POLITECNICO DI TORINO

Corso di Laurea Magistrale in

Ingegneria Aerospaziale

Tesi di laurea magistrale

Miglioramento di modelli di turbolenza RANS per flussi interni

Relatore
Prof. Francesco Larocca
Correlatore
Ing. Andrea Ferrero

Candidato
Gianmarco Iposi
Matricola 252671

A.A 2020

Ringraziamenti

Innanzitutto volevo ringraziare i miei genitori per tutto il sostegno che mi hanno dimostrato per tutta la vita e, in particolar modo, in questi anni da studente universitario, prima a Padova ed ora a Torino. Con il loro appoggio ho potuto affrontare al pieno delle mie capacità questo percorso e grazie ai loro insegnamenti sono maturato come persona.

In seguito ringrazio il Professor Larocca e l'Ingegner Ferrero per avermi dato la possibilità di sviluppare un lavoro così innovativo per il mio percorso di studi, permettendomi di esplorare l'orizzonte delle discipline legate all'ambiente fluidodinamico. Questo lavoro è stato possibile grazie ad una completa supervisione dei miei relatori, che si sono dimostrati sempre disponibili nell'aiutarmi a superare le mie incertezze e a chiarire ogni mio dubbio.

Infine, ma non per importanza, ringrazio tutti i miei amici che mi hanno sostenuto moralmente. Inoltre, nonostante le molte assenze, mi hanno sempre fatto sentire parte di un gruppo, presentandosi tanto nei momenti di difficoltà quanto in quelli di gioia.

Sommario

L'obiettivo di questa tesi è l'applicazione degli algoritmi di machine learning per ottenere un miglioramento dei modelli di turbolenza in ambito CFD. L'applicazione delle cosiddette simulazioni high-fidelity ottenute utilizzando i modelli DNS o LES, nonostante la loro capacità di rappresentare fedelmente la fenomenologia del flusso in esame, sono applicabili solo per geometrie semplici e caratterizzate da bassi numeri di Reynolds. Ben più sfruttati sono i modelli RANS che prevedono la scomposizione della quantità fluidodinamiche in una componente media ed una fluttuante. Data la natura approssimata dell'approccio RANS risulta evidente come insorgono delle discrepanze con il reale andamento del campo di moto. Nella tesi è stato scelto come modello di turbolenza quello di Spalart-Allmaras (SA). Per migliorare la capacità di soluzione dei modelli approssimati si utilizza il paradigma FIML (field inversion and machine learning) che si basa su due step principali: il primo, noto con il termine di inversione di campo, si basa sulla modifica del modello scelto introducendo un parametro che agisce sul termine di produzione di SA, il cui valore viene valutato puntualmente per ogni punto della discretizzazione tramite un processo di ottimizzazione realizzato con il metodo del gradiente. Per monitorare la qualità della soluzione ottenuta con il modello modificato si sfruttano i dati ricavati da una simulazione LES per il medesimo test case, che nello specifico di questo lavoro risulta essere un condotto curvo bidimensionale costituito da due pareti parallele di cui quella inferiore soggetta alla viscosità. Come variabile di confronto tra i due modelli è stato scelto il coefficiente di attrito a parete poiché si dispone del suo andamento in funzione della coordinata curvilinea della parete inferiore nel modello LES. In seguito si definisce una funzione obiettivo di cui si calcolerà il gradiente con il metodo dell'aggiunto. Questa funzione è la norma due dell'errore tra il coefficiente di attrito ricavato mediante simulazione LES e quello approssimato da SA lungo la parete solida del condotto. Tale discretizzazione è stata ricavata mediante il metodo degli elementi finiti discontinui di Galerkin per la griglia spaziale, mentre per quella temporale è stato utilizzato il metodo implicito di Eulero. A questo punto si imposta un problema di minimizzazione della funzione obiettivo muovendosi in direzione opposta al gradiente calcolato che causerà una variazione del termine di produzione che si ripercuoterà sul coefficiente di attrito. Procedendo per iterazioni successive è possibile ottenere un modello RANS molto più accurato. Una volta ricavata una distribuzione ottimale del parametro si procede con la seconda parte del paradigma, relativa al machine learning.

L’obiettivo è quello di estrapolare dal campo di moto delle variabili fluidodinamiche significative e cercare un’espressione del parametro in funzione di tali variabili. In questo modo non solo si ottiene un miglioramento dell’accuratezza del modello di SA nel caso in esame, ma si ottiene la possibilità di applicare il modello migliorato anche in problemi simili in quanto il coefficiente di correzione viene espresso in funzione di variabili fisiche. In questo lavoro sono state utilizzate e confrontate le soluzioni ricavate mediante reti neurali artificiali feedforward e programmazione genetica. In particolare si andranno a valutare i vantaggi e le criticità dei due approcci.

INDICE

INDICE DELLE FIGURE

INTRODUZIONE

1.1 Generalità sul machine learning

Il machine learning è una branca dell'informatica e può essere considerata come un sottoinsieme dell'intelligenza artificiale (AI) che si occupa di creare sistemi che apprendono e migliorano le performance in base ai dati di input utilizzati. Una delle definizioni più accreditate in questo ambito è quella coniata da Tom M.Mitchell (M.Mitchell, 1997), secondo il quale:" A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience". Supponendo di voler istruire un computer a riconoscere le diverse lingue che gli vengono presentate, tale classificazione rappresenta il compito T, mentre l'esperienza E è espressa dalle immagini/campioni che uso per istruire il programma, in quanto esse possiedono delle caratteristiche peculiari. La prestazione P non è altro che un indice della capacità del programma di eseguire il compito con successo. Quindi il machine learning rappresenta l'abilità di un computer di migliorare nell'esecuzione di un compito mediante l'analisi dei dati. Le prime sperimentazioni in questo ambito risalgono al 1950, quando gli scienziati iniziarono a pensare di utilizzare l'analisi delle probabilità per realizzare macchine che potessero prendere decisioni basate sulla possibilità di realizzazione di un dato evento. Uno degli esponenti di questo movimento è Alan Turing che ipotizzò la necessità di realizzare algoritmi specifici per la realizzazione di macchine "sapienti", ovvero in grado di apprendere (A.M.Turing, 1950). Nello stesso periodo vennero avanzati diversi studi sull'intelligenza artificiale e sulle reti neurali che tuttavia trovarono un forte ostacolo causato dalle difficoltà realizzative di questi sistemi intelligenti, dalla mancanza di investimenti in questo ambito di ricerca e dallo scetticismo popolare. Negli anni Ottanta arriva un grande successo in ambito di machine learning rappresentato dallo "Stanford Cart" ad opera degli studenti della Stanford University; si tratta di un robot dotato di software capace di evitare gli ostacoli in una stanza in maniera autonoma mappando i percorsi. Nonostante i successi i sistemi basati sulla statistica non vennero più studiati, preferendo l'approccio adottato da altre forme del machine learning. Solo negli anni Novanta il machine learning viene riscoperto, grazie ad un diverso ambito di applicazione non più incentrato sulla realizzazione di una AI, ma sulla soluzione di problemi di natura pratica.

A seconda del tipo di algoritmo utilizzato per l'apprendimento, ovvero a seconda delle modalità con cui la macchina impara ed accumula dati ed informazioni, si possono avere tre sistemi di apprendimento: supervisionato, non supervisionato e per rinforzo.

- L'apprendimento supervisionato consiste nel fornire al sistema informatico una serie di esempi ideali costituiti da coppie di input e output grazie ai quali esso elabora automaticamente previsioni su un nuovo database di ingresso. Alcuni esempi di questo metodo si trovano nell'identificazione della scrittura manuale o riconoscimento vocale. Si dice che il sistema elabori un'ipotesi induttiva (funzione) in grado di apprendere da valori noti (quelli forniti in fase di learning) e capace di avvicinarsi a dei risultati desiderati per tutti gli esempi non forniti. Si deduce quindi come il funzionamento di questi algoritmi sia strettamente legato all'esperienza: pochi dati di input possono portare alla realizzazione di una funzione poco efficiente, mentre con un'esperienza eccessiva la funzione potrebbe avere una complessità tale da rendere lenta l'esecuzione dell'algoritmo. Ad oggi le due classi principali di algoritmi possibili sono:

    1. Metodi generativi

       Si basano sulla realizzazione di un modello che viene successivamente utilizzato per predire gli output.

    2. Metodi discriminativi

       Quest'ultimi, al contrario, cercano di creare un modello direttamente dalla relazione intrinseca tra dati di ingresso ed uscita in modo da minimizzare una determinata funzione goal.

- L'apprendimento non supervisionato prevede che il sistema possa attingere a determinate informazioni senza avere alcun esempio del loro utilizzo. La macchina cerca di capire come catalogare tutte le informazioni, organizzarle ed impararne il significato, il loro utilizzo ed il risultato a cui portano. Si tratta di un approccio che offre maggiore libertà al sistema. Un esempio tipico di questi algoritmi lo si ha nei motori di ricerca. Le due classi principali sono rappresentate da: clustering e regole di associazione.

- L'apprendimento per rinforzo prevede che il sistema sia dotato di strumenti in grado di migliorare il proprio apprendimento e di comprendere le caratteristiche

dell'ambiente circostante. A differenza dei due sopracitati, questa tecnica di machine learning si occupa di problemi di decisioni sequenziali in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro. Il sistema deve essere dotato di una serie di elementi di supporto che permettono di rilevare quanto avviene nell'ambiente ed effettuare delle scelte per migliorare il loro adattamento in quest'ultimo. Esempio tipico di questo apprendimento è quello delle auto senza pilota che grazie ai sensori di posizione è in grado di percorrere le strade.

1.2 Applicazioni quotidiane

Oggigiorno le tecniche di apprendimento automatico si possono trovare in ogni ambito:

1) Riconoscimento facciale

   Si tratta di un tipo di approccio per identificare e caratterizzare le features di un'immagine. È una soluzione tecnologica in grado di identificare una persona attraverso la valutazione dell'immagine del suo volto basandosi su un database di foto che la ritraggono. Inoltre è possibile carpire da quelle foto altre caratteristiche quali età, genere, etnia, sentimento manifestato ed altro.

2) Classificazione delle notizie

   Permette all'utente di selezionare un particolare tipo di informazione da estrapolare da un determinato sito. Tra le macchine più utilizzate per la sua realizzazione ci sono le macchine a vettori di supporto, che sono dei modelli di apprendimento supervisionato associato ad algoritmi di apprendimento per la regressione e la classificazione.

3) Classificazione di nuove strutture astronomiche e previsione delle orbite dei detriti spaziali

   La NASA ha utilizzato algoritmi di apprendimento basati su alberi di decisione per classificare oggetti celesti per il Palomar Observatory Sky Survey (Nicholas Weir, 1995). Algoritmi di tipo gaussiano o di rete neurale sono stati sviluppati per prevedere la traiettoria di alcuni oggetti presenti in diverse orbite (Hao Peng, 2019)

4) Filtraggio e-mail/spam ed antivirus

   Entrambi i modelli richiedono un database di input che permetta all'algoritmo di distinguere tra le varie categorie ed immagazzinare al proprio interno un certo schema di riconoscimento.

5) Social media

   In questo gruppo fanno parte i già citati algoritmi per il riconoscimento facciale, ma non solo; infatti i social network (es. Facebook) suggeriscono all'utente persone che potrebbero conoscere in quanto sono state rappresentate nelle stesse foto o condividono il luogo di nascita, oppure altre caratteristiche similari carpite dal machine learning.

6) Marketing

   Un utente visita il sito Amazon perché interessato all'acquisto di un determinato articolo. L'algoritmo, al prossimo accesso, non solo gli mostrerà quello che già aveva cercato, ma anche una serie di articoli che possono essere di gradimento dell'utente.

7) Medicina

   In questo ambito le tecniche di machine learning hanno permesso di sviluppare modelli di previsione sulla possibilità di essere affetti da un determinato tipo di malattia in base a dei parametri in ingresso. Per maggiori informazioni in questo ambito consiglio di leggere il seguente articolo (Konstantina Kourou, 2014).

8) Giochi

   Un esempio classico di machine learning applicato al mondo dei giochi è quello di TD Gammon, il miglior programma di computer al mondo per il backgammon, il quale ha sviluppato la sua strategia giocando una moltitudine di partite contro sé stesso.

9) Finanza

   Siamo nell'ambito della prevenzione delle frodi, dei furti di dati e di identità. Gli algoritmi imparano ad agire mettendo in correlazione eventi, abitudini degli utenti, preferenze di acquisto per identificare eventuali comportamenti anomali.

Bibliografia

A.M.Turing. (1950). Computing Machinery and Intelligence. *Mind*, 433-460.

Hao Peng, X. B. (2019). Comparative evaluation of three machine learning algorithms on improving orbit prediction accuracy. *Astrodyn 3*, 352-343.

Konstantina Kourou, T. P. (2014). Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 9-16.

M.Mitchell, T. (1997). *Machine learning.* McGraw-Hill Science/Engineering/Math.

Nicholas Weir, U. M. (1995). Automated star/galaxy classification for digitized poss-II. *The astronomical journal*, 2041-2414.

Sito web http://www.intelligenzaartificiale.it/machine-learning/

CAPITOLO 2

2.1 Stato dell'arte dei modelli CFD

Le turbomacchine hanno diversi campi di applicazione, ma quello di nostro interesse è legato alla propulsione dei velivoli. Il design di una turbina a gas ad alta efficienza richiede un tradeoff tra requisiti, vincoli progettuali e un progetto di design aerodinamico e termico molto accurato, infatti l'efficienza globale di una turbomacchina nella conversione di energia in potenza dipende dall'efficienza termodinamica, dall'efficienza aerodinamica, dall'efficienza termica ed infine da quella propulsiva (Gregory M Laskowski, 2016). Ad oggi, il design di queste macchine viene coadiuvato dai metodi computazionali, i quali hanno un impatto diretto sulla previsione delle eventuali variazioni delle efficienze sopra riportate sulle prestazioni del sistema. Tuttavia questi metodi devono essere affidabili, poiché un errore nella previsione si ripercuote nell'ambito economico in quanto si rendono necessarie delle nuove verifiche e, dal punto di vista concorrenziale, un ritardo nella produzione e la conseguente uscita sul mercato. Una possibile soluzione a questi errori di predizione in fase di design può essere utilizzare ampi intervalli di margini di sicurezza. Il problema nel loro utilizzo risiede nel fatto che le turbomacchine vengono scelte anche in base al loro ritorno di investimento. Si immagini di utilizzare un coefficiente di sicurezza per sottostimare l'efficienza totale del sistema: un velivolo con efficienza bassa tende a bruciare più combustibile o generare meno potenza, quindi è necessario più tempo per poter vedere una qualche forma di guadagno sull'investimento. Ecco quindi la necessità di un metodo di design sempre più accurato. La generazione di turbine a gas ad oggi presente sul mercato è il risultato di oltre 6 decenni di design; stiamo parlando quindi di una tecnologia matura che presenta come punti di forza sia l'efficienza sia l'affidabilità. Ci si è posti dunque il problema su come ricavare delle migliorie in termini di prestazioni e durata. A tal fine, le industrie hanno improntato le proprie ricerche per migliorare l'efficienza della conversione di energia sull'utilizzo di nuovi materiali per la costruzione, su nuove tecniche di manufacturing, studi approfonditi di aerodinamica, termodinamica ecc. Uno dei metodi computazionali più usati è la CFD (Computational Fluid Dynamics) utilizzata soprattutto per il design di compressori, turbine e combustori. Nonostante il suo ampio utilizzo, anche questo metodo presenta delle imprecisioni dovute alla natura stocastica della turbolenza e alle interazioni tra statore e rotore che non permettono alla CFD di catturare con un

ampio margine di certezza la fenomenologia del flusso. I due fenomeni sopra citati sono caratterizzati da diverse frequenze e la loro interazione all'interno del flusso porta ad uno scambio di entalpia e di momento che, a loro volta, accrescono le irreversibilità del flusso ed abbattono di conseguenza l'efficienza. I metodi utilizzati dalla CFD per la risoluzione delle equazioni di Navier-Stokes si basano sui seguenti modelli:

- DNS (Direct Numerical Simulation)

  Le equazioni vengono risolte numericamente nella loro formulazione tridimensionale e non stazionaria. I risultati sono accurati a discapito di un costo computazionale elevatissimo (numero di celle proporzionale a $R_{e_l}^{\frac{9}{4}}$ e passi temporali necessari proporzionali a $R_{e_l}^{\frac{7}{8}}$ dove R_{e_l} è il numero di Reynolds basato sulla dimensione caratteristica l). Dunque questo tipo di analisi è limitato a flussi caratterizzati da bassi Re (poco turbolenti) e per geometrie semplici.

- LES (Large Eddy Simulation)

  Si basa sulle evidenze sperimentali che le strutture turbolente di dimensioni maggiore possiedono caratteristiche anisotropiche e dipendenti dal sistema specifico che costituiscono il contributo energetico più rilevante e quindi sono responsabili della gran parte degli effetti. Viceversa, le strutture più piccole sono isotropiche ed universali, ovvero non dipendono dal sistema specifico. Di conseguenza si utilizza un approccio numerico 3D non stazionario per le scale turbolente maggiori (large eddies) mentre quelle più piccole sono lasciate ad un opportuno modello. In termini di costo computazionale risultano a metà strada tra RANS e DNS, tuttavia in problemi di tipo industriale (caratterizzati da geometrie complesse e Reynolds elevati) sono comparabili alle DNS.

- RANS

  L'idea di fondo di questo modello risiede nell'interesse, per fini ingegneristici, nei valori medi delle grandezze di flusso per ottenere un andamento medio del campo di moto. Ogni quantità è scomposta in un

termine mediato nel tempo più un disturbo (o fluttuazione) attorno al valor medio. Il costo computazionale risulta notevolmente ridotto.

- URANS (Unsteady Reynolds Averaged Navier Stokes)

  RANS applicate ai casi di flusso non stazionario

I calcoli effettuati con questi modelli forniscono una visione preliminare sul campo di moto all'interno di una turbina a gas, ma spesso a discapito di tempo computazionale o di difficoltà realizzative per le reali condizioni operative (elevate pressioni e temperature). Nella pratica, un progettista di turbine a gas deve essere in grado di incorporare i margini di sicurezza all'interno dei risultati CFD per essere sicuro di incontrare i requisiti progettuali nonostante le incertezze nella previsione dei risultati. Infatti la soluzione per superare la stagnazione dei progressi in questo ambito risiede nella combinazione di risultati CFD ad alta accuratezza con appropriati sistemi di verifica/validazione per risolvere la fisica dello sviluppo del flusso all'interno della turbomacchina.

Tornando sulle incertezze delle metodologie CFD, che ricordo essere correlate, tra le tante, alla presenza combinata della turbolenza e della interazione statore e rotore, nel caso dei modelli di turbolenza basati sulla mediazione temporale delle quantità (RANS) è possibile ricondurlo al problema del gap spettrale (Sandberg, 2019). In questo modello l'energia del flusso viene espressa in termini di energia turbolenta senza considerare il contributo deterministico dovuto alle instabilità del flusso. Nel caso in cui la frequenza associata ai fenomeni deterministici sia inferiore a quella legata alla turbolenza, allora il modello di Reynolds risulta ancora valido (è infatti possibile individuare una differenza tra le due frequenze tale per cui non vi è interazione tra i due fenomeni). Tuttavia, nel caso in cui ci fosse una sovrapposizione tra i due fenomeni (frequenze simili) allora il gap scomparirebbe e i due fenomeni interagirebbero rendendo inutilizzabile il modello in quanto non pensato per considerare due fenomeni non costanti. Maggiori informazioni in merito si trovano nel seguente articolo (Tucker, 2011). Altre fonti di incertezza nei calcoli CFD derivano dalla presenza della curvatura e del gradiente avverso di pressione. La prima causa una redistribuzione dei flussi di taglio turbolenti responsabile sia della generazione di entalpia che della sua diffusione nel campo di moto generando delle nuove fonti di irreversibilità, mentre i gradienti di pressione, responsabili tanto della separazione del flusso quanto delle perdite, crescono sempre di più con l'aumentare dei rapporti di

compressione. Quest'ultimi causano una deformazione del flusso tale che il campo derivato eccede i limiti di validità del modello utilizzato (due equazioni).

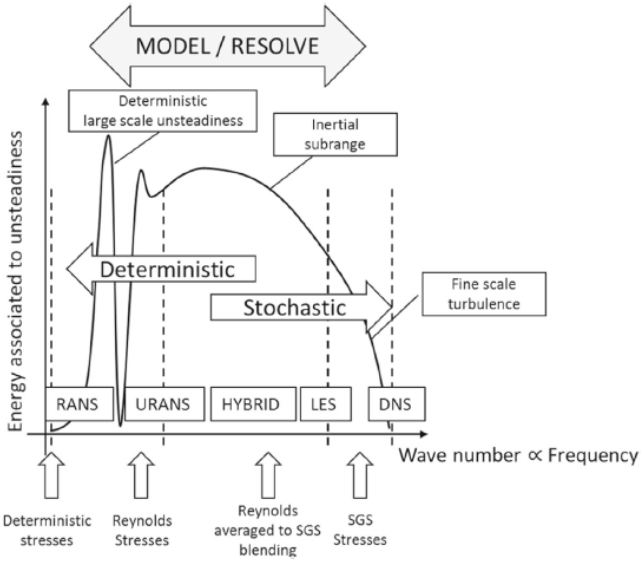


Figura 1 Modelli di turbolenza associati alla frequenza propria (Sandberg, 2019)

La figura precedente riassume quanto espresso all'inizio del capitolo in merito ai modelli di turbolenza da utilizzare in base alla scala di turbolenza. Qualora si voglia simulare l'intero campo di moto turbolento è necessario un modello/griglia molto fine che richiama l'utilizzo di una simulazione diretta. Nel caso unidimensionale, il numero di nodi richiesto per risolvere la più piccola scala turbolenta è correlato alla scala di Kolmogorov η_k ed alla scala più grande l, associata alla caratteristica del flusso. Considerando inoltre il processo di integrazione temporale, il costo computazionale risulta proporzionale a $R_{e_l}^3$. Nonostante la proposta di altri modelli che tendono a mitigare il costo computazionale senza perdere in accuratezza, il dominio tipico di una simulazione di un flusso in una turbomacchina è molto grande, di conseguenza le DNS non possono essere utilizzate per simulare l'evoluzione del flusso con una grado di accuratezza elevato (tranne nel caso di geometrie semplici). Invece di risolvere le equazioni con una griglia fitta per ogni scala di turbolenza, ci si sposta (nel grafico) nel campo inerziale dove mediante un'approssimazione delle scale turbolente più piccole (isotropiche) è possibile utilizzare per quest'ultime un modello detto Sub-Grid (SGS) mentre per le altre scale si utilizza una simulazione diretta. Nonostante la notevole riduzione del costo computazionale paragonato a quello delle DNS, le LES non sono ancora uno strumento di design utilizzabile, o meglio, non può essere utilizzato come modello per effettuare tanti passi di

iterazione nel processo di design data la dimensione e la complessità del dominio per alti valori del numero di Reynolds. Il metodo più utilizzato deriva dalla risoluzione delle RANS/URANS a cui sono associate le frequenze più basse tra cui quelle già citate tra rotore e statore. Nel processo di divisione delle quantità fluidodinamiche in componente media e fluttuante si introducono nell'equazione il tensore degli sforzi turbolenti ed il flusso termico turbolento. Per la risoluzione del modello sono necessarie delle ipotesi di chiusura.

Ad oggi una delle soluzioni più interessanti per la costruzione di modelli di turbolenza efficienti (basso costo computazionale) ed accurati, consiste nello sfruttare i dataset ricavati dalle simulazioni dirette (high fidelity) e il machine learning per migliorare le discrepanze tra RANS e DNS/LES. L'utilizzo dell'intelligenza artificiale ha portato al miglioramento delle analisi sia a priori sia a posteriori. Nel caso a priori si addestra l'AI con un database di input proveniente dalle simulazioni dirette; successivamente (a posteriori) si utilizza l'equazione precedentemente ricavata in una analisi CFD per migliorare i risultati su dei punti non studiati in fase di training. La maggior parte degli sforzi viene rivolta all'individuazione del punto ottimale di applicazione dell'AI nell'algoritmo del modello di turbolenza oltre che ad un addestramento sempre più completo per garantire generalità al modello. Tra le tecniche di machine learning più utilizzate si hanno le reti neurali, applicate (ad esempio) per la previsione dei flussi secondari (Ling, 2016), le reti neurali profonde usate per modellizzare gli stress di Reynolds (Ling J. J., 2016); Hočevar ha modellato alcune variabili della scia turbolenta (Hočevar, 2005), Duraisamy e colleghi hanno modellato il termine di produzione di Spalart-Allmaras e migliorato le relazioni tra le variabili nella modellizzazione della turbolenza (Anad Pratap Singh, 2016) (Singh, Medida, & Duraisamy, 2017). Un altro modello di machine learning molto utilizzato è quello basato sulla teoria dell'algoritmo evolutivo, detto Gene Expression Programming (GEP). Si tratta di una forma di regressione simbolica in cui l'output del programma è un'espressione analitica che meglio approssima i dati sperimentali, facilmente modificabile ed implementabile nei codici di calcolo CFD. GEP è stato implementato con successo nel caso di turbine ad alta pressione come riportato da (Weatheritt, 2017). Questo approccio innovativo di miglioramento dei modelli di chiusura della turbolenza è noto con il nome di inversione di campo e machine learning (Field Inversion and Machine Learning); tale argomento verrà trattato nel prossimo capitolo.

