

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

DIPARTIMENTO DI INGEGNERIA ENZO FERRARI DI MODENA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA DEL VEICOLO

SVILUPPO DI UN CODICE NUMERICO PER LA RISOLUZIONE DI PROBLEMI DI OTTIMIZZAZIONE DI FORMA TRAMITE RETI NEURALI

Candidato: Simone Tundo

Relatore: Prof. Francesco Mezzadri

A.A. 2025/2026

ABSTRACT

I problemi di ottimizzazione di forma consistono nella determinazione della forma da attribuire ad un oggetto in modo da renderne ottimale una qualche caratteristica. Essi possono pertanto essere un importante supporto alla progettazione ingegneristica. Tuttavia, la risoluzione di un problema di ottimizzazione di forma può essere computazionalmente onerosa. È quindi importante studiare tecniche numeriche efficienti di risoluzione. Recentemente, questo ha incluso anche l'applicazione di algoritmi basati su reti neurali. Scopo di questo elaborato è quindi quello di indagare la risoluzione di problemi di ottimizzazione di forma in ambito meccanico tramite reti neurali del tipo "Physics Informed Neural Networks" (PINN). Nella tecnica analizzata, la rete viene addestrata per prevedere la soluzione dell'equazione dell'elasticità lineare su domini di diversa forma, parametrizzati su un dominio comune tramite NURBS. Le previsioni della rete sono poi usate all'interno di un algoritmo convenzionale di ottimizzazione di forma al posto della risoluzione dell'equazione dell'elasticità, evitando la risoluzione esplicita di un problema differenziale ad ogni iterazione. Tali passaggi sono stati svolti tramite un codice numerico in linguaggio Python. A differenza di lavori esistenti che impiegano una tecnica analoga, l'utilizzo di reti PINN fa sì che l'addestramento della rete non sia basato unicamente su dei dati, ma che contenga anche delle informazioni sulla fisica del problema (fornite, in particolare, dal residuo dell'equazione differenziale). Oltre all'elaborazione ed alla validazione del codice, un obiettivo della tesi è quindi quello di analizzare le differenze tra le convenzionali reti NN (completamente "data-based") e le reti PINN al variare della profondità della rete stessa nell'ambito di un problema test che consiste nell'ottimizzazione della forma di una piastra forata. Ciò consente infine anche di valutare quale rete costituisca un efficiente compromesso tra accuratezza e il tempo speso per l'addestramento.

