fondamentali: da un lato, la scarsa **robustezza** degli algoritmi del gradiente, spesso soggetti a convergere verso ottimi locali; dall’altro, la loro **limitazione nell’uso esclusivo di variabili continue e differenziabili**, che ne impedisce l’applicazione in presenza di variabili intere o categoriche.

Gli algoritmi stocastici nascono proprio con l’intento di superare questi ostacoli. Si tratta di metodi che **non utilizzano il gradiente** della funzione obiettivo ma si basano unicamente sulla valutazione dei **valori della funzione stessa** in punti diversi dello spazio delle variabili. Questo li rende più semplici dal punto di vista matematico e più adatti a una vasta gamma di problemi pratici, anche di natura complessa o scarsamente formalizzabile.

Un aspetto chiave di questi algoritmi è il fatto di **non partire da un singolo punto iniziale**, come avviene nei metodi deterministici, ma da un insieme di configurazioni iniziali. Questa strategia, nota come **DOE (Design of Experiments)**, consente di esplorare fin da subito **diverse regioni del dominio della funzione**, fornendo così una visione più completa e globale del comportamento della funzione stessa. La conoscenza simultanea della funzione in molteplici punti aiuta a comprendere l’andamento generale del paesaggio della funzione obiettivo, aumentando significativamente la probabilità di trovare un **ottimo globale** o comunque una soluzione di buona qualità.

Inoltre, gli algoritmi stocastici sono particolarmente **flessibili** rispetto alla natura delle variabili: non richiedendo calcoli differenziali, possono essere applicati anche a problemi che coinvolgono **variabili discrete, intere o categoriche**, come ad esempio il numero di palette in una turbina, il numero di antenne in un sistema radar o la configurazione ottimale di moduli in un layout produttivo.

Un altro vantaggio significativo di questi algoritmi è la loro **facile parallelizzabilità**. Poiché il calcolo della funzione avviene indipendentemente in ciascun punto della popolazione, è possibile distribuire le valutazioni su più processori o nodi di calcolo. Questo è particolarmente utile in presenza di **modelli complessi**, come quelli basati su simulazioni numeriche intensive (ad esempio CFD, FEM, modelli termici), che richiedono tempi di calcolo elevati.

Tuttavia, i metodi stocastici presentano anche alcune **limitazioni**. In primo luogo, non avendo accesso al gradiente della funzione, non sono in grado di determinare con precisione la direzione in cui migliorare la soluzione. Questo significa che, pur riuscendo a individuare la **zona del massimo o minimo globale**, spesso **non riescono a determinare il punto ottimo con precisione analitica**. Il comportamento è simile a una “esplorazione a salti” dello spazio delle soluzioni, che richiede **molte iterazioni** per affinare il risultato.

Per ovviare a questo limite, è ormai comune l’uso di **strategie ibride**. In questi approcci, si utilizza un algoritmo stocastico nella fase iniziale per individuare l’area più promettente del dominio, e successivamente si applica un algoritmo basato sul gradiente (come Newton o SQP) per perfezionare la soluzione. In questo modo, si combinano i vantaggi dei due approcci: la **robustezza globale** degli algoritmi stocastici con l’**accuratezza locale** dei metodi deterministici.

Un’altra difficoltà è legata al trattamento dei **vincoli**. Nei metodi deterministici, i vincoli sono gestiti in modo rigoroso, ad esempio tramite i **moltiplicatori di Lagrange** e le condizioni di ottimalità di Karush-Kuhn-Tucker. Negli algoritmi stocastici, invece, il trattamento dei vincoli avviene quasi sempre mediante **funzioni di penalità**, che aggiungono un costo artificiale alla funzione obiettivo quando un vincolo è violato. Sebbene

questo approccio sia meno preciso e teoricamente meno elegante, nella pratica si dimostra comunque efficace nella maggior parte dei casi ingegneristici.

Dal punto di vista classificatorio, gli algoritmi stocastici si dividono in tre grandi famiglie principali, anche se ciascuna di esse comprende **numerose varianti** ed evoluzioni:

1. **Simplex (Nelder-Mead)**: un metodo deterministico ma senza uso di derivate, basato su una figura geometrica che si adatta alla topologia della funzione. È semplice e utile per problemi di bassa dimensione.
2. **Simulated Annealing**: ispirato al processo fisico di raffreddamento di un metallo, permette di accettare soluzioni peggiori con una certa probabilità, in modo da evitare di rimanere intrappolati in ottimi locali. È utile per l'esplorazione globale e funziona bene in presenza di rumore o discontinuità.
3. **Algoritmi evolutivi**: si ispirano ai principi dell'evoluzione biologica, come selezione naturale, mutazione e crossover. Tra questi, i più noti sono i **Genetic Algorithms**, la **Differential Evolution** e il **Particle Swarm Optimization**. Sono estremamente adattabili, robusti, e rappresentano oggi lo **stato dell'arte** per l'ottimizzazione globale, specialmente in problemi multiobiettivo o con variabili miste.

In conclusione, gli algoritmi stocastici hanno rivoluzionato il campo dell'ottimizzazione numerica, offrendo **robustezza, flessibilità e parallelizzabilità**, a scapito però della precisione locale. L'adozione di approcci ibridi rappresenta oggi una strategia matura per ottenere il meglio da entrambi i mondi: l'esplorazione globale fornita dagli algoritmi stocastici e la precisione locale garantita dai metodi del gradiente.

5.12.1 Algoritmo SIMPLEX (di Nelder-Mead)

L'algoritmo Simplex è una tecnica di ottimizzazione numerica utilizzata per **minimizzare una funzione reale di più variabili**:

$$\min f(x), \quad x \in \mathbb{R}^n$$

In questo approccio, eventuali **vincoli** sono gestiti indirettamente attraverso l'uso di **funzioni di penalità** (*penalty functions*), che modificano la funzione obiettivo in modo da scoraggiare soluzioni non ammissibili.

Il concetto di simplesso

L'idea fondamentale dell'algoritmo è basata su una figura geometrica chiamata **simplesso**. In uno spazio a n dimensioni, un simplesso è un poliedro costituito da $n + 1$ vertici. Alcuni esempi:

- In due dimensioni ($n = 2$), il simplesso è un **triangolo**.
- In tre dimensioni ($n = 3$), è un **tetraedro**.
- In generale, in $\mathbb{R}^n$, è la forma più semplice che può "muoversi" nello spazio ed esplorare il dominio della funzione.

Strategia dell'algoritmo

A ogni iterazione, l'algoritmo valuta i valori della funzione nei $n + 1$ vertici del simplesso. Tra questi, individua il **vertice peggiore**, cioè quello in cui la funzione assume il valore più alto (se si sta minimizzando).

L'obiettivo dell'algoritmo è **migliorare progressivamente la forma e la posizione del simplesso**, andando a sostituire il vertice peggiore con un nuovo punto che produca un valore migliore della funzione. Per fare

ciò, l'algoritmo utilizza una combinazione di **trasformazioni geometriche**, che vengono applicate in sequenza logica:

- **Riflessione** Il simplesso viene riflesso rispetto al centroide degli altri vertici, con l'intento di esplorare la direzione opposta al vertice peggiore.
- **Espansione** Se la riflessione ha portato a un miglioramento significativo, si prova a espandere ulteriormente il simplesso in quella direzione, cercando un valore ancora migliore.
- **Contrazione** Se la riflessione non ha migliorato la situazione, il simplesso viene contratto: si avvicina il vertice peggiore agli altri per esplorare zone più prossime del dominio.

In casi estremi in cui nessuna delle operazioni produce miglioramenti, l'intero simplesso può essere **ridotto** verso il vertice migliore, nella speranza di restringere la ricerca attorno a un minimo locale.

5.12.1.1 Riflessione

Consideriamo il caso in cui lo spazio abbia due dimensioni, ovvero $n = 2$. In questo caso, il simplesso sarà costituito da $n + 1 = 3$ vertici, che possiamo indicare come x_1, x_2, x_3. Supponiamo che il vertice x_H sia quello corrispondente al **valore peggiore** della funzione obiettivo, cioè:

$$x_H = \text{vertice con } f(x_H) = \max\{f(x_i)\}$$

L'idea della **riflessione** è quella di sostituire questo vertice sfavorevole con un nuovo punto riflesso rispetto al **centroide** degli altri due vertici, che hanno un valore della funzione migliore.

Il centroide x_0 è semplicemente il punto medio dei vertici rimanenti (cioè escluso x_H):

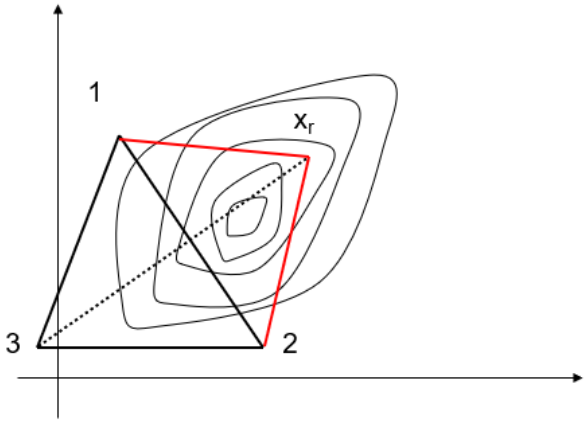
$$x_0 = \frac{1}{n} \sum_{\substack{i=1 \\ i \neq H}}^{n+1} x_i$$

Il nuovo punto riflesso x_R si calcola come:

$$x_R = (1 + \alpha)x_0 - \alpha x_H$$

dove:

- $\alpha > 0$ è un parametro di **riflessione** (tipicamente si pone $\alpha = 1$);
- il punto x_R si troverà quindi **dalla parte opposta rispetto a** x_H, rispetto al centroide x_0.



L'intuizione geometrica è che stiamo cercando di esplorare la **direzione opposta** al vertice peggiore, nella speranza che in quella direzione il valore della funzione sia più basso. Questo passaggio rappresenta il primo tentativo dell'algoritmo per migliorare la forma del simplesso.

- Se $f(x_R)$ risulta **migliore di tutti gli altri vertici**, allora abbiamo individuato una **direzione promettente**, e sarà il caso di procedere con un'**espansione** (per cercare punti ancora migliori nella stessa direzione).

5.12.1.2 Espansione

Dopo aver eseguito una **riflessione**, può succedere che il nuovo punto ottenuto, x_R, risulti **migliore** rispetto a tutti gli altri vertici del simplesso, cioè:

$$f(x_R) < f(x_i) \quad \text{per ogni } i \neq H$$

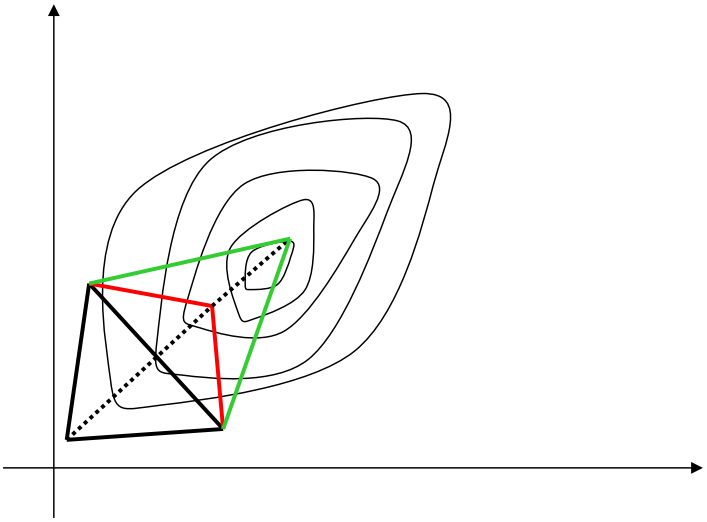
In questo caso, significa che la **direzione esplorata tramite la riflessione è particolarmente promettente**. Allora, invece di fermarci lì, possiamo **continuare a esplorare in quella stessa direzione**, andando oltre il punto riflesso. Questo passo prende il nome di **espansione**.

Il nuovo punto espanso x_E si ottiene estendendo il segmento tra il centroide x_0 e il punto riflesso x_R:

$$x_E = (1 - \gamma)x_0 + \gamma x_R = x_0 + \gamma(x_R - x_0)$$

dove:

- x_0 è il centroide dei vertici diversi da x_H,
- x_R è il punto riflesso,
- $\gamma > 1$ è il **coefficiente di espansione**, normalmente impostato a $\gamma = 2$.



In sostanza, x_E si trova **ancora più avanti rispetto a** x_R lungo la stessa direzione, partendo da x_0.

Dopo aver calcolato x_E, si confrontano i valori della funzione nei punti riflesso ed espanso:

- Se $f(x_E) < f(x_R)$: → L'espansione ha avuto successo: **si accetta** x_E come nuovo vertice al posto del peggiore x_H, perché ha portato a un miglioramento ancora maggiore.
- Se $f(x_E) \geq f(x_R)$: → L'espansione **non è risultata vantaggiosa**: si preferisce **mantenere** x_R, che resta comunque migliore di x_H.

Il concetto dietro l'espansione è semplice: se la riflessione è andata molto bene, forse ci stiamo muovendo nella **direzione del minimo**, e conviene tentare un passo più lungo. Ma per evitare di peggiorare la situazione, verifichiamo sempre che il punto espanso sia effettivamente migliore.

5.12.1.3 Contrazione

Dopo aver effettuato la **riflessione**, può succedere che il nuovo punto x_R sia **ancora peggiore** del vertice di partenza x_H, ovvero:

$$f(x_R) > f(x_H)$$

In questo caso, la direzione esplorata con la riflessione si è rivelata **non favorevole**, e anche proseguire in quella direzione potrebbe non essere utile. È quindi possibile che ci si trovi **in una zona del dominio sfavorevole**, e che convenga restringere la ricerca intorno alle **aree più promettenti** già note.

Per affrontare questa situazione, si procede con un'operazione di **contrazione**: invece di muoversi verso l'esterno del simplesso (come accade nella riflessione o espansione), si cerca di **avvicinare il vertice peggiore agli altri**, nella speranza che all'interno del simplesso la funzione assuma valori migliori.

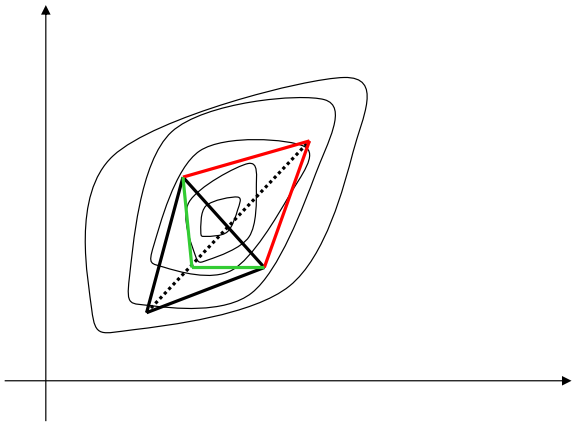
In altre parole, si costruisce un **nuovo punto** x_C (contratto) situato **tra il centroide** x_0 e il vertice peggiore x_H.

La posizione del punto contratto x_C è definita come:

$$x_C = (1-\beta)x_0 + \beta x_H = x_0 + \beta(x_H - x_0)$$

dove:

- x_0 è il centroide dei vertici migliori (escluso x_H),
- x_H è il vertice peggiore,
- $\beta \in (0,1)$ è il **coefficiente di contrazione**, tipicamente $\beta = 0.5$.



Il nuovo punto si colloca quindi **internamente al simplesso**, più vicino al centroide che al vertice peggiore.

- Se $f(x_C) < f(x_H)$: → la contrazione ha avuto successo → **si sostituisce** x_H **con** x_C nel simplesso.
- Se $f(x_C) \geq f(x_H)$: → la contrazione non ha prodotto miglioramenti → si esegue una **riduzione** (o "collasso") dell'intero simplesso verso il vertice migliore (strategia di emergenza per non rimanere bloccati).

Va sottolineato che la logica con cui vengono combinati **riflessione, espansione e contrazione** può variare. Esistono diverse implementazioni del Simplex di Nelder-Mead, a seconda delle scelte dello sviluppatore o dell'ambiente applicativo. Parametri come α, γ, β, e i criteri di accettazione dei nuovi punti possono essere **personalizzati** per migliorarne le prestazioni in contesti specifici.

5.12.1.4 Costruzione del Simplesso Iniziale

La creazione del simplesso iniziale è una fase fondamentale per l'efficacia dell'algoritmo Simplex (di Nelder-Mead). La sua qualità può influenzare in modo significativo la rapidità e la precisione della convergenza verso il minimo della funzione obiettivo.

5.12.1.5 Dipendenza dalla parametrizzazione

La modalità con cui si costruisce il simplesso iniziale dipende dal **tipo di parametrizzazione del problema**. In molte applicazioni ingegneristiche, soprattutto quando si lavora con modifiche a una **geometria di partenza** (es. ottimizzazione di forma), si parte da una configurazione iniziale per cui le variabili decisionali sono nulle.

In questi casi, si segue una regola semplice:

- Uno dei vertici del simplesso, chiamato solitamente $x^{(0)}$, è la **configurazione iniziale** (cioè il punto in cui tutte le variabili sono a zero).
    - Gli altri n vertici (in totale $n+1$ per un problema in $\mathbb{R}^n$) possono essere generati mediante un **DOE (Design of Experiments)** come:
    - **Reduced Factorial Design**
    - **Latin Hypercube Sampling**
    - **Sobol Sequences**
    - **Distribuzione casuale uniforme (Random)**

Questi metodi permettono di distribuire i punti nel dominio delle variabili in modo **statisticamente significativo** e di massimizzare l'informazione nella fase iniziale.

5.12.1.6 Costruzione di un simplesso iniziale

Un'alternativa molto usata è la **costruzione di un simplesso regolare**, cioè un simplesso in cui **tutti i lati hanno la stessa lunghezza** (equidistante), centrato nel punto iniziale $x^{(0)}$. Questa scelta permette di esplorare il dominio in modo simmetrico fin dall'inizio.

Siano:

- n: il numero di variabili decisionali (dimensione del problema),
- a: parametro positivo che regola la **dimensione del simplesso** (quanto è "grande"),
- $\mathbf{u}_i$: i versori lungo ciascuna direzione delle variabili,
- $x^{(0)}$: il centro iniziale del simplesso (punto di partenza).

Allora i vertici $x^{(i)}$, per $i = 1, \ldots, n$, si costruiscono nel seguente modo:

$$x^{(i)} = x^{(0)} + p \cdot \mathbf{u}_i + q \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{u}_j$$

dove i coefficienti p e q sono definiti da:

$$p = \frac{a}{2}\left(\sqrt{n+1} + n - 1\right)/n$$

$$q = \frac{a}{2}\left(\sqrt{n+1} - 1\right)/n$$

Questo schema garantisce la costruzione di un **simplesso equilatero** attorno al punto $x^{(0)}$. Ogni vertice è ottenuto spostandosi lungo una direzione principale $\mathbf{u}_i$, e contemporaneamente "bilanciando" lungo le altre direzioni. Il parametro a regola la "scala" dell'esplorazione iniziale: valori piccoli portano a un simplesso più compatto, valori grandi a un'esplorazione più ampia.

Questa costruzione può essere vista come un **DOE mirato all'algoritmo Simplex**, ottimizzato per fornire al metodo una **struttura iniziale bilanciata** e adatta a muoversi nel dominio della funzione in modo regolare e simmetrico.

La scelta del simplesso iniziale, che sia basata su DOE o su una costruzione regolare, è cruciale per il buon funzionamento dell'algoritmo Simplex. Deve essere coerente con:

- la scala delle variabili,
- la regolarità della funzione obiettivo,
- e la forma attesa della regione ottima.

In presenza di vincoli o geometrie complesse, si preferisce un inizializzazione basata su conoscenze pregresse o DOE specifici. In contesti più regolari e ben condizionati, un simplesso regolare centrato su $x^{(0)}$ risulta spesso efficace.

5.12.1.7 Convergenza dell'Algoritmo Simplex

L'algoritmo Simplex di Nelder-Mead è un metodo iterativo per l'ottimizzazione non vincolata di funzioni reali multivariate. A ogni passo, modifica la geometria di un **simplesso** (composto da $n+1$ punti in $\mathbb{R}^n$) allo scopo di avvicinarsi a un punto di minimo locale della funzione obiettivo.

Quando consideriamo l'algoritmo "in convergenza"?

Si considera che l'algoritmo abbia raggiunto la **convergenza** quando il simplesso si è **contratto significativamente**, ovvero quando **tutti i vertici del simplesso risultano molto vicini tra loro** e i valori della funzione nei vertici sono simili. In termini pratici, questo significa che il simplesso ha "collassato" attorno a un punto, che si assume come **punto di minimo trovato**.

All'inizio del processo, i punti del simplesso sono **distribuiti nello spazio** per esplorare il dominio della funzione. Durante le iterazioni, questi punti si muovono in base a trasformazioni geometriche (riflessione, espansione, contrazione, riduzione), e **gradualmente si accumulano attorno al minimo locale**.

5.12.1.8 Criterio numerico di convergenza

Per quantificare la dimensione del simplesso e valutare se è sufficientemente piccolo, si usa una **misura della dispersione dei valori della funzione** nei vertici:

$$Q = \sum_{i=1}^{n+1} \left| f(x^{(i)}) - f(x^{(0)}) \right| < \varepsilon$$

dove:

- $x^{(0)}$ è il **centroide** o il punto migliore tra i vertici del simplesso,
- $f(x^{(i)})$ è il valore della funzione al vertice i,
- ε è una **soglia di tolleranza**, scelta in base alla precisione desiderata.

Questa formula misura la **variazione dei valori della funzione obiettivo** nei vertici. Se tutti i valori sono vicini tra loro (cioè Q è piccolo), allora presumibilmente ci si trova vicini a un punto stazionario.

Inoltre, spesso si considera anche la **varianza spaziale dei vertici**, ovvero:

$$x^{(0)} = \frac{1}{n+1} \sum_{i=1}^{n+1} x^{(i)}$$

e si verifica se:

$$\max_{i=1,\dots,n+1} \| x^{(i)} - x^{(0)} \| < \delta$$

dove δ è un'altra soglia prefissata. Questo assicura che **i punti siano raggruppati anche geometricamente**, oltre che in termini di valore della funzione.

L'algoritmo Simplex **non utilizza derivate** e non necessita del gradiente della funzione. Questo lo rende **robusto** e particolarmente utile nei seguenti contesti:

- Quando la funzione obiettivo è **non derivabile**, **discontinua** o **rumorosa**;

- Quando il **numero di variabili è contenuto** (tipicamente $n < 10$);
- Quando la funzione è **complessa o costosa da valutare** e altri algoritmi (basati su gradiente) sono inapplicabili.

Tuttavia, questa robustezza viene pagata in termini di **accuratezza**:

- Il metodo tende a **convergere lentamente**;
- Può **ristagnare in minimi locali**;
- È **sensibile alla scelta del simplesso iniziale**;
- Non è adatto a **problemi di alta dimensionalità**, poiché il numero di vertici cresce linearmente con n, e le trasformazioni geometriche diventano meno efficaci.

Va sottolineato che il Simplex di Nelder-Mead è progettato per **ottimizzazione non vincolata**. Se sono presenti **vincoli sulle variabili**, questi vengono gestiti indirettamente tramite l'uso di **funzioni di penalità** (*Penalty Functions*), che modificano la funzione obiettivo aggiungendo una penalizzazione alle soluzioni non ammissibili:

$$f_{\text{penalizzata}}(x) = f(x) + P(x)$$

dove $P(x)$ è una funzione che assume valori elevati quando x viola uno o più vincoli. In questo modo, il minimo della funzione penalizzata tende a corrispondere a una soluzione ammissibile del problema originale.

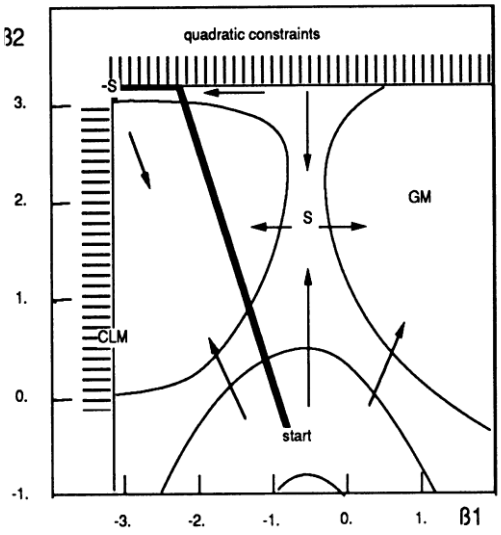
5.12.1.9 Vantaggi dell'Algoritmo Simplex

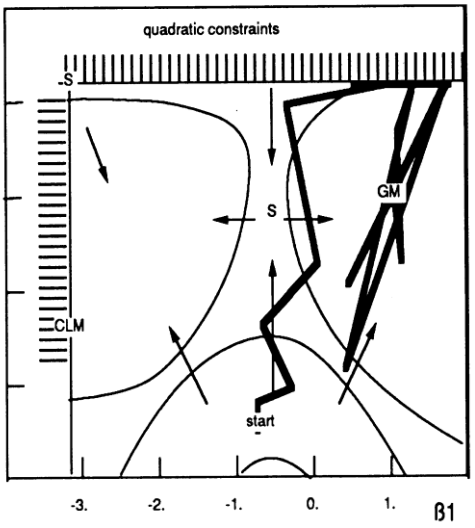
L'algoritmo Simplex di Nelder-Mead presenta una serie di vantaggi significativi che lo rendono particolarmente adatto in contesti specifici dell'ottimizzazione numerica. Questi vantaggi derivano principalmente dalla **natura geometrica e derivate-free** del metodo, cioè dal fatto che non richiede il calcolo esplicito del gradiente della funzione.

A differenza degli algoritmi basati sul gradiente, che studiano il **comportamento locale** della funzione (ossia come varia in un intorno infinitesimo di un punto), l'algoritmo Simplex utilizza una strategia **geometrica** che analizza **simultaneamente più punti** del dominio.

Implicazione pratica:

- Ogni iterazione del Simplex valuta la funzione in tutti i $n + 1$ vertici del simplesso, quindi possiede una **conoscenza più ampia del dominio** rispetto a un algoritmo che si basa solo su una direzione di discesa.
- Questa caratteristica aumenta la **probabilità di sfuggire ai minimi locali** e trovare un **ottimo globale**, soprattutto in funzioni multimodali (con più massimi e minimi).





Consideriamo una funzione in due dimensioni con due massimi:

- Un **massimo relativo** a sinistra,
- Un **massimo assoluto** a destra.

Se si parte da un punto vicino al massimo relativo, un algoritmo molto efficiente come **BFGS** (quasi-Newton), che sfrutta il gradiente per cercare la direzione di miglioramento, **convergerà rapidamente**... ma al **massimo relativo**. Questo accade perché il metodo **non ha informazioni globali**: esplora solo la direzione di discesa locale.

Invece, l'algoritmo **Simplex**, operando con trasformazioni geometriche (riflessioni, espansioni, contrazioni) su più punti distribuiti, **esegue movimenti più ampi e meno regolari**. Questo comporta un'esplorazione **meno efficiente localmente**, ma più **robusta globalmente**, con maggiore possibilità di trovare il **massimo assoluto**.

L'algoritmo Simplex non richiede il calcolo né del **gradiente** né della **matrice hessiana**, e nemmeno la funzione deve essere continua o differenziabile. È quindi adatto a:

- Funzioni **discontinue**,
- Funzioni **non lisce**,
- Funzioni **rumorose** o **simulate** (tipiche nelle simulazioni numeriche o nei problemi di ingegneria computazionale).

Questa caratteristica lo rende estremamente **robusto** in contesti dove metodi classici (come BFGS o Newton-Raphson) falliscono o non sono applicabili.

Un ulteriore vantaggio del Simplex è la sua **flessibilità rispetto al tipo di variabili**:

- È **naturale e efficace** con variabili **continue** (che assumono valori reali);
- Può essere adattato anche a **variabili discrete** (intere), purché l'operazione di riflessione/espansione/contrazione abbia ancora senso numerico e geometrico.

L'algoritmo non è adatto a problemi con **variabili categoriche** (es. colore = rosso/giallo/blu), perché:

- Le trasformazioni geometriche su cui si basa (riflessione, espansione, contrazione) **presuppongono una metrica**, cioè il concetto di distanza tra punti.
- In uno spazio categoriale, **la distanza tra categorie non è definita**, per cui il metodo non può "muoversi" correttamente.

In conclusione l'algoritmo Simplex è un metodo:

- **robusto**,
- **intuitivo**,
- **non derivativo**,
- e **versatile** per problemi di piccola o media dimensione.

È la scelta ideale **quando la funzione è difficile, discontinua, costosa o sconosciuta**, e si cerca una soluzione **affidabile** anche senza precisione millimetrica. Tuttavia, richiede **più valutazioni di funzione**, quindi può essere **computazionalmente costoso**, specialmente in alta dimensione.

5.12.2 Simulated Annealing

L'algoritmo **Simulated Annealing (SA)** è uno dei metodi di ottimizzazione stocastica più noti per la sua **estrema robustezza**, ossia la capacità di evitare i minimi locali e individuare soluzioni globali. Sebbene non sia largamente impiegato nell'ingegneria applicata a causa di alcuni svantaggi pratici, rappresenta un riferimento fondamentale nella storia dell'ottimizzazione.

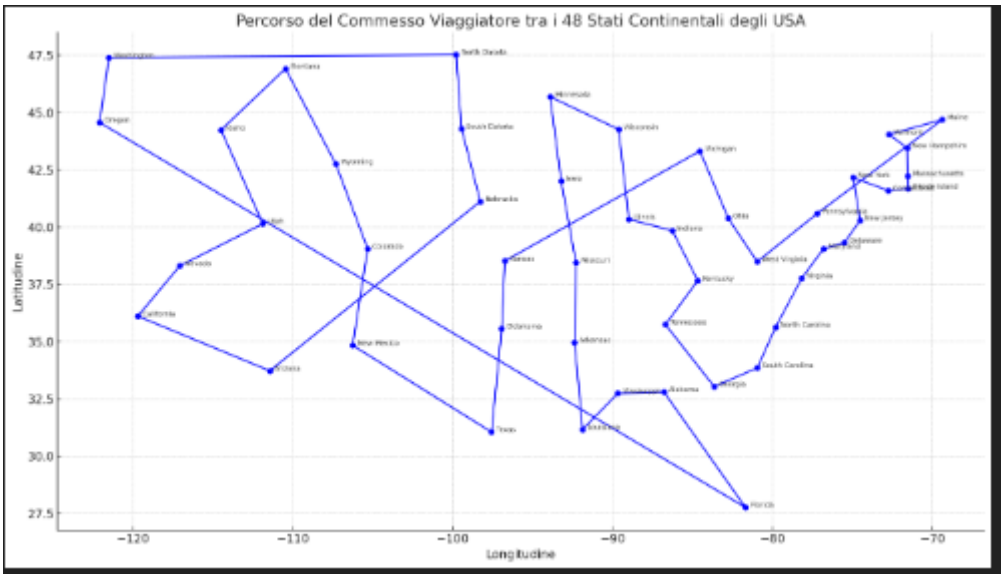
Caratteristiche principali

- **Robustezza elevatissima**: è tra gli algoritmi con **maggiore probabilità di identificare il massimo (o minimo) assoluto** della funzione obiettivo.
- **Applicabilità**: può essere utilizzato sia con **variabili continue**, sia con **variabili discrete o categoriali**.
- **Flessibilità**: non richiede il calcolo del gradiente e può quindi essere applicato anche a funzioni non derivabili o definite su domini non continui.

Limiti principali

- **Scarsa accuratezza** nella convergenza verso il massimo (o minimo) preciso.
- **Alto costo computazionale**: richiede numerose iterazioni e valutazioni della funzione per ottenere buoni risultati.

Simulated Annealing è storicamente rilevante perché fu **il primo algoritmo in grado di risolvere efficacemente il problema del commesso viaggiatore (TSP – Travelling Salesman Problem)**, un classico problema combinatorio che consiste nel trovare il percorso più breve che visita una serie di punti (città) una sola volta.



L'algoritmo trae ispirazione dai processi fisici di **ricottura metallurgica**, dove un materiale viene portato ad alta temperatura e poi raffreddato lentamente per raggiungere una configurazione stabile a bassa energia.

Nel contesto dell'ottimizzazione, la funzione obiettivo da massimizzare è:

$$\max f(x), \quad x \in \mathbb{R}^n$$

A ogni iterazione, si parte da un punto x_i e si genera un nuovo punto x_{i+1} mediante una **perturbazione casuale**. Si valutano le rispettive immagini $f(x_i)$ e $f(x_{i+1})$. A questo punto si presentano due casi:

- **Miglioramento**: se $f(x_{i+1}) > f(x_i)$, il nuovo punto viene **sempre accettato**.
- **Peggioramento**: se $f(x_{i+1}) < f(x_i)$, il nuovo punto viene **accettato con una certa probabilità**, definita dal **criterio di Metropolis**.

$$\begin{aligned}
&\bar{x}_0 \rightarrow f(\bar{x}_0) \\
&\downarrow \\
&\bar{x}_i \rightarrow f(\bar{x}_i) \\
&se \quad f(\bar{x}_i) < f(\bar{x}_{i-1}) \Rightarrow \text{sostituisco } \bar{x}_{i-1} \; con \; \bar{x}_i \\
&se \quad f(\bar{x}_i) > f(\bar{x}_{i-1}) \Rightarrow \text{applico Metropolis}
\end{aligned}$$

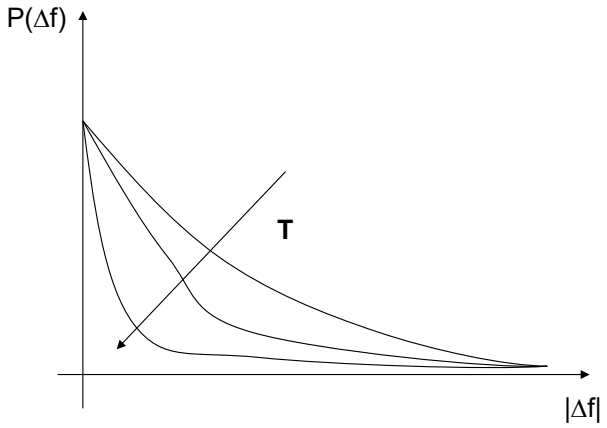
5.12.2.1 Criterio di Metropolis

La probabilità di accettare una soluzione peggiorativa è data da:

$$p(\Delta f) = e^{-\Delta f/(KT)}$$

dove:

- $\Delta f = f(x_i) - f(x_{i+1})$ rappresenta la perdita in qualità della soluzione;
- T è la **temperatura**, parametro che controlla la probabilità di accettazione;
- K è una costante positiva.