Bibliografia

Anad Pratap Singh, K. D. (2016). Using field inversion to quantify functional errors in turbolence closure. *Physics of Fluid 28*.

Gregory M Laskowski, J. K. (2016). Future Directions of High-Fidelity CFD for Aero-Thermal. *46th AIAA Fluid Dynamics Conference*, (p. 3322). Washington, D.C.

Hočevar, M. Š. (2005). A Turbulent-Wake Estimation Using Radial Basis Function Neural Networks. *Flow Turbulence Combust* , 291-308.

Ling, J. J. (2016). Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, 22-35.

Ling, J. K. (2016). Reynolds averaged turbulence modelling using deep neural. *J. Fluid Mech*, 155-166.

Sandberg, R. M. (2019). The Current State of High-Fidelity Simulations for Main Gas Path Turbomachinery Components and Their Industrial Impact. *Flow Turbulence Combust*, 797–848.

Singh, A. P., Medida, S., & Duraisamy, K. (2017). Machine-Learning-Augmented Predictive Modeling of Turbulent Separated Flows over Airfoils. *AIAA Journal*, 2215-2227.

Tucker, P. (2011). Computation of unsteady turbomachinery flows: Part 1–progress and challenges. *Progress in Aerospace Science*, 522-545.

Weatheritt, J. P. (2017). Machine Learning for Turbulence Model Development Using a High-Fidelity HPT Cascade Simulation. ASME Turbo Expo 2017: Turbomachinery Technical Conference and Exposition.

CAPITOLO 3

3.1 Inversione di campo

Esistono tre approcci per fare previsioni mediante dati ricavati da DNS/LES (Holland, Baeder, & Duraisamy, 2019). Il primo, banale, consiste nell'utilizzare direttamente questi dati per fare previsioni. Il problema principale di questo metodo è che non solo richiede una grande quantità di dati, ma anche la qualità degli stessi deve essere elevata. Quindi, in generale, questo approccio non viene utilizzato per un modello RANS in quanto la turbolenza è un fenomeno molto complesso e non ci sono abbastanza dati per poter effettuare delle previsioni per tutti i casi. Il secondo approccio consiste nell'inserire i dati ricavati dalle simulazioni high-fidelity direttamente nel modello. Nonostante la sua semplicità, questo approccio presenta un problema di fondo riguardante la natura dei dati LES/DNS. Questo tipo di dati contengono quantità fisiche (reali), al contrario di quelle presenti nei modelli RANS che sono, per l'appunto, modellizzate secondo un criterio scelto dal progettista basato su delle evidenze empirico-sperimentali. Vi è quindi una mancanza di consistenza tra dati importati nel modello e dati "modellati" che porta ad un degrado delle prestazioni nella fase predittiva (Hassan, Ugo, & Andrew, 2011). L'ultimo approccio è quello più appropriato per modelli RANS o in generale per modelli basati sulla fisica. Il paradigma dell'inversione di campo è stato introdotto per la prima volta da Duraisamy et al (K.Duraisamy, Zhang, & Singh, 2015) e Tracey et al. (Tracey, Duraisamy, & Alonso., 2015) per poi essere applicato in una versione più generale da Singh et al (Singh, Matai, Mishra, Duraisamy, & Durbin., 2017) ed essere ulteriormente sviluppato in altri lavori (Parish & K.Duraisamy, 2016) (Singh & Duraisamy, Using field inversion to quantify functional errors in turbulence closures, 2016). Questo approccio, noto con il nome di field inversion and machine learning (FIML), combina l'inferenza statistica con gli algoritmi di machine learning per produrre un modello di turbolenza con caratteristiche predittive migliori di quello di partenza. In figura 2 viene presentato lo schema del modello. Alla base dell'approccio si ha la scelta di un modello di chiusura della turbolenza alla quale si applica una funzione correttiva β in un punto del modello significativo. Questa funzione correttiva viene calcolata attraverso un problema di ottimizzazione multidimensionale al fine di eguagliare i dati ricavati dalle simulazioni (Korlaar, 2019). Poiché la perfetta uguaglianza è irrealizzabile si stabilisce una funzione obiettivo (goal) che racchiude le discrepanze tra i due modelli. A questo punto si prosegue

con la ricerca del valore di beta tale per cui si raggiunge un valore ottimale della funzione obiettivo. Questo passaggio viene risolto impostando il problema di inversione sulla discretizzazione realizzata sul campo di moto; ovvero viene cercato il valore del termine correttivo in ciascuna cella della griglia ed attraverso un algoritmo di ottimizzazione viene calcolato un nuovo valore di beta tale per cui il valore assunto dalla funzione obiettivo sia minimizzato. La seconda parte del paradigma, quella dedicata al machine learning, ha l'obiettivo di dedurre il valore di beta su altri tipi di flusso. Questa fase può essere divisa in due parti:

1) Fase di training

   Vengono inseriti nel modello di machine learning le caratteristiche del flusso con i valori della funzione ottimizzata ottenuta dal processo di inversione al fine di ottenere una funzione di correlazione tra dati di input e output.

2) Fase di previsione

   L'algoritmo di machine learning (appena allenato) viene utilizzato per prevedere il valore del parametro correttivo su altri casi di studio. Questo processo può essere realizzato per via iterativa nel momento in cui viene inserito nel codice CFD una versione del modello di chiusura della turbolenza che presenti la funzione correttiva. In questo modo, ad ogni passo di iterazione, il valore di beta viene aggiornato in base ai valori assunti dalle quantità fluidodinamiche.

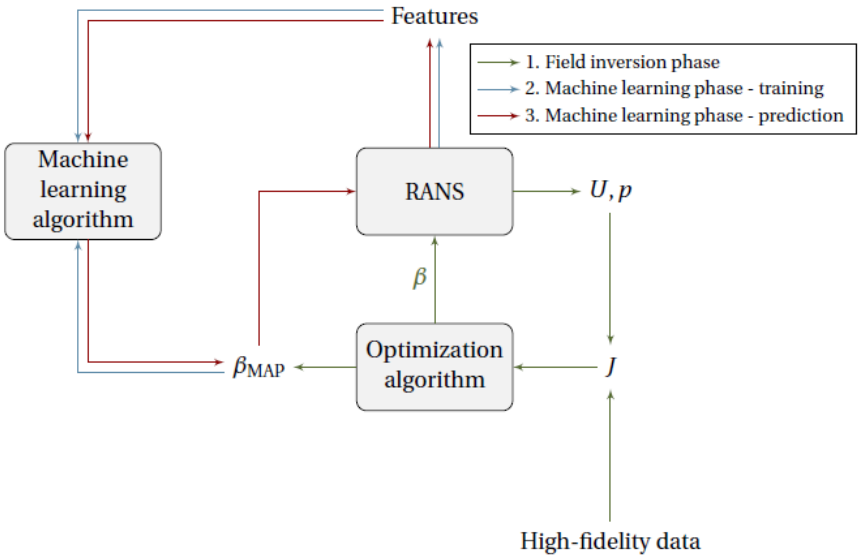


Figura 2 Diagramma di flusso per FIML (Holland, Baeder, & Duraisamy, 2019)

3.1.1 Procedura

L'approccio utilizzato in questa tesi segue il lavoro svolto da (Singh, Medida, & Duraisamy, 2017) che hanno utilizzato come modello di base quello di Spalart-Allmaras (Spalart & Allmaras, 1992). Inoltre i dati di base e procedimento per la raccolta dei valori di beta e della funzione di goal provengono dalla tesi di Davide Balbo (Balbo, Larocca, & Ferrero, 2019), da cui questa tesi continua la ricerca. L'equazione di fondo è

$$\frac{D\tilde{\nu}}{Dt} = P(\tilde{\nu}, U) - D(\tilde{\nu}, U) + T(V, U) \tag{3.1}$$

Dove

- U rappresenta le variabili fluidodinamiche mediate di Reynolds
- $P(\tilde{\nu}, U)$ rappresenta il termine di produzione
- $D(\tilde{\nu}, U)$ rappresenta il termine di distruzione
- $T(V, U)$ è il termine di trasporto diffusivo

Per il calcolo degli sforzi di Reynolds si utilizza l'assunzione di Boussinesque per la viscosità turbolenta

$$\nu_t = \tilde{\nu} f_{\nu 1} \tag{3.2}$$

Dove ν è la viscosità cinematica e $f_{\nu 1}$ è la funzione smorzamento

$$f_{v1} = \frac{\chi^3}{\chi^3 + C_{vi}^3} \quad \chi = \frac{\tilde{\nu}}{\nu} \tag{3.3}$$

La variabile $\tilde{\nu}$ obbedisce all'equazione di trasporto

$$\frac{D\tilde{\nu}}{Dt} = P - D + \frac{1}{\sigma}\left[\nabla * \left((\tilde{\nu} + \nu)\nabla\tilde{\nu}\right) + c_{b2}\left(\widetilde{\nabla\nu}\right)^2\right] \tag{3.4}$$

La formula precedente, che ha validità generale, può essere riscritta in termini di flusso comprimibile combinandola con l'equazione di conservazione della massa, ottenendo

$$\frac{\partial(\rho\tilde{\nu})}{\partial t} + \frac{\partial(\rho\tilde{\nu}v_j)}{\partial x_j} = \rho(P - D) + \frac{1}{\sigma} * \frac{\partial\left[\rho(\tilde{\nu} + \nu) * \frac{\partial\tilde{\nu}}{\partial x_j}\right]}{\partial x_j} + \frac{c_{b2}}{\sigma} * \rho * \left(\frac{\partial\tilde{\nu}}{\partial x_k}\right)^2 - \frac{1}{\sigma} * \frac{\partial\rho}{\partial x_k} * \frac{\partial\tilde{\nu}}{\partial x_k} \tag{3.5}$$

Nella quale:

$$P = c_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu} \tag{3.6}$$

$$D = \frac{\tilde{\nu}^2}{d} * \left(c_{w1}f_w - \frac{c_{b1}}{k^2}f_{t2}\right) \tag{3.7}$$

$$\tilde{S} = S + \bar{S}\ se\ \bar{S} \geq -c_{v2}S\ altrimenti\ \tilde{S} = S + \frac{S(S * c_{v2}^2 + c_{v3}\bar{S})}{S - S(c_{v3} - 2c_{v2})} \tag{3.8}$$

$$\bar{S} = \frac{\tilde{\nu}}{k^2 d^2} f_{v2} \tag{3.9}$$

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \tag{3.10}$$

$$f_w = g\left(\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right)^{\frac{1}{6}} \tag{3.11}$$

$$g = r + c_{w2}(r^6 - r) \tag{3.12}$$

$$r = \min\left(\frac{\tilde{\nu}}{k^2 d^2 \tilde{S}}, r_{limite}\right) \tag{3.13}$$

$$f_{t1} = c_{t1}g_t e^{-c_{t2} * \frac{\omega_t}{\Delta u^2} * \left(d^2 + g_t^2 d_t^2\right)} \tag{3.14}$$

$$f_{t2} = c_{t3}e^{-c_{t4}\chi^2} \tag{3.15}$$

$$\mu_t = \rho\tilde{\nu}f_{v1} \tag{3.16}$$

Le costanti utilizzate nel modello valgono: $c_{b1} = 0.1355$, $\sigma = \frac{2}{3}$, $c_{b2} = 0.622$, $k = 0.41$, $c_{w1} = \frac{c_{b1}}{k^2} = 0.806$, $c_{w2} = 0.3$, $c_{w3} = 2$, $c_{v1} = 7.1$, $c_{t1} = 1$, $c_{t2} = 2$, $c_{t3} = 1.3$, $c_{t4} = 0.5$, $r_{limite} = 10$, $c_{v2} = 0.7$, $c_{v3} = 0.9$

Il test case scelto per l'applicazione del modello è un condotto bidimensionale con curvatura di circa 90° a metà della sua lunghezza, composto da due pareti solide parallele che collegano ingresso ed uscita. La parete superiore viene considerata inviscida mentre la parete inferiore si considera adiabatica e soggetta alla viscosità. Il flusso quindi presenterà un profilo di velocità che sarà nullo a parete, aumentando in valore man mano che ci si allontana dalla stessa, generando lo strato limite.

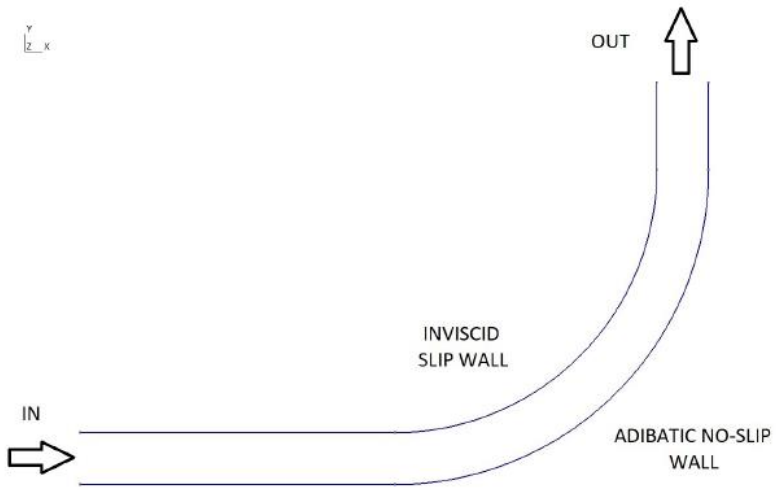


Figura 3 Test case (Balbo, Larocca, & Ferrero, 2019)

In seguito si utilizza il processo di inversione di campo andando ad agire sul termine di produzione nell'equazione di SA poiché esso influenza la viscosità turbolenta e conseguentemente tutto il campo di moto:

$$\frac{\partial(\rho\tilde{\nu})}{\partial t} + \frac{\partial(\rho\tilde{\nu}v_j)}{\partial x_j} = \rho(\beta P - D) + \frac{\partial\left[\rho(\tilde{\nu} + \nu) * \frac{\partial\tilde{\nu}}{\partial x_j}\right]}{\sigma\partial x_j} + \frac{c_{b2}}{\sigma} * \rho * \left(\frac{\partial\tilde{\nu}}{\partial x_k}\right)^2 - \frac{1}{\sigma} * \frac{\partial\rho}{\partial x_k} * \frac{\partial\tilde{\nu}}{\partial x_k} \quad (3.17)$$

Per verificare la bontà del modello si sceglie come parametro di controllo il coefficiente d'attrito a parete, il cui valore è influenzato dallo strato limite che si forma sulla parete e dalla relativa turbolenza, dunque coerente con l'approccio di inversione scelto precedentemente. Un'altra ragione che giustifica la scelta fatta risiede nella disposizione dei dati relativi a tale parametro ottenuti mediante una simulazione LES sullo stesso test

case. La differenza tra il valore del c_f ottenuto mediante SA e il valore reale (sperimentale), permette di definire la funzione obiettivo

$$g = \int \left(c_f - c_{f,reale}\right)^2 dl \tag{3.18}$$

Dove l rappresenta la lunghezza della parete inferiore. A questo punto è possibile calcolare il valore del gradiente della funzione goal rispetto al parametro beta che verrà utilizzato nel processo di ottimizzazione. È necessario tuttavia ragionare sul termine $\frac{\partial g}{\partial \beta}$, in quanto, data la multidimensionalità, è stato necessario utilizzare il metodo dell'aggiunto per ridurre il costo computazionale. Il gradiente risulta

$$\frac{dg}{d\beta} = \frac{\partial g}{\partial \beta} + \psi * \frac{\partial R}{\partial \beta} \tag{3.19}$$

L'equazione derivante dall'aggiunto

$$\left[\frac{\partial R}{\partial U}\right]^T \psi = -\left[\frac{\partial g}{\partial U}\right]^T \tag{3.20}$$

Dove ψ è la variabile derivante dal sistema dell'aggiunto e R rappresenta i residui del set di equazioni di governo. I termini presenti nella equazione 3.20 sono vettori e matrici di dimensione proporzionale al numero di celle della mesh moltiplicato per il numero di gradi di libertà di cella. Avendo definito tutti i termini necessari si procede con il processo di ottimizzazione. Si impone un valore di beta iniziale, uno nel lavoro in esame, che viene aggiornato iterativamente ad ogni passo temporale con il gradiente della funzione obiettivo

$$\beta_{k+1} = \beta_k - \gamma * \frac{dg}{d\beta} \tag{3.21}$$

γ rappresenta l'entità del passo che viene scelto arbitrariamente fino al raggiungimento di un valore ottimale per la simulazione. In sintesi la procedura consiste nel risolvere il sistema di equazioni fino al raggiungimento di una soluzione stabile, dalla quale si estraggono le informazioni necessarie per calcolare la funzione obiettivo ed il suo gradiente che andranno ad aggiornare il campo di beta. A causa della variazione di questo parametro, il termine di produzione di SA verrà modificato portando ad una perturbazione

nel campo di moto che richiederà un'ulteriore passo di calcolo per la convergenza. Il processo così descritto prosegue iterativamente fino al raggiungimento di risultati considerati accettabili, dove nel nostro caso è da intendersi come una distribuzione del c_f a parete simile a quello ricavabile dalla simulazione LES.

Bibliografia

Hassan, R., Ugo, P., & Andrew, P. (2011). Evaluation of turbulence models using direct numerical and large-eddy simulation data. *Journal of Fluids Engineering*, 133(2):021203.

K.Duraisamy, Zhang, Z. J., & Singh, A. P. (2015). New approaches in turbulence and transition modeling using data driven techniques. *53rd AIAA Aerospace Sciences Meeting*, 1284.

Parish, E. J., & K.Duraisamy. (2016). A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics, 305*, 758-774.

Singh, A. P., & Duraisamy, K. (2016). Using field inversion to quantify functional errors in turbulence closures. *Physics of Fluids 28*, 045110.

Singh, A. P., Matai, R., Mishra, A., Duraisamy, K., & Durbin., P. A. (2017). Data-driven augmentation of turbulence models for adverse pressure gradient flows. *23rd AIAA Computational Fluid Dynamics Conference*, 3626.

Singh, A. P., Medida, S., & Duraisamy, K. (2017). Machine-Learning-Augmented Predictive Modeling of Turbulent Separated Flows over Airfoils. *AIAA Journal*, 2215-2227.

Spalart, P., & Allmaras, S. (1992, Gennaio 6). A one equation turbolence model for aerodynamic flows., (p. 5-21). Reno,Nevada (USA).

Tracey, B. D., Duraisamy, K., & Alonso., J. J. (2015). A machine learning strategy to assist turbulence model development. *53rd AIAA Aerospace SciencesMeeting*, 1287.

Holland, J., Baeder, J., & Duraisamy, K. (2019). Tesi di laurea magistrale. Integrated field inversion and machine learning with embedded neural network training for turbolence modeling. Maryland, USA.

Korlaar, A. (2019, Febbraio 22). Tesi di laurea magistrale. Field inversion and machine learning in turbolence modeling. Delft, Paesi Bassi.

Balbo, D., Larocca, F., & Ferrero, A. (2019). Tesi di laurea magistrale. Miglioramento di modelli di turbolenza mediante inversione del campo. Torino.

CAPITOLO 4

4.1 Reti neurali

Le reti neurali artificiali rappresentano la trasposizione informatica delle reti neurali biologiche del cervello umano. Quest'ultimo è costituito da circa 10^{12} cellule celebrali deputate all'elaborazione dell'informazione (neuroni) che viene trasferita ad un corpo centrale, detto assone, la cui funzione è quella di propagare il segnale. Ogni neurone si può trovare o eccitato o inibito. Attivandosi, il neurone produce un impulso elettrico che viene trasportato dall'assone. Una volta che il segnale raggiunge la sinapsi avviene il rilascio di sostanze chimiche che entrano nel corpo di altri neuroni. In base al tipo di sinapsi, che possono essere inibitorie o eccitatorie, queste sostanze aumentano o diminuiscono la probabilità che il successivo neurone si attivi. Ad ogni sinapsi è associato un peso che ne determina il tipo e l'ampiezza dell'effetto corrispondente. Quindi, nella pratica, ogni neurone effettua una somma pesata degli ingressi provenienti da altri neuroni e se tale valore supera un valore soglia il neurone si attiva.

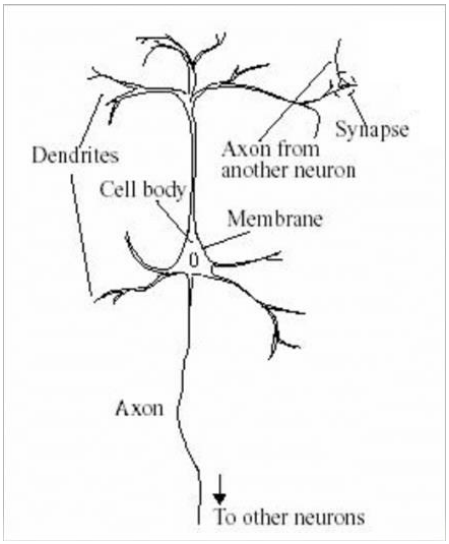


Figura 4 Schema neurone biologico (Fabio, 2005/06)

Una possibile definizione di rete neurale è quella fornita in (Haykin, 2009):

"Una rete neurale è un processore con distribuzione dell'attività in parallelo costituito da singole unità processuali che ha una naturale predisposizione ad immagazzinare conoscenza dovuta all'esperienza ed a renderla disponibile all'uso. Possiede due punti di similitudine con il cervello umano rappresentati dalla ricezione di informazioni utili alla conoscenza (processo di apprendimento/training) e dal loro immagazzinamento mediante

il peso sinaptico". La fase di apprendimento, nota con il nome di training process, ha l'obiettivo di modificare i pesi sinaptici della rete in maniera iterativa fino al raggiungimento di una distribuzione ottimale. La modifica dei pesi sinaptici costituisce il metodo tradizionale del progetto di una ANN (artificial neural network). L'utilizzo di una rete neurale offre le seguenti proprietà (Haykin, 2009):

- Non linearità

  La connessione di più neuroni non lineari costituisce una rete non lineare, il cui utilizzo è necessario nel caso di meccanismi fisici responsabili della generazione di segnali non lineari

- Correlazione input-output

  Dato un dataset di ingresso ed una risposta desiderata la rete neurale nella fase di training modifica i pesi sinaptici per minimizzare la differenza tra valore target e la risposta attuale. Questo processo viene ripetuto iterativamente per ogni elemento del dataset fino al raggiungimento dello steady state in cui non avvengono più significanti cambiamenti dei pesi.

- Adattamento

  Rappresenta la capacità intrinseca della rete neurale di modificare i pesi per conformarsi alle variazioni dell'ambiente di lavoro.

- Informazione contestuale

  Rappresenta la caratteristica della rete di essere composta da neuroni che sono intercomunicanti nell'attività globale.

- Flessibilità

  Rappresenta la capacità della rete di apprendere autonomamente le proprietà del campo in cui è applicata; ovvero il designer della rete non ha la necessità di conoscere le soluzioni analitiche a priori.

- Generalizzazione

  Una rete neurale addestrata su un dataset è in grado di produrre una risposta adeguata ad un dataset diverso da quello di training, ma che presenta qualche somiglianza con quello di addestramento. Questo è realizzabile perché la rete

tende ad estrarre le caratteristiche invarianti del modello piuttosto che memorizzarne uno specifico. La capacità di generalizzare è una caratteristica molto importante nei campi di applicazione dove è impossibile ottenere una collezione esaustiva di tutti i dati su cui la rete dovrà operare.

4.2 Neurone di McCulloch-Pitts

Abbiamo già definito il neurone come l'unità di base della rete neurale. Il più semplice modello matematico che rappresenti il funzionamento di tale neurone è quello proposto da McCulloch e Pitts (McCulloch & Pitts, 1990). Esso è composto da tre elementi base: un insieme di sinapsi caratterizzate dal relativo peso (efficacia sinaptica), un sommatore, la cui funzione è quella di fare una somma pesata dei dati di input e produrre in output una combinazione lineare degli input ed infine una funzione di attivazione per limitare il range dell'output.