INDICE

INTRODUZIONE 1

BACKGROUND 4

2.1. Generalità sulle reti neurali 4

2.1.1. Obiettivo del problema 5

2.1.2. Algoritmo di backpropagation 7

2.1.3. Physics-Informed Neural Network 8

2.2. Contesto meccanico dell'ottimizzazione di forma 12

2.3. Parametrizzazione mediante NURBS 17

2.3.1. Algoritmo NURBS 18

FORMULAZIONE DEL PROBLEMA 20

ESPERIMENTI NUMERICI 23

4.1. Addestramento 23

4.2. Validazione 27

4.3. Ottimizzazione di forma 31

CONCLUSIONI 36

APPENDICE 37

REFERENZE 43

INTRODUZIONE

In generale, si può definire un problema di ottimizzazione come un quesito matematico che richiede di determinare un punto ottimale (cioè un minimo o un massimo) di una qualche funzione obiettivo assegnata, spesso soggetta a vincoli, che possono essere di natura algebrica o differenziale (come, ad esempio, equazioni differenziali ordinarie o alle derivate parziali). Questa particolare branca della matematica può essere applicata ad una vasta serie di applicazioni, dall'ingegneria alla medicina, dalla finanza al settore energetico. Esistono diverse tipologie di problemi di ottimizzazione. Questo elaborato riguarda, in particolare, quelli che sono chiamati *problemi di ottimizzazione di forma* **[1]**, nei quali si cerca di determinare una geometria ottimale rispetto ad un certo obiettivo sotto un determinato insieme di vincoli. Ad esempio, in un problema di questo tipo si potrebbe quindi ricercare la forma che si deforma meno (obiettivo) quando sottoposta ad un certo insieme di condizioni di carico (vincoli). La risoluzione di questi problemi può essere ottenuta in vari modi. Recentemente, questo ha incluso l'impiego di reti neurali, cioè architetture matematiche strutturate in livelli (*layers*), ideate per approssimare la mappa tra vettori di input e output mediante l'ottimizzazione dei suoi parametri. Tale fase di ottimizzazione è comunemente chiamata *addestramento* o *training* e richiede di conoscere sia input che output per un dataset di addestramento, sulla cui base la rete "apprenderà" la relazione che li lega. Al termine dell'addestramento, la rete sarà in grado di effettuare "previsioni" della soluzione anche per input che non ha mai visto. Tuttavia, l'accuratezza di tali previsioni dipenderà dalla quantità (e qualità) di dati di training e dall'addestramento stesso. Uno svantaggio nell'impiego di una rete neurale è quindi proprio legato al tempo impiegato e all'elevata potenza di calcolo richiesti per generare i dati ed addestrare la rete. Il vantaggio fondamentale è invece che, una volta che la rete è addestrata, il suo utilizzo richiede un tempo trascurabile: gli output corrispondenti ad un determinato input possono essere generati in pochi secondi. Questo approccio ha quindi un elevato potenziale, in quanto se la rete risultasse ben addestrata, allora essa sarebbe in grado di risolvere anche problemi "simili" in un tempo estremamente contenuto. Ciò è attraente soprattutto in ambiti in cui sia necessario risolvere più volte una serie di problemi simili. Questo può essere, ad esempio, il caso delle procedure iterative, come quelle su cui si basa la risoluzione dei problemi di ottimizzazione di forma. Infatti, l'ottimizzazione di forma è generalmente effettuata tramite procedure che, partendo da una forma iniziale, la modificano cercando di avvicinarsi alla soluzione. Ad ogni iterazione, è necessario caratterizzare le proprietà della forma di volta in volta considerata. Matematicamente, ciò consiste nel risolvere l'equazione differenziale che descrive la fisica del problema di interesse (ad esempio, l'equazione dell'elasticità per problemi meccanici come quello di minimizzazione delle deformazioni menzionato sopra).

In questo elaborato viene trattato proprio un problema di ottimizzazione di forma di natura meccanica: determinare la geometria ottimale di una piastra forata, sottoposta ad una distribuzione di carico esterno costante. Reti neurali di diverso tipo sono addestrate per prevedere la soluzione dell'equazione dell'elasticità lineare su domini di diversa forma. L'utilizzo di reti diverse permetterà di individuare quale risulta essere la "migliore" per questa tipologia di problema. Vengono confrontate, in particolare, reti completamente data-

based (neural networks, NN) con le cosiddette physics-informed neural networks (PINN), introdotte da **Raissi M et al [2]**. Nell'analisi si considereranno reti di diversa profondità addestrate con diversi numeri di geometrie di training. La differenza tra le due tipologie di rete è strutturale: infatti, mentre NN dipende esclusivamente dal data-set di addestramento, una PINN è addestrata a cercare la minimizzazione del residuo dell'equazione differenziale, fornendo così informazioni anche sulla fisica. Per via di questa differenza, è importante caratterizzare le due reti. Verrà così effettuata una valutazione della precisione di entrambe, in modo da verificare se l'errore commesso da una PINN risulta inferiore rispetto ad una rete NN. Dopodiché, si analizzano come le reti ottimizzano il profilo circolare della piastra forata, sia visualizzando il profilo ottimizzato rispetto alla soluzione analitica, sia impiegando un grafico polare utile per determinare l'errore massimo commesso dalla rete nell'ottimizzare la rete rispetto al profilo analitico. Però per ottenere ciò, il modello non modifica fisicamente il profilo, ma bensì cambia i parametri per ogni punto senza modificarne la posizione, applicando un dominio parametrico. La parametrizzazione impiegata sfrutta una particolare tipologia di curve, definite *Non Uniform Rational Basis-Splines* (NURBS), ovvero una rappresentazione matematica impiegata per creare elementi geometrici. Vengono impiegate anche nella modellazione solida CAD. L'obbiettivo finale è quello di indicare quale struttura fornisce il miglior compromesso tra il tempo impiegato per l'addestramento e il risultato del problema di ottimizzazione. In questo modo, si fornisce una rete neurale con una determinata struttura che può essere impiegata per future sperimentazioni.