Per implementare il criterio:

- si genera un numero casuale $r \in (0,1)$;
- se $r < p(\Delta f)$, la nuova soluzione peggiorativa viene **accettata**;
- altrimenti, viene **rifiutata**.

Questo meccanismo consente all'algoritmo di **esplorare anche regioni sub-ottimali**, con lo scopo di **sfuggire ai minimi locali** nelle fasi iniziali della ricerca.

Il parametro T, detto **temperatura**, decresce progressivamente con il numero di iterazioni. All'inizio, T è alta, quindi anche peggioramenti significativi hanno una discreta probabilità di essere accettati. Con il tempo, T diminuisce, rendendo l'algoritmo sempre più **selettivo**.

Questa strategia consente:

- **esplorazione globale** nelle prime fasi;
- **raffinamento locale** nelle fasi finali.

Il **Simulated Annealing** è un algoritmo stocastico estremamente robusto, adatto a problemi di ottimizzazione globali e non differenziabili, anche con variabili discrete. La sua capacità di accettare soluzioni peggiorative lo rende ideale per **esplorare ampi spazi di soluzione**, anche se ciò avviene a discapito dell'efficienza e della precisione.

5.13 L'Algoritmo Genetico (Genetic Algorithm)

L'algoritmo genetico è una delle tecniche più note e utilizzate nell'ambito dell'**ottimizzazione stocastica**. Nato ispirandosi ai meccanismi dell'evoluzione biologica darwiniana, si è rapidamente affermato grazie alla sua **estrema versatilità e robustezza**. Esistono migliaia di versioni differenti dell'algoritmo genetico, ognuna adattata a specifici contesti applicativi, ma tutte basate su una struttura di base comune.

Caratteristiche principali

L'algoritmo genetico si è diffuso e ha avuto sempre più successo per diverse ragioni:

- **Robustezza**: È uno degli algoritmi più robusti nel panorama delle tecniche di ottimizzazione, anche se in genere leggermente meno robusto del Simulated Annealing. Può affrontare problemi complessi e altamente non lineari senza richiedere particolari ipotesi sulla funzione obiettivo.
- **Flessibilità** nella gestione delle variabili: non pone vincoli stringenti sulla **tipologia delle variabili** (possono essere continue, discrete, categoriche, binarie) né sulla natura della funzione obiettivo, che può essere **continua o discontinua**.
- **Pionieristico nel multi-obiettivo**: È stato uno dei primi algoritmi a essere utilizzato in problemi di **ottimizzazione multiobiettivo**, ossia problemi in cui si vogliono ottimizzare simultaneamente più criteri contrastanti.
- **Implementazione semplice**: L'algoritmo può essere implementato in modo molto semplice, anche con poche righe di codice. Al tempo stesso, esistono versioni avanzate e creative con operatori evoluti e meccanismi adattivi.
- **Paralellizzabilità**: Il fatto che gli individui (cioè le soluzioni candidate) possano essere valutati indipendentemente rende l'algoritmo **facilmente parallelizzabile**, caratteristica preziosa per sfruttare l'hardware moderno.
- **Efficienza su problemi complessi**: Si dimostra particolarmente **efficace quando il numero di variabili è elevato** e i vincoli del problema sono numerosi o complessi da trattare in forma esplicita.

Nonostante i molti vantaggi, l'algoritmo genetico presenta anche alcune **criticità**:

- **Bassa accuratezza**: Non possiede un vero e proprio **criterio di convergenza**. Spesso non garantisce il raggiungimento di un ottimo preciso, ma si limita ad avvicinarsi ad aree "buone" del dominio. Per questo motivo, viene talvolta **affiancato a metodi deterministici** in una struttura ibrida (ad esempio, algoritmi genetici seguiti da un metodo locale come il gradient descent).
- **Scelte arbitrarie nei parametri**: L'algoritmo richiede di scegliere **parametri liberi**, come il numero di individui nella popolazione e il numero di generazioni. Tipicamente, per una funzione con circa 10 variabili, si impiega una **popolazione di 40-50 individui** e non si lascia evolvere l'algoritmo per più di **8-10 generazioni**.
- **Fraintendimenti comuni**: Non è vero che l'algoritmo genetico sia **l'unico strumento per trattare problemi multiobiettivo**. Qualsiasi algoritmo di ottimizzazione, grazie alla Teoria dei Giochi, può essere adattato a questo scopo. Tuttavia, per ragioni storiche e pratiche, circa il **95% delle applicazioni multiobiettivo** utilizza ancora l'approccio genetico.

5.13.1 Il concetto di individuo

Tutto nell'algoritmo genetico ruota attorno al concetto di **individuo**, che rappresenta una **configurazione** del problema. Dato un problema di ottimizzazione:

$$f: \mathbb{R}^n \to \mathbb{R}$$

un individuo è una particolare combinazione di variabili $x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}$, a cui corrisponde un valore della funzione obiettivo $f(x_1^{(i)}, \ldots, x_n^{(i)})$.

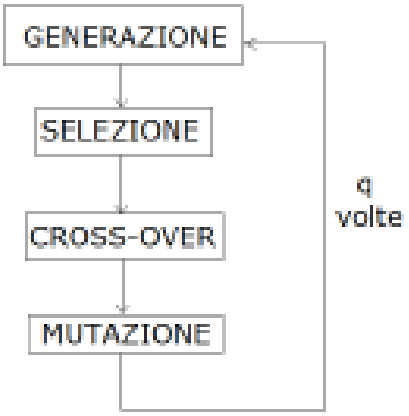
I **vincoli** del problema non sono gestiti in modo diretto ma vengono integrati nel processo di valutazione mediante l'uso di **funzioni penalità** (*penalty functions*), che aumentano artificialmente il valore della funzione obiettivo per le soluzioni che violano i vincoli, disincentivandone così la selezione.

5.13.2 Funzionamento dell'algoritmo

L'algoritmo segue una struttura iterativa, che simula il processo di evoluzione naturale. Si parte dalla **generazione iniziale**, creata in genere con un **Design of Experiment (DOE)** o casualmente, composta da p individui.

Ogni generazione subisce una serie di **operatori genetici**:

- **Selezione**: Vengono selezionati gli individui migliori in base al loro valore di fitness (funzione obiettivo).
- **Cross-over** (ricombinazione): Gli individui selezionati vengono "incrociati" per generare nuova prole, combinando le caratteristiche dei genitori.
- **Mutazione**: Alcuni individui subiscono variazioni casuali, per introdurre diversità genetica.
- **Sostituzione**: I nuovi individui rimpiazzano parte o tutta la popolazione precedente.



Questo ciclo si ripete per q generazioni. In totale, il numero di configurazioni esplorate durante l'intero processo è:

$$\text{n° configurazioni} = p \cdot q$$

L'algoritmo genetico è uno strumento potente e flessibile per l'ottimizzazione, specialmente in contesti complessi dove non è possibile (o conveniente) utilizzare metodi deterministici. La sua forza risiede nella **capacità di esplorare ampi domini di ricerca** in modo robusto, ma è fondamentale essere consapevoli dei suoi limiti in termini di precisione e controllo della convergenza.

Per migliorarne le prestazioni, è spesso utile **combinarlo con altri algoritmi** (ibridazione) o affiancarlo con meccanismi di analisi statistica che aiutino a comprendere meglio il comportamento della popolazione nel tempo.

5.13.3 Selezione

Nel contesto degli **algoritmi genetici**, l'**operatore di selezione** è uno dei meccanismi centrali del processo evolutivo. Il suo compito è quello di **scegliere, all'interno della popolazione corrente, gli individui che contribuiranno a generare la nuova generazione**, attraverso gli operatori di crossover e mutazione.

L'idea intuitiva sarebbe quella di **selezionare semplicemente gli individui con i valori migliori della funzione obiettivo**. Nel caso di un problema di **massimizzazione**, questo significherebbe scegliere le soluzioni con il valore più alto della funzione $f(x)$.

Tuttavia, **questo approccio ingenuo porta a un problema ben noto negli algoritmi genetici: la convergenza prematura**. Se si selezionano sempre e solo gli individui "migliori" (localmente), il rischio è che l'intera popolazione diventi rapidamente omogenea e che l'algoritmo converga verso un **ottimo locale**, perdendo la capacità di esplorare altre regioni del dominio potenzialmente più promettenti.

Questo comportamento limita gravemente la **robustezza** dell'algoritmo. Per evitarlo, è fondamentale introdurre una **componente stocastica** nella selezione: occorre cioè **lasciare una certa probabilità anche agli individui meno performanti** di essere selezionati. Questa strategia permette di **preservare la diversità genetica** all'interno della popolazione e di mantenere attiva l'esplorazione dell'intero spazio delle soluzioni.

Un'analogia utile è quella con l'**algoritmo Simulated Annealing**: anche lì si accettano transitoriamente soluzioni peggiori per evitare di rimanere intrappolati in minimi locali.

Nel corso degli anni sono stati sviluppati **numerosi operatori di selezione** (ce ne sono letteralmente centinaia), ciascuno con caratteristiche proprie.

L'operatore di selezione è fondamentale per il corretto funzionamento dell'algoritmo genetico. Deve **bilanciare due forze contrapposte**:

- L'**intensificazione**, cioè la pressione verso le soluzioni migliori;
- La **diversificazione**, ovvero la capacità di esplorare regioni nuove del dominio.

Una buona selezione consente all'algoritmo di **migliorare progressivamente** la qualità della popolazione senza sacrificare la **capacità esplorativa**. La scelta dell'operatore di selezione (e dei suoi parametri) incide fortemente sull'efficienza dell'algoritmo e deve essere fatta con attenzione in base al tipo di problema affrontato.

5.13.3.1 Selezione a roulette (Roulette Wheel Selection)

Questo metodo prende il nome dalla **roulette da casinò**, ed è una tecnica che assegna a ogni individuo una **probabilità di essere selezionato proporzionale al suo valore di fitness**. È utile nei casi in cui si vuole privilegiare gli individui migliori, ma senza escludere completamente quelli peggiori.

Esempio:

Supponiamo di voler massimizzare una funzione $f: \mathbb{R}^n \to \mathbb{R}$ e di avere una popolazione di 5 individui, a cui sono associati i seguenti valori della funzione obiettivo:

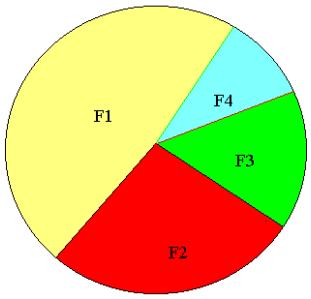
Individuo	$f(x)$
1	2.0
2	1.5
3	4.0
4	6.0
5	1.0

Si calcola quindi il totale dei valori:

$$\text{Somma totale} = 2.0 + 1.5 + 4.0 + 6.0 + 1.0 = 14.5$$

Si associa a ciascun individuo una **probabilità di selezione** pari al suo valore diviso per la somma totale:

Individuo	Probabilità
1	0.138
2	0.103
3	0.276
4	0.414
5	0.069



F1 > F2 > F3 > F4

Queste probabilità possono essere **rappresentate graficamente come settori di un grafico a torta**. Si estrae poi un numero casuale tra 0 e 1 (ad esempio, 0.8), e si seleziona l'individuo corrispondente a quel settore.

In questo caso, la **probabilità di selezionare l'individuo 4** (che ha il valore di fitness più alto) è la più elevata, ma non è l'unica possibilità. Anche individui come il 3 o addirittura il 5 possono essere selezionati, anche se con probabilità minori.

Limiti della selezione a roulette

Sebbene sia un metodo efficace, la selezione a roulette presenta alcuni **limiti**, specialmente quando la **popolazione è molto ampia** o quando i valori della funzione obiettivo sono **fortemente sbilanciati**. In questi casi:

- Gli individui con valori molto alti dominano la selezione.
- Gli individui con fitness medio o basso hanno probabilità quasi nulle di essere scelti.
- Si rischia di tornare al problema della **convergenza prematura**, annullando la diversità genetica.

5.13.3.2 Alternative alla roulette

Per contrastare questi problemi, esistono molte varianti e metodi alternativi di selezione, tra cui:

- **Tournament selection**: si scelgono a caso piccoli gruppi di individui e si seleziona il migliore all'interno di ciascun gruppo.
- **Rank selection**: si ordina la popolazione in base al valore di fitness e si assegna la probabilità di selezione in base al **rango**, non al valore assoluto.
- **Stochastic Universal Sampling (SUS)**: una generalizzazione della roulette che cerca di rendere la selezione più uniforme e meno casuale.
- **Elitism**: una tecnica spesso combinata con le altre, che **garantisce il passaggio automatico** alla generazione successiva di una parte (di solito piccola) dei migliori individui.

5.13.3.3 Selezione Locale

La **selezione locale** rappresenta un'evoluzione sofisticata del concetto classico di selezione negli algoritmi genetici. Nasce dall'osservazione che, per mantenere elevata la diversità genetica e aumentare la probabilità di trovare l'**ottimo globale**, può essere utile **suddividere la popolazione in sottogruppi** parzialmente isolati, che evolvono indipendentemente l'uno dall'altro per un certo numero di generazioni.

Questa idea ha portato allo sviluppo del cosiddetto **algoritmo genetico ad isole** (*Island Model Genetic Algorithm*), in cui:

- La popolazione complessiva viene suddivisa in più **sottopopolazioni** (dette isole), che evolvono separatamente;
- Periodicamente, alcuni individui vengono **scambiati tra isole** (migrazione), permettendo così la **condivisione del patrimonio genetico** tra regioni diverse del dominio.

Questa strategia è particolarmente efficace in ambienti paralleli o distribuiti, poiché consente l'esecuzione simultanea di più processi evolutivi, riducendo il rischio di **convergenza prematura** e migliorando l'efficienza dell'algoritmo.

Sebbene il modello ad isole sia potente, la sua **implementazione può risultare complessa**, soprattutto in contesti sequenziali o con risorse computazionali limitate. Per questo motivo è stato introdotto un

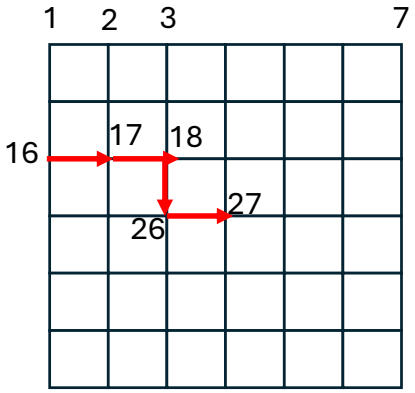
approccio semplificato: **riprodurre il concetto di isola all'interno di un'unica generazione**, tramite la **selezione locale**.

Si parte da una popolazione di p individui e li si **distribuisce su una griglia bidimensionale**, ad esempio una matrice di dimensioni $\sqrt{p} \times \sqrt{p}$. Ogni individuo occupa una **posizione fissa sulla griglia**, come se fosse in un reticolo.

Quando si deve effettuare una **selezione per il crossover**, non si cerca un partner nell'intera popolazione, ma **in una zona locale attorno a un individuo prescelto**. Questo processo genera naturalmente la formazione di **microisole** all'interno della stessa generazione: gruppi di individui vicini che interagiscono più frequentemente tra loro.

Supponiamo, ad esempio, di voler selezionare un partner per l'individuo in posizione 15. Si effettua una **"camminata locale"** a partire dalla sua posizione, muovendosi in modo casuale tra le celle vicine della griglia. Durante questa esplorazione, si visitano alcuni individui — ad esempio quelli nelle posizioni 13, 14, 20 e 21 — e si seleziona **il migliore tra questi** (in base alla fitness) come partner per il crossover.

Questa camminata può seguire una distribuzione casuale controllata, ad esempio scegliendo un numero random in intervalli definiti (es. [0, 0.25), [0.25, 0.5), ecc.) per determinare la direzione di spostamento (su, giù, destra, sinistra).



Vantaggi della selezione locale

- **Mantiene la diversità genetica** più a lungo, ritardando la convergenza e migliorando l'esplorazione.
- È un'**ottima approssimazione del modello ad isole**, ma più semplice da implementare.
- È particolarmente utile quando la funzione obiettivo ha **molti ottimi locali**.

Griglia toroidale

Per evitare problemi di **confine della griglia**, si assume che la griglia sia **toroidale**: un modello chiuso in cui i bordi "si toccano". In altre parole, la griglia non ha spigoli o margini: chi si sposta a sinistra dell'ultima colonna finisce nella prima, e lo stesso vale per righe e colonne.

Esempio:

Se mi trovo sull'individuo in posizione 13 e, durante la camminata, viene estratto un valore random che indica un movimento verso sinistra, potrei finire sull'individuo in posizione 18 (perché la griglia si avvolge su sé stessa).

Effetti sulla probabilità di selezione

Questo schema **abbassa la probabilità di incrocio tra individui molto distanti** sulla griglia, come ad esempio l'individuo 8 e l'individuo 29. Di conseguenza, all'interno della popolazione si formano delle **sottopopolazioni spontanee** che evolvono semi-indipendentemente: vere e proprie **microisole**.

La selezione locale è una tecnica semplice ma potente, che simula l'evoluzione indipendente di gruppi di individui senza la necessità di gestire strutture complesse come nel modello ad isole classico. Grazie alla sua capacità di **preservare la diversità** e di **favorire un'esplorazione più strutturata dello spazio delle soluzioni**, rappresenta una strategia efficace in molti problemi di ottimizzazione.

5.13.4 Crossover

Una volta selezionati gli individui più promettenti all'interno della popolazione, l'obiettivo successivo è **generare nuovi individui** che, idealmente, presentino caratteristiche migliori o comunque diverse, al fine di **esplorare efficacemente lo spazio delle soluzioni**.

Per farlo, si utilizza l'**operatore di crossover**, anche detto **ricombinazione**. Questo operatore prende in ingresso due (o più) individui, selezionati secondo una certa strategia, e **li combina per creare uno o più nuovi individui**, detti figli. L'idea è ispirata al **meccanismo di riproduzione sessuata** degli esseri viventi, dove le caratteristiche genetiche dei genitori vengono mescolate per produrre la prole.

Il crossover non si preoccupa direttamente del **valore della funzione obiettivo** associato agli individui. Lavora infatti sul **genotipo** (cioè sulla rappresentazione interna dell'individuo), combinando blocchi di informazione per generare nuove configurazioni.

5.13.4.1 Crossover binario classico

Il tipo più semplice e tradizionale di crossover è il **crossover binario**, così chiamato perché richiede che le **variabili dell'individuo siano codificate in forma binaria**.

Codifica degli individui

Supponiamo di avere due individui, ciascuno rappresentato da n variabili:

$$\text{Individuo 1} \rightarrow (x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)})$$

$$\text{Individuo 2} \rightarrow (x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)})$$

Le variabili possono essere **numeri reali o interi**, ma per applicare il crossover binario, si procede prima con la **codifica binaria** di ciascuna variabile. Il risultato è una **stringa binaria** per ogni individuo:

$$\text{Individuo 1} \rightarrow 010011100111010$$

$$\text{Individuo 2} \rightarrow 110111001001001$$

Crossover a un punto

Nel **crossover a un punto**, si sceglie in modo casuale un **punto di rottura** lungo la stringa binaria. Le due stringhe vengono poi tagliate in quel punto e **scambiate tra loro**:

$$\text{Individuo 1: } 0100111|00111010$$

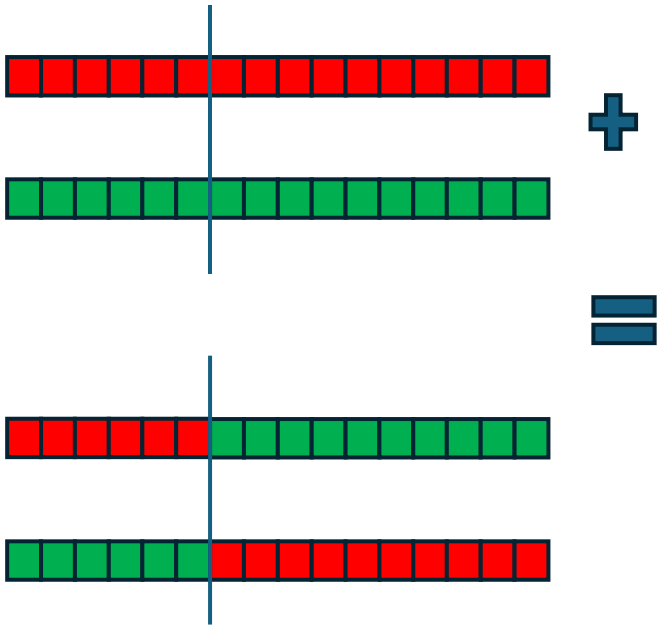
$$\text{Individuo 2: } 1101110|01001001$$

Incrociando le due metà si ottengono i nuovi individui:

$$\text{Nuovo individuo 1: } 0100111\mathbf{01001001}$$

Nuovo individuo 2: 1101110**00111010**

In questo modo, si generano due **figli** che combinano porzioni del patrimonio genetico dei genitori.



Vantaggi del crossover binario

- **Semplicità di implementazione**: è facile da codificare e computazionalmente leggero.
- **Indipendenza dalla natura delle variabili**: può trattare **variabili misurabili, discrete o categoriali**, dato che lavora solo a livello di rappresentazione binaria, senza concetti di distanza o continuità.
- **Trasferimento delle caratteristiche**: permette di **trasmettere ai figli i blocchi di variabili** (i geni) potenzialmente "buoni" ereditati dai genitori, facilitando la propagazione di soluzioni promettenti.

Il metodo, sebbene semplice, presenta alcune limitazioni:

- **Rischio di convergenza lenta**: se il numero di variabili è elevato, è probabile che il punto di crossover cada **in zone poco influenti**, e quindi che il mescolamento non produca nuove combinazioni significative.
- **Limitata variazione genetica**: con un solo punto di rottura, la **quantità di informazione realmente rimescolata è bassa**, soprattutto in lunghe stringhe binarie.

5.13.4.2 Crossover a due punti

Per superare questi limiti si utilizza una variante più efficace: il **crossover a due punti**. In questo caso si scelgono **due punti di rottura**, e si scambia **la porzione centrale** tra i due individui.

Esempio:

Individuo 1: 0100111|00111|010

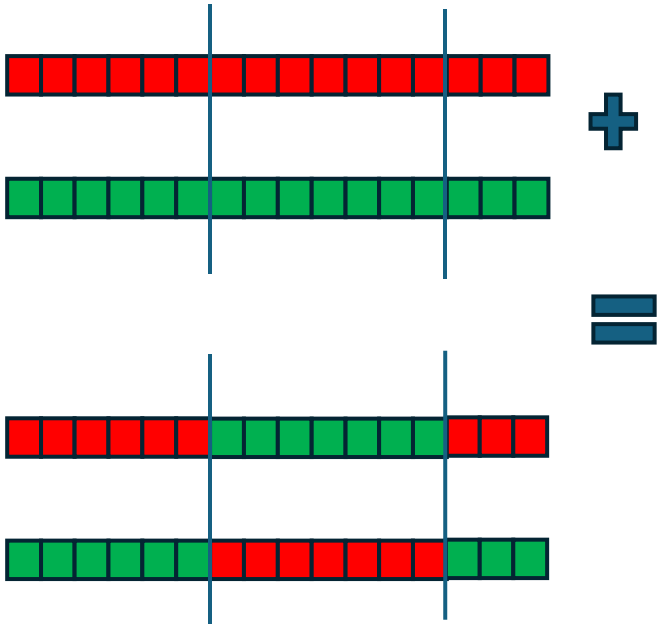
Individuo 2: 1101110|01001|001

Dopo lo scambio:

Nuovo individuo 1: 0100111**01001**010

Nuovo individuo 2: 1101110**00111**001

In questo modo, **si aumentano le possibilità di combinare sezioni significative** di genoma, generando maggiore diversità e accelerando il processo evolutivo.



Il crossover è un operatore chiave nell'algoritmo genetico: serve a **generare nuova informazione**, combinando quella esistente in modo controllato ma casuale. Il **crossover binario** è solo uno dei tanti approcci possibili (esistono anche crossover aritmetici, uniformi, per permutazioni, ecc.), ma rimane un ottimo punto di partenza per comprendere il meccanismo base.

La **scelta del tipo di crossover** e del **numero di punti di rottura** ha un impatto diretto su:

- la **diversità genetica** della popolazione,
- la **velocità di convergenza**,
- la **robustezza dell'ottimizzazione**.

In sintesi, il crossover è ciò che consente all'algoritmo genetico di "esplorare" lo spazio delle soluzioni, aprendosi a combinazioni nuove senza perdere i tratti vantaggiosi già scoperti.

5.13.4.3 Crossover Direzionale

Negli algoritmi genetici classici, il **crossover** si limita a combinare le informazioni genetiche di due individui selezionati senza tener conto del **valore della funzione obiettivo**. Tuttavia, in molti problemi complessi, è utile **guidare la generazione dei nuovi individui** verso le **regioni più promettenti del dominio**, cioè quelle in cui la funzione obiettivo tende ad aumentare.

Per questo motivo è stato introdotto il **crossover direzionale**, un operatore evoluto che tiene conto della **direzione di incremento** della funzione durante la fase di crossover. In altre parole, **non si combina l'informazione a caso**, ma si cerca di **muoversi nella direzione che migliora la soluzione**, rendendo l'algoritmo più **veloce e preciso**.

L'idea centrale è di **individuare una direzione nel dominio delle variabili lungo la quale il valore della funzione aumenta**. Una volta nota questa direzione, si possono generare nuovi individui che **esplorano intenzionalmente quella zona del dominio**, nella speranza di trovare valori ancora migliori.

Supponiamo di voler **massimizzare** una funzione:

$$\max f: \mathbb{R}^n \to \mathbb{R}$$

Si scelgono **tre individui** della popolazione: 1, 2 e 3. A ciascun individuo è associato il valore della funzione obiettivo:

$$f_1 = f(x^{(1)}), \quad f_2 = f(x^{(2)}), \quad f_3 = f(x^{(3)})$$

e supponiamo che valga:

$$f_2 > f_1, \quad f_3 > f_1$$

Questo suggerisce che passando da $x^{(1)}$ verso $x^{(2)}$ o $x^{(3)}$, il valore della funzione cresce: siamo quindi in una zona "buona", cioè una regione del dominio dove è probabile che si trovi l'ottimo globale.

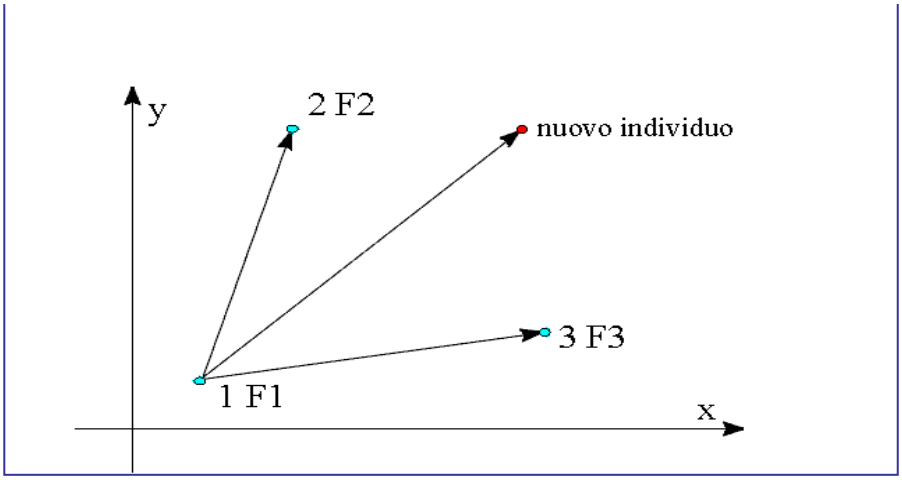
Per generare un nuovo individuo $x^{(\text{nuovo})}$, si utilizza una **combinazione vettoriale direzionata** dei tre individui:

$$x_i^{(\text{nuovo})} = x_i^{(1)} + s \cdot \text{segno}(f_2 - f_1) \cdot (x_i^{(2)} - x_i^{(1)}) + t \cdot \text{segno}(f_3 - f_1) \cdot (x_i^{(3)} - x_i^{(1)})$$

dove:

- s e t sono **numeri casuali** estratti uniformemente in $[0,1]$, che introducono **variazione stocastica**;
- $\text{segno}(f_j - f_1)$ restituisce +1 se $f_j > f_1$, −1 se $f_j < f_1$, e 0 se sono uguali;
- $x_i^{(k)}$ indica la variabile i-esima dell'individuo k.

In sostanza, **si parte dall'individuo peggiore** ($x^{(1)}$) e ci si sposta in una **combinazione casuale delle direzioni** verso i due individui migliori, sfruttando l'informazione contenuta nei loro valori di fitness.



Il crossover direzionale può essere visualizzato **come un movimento controllato nello spazio delle variabili**, che parte da un punto di riferimento meno promettente e si spinge verso regioni dove è stata osservata una crescita della funzione. Questa strategia introduce **una forma di gradiente implicito**, anche in problemi dove il gradiente non è noto o non è calcolabile.

Vantaggi

- **Maggiore velocità di convergenza**: la direzionalità accelera il raggiungimento di zone ad alta fitness, riducendo i tempi necessari a trovare buone soluzioni.
- **Migliore accuratezza**: guidando l'evoluzione in modo intelligente, si riduce la probabilità di convergere su ottimi locali sbagliati.
- **Robustezza migliorata**: l'informazione sul valore della funzione aiuta l'algoritmo a bilanciare meglio esplorazione ed esploitation.

Svantaggi

- **Limitazione sulle variabili**: il crossover direzionale **presuppone che le variabili siano quantitative** (continue o discrete), perché il calcolo delle **distanze** e delle **direzioni** nello spazio delle soluzioni non è definito per **variabili categoriali** (es. "rosso", "verde", "blu").
- **Parametrizzazione più rigida**: richiede una rappresentazione vettoriale coerente del dominio, cosa che non è sempre garantita nei problemi reali.

Il crossover direzionale è particolarmente utile quando:

- il dominio della funzione è continuo e ben definito;
- si vuole ottenere **una convergenza più rapida** rispetto a quella offerta da operatori casuali come il crossover binario classico;
- il problema presenta **zone "interessanti" ben distribuite**, che possono essere individuate con poche valutazioni relative.

Il crossover direzionale rappresenta un'evoluzione intelligente degli operatori classici, che consente agli algoritmi genetici di **"guidare" l'evoluzione verso le aree più promettenti dello spazio delle soluzioni**. A differenza del crossover binario, che mescola l'informazione senza alcuna logica relativa alla funzione

obiettivo, il crossover direzionale **utilizza attivamente l'informazione sulla fitness** per migliorare la generazione dei nuovi individui.

Tuttavia, la sua applicabilità dipende dalla natura del problema: **non è adatto in presenza di variabili categoriali**, e richiede una **rappresentazione coerente delle configurazioni** nello spazio $\mathbb{R}^n$.

5.13.5 Mutazione

L'**operatore di mutazione** è uno degli elementi fondamentali di ogni algoritmo genetico. Anche se nelle implementazioni moderne ha assunto un **ruolo meno centrale** rispetto ad altri operatori (come crossover o selezione), rimane comunque **indispensabile** per garantire la **diversità genetica** all'interno della popolazione e per **evitare la convergenza prematura** verso soluzioni subottimali.

La mutazione ha lo scopo di **introdurre casualmente nuove informazioni nel patrimonio genetico della popolazione**. È una forma di perturbazione stocastica che consente all'algoritmo di:

- **Esplorare regioni del dominio non ancora visitate**;
- **Uscire da ottimi locali** in cui la popolazione rischia di rimanere bloccata;
- **Evitare la stagnazione evolutiva**, ossia la progressiva uniformità genetica che riduce l'efficacia del crossover.

In altre parole, la mutazione agisce come un **meccanismo di esplorazione**, mentre il crossover è un meccanismo di **combinazione e sfruttamento** delle informazioni già note.

Di norma, la mutazione viene applicata a **una piccola percentuale della popolazione** a ogni generazione — tipicamente intorno al **5%**. Questo valore non è fisso: può variare in funzione del problema, della dimensione della popolazione e della strategia complessiva dell'algoritmo.

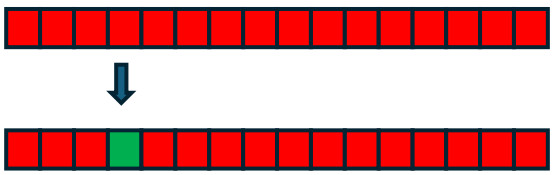
Nel caso di rappresentazione binaria degli individui (tipica dei crossover classici), la mutazione consiste **nel cambiare il valore di un singolo bit** all'interno della stringa binaria che rappresenta l'individuo.