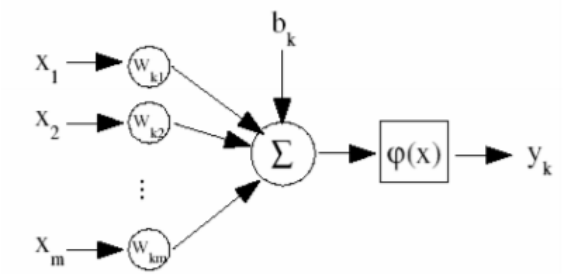


Figura 5 Neurone elementare (Samuel)

Tale neurone può essere visto come una funzione non lineare che trasforma le variabili in ingresso $x_1 \ldots . x_m$ nelle variabili di uscita y_k. Come già accennato, viene effettuata la somma ponderata degli ingressi usando come pesi i valori $w_{k1} \ldots . w_{km}$ ottenendo:

$$V_k = \sum_{i=1}^{m} w_{ki} x_i + b_k \qquad (4.1)$$

dove V_k è la combinazione lineare, mentre b_k è il termine di bias o threshold e corrisponde alla soglia di attivazione del neurone biologico. Andando a definire un ulteriore ingresso x_0 (posto costante ed uguale ad 1), si può riscrivere la funzione precedente nel seguente modo

$$V_k = \sum_{i=0}^{m} w_{ki} x_i \tag{4.2}$$

La funzione V_k assume ora il nome di potenziale di attivazione. Infine l'output viene ottenuto applicando a V_k una trasformazione non lineare $\varphi(V_k)$ detta funzione di attivazione.

$$y_k = \varphi\left(\sum_{i=0}^{m} w_{ki} x_i\right) \tag{4.3}$$

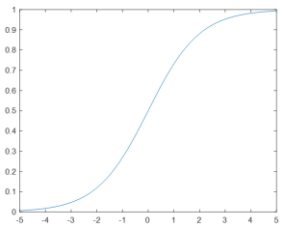
4.3 Funzione di attivazione

Le funzioni di attivazione sono funzioni aventi le proprietà di continuità e derivabilità all'interno del dominio di definizione e sono, inoltre, monotone non decrescenti. Alcune di queste funzioni sono particolarmente utilizzate poiché la loro derivata può essere espressa mediante i valori che la funzione stessa assume all'interno del dominio. Nelle reti neurali comuni si utilizzano spesso le funzioni sigmoidali in quanto godono della seguente proprietà

$$\frac{d}{dt} sig(x) = sig(x)\big(1 - sig(x)\big) \tag{4.4}$$

Dove sig(x) indica la generica sigmoide[1]. A volte si preferisce avere una funzione di attivazione nell'intervallo [-1,1], nel qual caso si utilizza la tangente iperbolica

[1] La generica sigmoide è una funzione del tipo $sig(x) = \frac{1}{1+e^{-x}}$ ed è un caso particolare della più generica funzione logistica.



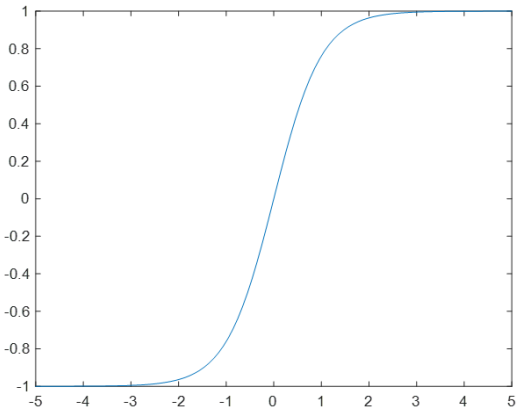


Figura 6 y=tanh(x)

Altri tipi di funzione di attivazione sono:

- Funzione di Heaviside

$$f(a) = \begin{cases} 1 & se\ a \geq 0 \\ 0 & se\ a < 0 \end{cases}$$

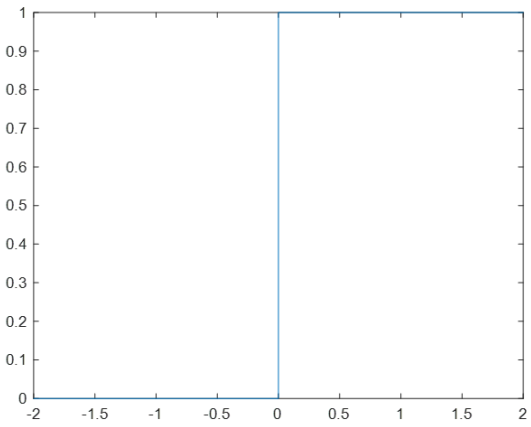


Figura 7 Funzione a gradino

È quella utilizzata nel modello di McCulloch e Pitts.

- Funzione lineare a tratti

$$f(a) = \begin{cases} 1 & se\ a \geq \frac{1}{2} \\ a & se -\frac{1}{2} < a < \frac{1}{2} \\ 0 & a \leq -\frac{1}{2} \end{cases}$$

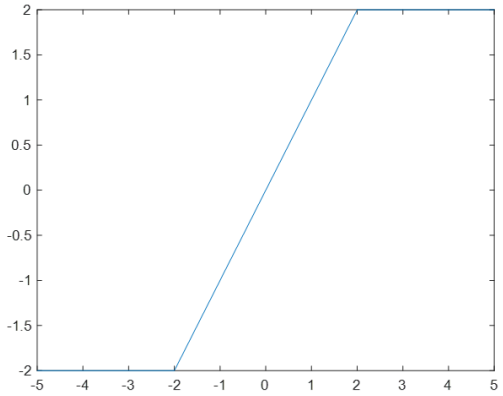


Figura 8 Funzione lineare a tratti

- Funzione identità

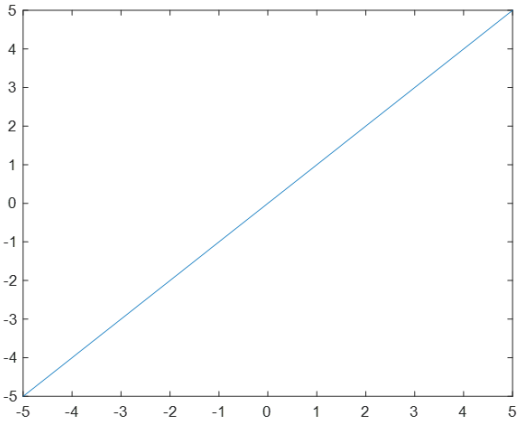


Figura 9 Funzione identità

4.4 Architettura di rete

Con il termine architettura di rete si indica lo schema di connessione dei neuroni. Esistono principalmente tre modelli di rete:

1. Reti feedforward ad uno strato

   In questa architettura si hanno i nodi di input (input layer) e uno strato di neuroni (output layer). Il segnale si propaga in maniera unidirezionale a partire dall'input layer e terminando in quello di output. All'inizio del capitolo si è parlato del neurone elementare. Se si immagina un insieme m di tali unità aventi ingressi comuni, si ottiene una rete neurale a singolo strato. L'output è dato da:

$$y_k = \varphi\left(\sum_{i=0}^{d} w_{ki} x_i\right), \quad k = 1, \dots, m \tag{4.5}$$

   Dove w_{ki} rappresenta il peso che connette l'ingresso i con l'uscita j; φ è la funzione di attivazione. Reti a singolo strato di questo tipo sono note con il termine perceptron ed hanno una capacità computazionale molto limitata. La funzione di attivazione adottata da questo modello è quella a gradino.

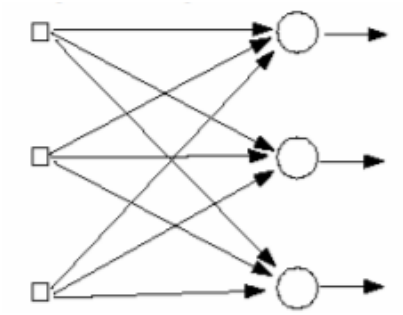


Figura 10 Rete feedforward ad uno strato (Samuel)

2. Reti feedforward a più strati

   Si distinguono dalle precedenti per il fatto che, tra lo strato di input e output, si hanno uno o più strati di neuroni nascosti, detti hidden layer. Ogni strato ha connessioni entranti dal precedente ed uscenti in quello successivo, quindi la propagazione del segnale avviene in avanti senza cicli e senza connessioni trasversali. Esse possono risolvere problemi più complessi delle reti a singolo

strato, ma l'addestramento risulta più complesso. Riprendendo il ragionamento fatto per le reti a single layer, l'equazione 4.5 è da interpretare come l'output dell'hidden layer, il quale, mediante una funzione di attivazione g (non necessariamente uguale a φ, produce il risultato della rete:

$$y_{k_{hl}} = g\left(\sum_{k=0}^{m} \widetilde{w}_{nk} y_k\right), \quad k = 1, \dots, c \tag{4.6}$$

Dove $\widetilde{w}_{nk}$ rappresenta il peso dell'unità nascosta j connessa all'uscita k.Infine

$$y_k = g\left(\sum_{k=0}^{m} \widetilde{w}_{nk}\, \varphi\left(\sum_{i=0}^{d} w_{ki} x_i\right)\right), \quad k = 1, \dots, c \tag{4.7}$$

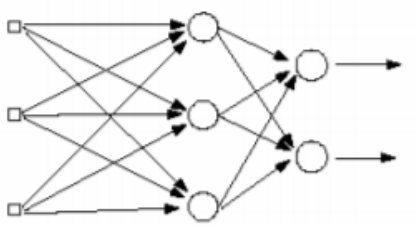


Figura 11 Rete feedforward a 2 strati (Samuel)

3. Reti ricorrenti

A differenza delle architetture precedentemente esposte, questo modello è dotato di un loop di feedback tale per cui la rete assume una connotazione dinamica che ne aumenta le connessioni neurali.

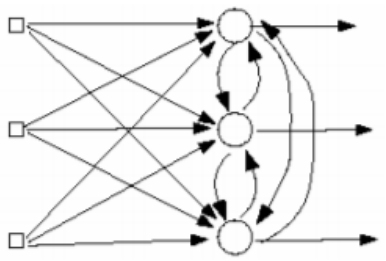


Figura 12 Recurrent Network (Samuel)

Si specifica che nella figura 12 è stato riportato un modello semplificato; infatti non vi è la presenza di hidden layers e il loop è stato implementato solo sullo strato di output.

4.5 Addestramento reti

In questa trattazione si parlerà dell'addestramento di reti multistrato, in quanto saranno quelle utilizzate per la tesi.

Con il termine addestramento/apprendimento di una rete neurale si indica il processo di adattamento dei pesi della rete stessa in modo incrementale al fine di ottenere determinate performance stabilite a priori (Hassoun, 1995). L'algoritmo di addestramento è noto con il termine di retro-propagazione degli errori ed è un esempio di gradiente decrescente che minimizza l'errore quadratico tra l'output di rete ed il target desiderato. Tale algoritmo è costituito da tre fasi: propagazione in avanti degli input, calcolo dell'errore e retro-propagazione dello stesso ed infine la regolazione dei pesi.

4.5.1 Algoritmo di retro-propagazione standard

Come già citato precedentemente esso consiste di tre fasi; nella prima ogni unità di ingresso riceve un segnale di input e lo trasmette ad ognuna delle unità nascoste le quali, a loro volta, si attivano e inviano il segnale ad ognuna delle unità di output. Quest'ultime effettuano la propria attivazione per fornire la risposta della rete per il dato schema di ingresso, confrontando il segnale ricevuto con quello di target per determinare l'errore (associato a quello schema e riferito a quella unità). Successivamente questo errore viene distribuito all'indietro nell' architettura della rete (questa operazione viene effettuata da ogni unità all'unità del layer precedente). Infine l'errore viene usato per aggiornare i pesi tra l'output e lo strato nascosto e tra lo strato nascosto e l'ingresso. Una volta determinati tutti i pesi dei segnali si aggiustano simultaneamente i pesi di tutti gli strati. Si intuisce quindi come tale apprendimento possa essere ricondotto ad un problema di ottimizzazione, ovvero la ricerca di una soluzione all'interno di uno spazio

multidimensionale di parametri (pesi e bias) che ottimizza una funzione predefinita. Si introduce ora la procedura matematica facendo una distinzione tra neurone nell'hidden layer e nell'output layer. La trattazione segue le linee guida presenti in (Samuel).

4.5.1.1 Neurone j di output

Il segnale d'errore del neurone di output j all'iterazione n è definito da:

$$e_j(n) = d_j(n) - y_j(n) \tag{4.8}$$

Dove d_j è il valore target del neurone j mentre y_j è l'output del neurone j. L'errore totale dell'output layer presentando l'n-esimo esempio del training set è definito come

$$E(n) = \frac{1}{2}\sum_{j=1}^{NN} e_j(n)^2 \tag{4.9}$$

Dove NN rappresenta il numero di neuroni nello strato di output. Dato un numero k di patterns nel training set, l'errore quadratico medio rappresenta la funzione di costo ed è data da

$$E_Q(n) = \frac{1}{k}\sum_{i=1}^{k} E(n) \tag{4.10}$$

La quale deve essere minimizzata utilizzando, per esempio, il metodo della discesa del gradiente dove quest'ultimo vale $\frac{\partial E(n)}{\partial w_{ji}(n)}$ e gli aggiornamenti dei pesi avvengono in verso opposto (minimizzazione) al gradiente

$$w_{ji}^{n+1} = w_{ji}^{n} + \Delta w_{ji}^{n} \tag{4.11}$$

$$\Delta w_{ji}^{n} = -\eta * \frac{\partial E(n)}{\partial w_{ji}(n)} \tag{4.12}$$

Dove η è il fattore di apprendimento.

Il gradiente, calcolato mediante la regola della catena, è dato da:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}\frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{4.13}$$

Nella quale v_j rappresenta il potenziale di attivazione dato dall'equazione 4.2. Andando ad esprimere le singole derivate

$$\frac{\partial E(n)}{\partial e_j(n)} = \frac{1}{2} * \sum_{k=1}^{NN} \frac{\partial e_k^2(n)}{\partial e_j(n)} = e_j(n) \tag{4.14}$$

Poiché si considera un neurone nello strato di output (k=j)

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial (d_j(n) - y_j(n))}{\partial y_j(n)} = -1 \tag{4.15}$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \varphi(v_j(n))}{\partial v_j(n)} = \dot{\varphi}\left(v_j(n)\right) \tag{4.16}$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \sum_{k=0}^{m} y_k(n) * \frac{\partial w_{jk}(n)}{\partial w_{ji}(n)} = y_i(n) \tag{4.17}$$

Da cui si ricava

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -\delta_j(n) y_i(n) \tag{4.18}$$

Dove δ_j è il gradiente locale

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) * \dot{\varphi}\left(v_j(n)\right) \tag{4.19}$$

Infine la variazione di peso sinaptico tra il neurone i e j relativo all'n-esimo esempio del training set è

$$\Delta w_{ji}(n) = \eta * \delta_j(n) * y_i(n) \tag{4.20}$$

4.5.1.2 Neurone j nascosto

In questo caso ci si trova nel mezzo dell'architettura, dunque il neurone j-esimo non dispone della risposta desiderata, quindi il segnale di errore deve essere determinato ricorsivamente dal segnale di errore di tutti i neuroni ai quali il neurone in esame è connesso. La seguente trattazione considera il nodo appartenente all'ultimo strato nascosto. Si riprende l'equazione 4.19 e si calcola la derivata $\frac{\partial E(n)}{\partial y_j(n)}$

$$\begin{aligned}\frac{\partial E(n)}{\partial y_j(n)} &= \frac{1}{2} * \sum_{k \in C} \frac{\partial e_k^2(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \\ &= -\sum_{k \in C} e_k(n) \dot{\varphi}\left(v_j(n)\right) w_{kj}(n)\end{aligned} \tag{4.21}$$

Dove C simboleggia l'insieme dei neuroni che formano l'output layer ed in questo caso j non appartiene a questo insieme. Si ottiene dunque

$$\delta_k(n) = e_k(n) * \dot{\varphi}\big(v_k(n)\big) \tag{4.22}$$

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k \in C} \delta_k(n) w_{kj}(n) \tag{4.23}$$

Questa formula, che si ricorda si è ricavata assumendo di essere sull'ultimo hidden layer, ha validità anche nel caso in cui j fosse collegato ad un altro hidden layer. In tal caso si deve considerare la sommatoria estesa a tutti i neuroni e, poiché $w_{kj} = 0$ se j non è adiacente a k, si può utilizzare la seguente formula generale

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k} \delta_k(n) w_{kj}(n) \tag{4.24}$$

Si conclude dicendo che il gradiente locale del neurone nascosto j-esimo relativo all'n-esimo esempio del training set è dato dalla relazione

$$\delta_j(n) = \dot{\varphi}\left(v_j(n)\right) \sum_{k} \delta_k(n) w_{kj}(n) \tag{4.25}$$

4.5.2 Metodo di Gauss-Newton

Data la natura del problema dell'adattamento di una rete che è riconducibile ad un problema ai minimi quadrati (si faccia riferimento all'equazione 4.9) o più in generale ad un problema di minimizzazione dell'errore, per la sua risoluzione è possibile adottare ,tra i tanti, il metodo di Gauss-Newton. Definendo il vettore dei residui $e(w) = (e_1, \dots, e_P)$ dove P è in numero di pattern nel training set, il problema dell'addestramento assume la seguente connotazione (Grippo & Sciandrone, 2002)

$$\min_w E = \frac{1}{2}\left||e(w)|\right|^2 \tag{4.26}$$

Le derivate di E si esprimono in funzione della matrice Jacobiana $J = \begin{pmatrix} \nabla e_1 \\ \dots \\ \nabla e_P \end{pmatrix}$. In particolare, si può scrivere il gradiente del vettore errore come

$$\nabla E = \sum_{p=1}^{P} e_p(w)\nabla e_p(w) = J(w)^T e(w) \tag{4.27}$$

La matrice Hessiana assume la forma

$$\nabla^2 E(w) = \sum_{p=1}^{P} \nabla e_p(w) e_p(w)^T + \sum_{p=1}^{P} e_p(w)\nabla^2 e_p(w) = J(w)^T J(w) + Q(w) \tag{4.28}$$

Ovvero è data dalla somma di due termini. Generalmente nei metodi ai minimi quadrati si assume che il termine contenente lo Jacobiano sia quello dominante all'interno dell'equazione e risulta giustificata in quei problemi con residui piccoli nell'intorno della soluzione.

Il metodo di Gauss-Newton si basa sulla regola di aggiornamento $w^{k+1} = w^k + d^k$ dove d è la soluzione del sistema

$$(J^k)^T J^k d = -(J^k)^T e^k \tag{4.29}$$

Questo metodo può essere facilmente ricondotto a quello generale di Newton, nel quale la direzione di ricerca si trova risolvendo il sistema

$$\nabla^2 E(w^k) d = -\nabla E(w^k) \tag{4.30}$$

Aggiungendo all'equazione 4.30 l'ipotesi sulla dominanza del termine Jacobiano nell'intorno della soluzione, si ottiene l'equazione 4.29. I vantaggi di questo metodo sono:

- L'approssimazione $\nabla^2 E(w^k) \cong (J^k)^T J^k$ permette di evitare il calcolo delle singole matrici Hessiane per ciascun pattern nel dataset di training
- L'efficienza del metodo è paragonabile a quella di Newton pur non considerando il termine $\sum_{p=1}^{P} e_p(w) \nabla^2 e_p(w)$

Per evitare il problema di convergenza in un minimo locale, questo metodo viene implementato nella seguente forma

$$w^{k+1} = w^k - \eta^k ((J^k)^T J^k + D^k)^{-1} (J^k)^T e^k \tag{4.31}$$

Dove η^k è il learning rate del metodo e D è una matrice diagonale tale che la matrice $(J^k)^T J^k + D^k$ sia uniformemente definita e positiva[2].

Se si sceglie la matrice D in modo tale che risulti un multiplo positivo della matrice identità, allora si utilizza un'alternativa al metodo di Gauss-Newton, noto con il nome di Levenberg-Marquardt, la cui k-esima iterazione è definita da

$$w^{k+1} = w^k - \eta^k ((J^k)^T J^k + \mu^k I)^{-1} (J^k)^T e^k \tag{4.32}$$

Dove μ è uno scalare sempre positivo chiamato coefficiente di smorzamento ed il suo valore influenza la convergenza del metodo; infatti nel caso in cui l'errore cali rapidamente, il coefficiente di smorzamento avrà un valore piccolo per trovare il valore di minimo, mentre nel caso in cui la decrescita dell'errore nel processo di iterazione non sia sufficientemente elevata, tale coefficiente assumerà un valore elevato.

[2] Una matrice A si definisce definita e positiva se la matrice A è quadrata e tale per cui, definito un vettore x ed il suo complesso coniugato $\bar{x}$, la parte reale di $\bar{x}Ax$ è positiva per ogni vettore complesso x diverso dal vettore nullo.

4.5.3 Metodo di Levenberg Marquardt

Si tratta del metodo più utilizzato per l'addestramento delle reti neurali feedforward per la sua velocità di convergenza e stabilità. Presenta anche delle limitazioni inerenti alla sensibilità sulla scelta dei pesi iniziali della rete e la tendenza all'overfitting, ma mediante strategie opportune è possibile limitare questi fattori. Avendo già introdotto la spiegazione matematica nel precedente paragrafo, verranno ora illustrati i passi dell'algoritmo:

1. Calcolo dello Jacobiano della funzione errore.
2. Calcolo del termine $(J^k)^T e^k$.
3. Risoluzione equazione 4.29 per la ricerca del passo di iterazione.
4. Aggiornamento dei pesi mediante termine d (ricavato al punto precedente).
5. Dato il nuovo vettore dei pesi si calcola il vettore E al passo successivo.
6. Se il rapporto tra gli errori di due passi di iterazione successivi non è diminuito, si calcola un nuovo vettore dei pesi agendo sul coefficiente di smorzamento μ.
7. Si riparte dal punto 3 fino a convergenza.

4.6 Overfitting

Quando si sono elencate le proprietà di una rete neurale era stato specificato come tra queste la più critica fosse la generalizzazione, ovvero la capacità della rete non solo di approssimare i dati sperimentali forniti in input, ma anche di fornire risposte affidabili quando vengono presentati dei casi diversi da quelli dell'addestramento. Per ottenere una generalizzazione efficace si potrebbe pensare di aumentare indefinitivamente le dimensioni della rete (in termini di strati nascosti e neuroni per ciascun layer). Tuttavia nelle applicazioni pratiche i dataset in input sono affetti da un certo livello di "rumore" e una rete eccessivamente elaborata è in grado di imparare tali errori rendendo inefficace ogni processo di estrapolazione. Infatti tali perturbazioni possono essere tipici di quel modello, ma non essere caratteristici per la funzione che si sta cercando. Questo fenomeno è riferito con il termine di overfitting o overtraining ed indica il processo di perdita di generalizzazione nella mappatura tra dati di ingresso ed uscita.

4.6.1 Tecniche per prevenire l'overfitting

Una delle metodologie più comuni e facili da implementare per evitare il fenomeno consiste nel dividere l'intero dataset in due subset: un insieme per l'addestramento (training set) ed un insieme di validazione (validation set). Si addestra la rete neurale utilizzando solo i dati presenti nel training set, ma ogni tanto si interrompe l'apprendimento e si testano le sue prestazioni sul set di validazione. In questa fase non avviene il processo di aggiornamento dei pesi. Poiché i dati di convalida sono indipendenti dai dati di apprendimento, le prestazioni rilevate possono essere valutate come la capacità di generalizzare della rete. Fintantoché la rete "impara" la mappatura tra input e output le prestazioni sul validation set miglioreranno, mentre quando essa comincerà a riconoscere i disturbi presenti nei dati campione le prestazioni sul set di validazione smetteranno di crescere. Nella figura successiva viene riportato un tipico esempio di overfitting.

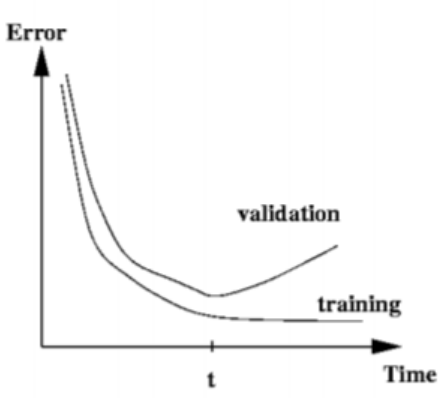


Figura 13 Andamento dell'errore sul set di addestramento e validazione (Gallo & De Bonis)

Dalla figura si può notare come il fenomeno dell'overtraining si manifesti al tempo t; è possibile dunque interrompere l'addestramento per tale valore al fine di ottenere delle prestazioni ottimali dalla rete. Questo metodo è comunemente noto come arresto anticipato (dall'inglese early stopping). Una particolarità di questa applicazione è che richiede nella sua implementazione un ulteriore set di dati ricavati dal database originario (se sufficientemente grande) oppure indipendente dai primi due, noto con il termine di test set. Il set di validazione viene utilizzato per decidere quando interrompere l'addestramento e quindi la rete così formata non sarà più del tutto indipendente da tale set. La presenza del test set è giustificata (e necessaria) per misurare la capacità di generalizzazione della rete. Un altro metodo per ridurre l'importanza delle caratteristiche

di rumore specifico associato al proprio insieme di dati è quello di aumentare la fonte di disturbo aggiungendo una quantità supplementare di rumore molto piccola (ε) e caratterizzata da un valore medio pari a zero. Nonostante l'introduzione di una perturbazione sui propri dati possa sembrare strana, è dimostrato come in alcuni casi possa effettivamente ridurre l'overfitting e migliorare la generalizzazione della rete. L'ultimo metodo presentato è noto con il termine di decadimento dei pesi. Questa applicazione prevede l'aggiunta di una penalità per la funzione di errore:

$$\tilde{E} = E + \nu\Omega \tag{4.33}$$

Dove

$$\Omega = \frac{1}{2}\sum_i w_i^2 \tag{4.34}$$

Nel processo di retro-propagazione dell'errore, nella quale si ricorda vengono aggiornati i pesi sinaptici per minimizzare l'errore, a causa della nuova definizione della funzione errore si ha una nuova funzione di aggiornamento:

$$\Delta w_{ij} = -\mu * \frac{\partial \tilde{E}}{\partial w_{ij}} = -\mu * \frac{\partial E}{\partial w_{ij}} = -\mu\nu w_{ij} \tag{4.35}$$

Cosicché i pesi sinaptici di dimensione maggiore, dai quali si intuisce che la mappatura input-output è nella fase di overfitting, vengono penalizzati e la loro dimensione ridotta in funzione di quella originaria. (Gallo & De Bonis)

Bibliografia

Fabio, R. (2005/06). Tesi di laurea specialistica in economia e gestione delle reti e dell'innovazione. *La motivazione come determinante del comportamento di organismi artificiali: una simulazione di artificial life*. Modena, Italia.