La tesi è strutturata come segue: Sezione 2 riporta una serie di risultati noti di letteratura, su cui si fondano le metodologie applicate nel seguito. Tale sezione costituisce anche un riassunto dello stato dell'arte su diverse procedure pertinenti, come quelle connesse all'ottimizzazione di forma. Sezione 3 presenta poi in dettaglio il problema di interesse per questa tesi e la metodologia utilizzata per risolverlo. Gli esperimenti numerici sulla piastra forata sono poi svolti e commentati in Sezione 4. Infine, conclusioni di questo lavoro sono riportate in Sezione 5. Il codice completo delle reti usate è infine riportato in Appendice. La presente sezione si chiude invece con una tabella riassuntiva della principale notazione utilizzata.

Simbolo	Definizione	Paragrafo
P	Punto di controllo	(2.4.1)
w	Peso	(2), (2.1), (2.1.1)
b	Bias	(2)
(x, y)	Coordinate del dominio fisico	(2.2), (2.4), (3)
(ξ, η)	Coordinate del dominio parametrico	(2.2), (2.4), (2.4.1), (3)
Ω	Dominio fisico	(2.2), (2.3), (3)
$\widehat{\Omega}$	Dominio parametrico	(2.2), (2.3), (3)
$\mathcal{L}$	Loss Function	(2.1), (2.2), (2.3), (3), (4.2)
E	Modulo di Young	(2.2), (2.3), (3), (4.2)
ν	Coefficiente di Poisson	(2.2), (2.3), (3), (4.2)
λ, μ	Parametri di Lamé	(2.3)
u	Spostamento	(2.1), (2.2), (2.3), (3), (4.2)
σ	Tensione	(2.2), (2.3), (3), (4.2)
ε	Deformazione	(2.2), (3), (4.2)
J	Matrice Jacobiana	(2.2), (3)
$\mathcal{L}(\Omega, u, p)$	Funzione Lagrangiana	(2.3)
$\mathcal{T}$	Errore di addestramento	(2.1.1)
net	Eccitazione	(2.1.1)
γ	Learning rate	(2.1.1)
B	B-spline	(2.4.1)
V	Volume	(2.3), (4.3)
c	Compliance	(4.3)
e	Errore	(4.2), (4.3)
r	Raggio	(4.3)

Tabella 1 - Notazione

BACKGROUND

2.1. Generalità sulle reti neurali

Una Rete Neurale (Neural Network NN) può essere descritta come uno schema di neuroni, suddivisi in layer, dove in funzione di determinati dati in ingresso la rete fornisce dei dati in uscita (*Figura 1*). I neuroni sono gli elementi fondamentali nei quali avvengono i calcoli computazionali. Tra di loro i neuroni sono collegati tramite relazioni matematiche: queste ultime sono caratterizzate da somme pesate. Ad esempio, assumendo tre variabili in ingresso x_1, x_2 e x_3, il valore del primo neurone del layer vale

$$a_1^{(1)} = w_{11}^{(1)} x_1 \cdot w_{21}^{(1)} x_2 \cdot w_{31}^{(1)} x_3 = \sum_{i=1}^{3} w_{i1}^{(1)} x_i \ . \tag{2.1}$$

A pedice si indica il rispettivo numero del neurone, mentre ad apice il rispettivo layer a cui è associato quel neurone. Generalizzando, nel caso di n input la relazione può essere scritta come

$$a_1^{(1)} = \sum_{i=1}^{n} w_{i1}^{(1)} x_i \ , \tag{2.2}$$

invece, esprimendo la relazione per il j -esimo neurone del layer

$$a_j^{(1)} = \sum_{i=1}^{n} w_{ij}^{(1)} x_j \quad per\ j = 1, \dots, n_1 \ . \tag{2.3}$$

Dunque, le relazioni tra i vari neuroni sono caratterizzate dai pesi, che hanno come obiettivo quello di istaurare un particolare legame tra i neuroni della rete. In realtà, il modello presenta altri due elementi fondamentali: i biases e le funzioni di attivazione. Un bias è un elemento dotato di peso unitario che si somma al'*Eq. 2.3*