Esempio:

Originale: 010011100111010

Mutato: 010111100111010

In questo esempio, è stato modificato un bit (il quinto da sinistra). Il risultato è un **individuo completamente nuovo**, che può trovarsi anche in una zona molto diversa del dominio delle soluzioni.



Se l'individuo è rappresentato da **variabili continue** (come nel crossover direzionale), allora la mutazione consiste **nel modificare in modo casuale il valore di una variabile reale**. La modifica può avvenire, ad esempio:

- aggiungendo un **rumore casuale** (es. gaussiano);
- ricalcolando il valore della variabile secondo una **nuova estrazione random**;
- perturbando la variabile con un piccolo **offset controllato**.

Esempio:

$$x_i \to x_i + \varepsilon \quad \text{dove } \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

Vantaggi della mutazione

- **Previene la convergenza prematura** verso un ottimo locale;
- **Mantiene la diversità genetica** nella popolazione;
- **Aumenta la capacità esplorativa** dell'algoritmo;
- È molto semplice da implementare e **leggera computazionalmente**.

Limiti e considerazioni

- **Troppa mutazione può eliminare individui buoni**: se applicata eccessivamente, la mutazione può diventare controproducente, cancellando combinazioni favorevoli trovate dal crossover;
- **Troppa poca mutazione porta a stagnazione**: senza un minimo di mutazione, l'algoritmo rischia di non esplorare lo spazio delle soluzioni e restare bloccato in ottimi locali;
- Non è un operatore "intelligente": **la mutazione è completamente casuale**, quindi ha bisogno di essere ben bilanciata per essere efficace.

Nei moderni algoritmi genetici, la mutazione è vista come un **meccanismo di salvataggio**: viene usata per **riattivare l'evoluzione** quando la popolazione ha raggiunto una situazione di **scarsa variabilità**. In molte implementazioni, infatti, il **tasso di mutazione è adattivo**, ovvero cresce se l'algoritmo rileva una perdita di diversità nella popolazione.

La **mutazione** è l'operatore che garantisce la **capacità esplorativa** dell'algoritmo genetico. Anche se da sola non basta per garantire l'ottimizzazione efficace, la sua presenza è **essenziale per mantenere attiva l'evoluzione** e per evitare l'intrappolamento in soluzioni subottimali. È un operatore semplice, ma la sua efficacia dipende fortemente da **come e quanto viene utilizzato**.

5.13.5.1 Elitismo

Un'importante miglioramento dell'algoritmo genetico è senz'altro il concetto di **elitismo.** Esso consiste nel non modificare il migliore individue nella generazione n e trasportarlo nella generazione n+1. Questo processo migliore la velocità di convergenza dell'algoritmo stesso, in particolare nell'ottimizzazione multi obiettivo (dove si trasporteranno senza modifiche nella generazione successiva i migliori individui di ogni obiettivo).

5.13.6 Convergenza dell'Algoritmo Genetico

Uno degli aspetti più discussi e, in certi casi, criticati degli **algoritmi genetici (GA)** è l'**assenza di un vero e proprio criterio matematico di convergenza**. A differenza di altri metodi numerici (come il gradiente o il Newton-Raphson), dove si può definire un criterio esplicito di arresto basato, ad esempio, sulla norma del gradiente o sulla variazione della funzione, nel GA **non si può determinare in modo preciso quando l'ottimo assoluto è stato raggiunto**.

In mancanza di un criterio rigoroso, si ricorre a una **strategia operativa empirica**: si **fissano a priori** il **numero di generazioni** e il **numero di individui per generazione**. Questi due parametri definiscono l'intera dimensione della ricerca:

$$\text{n° configurazioni totali} = (\text{n° individui}) \times (\text{n° generazioni})$$

In pratica, si esplora un numero finito e noto di configurazioni dello spazio delle soluzioni.

- Il **numero di individui per generazione** è in genere compreso tra **30 e 50**. Questo numero consente un buon compromesso tra varietà genetica e carico computazionale.
- Il **numero di generazioni** dipende dalla complessità del problema, ma spesso non supera le **8-10 generazioni** per problemi con circa **10 variabili decisionali**.

Questa impostazione introduce però una **grande limitazione strutturale**: l'algoritmo non sa quando ha effettivamente trovato una soluzione soddisfacente. Potrebbe convergere troppo presto oppure non convergere affatto.

In alcune implementazioni si utilizza un **criterio empirico di arresto anticipato**, che monitora il valore massimo della funzione obiettivo all'interno della popolazione. Se dopo **3 o 4 generazioni consecutive** il massimo non varia più in modo significativo, si **interrompe manualmente l'evoluzione**, ipotizzando che l'algoritmo si trovi ormai in prossimità di un ottimo (relativo o assoluto).

Questo meccanismo, seppur semplice, consente di **risparmiare tempo computazionale** evitando generazioni inutili, soprattutto quando il miglioramento diventa trascurabile.

La **prima generazione** è di importanza cruciale, perché da essa dipende la **variabilità iniziale della popolazione**, e dunque la **capacità esplorativa dell'algoritmo**.

- Quando il numero di **variabili è elevato**, si utilizza una **generazione casuale pura (random)** per garantire una buona dispersione iniziale nello spazio delle soluzioni.
- Quando il numero di **variabili è ridotto** (tipicamente meno di 10), si preferisce una generazione **quasi-casuale**, ad esempio mediante sequenze di **Sobol** o **Halton**. Queste sequenze producono punti più uniformemente distribuiti rispetto a quelli ottenuti con l'estrazione casuale semplice.

In entrambi i casi, la **prima generazione è a tutti gli effetti un DOE (Design of Experiments)**. È un esperimento iniziale di campionamento del dominio delle variabili che serve a inizializzare l'evoluzione.

Se esiste **una configurazione nota e promettente** (ad esempio ottenuta da un'ottimizzazione precedente o da esperienza ingegneristica), è buona pratica **inserire forzatamente tale configurazione nella prima generazione**. In questo modo si "semina" l'evoluzione con un individuo potenzialmente molto performante.

La convergenza dell'algoritmo genetico **non può essere formalmente garantita**, ma può essere gestita con approcci empirici e strategici:

- limitando il numero di generazioni,
- monitorando la stagnazione del valore massimo,
- inizializzando bene la popolazione,

Questi strumenti permettono di **contenere i limiti strutturali del metodo** e di adattarlo a molti contesti ingegneristici e scientifici reali, dove robustezza e flessibilità sono più importanti della precisione assoluta.

5.13.6.1 Ibridizzazione

A questo proposito è utile richiamare il concetto di **ibridizzazione del processo di ottimizzazione**, che prevede l'impiego, nella fase iniziale, di un algoritmo stocastico a carattere globale, seguito da una fase di affinamento mediante un algoritmo locale. In tal modo si sfrutta la robustezza garantita dall'esplorazione globale nella prima parte del processo e l'accuratezza dell'ottimizzazione locale nella fase successiva.

5.13.7 Implementazione dei vincoli: Penalty Function

Uno degli aspetti più delicati dell'ottimizzazione è la **gestione dei vincoli**. L'algoritmo genetico non ha, per natura, un meccanismo interno per far rispettare i vincoli. La strategia più diffusa per introdurli consiste nell'utilizzare una **penalty function**, cioè nel **penalizzare il valore della funzione obiettivo** ogni volta che un vincolo viene violato.

Un vantaggio pratico dell'algoritmo genetico è che **non richiede la definizione manuale di un parametro di penalità**. In molte implementazioni, infatti, la penalità viene calcolata automaticamente in base alle caratteristiche della popolazione corrente.

Supponiamo di essere alla generazione i. Si calcolano:

- $F_{\max}$: massimo valore della funzione obiettivo nella generazione i;
- $F_{\min}$: minimo valore della funzione obiettivo nella stessa generazione.

Allora la **penalità** p viene definita come:

$$p = F_{\max} - F_{\min}$$

In altre parole, la penalità è pari all'**ampiezza di variabilità della popolazione**. È una soglia interna che permette di regolare quanto "grave" è la violazione di un vincolo.

Valutazione della fitness

Per ogni individuo x, si calcola la **funzione di fitness corretta** come:

$$\text{FIT}(x) = f(x) - p \cdot g(x)$$

dove:

- $f(x)$ è la funzione obiettivo;
- $g(x)$ è una funzione che misura la **violazione del vincolo** (ad esempio: $g(x) = 0$ se x è ammissibile; $g(x) > 0$ se viola il vincolo).

In questo modo, **qualunque individuo che viola i vincoli riceve una penalità automatica** proporzionale alla distanza dai vincoli e viene **considerato peggiore anche del peggior individuo ammissibile**.

5.14 Riflessione sulla parallelizzazione

È evidente come il concetto di **CAO (Computer-Aided Optimization)** rivesta un ruolo centrale nella progettazione moderna e innovativa, dove l'obiettivo è ottenere prestazioni sempre migliori riducendo tempi e costi di sviluppo.

Un aspetto critico nell'adozione di tale approccio riguarda i **tempi complessivi necessari al raggiungimento di una soluzione ottimale**. Infatti, la valutazione delle prestazioni dei modelli progettuali avviene generalmente attraverso **simulazioni numeriche**, le quali sono onerose in termini di tempo computazionale.

Per ridurre tali tempi, una strategia particolarmente efficace è rappresentata dalla **parallelizzazione del processo di ottimizzazione**, che può essere implementata secondo due modalità principali:

Parallelizzazione della singola valutazione In questo caso si suddivide il calcolo di una singola simulazione su più processori. Il tempo complessivo richiesto per raggiungere l'ottimo può essere stimato come:

$$T_1 = \alpha_1 \cdot \frac{t}{p} \cdot n + \beta_1$$

dove:

t è il tempo necessario per una singola simulazione su un processore;

p è il numero di processori disponibili;

n è il numero di valutazioni richieste;

$\alpha_1 < 1$ rappresenta l'efficienza della parallelizzazione, che non è mai perfetta;

β_1 rappresenta il tempo non parallelizzabile (ad esempio operazioni CAD o fasi di pre/post-processing).

Parallelizzazione del processo di ottimizzazione In questo approccio non si suddivide una singola valutazione, ma si distribuiscono su processori diversi più configurazioni progettuali da valutare in parallelo. Ad esempio, in un algoritmo genetico di tipo popolazionale, è possibile valutare simultaneamente p individui su p processori. In questo caso, il tempo totale risulta:

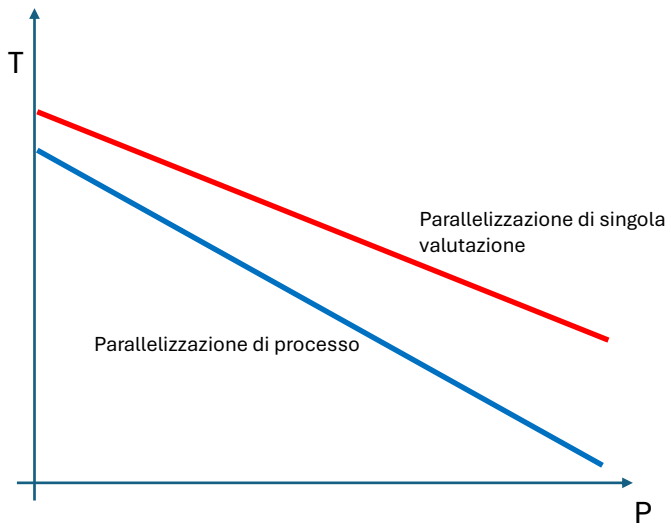
$$T_2 = \alpha_2 \cdot t \cdot \frac{n}{p} + \beta_2$$

con:

α_2 che rappresenta il coefficiente di efficienza della parallelizzazione (in genere $\alpha_2 > \alpha_1$);

β_2 che quantifica le operazioni non parallelizzabili (di norma $\beta_2 < \beta_1$).

Il confronto tra le due strategie evidenzia come, nella maggior parte dei casi, la **parallelizzazione del processo** sia più efficiente della parallelizzazione della singola simulazione. Ciò accade perché le fasi non parallelizzabili risultano più limitate e l'efficienza complessiva (α_2) è generalmente superiore, rendendo particolarmente vantaggiosa l'applicazione di tecniche evolutive o stocastiche di tipo popolazionale.



6 Ottimizzazione Multiobiettivo e Teoria Dei Giochi

Nella pratica progettuale — e in particolare nella fase di *predesign* — raramente ci si trova ad affrontare un problema *mono-obiettivo*. Al contrario, la maggior parte dei problemi reali prevede molteplici obiettivi in competizione tra loro, ciascuno dei quali riflette un criterio di prestazione o una caratteristica desiderata del sistema da progettare.

6.1 Formalizzazione dell'ottimizzazione multiobiettivo

Da un punto di vista matematico, un problema di ottimizzazione multiobiettivo può essere espresso come segue:

$$\max f(x) \in \mathbb{R}^m, \quad x \in \mathbb{R}^n$$

soggetto a:

$$g_j(x) \leq 0 \quad (j = 1, \dots, p)$$

$$h_j(x) = 0 \quad (j = 1, \dots, q)$$

Dove:

- $f(x) = [f_1(x), f_2(x), \dots, f_m(x)]$ rappresenta il vettore delle funzioni obiettivo;
- $g_j(x)$ sono i vincoli di disuguaglianza;
- $h_j(x)$ sono i vincoli di uguaglianza.

Tutti gli algoritmi precedentemente analizzati operavano nel caso particolare in cui $m = 1$, cioè un'unica funzione obiettivo. L'estensione a $m > 1$ implica la necessità di trattare simultaneamente più obiettivi, con tutte le complessità che ne derivano.

6.2 Metodo delle funzioni pesate: origine e limiti

Uno dei primi approcci per affrontare problemi multiobiettivo, sviluppato e diffusosi nella prima metà degli anni '90, consisteva nel combinare le diverse funzioni obiettivo in un'unica funzione scalare, detta *funzione pesata*:

$$\text{FIT}(x) = \sum_{i=1}^{m} w_i \, f_i(x)$$

dove w_i sono coefficienti di ponderazione, scelti per riflettere l'importanza relativa dei diversi obiettivi.

6.2.1 Problema della scelta dei pesi

Un primo limite evidente di questo approccio è la difficoltà nella **definizione corretta dei pesi**. Cambiando il rapporto tra i pesi, si modifica la soluzione finale dell'ottimizzazione. Ad esempio, consideriamo un problema in cui si desidera:

$$\max C_L \quad \text{e} \quad \min C_D$$

Una funzione pesata potrebbe essere formulata come:

$$\text{FIT}(x) = 0.5C_L(x) - 0.5C_D(x)$$

Ma modificando i pesi in modo asimmetrico:

$$\text{FIT}_1(x) = 0.8C_L(x) - 0.2C_D(x)$$

$$\text{FIT}_2(x) = 0.2C_L(x) - 0.8C_D(x)$$

si ottengono soluzioni drasticamente differenti. Questo evidenzia quanto sia **arbitraria e soggettiva** la scelta dei pesi, soprattutto in assenza di informazioni approfondite sulla rilevanza dei diversi criteri.

6.2.2 Problema della scala e dell'ordine di grandezza

Un altro limite riguarda la **diversa scala** delle funzioni obiettivo. Se ad esempio $C_L = \mathcal{O}(1)$ e $C_D = \mathcal{O}(0.01)$, l'effetto del secondo termine nella funzione pesata sarà trascurabile, a meno di introdurre fattori correttivi (ad esempio moltiplicatori o adimensionalizzazione):

$$\text{FIT}(x) = 0.5C_L(x) - 0.5 \cdot 100C_D(x)$$

Ma anche in questo caso il problema non si risolve completamente, poiché i **rapporti di scala possono variare da una configurazione all'altra**. Per esempio, in una configurazione potremmo avere:

$$C_L = 1, \quad C_D = 0.01$$

e in un'altra:

$$C_L = 2, \quad C_D = 0.03$$

Ciò significa che anche i **fattori di adimensionalizzazione** dovrebbero adattarsi dinamicamente alla configurazione corrente, complicando ulteriormente l'analisi e rischiando di snaturare il problema originario.

6.2.3 Perdita del significato originario del problema

Infine, combinare le funzioni obiettivo in una sola comporta la **perdita del significato fisico e tecnico originario** del problema. Si rischia di ottenere una soluzione che ottimizza una funzione artificiale, senza un chiaro riscontro nella realtà ingegneristica.

Alla luce delle criticità esposte, risulta evidente che è **necessario sviluppare tecniche di ottimizzazione che trattino ogni obiettivo in modo indipendente**, evitando la fusione precoce delle funzioni obiettivo.

Un errore metodologico comune, infatti, è costruire una funzione pesata **all'inizio della fase di progettazione**, quando le informazioni sulle priorità tra gli obiettivi sono ancora incomplete o del tutto assenti.

L'ottimizzazione multiobiettivo rappresenta una sfida fondamentale nella progettazione ingegneristica. Gli approcci storici basati sulla combinazione pesata degli obiettivi presentano gravi limiti sia teorici che pratici. Al contrario, i metodi moderni — basati sulla Teoria dei Giochi — offrono una visione più flessibile e realistica del problema, rendendo possibile una progettazione più consapevole, robusta e mirata.

6.3 Teoria dei Giochi e Ottimizzazione Multiobiettivo

La **teoria dei giochi** rappresenta un potente strumento per affrontare problemi di ottimizzazione in presenza di conflitti tra più obiettivi o tra più agenti decisori. In ambito ingegneristico e progettuale, essa viene impiegata principalmente in tre formulazioni distinte:

- **Pareto (ottimalità secondo Pareto)**
- **Nash (equilibrio di Nash)**
- **Stackelberg (leader–follower)**

Queste tre formulazioni sono elencate sia in **ordine storico di sviluppo** che in ordine di **diffusione applicativa** negli algoritmi di ottimizzazione: la teoria di **Pareto** è la prima ad essere stata introdotta e rimane tuttora la più largamente impiegata.

6.3.1 Teoria di Pareto

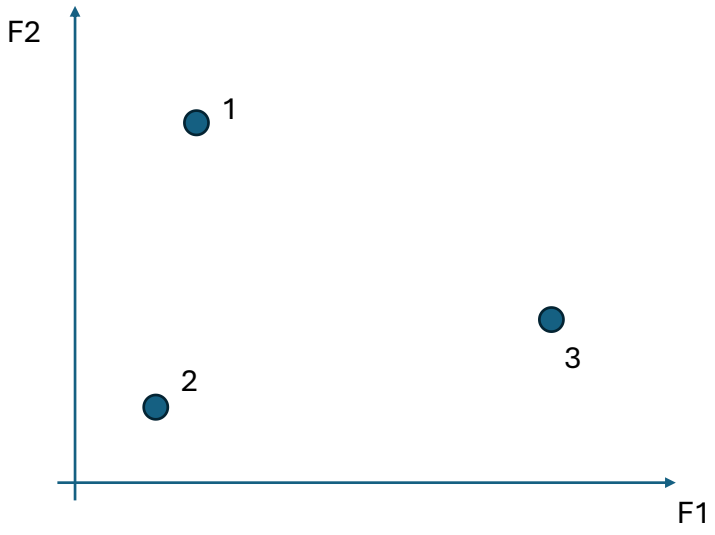
Si consideri un problema di ottimizzazione con due funzioni obiettivo:

$$\max f_1(x), \quad \max f_2(x)$$

Sia dato un insieme di configurazioni progettuali:

- Configurazione 1: $(f_1^{(1)}, f_2^{(1)})$
- Configurazione 2: $(f_1^{(2)}, f_2^{(2)})$
- Configurazione 3: $(f_1^{(3)}, f_2^{(3)})$

Ogni configurazione può essere rappresentata come un punto in uno **spazio obiettivo a due dimensioni**, con coordinate (f_1, f_2).



Dal confronto tra le configurazioni si osservano alcune proprietà:

- Se $f_1^{(1)} > f_1^{(2)}$ **e** $f_2^{(1)} > f_2^{(2)}$, allora la configurazione 1 **domina** la configurazione 2.
- Se invece $f_1^{(1)} > f_1^{(3)}$ **ma** $f_2^{(1)} < f_2^{(3)}$, allora non è possibile affermare quale delle due configurazioni sia "migliore": ciascuna è preferibile per uno degli obiettivi.

6.3.1.1 Dominanza secondo Pareto

In generale, si definisce **dominanza secondo Pareto** nel modo seguente:

Una configurazione A **domina** una configurazione B, e si scrive $A \succ_P B$, se:

$$\begin{cases} f_i(A) \geq f_i(B) & \forall i = 1, \dots, m \\ \exists j \text{ tale che } f_j(A) > f_j(B) \end{cases}$$

In altre parole, A è almeno altrettanto buona di B su tutti gli obiettivi, ed è strettamente migliore almeno su uno.

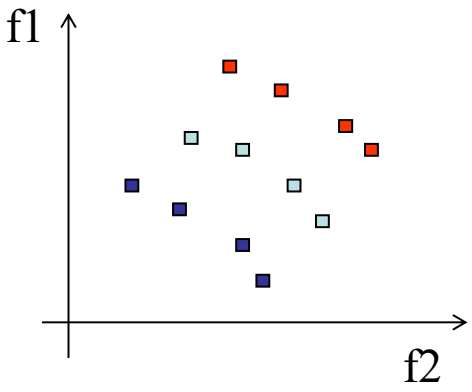
6.3.1.2 Fronte di Pareto

Le **configurazioni non dominate** formano il cosiddetto **fronte di Pareto**, ovvero l'insieme delle soluzioni ottimali in senso Pareto. Non esiste una singola "migliore" soluzione, ma un insieme di **soluzioni efficienti**, ognuna rappresentante un compromesso diverso tra gli obiettivi.

In un contesto di **Design of Experiments (DOE)** ideale — con esplorazione completa dello spazio delle soluzioni — il fronte di Pareto rappresenterebbe l'inviluppo superiore di tutte le configurazioni non dominate.

Soluzioni finali:

Fronte di Pareto = soluzioni non dominate



6.3.1.3 Implementazione della Teoria di Pareto negli Algoritmi di Ottimizzazione

La teoria di Pareto non è compatibile con algoritmi **basati sul gradiente** o con metodi **deterministici come il Simplex**, poiché questi operano su una funzione obiettivo scalare e mirano a una singola direzione di miglioramento. In ambito multiobiettivo, non esiste un'unica direzione di ottimizzazione.

L'implementazione più efficace della teoria di Pareto si realizza tramite **algoritmi evolutivi**, in particolare gli **algoritmi genetici**, che lavorano su popolazioni e sono intrinsecamente adatti a esplorare fronti di soluzioni.

Negli algoritmi genetici classici, la selezione avviene attraverso metodi probabilistici, come la **roulette wheel selection**, basati su valori scalari delle funzioni obiettivo.

Nel caso multiobiettivo, si utilizza invece una **selezione basata sulla dominanza**:

- Le configurazioni che **non sono dominate** da nessun'altra vengono selezionate con priorità;
- Si formano **gruppi o "isole" di dominanza**, che vengono poi mantenuti e incrociati nella generazione successiva;
- Non si assegna un singolo valore di fitness, ma si classifica la popolazione in base al grado di dominanza.

Il **cross-over classico** (basato sull'incrocio dei cromosomi/variabili) resta valido anche nel contesto multiobiettivo, in quanto agisce solo sulle variabili decisionali e non dipende dai valori delle funzioni.

Tuttavia, per **cross-over direzionali**, è necessario un adattamento.

Cross-over direzionale mono-obiettivo:

$$v_m = v_i + s \cdot \text{sign}(F_m^{(i)} - F_m^{(j)}) \cdot (v_i - v_j) + t \cdot \text{sign}(F_m^{(i)} - F_m^{(k)}) \cdot (v_i - v_k)$$

Cross-over direzionale multiobiettivo (modificato):

In presenza di più obiettivi $R = 1, \ldots, n$, il cross-over viene generalizzato:

$$v_m = v_i + \sum_{q=1}^{n} \left[s_q \cdot \text{sign}(F_q^{(i)} - F_q^{(j)}) \cdot (v_i - v_j) + t_q \cdot \text{sign}(F_q^{(i)} - F_q^{(k)}) \cdot (v_i - v_k) \right]$$

Dove:

- F_q è la q-esima componente della funzione obiettivo (vettoriale),
- s_q, t_q sono i parametri di esplorazione lungo ogni direzione,
- La somma tiene conto di tutti gli obiettivi simultaneamente.

L'unico limite strutturale è che questo approccio **non può essere utilizzato con variabili categoriche o discrete non numeriche**, poiché le operazioni vettoriali e direzionali non hanno significato in tali spazi.

L'applicazione della **teoria di Pareto** all'ottimizzazione multiobiettivo rappresenta un fondamento cruciale nella progettazione moderna, permettendo di esplorare soluzioni equilibrate in scenari complessi. L'impiego di algoritmi genetici e tecniche di selezione basate sulla dominanza costituisce un'alternativa robusta ai metodi classici, fornendo al progettista un **ventaglio di opzioni ottimali** tra cui scegliere in base a criteri non esclusivamente numerici ma anche qualitativi e strategici.

6.3.2 Teoria di Nash

La **teoria di Nash** rappresenta un approccio multiobiettivo all'ottimizzazione basato sull'**equilibrio competitivo** tra due entità (giocatori), ciascuna delle quali ottimizza un proprio obiettivo agendo su un sottoinsieme specifico delle variabili decisionali.

Siamo nel contesto di **ottimizzazione multiobiettivo a due componenti** (cioè con due funzioni obiettivo). Si suppone di avere un insieme di n variabili decisionali che viene suddiviso in due sottoinsiemi distinti:

- Il primo sottoinsieme, $x \in A$, è controllato dal **giocatore A**
- Il secondo sottoinsieme, $y \in B$, è controllato dal **giocatore B**

Quindi il dominio complessivo della funzione è dato dall'unione dei due insiemi:

$$A \cup B \subset \mathbb{R}^n$$

Le due funzioni obiettivo sono:

$$f_A(x, y): A \times B \to \mathbb{R}$$

$$f_B(x, y): A \times B \to \mathbb{R}$$

6.3.2.1 Equilibrio di Nash

L'**equilibrio di Nash** si raggiunge in un punto (x^*, y^*) tale che:

- Fissato y^*, **non è possibile migliorare** f_A modificando solo x
- Fissato x^*, **non è possibile migliorare** f_B modificando solo y

Formalmente:

$$x^* = \underset{x \in A}{\operatorname{argmin}} f_A(x, y^*) \quad \text{(con } y^* \text{ fisso)}$$

$$y^* = \underset{y \in B}{\operatorname{argmin}} f_B(x^*, y) \quad \text{(con } x^* \text{ fisso)}$$

In altri termini, ciascun giocatore fa del proprio meglio supponendo che l'altro non cambi strategia. Nessuno dei due può migliorare la propria funzione obiettivo unilateralmente.

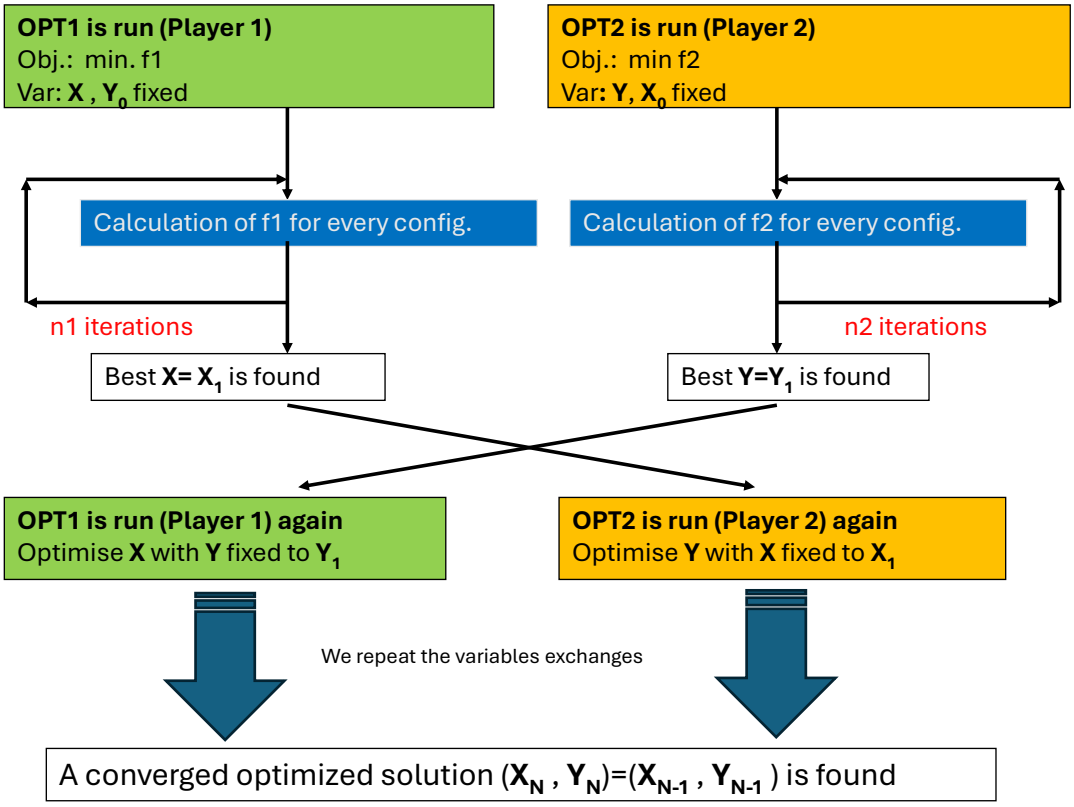
Il contesto di Nash è **competitivo**, non cooperativo:

- **Giocatore A** agisce esclusivamente sulle variabili x per migliorare la funzione f_A
- **Giocatore B** agisce esclusivamente sulle variabili y per migliorare f_B

Ogni giocatore è limitato nella propria libertà di azione, poiché può intervenire solo su parte delle variabili.

6.3.2.2 Implementazione della Teoria di Nash

- **Inizializzazione**: si parte da una configurazione iniziale $(x^{(0)}, y^{(0)})$
- **Mossa del giocatore A**: ottimizza $f_A(x, y^{(0)})$ rispetto a x, ottenendo $x^{(1)}$
- **Mossa del giocatore B**: con $x^{(1)}$ fissato, ottimizza $f_B(x^{(1)}, y)$ rispetto a y, ottenendo $y^{(1)}$
- Il processo si ripete, generando una sequenza di iterazioni:
- $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots$
- Si arresta quando si raggiunge una configurazione $(x^{(n)}, y^{(n)})$ che **non può più essere migliorata** da nessuno dei due giocatori: è il punto di equilibrio.



- **Flessibilità algoritmica**: è possibile usare qualunque algoritmo mono-obiettivo (inclusi quelli basati su **gradiente**, molto più veloci rispetto agli algoritmi genetici).
- Si **ottimizzano le funzioni in parallelo**, ma separatamente: ogni funzione ha un suo ottimizzatore.

Confronto Nash vs Pareto

Caratteristica	Nash	Pareto
Tipo di soluzioni	Un **solo punto** di equilibrio	**Molteplici soluzioni** ideali
Approccio	Competitivo	Cooperativo
Complessità computazionale	Relativamente **più semplice**	Generalmente **più complesso**
Algoritmi richiesti	Anche algoritmi **locali/veloci** (es. gradiente)	Tipicamente **algoritmi evolutivi** o globali
Sensibilità alla scelta delle variabili	**Alta**: la divisione tra x e y è critica	Non serve dividere le variabili
Precisione (a parità di risorse)	Migliore in casi a bassa dimensionalità	Migliore per esplorazione globale

6.3.3 Nota sulla velocità e accuratezza

Una delle differenze fondamentali tra l'approccio **Pareto** e quello **Nash** nell'ottimizzazione multiobiettivo risiede nella **natura e quantità delle soluzioni** che ciascuna teoria produce:

- **Pareto** mira a costruire un **insieme di soluzioni ottimali** (il cosiddetto *fronte di Pareto*), tutte non dominabili: ciò significa che **nessuna di esse può essere migliorata in un obiettivo senza peggiorare almeno un altro**. In altre parole, tutte queste soluzioni rappresentano configurazioni "ideali" a cui il decisore può ispirarsi, in base a preferenze esterne.
- **Nash**, al contrario, conduce a una **singola configurazione di equilibrio**, in cui ogni "giocatore" (o funzione obiettivo) ha ottimizzato il proprio obiettivo in modo competitivo, rispetto al sottoinsieme di variabili di propria competenza.

Prestazioni teoriche: Pareto ≥ Nash

Se si ipotizzasse di disporre di **algoritmi perfetti**, capaci di trovare sempre il massimo globale per ogni funzione obiettivo, allora:

- Il **fronte di Pareto ottimale** (indicato con P_∞) rappresenterebbe **tutte le migliori soluzioni possibili**.
- La **soluzione di Nash ottimale** (indicata con N_∞) sarebbe **una sola tra quelle del fronte**, oppure in casi specifici, **al di sotto di esso** in termini di qualità globale.

In termini pratici:

- Se $N_\infty \in P_\infty$, allora Nash è una delle soluzioni di Pareto.
- Se $N_\infty \notin P_\infty$, allora Nash ha performance inferiori.

Dunque, **Pareto è teoricamente superiore**, perché include Nash come caso particolare e offre più alternative al decisore.

Prestazioni computazionali: Nash spesso più conveniente

Nonostante la superiorità teorica di Pareto, l'**implementazione pratica** dei due approcci può capovolgere il quadro:

Vantaggi computazionali dell'approccio di Nash

- Nash consente l'utilizzo di **algoritmi classici di ottimizzazione mono-obiettivo**, come ad esempio il **metodo del gradiente**.
- Ogni funzione obiettivo può essere ottimizzata con un **algoritmo dedicato**, veloce ed efficiente.
- Le funzioni vengono **ottimizzate in parallelo**, ma in modo indipendente: ogni giocatore lavora sul proprio sottoinsieme di variabili.
- **Iterativamente**, il processo converge rapidamente verso il punto di equilibrio.