Gallo, C., & De Bonis, M. (s.d.). *Reti neurali artificiali.* Foggia.

Grippo, L., & Sciandrone, M. (2002). *Metodi di ottimizzazione per le reti neurali.* Roma.

Hassoun, H. (1995). *Fundamentals of artificial neural network.* MIT Press.

Haykin, S. (2009). *Neural networks and learning machines (third edition).* Ontario: Pearson education.

McCulloch, W. S., & Pitts, W. (1990). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology vol 52*, 99-115.

Samuel, R. B. (s.d.). *Appunti di reti neurali*.

CAPITOLO 5

5.1 Programmazione genetica

La programmazione genetica (GP) è una versione di algoritmo evolutivo (EA) che trasporta in linguaggio informatico quello che accade nel processo evolutivo dell'essere umano. L'algoritmo cerca di evolvere una soluzione adottando il principio Darwiniano della selezione naturale e riproduzione dell'individuo con maggiore capacità di adattamento, il quale, come accade nelle specie biologiche, subirà delle trasformazioni che si ripercuoteranno sulle generazioni successive. Più in generale, la programmazione genetica è un algoritmo di ottimizzazione ispirato dalla teoria evolutiva. Appartenenti a questa categoria di algoritmi si citano i GA (Genetic Algorithm) introdotti da Holland (Holland, 1975). Essi risolvono problemi di ottimizzazione mediante la ricerca dell'individuo più adatto all'interno di una popolazione (insieme di individui). Lo schema di procedimento è il seguente: si crea in maniera randomica la popolazione iniziale e per ciascun individuo si assegna un valore rappresentativo (fitness) delle sue capacità per un determinato compito. La generazione successiva viene creata scegliendo in maniera casuale gli individui con una fitness più alta presenti nella popolazione originale. Una volta selezionati, questi individui subiscono delle operazioni di alterazione come mutazioni e crossover oppure vengono direttamente inseriti nella generazione successiva. Questo processo si itera fintantoché non vengono raggiunte delle condizioni di arresto come ad esempio il numero massimo di generazioni o un grado di fitting ottimale. Negli algoritmi genetici, gli individui vengono rappresentati mediante il loro genoma (insieme di geni), ovvero una struttura di dati che immagazzinata tutte le informazioni necessarie per caratterizzare l'individuo stesso. Per poter effettivamente implementare il processo su un calcolatore questa struttura di dati viene codificata in linguaggio binario. Koza (Koza, 1998) è stato tra i primi ad estendere il concetto di apprendimento genetico ai programmi informatici. Per rappresentare i diversi individui che andranno incontro al processo evolutivo, Koza utilizza la semantica derivante dal programma Lisp. Ciascun GP può essere ricondotto ad una specifica funzione Lisp rappresentata mediante un albero di analisi (o albero sintattico).

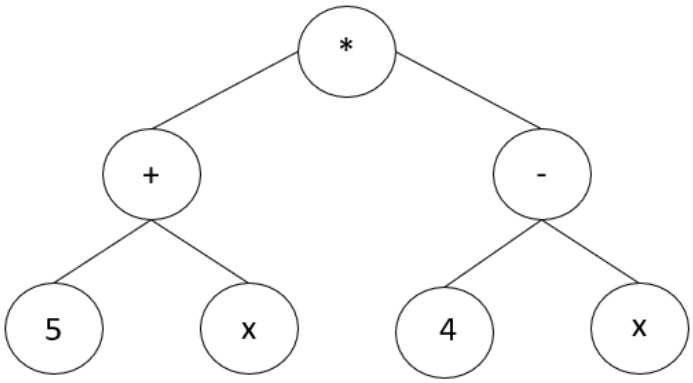


Figura 14 Rappresentazione grafica della funzione LISP (*(+5 x)(-4 x))

Questi alberi sono composti da nodi che possono essere costanti o funzioni, ciascuno dei quali viene opportunamente scelto in un determinato set, chiamati rispettivamente terminal set e function set. Durante la creazione della prima generazione ciascun albero viene creato a partire dal nodo radice (nella figura 14 è rappresentato dal *). Per assicurare la correttezza sintattica e la complessità del sistema si devono rispettare alcune regole fondamentali. Al fine di regolare la lunghezza di un albero si impone a priori una profondità limite tale per cui, una volta che l'albero si sarà sviluppato fino a questo livello, verrà scelto obbligatoriamente un nodo derivante dal terminal set. Per risolvere il problema semantico è necessario implementare un sistema di riconoscimento del numero di input necessari per applicare ciascuna funzione. Successivamente avviene la fase di valutazione di ciascun programma rispetto ad un determinato obiettivo. Nei problemi di regressione le prestazioni vengono valutate considerando quanto l'output restituito dalla programmazione genetica si avvicini al valore target, ovvero si tratta di un problema di minimizzazione dell'errore. Il valore di fitness viene usato come discriminante per individuare gli individui più adatti a quello scopo e, conseguentemente, avranno maggior probabilità di essere scelti come genitori della nuova prole. Una volta scelti, subiranno le operazioni genetiche di mutazione e crossover. Terminata la fase di modifica la nuova generazione rimpiazza quella precedente, viene nuovamente valutata la fitness ed il processo si itera fintantoché non vengono raggiunte delle condizioni di stop. Una volta terminato il processo, l'individuo avente maggior affinità con il task scelto viene fornito in output come risultato della GP. La programmazione genetica è stata adottata per la soluzione di diverse problematiche: Koza nel suo libro (Koza, 1998) analizza quattro diversi ambiti di applicazione, tra cui regressione e clustering, mostrando le potenzialità

e i limiti che tale algoritmo può incontrare. Sempre legate al lavoro di Koza si ricorda lo sviluppo di un GP in grado di rappresentare lo sviluppo sequenziale delle proteine senza predefinire la lunghezza che devono raggiungere (Koza, Bennet, Andre, & Keane, 1996) e l'utilizzo di un algoritmo che permette l'animazione 3D mediante un'analisi sui vincoli al movimento (Gritz & Hahn, 1997). Legate all'ambiente del data mining si ricorda (Al-Madi & Ludwig, 2012) nel quale vengono presentate diverse versioni dello stesso programma per evidenziare l'influenza dei vari parametri nella GP. Un altro esempio di applicazione della programmazione genetica è quello offerto da Azamathulla et al. (Azamathulla & Ab Ghani, 2011) i quali hanno sviluppato un programma che fosse in grado di prevedere il coefficiente di dispersione degli agenti inquinanti nell'acqua. Più legate all'ambiente aeronautico si citano i lavori di Dennis et al. (Dennis, Dulikravich, & Han, 2001) e Polynkin et al. (Polynkin, Toporov, & Shahpar, 2010). I primi utilizzano la programmazione genetica per minimizzare le perdite di pressione totale al fine di individuare il profilo più adatto per la paletta di una turbina, mentre Polynkin et al. hanno combinato le analisi CFD e le potenzialità del GP per creare un programma di ottimizzazione multidisciplinare che considerasse tanto i vincoli aerodinamici quanto quelli meccanici per il processo di design delle palettature delle turbomacchine. La programmazione genetica permette di superare la limitazione riscontrata negli algoritmi genetici incentrata sulla rappresentazione degli individui che, nel caso delle GA, è basata su stringhe di lunghezza prefissata, mentre per la GP si hanno degli alberi che variano la propria natura e complessità man mano che si procede con il programma. Koza definisce gli individui soggetti al processo di adattamento come una gerarchia di programmi informatici con la capacità dinamica di modificare la propria struttura (Koza R. J., 1998). Riassumendo i passaggi salienti che portano alla produzione di un determinato programma informatico capace di risolvere un dato problema:

1. Generazione della popolazione "zero" mediante composizione randomica di nodi provenienti dal function e terminal set.
2. Valutare ciascun individuo assegnando un valore di fitness coerente con la sua capacità di risolvere il problema.
3. Scelta degli individui più adatti per generare la prole mediante un criterio probabilistico basato sul livello di fitness.

4. Una volta trovati gli individui al punto 3, si applicano le operazioni genetiche di: riproduzione diretta, mutazione e crossover.
5. Introduzione della nuova generazione.
6. Itero i passaggi da 2 a 5 finché l'individuo con livello di fitting più alto viene fornito come output della programmazione genetica e, dunque, soluzione del problema.

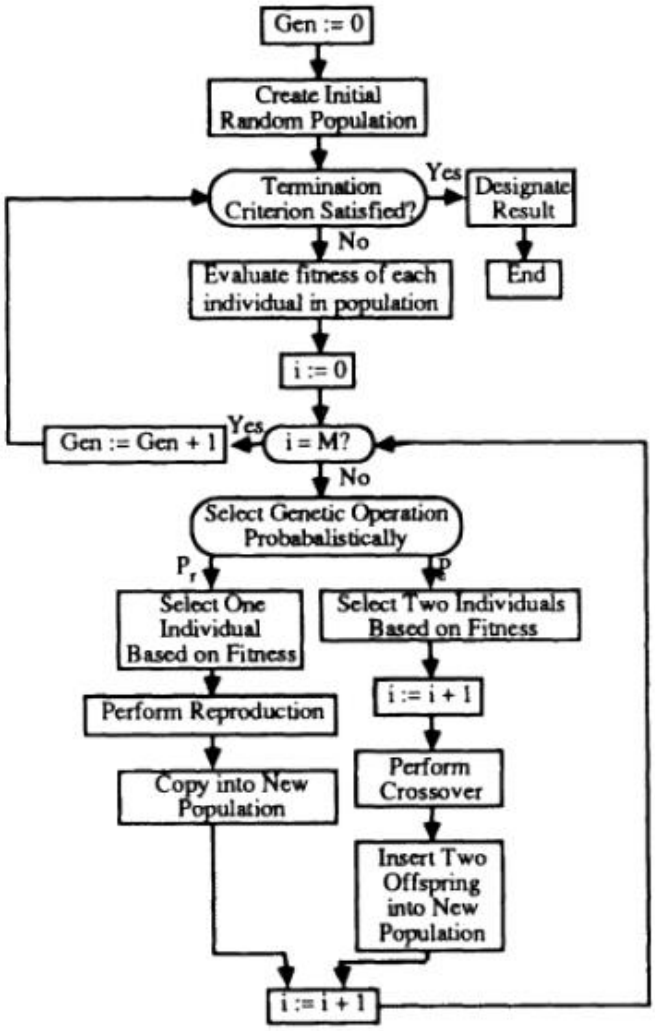


Figura 15 Diagramma di flusso del metodo programmazione genetica (Koza R. J., 1998)

Un importante aspetto della programmazione genetica è la capacità di effettuare un processo di ottimizzazione a livello strutturale del modello, facendo evolvere in contemporanea sia le funzioni che i parametri in esso presenti. Questo processo è particolarmente indicato nei problemi di regressione dove il progettista vuole scoprire la relazione matematica tra dati di ingresso ed uscita. Si capisce, di conseguenza, come la

GP sia fortemente dipendente dalla costruzione degli alberi sintattici che costituiranno i vari individui appartenenti allo spazio di ricerca della soluzione. Nei prossimi paragrafi verranno trattati tutti gli aspetti legati alla programmazione genetica.

5.2 Set di funzioni e terminali

Si definisce come function set l'insieme di N_f funzioni appartenenti all'insieme $F = \{f_1, f_2, \dots, f_{N_f}\}$ da cui verranno estratte in maniera casuale quelle che costituiranno l'individuo. Ciascuna funzione per essere implementata richiede un numero di argomenti r_i, ovvero si dice che ciascuna funzione f_i possiede arità r_i. Le funzioni presenti nel set possono essere di varia natura:

- Funzioni matematiche (esempio seno, coseno, logaritmo…)
- Operazioni algebriche (esempio somma, prodotto…)
- Cicli ricorsivi/condizionali (esempio for, if-then-else…)
- Operazioni Booleane (esempio and, not, or)
- Qualsiasi funzione definita dall'utente basata sulla sua conoscenza a priori del fenomeno analizzato.

Il set di terminali è l'insieme di N_t parametri appartenenti all'insieme $T = \{a_1, a_2, \dots, a_{N_t}\}$ da cui, come nel caso del function set, si estrarranno i nodi per la costruzione dell'albero sintattico. Questi parametri sono tipicamente delle costanti generiche oppure delle variabili note al progettista, come ad esempio le variabili di ingresso per la costruzione della funzione regressione. Si consideri l'insieme $S = \{+, -, *, log, x, y, 1\}$. Una sua possibile rappresentazione matematica può essere ricavata una volta costruito l'albero logico. Per far questo si suddivide l'insieme S nel function set $F = \{-, *, log\}$ e nel terminal set $T = \{x, 4\}$.

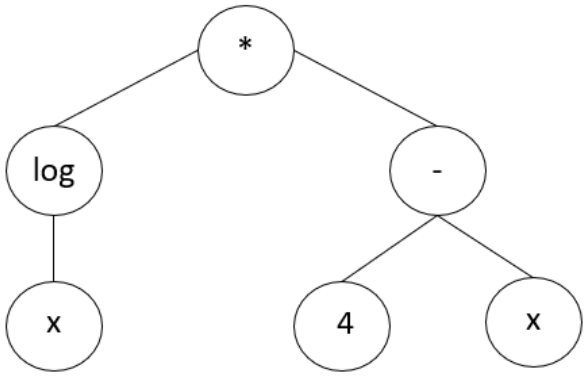


Figura 16 Esempio di albero sintattico rappresentante l'equazione $\log(x) * (4 - x)$

L'albero viene letto dall'alto verso il basso e da sinistra verso destra in maniera ricorsiva e ciascun nodo possiede un numero di rami pari al numero di argomenti richiesti; nel caso della differenza e prodotto si ha arità pari a 2 mentre per il logaritmo essa vale 1. Nella programmazione genetica, il set di funzioni e il set delle costanti devono rispettare i requisiti di sufficienza e chiusura.

5.2.1 Il requisito di chiusura

La proprietà di chiusura richiede che ciascuna funzione appartenente al set di funzioni sia in grado di accettare, come suoi argomenti, qualunque tipo di dato e valore che può essere restituito da una qualunque funzione (sempre appartenente al function set) e qualunque tipo di dato e valore che può essere assunto da un qualunque terminale del terminal set (Koza R. J., 1998). Il problema della chiusura può essere visto a sua volta come un problema di consistenza e valutazione della sicurezza (Poli, Langdon, & McPhee, 2008). Il problema della consistenza deriva dall'applicazione delle operazioni genetiche, le quali alterano la struttura dell'albero cambiando i nodi in essa presenti. È necessario dunque che ogni funzione sia in grado di accettare ogni tipo di ingresso poiché nel processo evolutivo è sempre possibile che si crei una determinata combinazione di funzioni/terminali. È necessario dunque implementare un metodo che permette a ciascuna funzione del set di avere un output coerente con i suoi ingressi e viceversa. Un'altra possibile soluzione è quella di implementare un sistema di verifica sintattica durante la fase di modifica genetica. Questo processo consiste nell'applicare gli operatori genetici di crossover e mutazione in modo tale che i figli prodotti da tali alterazioni non

possiedono delle porzioni di albero incoerenti con i nodi in esso presenti. Ciò può essere realizzato andando a verificare l'input/output della porzione mutata con quella originale; se essi sono simili/consistenti allora l'operazione genetica può essere applicata. L'altra proprietà da analizzare è la valutazione della sicurezza. Essa si rende necessaria dal momento che molte funzioni, a seguito del processo evolutivo, si ritrovano in ingresso valori per cui essa non risulta più definita. Classici esempi sono la divisione per zero, il logaritmo o la radice di un numero negativo. Questa problema può essere facilmente superato adottando una forma "protetta" delle funzioni. Questa versione delle funzioni prima ricerca eventuali problematiche e, qualora venissero riscontrate, forniscono in output un valore di default. Nel caso della divisione se il denominatore fosse zero non avremmo un numero in uscita e si violerebbe quindi la proprietà di consistenza, allora si usa la versione protetta della divisione che mi fornisce come output un numero predefinito dall'utente. Generalmente la scelta ricade sul valore uno. Nel caso di numero negativo come argomento della radice è possibile adottare la corrispettiva funzione protetta per considerare la radice del valore assoluto dell'argomento e risolvere il problema della chiusura.

5.2.2 Requisito di sufficienza

Con il termine sufficienza si definisce la necessità di poter rappresentare la soluzione del problema mediante i set di funzioni e terminali a disposizione. Tuttavia tale sicurezza è possibile solo nel caso in cui si ha la certezza che quei dati set soddisfano il problema in esame. Si voglia trovare la legge che governa la relatività $E = mc^2$ avendo come funzioni base solo la somma e differenza. È logico pensare che non si otterrà mai la soluzione desiderata poiché il set di funzioni non è sufficiente ad esprimere tale relazione. L'esempio precedente, pur essendo banale, rende l'idea che è necessario avere una certa conoscenza del problema. Si aggiunge che, qualora si adottasse un function set maggiore di quello realmente necessario per poter assicurare la sufficienza, non è stato riscontrato un eccessivo aumento del tempo computazionale, anche se si sono presentati dei casi in cui il processo evolutivo del sistema risulta instabile.

5.3 Inizializzazione della popolazione

Per la costruzione della popolazione iniziale viene scelta una funzione appartenente al function set mediante una distribuzione della probabilità uniforme per ogni elemento del set. Tale nodo viene chiamato radice dell'albero ed è ristretto al caso di una funzione per poter realizzare quella che Koza (Koza R. J., 1998) chiama una struttura gerarchica. In questo modo vengono evitati alberi di struttura degenere, ovvero costituito da un solo nodo terminale. Una volta scelto il nodo radice, da esso partiranno un numero di rami pari agli argomenti necessari per definire quella funzione. A questo punto per la scelta dei nodi appartenenti a ciascun ramo si prenderanno degli elementi appartenenti all'insieme unione del function e terminal set. Qualora venisse scelto un elemento del set di funzioni, il processo viene reiterato altrimenti, nel caso di una costante, questo nodo diventa terminale per quel ramo e il processo di generazione del ramo stesso si conclude. Definendo la profondità di un albero come il livello raggiunto dal ramo più lungo che collega la radice con le estremità, i tre metodi più usati per l'inizializzazione della popolazione sono:

1. Metodo completo (full)

   Secondo questo metodo la popolazione da creare ha una limitazione sulla profondità che può raggiungere, specificata dal parametro tree_depth_max. Nella pratica questo metodo viene realizzato imponendo la selezione ad un elemento del function set per profondità minori di quella massima ed una volta raggiunto questo parametro, la scelta viene forzata ad elementi del terminal set per completare tutti i rami dell'albero.

2. Metodo crescita (grow)

   Questo metodo crea degli alberi che possono variare di forma mediante la selezione di un elemento appartenente all'unione del function e terminal set se mi trovo ad una profondità inferiore a quella massima e la scelta viene forzata su un elemento del set di costanti solo quando mi trovo a profondità massima.

Poiché non si conoscono a priori né la forma né le dimensioni dell'albero tali per cui si abbia una soluzione accettabile del problema, si preferisce utilizzare una combinazione dei due metodi nota con il nome di ramped half and half.

3. Ramped half and half

   La popolazione viene creata usando un egual numero di alberi mediante un parametro di profondità che varia da 2 alla profondità massima specificata dal designer. Se per esempio si sceglie come profondità massima il numero 6, allora il 20% degli alberi avrà profondità 2, il 20% avrà profondità 3 e così fino alla profondità massima. A questo punto, per ogni valore di profondità, il 50% degli alberi verrà creata con metodologia full ed il restante 50% con il metodo grow.

Questo metodo risulta particolarmente vantaggioso perché permette la generazione di alberi di forma e dimensioni differenti al contrario dei due metodi precedentemente esposti. Infatti il metodo full crea degli alberi aventi tutta la stessa lunghezza e forma, mentre nel metodo grow manca la componente di generalità ottenibile con la procedura ramped. Con il termine generalità si intende la percentuale di individui per i quali non esistono delle copie identiche a sé stesso all'interno della popolazione. Questi doppioni devono essere eliminati all'interno della popolazione iniziale in quanto comportano un mal condizionamento del problema, favorendo una soluzione particolare all'interno dello spazio delle soluzioni già agli stati iniziali della programmazione genetica. Infatti bisogna ricordare che la presenza di copie con l'avanzare dei calcoli significa che si è prossimi alla soluzione. Vengono di seguito forniti degli esempi di inizializzazione della popolazione mediante i metodi full e grow. In questi particolari esempi (Poli, Langdon, & McPhee, 2008) viene utilizzato un function set $F = \{+, -, *, /\}$ ed un terminal set $T = \{x, y, 0, 1, 2\}$ con profondità massima pari a 2:

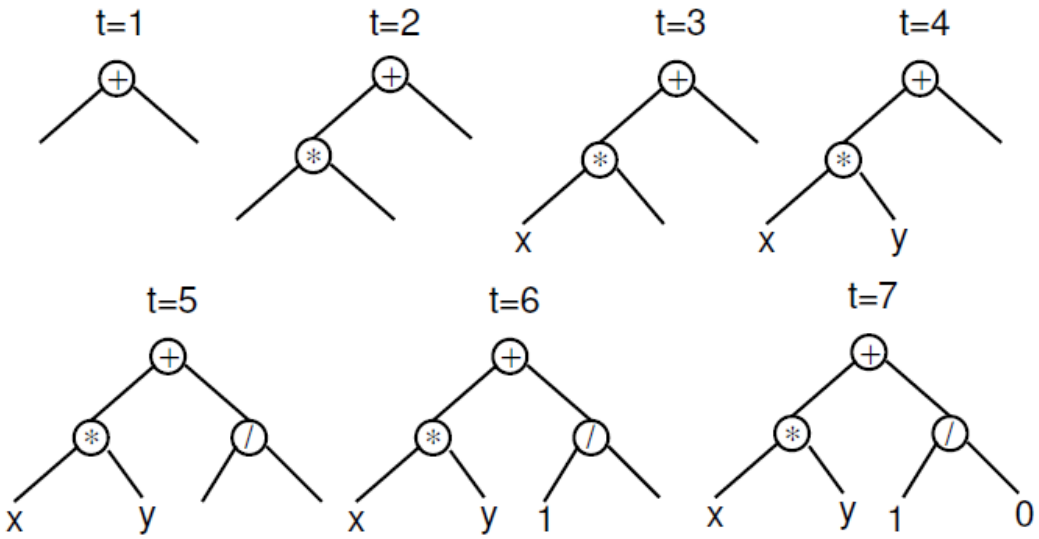


Figura 17 Processo di inizializzazione mediante metodo full. Il parametro t simboleggia i passaggi cronologici del metodo. Nell'intervallo di tempo dall'1 al 4 si ha la creazione del primo ramo; in particolare per t=1,2 trovandoci a profondità minore di 2 si sceglie un elemento del function set, mentre al tempo 3 e 4, avendo raggiunto la tree_depth_max si sceglie una costante. Il processo viene ripetuto per l'altro ramo (t=5,6,7). (Poli, Langdon, & McPhee, 2008)

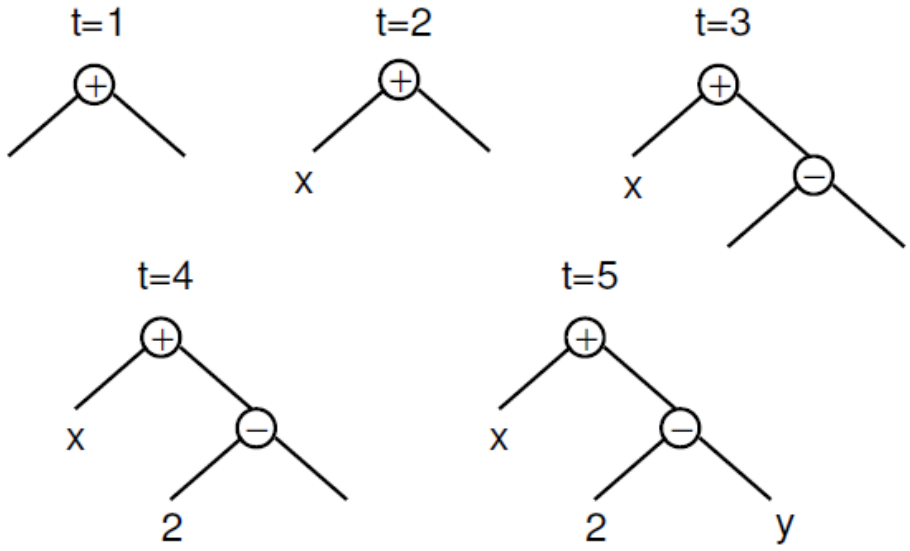


Figura 18 Processo di inizializzazione mediante metodo grow. Il parametro t simboleggia i passaggi cronologici del metodo. Si noti come, a differenza della figura precedente, la scelta tra nodo nel function set e terminal set non risulta vincolata (t=2) finché non viene raggiunta la profondità massima impostata dove la scelta è ristretta alla famiglia del terminal set (t=3,4,5). (Poli, Langdon, & McPhee, 2008)

5.4 Processo di selezione

Esistono diversi metodi per la scelta degli individui che andranno a compiere la nuova generazione. Quello utilizzato nella seguente tesi e più comunemente implementato è la selezione mediante torneo. Esso si sviluppa nel seguente modo:

1. Viene impostato un parametro k che costituisce la dimensione del torneo, ovvero il numero di individui nella generazione corrente che parteciperanno alla competizione. La scelta di questi individui è di natura stocastica.
2. Da ogni torneo si ottiene un vincitore, rappresentato dall'individuo con fitness maggiore, il quale viene inserito in un gruppo contenente gli individui che subiranno le operazioni genetiche. Questo gruppo possiede un valor medio di fitness maggiore di quello presente nella generazione corrente.