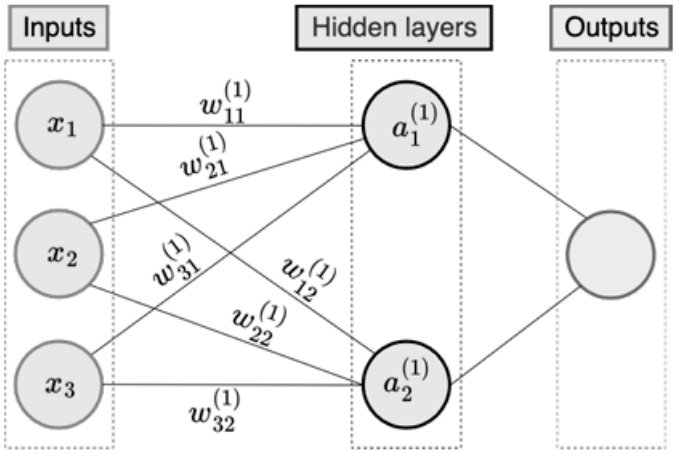


Figura 1 - Neural Networks.

$$a_j^{(1)} = \sum_{i=1}^{n} w_{ij}^{(1)} x_j + b^{(1)} \ \ per\ j = 1, \dots, n_1 \ , \tag{2.4}$$

mentre le funzioni di attivazione hanno come obiettivo quello di aggiungere complessità al modello

$$a_j^{(1)} = f\left(\sum_{i=1}^{n} w_{ij}^{(1)} x_j + b^{(1)} \right) \ \ per\ j = 1, \dots, n_1 \ . \tag{2.5}$$

Esistono diversi tipi di funzioni di attivazione, come la Rectified Linear Unit (ReLU)

$$max\{0, x\} \ , \tag{2.6}$$

che è uguale alla somma pesata se quest'ultima è positiva, altrimenti è nulla. Un'altra funzione è la tangente iperbolica

$$tanh\, x \ , \tag{2.7}$$

che tende a "schiacciare" la somma pesata in un intervallo compreso tra -1 e 1. Un'ultima relazione è la sigmoide

$$\frac{1}{1 + e^{-x}} \ , \tag{2.8}$$

caso simile alla tangente iperbolica ma in questo caso l'intervallo di riferimento è compreso tra 0 e 1. Le relazioni riportate nel caso precedente sono valide solo per un layer di neuroni: nel caso avessimo molteplici strati di neuroni, allora indicando il k-esimo layer si può scrivere la relazione generale

$$\begin{aligned} a_j^{(1)} &= f\left(\sum_{i=1}^{n_1} w_{ij}^{(1)} x_j + b_j^{(1)} \right) \ \ per\ j = 1, \dots, n_1 \ , \\ a_j^{(k)} &= f\left(\sum_{i=1}^{n_k} w_{ij}^{(k)} a_i^{(k-1)} + b_j^{(k)} \right) \ \ per\ j = 1, \dots, n_k \ . \end{aligned} \tag{2.9}$$

2.1.1. Obiettivo del problema

Dato un problema e i vincoli collegati ad esso, l'obiettivo è quello di minimizzare la Loss Function

$$\min_{w} \mathcal{L}_{NN} \ , \tag{2.10}$$

dove la w indica i pesi della funzione loss che saranno poi aggiornati tramite l'algoritmo di backpropagation. La funzione loss viene impiegata per capire se il risultato fornito dal modello analitico si avvicina alla soluzione desiderata. Prima di descrivere la relazione, è necessario indicare sia la condizione iniziale, ovvero il valore assunto dalla soluzione al tempo zero, sia la condizione al contorno, ovvero il valore assunto dalla soluzione vincolata al bordo del dominio di definizione $\partial\Omega$. La funzione loss per NN può essere caratterizzata dal contributo di tre residui

$$\mathcal{L}_{NN} = |u - u^*|_{\Omega} + |u - u_0^*|_{t=0} + |u - u^*|_{\partial\Omega} \ ,$$