Svantaggi pratici dell'approccio di Pareto

- Pareto richiede un'esplorazione simultanea e bilanciata **di tutte le funzioni obiettivo**, che interagiscono tra loro.

- Ciò rende necessaria l'adozione di **algoritmi globali** e generalmente più complessi, come gli **algoritmi genetici**, che operano per popolazioni e valutazioni evolutive.
- Gli algoritmi evolutivi, sebbene potenti, sono anche **computazionalmente costosi** e lenti nel convergere a soluzioni ottimali.

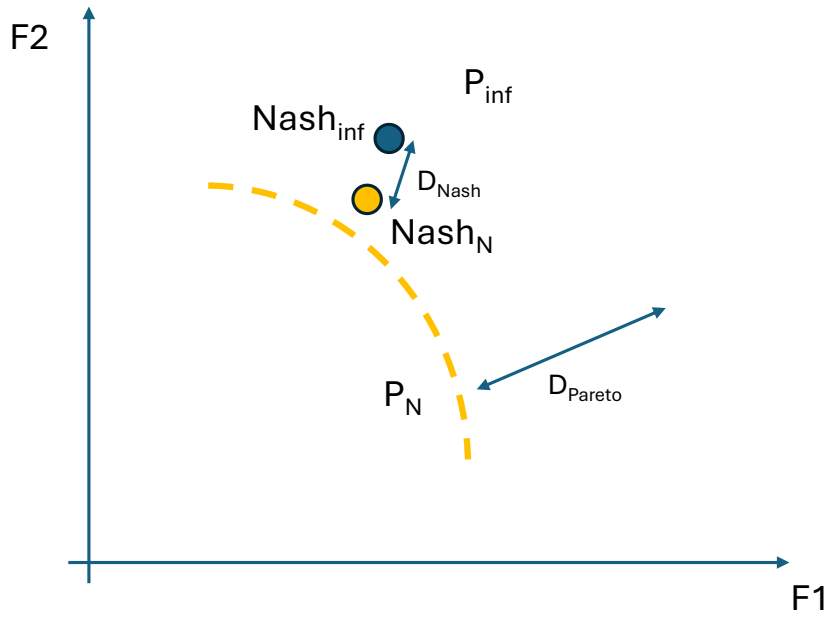
Precisione comparata con risorse limitate

In contesti reali, dove si fissano vincoli sulle risorse computazionali (ad esempio, il numero massimo di iterazioni o valutazioni N), si osserva spesso che:

$$D_N < D_P$$

Dove:

- D_N è l'**errore di approssimazione** della soluzione Nash
- D_P è l'**errore di approssimazione** del fronte di Pareto



Questo accade perché:

- Nash lavora con **algoritmi mono-obiettivo**, più stabili e veloci nel trovare una buona soluzione locale.
- Pareto, per definizione, deve **esplorare simultaneamente molte direzioni**, quindi la qualità delle sue soluzioni cresce più lentamente con lo stesso budget computazionale.

In molte applicazioni si utilizza una **strategia combinata**:

- Si calcola un punto di **Nash** come punto iniziale buono e veloce
- Si usa tale punto come **seme iniziale** per costruire il **fronte di Pareto**, sfruttando algoritmi genetici

La teoria di Nash offre una valida alternativa per affrontare problemi multiobiettivo, soprattutto quando:

- Si vogliono evitare costi computazionali elevati
- Si ha una buona conoscenza a priori della struttura del problema
- Si è interessati a **una soluzione unica** e stabile

Tuttavia, è meno completa di Pareto in termini di varietà di soluzioni e richiede una buona **ingegnerizzazione della suddivisione delle variabili**.

6.3.4 Teoria di Stackelberg (o Giochi Gerarchici)

La teoria di Stackelberg è un'estensione della teoria dei giochi non cooperativi, simile alla teoria di Nash, ma con una **struttura gerarchica** tra i giocatori. Questo significa che uno dei giocatori prende **decisioni prima dell'altro**, e l'altro risponde ottimizzando la propria strategia in base alla scelta del primo.

6.3.4.1 Struttura del gioco

- **Due giocatori**:
    - Giocatore A → **Leader**, controlla le variabili **x**
    - Giocatore B → **Follower**, controlla le variabili **y**

La particolarità di Stackelberg è proprio questa **asimmetria informativa e decisionale**: il **Leader** agisce per primo, fissando la propria strategia, mentre il **Follower** osserva quella scelta e **reagisce ottimizzando** la propria.

6.3.4.2 Equilibrio di Stackelberg

La soluzione ottima in questo contesto si chiama **equilibrio di Stackelberg**, ed è data da una gerarchia di ottimizzazioni:

$$\begin{cases} y^*(x) = \underset{y}{\operatorname{argmin}} f_B(x, y) & \text{(risposta ottima del follower per ogni } x) \\ x^* = \underset{x}{\operatorname{argmin}} f_A(x, y^*(x)) & \text{(il leader tiene conto della risposta del follower)} \end{cases}$$

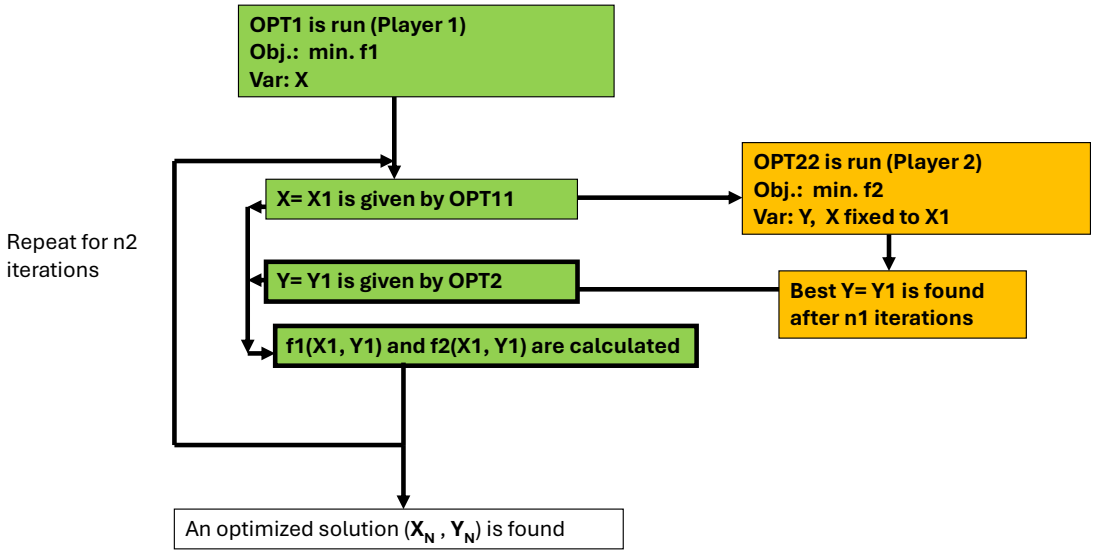
Cioè:

- Il **Follower** risolve completamente la propria ottimizzazione **per ogni** possibile valore di x deciso dal Leader.
- Il **Leader**, sapendo che il Follower reagirà nel modo migliore per sé, sceglie **la strategia che gli conviene di più considerando quella reazione**.

Nel contesto computazionale o ingegneristico, si parte da una configurazione iniziale (x_0, y_0). Poi:

- Il **Leader (A)** fa un **passo iterativo** su x, ottenendo x_1.
- Il **Follower (B)** osserva il nuovo valore x_1 e **risolve completamente** la propria funzione obiettivo, ottenendo $y_1 = \operatorname{argmin}_y f_B(x_1, y)$.
- A questo punto si ha la nuova configurazione (x_1, y_1).

- Il processo si ripete: ad ogni passo del Leader, il Follower rifà tutta la sua ottimizzazione.



Una delle principali criticità della teoria di Stackelberg riguarda il suo elevato costo computazionale. Infatti, ogni volta che il Leader compie un singolo passo nella sua strategia, il Follower deve risolvere un'intera ottimizzazione per reagire nel modo più efficace. Questo comporta una mole di calcoli molto elevata, soprattutto se si considerano scenari complessi o ad alta dimensionalità.

Proprio per questa ragione, il modello di Stackelberg risulta spesso poco adatto alle applicazioni industriali in tempo reale, dove le decisioni devono essere prese rapidamente e i sistemi non possono permettersi lunghe elaborazioni. Inoltre, quando lo spazio delle decisioni (cioè il numero di variabili coinvolte) è molto ampio o quando le funzioni obiettivo sono particolarmente articolate, l'approccio diventa difficilmente gestibile, se non del tutto impraticabile.

Nonostante queste difficoltà, la teoria di Stackelberg trova applicazione in diversi ambiti. È particolarmente utile, ad esempio, in economia, per modellare situazioni di concorrenza asimmetrica: un classico caso è quello di un'impresa dominante che fissa il prezzo di un prodotto, costringendo le aziende concorrenti a scegliere le loro strategie in base a quella scelta.

Anche in ingegneria dei sistemi il modello è impiegato per risolvere problemi in cui un sistema principale prende decisioni che impongono vincoli o condizioni operative a sottosistemi secondari, che a loro volta reagiscono ottimizzando le proprie prestazioni. Un altro ambito rilevante è quello della pianificazione logistica gerarchica, dove una struttura centrale stabilisce direttive o priorità che influenzano le decisioni operative prese localmente da singole unità o centri.

In tutti questi casi, il vantaggio del modello sta proprio nella sua capacità di rappresentare in modo realistico dinamiche decisionali sbilanciate, dove non tutti i soggetti coinvolti operano sullo stesso piano, ma seguono una sequenza ben precisa di azioni e reazioni.

7 Multi-Criteria Decision Making (MCDM)

Il *Multi-Criteria Decision Making* (MCDM) rappresenta una vasta famiglia di metodi volti a supportare i processi decisionali in presenza di molteplici criteri. Tali metodi trovano applicazione in numerosi contesti, dalla gestione industriale alla pianificazione strategica, fino all'ingegneria, alla finanza e alla pubblica amministrazione. L'idea centrale alla base dell'MCDM è che in molte situazioni decisionali reali non esiste un unico criterio da ottimizzare, bensì una pluralità di aspetti da considerare, spesso in conflitto tra loro.

Un esempio semplice e intuitivo può aiutarci a comprendere la logica sottostante: si pensi alla scelta di un'automobile da acquistare. Ogni auto disponibile sul mercato è caratterizzata da una serie di **attributi** o **criteri** rilevanti per il potenziale acquirente:

- prezzo,
- potenza del motore,
- accelerazione,
- velocità massima,
- consumi di carburante,
- livello di emissioni,
- comfort,
- affidabilità, ecc.

Questi attributi possono essere tra loro **in conflitto**: ad esempio, un'auto sportiva con alte prestazioni (alta potenza e accelerazione) tende ad avere consumi elevati e un prezzo maggiore. In tal caso, non esiste una soluzione "migliore" in senso assoluto, ma piuttosto è necessario individuare l'alternativa che **meglio soddisfa le preferenze soggettive del decisore**.

I metodi MCDM sono progettati proprio per rispondere a questa esigenza: **dati un insieme di alternative, i relativi attributi e le preferenze espresse dal decisore, forniscono un criterio strutturato per selezionare la soluzione più soddisfacente**.

Ad esempio, nel caso in cui si sia eseguita un'**ottimizzazione multi-obiettivo**, è possibile ottenere una moltitudine di soluzioni efficienti (ottime in senso di Pareto), rappresentate nel cosiddetto **fronte di Pareto**. In questo contesto, i metodi MCDM consentono di **selezionare una soluzione tra le molte ottime**, sulla base delle preferenze espresse dal decisore.

7.1 MADM: Multi-Attribute Decision Making

All'interno della grande famiglia dell'MCDM, possiamo distinguere in particolare i **metodi MADM (Multi-Attribute Decision Making)**, che operano su un insieme **discreto** di alternative, valutandole attraverso **attributi finiti e ben definiti**. L'informazione di partenza nei metodi MADM è tipicamente strutturata in forma di **matrice decisionale**.

Matrice decisionale

La matrice decisionale è così strutturata:

- **le righe** rappresentano le alternative da valutare (ad es. le diverse auto);
- **le colonne** rappresentano gli attributi/criteri considerati (ad es. prezzo, potenza, consumi);
- **le celle** contengono i valori numerici (o valutazioni normalizzate) degli attributi per ciascuna alternativa.

Non tutti gli attributi hanno la stessa importanza nel processo decisionale. Per tener conto della diversa rilevanza dei criteri, alla matrice decisionale si associa un **vettore dei pesi**:

$$\mathbf{w} = \{w_1, w_2, \ldots, w_k\}$$

dove ogni $w_i \in [0,1]$ rappresenta il peso relativo assegnato all'attributo i-esimo. La somma di tutti i pesi può essere normalizzata a 1.

Alternative designs	Attributes			
	y_1	y_2	$\cdots$	y_k
a_1	y_{11}	y_{12}	$\cdots$	y_{1k}
a_2	y_{21}	y_{22}	$\cdots$	y_{2k}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a_n	y_{n1}	y_{n2}	$\cdots$	y_{nk}

Sports & GT Car	Price ($1000s)	Curb Weight (lb.)	Horsepower	Zero to 60 (Seconds)	Speed at 1/4 mile (mph)
Accura Integra Type R	25,035	2577	195	7	90,7
Accura NSX-T	93,758	3066	290	5	108
BMW Z3 2.8	40,900	2844	189	6,6	93,2
Chevrolet Camaro Z28	24,865	3439	305	5,4	103,2
Chevrolet Corvette Convertil	50,144	3246	345	5,2	102,1
Dodge Viper RT/10	69,742	3319	450	4,4	116,2
Ford Mustang GT	23,200	3227	225	6,8	91,7
Honda Prelude Type SH	26,382	3042	195	7,7	89,7
Mercedes-Benz CLK320	44,988	3240	215	7,2	93
Mercedes-Benz SLK230	42,762	3025	185	6,6	92,3
Mitsubishi 3000GT VR-4	47,518	3737	320	5,7	99
Nissan 240SX SE	25,066	2862	155	9,1	84,6
Pontiac Firebird Trans Am	27,770	3455	305	5,4	103,2
Porsche Boxster	45,560	2822	201	6,1	93,2
Toyota Supra Turbo	40,989	3505	320	5,3	105
Volvo C70	41,120	3285	236	6,3	97

7.2 Adimensionalizzazione degli attributi

Uno degli aspetti fondamentali da affrontare prima di iniziare qualsiasi procedura di **MCDM (Multi-Criteria Decision Making)** è l'**adimensionalizzazione dei dati**. Questa fase preliminare è cruciale, perché come abbiamo già osservato, i dati in ingresso possono avere **natura molto diversa**: possono trattarsi di pesi, potenze, rendimenti, costi, tempi, indici di affidabilità e così via. Il problema che si pone è evidente: grandezze espresse con **unità di misura differenti** o caratterizzate da ordini di grandezza molto distanti non sono immediatamente confrontabili. Per questo motivo diventa necessario trasformarle in **valori adimensionali**, in modo da poterle trattare tutte su una **base comune** senza distorsioni dovute alle scale originarie.

Esistono diversi metodi di adimensionalizzazione, e la scelta del più appropriato dipende dal tipo di problema e dal contesto applicativo. In altre parole, spetta al **team di progettazione** valutare quale metodologia si adatti meglio al caso in esame. In questo corso prenderemo in esame tre modalità principali:

- **Adimensionalizzazione classica**
- **Z-score**
- **Fuzzy logic**

7.2.1 Adimensionalizzazione classica

Si tratta della tecnica più semplice e immediata, che consiste nel **ridimensionare ogni attributo** in modo che assuma valori compresi tra 0 e 1. La formula utilizzata è la seguente:

$$X_{adim} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

dove:

- X è il valore dell'attributo considerato,
- $X_{\min}$ e $X_{\max}$ rappresentano rispettivamente il minimo e il massimo valore osservato per quell'attributo.

Questa trasformazione porta ogni dato in un **intervallo normalizzato** $[0,1]$, facilitando il confronto diretto tra grandezze di diversa natura.

Tuttavia, questo approccio presenta un limite importante: non tiene in alcun modo conto della **distribuzione statistica dei dati**. Se i valori sono distribuiti in maniera molto asimmetrica, o se vi sono forti concentrazioni in determinate zone dell'intervallo, questa procedura rischia di **appiattire le informazioni**, facendo perdere elementi utili all'analisi.

7.2.2 Metodo Z-score

Per superare queste difficoltà è stato sviluppato l'approccio basato sullo **Z-score**, che tiene conto non solo del valore dell'attributo, ma anche della sua **variabilità statistica**. La formula è la seguente:

$$Z = \frac{X - \mu}{\sigma}$$

dove:

- X è il valore dell'attributo,
- μ è la **media** dei valori osservati,
- σ è la **deviazione standard** della distribuzione.

In questo modo il dato viene trasformato in funzione della sua distanza dalla media, espressa in termini di **numero di deviazioni standard**. Il risultato è un insieme di valori che, indipendentemente dalla scala di partenza, ha media pari a 0 e deviazione standard pari a 1. Il vantaggio di questa metodologia è che consente di **evidenziare gli scostamenti significativi** rispetto alla media e di tener conto della distribuzione statistica, evitando le distorsioni tipiche della normalizzazione classica.

7.2.3 Fuzzy logic

Uno degli elementi che spesso può rivelarsi particolarmente utile per il **team di progettazione** è la possibilità di intervenire direttamente nella fase di **adimensionalizzazione**. In questo modo si riesce a far percepire ai successivi algoritmi di analisi alcuni aspetti del problema che, se trattati con una normalizzazione tradizionale, risulterebbero difficili da rappresentare.

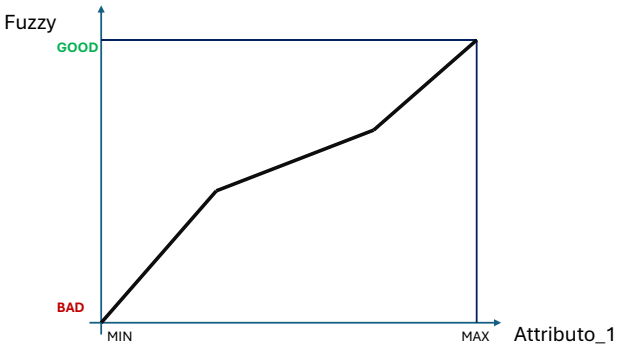
Tra le tecnologie moderne, una delle più diffuse è il metodo basato sulla **Fuzzy Logic**. L'idea di fondo è quella di trasformare un qualunque parametro numerico nella sua **qualità percepita** rispetto al problema decisionale. In altre parole, non ci limitiamo a normalizzare i valori in un intervallo predefinito, ma cerchiamo di tradurli in termini di "quanto sono buoni" o "quanto sono accettabili" per la scelta finale.

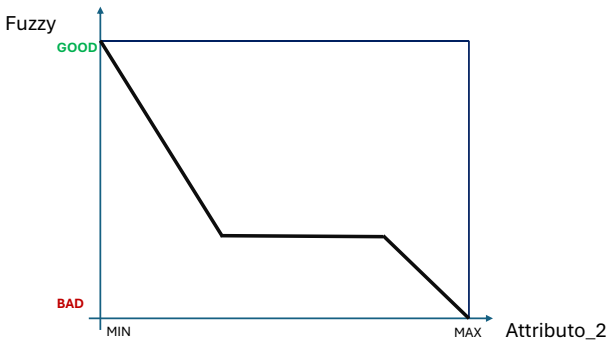
Un esempio classico è quello in cui due parametri tecnici (ad esempio **consumo energetico** ed **affidabilità**) vengono trasformati, attraverso funzioni fuzzy, in un indice qualitativo di "efficienza globale". Questa trasformazione non avviene in maniera rigida e lineare, ma mediante **funzioni di appartenenza** che permettono di descrivere gradi intermedi come *scarso, medio, buono, ottimo*, assegnando a ciascun valore un'appartenenza compresa tra 0 e 1.

Le funzioni fuzzy possono essere rappresentate graficamente tramite curve (triangolari, trapezoidali, gaussiane, sigmoidi, ecc.), che descrivono in che misura un valore numerico appartiene a una certa categoria qualitativa.
Per esempio:

- un rendimento del 70% potrebbe avere un grado di appartenenza pari a 0.3 nella categoria *ottimo*,
- mentre un rendimento del 90% potrebbe raggiungere un grado di appartenenza di 0.9.

Grazie a questa trasformazione, i parametri vengono espressi in termini più vicini al **giudizio qualitativo umano**, permettendo all'algoritmo decisionale di gestire anche informazioni complesse o non lineari.





I principali vantaggi di questo metodo possono essere così riassunti:

- possibilità di **inserire non linearità** nel processo di adimensionalizzazione, evitando la rigidità dei metodi classici;
- capacità di **modellare la percezione qualitativa** dei parametri, in linea con l'esperienza o le esigenze del progettista;
- flessibilità nell'**intervenire direttamente nella definizione della qualità** del singolo attributo, adattando il processo alle priorità del problema in esame;
- possibilità di integrare conoscenza **esperta** (ad esempio di un ingegnere o di un decisore) all'interno di un processo matematico formalizzato.

La Fuzzy Logic non si limita a uniformare i dati su una scala numerica, ma fornisce un linguaggio intermedio tra numerico e qualitativo, in grado di arricchire l'analisi multicriterio con sfumature difficili da catturare con altre metodologie.

7.3 Metodi per l'attribuzione dei pesi

La determinazione dei pesi è una fase cruciale. Esistono diverse metodologie per assegnare i pesi agli attributi. Tra le più comuni, ricordiamo:

7.3.1 Attribuzione diretta

È il metodo più semplice: si chiede al decisore di assegnare un valore numerico a ciascun attributo, utilizzando una scala prestabilita (ad esempio da 0 a 10). Un punteggio più elevato indica una maggiore importanza. **Vantaggi**:

- semplicità e immediatezza;
- rapidità di esecuzione.

Svantaggi:

- scarsa precisione;
- soggettività elevata;
- non garantisce coerenza tra le valutazioni.

7.3.2 Attribuzione indiretta – Metodo degli autovettori (Analytic Hierarchy Process – AHP)

Proposto da **Thomas Saaty**, l'AHP si basa su confronti **binari** tra attributi. Invece di assegnare un punteggio assoluto, il decisore confronta due attributi alla volta e ne valuta l'importanza relativa. Viene utilizzata una **scala semantica discreta**:

Valore	Interpretazione
1	pari importanza
3	importanza debole
5	importanza forte
7	importanza molto forte
9	importanza assoluta

Viene costruita così una **matrice di confronto a coppie** (matrice quadrata A di ordine n, dove n è il numero di attributi), le cui proprietà fondamentali sono:

- la diagonale principale contiene solo valori pari a 1 (confronto di un attributo con sé stesso);
- la matrice è reciproca, ossia se $a_{ij} = x$ allora $a_{ji} = 1/x$;
- in caso di perfetta coerenza, vale la **proprietà transitiva**:

$$a_{ij} \cdot a_{jk} = a_{ik}$$

Esempio:

Se confronto 3 attributi, posso ottenere una matrice come la seguente:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 1/3 & 1 & 2 \\ 1/5 & 1/2 & 1 \\ \end{bmatrix}$$

Per ottenere il vettore dei pesi **w**, si calcola il **vettore autovettore principale** associato al **massimo autovalore** λ_{max} della matrice A. In pratica si risolve:

$$A \cdot \mathbf{w} = \lambda_{max} \cdot \mathbf{w}$$

Coerenza delle valutazioni – Indice di Saaty

Dato che il metodo si basa su giudizi soggettivi, è importante verificare la **consistenza** delle valutazioni. A tal fine Saaty definisce un **Indice di Consistenza (CI)**:

$$CI = \frac{\lambda_{max} - n}{n - 1}$$

dove:

- λ_{max} è l'autovalore massimo della matrice di confronto;
- n è il numero di attributi.

Per verificare se il valore ottenuto è accettabile, si confronta il CI con un **Indice Casuale (RI)** (tabulato in funzione di n) calcolando il **Consistency Ratio (CR)**:

$$CR = \frac{CI}{RI}$$

Se:

$$CR < 0.10$$

allora la matrice può considerarsi sufficientemente **consistente**. In caso contrario, è consigliabile rivedere i confronti a coppie.

7.3.3 Metodo dell'Entropia

Il **metodo dell'entropia** è un approccio oggettivo per la determinazione dei pesi da attribuire agli attributi (criteri) in un problema di decisione multicriterio. Risulta particolarmente utile quando il decisore non ha a disposizione preferenze esplicite o conoscenze approfondite per assegnare i pesi in modo soggettivo, oppure quando si desidera analizzare la variabilità e l'informazione contenuta nei dati di una **matrice decisionale**.

È consigliato in contesti in cui si vogliano investigare **differenze strutturali** all'interno del set di dati, come indicato in [2], a partire da matrici di decisione analoghe a quella riportata in Tabella 1.1. Di seguito si descrivono i passaggi fondamentali per l'applicazione del metodo.

7.3.3.1 Step 1 – Normalizzazione della matrice decisionale

Il primo passo consiste nella **normalizzazione** dei valori della matrice decisionale. Dato un insieme di n alternative e k attributi, per ogni elemento y_{ij} della matrice (valore dell'alternativa i rispetto all'attributo j), si calcola la **probabilità relativa** p_{ij} come segue:

$$p_{ij} = \frac{y_{ij}}{\sum_{i=1}^{n} y_{ij}} \quad \forall i = 1, \ldots, n;\ \forall j = 1, \ldots, k$$

Tale normalizzazione consente di trasformare ogni colonna della matrice in una distribuzione di probabilità, necessaria per il calcolo dell'entropia informativa.

7.3.3.2 Step 2 – Calcolo dell'entropia E_j per ogni attributo

L'**entropia** E_j misura il livello di disordine o incertezza associato all'attributo j. Essa è calcolata con la formula:

$$E_j = -\alpha \sum_{i=1}^{n} p_{ij} \cdot \ln(p_{ij}) \quad \forall j = 1, \ldots, k$$

dove:

- $\ln(p_{ij})$ è il logaritmo naturale di p_{ij},
- α è una costante normalizzante definita come:

$$\alpha = \frac{1}{\ln(n)}$$

Questa costante garantisce che il valore di entropia sia compreso tra 0 e 1, ossia:

$$0 \leq E_j \leq 1$$

Maggiore è l'entropia, minore è la capacità dell'attributo di **differenziare** tra le alternative, ossia **minore è l'informazione utile** contenuta in esso ai fini decisionali.

7.3.3.3 Step 3 – Calcolo dei pesi oggettivi

Sulla base dell'entropia calcolata, si determina il grado di informazione apportata da ciascun attributo. Si definisce quindi una **misura di differenziazione** per ogni attributo:

$$d_j = 1 - E_j$$

Un valore d_j elevato indica che l'attributo j è maggiormente informativo (ha bassa entropia) e quindi più rilevante nella decisione. I **pesi normalizzati** w_j da attribuire agli attributi sono allora calcolati come:

$$w_j = \frac{d_j}{\sum_{i=1}^{k} d_i} \quad \forall j = 1, \ldots, k$$

7.3.3.4 Correzione dei pesi soggettivi

Nel caso in cui il decisore abbia già attribuito a priori dei pesi soggettivi c_j, derivanti da esperienza, competenza o metodi qualitativi (ad esempio: AHP, attribuzione diretta), questi possono essere **corretti** attraverso il metodo dell'entropia. I nuovi pesi corretti f_j sono ottenuti combinando i pesi soggettivi c_j con quelli oggettivi w_j:

$$f_j = \frac{c_j \cdot w_j}{\sum_{i=1}^{k} c_i \cdot w_i} \quad \forall j = 1, \ldots, k$$

Questa operazione consente di tenere conto sia dell'expertise del decisore, sia della struttura informativa effettiva contenuta nei dati, producendo una **sintesi bilanciata** tra approccio soggettivo e oggettivo.

Il metodo dell'entropia si distingue per la sua capacità di **estrarre automaticamente informazione dai dati**, senza richiedere intervento diretto del decisore nella fase di assegnazione dei pesi. Tuttavia, proprio per la sua natura puramente oggettiva, può risultare meno adeguato in situazioni dove le **preferenze del decisore** o elementi esterni al dataset (come vincoli normativi, economici o ambientali) abbiano un ruolo cruciale.

Il MCDM e in particolare i metodi MADM permettono di affrontare in modo strutturato e razionale problemi decisionali complessi, caratterizzati dalla presenza di molteplici criteri. Attraverso la costruzione della matrice decisionale e la ponderazione degli attributi, è possibile selezionare l'alternativa che meglio soddisfa le preferenze del decisore. Metodi come l'AHP, pur introducendo un certo grado di complessità, forniscono un valido strumento per garantire la coerenza e la trasparenza del processo decisionale.

7.4 Famiglie di metodi MCDM

Nell'ambito dell'analisi decisionale multicriterio (MCDM – Multi Criteria Decision Making), è possibile classificare i metodi esistenti in **tre principali famiglie** teoriche, a seconda dell'approccio adottato per confrontare e aggregare le alternative decisionali.

1. **Multiple Attribute Utility Theory (MAUT)**: Questa famiglia di metodi si fonda sull'assunzione che esista una funzione di utilità aggregata in grado di rappresentare le preferenze del decisore. Le preferenze vengono modellate tramite una funzione additiva o più in generale separabile, permettendo di assegnare un punteggio globale a ciascuna alternativa. Tra i metodi rappresentativi di questa famiglia si colloca **UTA (UTilités Additives)**, che costruisce la funzione di utilità a partire da confronti ordinali tra alternative.

2. **Outranking Methods**: Questi metodi non cercano necessariamente di aggregare tutti i criteri in un unico valore numerico, ma si basano invece sul confronto diretto tra coppie di alternative. L'obiettivo è determinare se un'alternativa "sovrasta" (outranks) un'altra secondo una serie di regole e soglie concordate. I metodi più noti appartenenti a questa categoria includono **ELECTRE**, **CODASID** e **TOPSIS**. Questi approcci sono particolarmente utili in presenza di criteri conflittuali o difficilmente aggregabili, e trovano ampio impiego in contesti decisionali reali complessi.

3. **Interactive Methods**: Questa categoria include metodi in cui il decisore è coinvolto attivamente durante il processo iterativo di analisi, fornendo preferenze progressivamente e adattando il modello di decisione in tempo reale. L'interazione continua consente di migliorare la qualità del processo decisionale, specialmente nei casi in cui le preferenze del decisore non siano chiaramente definite ex ante o siano soggette a revisione.

Queste tre famiglie coprono un ampio spettro di approcci e strumenti utilizzabili nell'analisi multicriterio, e la loro scelta dipende dalla natura del problema, dalla disponibilità di informazioni sulle preferenze e dal grado di partecipazione desiderato da parte del decisore.

7.5 Il metodo UTA

Un numero considerevole di lavori è stato pubblicato sul tema della **Multi Attribute Utility Theory (MAUT)**, ampiamente utilizzata nell'ambito dell'**analisi decisionale** (Decision Analysis). Tutti i metodi appartenenti a questa famiglia si basano su una comune assunzione teorica: **l'esistenza di una funzione di utilità additiva**.

7.5.1 La funzione di utilità additiva (Additive Utility Function)

Come già discusso nelle sezioni precedenti, nei problemi di tipo **Multi Criteria Decision Making (MCDM)** si assume di dover operare una scelta tra un insieme di azioni A, dette anche **alternative**, **design**, o **decisioni candidate**, ciascuna valutabile secondo un insieme di criteri $g = (g_1, \dots, g_k)$.

L'idea di poter aggregare tutti questi criteri in un'unica espressione sintetica – cioè una **funzione di utilità complessiva** – è stata a lungo predominante tra i metodi classici di risoluzione dei problemi decisionali [6]. In questo contesto, si scrive:

$$U(g) = U(g_1, \dots, g_k)$$

Introducendo alcune nozioni fondamentali di teoria delle preferenze, si definiscono:

- P: **relazione di preferenza stretta** (strict preference relation),
- I: **relazione di indifferenza** (indifference relation),

le quali conducono alle seguenti condizioni di coerenza per la funzione di utilità:

$$U[g(a)] > U[g(b)] \Leftrightarrow aPbU[g(a)] = U[g(b)] \Leftrightarrow aIb$$

Da queste si ricava una relazione $R = P \cup I$, definita in letteratura come **relazione d'ordine debole** [3].

La funzione di utilità è detta **additiva** se può essere scritta come:

$$U(g) = \sum_{i=1}^{k} u_i\,(g_i)$$

dove ciascuna $u_i(g_i)$ rappresenta la **funzione di utilità marginale** relativa al criterio g_i.

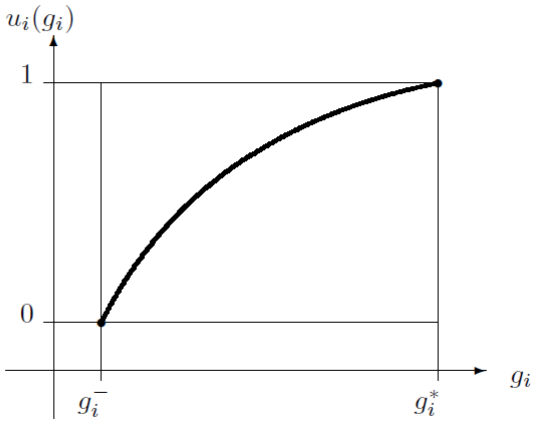
Nelle applicazioni pratiche, è frequente **normalizzare** la funzione di utilità. Sotto l'ipotesi (comune) che le utilità marginali siano **monotone crescenti** in ciascun criterio, si arriva alla rappresentazione schematica riportata in **Figura 1.2**.