Il driver di questo modello è la pressione di selezione, ovvero un valore probabilistico della partecipazione di un individuo ad un torneo di dimensione k. Più questa pressione è elevata, più gli individui a fitness elevata vengono preferiti per partecipare al torneo, garantendo una velocità di convergenza più alta. Tuttavia, nel caso in cui la pressione sia troppo elevata, si rischia di convergere in un minimo locale e non globale della soluzione. Il valore di tale pressione di selezione è dato dalle dimensioni del torneo: se aumento le dimensioni del torneo e, conseguentemente, il numero di partecipanti, il vincitore avrà in media una fitness maggiore rispetto al valore della fitness di un vincitore di un torneo di piccole dimensioni.

5.5 Operazioni genetiche

Con il termine operazioni genetiche si indicano quelle trasformazioni che subiscono gli individui scelti nella fase del torneo, a seguito delle quali si otterrà la nuova generazione che subirà un nuovo ciclo di adattamento. Tra le svariate operazioni implementabili, nel seguente lavoro verranno trattate quelle principali: il crossover e la mutazione.

5.5.1 Crossover

Questa operazione consiste nella cessione da parte degli individui genitori di una loro parte per la creazione di un nuovo individuo. Una volta che vengono scelti gli individui, l'operatore crossover sceglie secondo una probabilità uniforme distribuita il punto in cui avverrà l'operazione. Il nodo scelto per l'operazione e tutti quelli appartenenti allo stesso ramo, ma a livelli inferiori, costituiscono quello che si definisce partizione di crossover. I due individui che entreranno nella nuova generazione vengono creati mediante uno scambio di partizioni di crossover dei genitori, ovvero un figlio viene creato cancellando dal genitore 1 la partizione ad esso relativa e quest'ultima va a sostituire la partizione di crossover del genitore 2. Il secondo figlio viene generato in maniera simmetrica. Per chiarire questi passaggi si utilizza l'esempio riportato in (Koza R. J., 1998).

Si considerino due alberi sintattici costituiti da nodi provenienti dal function set $F = \{+, -, *\}$ e dal terminal set $T = \{A, B, C\}$.

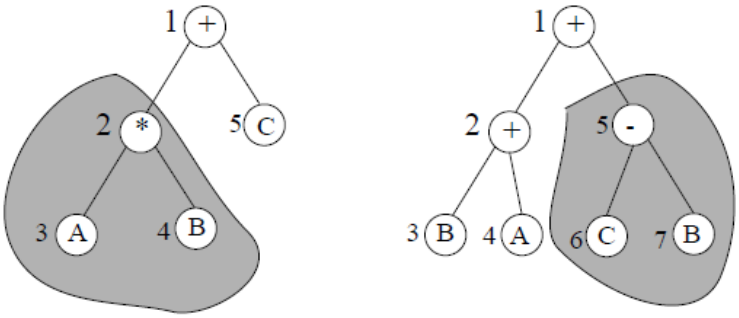


Figura 19 Due individui genitori: il primo (quello riportato a sinistra) corrisponde all'equazione $(A * B) + C$, mentre quello di destra esprime l'equazione $(B + A) + (C - B)$. (Koza R. J., 1998)

Si immagini di aver numerato ciascun nodo nell'individuo e che l'operatore di crossover abbia selezionato il nodo 2 per il genitore 1, che corrisponde alla funzione prodotto, ed il nodo 5 per il genitore 2, che corrisponde alla funzione sottrazione. Il ramo evidenziato

viene definito come frammento o partizione di crossover, mentre la restante parte dell'individuo è detta rimanenza. Mediante il processo di crossover si generano i due figli

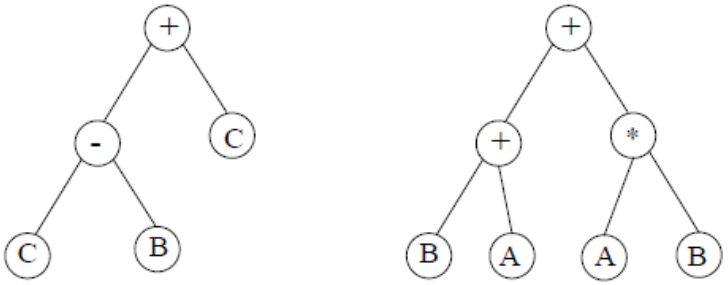


Figura 20 Risultato del crossover: il figlio di sinistra esprime l'equazione $(C - B) + C$, mentre quello di destra $(B + A) + (A * B)$. (Koza R. J., 1998)

Si noti in particolare come gli individui risultanti siano sintatticamente corretti in modo tale da rispettare il requisito di chiusura e la profondità raggiunta non superi il limite preimpostato.

5.5.2 Mutazione

Questa operazione genetica seleziona in maniera casuale un nodo all'interno dell'individuo ed elimina il ramo contenente il nodo selezionato e tutti quelli presenti ai livelli inferiori, sostituendolo con un altro ramo, sintatticamente coerente, generato in modo randomico. Si specifica tuttavia che il ramo creato dall'algoritmo è comunque costituito dagli elementi del function e terminal set. Come nel caso dell'operatore crossover, anche la mutazione necessita di una verifica sulla profondità raggiunta dal ramo generato. Un particolare tipo di mutazione è quella definita puntuale. In questo caso avviene la sostituzione di un ramo di un individuo con un nodo terminale scelto casualmente dal terminal set. Viene riportato l'esempio di (Koza R. J., 1998). Si consideri un individuo costituito dai nodi appartenenti al function set $F = \{+, , *\}$ e dal terminal set $T = \{A, B, C\}$.

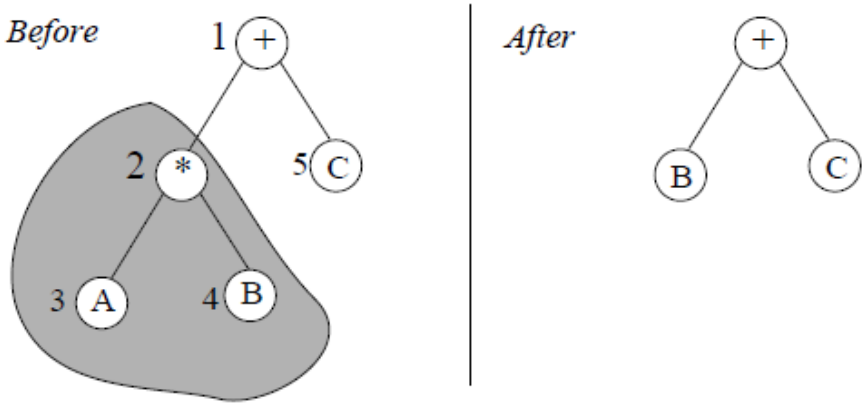


Figura 21 Individuo sottoposto a mutazione. Viene evidenziato il ramo che subirà la trasformazione. (Koza R. J., 1998)

Nella figura precedente si nota come l'operatore mutazione abbia selezionato il nodo 2 come punto di modifica dell'individuo; tale operazione influisce anche sui nodi 3 e 4 poiché situati sullo stesso ramo di 2, ma ad un livello più basso. Dopo la modifica, tale ramo è stato sostituito da un nodo terminale (B) scelto casualmente tra quelli disponibili nel terminal set.

5.6 GPTIPS

Il software utilizzato per implementare l'algoritmo di programmazione genetica è GPTIPS (Genetic Programming Toolbox for the Identification of Physical System), un programma open source scritto in linguaggio MATLAB per risolvere problemi di regressione sfruttando una tecnica di machine learning ispirata alla selezione naturale darwiniana. Più in generale questo programma risolve problemi di data mining di tipo simbolico (Symbolic Data Mining), ovvero quel processo di estrapolazione di informazioni e relazioni nei dati di input nella forma di un'equazione simbolica (Searson, 2015). Tra i vari problemi che possono essere definiti come SDM, quelli di nostro interesse sono legati alla regressione simbolica, differente dalla sua controparte classica in quanto presenta il vantaggio di modificare simultaneamente la struttura di fondo ed i parametri del modello matematico. Un modello di regressione simbolica può essere il seguente:

$$\hat{y} = f(x_1, \dots, x_M) \tag{5.1}$$

Dove $\hat{y}$ rappresenta l'output restituito dal programma una volta che gli si forniscono gli input $(x_1, \dots, x_M)$ e la risposta voluta y. Il programma restituirà una combinazione dei dati in ingresso espressa mediante una funzione o combinazione di funzioni *f*. Si sottolinea il fatto che le variabili x possono essere o meno relazionate alla risposta voluta y, infatti sarà il programma stesso ad indirizzarci sulla qualità della soluzione trovata grazie ad un indice di qualità che nel caso in esame è rappresentato dal coefficiente di determinazione o R^2, il quale è definito come:

$$R^2 = 1 - \frac{\sum_i^n (y_i - y_{pred})^2}{\sum_i^n (y_i - \bar{y})^2} \tag{5.2}$$

Dove y_{pred} rappresenta l'output desiderato mentre $\bar{y}$ rappresenta il valor medio.

Questo software utilizza una versione modificata dell'algoritmo classico di programmazione genetica in quanto tende a sviluppare individui costituiti da più alberi sintattici, che nella nomenclatura del metodo sono definiti geni, da cui il nome di programmazione genetica multi-gene (Multi Gene Genetic Programming). Un esempio di albero/gene utilizzato dal programma viene riportato nella figura successiva.

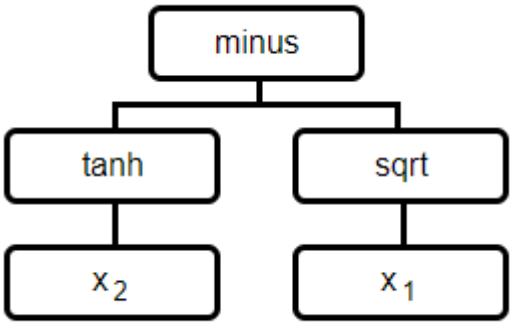


Figura 22 Gene corrispondente all'equazione $tanh(x_2) + \sqrt{x_1}$. $x_1 e\, x_2$ appartengono al terminal set, mentre tanh e sqrt fanno parte del function set.

5.6.1 Regressione simbolica multi-gene

Come già espresso precedentemente, essa si ottiene lavorando con più geni per definire un singolo individuo cosicché si ottiene un miglioramento della risposta del programma.

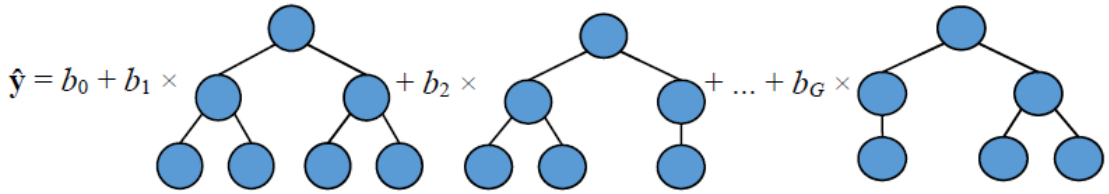


Figura 23 Esempio di risposta del sistema (Searson, 2015)

Dalla figura precedente si notano:

- b_0 termine di bias
- $b_1, b_2, \ldots, b_G$ sono coefficienti per modificare l'output di ciascun gene, detti genericamente pesi.

Si è dimostrato come la presenza del termine di bias e dei pesi possa rappresentare in maniera più efficiente delle relazioni non lineari rispetto ad un qualunque altro metodo di regressione simbolica, come ad esempio quello scalato (Scaled Symbolic Regression) (Searson, 2015).

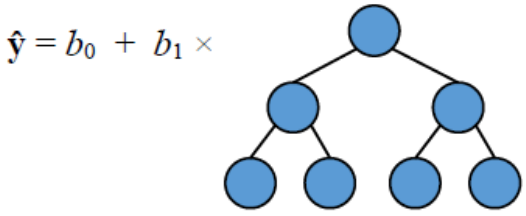


Figura 24 Esempio di output generato dalla regressione simbolica scalata. Si noti la presenza di un solo gene. (Searson, 2015)

Nel seguito viene riportato come il programma calcola i coefficienti di regressione, seguendo la trattazione specificata in (Searson, 2015).

Dato il vettore di output desiderato y di dimensione [Nx1], la risposta del programma è

$$\hat{y} = b_0 + b_1 t_1 + \cdots + b_G t_G \tag{5.3}$$

Dove t_i è il vettore [Nx1] dell'output generato dall'i-esimo gene. Successivamente, si definisce con G la matrice [Nx(G+1)] di risposta dell'individuo rappresentata come

$$G = [1\ t_1 \dots t_G] \tag{5.4}$$

In cui la colonna di 1 (Nx1) simboleggiano l'inizializzazione del termine di bias. A questo punto l'equazione 5.3 può essere riscritta come

$$\hat{y} = Gb \tag{5.5}$$

Mediante il metodo dei minimi quadrati si trovano i coefficienti $b_0, b_1, \dots, b_G$ utilizzando il vettore di output desiderato e la matrice G

$$b = (G^T G)^{-1} G^T y \tag{5.6}$$

5.6.2 Programmazione genetica multi-gene

Lo sviluppo dell'algoritmo genetico ricalca quello già espresso precedentemente. Nella fase iniziale avviene l'inizializzazione della popolazione, ovvero la creazione randomica degli individui mediante il metodo ramped half-half. Ciascun individuo è costituito da un numero di geni che varia da 1 a G_{max}, dove G_{max} è il numero massimo di geni definito dall'utente. Ciascun gene, rappresentato da un albero sintattico sottoposto alle limitazioni di profondità preimpostate dall'utente, viene creato utilizzando sia elementi del function set che del terminal set e, qualora si imponessero, anche dalle Ephemeral Random Constant (ERC). Quest'ultime rappresentano delle costanti appartenenti ad un intervallo predefinito che vengono generate in modo casuale ed utilizzate gli elementi del terminal set. Successivamente avviene il processo di selezione degli individui mediante un processo probabilistico impostato dall'utente, che può essere:

- Torneo classico
- Torneo a Pareto

Nel caso del torneo a Pareto l'individuo viene scelto sia in base al livello di fitness sia in base alla sua complessità, preferendo individui meno complessi. La funzione di fitness utilizzata nell'algoritmo è l'errore quadratico medio. Posto il vettore errore come la differenza tra i valori del dataset di training ed i valori che si vogliono ottenere, l'errore quadratico medio è definito come la radice quadrata del rapporto tra il prodotto del vettore errore trasposto per il vettore stesso diviso il numero di valori nel dataset di training. In questo lavoro si definisce complessità la somma ricorsiva dei nodi ad una data profondità

presenti in ciascun ramo a partire dai livelli a profondità maggiore. Per chiarire il criterio di valutazione si faccia riferimento alla figura 22. Se si considera il nodo radice (minus) posto al livello zero, allora si avranno a profondità 1 i nodi del function set (tanh e sqrt), mentre a profondità 2 i nodi del terminal set (x_2, x_1). Si comincia dal livello 2: per il ramo di sinistra e di destra ho un solo nodo, quindi la complessità alla profondità 2 è di 2. Si passa al livello 1: sia per il ramo di destra che di sinistra ho 2 nodi derivanti dal livello inferiore (x_2, x_1) a cui si aggiungono quelli allo stesso livello (tanh,sqrt) per un totale di 4 nodi. Infine si raggiunge il nodo radice per il quale, facendo il ragionamento appena descritto, ho 5 nodi. A questo punto si effettua la somma delle complessità per ogni livello di profondità:

Profondità	*Complessità*
2	2
1	4
0	5
Complessità individuo	**11**

A questo punto vengono utilizzate le operazioni genetiche per garantire una certa diversità all'interno della generazione. Esistono due tipi di crossover, uno definito high-level crossover e l'altro subtree crossover o low level crossover. Quest'ultimo è riconducibile all'operatore classico già presentato nel paragrafo 5.5.1 con la differenza che, trattando individui a più geni, inizialmente viene scelto un gene per ciascun genitore e successivamente avviene lo scambio di una porzione di questi alberi. Nel caso invece di high level crossover, i due genitori selezionati mediante Pareto o torneo possono scambiarsi interi geni/alberi. Si riporta il seguente esempio:

Genitore 1	A,B,C,D
Genitore 2	E,F,G,H

Si supponga di aver imposto un $G_{max} = 6$. L'algoritmo ha identificato il gene A del genitore 1 e il gene G del genitore 2 per l'operazione di crossover. In seguito ad essa si avranno:

Figlio 1	G,B,C,D
Figlio 2	E,F,A,H

Qualora un figlio possedesse un numero di geni superiore a G_{max} vengono scelti in maniera casuale e poi rimossi tanti geni quanti quelli necessari per non violare la condizione.

Bibliografia

Al-Madi, N., & Ludwig, S. A. (2012). Adaptive Genetic Programming applied to Classification in Data Mining. *Fourth World Congress on Nature and Biologically Inspired Computing*, (p. 79-85). Mexico City.

Azamathulla, H. M., & Ab Ghani, A. (2011, Aprile). Genetic Programming for Predicting Longitudinal Dispersion Coefficients in Streams. *Water Resources Management*, p. 1537-1544.

Dennis, B. H., Dulikravich, G. S., & Han, Z.-X. (2001, Settembre-Ottobre). Optimization of turbomachinary airfoil with a genetic/sequential-quadratic-programming algorithm. *Journal of propulsion and power*, p. 1123-1128.

Gritz, L., & Hahn, J. K. (1997). Genetic Programming Evolution of Controllers for 3-D. *GENETIC PROGRAMMING 1997: PROCEEDINGS OF THE SECOND ANNUAL CONFERENCE* (p. 139-146). San Francisco: Morgan Kaufmann.

Holland, J. (1975). *Adaptation in natural and artificial system.* The University of Michigan Press.

Koza, R. J. (1998). *Genetic Programming on the programming of computers by means of natural selection.* Cambrige: Sixth printing.

Koza, R., Bennet, F. H., Andre, D., & Keane, M. A. (1996). Four problems for which a computer program evolved by genetic programming is competitive with human performance. *Evolutionary Computation*, (p. 1-10).

Poli, Langdon, & McPhee. (2008). *A field guide to genetic programming.*

Polynkin, A., Toporov, V., & Shahpar, S. (2010). Multidisciplinary Optimization of turbomachinary based on metamodel built by Genetic Programming. *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, (p. 1-18).

Searson, D. (2015). GPTIPS 2: an open-source software platform for symbolic data mining. In A. G. al, *Handbook of Genetic Programming Applications* (p. Chapter 22). New York: Springer.

CAPITOLO 6

6.1 Risultati

Come già anticipato precedentemente, questo lavoro è da considerarsi come la continuazione del lavoro di tesi svolto da (Balbo, Larocca, & Ferrero, 2019), dal quale si sono ottenuti i valori delle variabili fluidodinamiche per la ricerca di una relazione del campo di beta. Questi valori sono quelli appartenenti al modello di SA modificato mediante l'approccio di inversione, nel momento in cui si è ottenuta una soluzione CFD stabile e una minimizzazione significativa della funzione goal. Per quanto riguarda l'analisi del campo e del test case, si rimanda al seguente lavoro (Balbo, Larocca, & Ferrero, 2019). Il termine di produzione di SA risulta dipendente da quattro quantità fluidodinamiche locali: Ω, che rappresenta la vorticità, $\hat{\nu}$ che rappresenta la viscosità cinematica turbolenta, ν rappresenta la viscosità cinematica e d rappresenta la distanza a parete. Poiché l'utilizzo di queste quantità fisiche rappresenta un ostacolo per il processo di addestramento di machine learning in quanto risultano valori molto distanti tra di loro e l'algoritmo di machine learning spesso presenta il fenomeno di overfitting (Singh, Medida, & Duraisamy, 2017), si procede con un ridimensionamento delle variabili mediante delle quantità rilevanti per il campo di moto: $\chi, d\ e\ \hat{\nu} + \nu$. Successivamente si applica il teorema di Buckingham per esprimere le quantità fluidodinamiche locali in parametri adimensionali ricavando:

$X_1)\ \bar{\Omega} = \frac{d^2}{\hat{\nu}+\nu} * \Omega$

$X_2)\ \chi = \frac{\hat{\nu}}{\nu}$

$X_3)\ \frac{S}{\Omega}$

$X_4)\ \frac{\tau}{\tau_w}$

$X_5)\ f_d = 1 - \tanh\left((r_d)^{\frac{1}{2}}\right)$

$X_6)\ \frac{P}{D}$

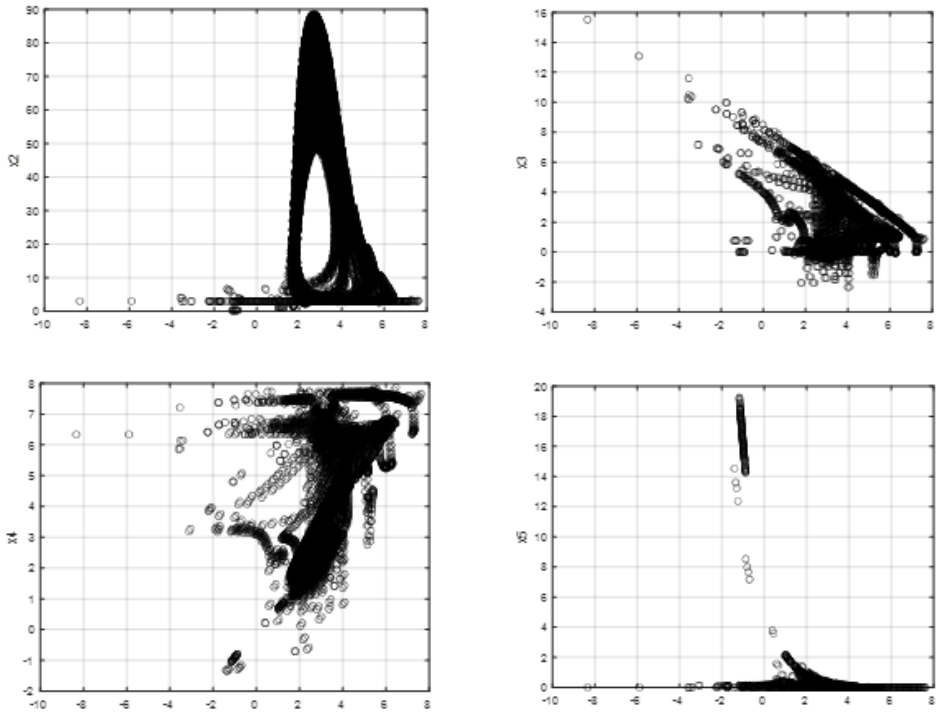
$X_7)\ \frac{d}{x_{te}-x_{le}}$

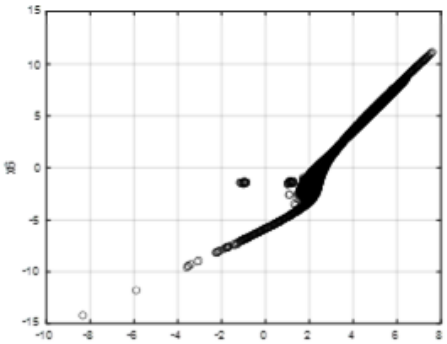
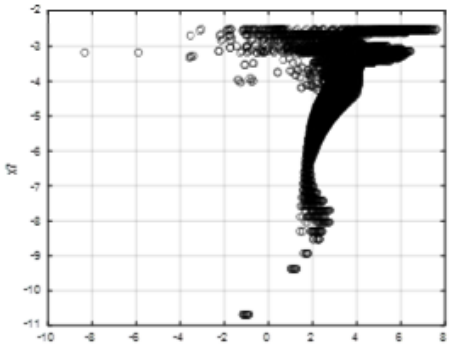
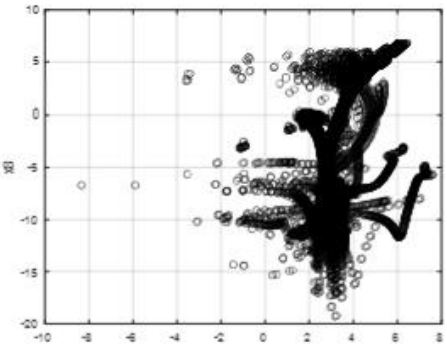
$X_8)\ \frac{\partial \hat{\nu}}{\partial x_i}\frac{\partial \hat{\nu}}{\partial x_i}$

Le variabili sopra riportate derivano dagli studi di (Singh, Medida, & Duraisamy, 2017).