$$|u - u^*| = \frac{1}{n}\sum_{i=1}^{n}(u_i - u_i^*)^2 \ . \tag{2.11}$$

I termini $|u - u_0^*|_{t=0}$ e $|u - u^*|_{\partial\Omega}$ rappresentano, rispettivamente, i residui della condizione iniziale e al contorno. $|u - u^*|_{\Omega}$ mette in relazione la soluzione calcolata nell'iterazione rispetto alla soluzione desiderata ed è basata sulle conoscenze di alcuni punti interni del dominio. Naturalmente, non tutti questi punti sono noti sempre. Ad esempio, se una NN è utilizzata per risolvere un problema differenziale in avanti, i dati disponibili saranno solamente quelli al contorno e all'inizio. Al contrario, nei problemi inversi la soluzione dell'equazione differenziale potrebbe essere nota in alcuni punti interno al dominio ma potenzialmente non al contorno. Questa formulazione della loss function può risultare semplice dal punto di vista analitico, ma potrebbe presentare delle limitazioni dal punto di vista numerico: infatti, vi sono della difficoltà ben note che spesso possono influenzare affidabilità e accuratezza del modello, specialmente per problemi applicati oppure complessi. Per questo vengono inseriti dei pesi che moltiplicano i termini della loss function: si effettua questa modifica perché il residuo della equazione differenziale e i termini relativi alla condizione iniziale e al contorno evolvono in modo diverso, facendo sì che il problema di ottimizzazione sia "sbilanciato". In questo contesto, un fenomeno ben noto è chiamato Neural Tangent Kernel (NTK): in una banda limitata, le reti completamente connesse con attivazioni standard si comportano come macchine kernel il cui NTK associato ha autovalori che decadono molto rapidamente con la frequenza. Questo significa che, durante la discesa del gradiente, le componenti ad alta frequenza della soluzione sono esponenzialmente più lenti delle componenti a bassa frequenza. In una PINN la situazione è peggiore: in aggiunta del residuo dell'equazione differenziale, esso tipicamente presenta derivate di alto ordine, le quali amplificano i modi ad alta frequenza della discesa del gradiente. Poiché le condizioni iniziale e al contorno sono spesso più elevate in ampiezza, l'ottimizzatore è costretto a soddisfarli prima e trascorre quasi tutto il tempo di addestramento ad adattare la parte a bassa frequenza della soluzione, lasciando quasi nessuna capacità di ridurre il contenuto ad alta frequenza richiesto dall'operatore differenziale. Il risultato è che il network soddisfa le condizioni iniziali e al contorno in modo accettabile producendo però una soluzione che viola l'equazione differenziale in regioni dove si necessita una scala fine. Le strategie di ponderazione adattativa che tengono esplicitamente conto dello spettro NTK possono essere utilizzate per mitigare questo problema. Considerando, ad esempio, la funzione loss (*Eq. 2.11*), ogni termine è ponderato tramite un insieme di pesi

$$\mathcal{L}_{NN} = w_1|u - u^*|_{\Omega} + w_3|u - u_0^*|_{t=0} + w_4|u - u^*|_{\partial\Omega} \ . \tag{2.12}$$

Il problema consiste nel far sì che tutti i termini abbiano più o meno lo stesso peso nell'ottimizzazione. A questo scopo, dopo un numero prescritto di iterazioni, i pesi vengono aggiornati per adattarsi a come i termini cambiano durante l'addestramento. In particolare, viene calcolato il gradiente di ciascun termine individuale rispetto ai parametri della rete. Successivamente, il valore assoluto massimo di questi gradienti viene utilizzato per impostare i pesi di ciascun termine (in modo che i termini con gradienti piccoli ricevano pesi molto più

grandi). Questo bilancia i termini e supporta l'addestramento. L'aggiornamento viene solitamente eseguito solo dopo un certo numero di iterazioni perché può risultare piuttosto costoso dal punto di vista computazionale. Per una PINN, il residuo della PDE richiede il calcolo ripetuto di derivate di ordine superiore della rete neurale tramite differenziazione automatica. Quando la soluzione contiene gradienti accentuati, strati limite, shock, componenti ad alta frequenza o è dominata dall'advezione, lo spettro del gradiente della perdita diventa molto rigido. L'ottimizzatore, quindi, fa fatica a bilanciare i diversi contributi e spesso converge a soluzioni che ignorano completamente la fisica corretta, oppure si blocca semplicemente in minimi locali di scarsa qualità. Problemi con molteplici scale temporali o spaziali (ad esempio reazione-diffusione con tassi di reazione molto diversi, flussi ad alto numero di Reynolds, propagazione di onde su tempi lunghi) sono particolarmente patologici, e persino alcuni problemi di benchmark "semplici" che sono banali per i metodi classici agli elementi finiti, a volumi finiti o spettrali rimangono essenzialmente irrisolvibili con NN/PINN standard senza adattamenti speciali; il metodo caratterizzato dall'*Eq. 2.11* è diffuso sia per la sua semplicità di implementazione, sia per la qualità dei risultati ottenibili. L'algoritmo backpropagation modifica i pesi in base alle "informazioni" date dall'ottimizzatore, in modo da ridurre l'obiettivo tramite la modifica dei pesi. Si tratta di ricercare la combinazione di pesi che minimizza la funzione errore; questa sarà considerata la soluzione del problema di apprendimento. Si possono usare diversi tipi di funzione di attivazione: l'unica condizione a cui devono sottostare è quella di essere differenziabili in ogni punto e di avere sempre la derivata non nulla. Questo secondo aspetto è dovuto al fatto che il metodo di ricerca del minimo della funzione errore è il metodo del gradiente e quindi si capisce che avere una funzione attivazione che in qualche punto abbia derivata nulla può causare dei gravi problemi all'algoritmo.