Con questa impostazione, la normalizzazione si traduce nei seguenti vincoli:

$$\sum_{i=1}^{k} u_i\,(g_i^*) = 1 \quad ; \quad u_i(g_i^-) = 0 \quad \forall i$$

Dove:

- g_i^-: valore minimo di riferimento per il criterio g_i,
- g_i^*: valore massimo di riferimento per lo stesso criterio.



7.5.2 Sviluppo del metodo UTA

Sia $G_i = [g_i^-, g_i^*]$, per ogni $i = 1, \dots, k$, l'intervallo di valori ammissibili per ciascun criterio g_i. Si suppone che un sottoinsieme di design (o alternative), reali o immaginarie, detto A', sia stato **ordinato dal decisore** secondo le sue preferenze complessive, utilizzando la relazione $R = P \cup I$ definita precedentemente.

Il metodo UTA si articola in due **fasi principali**:

Step 1 – Costruzione di una funzione di utilità ottimale $U^*(g)$

Si assume che ogni funzione u_i sia **lineare a tratti** (*piecewise linear*). L'intervallo G_i viene suddiviso in $\alpha_i - 1$ sottointervalli equidistanti. I punti finali di tali intervalli sono calcolati come:

$$g_i^j = g_i^- + \frac{j-1}{\alpha_i - 1}(g_i^* - g_i^-) \quad \text{per } j = 1, \dots, \alpha_i$$

Le variabili da stimare sono le utilità marginali nei punti g_i^j, ovvero $u_i(g_i^j)$. Per un'alternativa a tale che $g_i(a) \in [g_i^j, g_i^{j+1}]$, si può interpolare linearmente:

$$u_i[g_i(a)] = u_i(g_i^j) + \frac{g_i(a) - g_i^j}{g_i^{j+1} - g_i^j} \cdot \left[u_i(g_i^{j+1}) - u_i(g_i^j)\right]$$

Pertanto, utilizzando le equazioni (1.16) e (1.17), si definisce una funzione approssimata:

$$U'[g(a)] = \sum_{i=1}^{k} u_i\,[g_i(a)] + \sigma(a) \quad \forall a \in A'$$

dove:

- $\sigma(a) > 0$ rappresenta un **errore potenziale** per l'alternativa a.

In forma semplificata:

$$U[g(a)] = \sum_{i=1}^{k} u_i\,[g_i(a)]$$

È utile riscrivere le disuguaglianze espresse dalle preferenze così:

$$U'[g(a)] - U'[g(b)] > \delta \quad \text{se } aPb$$

$$U'[g(a)] - U'[g(b)] = 0 \quad \text{se } aIb$$

dove $\delta > 0$ è un valore piccolo che garantisce **separazione numerica** tra le classi. Siskos e Jacquet-Lagrèze [3] consigliano $\delta \in \left[\frac{1}{10Q}, \frac{1}{Q}\right]$, con Q numero di classi di indifferenza. In pratica, è spesso adottato $\delta = 0.01$.

Per minimizzare la **deviazione complessiva** si utilizza la seguente **funzione obiettivo lineare**:

$$F = \sum_{a \in A'} \sigma\,(a)$$

Eventualmente si possono **pesare gli errori** con coefficienti $p(a)$ per tenere conto del diverso grado di affidabilità nelle classificazioni:

$$F = \sum_{a \in A'} p\,(a) \cdot \sigma(a)$$

Il problema di programmazione lineare da risolvere (PL1) è:

[min] $$F = \sum_{a \in A'} \sigma\,(a)$$

soggetto a:

$$\sum_{i=1}^{k} u_i\,[g_i(a)] - \sum_{i=1}^{k} u_i\,[g_i(b)] + \sigma(a) - \sigma(b) > \delta \text{se } aPb$$

$$\sum_{i=1}^{k} u_i\,[g_i(a)] - \sum_{i=1}^{k} u_i\,[g_i(b)] + \sigma(a) - \sigma(b) = 0 \text{se } aIb$$

$$u_i(g_i^{j+1}) - u_i(g_i^{j}) > 0 \qquad \forall i, j$$

$$\sum_{i=1}^{k} u_i\,(g_i^{*}) = 1 \quad ; \quad u_i(g_i^{-}) = 0$$

$$u_i(g_i^{j}) > 0 \quad ; \quad \sigma(a) > 0 \quad \forall a \in A'$$

con $i = 1, \dots, k$, e $j = 1, \dots, \alpha_i - 1$.

Step 2 – Analisi post-ottimizzazione

Poiché il problema (1.26) può avere **più soluzioni ottimali**, è utile esplorare lo **spazio delle soluzioni efficienti**.

Fissata F^*, cioè la **soluzione ottima** del problema (1.26), si considera una soglia $\kappa(F^*)$, piccola frazione di F^*, per consentire una **tolleranza**. Si esplorano allora le soluzioni che soddisfano:

$$F \leq F^* + \kappa(F^*)$$

ovvero, riformulando:

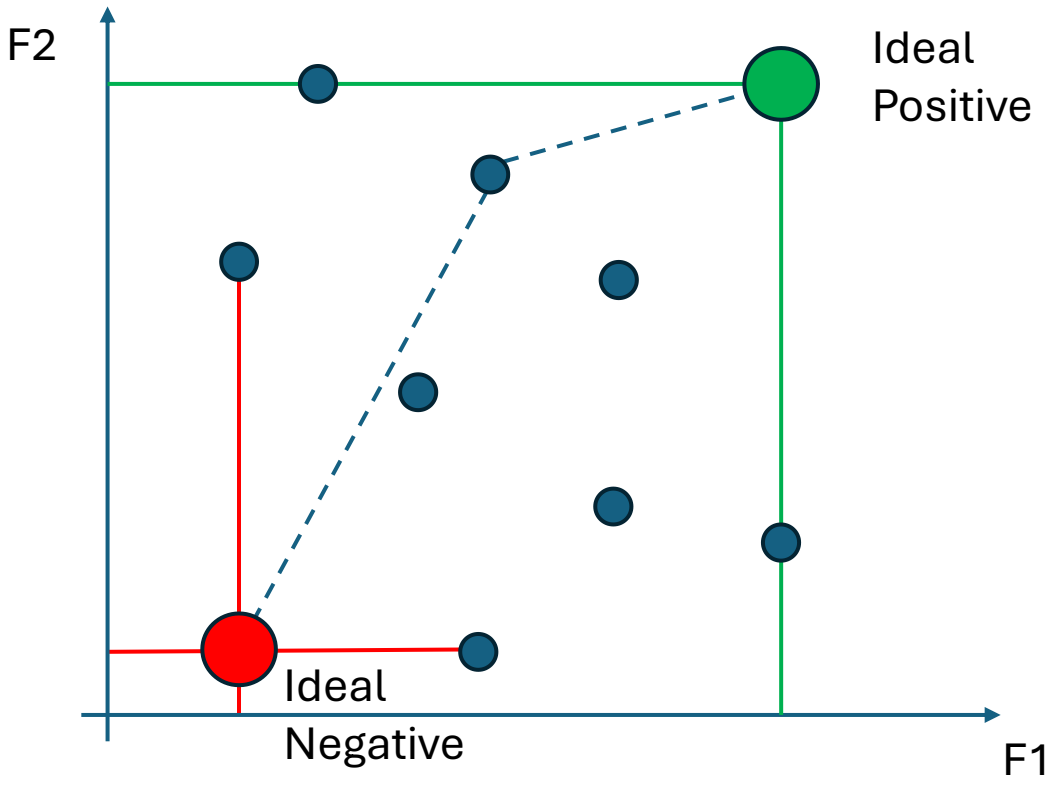
$$-\sum_{a \in A'} \sigma\,(a) > -[F^* + \kappa(F^*)]$$

Questa ulteriore disuguaglianza viene **aggiunta al sistema di vincoli** del problema (1.26) per generare **soluzioni alternative ammissibili**, che permettano di analizzare la **stabilità del risultato** ottenuto.

7.6 L'algoritmo TOPSIS

L'algoritmo **TOPSIS** (*Technique for Order Preference by Similarity to Ideal Solution*), proposto da **Hwang e Yoon**, appartiene alla famiglia degli **outranking methods**, ed è uno dei più noti ed efficaci strumenti per affrontare problemi di decisione multicriterio.

L'idea centrale alla base di TOPSIS è semplice e intuitiva: **la soluzione migliore è quella che si trova contemporaneamente alla distanza minima dall'alternativa ideale e alla massima distanza da quella peggiore (o nadir)**. In altre parole, si cerca il compromesso ottimale tra **prossimità al meglio** e **lontananza dal peggio**.



L'applicazione del metodo richiede come **input**:

- una **matrice decisionale** che raccoglie le performance di ogni alternativa rispetto ai vari criteri, e
- un **vettore di pesi** che esprima l'importanza relativa attribuita a ciascun criterio dal decisore, L'algoritmo TOPSIS si sviluppa attraverso i seguenti **sei step operativi**:

Step 1 – Costruzione della matrice decisionale normalizzata

Dato il problema formulato nella forma tabellare, si procede a costruire una **matrice normalizzata**, i cui elementi z_{ij} sono definiti come:

$$z_{ij} = \frac{y_{ij}}{\sqrt{\sum_{i=1}^{n} y_{ij}^2}} \quad \text{con } i = 1, \dots, n;\ j = 1, \dots, k$$

Questa normalizzazione consente di eliminare eventuali differenze di scala tra i criteri, rendendo i dati confrontabili.

Step 2 – Calcolo della matrice pesata

Si costruisce quindi la **matrice normalizzata pesata**, moltiplicando ciascun elemento z_{ij} per il peso corrispondente w_j del criterio:

$$x_{ij} = w_j \cdot z_{ij} \quad \text{con } i = 1, \dots, n;\ j = 1, \dots, k$$

In questo modo, si tiene conto del contributo relativo di ogni criterio nella valutazione complessiva.

Step 3 – Definizione dei punti ideali

Si identificano due punti di riferimento:

- **Punto ideale positivo** a^*, ovvero la combinazione ottimale di valori sui criteri;
- **Punto ideale negativo (nadir)** a^-, rappresentante la combinazione peggiore.

Essi sono calcolati come segue:

$$a^* = \left\{ \max_i x_{ij} \text{ se } j \in J \; ; \; \min_i x_{ij} \text{ se } j \in \hat{J} \right\} = \{x_1^*, x_2^*, \dots, x_k^*\}$$

$$a^- = \left\{ \min_i x_{ij} \text{ se } j \in J \; ; \; \max_i x_{ij} \text{ se } j \in \hat{J} \right\} = \{x_1^-, x_2^-, \dots, x_k^-\}$$

Dove:

- J è l'insieme degli indici dei criteri da **massimizzare** (ad es. benefici),
- $\hat{J}$ è l'insieme degli indici dei criteri da **minimizzare** (ad es. costi).

Step 4 – Calcolo delle distanze

Per ogni alternativa a_i, si calcola:

- la **distanza euclidea** dal punto ideale a^*:

$$S_i^* = \sqrt{\sum_{j=1}^{k} (x_{ij} - x_j^*)^2} \quad \text{per } i = 1, \dots, n$$

- e la distanza dal punto nadir a^-:

$$S_i^- = \sqrt{\sum_{j=1}^{k} (x_{ij} - x_j^-)^2} \quad \text{per } i = 1, \dots, n$$

Queste distanze rappresentano rispettivamente **quanto l'alternativa si discosta dall'ottimo** e **quanto si allontana dal pessimo**.

Step 5 – Calcolo della vicinanza relativa

Si calcola per ciascuna alternativa la cosiddetta **vicinanza relativa al punto ideale**, espressa come:

$$C_i^* = \frac{S_i^-}{S_i^- + S_i^*} \quad \text{per } i = 1, \dots, n$$

Questa quantità è compresa tra 0 e 1, e misura **quanto un'alternativa è vicina all'ideale perfetto**, tenendo conto anche della distanza dal nadir.

Step 6 – Ordinamento delle alternative

Infine, si ordina l'insieme delle alternative in base ai valori C_i^*: maggiore è tale valore, migliore è l'alternativa. In particolare, se $C_i^* > C_j^*$, allora si dice che **l'alternativa a_i è preferibile (o "supera") a_j**.

L'efficacia dell'algoritmo TOPSIS risiede nella sua **semplicità implementativa**, nella **chiarezza dei risultati** e nella **capacità di gestire più criteri sia da massimizzare che da minimizzare**. È ampiamente utilizzato in settori quali l'ingegneria, la logistica, la gestione dei fornitori, la pianificazione industriale e l'analisi ambientale, tra gli altri.

8 Superfici di risposta

Da un punto di vista ingegneristico, nei problemi di ottimizzazione, il flusso dei dati segue una logica ben definita: l'obiettivo è individuare la combinazione di variabili progettuali che consenta di ottenere le migliori prestazioni possibili, secondo determinati criteri o obiettivi. Tuttavia, il nodo critico si trova quasi sempre nel momento in cui il sistema deve "valutare" ogni configurazione: questa valutazione viene normalmente affidata a un **solver numerico**, che spesso richiede tempi di calcolo elevati.

Infatti, in molti casi reali, i problemi da affrontare sono altamente complessi, e il tempo necessario per passare dalla definizione di una configurazione alla conoscenza delle sue prestazioni può essere molto lungo. L'ottimizzazione, di per sé, è un processo iterativo che richiede di esplorare molte configurazioni diverse: di conseguenza, il tempo complessivo di calcolo può diventare rapidamente ingestibile.

Per affrontare questo problema, la pratica ingegneristica propone diverse soluzioni, che vanno dall'uso di algoritmi più efficienti alla modifica stessa del modo in cui si interroga il modello.

Efficienza degli Algoritmi di ottimizzazione

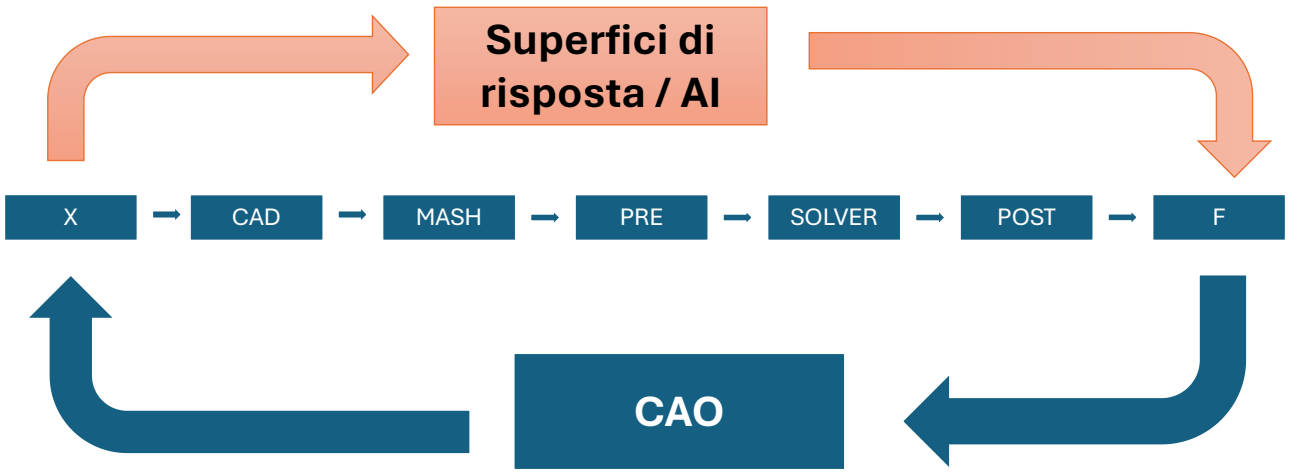
Una prima strategia consiste nel migliorare l'efficienza dell'**algoritmo di ottimizzazione**. In altre parole, si cerca di raggiungere i punti ottimali della funzione obiettivo (o degli obiettivi) riducendo al minimo il numero di valutazioni necessarie. Questo approccio, tuttavia, richiede competenze sia nella scelta dell'algoritmo più adatto al problema (per esempio algoritmi genetici, algoritmi di gradiente, swarm intelligence, ecc.), sia nella sua implementazione. Ancora più importante, è fondamentale che il problema sia stato "preparato" correttamente: cioè che si sia svolta un'**analisi statistica preliminare** per eliminare variabili inutili, ridurre la dimensionalità del problema e comprendere l'importanza relativa degli obiettivi. Questo lavoro può essere supportato dalla costruzione di un **DOE** (Design of Experiments), che permette di esplorare in modo sistematico lo spazio delle variabili e fornire le basi per un'ottimizzazione più mirata.

Parallelizzazione

Un'altra possibilità, che offre vantaggi significativi soprattutto in ambienti con risorse hardware adeguate, è la **parallelizzazione del calcolo**. In pratica, invece di valutare una configurazione alla volta con un singolo processore, è possibile assegnare più configurazioni a più processori (per esempio n configurazioni a n processori) e ottenere così tutte le prestazioni simultaneamente. Questo consente di ridurre drasticamente i tempi, anche se non elimina la dipendenza dal tempo richiesto dal singolo solver.

Superfici di risposta

Una terza strategia, forse la più innovativa ma anche la più delicata da gestire, è la costruzione di un **modello analitico approssimato** del solver. In questo caso, invece di interrogare ogni volta il solver vero e proprio, si costruisce una funzione $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ che lo approssima. Questo modello, spesso definito **superficie di risposta** o **surrogate model**, permette di stimare le prestazioni di una configurazione in tempi estremamente ridotti rispetto alla funzione reale. Tuttavia, l'utilizzo di questo approccio introduce inevitabilmente un **errore**: l'ottimo individuato sulla superficie approssimante può non coincidere con l'ottimo reale. Se la superficie non è sufficientemente accurata, il rischio è di ottenere una soluzione che appare ottima in teoria, ma che in realtà non lo è.



Per limitare questo rischio, si ricorre spesso a strategie ibride. Un esempio è dato dall'integrazione tra il solver e la superficie di risposta all'interno di un algoritmo evolutivo, come quello genetico: una parte degli individui (ad esempio il 70%) viene valutata usando la funzione approssimante, mentre il restante 30% viene calcolato con il solver reale. In questo modo si riesce a ridurre sensibilmente il tempo totale di calcolo, ma allo stesso tempo si mantiene un controllo sulla qualità delle soluzioni trovate, evitando che l'approssimazione porti fuori strada.

Per poter costruire una superficie di risposta affidabile è però necessario disporre di un **data set di soluzioni conosciute (training set)**: ovvero un insieme di configurazioni per cui siano già noti i valori delle variabili di input e le corrispondenti prestazioni. Questo set può essere ottenuto in due modi: il primo consiste nell'eseguire un DOE mirato, e utilizzare quei dati per costruire il modello; il secondo prevede una breve fase iniziale di ottimizzazione direttamente con il solver, in modo da accumulare informazioni utili a definire la superficie di risposta.

Infine, è fondamentale valutare la **qualità** della superficie di risposta sul database **validation set**. Tra le caratteristiche più importanti da monitorare vi sono l'errore medio di approssimazione (per esempio tramite RMSE), la capacità predittiva su dati non inclusi nell'addestramento (validazione incrociata), e la robustezza del modello. Solo con un modello approssimante ben costruito e validato è possibile ottenere un reale vantaggio in termini di tempo, senza compromettere l'affidabilità delle soluzioni progettuali finali.

8.1 Bontà delle superfici di risposta

Per poter sfruttare efficacemente una superficie di risposta all'interno di un processo di ottimizzazione, è essenziale valutarne con attenzione la **qualità**. Una superficie di risposta non è altro che un modello matematico approssimato, e come tale introduce inevitabilmente un certo grado di errore rispetto alla funzione reale. Per questo motivo è importante analizzare alcune caratteristiche fondamentali che ne determinano l'affidabilità.

8.1.1 Accuratezza

L'accuratezza è la proprietà più importante di una superficie di risposta. Essa indica quanto il modello approssimante si avvicina ai valori reali restituiti dal solver. A differenza dell'accuratezza di un algoritmo di ottimizzazione, che può essere difficile da quantificare a priori, l'accuratezza della superficie di risposta è effettivamente **misurabile**, grazie al fatto che il modello viene costruito a partire da un data base di soluzioni note.

In pratica, il processo prevede la suddivisione del data base in due sottoinsiemi:

- un **training set**, solitamente pari all'80% delle configurazioni note, usato per costruire il modello approssimante;
- un **validation set**, pari al restante 20%, che viene utilizzato per testare il modello su dati non "visti" durante l'addestramento.

Supponiamo di aver creato un modello approssimante f_RS: $\mathbb{R}^n \rightarrow \mathbb{R}^m$ a partire dal training set. A questo punto si procede a valutare il modello sul validation set. Per ciascuna configurazione presente nel validation set, si calcolano le prestazioni stimate dal modello approssimante, e le si confronta con i valori reali (cioè quelli forniti in precedenza dal solver).

Questo confronto permette di calcolare un **errore di previsione**, ad esempio tramite la somma degli scarti assoluti o quadrati tra valori stimati e valori reali:

$$E = \sum_{i \in V_{set}} \| f^{RS}(x_i) - f(x_i) \|$$

Diverse superfici di risposta, costruite con differenti metodi o iperparametri, daranno inevitabilmente valori di errore diversi. Quella che presenta l'errore minore sul validation set viene considerata **più accurata** e, quindi, più adatta all'uso in fase di ottimizzazione.

8.1.2 Tempi di calcolo

Oltre all'accuratezza, è necessario considerare anche i **tempi di calcolo** richiesti dalla superficie di risposta. In generale, una delle principali motivazioni per l'uso di un modello approssimante è proprio la **drastica riduzione dei tempi di valutazione**, rispetto all'utilizzo diretto del solver. Tuttavia, non tutti i modelli approssimanti sono uguali in termini di efficienza computazionale.

Alcuni modelli (come le reti neurali complesse o i metodi basati su kriging) possono richiedere tempi non trascurabili, soprattutto in fase di addestramento o quando la dimensionalità del problema è elevata. È quindi importante valutare **a priori** se il guadagno in tempo è sufficientemente significativo da giustificare l'uso della superficie di risposta.

In alcuni casi, ad esempio, il tempo di costruzione del modello può essere superiore al tempo che si impiegherebbe usando direttamente il solver per un numero contenuto di valutazioni. Solo un'analisi comparativa può stabilire se conviene davvero adottare un surrogate model.

8.1.3 Parametri di configurazione (Setting)

Un altro aspetto spesso sottovalutato ma cruciale riguarda i **parametri di setting** della superficie di risposta. Ogni metodologia di approssimazione (che si tratti di regressione polinomiale, SVM, reti neurali, alberi di decisione, ecc.) offre all'utente una serie di parametri regolabili, i cosiddetti **iperparametri**.

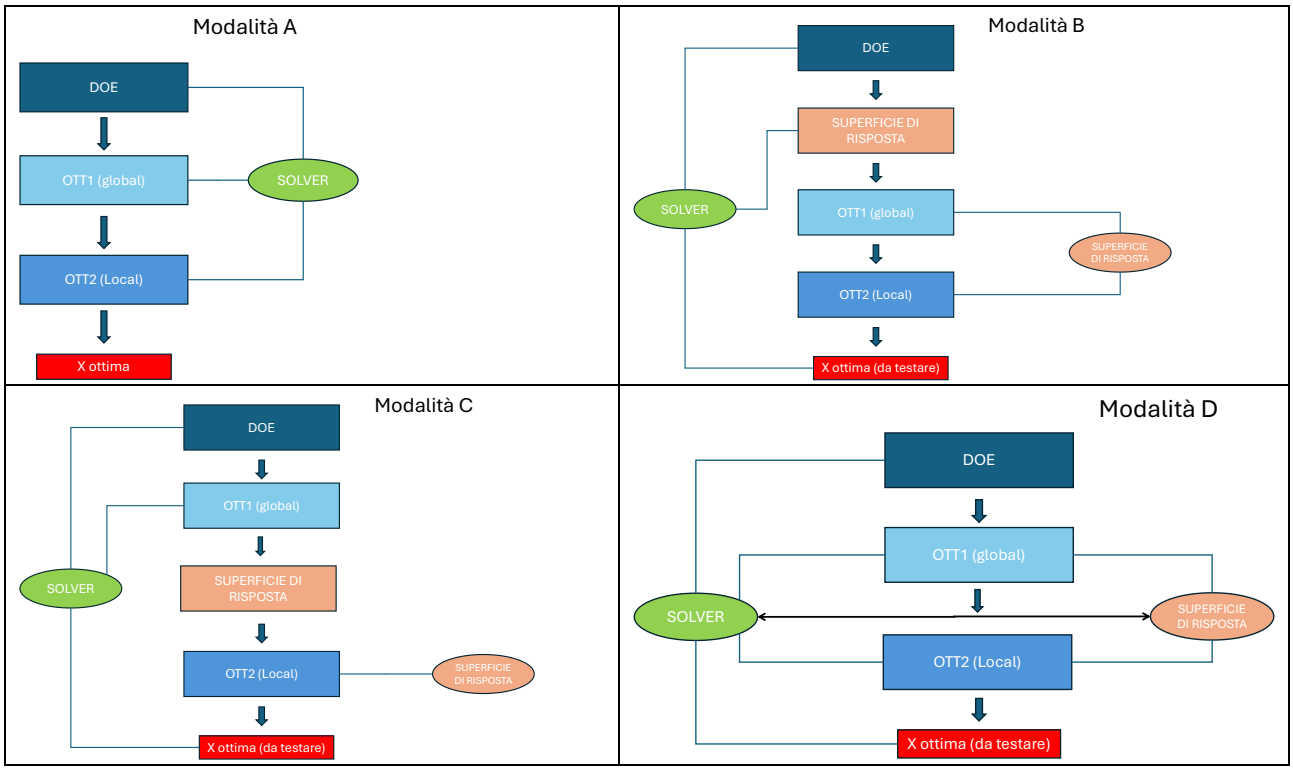
Da un lato, questa flessibilità costituisce un vantaggio, perché consente di adattare il modello al problema specifico e di migliorarne la precisione tramite opportune regolazioni. Dall'altro, però, rappresenta anche un punto critico, perché l'**accuratezza del modello dipende fortemente dalla corretta impostazione** di questi parametri.

Parametri come il grado del polinomio, il numero di neuroni in una rete, la funzione di attivazione, la profondità di un albero o il kernel utilizzato, possono cambiare radicalmente il comportamento del modello. Una scelta non oculata può condurre a **overfitting** (modello troppo preciso sul training set, ma poco generalizzabile) oppure a **underfitting** (modello troppo grezzo per catturare le tendenze reali).

È quindi essenziale disporre di competenze tecniche nella modellazione, oppure affidarsi a processi automatizzati di tuning (come la grid search o l'ottimizzazione bayesiana) per selezionare la configurazione ottimale.

8.2 Integrazione delle Superfici di Risposta con gli algoritmi di ottimizzazione

L'integrazione tra superfici di risposta e processo di ottimizzazione può essere realizzata attraverso approcci differenti; la scelta della strategia più appropriata è responsabilità del team di progettazione, che deve perseguire un bilanciamento ottimale tra efficienza computazionale e accuratezza della soluzione finale.



Negli schemi precedentemente esposti sono illustrate quattro differenti tipologie di integrazione, a partire dalla configurazione di base (A), in cui non è contemplato l'impiego di superfici di risposta.

È opportuno sottolineare che, nell'utilizzo delle response surfaces, la validazione finale della soluzione deve essere sempre condotta mediante il solver. Una valutazione esclusivamente basata sulla superficie di risposta, infatti, potrebbe introdurre significativi errori di approssimazione nelle prestazioni stimate.

Si ricorda, inoltre, che la definizione di una superficie di risposta presuppone necessariamente la disponibilità di un opportuno training set. In tale prospettiva, risulta evidente come, passando dallo schema B allo schema D, la costruzione del training set venga progressivamente integrata all'interno del processo di ottimizzazione. Tale integrazione consente di raffinare l'accuratezza della superficie di risposta proprio nelle regioni di maggiore rilevanza, ossia quelle associate agli estremi di prestazione.

Come verrà discusso in seguito, questo approccio si dimostra particolarmente efficiente quando applicato nell'ambito dei processi gaussiani.

8.3 Tecnologie di superfici di risposta

Nel contesto della progettazione ingegneristica CAO, esistono **numerose metodologie per la costruzione di superfici di risposta**, che vanno dai modelli più semplici e facilmente implementabili fino a quelli più avanzati, basati su concetti moderni di **intelligenza artificiale** e **apprendimento automatico**.

La scelta del modello da utilizzare dipende da diversi fattori: la quantità e la qualità dei dati disponibili, la complessità del fenomeno da approssimare, la necessità di interpretabilità, e naturalmente i tempi computazionali a disposizione. In generale, si può immaginare uno spettro continuo di modelli che va da **approssimazioni locali e analitiche**, come le **serie di Taylor**, fino a **modelli probabilistici non parametrici**, come i **processi gaussiani**, passando per metodi basati su **statistica multivariata**, **ricerca nei dati** e **reti neurali**.

In questa dispensa ci concentreremo sull'analisi di **quattro diverse metodologie**, scelte perché rappresentano, nel complesso, le principali famiglie di tecniche utilizzate in ambito ingegneristico per la costruzione di superfici di risposta. Esse offrono un'ampia panoramica sia in termini di struttura teorica che di applicabilità pratica.

Nello specifico, studieremo:

Serie di Taylor

Le **serie di Taylor** rappresentano il metodo più semplice e analiticamente trasparente per approssimare una funzione locale. Basandosi sullo sviluppo in serie attorno a un punto noto, questo approccio utilizza le derivate della funzione per costruire una stima polinomiale. Sebbene siano limitate a zone in cui la funzione è sufficientemente regolare e differenziabile, le serie di Taylor offrono una buona approssimazione locale e risultano utili nei casi in cui si vogliano ottenere superfici interpretabili, locali e a basso costo computazionale.

Algoritmo k-Nearest Neighbors (k-NN)

Il metodo **k-Nearest Neighbors** è una tecnica semplice ma potente basata sulla nozione di vicinanza tra punti nello spazio delle variabili. Per stimare l'uscita associata a una nuova configurazione, il metodo k-NN cerca i **k punti più vicini** nel data-set di soluzioni conosciute e calcola una media (o media pesata) dei valori corrispondenti. Non richiede alcuna assunzione sulla forma della funzione e si adatta bene a problemi con forte località. Tuttavia, la sua efficacia decresce in spazi ad alta dimensionalità (fenomeno della "maledizione della dimensionalità") e la sua performance può dipendere fortemente dalla scelta della metrica di distanza e dal valore di k.

Reti neurali artificiali (Artificial Neural Networks)

Le **reti neurali** sono modelli ispirati al funzionamento del cervello umano e sono oggi ampiamente utilizzate in ambito ingegneristico, specialmente quando si hanno a disposizione grandi quantità di dati e si devono approssimare fenomeni altamente non lineari. Composte da uno o più strati di neuroni artificiali collegati tra loro, le reti neurali imparano una mappatura tra input e output attraverso un processo iterativo di **addestramento**. Sebbene possano fornire superfici di risposta estremamente flessibili e potenti, presentano anche alcune criticità: richiedono tempo e risorse per essere addestrate correttamente, possono soffrire di overfitting se non regolarizzate adeguatamente, e sono in genere difficili da interpretare.

Processi gaussiani (Gaussian Processes)

I **processi gaussiani** rappresentano uno dei metodi più avanzati per la costruzione di superfici di risposta. Si tratta di modelli probabilistici non parametrici che, a partire da un insieme di dati noti, forniscono non solo una stima puntuale dell'uscita ma anche un **intervallo di confidenza** sulla predizione. Questo renderà possibile una innovativa metodologia di Design of Experiments. Dal punto di vista computazionale, però, i processi gaussiani possono diventare onerosi per data set molto estesi, in quanto richiedono l'inversione di matrici dense. Nonostante ciò, la loro capacità di adattarsi bene ai dati e di quantificare l'incertezza li rende estremamente apprezzati in contesti ingegneristici di alta precisione.

8.4 Superfici di risposta tramite serie di Taylor

La costruzione di una superficie di risposta può essere effettuata anche mediante lo **sviluppo in serie di Taylor**. In questo caso, si rappresenta la funzione di interesse come una **espansione attorno a un punto noto** x_0. Nella pratica ingegneristica, questa serie viene **troncata** generalmente ai primi termini, al fine di contenere la complessità del modello e ridurre il costo computazionale.

L'idea di fondo è approssimare la funzione $f(x)$, che può rappresentare una prestazione o una funzione obiettivo, tramite una forma polinomiale basata su derivate calcolate nel punto x_0. In tal modo, si costruisce una superficie approssimante $a(x)$ che restituisce valori simili a quelli della funzione reale in un intorno di x_0.

A seconda del tipo di sviluppo e della struttura della funzione, si possono distinguere **diverse varianti** della superficie di risposta basata su serie di Taylor:

*Variante **lineare***

È la forma più semplice e corrisponde alla **troncatura al primo ordine**:

$$a(x) = f(x_0) + \sum_{i=1}^{m} \frac{\partial f}{\partial x_i}\bigg|_{x_0} (x^i - x_0^i)$$

Questa approssimazione assume che la funzione sia sufficientemente regolare e che l'intorno di validità dell'espansione sia piccolo. Il modello risultante è interpretabile e computazionalmente leggero, ma può risultare poco accurato in presenza di fenomeni non lineari.

*Variante **reciproca***

Qui si considera una trasformazione dei termini dell'espansione, introducendo un rapporto tra coordinate:

$$a(x) = f(x_0) + \sum_{i=1}^{m} \frac{\partial f}{\partial x_i}\bigg|_{x_0} (x^i - x_0^i) \cdot \frac{x_0^i}{x^i}$$

Questo tipo di superficie di risposta può risultare utile in presenza di variabili che agiscono su **scale molto diverse** o che mostrano **comportamenti inversamente proporzionali**, come in alcuni problemi termodinamici o meccanici.