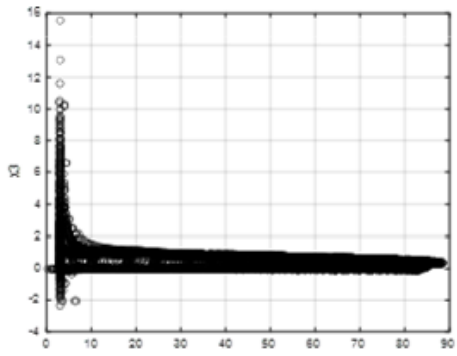
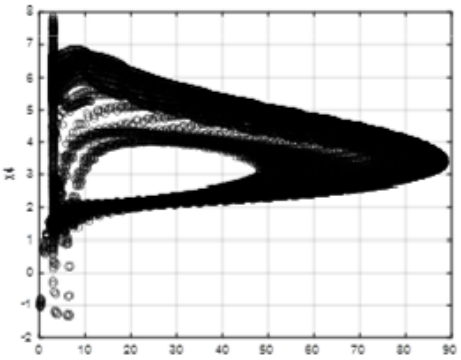
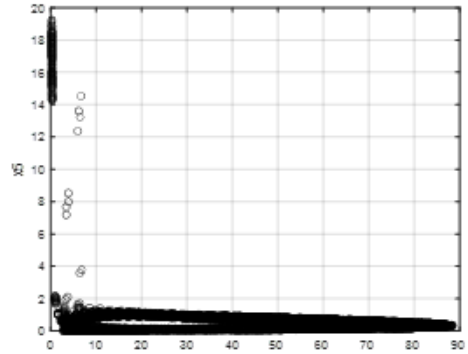
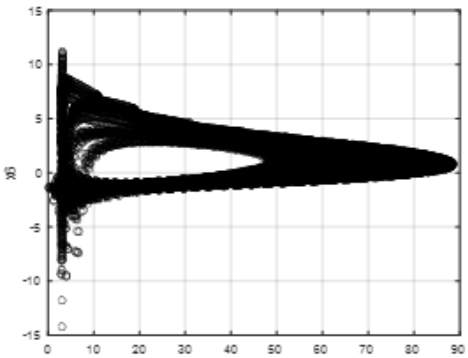
Prima di effettuare la scelta delle variabili che andranno ad addestrare gli algoritmi di machine learning è necessario verificare se tali parametri sono linearmente indipendenti. Nel caso in cui due variabili avessero una qualche proporzionalità si potrebbe pensare di includerne solamente una delle due, poiché la loro influenza sulla distribuzione del beta nel condotto sarebbe paragonabile. Se si utilizzano variabili con andamenti simili come input per gli algoritmi genetici si rischia non solo un calo di prestazioni in termini di fitting, dato dal fatto che si effettua il processo di training su un dataset molto simile, ma anche un peggioramento delle performance in ambito CFD, poiché l'algoritmo ha "imparato" una relazione ricavata da un dataset poco generalizzato e non appena gli si sottopongono dei nuovi punti (ricavati dall'iterazione temporale), l'analisi CFD o si interrompe o risulta instabile, generando risultati non interpretabili fisicamente. Nel seguito vengono riportate le tabelle che riassumono le dipendenze reciproche.

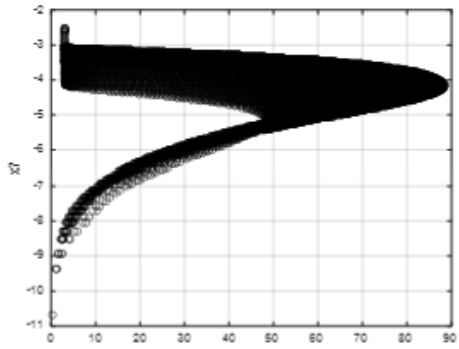
- Variabili in funzione di x_1

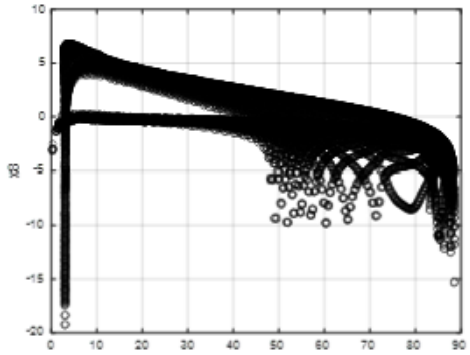


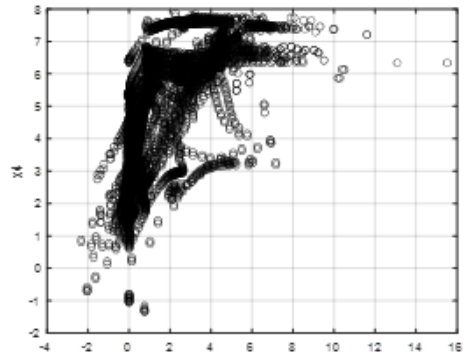
- Variabili in funzione di x_2

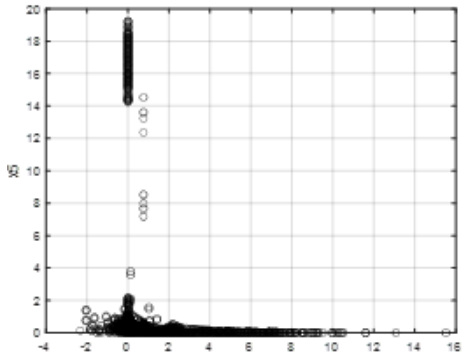





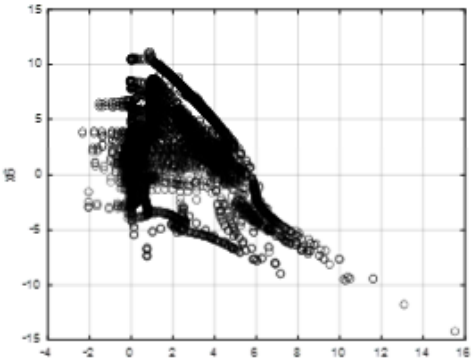


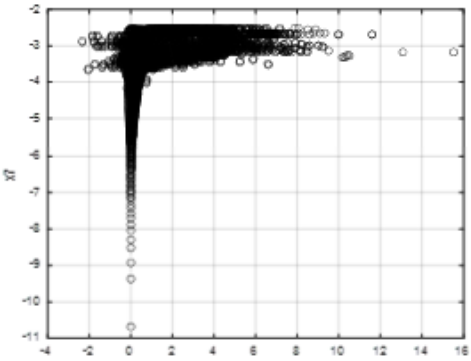


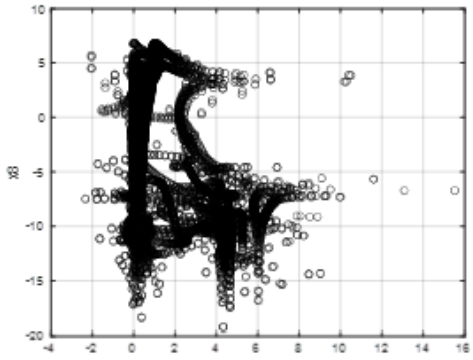
- Variabili in funzione di x_3



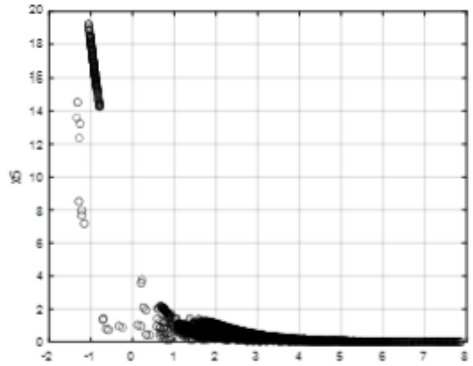
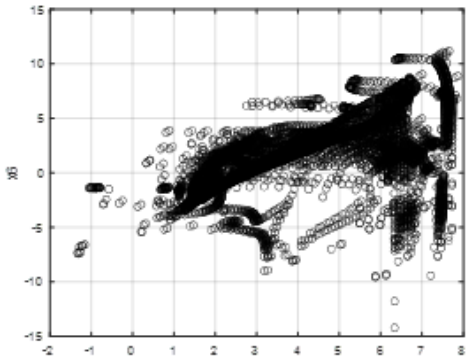
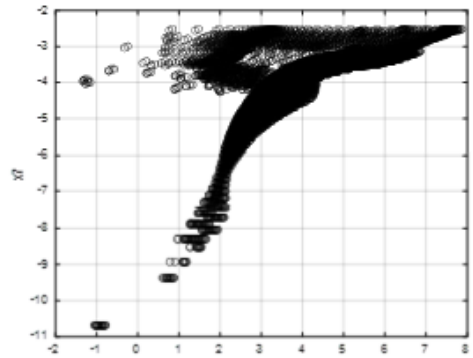
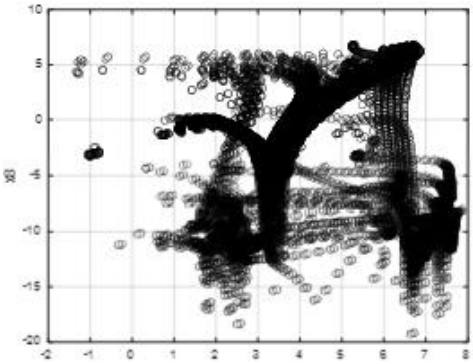




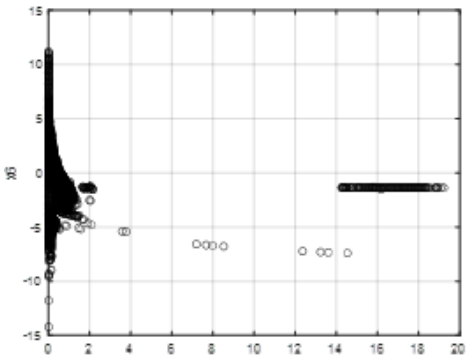
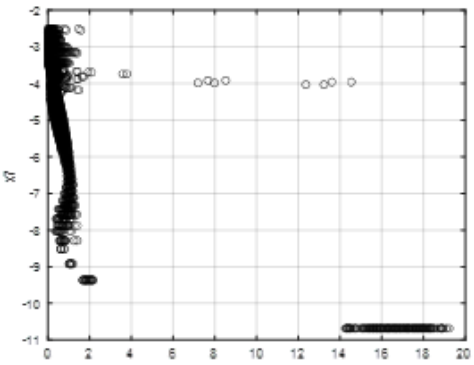
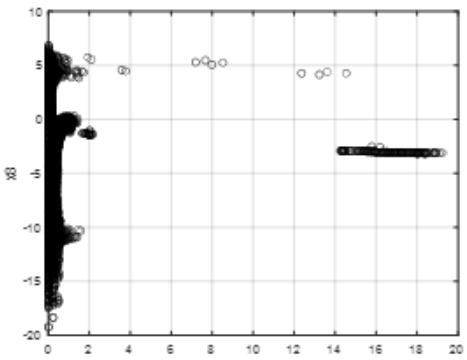




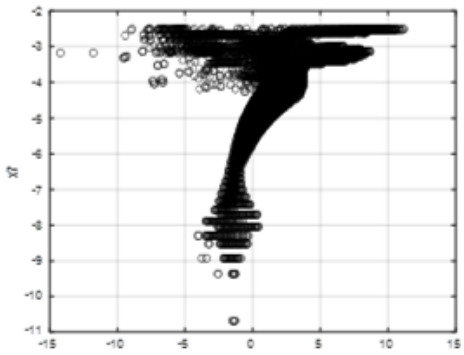
- Variabili in funzione di x_4

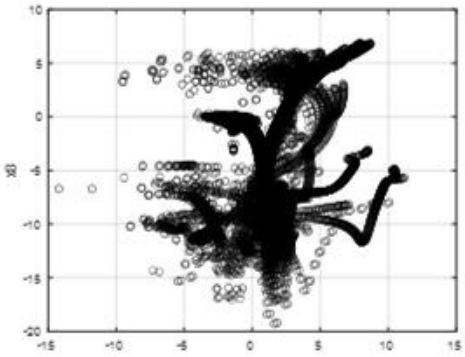





- Variabili in funzione di x_5

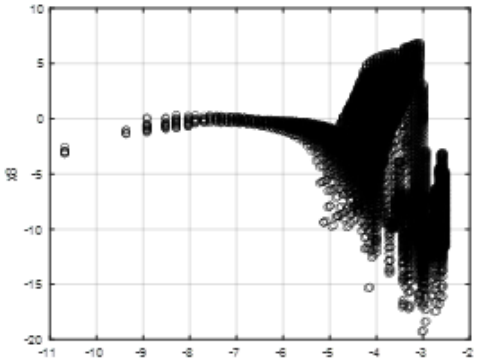




- Variabili in funzione di x_6



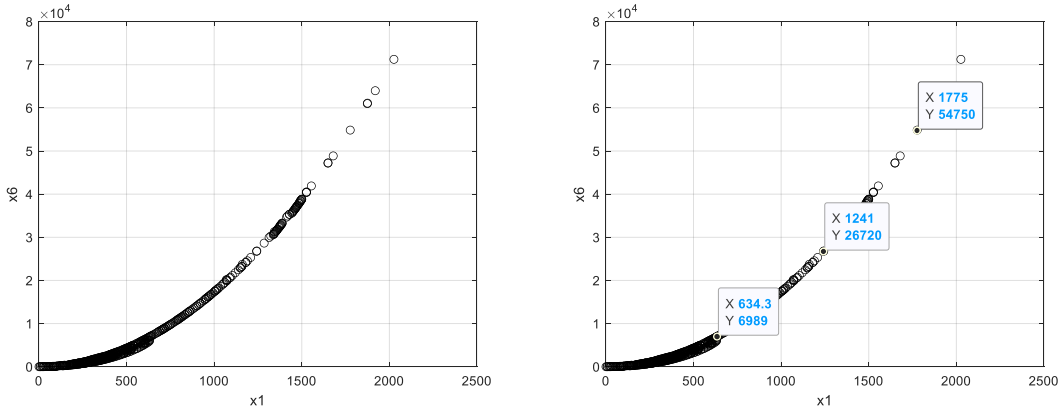


- Variabili in funzione di x_7



Si specifica che nelle tabelle sopra riportate le variabili X_1, X_3, X_4, X_6, X_7 e X_8 sono state scalate con il logaritmo in base 10 per una migliore risoluzione grafica. Si aggiunge inoltre che non sono state riportate le ovvie relazioni di identità e le relazioni inverse; ovvero una volta riportato il grafico (x_i, x_j) per la variabile i-esima non è stato ricavato il grafico (x_j, x_i) per la variabile j-esima.

Dai grafici tabulati precedentemente si nota una certa dispersione dei valori, fattore positivo per l'addestramento. Da notare in particolare la relazione tra x_1 e x_6 che presenta un andamento simil lineare. Prima di escludere una delle due dalle possibili candidate al training degli algoritmi genetici, si va a verificare tale relazione senza l'operatore logaritmo, ottenendo:



A prima vista l'andamento sembra essere del tipo parabolico, ma utilizzando lo strumento di selezione puntuale per i grafici di Matlab si nota come tale andamento in realtà è dovuto alla diversa scala degli assi cartesiani. Si conclude quindi dicendo che tutte le variabili in esame possono essere utilizzate in una qualche combinazione di esse per addestrare gli algoritmi di machine learning al fine di trovare un'espressione del coefficiente beta per il campo di moto in esame.

La soluzione di riferimento, connotata in questo lavoro come "soluzione al primo passo temporale", è ottenuta inizializzando la soluzione con la soluzione ottimale ottenuta dal processo di inversione di campo presentato nel lavoro (Balbo, Larocca, & Ferrero, 2019).

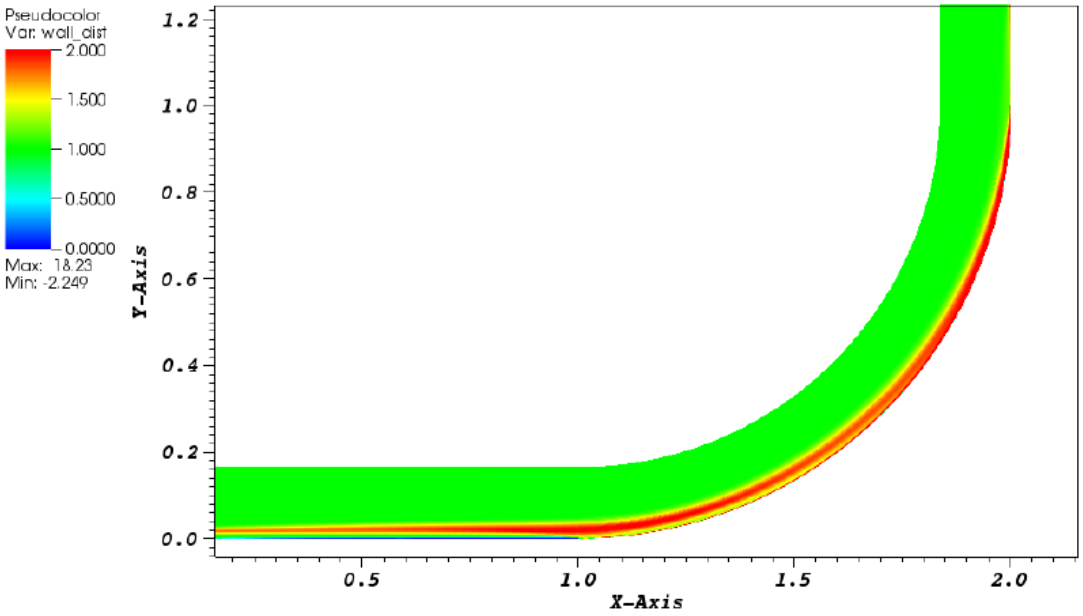


Figura 25 Distribuzione beta condotto per soluzione alla prima iterazione temporale (Balbo, Larocca, & Ferrero, 2019)

Lo sviluppo della rete neurale è stato realizzato mediante il pacchetto Deep Learning Toolbox presente nel software Matlab. Per la costruzione della rete neurale si è inizialmente adottata un'architettura a 4 layers in cui i due layers nascosti sono costituiti ciascuno da 20 neuroni. Questa architettura deriva da analisi effettuate su un campo di moto con alcune caratteristiche analoghe a quello in esame (Ferrero, Iollo, & Larocca, Field inversion for data-augmented RANS modelling in turbomachinery flows, 2020). Si specifica inoltre che per garantire una maggiore stabilità del codice CFD si è adottata una funzione di attivazione nell'output layer uguale alla funzione identità, mentre per i singoli layer si è utilizzata la funzione tangente sigmoidea. L'algoritmo di addestramento è quello riportato al paragrafo 4.5.3. La prima prova è stata realizzata incorporando nell'algoritmo tutti e otto gli input con risultati dal punto di vista del fitting molto elevati. Nella figura successiva si possono notare quattro grafici che indicano i livelli di fitting presentati rispettivamente sul dataset di training (70%), validation (15%), test (15%) e sul database completo.

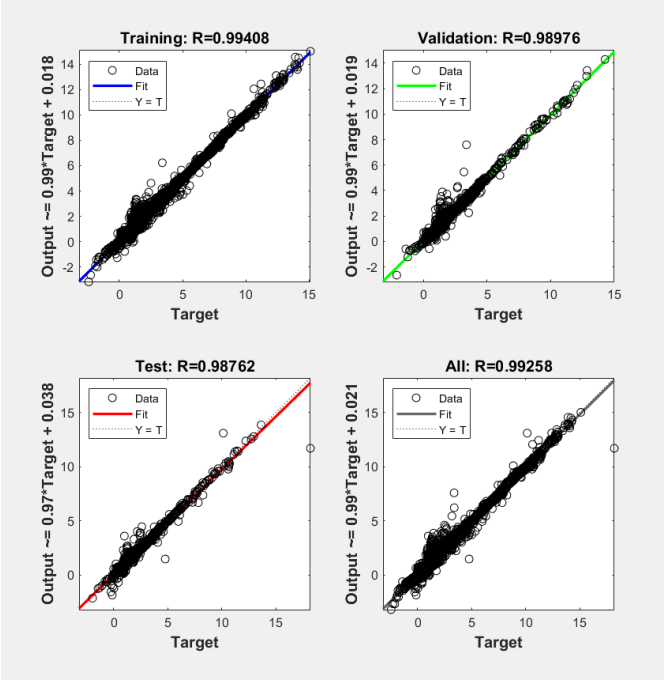


Figura 26 Riassunto prestazioni rete neurale 2x20 a 8 input

Una volta salvati in un file .txt i pesi, bias, output ed architettura di rete, mediante uno script generato in Fortran che permette di leggere la rete, la si implementa nel codice CFD sviluppato da Larocca e Ferrero (Ferrero, 2014) per vedere l'evoluzione del campo ad istanti temporali successivi. Al primo passo di iterazione, figura 27, accade che il codice esplode, generando una soluzione insensata, soprattutto se paragonata con quella ricavata

dal metodo di inversione di campo, alla quale questa soluzione dovrebbe quantomeno avvicinarsi. Da notare in particolare la zona affetta da strato limite, dove la variazione del beta in direzione ortogonale al vettore velocità ha un andamento altamente disomogeneo.

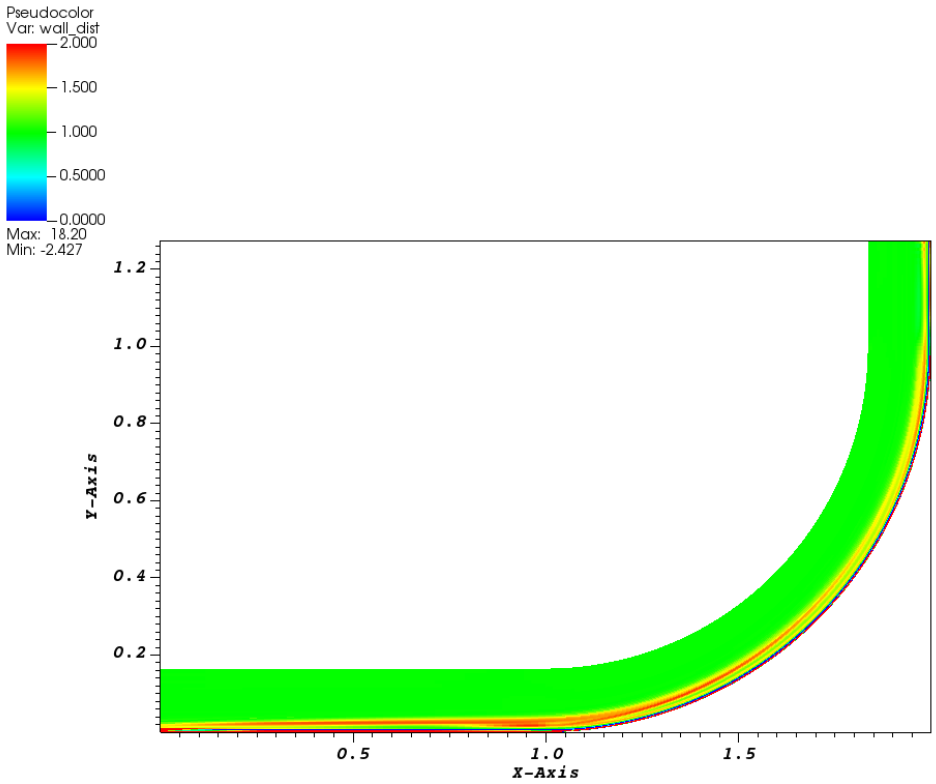


Figura 27 Distribuzione del beta al primo passo temporale.

Data l'instabilità del calcolo CFD in presenza della rete neurale, si è presa in considerazione la possibilità che la rete ottenuta fosse estremamente sensibile agli input e quindi responsabile dell'instabilità osservata nelle simulazioni. Si è pensato, dunque, di analizzare l'influenza di ciascun parametro sul campo di moto mediante l'analisi delle derivate. Innanzitutto è stato creato uno script Fortran che calcola le derivate del parametro beta rispetto ai singoli input mediante l'approssimazione con rapporti incrementali. A questo punto si procede con il calcolo della norma 2 del vettore gradiente (rispetto alla singola variabile) per avere un parametro che ci fornisce un informazione sul condizionamento di quell'input all'interno del campo di moto.

$$\left\| \frac{\partial \beta}{\partial x_i} \right\| = \sqrt{\frac{\sum_{j=1}^{N_{tot}} \left(\frac{\partial \beta}{\partial x_i} \right)_j^2}{N_{tot}}} \tag{6.1}$$

Dove la sommatoria è estesa a tutti i punti del dominio di discretizzazione N_{tot} ed i è l'indice riferito alla variabile in esame.

Da questa analisi risulta:

Gradiente	*Valore*
$\frac{\partial \beta}{\partial x_1}$	0.66
$\frac{\partial \beta}{\partial x_2}$	0.89
$\frac{\partial \beta}{\partial x_3}$	$1.29 * 10^{-4}$
$\frac{\partial \beta}{\partial x_4}$	0.51
$\frac{\partial \beta}{\partial x_5}$	24
$\frac{\partial \beta}{\partial x_6}$	$1.83 * 10^{-3}$
$\frac{\partial \beta}{\partial x_7}$	4730
$\frac{\partial \beta}{\partial x_8}$	0.569

Come si può osservare dalla tabella sopra riportata, alcuni valori dei gradienti risultano essere molto diversi dall'andamento medio. In particolare la variabile x_7 presenta una forte variabilità alle condizioni di calcolo specifiche data dall'alto valore della norma del gradiente, ragion per cui si è pensato di toglierla dalle variabili in ingresso alla rete. In termini di fitting siamo sempre su valori elevati (>0.91), ma una volta visualizzata la soluzione CFD si notano alcune problematiche, riportate in figura 28.