2.1.2. Algoritmo di backpropagation

Il primo step consiste nel valutare l'errore di addestramento $\mathcal{T}$. Il vettore n-dimensionale degli input $o = (o_1, \dots, o_n)$ viene trasformato nel vettore $oi = (o_1, \dots, o_n, 1)$. L'eccitazione $net_j^{(k)}$ del j-esimo nodo del piano k-esimo è uguale a

$$net_j^{(k)} = \sum_{i=1}^{n+1} w_{ij}^{(k)} oi_i \quad \begin{matrix} per\ k = 1, \dots, n \\ per\ j = 1, \dots, n_k \end{matrix} \quad . \tag{2.13}$$

Iterando questo procedimento per tutti gli m output è possibile definire l'errore per l'addestramento del modello i

$$\mathcal{T} = \frac{1}{2} \sum_{i=1}^{m} \left(o_i^{(n)} - y_i^* \right)^2 \quad . \tag{2.14}$$

Ovviamente $\mathcal{T} > 0$ e quindi bisogna modificare il valore dei pesi. Per far ciò sono necessari ulteriori passi. Nel prossimo step si procede a trovare il valore finale del gradiente della funzione errore rispetto ai pesi dell'ultimo strato di neuroni

$$\begin{cases} \dfrac{\partial \mathcal{T}}{\partial w_{ij}^{(n)}} = \dfrac{\partial \mathcal{T}}{\partial o_j^{(n)}} \dfrac{\partial o_j^{(n)}}{\partial net_j^{(n)}} \dfrac{\partial net_j^{(n)}}{\partial w_{ij}^{(n)}} = \left(o_j^{(n)} - y_i^*\right) o_j^{(n)} \left(1 - o_j^{(n)}\right) o_i^{(n-1)} \\ \delta_j^{(n)} = \left(o_j^{(n)} - y_i^*\right) o_j^{(n)} \left(1 - o_j^{(n)}\right) \end{cases} . \tag{2.15}$$

$$\Downarrow$$

$$\frac{\partial \mathcal{T}}{\partial w_{ij}^{(n)}} = o_i^{(n-1)} \delta_j^{(n)}$$

Ultimo step è quello di determinare il gradiente della funzione errore anche per gli strati interni

$$\begin{cases} \dfrac{\partial \mathcal{T}}{\partial w_{ij}^{(k)}} = \dfrac{\partial \mathcal{T}}{\partial o_j^{(k)}} \dfrac{\partial o_j^{(k)}}{\partial net_j^{(k)}} \dfrac{\partial net_j^{(k)}}{\partial w_{ij}^{(k)}} = \left(o_j^{(k)} - y_i^*\right) o_j^{(k)} \left(1 - o_j^{(k)}\right) o_i^{(k-1)} \\ \delta_j^{(k)} = \left(o_j^{(k)} - y_i^*\right) o_j^{(k)} \left(1 - o_j^{(k)}\right) \end{cases} \quad \begin{matrix} per\ k = 1, \dots, n-1 \\ per\ j = 1, \dots, n_k \end{matrix} . \tag{2.16}$$

$$\Downarrow$$

$$\frac{\partial \mathcal{T}}{\partial w_{ij}^{(k)}} = o_i^{(k-1)} \delta_j^{(k)}$$

Determinati i gradienti per ogni strato, il modello si aggiorna andando a modificare i pesi, in modo da trovare il minimo della funzione. La maniera più semplice per effettuare ciò è la seguente

$$\Delta w_{ij}^{(k)} = \gamma o_i^{(k-1)} \delta_j^{(k)} \quad \begin{matrix} per\ k = 1, \dots, n \\ per\ i, j = 1, \dots, n_k \end{matrix} , \tag{2.17}$$

con γ definito learning rate e di solito viene preso costante (determinato per via empirica).