*Variante **non lineare con parametro** r*

Una generalizzazione più flessibile è data dalla forma:

$$a(x) = f(x_0) + \frac{1}{r}\sum_{i=1}^{m} (x_0^i)^{1-r} (x^i)^{r-1} \frac{\partial f}{\partial x_i}\bigg|_{x_0} (x^i - x_0^i)$$

In questo caso compare un **parametro** r che modula il comportamento dell'interpolazione. Questa formulazione consente di **modellare fenomeni non lineari** con maggiore precisione rispetto al caso lineare, mantenendo comunque una struttura analitica controllata.

Il parametro r non deriva da considerazioni teoriche legate alla funzione da approssimare, ma è a tutti gli effetti un **parametro di setting**. La sua scelta influisce direttamente sull'**accuratezza della superficie di risposta**. In pratica, si può **selezionare il valore ottimale di** r tramite una procedura di validazione: si costruisce la superficie con diversi valori di r, si valuta l'errore sul **Validation Set**, e si sceglie il valore che minimizza l'errore.

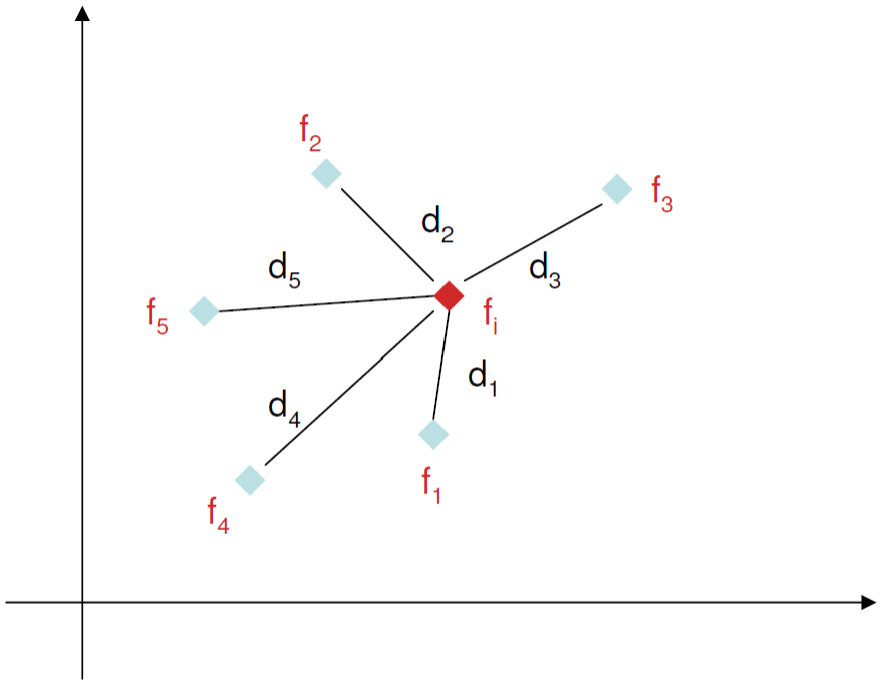
Questo approccio consente di adattare la superficie alle caratteristiche del problema, ma comporta un **aumento dei tempi di calcolo**: per ogni valore di r considerato, è necessario ricostruire la superficie e calcolare l'errore di approssimazione, rendendo il processo più oneroso.

In sintesi, la serie di Taylor rappresenta uno strumento semplice ma efficace per la costruzione di superfici di risposta, soprattutto nelle prime fasi dell'ottimizzazione o quando si cerca un modello interpretabile. Tuttavia, è fondamentale ricordare che la sua **validità è locale**, e la scelta della variante più adatta – lineare, reciproca o non lineare – dipende dal problema specifico e dalla precisione richiesta.

8.5 Superficie di risposta basata sull'algoritmo K-Nearest Neighbors

L'algoritmo **K-Nearest Neighbors** (K-NN) rappresenta una delle tecniche più semplici ed efficaci per costruire una superficie di risposta, soprattutto quando si ha a disposizione un **numero limitato di dati** ma si devono trattare problemi con **numerose variabili**.

L'idea alla base di K-NN è estremamente intuitiva: per stimare il valore della funzione in un nuovo punto x_0, si guarda a cosa succede "nei dintorni", ovvero **nei punti più vicini** presenti nel **data-base delle soluzioni conosciute**.



Supponiamo di voler approssimare una funzione:

$$f: \mathbb{R}^2 \to \mathbb{R}$$

con due variabili in ingresso, x^1 e x^2, e di disporre di un database contenente 4 configurazioni già note (per esempio: $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$) con i rispettivi valori della funzione $f(x^{(i)})$ calcolati tramite solver.

Ora desideriamo conoscere il valore di $f(x_0)$, ma **senza interrogare il solver**, bensì usando l'approssimazione basata sui dati noti.

Per prima cosa si calcola la **distanza** tra il punto x_0 e ciascun punto noto del data-base. Una distanza comunemente utilizzata è quella **euclidea**:

$$d_i = \sqrt{\sum_{j=1}^{n} (x_0^j - x_i^j)^2}$$

Dove n è il numero di variabili, x_i è una configurazione del data-base, e x_0 è la configurazione di interesse.

Dopo aver calcolato le distanze, si selezionano i **K punti più vicini** a x_0 (cioè i "K-Nearest Neighbors"). Il valore della funzione nel punto x_0 viene calcolato come **media pesata** dei valori della funzione nei punti vicini, pesati in base alla distanza:

$$f(x_0) = \frac{\sum_{i=1}^{K} \frac{f(x_i)}{d_i^\alpha}}{\sum_{i=1}^{K} \frac{1}{d_i^\alpha}}$$

In questa formula:

- K rappresenta il numero di punti più vicini considerati;
- α è un **parametro di setting** che controlla **quanto influisce la distanza**:
    - Se $\alpha = 1$, la funzione è lineare rispetto alla distanza;
    - Se $\alpha = 2$, il contributo dei punti più lontani cala molto più rapidamente (comportamento quadratico).

Maggiore è la distanza d_i, **minore è il peso** assegnato a $f(x_i)$. Questo permette di privilegiare le informazioni provenienti da configurazioni **vicine a quella di interesse**, aumentando la precisione locale dell'approssimazione.

L'algoritmo K-NN ha due parametri fondamentali, entrambi da scegliere con attenzione:

- K: è il numero di punti del data-base considerati per il calcolo del valore della funzione.
    - Se K è **troppo piccolo**, la superficie sarà **molto fluttuante**, sensibile al rumore nei dati.
    - Se K è **troppo grande**, la superficie diventa **molto liscia**, ma potrebbe **perdere precisione locale**, appiattendo eventuali picchi della funzione reale.
- α: regola l'influenza della distanza nella media pesata.
    - Valori piccoli (vicini a 1) rendono la funzione più **"distribuita"**, cioè ogni vicino contribuisce in modo simile.

- Valori più grandi (come 2 o 3) **penalizzano fortemente i punti lontani**, concentrando la stima su quelli più vicini.

Entrambi questi parametri **non derivano da una teoria generale**, ma sono scelti in base al problema specifico. È quindi prassi comune effettuare una **validazione empirica** (cross-validation), provando diversi valori di K e α e scegliendo quelli che **minimizzano l'errore** di approssimazione sul **Validation Set**.

Certo! Ecco la tua sezione riformulata in **stile discorsivo**, meno schematica e più adatta a una **dispensa fluida e argomentata**:

Uno dei principali punti di forza del metodo K-NN risiede nella sua **estrema semplicità di implementazione**. Non è necessario costruire modelli complessi né assumere una forma particolare per la funzione da approssimare: l'approccio si basa esclusivamente sui dati osservati e sul concetto di vicinanza tra punti. Proprio per questa ragione, risulta particolarmente indicato nei casi in cui si abbiano a disposizione **molte variabili ma un numero limitato di dati noti**, una situazione frequente nelle fasi preliminari della progettazione ingegneristica.

Inoltre, grazie alla sua struttura intuitiva, K-NN è uno strumento ideale per ottenere **stime rapide** delle prestazioni senza la necessità di avviare processi di ottimizzazione complessi o dispendiosi in termini di tempo. È quindi spesso utilizzato come metodo di base, utile per **una prima esplorazione dello spazio delle soluzioni**.

Tuttavia, il metodo presenta anche alcune **limitazioni importanti**. Innanzitutto, non può essere impiegato con variabili di tipo **di categoria**, in quanto il concetto di distanza – su cui si fonda l'intera logica del metodo – non è ben definito in questi casi. Un altro aspetto critico riguarda la **scelta della metrica**: la distanza Euclidea è la più comune, ma non sempre la più adatta. Metriche alternative, come quella di Manhattan, possono portare a risultati molto diversi e vanno scelte con cura in funzione del problema.

Dal punto di vista funzionale, K-NN **non restituisce una superficie analitica continua**. Si tratta infatti di un interpolatore basato esclusivamente sui dati esistenti: non esiste una funzione esplicita f(x)f(x) ricostruita, ma solo una procedura per stimare nuovi valori. Questo limita l'uso del metodo in contesti in cui sia necessario disporre di una formula chiusa o esportabile del modello.

Infine, il metodo soffre sensibilmente in **spazi ad alta dimensionalità**. All'aumentare del numero di variabili, il concetto stesso di "vicinanza" perde significato, e il metodo diventa progressivamente meno efficace, a meno di non intervenire con tecniche di riduzione dimensionale o filtraggio dei dati.

Il metodo K-Nearest Neighbors rappresenta il **livello base delle superfici di risposta**: semplice, efficace, e molto utile nei contesti in cui si hanno **poche informazioni numeriche** ma **molte variabili misurabili**. Sebbene non sia il più preciso tra i modelli disponibili, costituisce un ottimo punto di partenza per un'analisi esplorativa o per ottenere risultati rapidi senza dover addestrare modelli complessi.

8.6 Reti Neurali: introduzione

Nei capitoli precedenti si è messo in evidenza come gli algoritmi numerici di ottimizzazione possano rivelarsi strumenti estremamente utili nella progettazione ingegneristica e scientifica. Essi permettono infatti di affrontare problemi complessi, nei quali il numero di variabili è elevato, le funzioni obiettivo non sono espresse in forma analitica chiusa e i vincoli non sono sempre semplici da trattare. In tali contesti,

l'ottimizzazione numerica consente di esplorare lo spazio delle soluzioni possibili, identificando quelle più convenienti in termini di prestazioni, costi, tempi o altri criteri specifici del problema in esame.

Sono stati inoltre analizzati i principali campi di applicazione di questi metodi, mostrando come la scelta della tecnica più adatta dipenda strettamente dalla natura del problema proposto: esistono infatti algoritmi particolarmente adatti a problemi continui, altri che si prestano meglio a contesti discreti o combinatori, ed altri ancora che risultano efficaci in situazioni ibride.

Un aspetto fondamentale che è stato già messo in luce riguarda però le difficoltà pratiche nell'applicazione di tali algoritmi. L'uso di procedure numeriche di ottimizzazione, sebbene teoricamente efficace, si scontra spesso con la necessità di effettuare un numero molto elevato di simulazioni del processo o del sistema in esame. Ogni simulazione, a seconda della complessità del modello, può richiedere tempi di calcolo anche considerevoli, rendendo l'intero procedimento non sempre conveniente o addirittura impraticabile in contesti reali.

Si pone dunque la necessità di ridurre in maniera significativa il tempo complessivo richiesto dal processo di ottimizzazione. Due sono essenzialmente le direzioni di intervento possibili:

- **Miglioramento degli algoritmi**, attraverso l'introduzione di varianti, adattamenti o operatori più efficienti, capaci di accelerare la convergenza verso soluzioni ottimali riducendo così il numero di iterazioni necessarie.
- **Riduzione del numero di simulazioni**, ottenuta mediante l'impiego di modelli approssimanti che sostituiscano, almeno parzialmente, le valutazioni dirette della funzione obiettivo.

Il primo approccio è stato ampiamente trattato nella prima parte di queste dispense, dove si sono studiati diversi algoritmi di ottimizzazione e, in particolare, si è approfondito l'algoritmo genetico. Sono stati proposti e analizzati nuovi operatori con lo scopo di renderlo più efficiente, accorciando i tempi di convergenza e garantendo una maggiore robustezza rispetto ai metodi tradizionali.

Il secondo approccio, invece, costituisce il nucleo di questa sezione. Per ridurre il numero di simulazioni necessarie è infatti opportuno introdurre delle **superfici di risposta**, ossia modelli matematici capaci di approssimare il comportamento della funzione obiettivo a partire da un insieme limitato di dati noti. L'idea è quella di sostituire le valutazioni dirette, spesso onerose in termini di tempo computazionale, con stime ottenute dal modello approssimante, mantenendo comunque una buona fedeltà nei risultati.

In altre parole, data una funzione obiettivo di difficile calcolo diretto, si cercano dei modelli semplificati in grado di riprodurne l'andamento: in questo modo, una volta addestrato il modello sulle valutazioni già note, sarà possibile stimare rapidamente i valori per nuove combinazioni di variabili, riducendo così drasticamente il carico computazionale complessivo.

La scelta del modello di superficie di risposta non è tuttavia banale. Diverse sono le tecniche che si possono impiegare – interpolazioni polinomiali, metodi di regressione, modelli statistici, funzioni radiali – ma tra tutte, un approccio che si è dimostrato particolarmente efficace per l'approssimazione di funzioni generiche è quello delle **reti neurali artificiali**.

Sviluppate alla fine degli anni '80, le superfici di risposta hanno avuto un impatto significativo nel campo della progettazione ingegneristica e scientifica, grazie alle loro caratteristiche particolarmente versatili e alla capacità di modellare in maniera efficace sistemi complessi. Queste superfici si sono rivelate strumenti potenti per rappresentare in maniera approssimata il comportamento di un processo o di una funzione obiettivo, consentendo agli ingegneri di ottimizzare progetti senza dover necessariamente ricorrere a simulazioni o esperimenti continui e dispendiosi.

Tra i principali vantaggi delle superfici di risposta si possono evidenziare:

- **Elevata flessibilità**: Le superfici di risposta sono estremamente adattabili e possono rappresentare funzioni anche fortemente non lineari. Grazie alla possibilità di scegliere diverse forme funzionali (polinomiali, radiali, spline, reti neurali, ecc.), è possibile modellare fenomeni complessi senza dover conoscere a priori la struttura analitica della funzione.
- **Criterio multiobiettivo intrinseco**: Alcuni approcci di superficie di risposta sono stati sviluppati fin dall'inizio considerando più obiettivi contemporaneamente. Ciò significa che, oltre a fornire una stima della funzione principale, possono essere adattati per ottimizzare simultaneamente più parametri o criteri di progetto, senza dover ricorrere a procedure aggiuntive di aggregazione o pesatura.
- **Compatibilità con variabili di categoria**: Queste superfici non si limitano a gestire solo variabili continue; possono integrare anche variabili discrete o categoriali, come materiali, tipi di componenti o configurazioni alternative, ampliando notevolmente la gamma di problemi affrontabili.
- **Buona performance con dati limitati**: In contesti in cui le informazioni disponibili sono scarse o costose da ottenere, le superfici di risposta sono in grado di fornire stime utili senza richiedere un numero elevato di simulazioni o esperimenti. Questa caratteristica le rende particolarmente adatte a fasi preliminari di progettazione o a contesti in cui la raccolta dei dati è onerosa.

Nonostante questi vantaggi, le superfici di risposta presentano anche alcuni limiti importanti, che devono essere considerati nella loro applicazione pratica:

- **Accuratezza non massima**: Poiché si tratta di modelli approssimanti, le stime fornite non corrispondono mai esattamente ai valori reali della funzione. In problemi altamente sensibili, questa discrepanza può introdurre errori significativi se non viene adeguatamente controllata.
- **Tempi di calcolo**: Alcuni tipi di superficie, soprattutto quelli basati su modelli complessi come le reti neurali o le funzioni radiali, possono richiedere tempi di addestramento elevati, soprattutto al crescere del numero di variabili e dei dati disponibili.
- **Problemi di "setting"**: La flessibilità intrinseca introduce anche un elevato grado di libertà nella scelta di parametri, forma della funzione e configurazione della superficie. Questa libertà, se non gestita correttamente, può portare a risultati incoerenti o subottimali e richiede competenza e sperimentazione per ottenere una superficie ben calibrata.
- **Impossibilità di stimare l'errore**: A differenza di metodi statistici più rigorosi, molte superfici di risposta non forniscono una misura diretta dell'incertezza associata alla stima. Ciò rende difficile valutare la sicurezza delle previsioni in contesti critici.
- **Overfitting**: Se il modello è troppo complesso rispetto alla quantità di dati disponibile, la superficie può adattarsi eccessivamente ai dati di addestramento, perdendo capacità di generalizzare su nuovi punti. Questo fenomeno, noto come overfitting, compromette l'affidabilità delle previsioni e richiede strategie di controllo, come la regolarizzazione o la riduzione della complessità del modello.

Quindi le superfici di risposta rappresentano uno strumento estremamente utile e versatile nel processo di progettazione e ottimizzazione, ma richiedono attenzione nella loro implementazione per bilanciare flessibilità, accuratezza e capacità predittiva. La scelta del tipo di superficie, dei parametri e della quantità

di dati da utilizzare deve essere effettuata con cura, al fine di ottenere risultati utili senza incorrere nei principali limiti sopra descritti.

Perfetto! Allora procedo con il **Capitolo 2 – La rete neurale**, riscrivendolo in maniera più lunga, fluida e argomentata, senza tagliare nulla ma ampliando i passaggi e rendendo più organico il discorso.

8.6.1 La rete neurale: schema numerico

La **rete neurale artificiale** è un modello matematico e computazionale che trae ispirazione, in forma estremamente semplificata, dal funzionamento del cervello umano. L'idea di fondo è quella di imitare, almeno nelle linee generali, il modo in cui il cervello elabora le informazioni, cioè tramite una rete fittissima di cellule specializzate, i neuroni, che comunicano tra loro attraverso connessioni chiamate sinapsi.

Il cervello, infatti, è costituito da un numero straordinariamente elevato di neuroni, stimato in decine di miliardi. Ogni neurone è connesso a molti altri mediante terminazioni nervose che trasmettono impulsi elettrici e chimici. Quando un neurone riceve degli impulsi dai neuroni vicini, elabora tali segnali in base a una propria "soglia" di attivazione e, se questa soglia viene superata, genera a sua volta un segnale che trasmette agli altri neuroni collegati. In questo modo, l'informazione si propaga lungo la rete, dando origine a processi complessi di percezione, memoria, apprendimento e decisione.

Le **reti neurali artificiali** prendono ispirazione da questo meccanismo biologico per costruire modelli matematici capaci di apprendere relazioni tra variabili, di interpolare dati e di approssimare funzioni complesse. Naturalmente la rappresentazione artificiale è molto più semplice rispetto alla realtà biologica, ma mantiene alcuni elementi chiave: nodi (che rappresentano i neuroni), connessioni (che simulano le sinapsi) e pesi associati alle connessioni (che rappresentano l'importanza relativa di ciascun segnale).

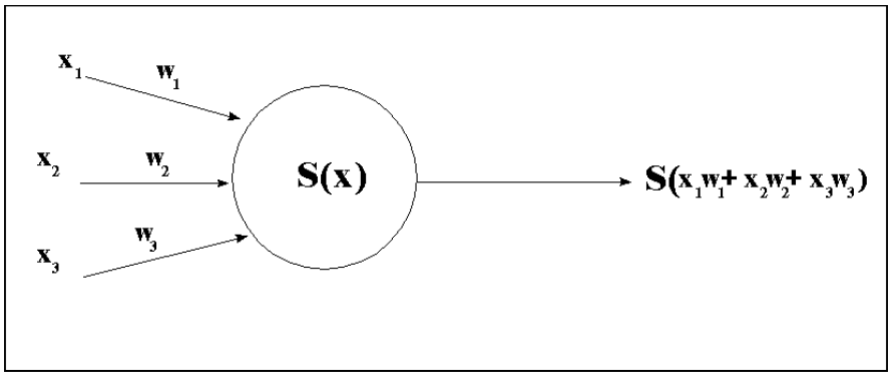
Il compito della rete neurale è dunque quello di creare un modello della funzione che si desidera approssimare. Il procedimento è il seguente: si parte da un insieme di dati noti, costituiti da valori delle variabili indipendenti (input) e dal corrispondente valore della funzione obiettivo (output). La rete viene "allenata" utilizzando questi dati, cioè modifica progressivamente i pesi delle connessioni interne in modo da riprodurre nel modo più accurato possibile l'output desiderato. Una volta terminata la fase di apprendimento, la rete è in grado di stimare con buona precisione il valore della funzione anche per nuove combinazioni di variabili che non erano presenti nei dati iniziali.

Da questo punto di vista, l'applicazione più naturale delle reti neurali è proprio quella di **estrapolatori di funzioni**. Quando il calcolo diretto della funzione obiettivo è oneroso, la rete può costituire un modello surrogato: addestrata su un insieme di simulazioni note, sarà poi in grado di fornire stime veloci per altri punti dello spazio delle variabili, riducendo così in modo considerevole i tempi di calcolo complessivi.

8.6.2 Il neurone e la funzione di attivazione

Per comprendere il funzionamento della rete, è utile partire dal singolo elemento: il neurone artificiale. Esso rappresenta la versione semplificata di un neurone biologico. In termini matematici, può essere visto come una **funzione primitiva (detta fuinzione di attivazione)** che trasforma un insieme di input in un output ben definito.

Il funzionamento è semplice: al neurone arrivano diversi segnali in ingresso, ciascuno moltiplicato per un peso che ne rappresenta l'importanza relativa. La somma pesata di questi ingressi viene poi elaborata attraverso una funzione di attivazione, che produce l'uscita del neurone. In questo modo, il nodo realizza una trasformazione non lineare dei dati, e proprio grazie a questa non linearità la rete, nel suo complesso, riesce ad approssimare funzioni anche molto complesse.



Quando si definisce una rete neurale, uno degli elementi fondamentali da scegliere è la **funzione di attivazione** f da utilizzare per ciascun neurone. La funzione di attivazione ha il compito di trasformare l'input pesato che arriva al neurone in un output non lineare, permettendo alla rete di modellare relazioni complesse e non lineari tra le variabili di input e l'output desiderato.

Affinché una funzione di attivazione sia utilizzabile in una rete neurale, deve soddisfare alcune condizioni essenziali:

- **Derivabilità in tutti i punti**: la funzione deve essere continua e derivabile in ogni punto del suo dominio, perché la procedura di addestramento della rete si basa sul calcolo dei gradienti (backpropagation). Se la funzione non fosse derivabile in qualche punto, l'algoritmo non potrebbe aggiornare correttamente i pesi.
- **Derivata prima diversa da zero**: è importante che la derivata non sia mai nulla, almeno nella maggior parte dell'intervallo operativo, perché una derivata nulla impedirebbe la propagazione dell'errore e quindi l'apprendimento dei pesi associati a quel neurone.

Esistono numerose funzioni di attivazione che rispettano questi requisiti. Tuttavia, **la funzione più comunemente utilizzata è la sigmoide**, definita come:

$$S(x) = \frac{1}{1 + e^{-Cx}}$$

dove C è un parametro positivo che controlla la pendenza della curva. Al crescere di C, la sigmoide diventa sempre più simile a uno **scalino**, passando rapidamente da valori vicini a 0 a valori vicini a 1.

Le caratteristiche principali della funzione sigmoide sono:

- Il suo valore è sempre compreso tra 0 e 1, il che la rende adatta a modellare probabilità o output normalizzati;
- È una funzione **monotona crescente**, quindi input maggiori producono output maggiori;
- Presenta **asintoti** orizzontali a 0 e 1, il che significa che valori estremi dell'input producono output prossimi ai limiti della funzione;
- Il parametro C è normalmente scelto pari a 1 o 2, valori che garantiscono un buon compromesso tra linearità locale e rapidità di saturazione;
- È sempre derivabile, e la sua derivata non è mai nulla all'interno dell'intervallo operativo significativo.

Un motivo fondamentale per cui la sigmoide è così diffusa è che **la derivata della sigmoide può essere espressa in funzione della sigmoide stessa**:

$$\frac{dS(x)}{dx} = S(x) \cdot (1 - S(x))$$

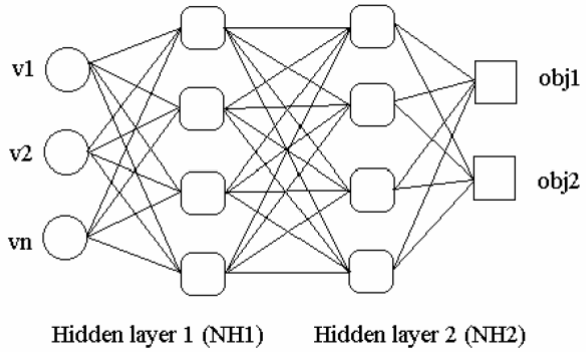
Questa proprietà è estremamente vantaggiosa nei calcoli numerici: poiché la derivata della funzione sarà molto utilizzata durante l'addestramento della rete (nella retropropagazione dell'errore), conoscere una formula diretta in funzione dell'output del neurone evita di dover calcolare numericamente la derivata. Questo riduce i tempi di calcolo e migliora l'efficienza complessiva della rete.

8.6.3 Topologia di una rete neurale

Una tipica rete neurale è organizzata in **strati**:

- lo **strato di input**, che riceve le variabili indipendenti (ad esempio x, y, z nel caso di una funzione f(x, y, z));
- uno o più **strati nascosti**, nei quali i nodi elaborano gli input combinandoli tra loro attraverso in neuroni e i pesi delle connessioni;
- lo **strato di output**, che fornisce il valore della funzione approssimata. Ovviamente se la funzione che deve essere estrapolata è multi obiettivo lo strato di output avrà m nodi di output.

In figura si osserva che i nodi dei vari strati sono collegati da connessioni dotate di pesi, indicati con w_1, w_2, …, w_n. Variando i pesi, la rete modifica il modo in cui combina gli input, e di conseguenza cambia il valore finale della funzione di rete.



Questa funzione di rete è la rappresentazione globale che la rete fornisce della funzione obiettivo: i dati di ingresso vengono trasformati progressivamente, passando da uno strato all'altro, fino a produrre il valore finale che costituisce la stima dell'uscita.

Ogni rete neurale è caratterizzata da tre elementi fondamentali:

1. **La struttura dei nodi**: riguarda il modo in cui è definita la funzione primitiva di ciascun neurone, cioè il tipo di trasformazione che esso applica agli input ricevuti. Esistono diverse funzioni di attivazione possibili (lineare, sigmoide, tangente iperbolica, ReLU e molte altre), ognuna con proprietà differenti e adatta a contesti specifici.
2. **La topologia della rete**: si riferisce all'organizzazione complessiva della rete, ossia al numero di strati, al numero di neuroni in ciascuno strato e al tipo di connessioni tra i nodi. Una rete

"feedforward", ad esempio, prevede che le informazioni fluiscano solo in avanti dagli input verso l'output, mentre le reti "ricorrenti" includono connessioni che riportano indietro l'informazione, permettendo di trattare dati sequenziali e temporali.

3. **L'algoritmo di apprendimento**: rappresenta la regola con cui i pesi delle connessioni vengono aggiornati durante la fase di addestramento. L'obiettivo è minimizzare l'errore tra il valore previsto dalla rete e quello effettivamente noto nei dati di addestramento. Uno degli algoritmi più diffusi è il metodo del **backpropagation**, che calcola i gradienti dell'errore rispetto ai pesi e li aggiorna in modo iterativo, seguendo la direzione di discesa più rapida.

8.6.4 L'algoritmo backpropagation

Per comprendere come una rete neurale apprende, è fondamentale analizzare il **calcolo del gradiente dell'errore**, cioè come varia l'errore della rete al variare dei pesi delle connessioni, che di conseguenza diventano i paramentri liberi della rete neurale su cui possiamo agire per farle imparare il comportamento della funzione da approssimare. In questa sezione consideriamo un **caso semplificato**: la rete ha un **unico obiettivo** (un solo output) e un **unico strato nascosto**. La funzione della rete può quindi essere descritta come:

$$f: \mathbb{R}^n \to \mathbb{R}$$

Per calcolare il valore della rete neurale per un insieme di variabili di input, utilizziamo il **metodo della propagazione in avanti**. Scegliamo come funzione di attivazione la **sigmoide** con parametro $C = 1$:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Sia x_i la variabile di input i-esima e $w_{ij}^{(1)}$ il peso tra l'input i e il neurone nascosto j. L'attivazione net del neurone nascosto j è:

$$\text{net}_j^{(1)} = \sum_{i=1}^{n} w_{ij}^{(1)} x_i$$

Applichiamo la funzione di attivazione per ottenere l'output del neurone nascosto:

$$O_j^{(1)} = S(\text{net}_j^{(1)})$$

Successivamente, calcoliamo la somma pesata delle uscite dei neuroni nascosti verso il neurone di output (dato che non siamo nel caso multiobiettivo, abbiamo un solo output):

$$\text{net}^{(2)} = \sum_j w_j^{(2)} O_j^{(1)}$$

Infine, l'output della rete, indicato come **RIS**, è dato dalla sigmoide applicata a questa somma pesata:

$$\text{RIS} = S(\text{net}^{(2)})$$

In questo modo, partendo dalle variabili di input, arriviamo al valore stimato della funzione.

Se conosciamo il valore reale della funzione per quel punto (per esempio perché appartiene al **Training Set** o al **Validation Set**), possiamo definire l'errore come:

$$E = \frac{1}{2}(\text{RIS} - f)^2$$

dove f è il valore reale della funzione.

Per aggiornare i pesi della rete, dobbiamo calcolare le derivate parziali dell'errore rispetto ai pesi, sia per quelli dello **strato di output** sia per quelli dello **strato nascosto**.

Per un generico peso $w_j^{(2)}$ tra il neurone nascosto j e l'output:

$$\frac{\partial E}{\partial w_j^{(2)}} = \frac{\partial E}{\partial \text{RIS}} \cdot \frac{\partial \text{RIS}}{\partial \text{net}^{(2)}} \cdot \frac{\partial \text{net}^{(2)}}{\partial w_j^{(2)}}$$

Calcolando ciascun termine:

- $\frac{\partial E}{\partial \text{RIS}} = \text{RIS} - f$
- $\frac{\partial \text{RIS}}{\partial \text{net}^{(2)}} = S(\text{net}^{(2)}) \cdot (1 - S(\text{net}^{(2)})) = \text{RIS} \cdot (1 - \text{RIS})$
- $\frac{\partial \text{net}^{(2)}}{\partial w_j^{(2)}} = O_j^{(1)}$

Quindi:

$$\frac{\partial E}{\partial w_j^{(2)}} = (\text{RIS} - f) \cdot \text{RIS} \cdot (1 - \text{RIS}) \cdot O_j^{(1)}$$

Per un peso $w_{ij}^{(1)}$ tra input i e neurone nascosto j, il gradiente si calcola tenendo conto del contributo di tutti i neuroni di output:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial \text{RIS}} \cdot \frac{\partial \text{RIS}}{\partial \text{net}^{(2)}} \cdot \frac{\partial \text{net}^{(2)}}{\partial O_j^{(1)}} \cdot \frac{\partial O_j^{(1)}}{\partial \text{net}_j^{(1)}} \cdot \frac{\partial \text{net}_j^{(1)}}{\partial w_{ij}^{(1)}}$$

Sostituendo le derivate:

- $\frac{\partial \text{net}^{(2)}}{\partial O_j^{(1)}} = w_j^{(2)}$
- $\frac{\partial O_j^{(1)}}{\partial \text{net}_j^{(1)}} = S(\text{net}_j^{(1)}) \cdot (1 - S(\text{net}_j^{(1)})) = O_j^{(1)} \cdot (1 - O_j^{(1)})$
- $\frac{\partial \text{net}_j^{(1)}}{\partial w_{ij}^{(1)}} = x_i$

Quindi otteniamo:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = (\text{RIS} - f) \cdot \text{RIS} \cdot (1 - \text{RIS}) \cdot w_j^{(2)} \cdot O_j^{(1)} \cdot (1 - O_j^{(1)}) \cdot x_i$$

Una volta calcolati i gradienti, possiamo aggiornare i pesi usando un semplice metodo di discesa del gradiente, chiamato **Metodo di Cauchy semplificato**:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

dove η è il **tasso di apprendimento (learning rate)**.

Questo processo viene ripetuto iterativamente su tutti i punti del training set fino a ridurre l'errore complessivo della rete.

8.6.5 Metodi avanzati di aggiornamento dei pesi

Il **metodo di Cauchy semplificato** è una forma di discesa del gradiente utilizzata per aggiornare i pesi in una rete neurale. L'idea fondamentale è **muoversi nella direzione opposta al gradiente dell'errore**, così da ridurre progressivamente l'errore della rete.