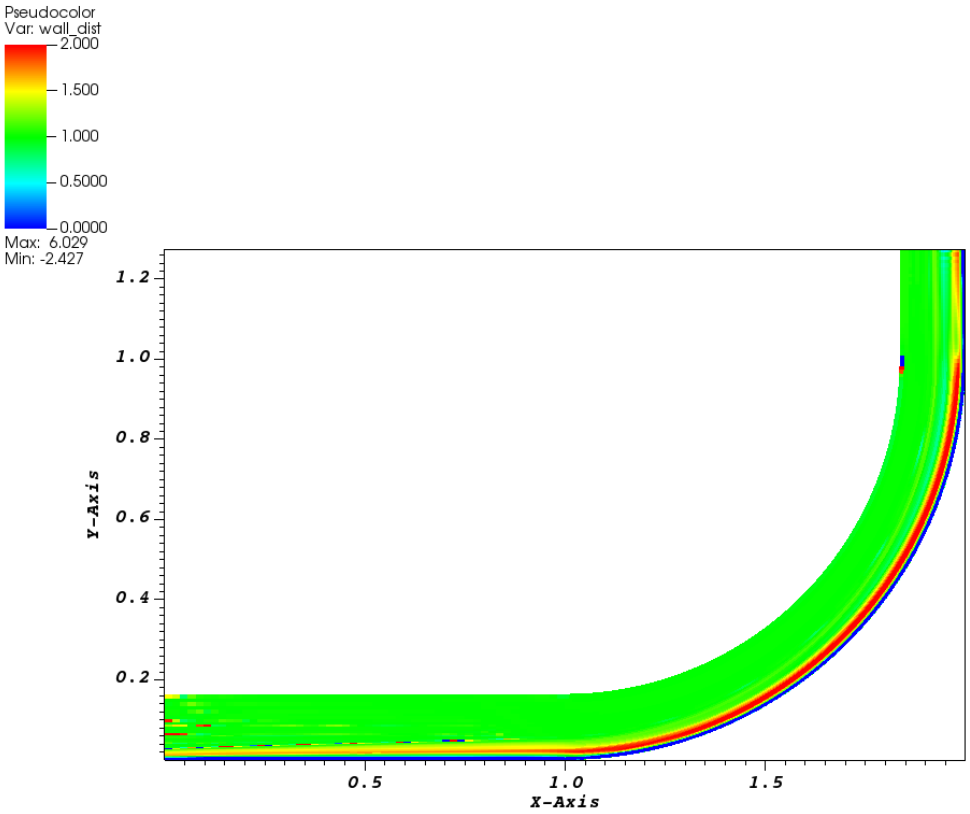


Figura 28 Distribuzione beta 7 input

Nonostante il codice effettui 60 iterazioni temporali e all'interno dello strato limite la stratificazione sia più definita rispetto al caso degli otto input, è possibile notare la comparsa di zone a beta non unitario nella regione dove, essendo il moto uniforme, non dovrebbe avvenire alcuna correzione del modello di SA originale. Questa regione è rappresentata in figura dal campo verde. Avendo diminuito il numero di input, ma mantenuto la stessa architettura di rete, è possibile essere caduti nel problema di overfitting. A questo punto si è proceduto analizzando un' architettura di rete simile a quella precedentemente trattata, ma con la differenza di avere 10 neuroni in ciascuno degli hidden layers. Il primo caso trattato é quello di una rete neurale avente tutti e 8 gli input, in quanto si vuole stabilire se è possibile raggiungere una stabilità del codice CFD ora che si ha un'architettura di rete più semplice. Il risultato ottenuto è presentato in figura 29:

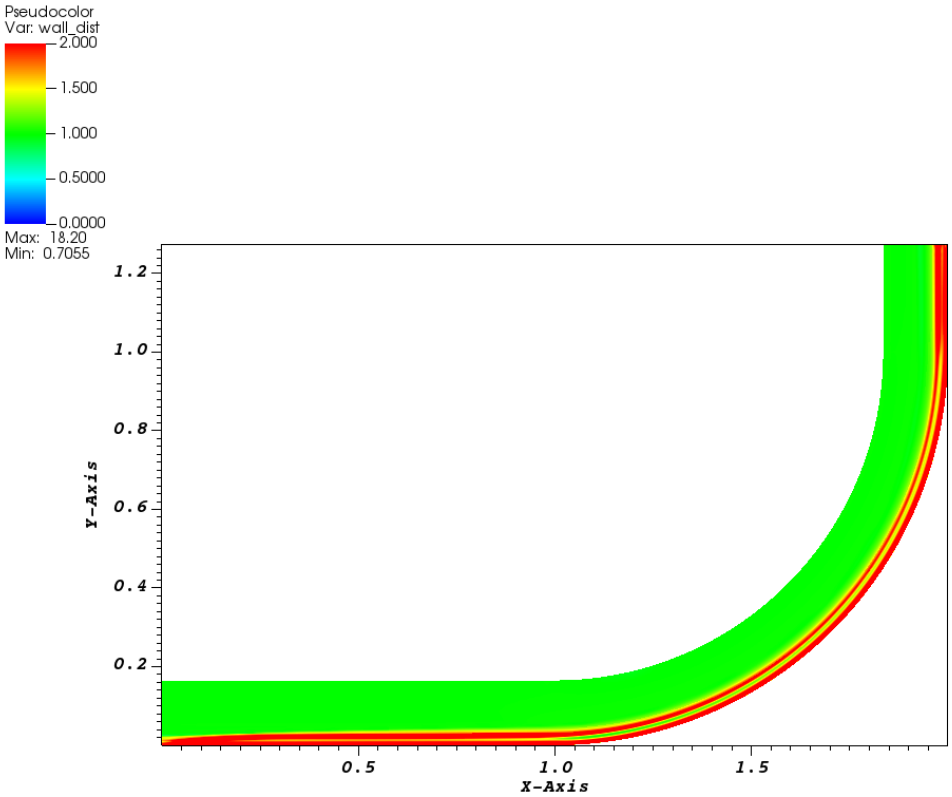


Figura 29 Distribuzione beta 8 input per rete 2x10

Rispetto al caso di rete 2x20, il codice è riuscito a fare 30 iterazioni prima di arrestarsi. Tuttavia i risultati riportati non sono soddisfacenti, infatti in questo caso si ha una sovrastima del coefficiente di correzione, che si estende ben oltre la zona interessata dallo strato limite. Inoltre la correzione all'interno dello stesso ha un andamento quasi uniforme (zona rossa) tanto nella superficie di ingresso quanto in quella curvilinea della parete inferiore, il che è in disaccordo con il modello classico di SA. Verificato quindi che avere 8 input crea una instabilità nel codice, si è deciso di provare la stessa architettura di rete avente tutte le variabili tranne x_7, che si ricorda essere quella fortemente dipendente dalle condizioni di calcolo.

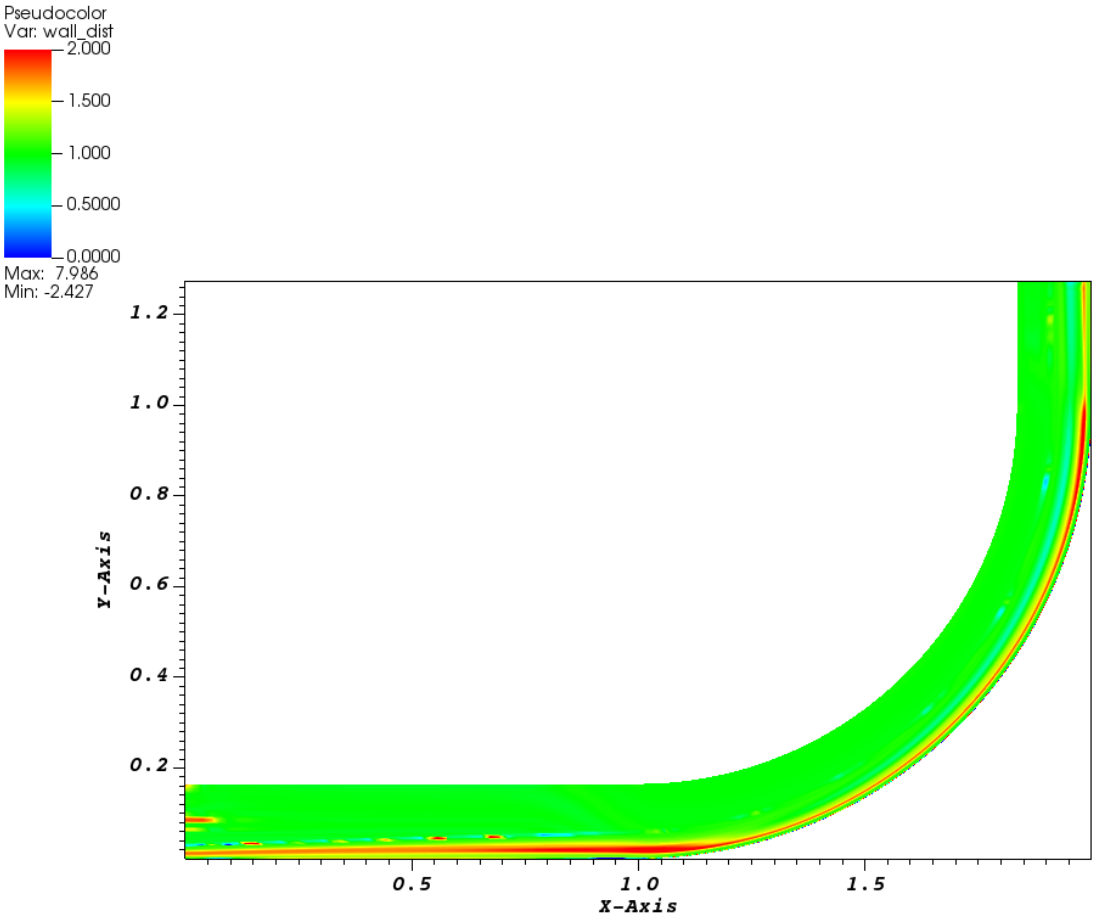


Figura 30 Distribuzione beta 7 input per rete 2x10

In figura 30 si può notare come il fattore di correzione abbia un andamento migliore sia rispetto al caso di otto input con la stessa architettura, in quanto presenta un andamento graduale del fattore di correzione all'interno dello strato limite, sia rispetto al caso con lo stesso numero di input, ma architettura 2x20, poiché nel tratto curvilineo la stratificazione del beta non è sovrastimata. In seguito è stata testata una rete neurale che non presenta né la variabile x_5 né la variabile x_7, al fine di vedere l'influenza sul CFD della mancanza delle variabili aventi norma del gradiente più alto. La rete neurale 2x10 presenta un livello di fitting molto basso ($\sim$0.7) e la soluzione CFD, di conseguenza, non risulta accettabile; in particolare si può notare in figura 31 come, secondo tale modello, le modifiche necessarie da apportare al modello base di SA nel tratto curvilineo inferiore sono quasi del tutto inesistenti all'interno dello strato limite e massime all'interfaccia tra strato limite e campo uniforme.

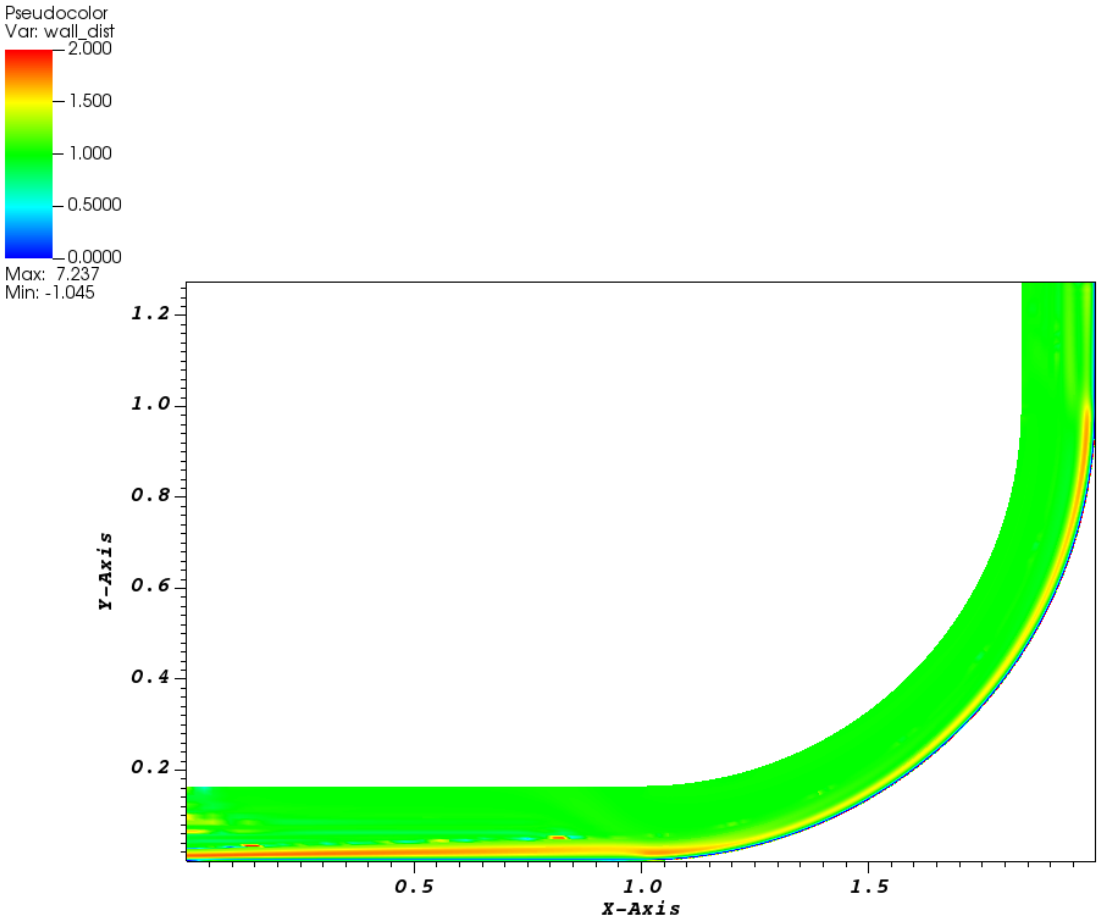


Figura 31 Distribuzione del beta per rete neurale 2x10 senza x_5 e x_7

Tuttavia persiste il problema all'ingresso e nella zona di moto uniforme, dove si ricorda che non è necessario effettuare alcuna modifica del termine di produzione del modello di Spalart-Allmaras. Data la continua presenze di zone atipiche nel campo di moto (vedere figura 32), si è pensato di rivolgere l'attenzione su quelle variabili che, al contrario di x_5 e x_7, hanno un valore della norma del gradiente piccolo. In particolare le variabili in esame sono x_3 e x_6, ovvero quelle con gradienti molto minori dell'unità. Dato che la loro variabilità non sembra essere dovuta alle particolari condizioni del campo, si è supposto che esse fungono da fonte di rumore per il calcolo della rete neurale. Per verificare la veridicità di tale ipotesi è stata realizzata una rete neurale avente in ingresso tutti gli input a meno di x_3 e x_6 e con architettura 2x10.

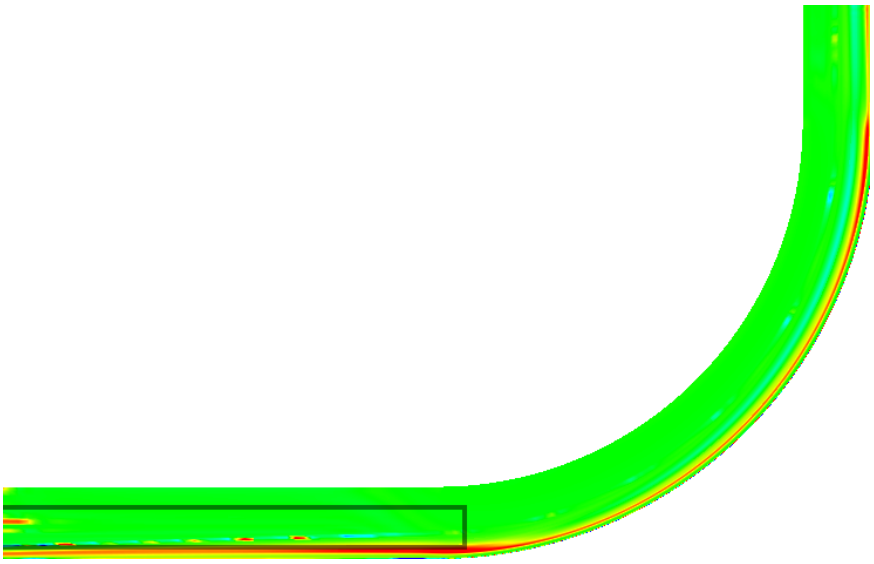


Figura 32 Evidenziata problematicità sulla distribuzione del parametro beta

Il fitting della rete è molto buono (~0.93) e dal punto di vista CFD la soluzione è rappresentata in figura 33:

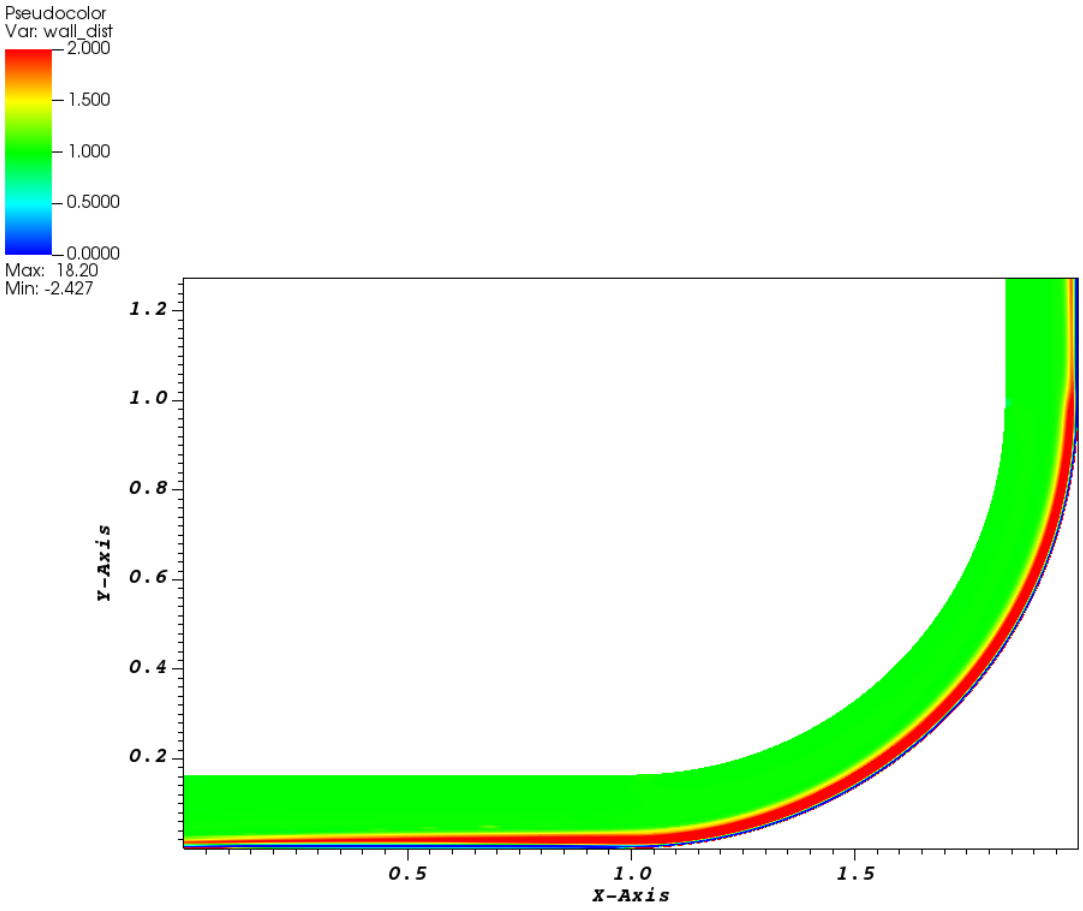


Figura 33 Distribuzione beta 6 input (esclusi x_3 e x_6) per rete 2x10

Nonostante la distribuzione del beta sia ancora una volta sovrastimata, è possibile notare come la distribuzione all'ingresso e nella zona di campo a moto uniforme si sia uniformata, confermando l'ipotesi che le variabili a gradienti più bassi fungono da fonti di disturbo per l'allenamento dell'algoritmo di machine learning e, conseguentemente, per il codice CFD.

A questo punto si procede con lo sviluppo dell'algoritmo genetico. La scelta dei parametri da adottare è stata fatta mediante diverse prove sperimentali operate sul mio computer personale da 8 GB di RAM in cui un solo parametro è stato cambiato ricorsivamente. I principali fattori che influenzano la programmazione genetica sono:

- Il numero di popolazione

  Secondo quando riportato in (Koza R. J., 1998), nei problemi di regressione di fenomeni altamente non lineari, tale numero deve essere posto superiore a 1000. Dai test effettuati questa affermazione risulta verificata. Si può notare come ad un aumento della popolazione segue un aumento del costo computazionale.

Numero popolazione	R^2_{train}	R^2_{test}	$R^2_{validation}$	Tempo di calcolo [min]
200	0.1427	0.1508	0.1524	<1
500	0.1809	0.1894	0.1885	5
800	0.2097	0.2109	0.1921	16
1000	0.2707	0.2723	0.2843	27

- Numero massimo di geni

  Per quanto riguarda questo parametro non esiste una regola prestabilita; tuttavia aumentando il numero di alberi che vanno a costituire il singolo individuo si è in grado di catturare con maggiore precisione gli andamenti non lineari e, conseguentemente, un maggior grado di complessità dell'individuo. Un problema di eccesso di geni conduce al già citato fenomeno di overfitting, quindi il processo di tuning di questo parametro va fatto rispetto al caso specifico.

Geni	R^2_{train}	R^2_{test}	$R^2_{validation}$	Tempo idi calcolo [min]	Complessità
2	0.3061	0.2831	0.3269	5	18
5	0.3102	0.2937	0.3175	7	23
8	0.3378	0.3374	0.3285	9	49
10	0.3547	0.3472	0.3483	9	72

- Profondità albero

  Questo parametro viene trattato come il precedente e spesso il processo di tuning viene fatto per diverse combinazioni di questi ultimi. Le riflessioni sono analoghe a quelle relative al numero di geni.

Profondità	R^2_{train}	R^2_{test}	$R^2_{validation}$	Tempo idi calcolo [min]	Complessità
2	0.1835	0.1764	0.1943	6	14
4	0.2861	0.2963	0.2908	6.5	41
8	0.3705	0.3842	0.3567	7	284
10	0.3774	0.3834	0.3885	21	1989

- Operazioni genetiche

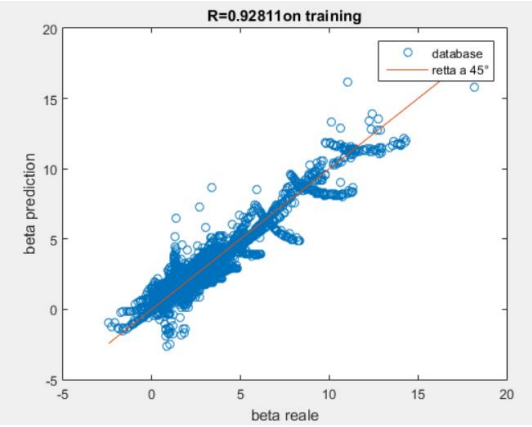
  Anche in questo campo non esiste una formulazione predefinita. Koza nel suo lavoro (Koza R. J., 1998) suggerisce di utilizzare quasi esclusivamente l'operazione di crossover in quanto da lui sperimentata come la più efficace. Altri autori (Poli, Langdon, & McPhee, 2008) suggeriscono invece l'importanza della mutazione per garantire una certa generalità all'interno dell'algoritmo. In definitiva anche la suddivisione delle probabilità che si verifichi una determinata operazione genetica dipende dal caso in esame.