In forma semplificata, l'aggiornamento dei pesi può essere scritto come:

$$w^{(n)} = w^{(n-1)} - g \frac{\partial E}{\partial w}$$

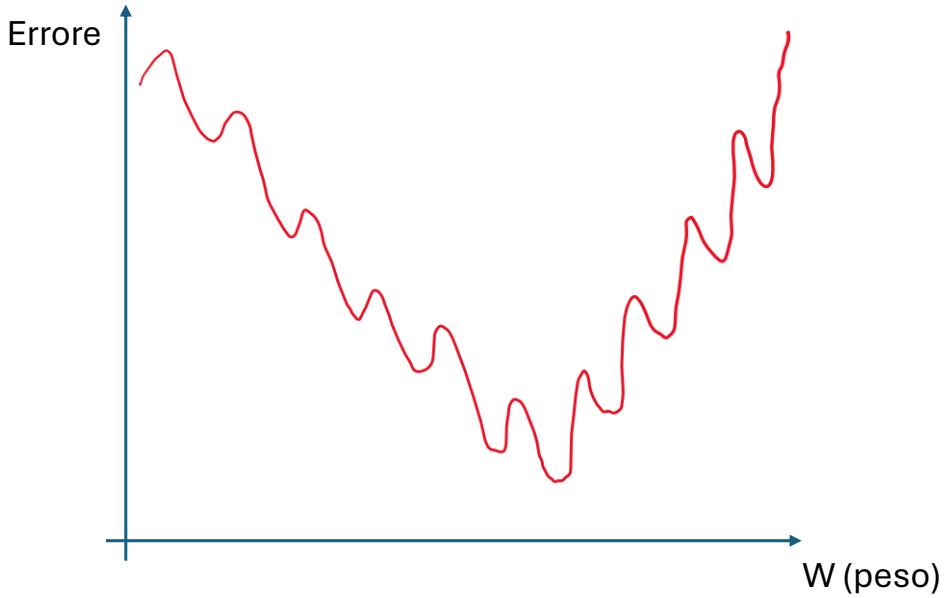
dove:

- $w^{(n)}$ è il vettore dei pesi alla n-esima iterazione;
- $\frac{\partial E}{\partial w}$ è il **gradiente dell'errore** rispetto ai pesi;
- g è il **parametro di apprendimento** (learning rate), che regola l'entità dello spostamento lungo il gradiente;
- Il segno "-" indica che ci muoviamo **in direzione opposta al gradiente**, cioè verso il minimo dell'errore.

Il parametro g è fondamentale:

- Se g è grande, i passi di aggiornamento sono ampi e la rete può muoversi velocemente, ma c'è il rischio di oscillazioni o di saltare oltre il minimo;
- Se g è piccolo, i passi sono più lenti, con maggiore stabilità, ma l'apprendimento può richiedere molte iterazioni.

Tipicamente, valori iniziali di g sono molto piccoli, per esempio $g = 10^{-3}$.



Uno dei problemi principali nell'addestramento di una rete neurale è che la **funzione errore è molto irregolare**, con micro-avvallamenti e picchi locali che possono intrappolare il gradiente. Per superare questo problema, si introduce un **parametro d'inerzia** α, che tiene conto dello spostamento dei pesi nelle iterazioni precedenti:

$$w^{(n)} = w^{(n-1)} - g\frac{\partial E}{\partial w} + \alpha(w^{(n-1)} - w^{(n-2)})$$

In questo modo, anche se il gradiente locale è piccolo o la funzione è disturbata, l'**inerzia del passo precedente** aiuta a "saltare" micro-avvallamenti e a mantenere la direzione complessiva dell'apprendimento.

Un ulteriore miglioramento consiste nell'uso di un **learning rate adattativo**, cioè un g che **non rimane fisso**, ma può modificarsi durante l'addestramento a seconda del comportamento del gradiente. L'idea è semplice:

- Partiamo da un valore minimo di g (per esempio $g_{\min} = 10^{-3}$).
- Calcoliamo il gradiente rispetto al peso due volte consecutive: $\frac{\partial E^{(n)}}{\partial w}$ e $\frac{\partial E^{(n-1)}}{\partial w}$.
- Valutiamo il prodotto dei due gradienti:
  - Se $\frac{\partial E^{(n)}}{\partial w} \cdot \frac{\partial E^{(n-1)}}{\partial w} > 0$, significa che la **direzione del gradiente è costante** tra due iterazioni successive. Questa direzione è considerata "sicura", quindi possiamo **aumentare il passo**, moltiplicando g per un fattore $k > 1$ (tipicamente $1.2 \leq k \leq 2$).
  - Se invece $\frac{\partial E^{(n)}}{\partial w} \cdot \frac{\partial E^{(n-1)}}{\partial w} < 0$, significa che il gradiente ha **cambiato segno**, indicando che ci troviamo in una zona instabile o vicino a un minimo locale. In questo caso, il passo deve essere **ridotto**, quindi manteniamo $g = g_{\min}$.

In questo modo il learning rate si adatta dinamicamente, permettendo alla rete di:

- Accelerare l'apprendimento quando la direzione è stabile;
- Procedere con cautela in zone critiche della funzione errore.

I vantaggi sono:

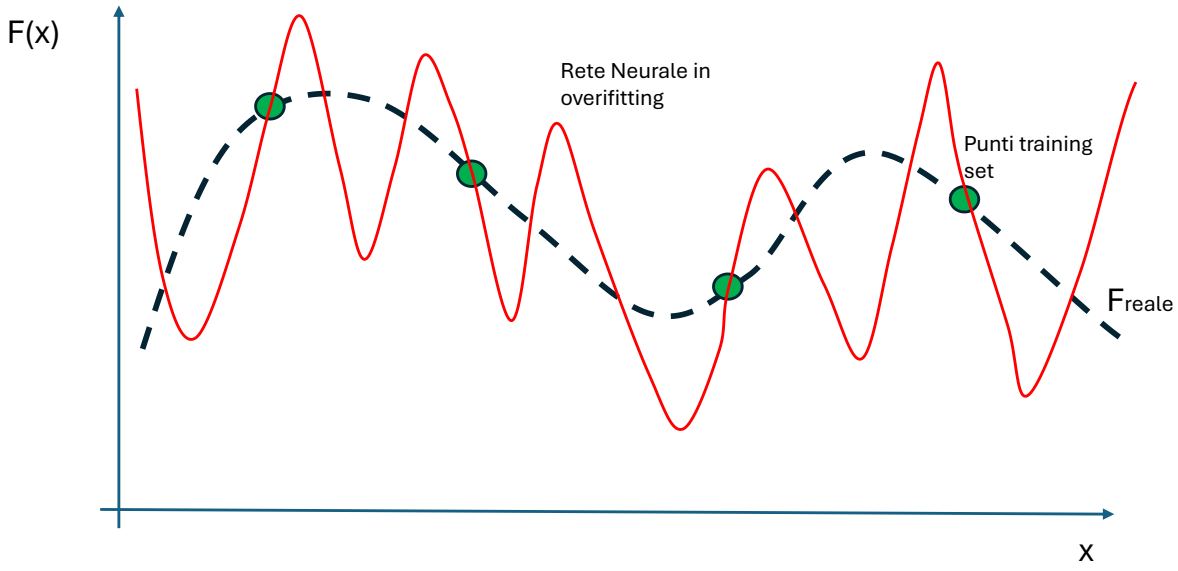
- Riduce il rischio di rimanere intrappolati in minimi locali o micro-avvallamenti;
- Permette di combinare **stabilità** (g piccolo nelle zone pericolose) e **velocità** (g maggiore nelle zone sicure);
- L'inerzia aiuta a mantenere continuità nel movimento dei pesi, evitando oscillazioni troppo brusche;
- Il learning rate adattativo ottimizza i tempi di apprendimento senza richiedere la scelta di un singolo valore fisso per tutta la rete.

8.6.6 Il problema di overfitting

Uno dei problemi fondamentali nella progettazione di una rete neurale è che **la topologia della rete** – cioè il numero di strati nascosti e di nodi per ciascun strato – è **decisa dall'utente**. Questa scelta influisce in maniera critica sulla capacità della rete di apprendere correttamente la funzione obiettivo senza incorrere in fenomeni indesiderati.

Supponiamo di voler modellare una funzione che dipende da 10 variabili e ha un unico output. Le scelte possibili sono molto ampie: possiamo utilizzare un singolo strato nascosto con **un solo nodo**, oppure costruire una rete molto complessa con **100 strati nascosti e 100 nodi ciascuno**. Entrambe le configurazioni hanno vantaggi e svantaggi.

- **Vantaggio di molti nodi:** l'errore della rete può avvicinarsi a zero, perché la rete ha sufficiente capacità per "passare" esattamente attraverso tutti i punti di dati disponibili.
- **Svantaggio:** si incorre facilmente nel fenomeno chiamato **overfitting**.



L’overfitting si verifica quando la rete neurale **si adatta troppo ai dati di training**, riuscendo a predire perfettamente i valori noti, ma **perdendo capacità di generalizzazione** su punti nuovi o non visti durante l’allenamento.

In pratica, se la rete contiene più nodi o strati di quanti siano necessari per rappresentare la funzione reale, essa diventa “rumorosa”: cattura anche piccole fluttuazioni o errori dei dati di training, che non riflettono la vera natura della funzione.

Per controllare l’overfitting si utilizza un **Validation Set**, cioè un insieme di dati che **non viene usato durante l’addestramento**, ma serve per verificare se la rete è in grado di predire correttamente valori nuovi. Se l’errore sul Validation Set aumenta nonostante l’errore sul Training Set sia vicino a zero, significa che la rete è overfittata.

In molti casi, prima di utilizzare il Validation Set, si adotta una **strategia iterativa per ottimizzare il numero di nodi**:

1. Si parte con una rete relativamente ampia, ad esempio **due strati nascosti**, ciascuno con un numero di nodi pari al numero di variabili di input.
2. Si calcola l’errore della rete: se l’errore è vicino a zero, significa che la rete ha sufficiente capacità per rappresentare la funzione.
3. Per evitare l’overfitting, si riduce gradualmente il numero di nodi: si tolgono uno o più nodi per strato e si ricalcola l’errore.
4. Se l’errore rimane vicino a zero, si può continuare a ridurre il numero di nodi; al contrario, se l’errore aumenta significativamente, significa che si è raggiunta la **capacità minima necessaria** per modellare correttamente la funzione.

In questo modo si individua il **minimo numero di nodi** necessario affinché la rete possa apprendere la funzione senza cadere nell’overfitting.

9 I processi gaussiani e il concetto di Design of Experiments adattativo

Il **Design of Experiments (DOE)** nasce come strumento per cogliere il massimo numero di informazioni possibili da un sistema, esplorando in modo sistematico lo spazio delle variabili di interesse. In termini pratici, costruire un DOE significa definire una serie di configurazioni – cioè combinazioni dei valori assunti dalle variabili indipendenti – che andranno poi valutate. Queste configurazioni rappresentano i punti nello spazio di definizione sui quali viene eseguita la simulazione o l'esperimento reale.

Tuttavia, in questa formulazione classica, il DOE e il calcolo della funzione obiettivo rimangono due processi separati e indipendenti. Prima si stabilisce il piano sperimentale, ossia il set dei punti da analizzare, e solo successivamente si calcola il valore della funzione obiettivo o della funzione di risposta nei punti selezionati. Questo significa che il DOE non utilizza alcuna informazione derivata dalla funzione obiettivo stessa: i punti vengono scelti in anticipo, senza che i risultati già ottenuti possano influenzare o guidare la scelta delle configurazioni successive.

Un approccio analogo si riscontra nella costruzione delle **superfici di risposta**. Anche qui, il procedimento tradizionale prevede innanzitutto la raccolta di un insieme di dati sperimentali o numerici (il cosiddetto data-base iniziale), e solo in un secondo momento la costruzione della superficie di risposta vera e propria, ossia un modello approssimato che riproduce il comportamento della funzione obiettivo nello spazio delle variabili. In questo schema sequenziale, prima si osservano i dati e poi si costruisce la superficie: non c'è un meccanismo che consenta al modello in costruzione di orientare in tempo reale la scelta dei punti da calcolare.

Per superare questo limite, è stato introdotto il concetto di **DOE adattativo**. A differenza del DOE classico, il DOE adattativo non si basa su una griglia predefinita e immutabile di configurazioni, ma posiziona i punti di campionamento dove realmente servono, ossia in quelle regioni dello spazio delle variabili che risultano più rilevanti o più incerte rispetto alla costruzione del modello. In altre parole, la scelta dei nuovi esperimenti non è indipendente dai risultati già ottenuti: è proprio l'analisi delle informazioni disponibili che guida la decisione su quali configurazioni esplorare in seguito.

Questo tipo di approccio è reso possibile dall'impiego dei **processi gaussiani**. I processi gaussiani costituiscono una potente classe di modelli probabilistici utilizzati per costruire superfici di risposta, con la caratteristica peculiare di integrare in modo parallelo la raccolta delle configurazioni sperimentali e la costruzione della superficie stessa. Ogni volta che si aggiunge un nuovo punto calcolato, il modello aggiornato fornisce sia una stima della funzione obiettivo in tutto lo spazio delle variabili, sia una misura dell'incertezza associata a tale stima. Proprio questa informazione sull'incertezza diventa fondamentale per guidare l'algoritmo nella selezione dei punti successivi, che saranno scelti preferibilmente nelle regioni dove il modello è meno affidabile o dove ci si aspetta di ottenere informazioni più utili.

9.1 Schema operativo design of experiments adattativo

Supponiamo, per semplicità, di avere una funzione definita in una sola variabile. L'obiettivo è costruire una **superficie di risposta** (in questo caso una curva di risposta) a partire da un certo numero di punti noti. Immaginiamo quindi di avere già a disposizione alcuni valori campionati della funzione, ad esempio nei punti $x = 1{,}2{,}3{,}4$. Questi dati costituiscono la base iniziale di conoscenza, ossia il nostro piccolo "database" di riferimento.

Il comportamento reale della funzione, cioè quello che il solver restituirebbe, è noto soltanto in corrispondenza di questi punti, mentre negli intervalli intermedi non abbiamo informazioni dirette. Per stimare il comportamento della funzione tra i punti noti, costruiamo un modello di **processo gaussiani**, cioè un processo che approssima la funzione sulla base dei dati disponibili.

Il processo gaussiani ha due caratteristiche importanti:

- **Interpola i punti noti**: la curva ottenuta passa esattamente per i valori dati nei punti 1, 2, 3 e 4.
- **Introduce la stima dell'errore tra i punti noti**: tra un punto conosciuto e l'altro, la curva del modello non coincide necessariamente con la funzione reale, e quindi esiste un certo errore di approssimazione.

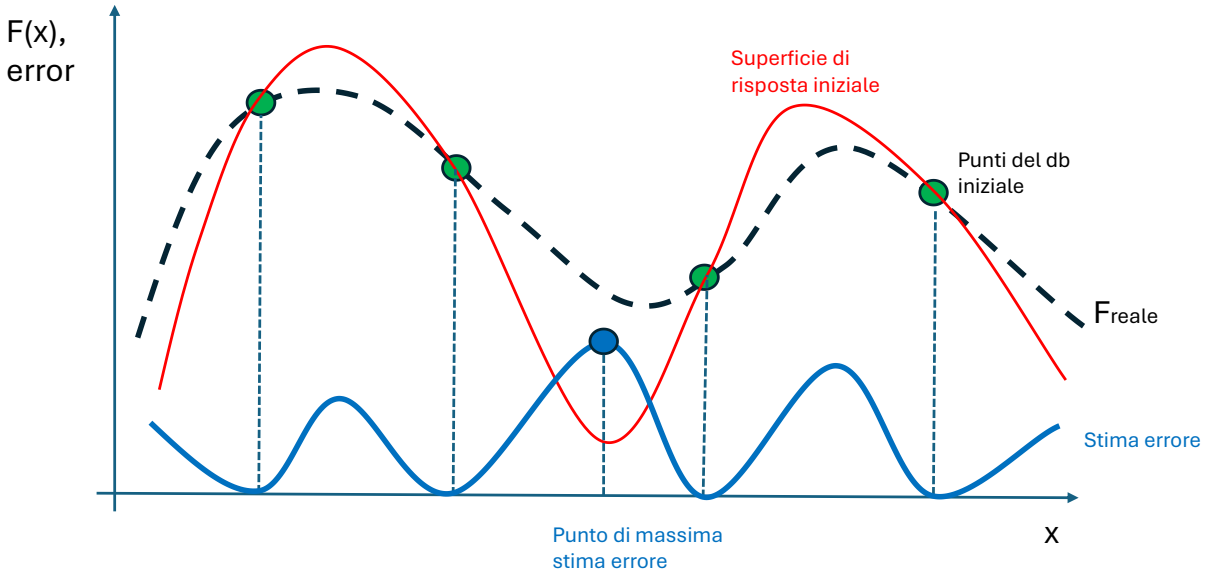
La forza dei processi gaussiani non risiede soltanto nella capacità di approssimare la funzione, ma anche nella possibilità di **quantificare l'incertezza mediante una sua stima**. Infatti, insieme alla stima della funzione, il modello di Kriging fornisce una misura dell'errore previsto, spesso indicata con $\sigma(x)$. Questa quantità può essere rappresentata graficamente come una fascia attorno alla curva stimata:

- in corrispondenza dei punti noti, l'errore σ è nullo, perché il modello passa esattamente per quei valori;
- negli intervalli tra i punti, invece, σ cresce, indicando che il modello è meno sicuro della stima in quelle regioni.

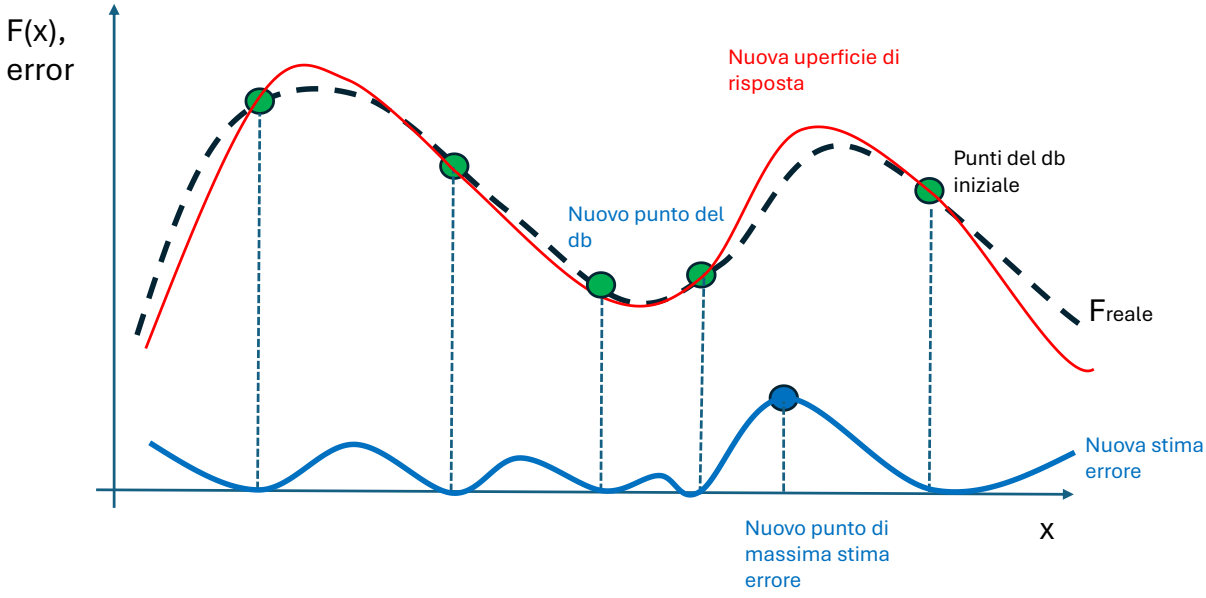
Questa proprietà consente di costruire un **DOE adattativo**. Poiché conosciamo la distribuzione dell'errore $\sigma(x)$, possiamo decidere dove inserire un nuovo punto di campionamento: la scelta ottimale è posizionarlo proprio dove l'errore stimato è massimo, cioè dove ci aspettiamo di ottenere la maggiore quantità di informazione aggiuntiva. Supponiamo quindi di individuare il punto 5, cioè la nuova configurazione del DOE: calcolando il valore reale della funzione in quel punto ed aggiornando il modello di Kriging, la superficie di risposta diventa immediatamente più accurata.

Questo approccio permette di **costruire la superficie di risposta e, contemporaneamente, ottimizzare il DOE**. Non si tratta più di definire a priori tutti i punti da campionare, ma di aggiungerli progressivamente nelle zone che lo richiedono. Il vantaggio è evidente: si ottiene un modello molto più preciso con un numero di punti decisamente inferiore rispetto a un DOE classico casuale o regolare.

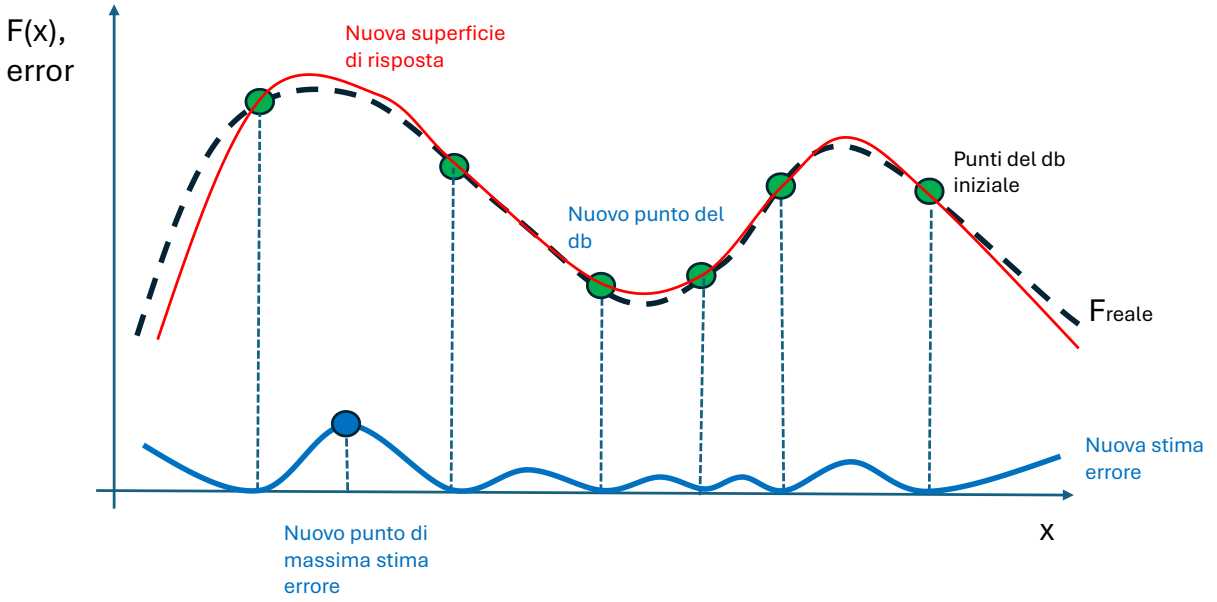
Prima iterazione



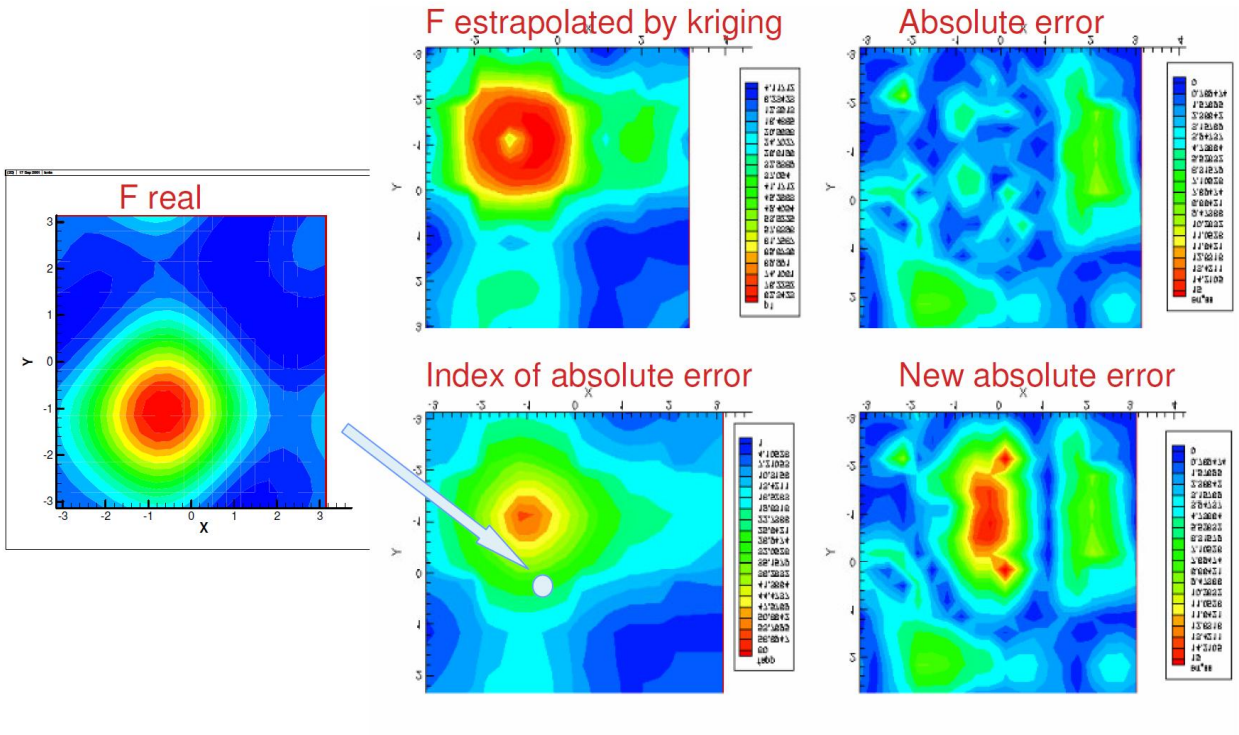
Seconda iterazione



Terza iterazione



Ora immaginiamo di avere una funzione a due variabili. Se generiamo un DOE di 100 punti con distribuzione **random**, questi saranno sparsi casualmente sul piano, senza particolare logica né concentrazione nelle zone più significative. Con il DOE adattativo, invece, possiamo guidare la scelta dei punti sulla base della distribuzione dell'errore σ.



Per farlo, possiamo definire due tipi di indici:

Errore assoluto

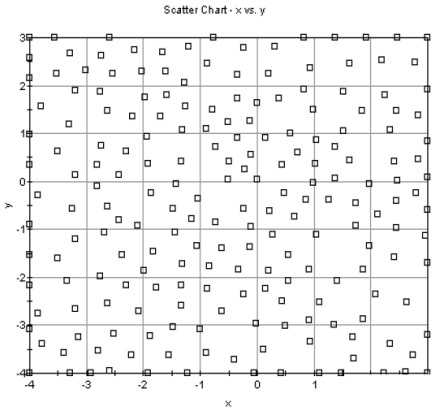
$$I_{ERR}^{ASS} = |f * s|$$

dove f è il valore reale della funzione e s il valore stimato dal modello. In questo caso i nuovi punti del DOE verranno posizionati nelle regioni in cui l'errore assoluto è massimo. Ne risulta che le aree in cui la funzione ha valori elevati riceveranno più punti, perché proprio lì l'errore assoluto tende a crescere di più. In pratica, si concentra il campionamento nelle zone di massimo della funzione.

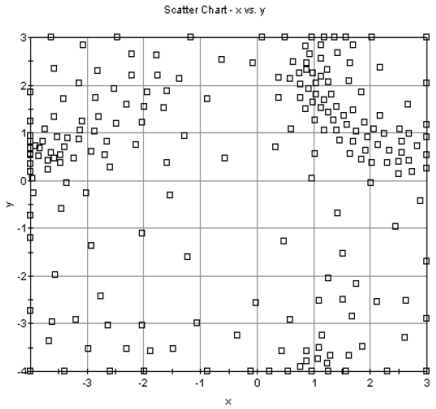
Errore relativo

$$I_{ERR}^{REL} = \frac{|s|}{|f|}$$

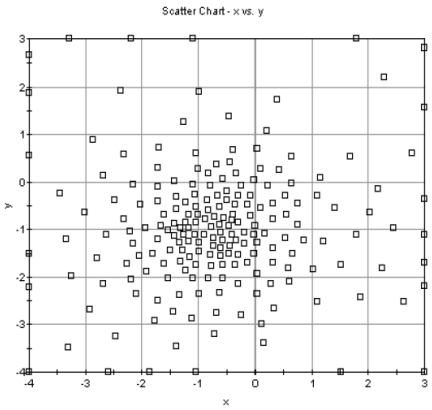
In questo caso i punti vengono distribuiti soprattutto nelle regioni in cui la funzione assume valori bassi, poiché in quelle zone l'errore relativo diventa più significativo.



Plain Crossvalidation



Relative Error Crossvalidation



Absolute Error Crossvalidation

La scelta tra errore assoluto e relativo dipende dallo scopo dell'analisi:

- se l'obiettivo è **modellare fedelmente l'intero andamento della funzione obiettivo**, allora ha senso privilegiare l'errore relativo, perché vogliamo che la superficie di risposta sia ben calibrata ovunque, anche nelle zone in cui la funzione ha valori ridotti;

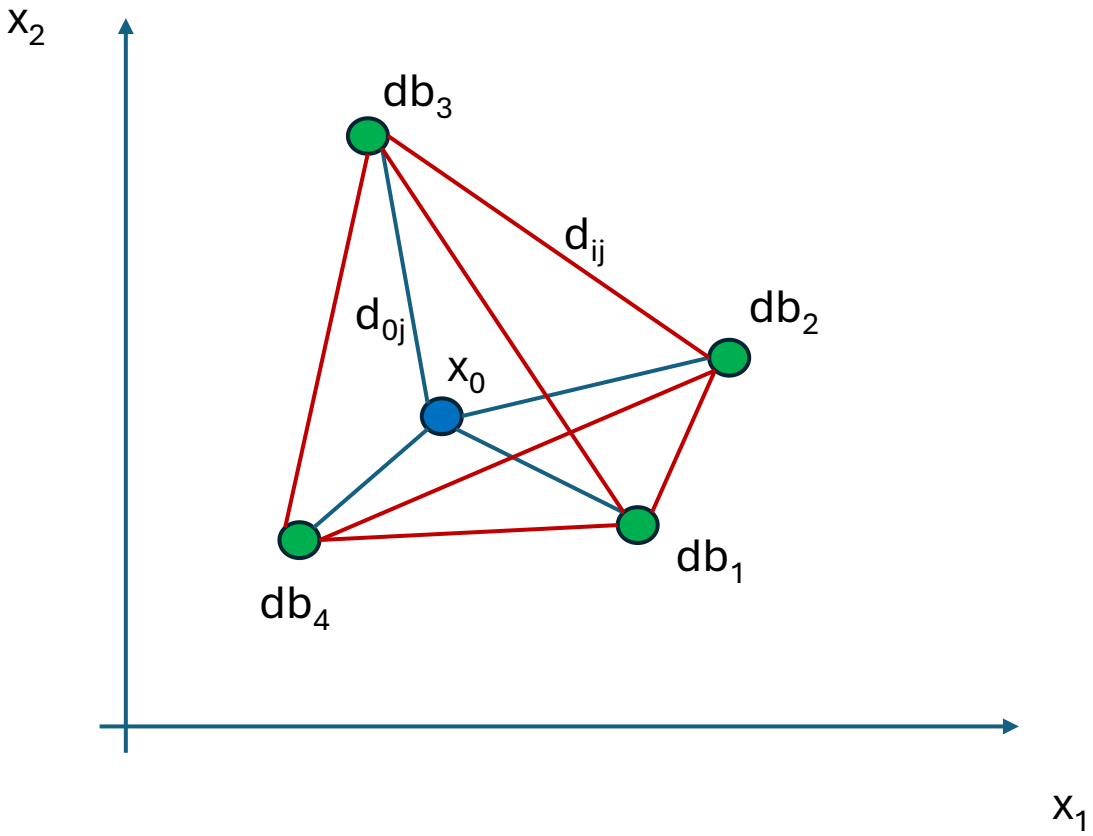
- se invece vogliamo **conoscere con la massima precisione le regioni in cui la funzione raggiunge prestazioni elevate** (ad esempio i massimi globali o locali), allora è più utile basarsi sull'errore assoluto, concentrando i punti nelle zone di maggiore interesse pratico.

In questo modo, il DOE adattativo non è solo un metodo più efficiente di campionamento, ma diventa anche uno strumento flessibile che si adatta alle esigenze specifiche del problema: modellare l'intero spazio delle variabili con buona accuratezza, oppure focalizzarsi solo sulle regioni critiche.

Ora quello che è importante è comprendere come i processi gaussiani posso valurtre sia il valore della funzione che il valore della stima dell'errore commesso. I metodoiche studieremo sono il metodo semplificato kriging che il metodo DACE.

9.2 Modello Kriging

Il **Kriging** rappresenta il primo modello basato su **processi gaussiani** ad essere stato formalizzato e utilizzato in ambito di progettazione numerica e statistica. Si tratta di un metodo di interpolazione e approssimazione che, a differenza di tecniche più semplici come il **K-Nearest Neighbors (K-NN)**, non si limita a stimare un punto sulla base della vicinanza con punti noti, ma integra informazioni aggiuntive di natura probabilistica e statistica. In particolare, il Kriging costruisce un **predittore lineare a varianza minima**, sfruttando la correlazione fra i punti noti e quelli da stimare.



Sia

$$f: \mathbb{R}^n \to \mathbb{R}$$

una funzione reale di n variabili, di cui non conosciamo l'andamento analitico ma soltanto un insieme finito di valutazioni ottenute tramite un **solver** o una simulazione numerica molto costosa. Chiamiamo questo insieme di dati disponibili **database delle informazioni**:

$$\mathcal{D} = \{(x_i, f(x_i)), \quad i = 1, \dots, N_{DB}\}$$

dove $x_i \in \mathbb{R}^n$ rappresenta il vettore delle variabili di input, e $f(x_i)$ il corrispondente valore della funzione reale.

L'obiettivo è stimare il valore della funzione in un **nuovo punto** x_0, senza dover richiamare il solver. Il modello Kriging approssima il valore incognito come combinazione lineare pesata dei valori noti:

$$f_{KR}(x_0) = \sum_{i=1}^{N_{DB}} w_i \, f(x_i)$$

dove i pesi w_i dipendono dal punto di interpolazione x_0 e non sono fissati a priori.

Il principio guida è che i pesi vengano determinati in modo da **minimizzare la varianza dell'errore** fra la funzione stimata e quella reale, garantendo inoltre la condizione di **non distorsione** (unbiasedness):

$$\sum_{i=1}^{N_{DB}} w_i = 1.$$

Il calcolo dei pesi porta alla risoluzione di un **sistema lineare** che coinvolge una **funzione di correlazione** γ, la quale quantifica la dipendenza statistica fra due punti in funzione della loro distanza. Formalmente, i pesi soddisfano:

$$\begin{cases} \sum_{j=1}^{N_{DB}} w_j \, \gamma(d_{ij}) + \lambda = \gamma(d_{i0}), \quad i = 1, \dots, N_{DB} \\ \sum_{j=1}^{N_{DB}} w_j = 1 \end{cases}$$

dove:

- $d_{ij} = \| x_i - x_j \|$ è la distanza (in norma euclidea o altra metrica) fra i punti x_i e x_j,
- $d_{i0} = \| x_i - x_0 \|$ è la distanza fra il punto incognito e un punto noto,
- λ è il moltiplicatore di Lagrange legato al vincolo di non distorsione.

In forma matriciale, questo sistema si scrive:

$$\begin{bmatrix} \Gamma & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \lambda \end{bmatrix} = \begin{bmatrix} \gamma_0 \\ 1 \end{bmatrix}$$

dove:

- $\Gamma \in \mathbb{R}^{N_{DB} \times N_{DB}}$ è la matrice delle correlazioni con $\Gamma_{ij} = \gamma(d_{ij})$,
- $\mathbf{1}$ è un vettore colonna di 1,

- $\gamma_0 = (\gamma(d_{10}), \gamma(d_{20}), \dots, \gamma(d_{N_{DB},0}))^T$.

La scelta della funzione di correlazione γ è cruciale. Essa descrive come la similarità fra i punti decresce con la distanza. Una formulazione semplice, spesso usata nei modelli base, è:

$$\gamma(d) = d^k, \quad k \in [1,2].$$

Per definizione:

$$\gamma(0) = 1 \quad \Rightarrow \quad \gamma(d_{ii}) = 1.$$

In modelli più evoluti (Kriging geostatistico o DACE – Design and Analysis of Computer Experiments), vengono usate altre funzioni come:

- il modello esponenziale: $\gamma(d) = \exp(-\theta d)$,
- il modello gaussiano: $\gamma(d) = \exp(-\theta d^2)$,

Uno degli aspetti più potenti del Kriging è la possibilità di stimare la **varianza predittiva** $\sigma^2(x_0)$ associata alla predizione in x_0. In particolare, vale:

$$f_{KR}(x_0) = f(x_0) \pm 3\,\sigma(x_0),$$

con probabilità del 99.9% (proprietà del processo gaussiano).

Formalmente:

$$\sigma^2(x_0) = \sum_{i=1}^{N_{DB}} w_i\ \gamma(d_{i0}).$$

La disponibilità di una stima dell'errore rende possibile l'adozione di strategie di **DOE adattativo** (Design of Experiments). In questo schema, i nuovi punti da simulare non vengono scelti a priori in maniera uniforme o casuale, ma in modo da ridurre progressivamente le zone di maggiore incertezza, aumentando l'efficienza del processo di apprendimento del modello.

Il Kriging presenta numerosi vantaggi pratici:

- **Accuratezza**: il modello passa esattamente per i punti noti (interpolazione esatta).
- **Efficienza computazionale**: una volta costruito il modello, stimare nuovi punti richiede solo la risoluzione di un sistema lineare.
- **Robustezza**: non richiede scelte complicate di parametri (l'unico vero parametro di setting è k o, nei modelli avanzati, i parametri di correlazione θ).
- **Interpretazione statistica**: non solo fornisce una stima del valore della funzione, ma anche una misura dell'**incertezza** associata a tale stima.

Nonostante i numerosi vantaggi, il Kriging presenta anche alcuni limiti:

- Non è adatto per variabili **categoriche**.
- Quando il numero di configurazioni note è minore del numero di variabili ($N_{DB} < n$), il modello diventa scarsamente informativo e può produrre predizioni poco affidabili.

9.3 DACE Design and Analysis of Computer Experiments

Immaginiamo di voler analizzare un fenomeno deterministico, che dipende da un insieme di **k variabili indipendenti**. Abbiamo a disposizione un certo numero di osservazioni, diciamo **n punti campionati**, che indichiamo con:

$$x^{(i)} = (x_1^{(i)}, \dots, x_k^{(i)}), \quad i = 1, \dots, n$$

e per ciascun punto abbiamo misurato il valore corrispondente della funzione:

$$y^{(i)} = y(x^{(i)}).$$

Il problema che si pone è costruire una **superficie di risposta**, cioè un modello matematico in grado di descrivere in maniera continua il fenomeno, partendo solo da questi dati discreti.

Un approccio immediato, il più semplice, è quello della **regressione lineare**: si assume cioè che i dati osservati siano spiegati da una media costante più un termine di errore. Formalmente si scrive:

$$y(x^{(i)}) = \mu + \varepsilon(x^{(i)}), \quad i = 1, \dots, n,$$

dove μ rappresenta la **media del fenomeno**, mentre $\varepsilon(x^{(i)})$ è un **errore casuale**, che assumiamo distribuito secondo una normale $\mathcal{N}(0, \sigma^2)$.

Fin qui, la formulazione è identica a quella del **kriging ordinario**. La differenza sostanziale, che introduce il modello **DACE (Design and Analysis of Computer Experiments)**, è che gli errori $\varepsilon(x^{(i)})$ non vengono considerati indipendenti, ma **correlati**.

È infatti ragionevole pensare che due punti vicini nello spazio delle variabili producano valori della funzione tra loro simili, quindi anche gli errori associati siano fortemente correlati. Viceversa, se due punti sono lontani, la correlazione tra i rispettivi errori tenderà a diminuire fino a diventare trascurabile.

Per quantificare questa correlazione, si introduce una **funzione di distanza generalizzata** fra due punti $x^{(i)}$ e $x^{(j)}$:

$$d(x^{(i)}, x^{(j)}) = \sum_{h=1}^{k} \theta_h \, |x_h^{(i)} - x_h^{(j)}|^{p_h}, \quad \theta_h > 0, \; p_h \in [1,2].$$

Questa non è la distanza euclidea classica: infatti, nella distanza euclidea ogni variabile ha lo stesso peso, mentre qui introduciamo parametri θ_h e p_h che consentono di modulare l'importanza relativa delle variabili.

- Il parametro θ_h rappresenta una misura dell'**attività** della variabile x_h. Se θ_h è molto grande, anche piccole differenze tra $x_h^{(i)}$ e $x_h^{(j)}$ generano una distanza elevata, e quindi bassa correlazione. Dire che una variabile è **attiva** significa proprio che influisce sensibilmente sul comportamento della funzione.
- Il parametro p_h regola invece la rapidità con cui decresce la correlazione rispetto alla distanza: valori vicini a 2 producono un decadimento più "dolce", valori vicini a 1 un decadimento più rapido.

La correlazione tra i termini di errore $\varepsilon(x^{(i)})$ e $\varepsilon(x^{(j)})$ è modellata tramite la funzione:

$$\mathrm{Corr}[\varepsilon(x^{(i)}), \varepsilon(x^{(j)})] = \exp\left(-d(x^{(i)}, x^{(j)})\right).$$

Questa scelta garantisce che:

- se $x^{(i)}$ e $x^{(j)}$ sono vicini, $d(x^{(i)}, x^{(j)}) \approx 0$, quindi la correlazione tende a 1;
- se i punti sono lontani, $d(x^{(i)}, x^{(j)})$ diventa grande e la correlazione tende a 0.

Abbiamo dunque trasformato l'errore $\varepsilon(x)$ in un **processo stocastico correlato**, indicizzato dallo spazio delle variabili. Per questo il modello complessivo prende il nome di **stochastic process model** o più precisamente **DACE stochastic process model**.

Il modello DACE contiene in totale $2k + 2$ parametri:

$$\mu,\ \sigma^2,\ \theta_1, \dots, \theta_k,\ p_1, \dots, p_k.$$

Questi parametri devono essere stimati dai dati. La stima si effettua tramite il metodo della **massima verosimiglianza**, ossia scegliendo i valori che massimizzano la probabilità di osservare il campione $y = (y_1, \dots, y_n)^T$.

Definendo:

- R la matrice di correlazione $n \times n$, con elementi $R_{ij} = \text{Corr}[\varepsilon(x^{(i)}), \varepsilon(x^{(j)})]$,
- $\mathbf{1}$ il vettore colonna di dimensione n i cui elementi valgono tutti 1,

la funzione di verosimiglianza risulta:

$$L(\mu, \sigma^2, \theta, p) = \frac{1}{(2\pi)^{n/2}\sigma^n |R|^{1/2}} \exp\left(-\frac{(y - \mu\mathbf{1})^T R^{-1} (y - \mu\mathbf{1})}{2\sigma^2}\right).$$

Dalla massimizzazione di questa funzione si ottengono gli stimatori:

$$\hat{\mu} = \frac{\mathbf{1}^T R^{-1} y}{\mathbf{1}^T R^{-1} \mathbf{1}}, \qquad \hat{\sigma}^2 = \frac{(y - \hat{\mu}\mathbf{1})^T R^{-1} (y - \hat{\mu}\mathbf{1})}{n}.$$

Sostituendo questi valori nella verosimiglianza, si ottiene la concentrated likelihood function, che dipende solo dai parametri θ_h, p_h. È proprio questa funzione che si massimizza in pratica per stimare i parametri di correlazione.

Supponiamo ora di voler prevedere il valore della funzione in un nuovo punto x^*.

Definiamo il vettore r, le cui componenti sono le correlazioni tra l'errore in x^* e quelli nei punti campionati:

$$r_i(x^*) = \text{Corr}[\varepsilon(x^*), \varepsilon(x^{(i)})].$$

La **miglior previsione lineare non distorta (BLUP)** di $y(x^*)$ risulta:

$$\hat{y}(x^*) = \hat{\mu} + r^T R^{-1} (y - \hat{\mu}\mathbf{1}).$$

Questa formula mostra bene la logica del metodo: il primo termine $\hat{\mu}$ rappresenta la previsione basata solo sulla regressione costante, mentre il secondo termine è una correzione dovuta alla correlazione con i punti vicini.

- Se x^* è vicino a un punto campionato, la correzione è grande e l'interpolazione si adatta al dato.
- Se x^* è molto lontano dai dati, $r \approx 0$, quindi $\hat{y}(x^*) \approx \hat{\mu}$.

Il modello DACE quindi interpola i dati nei punti campionati e tende alla media stimata lontano da essi.

9.3.1.1 L'errore di previsione

Anche l'incertezza associata alla previsione può essere calcolata in forma chiusa. La **varianza della previsione** in x^* è:

$$s^2(x^*) = \sigma^2 \left[1 - r^T R^{-1} r + \frac{(1 - \mathbf{1}^T R^{-1} r)^2}{\mathbf{1}^T R^{-1} \mathbf{1}}\right].$$

- Se $x^* = x^{(i)}$, la varianza si annulla ($s^2(x^{(i)}) = 0$), perché il modello interpola esattamente i dati.
- Se x^* è molto lontano dai dati, $r \approx 0$ e la varianza tende a σ^2.

La radice quadrata $s(x^*)$ fornisce il **RMSE (Root Mean Squared Error)**, che rappresenta l'incertezza della previsione.

Il DACE può essere visto come una sorta di "immagine speculare" della regressione lineare.

- Nella regressione, si scelgono regressori (polinomi, funzioni base) e si assumono errori indipendenti.
- Nel DACE, si usa un regressore banale (solo una costante), ma si modella in dettaglio la struttura di correlazione degli errori.

Per questo i due approcci sono in qualche modo complementari: la regressione si concentra sulla forma della funzione, il DACE sulla correlazione locale tra i dati.

Il modello DACE permette quindi di interpolare ed estrapolare in maniera coerente i valori di una funzione simulata o sperimentale, catturandone il **comportamento caratteristico** attraverso i parametri di correlazione. Le previsioni risultano esatte nei punti campionati e smussate altrove, con un'incertezza che dipende dalla distanza dai dati osservati.

10 Robust Design

L'Ottimizzazione Multidisciplinare (Multidisciplinary Design Optimization, MDO) sta assumendo un'importanza sempre maggiore, in particolare nella comunità aerospaziale. L'Associazione AIAA (American Institute of Aeronautics and Astronautics) ha organizzato numerose sessioni dedicate al MDO e, recentemente, si è svolta la Prima Sessione dedicata ai MDO per specialisti. Questo crescente interesse riflette l'esigenza di sviluppare metodologie numeriche avanzate in grado di affrontare problemi complessi di progettazione, supportando l'industria durante le fasi di sviluppo di sistemi sofisticati.

È utile sottolineare che i progetti aeronautici sono intrinsecamente molto complessi, sia per la complessità dei modelli fisici coinvolti, sia per l'elevato numero di parametri di ingresso e di uscita che devono essere considerati. La gestione efficace di questa complessità è fondamentale per ottenere soluzioni ottimali e affidabili.

Un aspetto cruciale nella progettazione industriale riguarda la gestione delle incertezze, con l'obiettivo di individuare soluzioni che siano poco sensibili alle variazioni stocastiche dei parametri. Questo approccio prende il nome di "Progettazione Robusta" (Robust Design).

L'esigenza di metodologie di Progettazione Robusta si manifesta in molti contesti, e in particolare nel MDO. Infatti, le incertezze possono presentarsi in diverse fasi: durante il processo di progettazione preliminare, i valori esatti di alcuni parametri di input potrebbero non essere ancora noti, oppure i parametri stessi potrebbero subire modifiche nelle fasi successive del progetto. In tali circostanze, l'obiettivo principale diventa la ricerca di soluzioni che dipendano il meno possibile dai parametri incerti o variabili.

Altre considerazioni importanti riguardano la capacità di ottenere soluzioni che siano insensibili alle tolleranze di fabbricazione, alle fluttuazioni delle condizioni operative e alle variazioni numeriche che possono emergere nei modelli di simulazione ad alta fedeltà. Garantire tale robustezza permette di aumentare l'affidabilità complessiva del progetto e di ridurre i rischi associati a condizioni operative o di produzione impreviste.

Lo studio delle incertezze in ingegneria ha le sue origini nei lavori di Taguchi, il quale ha formalizzato una metodologia per l'ingegneria della qualità. Taguchi suddivide il processo di progettazione in tre fasi distinte: la prima fase, chiamata **progettazione di sistema (system design)**, ha il compito di determinare la regione più fattibile all'interno della quale verranno condotte le ottimizzazioni numeriche successive; la seconda fase, denominata **progettazione robusta (robust design)**, serve a identificare i parametri ottimali che massimizzano la qualità finale del sistema considerato; infine, nella terza fase, detta **progettazione delle tolleranze (tolerance design)**, viene effettuata una taratura finale dei parametri per ottenere la migliore soluzione possibile, considerando le inevitabili variabilità dei componenti e delle condizioni operative.

La necessità di studiare le incertezze è particolarmente rilevante in ambito aeronautico. A conferma di ciò, l'AIAA (American Institute of Aeronautics and Astronautics) definisce l'incertezza come segue:

Definizione Incertezza: *una potenziale carenza in qualsiasi fase o attività del processo di modellazione, dovuta alla mancanza di conoscenza.*

Si osservi che l'incertezza è quindi definita come una **mancanza di conoscenza**, il che implica l'esigenza di adottare approcci specifici per analizzare e studiare il modello, diversi da quelli tradizionali usati in assenza di incertezze.

Dal punto di vista numerico, lo studio di un modello influenzato da incertezze può essere formalizzato come:

$$f: A \times B \to \mathbb{R}$$

dove:

- $a \in A$ rappresenta i parametri di progetto scelti dai progettisti;
- $b \in B$ rappresenta i parametri di ingresso permeati da incertezze, e quindi non controllabili dai progettisti.

Le incertezze comuni (b) relative all'aerodinamica esterna sono descritte in dettaglio, in particolare nel contesto della progettazione di profili alari bidimensionali. Esse comprendono principalmente:

- **Incertezze sui parametri geometrici**, dovute alle tolleranze di fabbricazione, che modificano i parametri geometrici della superficie alare. Questa situazione è trattata approfonditamente in [6], dove viene progettato un profilo alare con **resistenza minima** ottimizzato rispetto alle incertezze geometriche.
- **Incertezze sulle condizioni operative** (design point), generalmente analizzate considerando fluttuazioni sul numero di Mach del flusso libero $[M_{\min}, M_{\max}]$.

Questa formalizzazione consente di analizzare numericamente l'impatto delle incertezze sul comportamento del sistema, e rappresenta la base metodologica per lo sviluppo di approcci di **progettazione robusta e ottimizzazione multidisciplinare**, dove l'obiettivo non è solo trovare la soluzione ottimale nominale, ma anche garantire la sua **affidabilità rispetto alle variazioni inevitabili dei parametri**.

Molti studi dimostrono chiaramente perché lo studio delle fluttuazioni sia di primaria importanza nell'aerodinamica esterna. Nello studio si mostra come, nel caso della minimizzazione del **coefficiente di resistenza (drag coefficient)** di un profilo alare sotto condizioni operative fisse — in particolare per un determinato **numero di Mach del flusso libero** — la soluzione finale ottimizzata presenti **buone prestazioni nel punto di progetto**, ma manifesti caratteristiche scadenti al di fuori del punto di progetto (off-design), come illustrato in Figura 1.

Questo fenomeno è noto come **over-optimization** (iperottimizzazione), e rappresenta un problema cruciale nella progettazione aeronautica: una soluzione troppo ottimizzata per condizioni specifiche rischia infatti di essere poco robusta, mostrando prestazioni significativamente degradate quando le condizioni operative variano. Lo studio delle fluttuazioni, quindi, diventa essenziale per sviluppare progetti che siano non solo efficienti nel punto nominale, ma anche **robusti e affidabili in un intervallo realistico di condizioni operative**, riducendo i rischi associati a variazioni non prevedibili dei parametri di volo o della geometria.

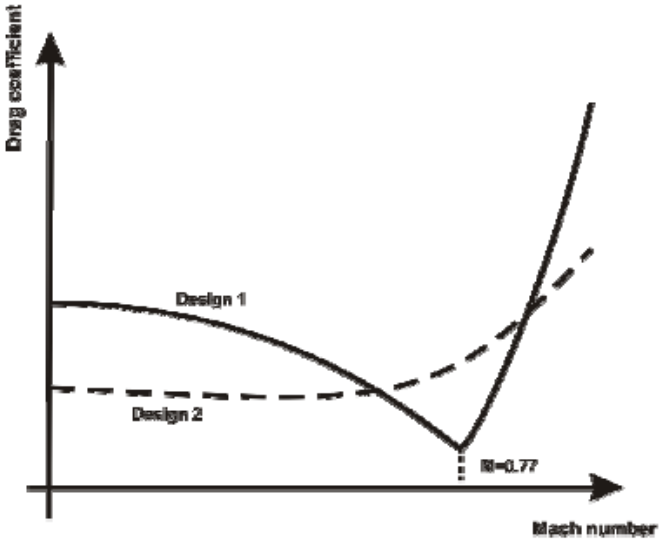


Figure 1: Drag profile for stable (Design 2) e not stable (Design 1) solution respect to Mach number

Questo comportamento diventa ancora più evidente nel caso dei **profili supercritici**, dove la relazione tra resistenza aerodinamica (drag) e velocità del flusso libero è **non lineare**, a causa delle fluttuazioni nella posizione delle onde d'urto sulla superficie del profilo alare.

Di conseguenza, la possibilità di determinare soluzioni che mantengano **buone prestazioni su un intervallo di condizioni operative** risulta particolarmente interessante, anche per evitare cambiamenti improvvisi nel comportamento del sistema. Vale la pena ricordare che un comportamento stabile contribuisce a **minimizzare il rischio operativo** del sistema, aumentando l'affidabilità complessiva del progetto.

Numerosi metodi numerici sono stati sviluppati per ottimizzare sistemi in presenza di incertezze, in particolare nel caso di fluttuazioni delle condizioni operative.

E' stata proposta una metodologia basata direttamente sulla **ottimizzazione multi-points**, utilizzando una formulazione con somma ponderata dei risultati. Nell'esempio esaminato, si effettua la **minimizzazione del coefficiente di resistenza** di un profilo alare vincolato al mantenimento del portanza, considerando incertezze sul numero di Mach di crociera. La formulazione proposta è:

$$\min_{a} D = \sum_{i=1}^{n} w_i \; C_d(M_i, a_i, d_i)$$

soggetta al vincolo:

$$C_l(M_i, a_i, d_i) \geq C_l^*, \quad \text{per } 1 \leq i \leq n$$

dove:

- w_i sono **pesi arbitrari**,
- d_i rappresenta l'insieme delle **variabili geometriche di progetto** che definiscono il profilo alare,
- C_d e C_l sono rispettivamente i **coefficienti di resistenza e portanza**, definiti come funzione del numero di Mach M_i e dell'angolo di attacco a_i, che possono fluttuare attorno ai valori nominali di progetto,
- C_l^* è la portanza richiesta.

Il principale limite di questa formulazione è che il risultato finale dipende fortemente dalla scelta dei **pesi** w_i, i quali risultano troppo arbitrari e difficili da definire in maniera oggettiva. Questo rende necessario lo sviluppo di metodologie più robuste e indipendenti dalla scelta dei pesi, in grado di garantire soluzioni stabili e affidabili su tutto l'intervallo delle condizioni operative.

Viene anche introdotto un **nuovo concetto per la Progettazione Robusta (Robust Design)**, adottando un approccio differente rispetto alla tradizionale ottimizzazione multi-punto. L'idea innovativa consiste nella **formulazione di un rischio** r da minimizzare:

$$r_{\min} = \int_{M_{\min}}^{M_{\max}} C_d\,(d, M)\, f(M)\, dM$$

dove $f(M)$ rappresenta la **densità di probabilità** del numero di Mach in crociera. In questo contesto, il rischio, preso dalla teoria bayesiana, rappresenta il **valore medio del coefficiente di resistenza** considerando le fluttuazioni delle condizioni operative. L'autore propone di risolvere l'integrale mediante una **serie di Taylor di secondo ordine**, per ottenere un calcolo approssimato ma efficace.

Al fine di evitare l'arbitrarietà della formulazione del Robust Design viene presentata una metodologia alternativa molto interessante. Questa metodologia modifica **iterativamente il numero di Mach** nell'equazione di partenza (Eq. 1) per calcolare l'integrale della Eq. 3 utilizzando la **regola del trapezio**, rendendo il calcolo più sistematico e meno dipendente da scelte arbitrarie.

Un ulteriore approccio innovativo è proposto in uno studio, dove gli autori dimostrano che il problema della Progettazione Robusta **deve essere affrontato tramite un approccio di Ottimizzazione Multi-Obiettivo (Multi Objective Optimization, MOO)**. Partendo dalla definizione di stabilità, il problema numerico si formalizza come:

$$\min(E[C_d(d, M)],\, s[C_d(d, M)])$$

soggetto al vincolo:

$$C_l(d, M) = C_l^*, \quad M \in W$$

dove:

- $E[C_d(d, M)]$ rappresenta il valore medio del coefficiente di resistenza,
- $s[C_d(d, M)]$ rappresenta la varianza del coefficiente di resistenza, definita come:

$$E[C_d(d)] = \int_{M_{\min}}^{M_{\max}} C_d\,(d, M)\, p(M)\, dM$$

$$s^2[C_d(d)] = \int_{M_{\min}}^{M_{\max}} (\,C_d(d, M) - E[C_d(d)])^2\, p(M)\, dM$$

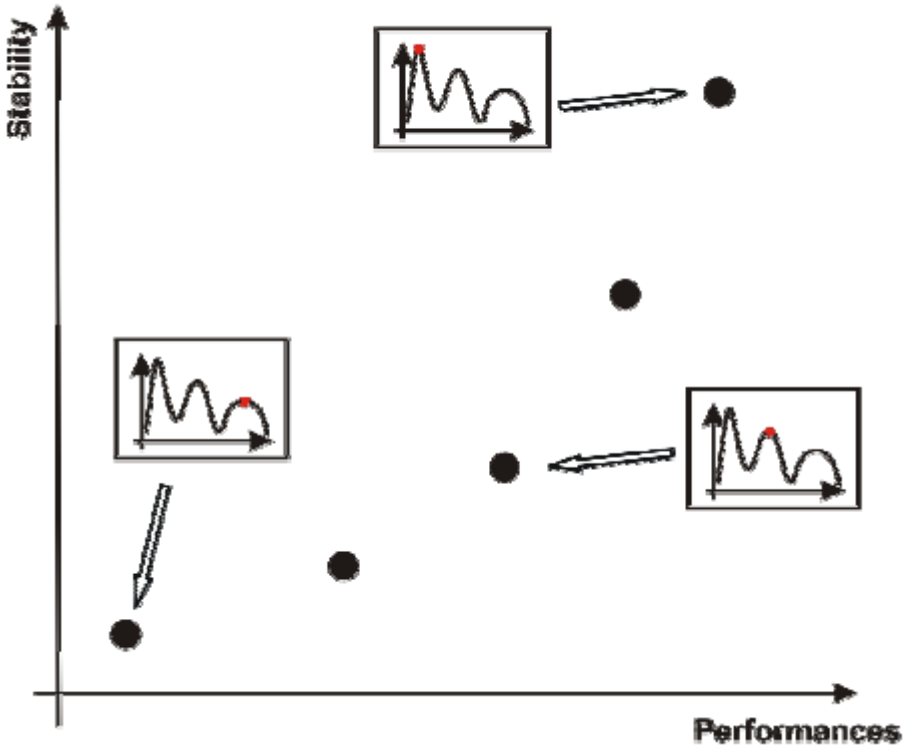
con $p(M)$ funzione di densità di probabilità del numero di Mach definita nell'intervallo $M_{\min} < M < M_{\max}$.

Questa formulazione di **Robust Design** permette di esplorare due direzioni principali nell'ottimizzazione:

- Minimizzando la varianza del coefficiente di resistenza, è possibile ridurre il degrado delle prestazioni fuori dal punto di progetto (off-design), come mostrato nella figura 2 nel design "inferiore".
- Ottimizzando il valore medio delle prestazioni (in questo caso il coefficiente di resistenza), si privilegiano le prestazioni nominali, come illustrato nella figura 2 nel design "superiore".

È importante notare che questa formulazione si basa su un **approccio multi-obiettivo**, quindi il risultato finale è rappresentato dalla **frontiera di Pareto**, cioè l'insieme delle migliori soluzioni di compromesso tra le diverse funzioni obiettivo (Figura 2, tutte le soluzioni della frontiera di Pareto).

La principale difficoltà di questo approccio risiede nel **numero elevato di analisi ad alta fedeltà** necessarie per determinare la frontiera di Pareto. Per questo motivo, viene proposta una metodologia alternativa che utilizza una **direzione di discesa** in grado di ridurre il coefficiente di resistenza in modo **simultaneo e proporzionale** su tutto l'intervallo di numeri di Mach considerati, rendendo l'ottimizzazione più efficiente e meno costosa computazionalmente.



Interessante è anche l'approccio presentato in [12], dove viene proposta una funzione a **somma ponderata** (simile all'Eq. 1). Per evitare l'arbitrarietà della scelta dei pesi, l'autore propone una **metodologia adattativa** per determinarli:

$$w_i^{\text{new}} = w_i^{\text{old}} + \frac{1}{N}\sum_{i=1}^{N}(C_{d,i} - C_{d,\text{avg}})$$

Questa formula consente di aggiornare i pesi in maniera iterativa, basandosi sulle prestazioni del sistema, riducendo così l'influenza di scelte iniziali arbitrarie e rendendo la procedura più sistematica e automatica.

Per ridurre l'elevato costo computazionale associato alla risoluzione dei problemi di **Robust Design**, Trosset ha proposto nuovi metodi basati sulla **teoria decisionale statistica**, in particolare sfruttando la formulazione bayesiana. Partendo dal metodo di Taguchi per l'ingegneria della qualità, Trosset ha dimostrato come sia possibile sostituire una funzione obiettivo **computazionalmente costosa** con un **modello numerico surrogato**, meno oneroso, utilizzando la metodologia **DACE (Design and Analysis of Computer Experiments)** sviluppata da Sachs e Welch .

Infine, è importante sottolineare che un campo di applicazione particolarmente interessante per il **Robust Design** è la **progettazione preliminare di sistemi ingegneristici complessi**, che coinvolgono **sottosistemi multidisciplinari**. In questo contesto, l'obiettivo del Robust Design è ridurre gli effetti negativi sulle prestazioni dell'intero sistema causati dalle decisioni prese nella progettazione di un singolo sottosistema, come illustrato in Figura 3.

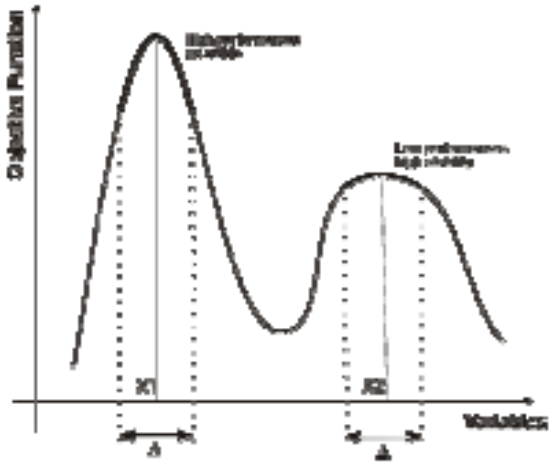
Questo approccio garantisce che, anche in presenza di incertezze o di modifiche nei sottosistemi, le prestazioni complessive rimangano **stabili e affidabili**, evitando che un'ottimizzazione locale comprometta il funzionamento globale del sistema.

In questo capitolo viene presentato un **nuovo metodo per l'ottimizzazione della Progettazione Robusta (Robust Design)**. L'idea principale consiste nell'utilizzare un **approccio multi-obiettivo** per raggiungere il miglior compromesso possibile tra **prestazioni** e **stabilità** del progetto.

Facendo riferimento alla Figura 4, la funzione considerata presenta un **estremo assoluto** e un **estremo relativo**, corrispondenti rispettivamente alle coordinate x_1 e x_2. In questo caso, le **incertezze** sono rappresentate dalla **tolleranza** D del parametro di ingresso x (come nel caso delle tolleranze di fabbricazione).

Ovviamente, un'ottimizzazione standard, che non tenga conto delle fluttuazioni, identificherebbe il punto x_1, che rappresenta il massimo assoluto, ma caratterizzato da **scarsa stabilità**. Nel caso dell'**ottimizzazione Robust Design**, considerando la tolleranza D, occorre valutare due obiettivi distinti: le **prestazioni medie** e la **stabilità** delle soluzioni, secondo le idee presentate in [8].

Se si considera la prestazione media all'interno della tolleranza D, la configurazione ottimale risulterebbe x_1, poiché il valore medio della funzione è il più alto. Tuttavia, per quanto riguarda la stabilità, valutata come la **varianza della funzione** $f(x)$ all'interno del campo D (Eq. 7), la configurazione migliore è rappresentata dal punto x_2, in quanto la funzione mostra una **minore variabilità** all'interno della tolleranza attorno a x_2.



È quindi interessante osservare che, nell'ottimizzazione Robust Design, la **regione più stabile** non corrisponde necessariamente a quella più performante. Per ottimizzare un sistema sotto fluttuazioni, il modo migliore consiste nel definire **due obiettivi distinti** per ogni funzione da ottimizzare: il **valore medio** e la **varianza**. In termini matematici:

$$E[f] = \int f(x,q)\, p(q)\, dq, \quad s^2[f] = \int (f(x,q) - E[f])^2\, p(q)\, dq$$

dove f è la funzione multi-obiettivo da massimizzare e q sono i parametri di incertezza, modellati tramite la **funzione di densità di probabilità** $p(q)$.

In questo modo, il problema dell'ottimizzazione sotto incertezze si trasforma in un **problema di Ottimizzazione Multi-Obiettivo**, in cui gli obiettivi sono **stabilità** e **prestazioni**. Per risolvere questo problema è consigliabile adottare la **Teoria dei Giochi (Game Theory, vedi Capitolo 2)**, che rappresenta la metodologia più adatta per affrontare un vero problema multi-obiettivo senza ricorrere a funzioni ponderate come:

$$\max w_1 f + w_2 s$$

Assegnare un valore ai pesi w_i risulta infatti delicato e spesso arbitrario, ed è per questo che l'approccio basato sulla **Game Theory** è preferibile.

È interessante notare che, al termine della fase di ottimizzazione, utilizzando un approccio basato sulla **Pareto Frontier**, il progettista non ottiene una sola soluzione, ma un **insieme di soluzioni** (la frontiera di Pareto) che rappresenta il **miglior compromesso possibile tra gli obiettivi**. Un esempio di frontiera di Pareto per un'ottimizzazione Robust Design è mostrato nella Figura 2; all'interno della frontiera di Pareto è possibile scegliere diversi compromessi tra **prestazioni e stabilità**, offrendo maggiore flessibilità rispetto a un'ottimizzazione standard, dove la soluzione è unica.

È importante sottolineare che l'approccio Robust Design può affrontare un'ampia gamma di problemi, come **piccoli errori di processo di fabbricazione, fluttuazioni nelle condizioni operative, parametri di ingresso incerti**, ecc. Inoltre, il metodo è estendibile a più funzioni da ottimizzare: ad esempio, è possibile migliorare simultaneamente **portanza e resistenza** di un profilo alare in presenza di fluttuazioni della velocità di volo, senza necessità di una funzione ponderata per legare le due prestazioni differenti.