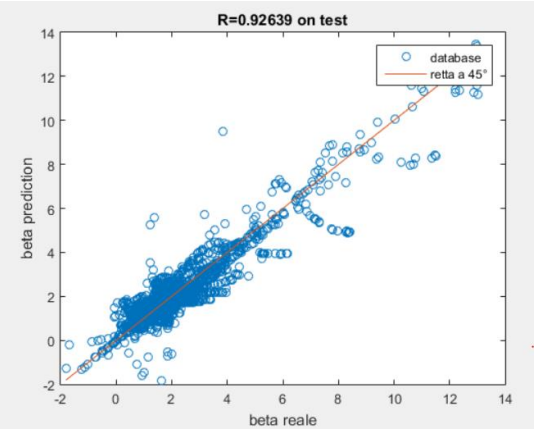
Crossover	Mutazione	Riproduzione	R^2_{train}	R^2_{test}	$R^2_{validation}$	Complessità
0.6	0.38	0.02	0.1543	0.1524	0.1598	25
0.7	0.28	0.02	0.2327	0.2437	0.2248	21
0.8	0.18	0.02	0.2151	0.2250	0.2244	15
0.9	0.08	0.02	0.1436	0.1439	0.1396	12

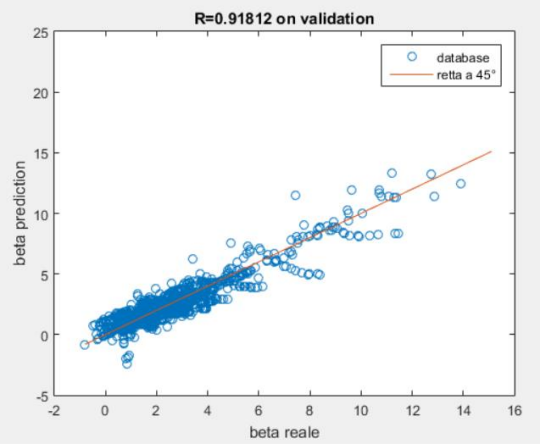
Per il caso iniziale si è deciso di testare le possibilità della GP utilizzando tutti gli input e le variabili genetiche sono state settate nel seguente modo:

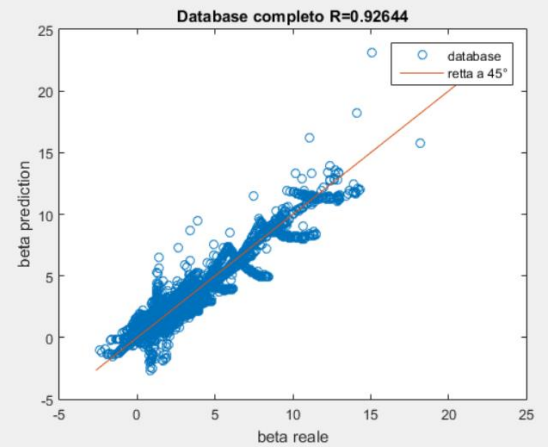
Popolazione	2500
Numero massimo di geni (per individuo)	10
Profondità massima albero	8
Dimensione del torneo	250
Tipo di selezione	0.3 Pareto e 0.7 Classico
Operazioni genetiche	0.7 Crossover; 0.28 Mutazione; 0.02 Riproduzione
Function set	{'times','minus','plus','rdivide','square','tanh', 'exp','log','sqrt','cube','negexp','abs'};
Terminal set	$\{ERC, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$

Come nel caso della rete neurale il livello di fitting è molto alto.









Una volta implementata la funzione nel codice CFD, quello che risulta è un andamento stabile del calcolo, a differenza di quello della rete neurale che, nel medesimo caso di otto input, si arrestava alla prima iterazione.

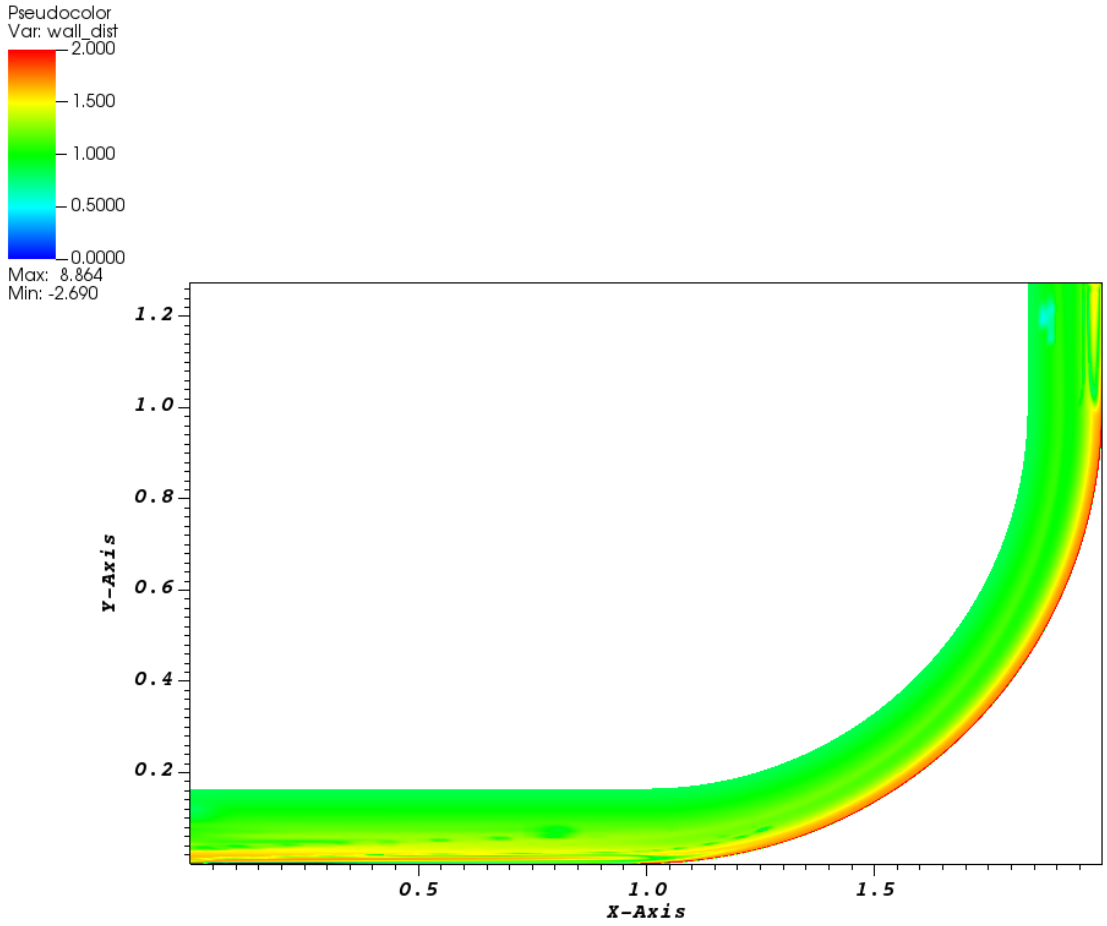


Figura 34 Primo passo temporale GP con 8 input

Dalla figura 34 si può notare come il campo iniziale non sia esattamente come quello ricavato dall'inversione di campo, in particolare differisce da quest'ultimo nelle sezioni di ingresso ed uscita, mentre presenta una buona approssimazione sul tratto curvilineo. Da notare come nel tratto di uscita la correzione del coefficiente beta non sembra risentire della vicinanza a parete mentre, al contrario, sulla sezione di ingresso sembra necessaria una modifica anche laddove l'effetto dello strato limite non dovrebbe esserci.

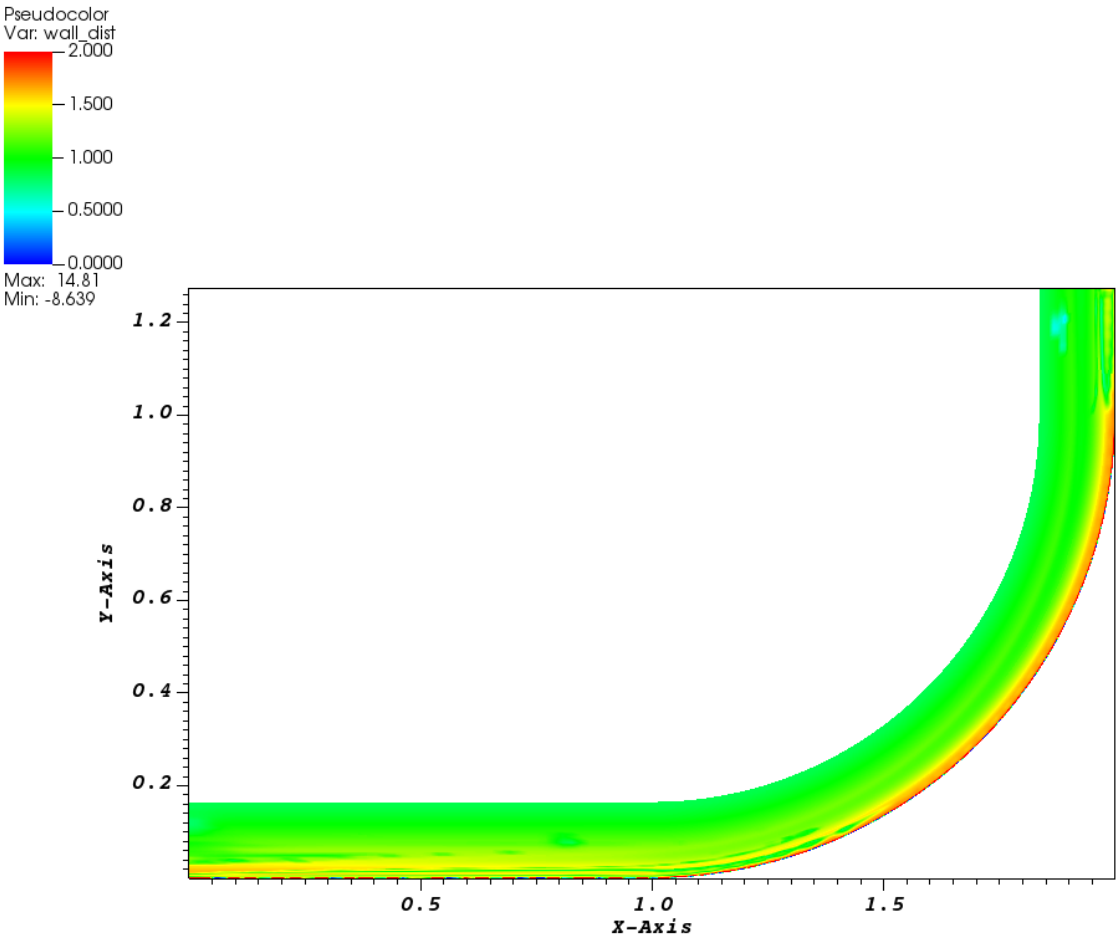


Figura 35 Soluzione della GP a 8 input dopo 700 iterazioni temporali

Dopo aver lasciato che il flusso evolvesse, il codice CFD si è arrestato dopo 700 iterazioni, mostrando un errore in una cella appartenente alla sezione di uscita. La soluzione è riportata in figura 35. In questa sezione infatti si può notare come il coefficiente di correzione risulti ormai inalterato, indicando quindi l'errore della GP. Dato che questo andamento discontinuo si era già presentato nel caso della rete neurale, si è deciso di sfruttare le conoscenze derivanti da essa per sviluppare la programmazione genetica; nello specifico si utilizzeranno le variabili aventi gradienti più grandi: x_2, x_5, x_7, x_8.

Come primo risultato è stata creata la seguente GP:

Popolazione	2000
Numero massimo di geni (per individuo)	10
Profondità massima albero	10
Dimensione del torneo	300
Tipo di selezione	0.3 Pareto e 0.7 Classico
Operazioni genetiche	0.7 Crossover; 0.28 Mutazione; 0.02 Riproduzione
Function set	{'times','minus','plus','rdivide','square','tanh', 'exp','log','sqrt','cube','negexp','abs'};
Terminal set	$\{ERC, x_2, x_5, x_7, x_8\}$

La soluzione trovata presenta ottimi valori di fitting

R^2_{train}	R^2_{test}	$R^2_{validation}$
0.776	0.762	0.7574

Per la soluzione CFD si riscontra il problema presentato nel caso della rete neurale ad otto input, ovvero la tendenza a fare la prima iterazione temporale per poi arrestarsi.

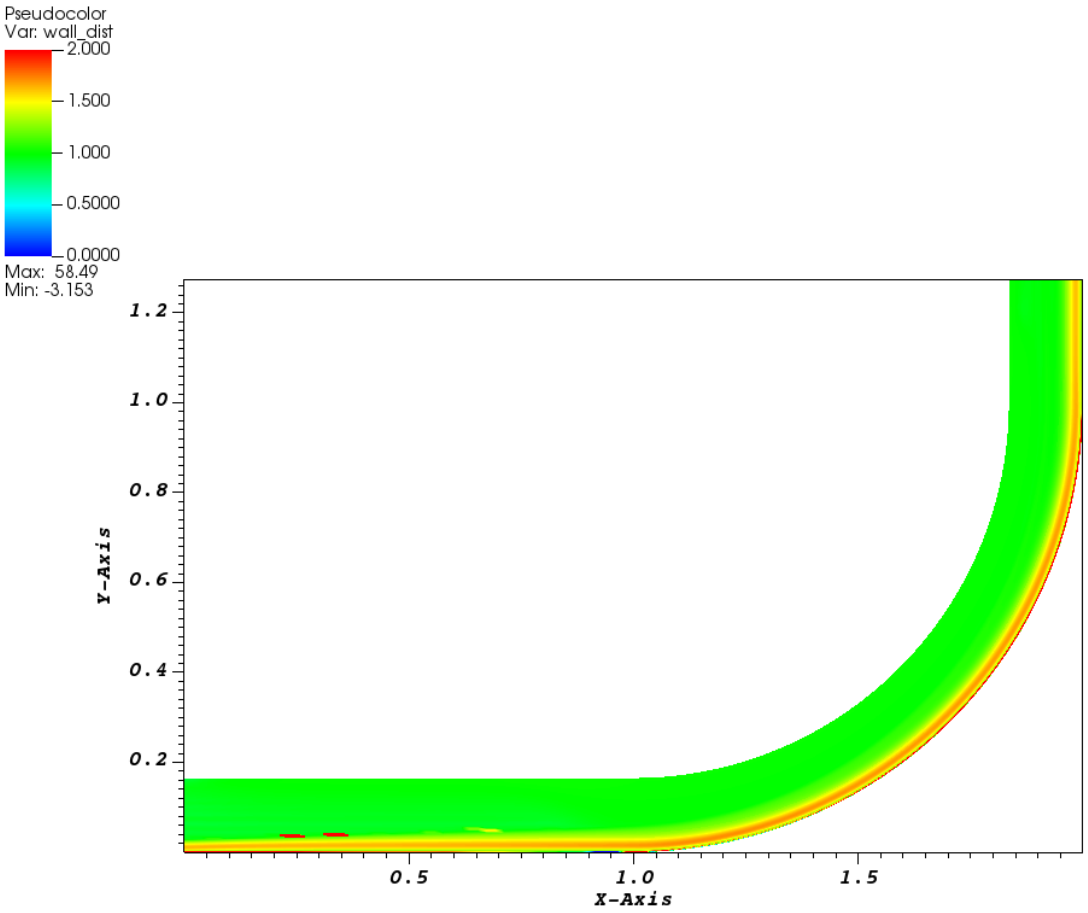


Figura 36 Primo passo GP 4 input

Si può fin da subito notare in figura 36 come nel tratto curvilineo la correzione del coefficiente beta sia sottostimata rispetto ai risultati ricavati dall'inversione di campo, tuttavia rispetto al caso otto input presenta una continuità maggiormente definita. I problemi principali si hanno nella sezione di ingresso, in particolare nella zona verde segnata nella figura sopra si presentano delle forti variazioni che non sono giustificate dal punto di vista fisico. La prossima funzione GP è stata studiata per cercare di eliminare queste zone che sembrano essere dovute ad un overfitting dell'algoritmo, in quanto si ricordi che tutte le fonti di disturbo sono state eliminate.

La nuova GP presenta:

Popolazione	2000
Numero massimo di geni (per individuo)	8
Profondità massima albero	8
Dimensione del torneo	300

Tipo di selezione	0.3 Pareto e 0.7 Classico
Operazioni genetiche	0.7 Crossover; 0.28 Mutazione; 0.02 Riproduzione
Function set	{'times','minus','plus','rdivide','square','tanh', 'exp','cube','negexp','abs'};
Terminal set	$\{ERC, x_2, x_5, x_7, x_8\}$

Rispetto al caso precedente si sono abbassati i parametri relativi alla costruzione dell'albero/gene di cui si compongono gli individui, in quanto essi sono fortemente correlati al fenomeno di overfitting. Inoltre sono state tolte dal Function set tutte quelle funzioni che tendono a restituire numeri complessi. Quest'ultima osservazione deriva dall'analisi post-GP della funzione creata da GPTIPS che presentava, nel caso precedente, molti valori 'real'.

I valori di fitting sono superiori rispetto al caso precedente

R^2_{train}	R^2_{test}	$R^2_{validation}$
0.788	0.801	0.780

In termini CFD, nonostante la soluzione si arresti ancora alla prima iterazione (figura 37), si può notare come lo scopo prefissato si sia raggiunto: non ci sono più problemi nel tratto iniziale. Tuttavia permane ancora lungo tutto il tratto curvilineo una sottostima del coefficiente di correzione e nella sezione di uscita si ha non solo una piccola variazione di tale coefficiente, ma anche una zona dove teoricamente tale parametro dovrebbe essere inalterato, mentre dal punto di vista sperimentale sembra dover essere dimezzato (tratto azzurro).

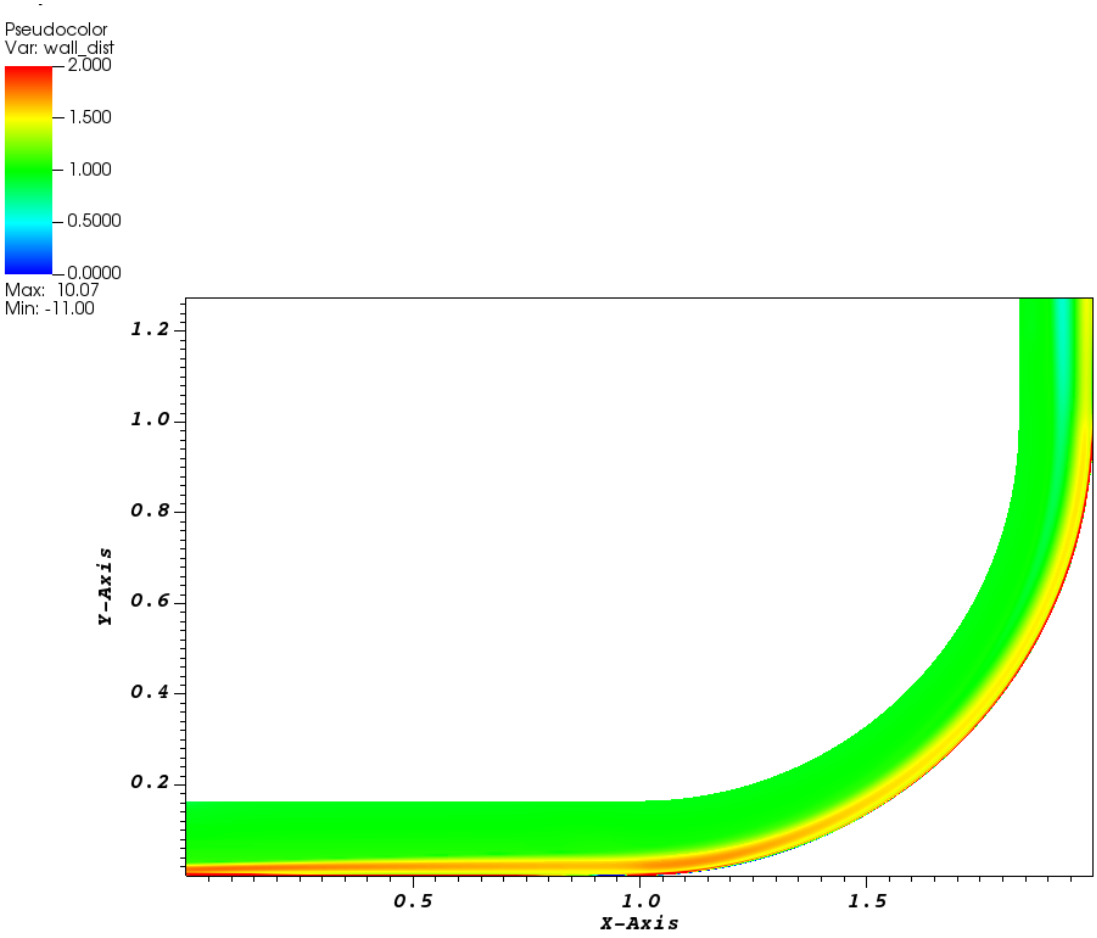


Figura 37 CFD caso GP_modificato 4 input

Bibliografia

Ferrero, A. (2014). Tesi di dottorato. Computational fluid dynamics for aerospace propulsion system: an approch based on discontinuous finite elements. Politecnico di Torino.

Ferrero, A., Iollo, A., & Larocca, F. (2020, aprile 15). Field inversion for data-augmented RANS modelling in turbomachinery flows. Computers and Fluids (201).

PROSPETTIVE FUTURE

Dal lavoro emerso si è potuto notare come l'applicazione del paradigma inversione di campo e machine learning (FIML) non sia di così semplice applicazione. In particolare tra i due metodi analizzati si è notato come la rete neurale, pur essendo una black box, è in grado di propagare in maniera continua e migliore la soluzione CFD. D'altra parte la programmazione genetica è in grado di fornirci un'equazione matematica chiara ed esplicativa della distribuzione del parametro beta in funzione delle variabili fluidodinamiche. In questo lavoro si è svolto uno studio preliminare in cui si è messa in luce la difficoltà di ottenere una descrizione robusta del termine di correzione e sono stati resi noti i parametri su cui lavorare per sviluppare ulteriormente il modello. Inoltre bisogna ricordare come il processo di inversione di campo è stato operato mediante la modifica del termine di produzione di SA e si è utilizzato il solo coefficiente di attrito a parete per verificare la correttezza dei risultati. Nulla vieta di generalizzare la funzione goal usata durante l'inversione di campo combinando assieme gli errori relativi a diverse grandezze fisiche di cui si ha un riscontro sperimentale. Secondo un ragionamento logico, maggiore è il numero di parametri introdotti nel modello e il numero di variabili fluidodinamiche monitorate, tanto più risulterà migliorata l'accuratezza del metodo modificato rispetto a quello di partenza. Si noti come la procedura utilizzata sia generica e applicabile a qualunque modello RANS, non solo a SA. Nell'aumentare del numero di parametri ci si aspetta, oltre che un aumento dell'accuratezza, un aumento nella complessità di gestione dei risultati derivanti dalla loro implementazione, con la conseguente necessità di gestione delle interdipendenze tra questi fattori; ovvero bisognerà considerare quelli che effettivamente migliorano il modello. Un altro possibile campo di intervento riguarda la scelta delle variabili per esprimere i fattori correttivi. Il set di input selezionato deriva non solo dall'analisi delle dipendenze reciproche, ma anche dai risultati in termini di fitting dell'algoritmo e stabilità del codice CFD. Si intuisce quindi come ci sia una forte correlazione tra la relazione esplicita del beta in funzione del set di features impostato ed il problema trattato. Resta dunque da determinare la reale efficacia di questo fattore beta se applicato ad un test case diverso come ad esempio una turbomacchina. Sempre legato ai valori di input per l'algoritmo di machine learning, un problema riscontrato è stato su come trattare questi valori, in quanto ciascuno possiede una sua magnitudine che non necessariamente coincide con quello degli altri. Inizialmente si era pensato di considerare il logaritmo in base 10 degli input di scala

superiore, ma una volta realizzata la rete neurale/programmazione genetica, le quali mostravano dei fitting molto elevati, il codice CFD risultava instabile. Si è quindi scelto di non operare con i logaritmi accettando le scale di grandezza connaturate nei dati di ingresso. Ancora una volta si tende a sottolineare come questo ragionamento valga nel caso del condotto bidimensionale curvo in esame, ma potrebbe non essere applicabile su una geometria diversa.

Per quanto riguarda ricerche e miglioramenti legati alla seconda parte del paradigma FIML, quella legata al machine learning, oltre alla possibilità di indagare altri metodi di apprendimento automatico, sono state avanzate nuove teorie per migliorare la convergenza e stabilità degli algoritmi. Tra i più innovativi cito il lavoro di Holland (Holland, Baeder, & Duraisamy, 2019) che nella sua tesi ha sviluppato un modello di paradigma definito FILM-Embedded, grazie al quale è possibile sfruttare più parametrizzazioni diverse in contemporanea, con tempi di calcoli ridotti rispetto alla trattazione classica. Tale modello incorpora il processo di inversione, basato sulla minimizzazione della funzione obiettivo, e di retro-propagazione dell'errore, per il calcolo dei pesi della rete neurale, direttamente nel solutore CFD. Tra i metodi proposti per migliorare la generalizzazione di un dato modello su test case differenti, quello con risultati più promettenti è quello esposto in (Ling, Jones, & Templeton, May 2016) . Il problema trattato è la modellizzazione delle differenze nella rappresentazione del tensore degli sforzi di Reynolds rispetto a quelli ricavabili, per ogni test case trattato, da simulazioni DNS. Ling et al. hanno sviluppato una tecnica di apprendimento automatico in cui il modello sviluppato soddisfa automaticamente l'invarianza galileiana[3]: simili concetti potrebbero essere efficacemente usati anche nei metodi utilizzati in questa tesi.

[3] Una funzione si dice essere invariante galileiana rispetto ad una data trasformazione se il suo valore coincide con quello che si ottiene applicando la trasformazione ai dati di ingresso. Nella meccanica dei fluidi, l'invarianza galileiana si ottiene quando le quantità fluidodinamiche sono invarianti rispetto a rotazione, riflessione e traslazione del sistema di riferimento.

Bibliografia

Ling, J., Jones, R., & Templeton, J. A. (May 2016, May). Machine learning strategies for system with invariance properties. *Journal of Computational Physics 318*.