2.1.3. Physics-Informed Neural Network

Una Physics-Informed Neural Network (PINN) è una particolare rete impiegata per la risoluzione di equazioni differenziali. Qui considereremo in particolare il caso di equazione differenziale alle derivate parziali (PDE), descritta nella forma

$$\mathcal{D}u = b \ , \tag{2.18}$$

dove $\mathcal{D}$ denota un operatore differenziale che agisce sulla distribuzione di u in un qualche dominio Ω con contorno $\partial\Omega$. La soluzione dell'equazione differenziale può essere modellata tramite rete neurale per problemi diretti o inversi. Può essere impiegata per vari scopi:

- Risoluzione delle PDE: si assume che la soluzione sia nota al contorno del dominio di definizione.
- Creare un modello surrogato della soluzione: si assume di avere una nuvola di punti dove la soluzione della PDE è nota (ad esempio, ottenuti attraverso una osservazione sperimentale). Usando una PINN, è possibile prevedere la soluzione in punti differenti del dominio, dopo un accurato addestramento e parametrizzazione della rete. Oppure similmente, nota la soluzione di una PDE parametri-dipendente, allenando una PINN con questo problema è possibile sviluppare un modello che sia in grado di

effettuare delle previsioni sulla soluzione a fronte di una variazione dei parametri. L'utilizzo di reti PINN come modelli surrogati nell'ambito dell'elasticità lineare è stato particolarmente indagato da **Haghighat E et al [3]**.

- Per queste loro caratteristiche, le reti PINN sono state recentemente usate in diversi problemi differenziali, tra cui le equazioni di Navier-Stokes **[4]**, l'equazione di Reynolds **[5]**, high speed flow **[6]** e l'equazione di Chen-Lee-Liu **[7]**.

Il compito fondamentale dell'addestramento è quello di andare ad ottenere i pesi che andranno a caratterizzare la rete neurale. Ottenuti tali parametri, la rete neurale può essere sottoposta alla convalida. Le previsioni, comunque, possono essere accurate solo a fronte di un allenamento ottenuto con un soddisfacente numero di dati di allenamento. Il training di PINN può essere schematizzato nel seguente modo:

1. Inizializzare il network mediante pesi casuali.
2. Inserimento dei dati in input per i quali si conosce già la soluzione.
3. Il network produrrà un output che, in generale, è lontano dalla soluzione attuale.
4. Definizione di una misura quanto lontana è la previsione della PINN rispetto alla soluzione attuale. Questa misura si effettua tramite una funzione definita loss function. Maggiore è la distanza tra le due soluzioni, maggiore sarà il valore della funzione loss. Dunque, affinché si ottenga una soluzione accurata, si desidera avere una loss bassa

$$\mathcal{L}_{PINN} = |u - u^*|_{\Omega} + |\mathcal{D}u - b^*| + |u - u_0^*|_{t=0} + |u - u^*|_{\partial\Omega} \, . \tag{2.19}$$

   La componente che caratterizza la fisica del problema è $|\mathcal{D}u - b^*|$.
5. Se la loss è elevata, si necessita di ritoccare i pesi e i biases per indirizzare il modello in un'altra direzione. Per questo aspetto, si impiegano degli ottimizzatori per ridurla. In pratica, si impiega un algoritmo basato sulla discesa del gradiente. Una loss function molto utilizzata e il mean square error (MSE). Se definisco y_i l'output e y_1^* l'attuale soluzione, allora

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^*)^2 \ , \tag{2.20}$$

   per il quale la derivata rispetto a y_i è semplice da ottenere.
6. Variazione dei pesi e biases della PINN in modo da seguire una direzione discendente della loss rispetto al punto precedente.

Dato che il problema di nostro interesse è caratterizzato dalla presenza di equazioni differenziali, l'applicazione di una PINN può risultare molto utile. Si considera un problema meccanico (in cui la fisica è quindi descritta dall'equazione dell'elasticità). Assumendo che le equazioni dell'elasticità vengano studiate sul piano degli stress bidimensionale, allora esse sono descritte da

$$\sigma_{xx} = \left(\varepsilon_{xx} + \nu\varepsilon_{yy}\right)\frac{E}{1-\nu^2} \ ,$$
$$\sigma_{yy} = \left(\nu\varepsilon_{xx} + \varepsilon_{yy}\right)\frac{E}{1-\nu^2} \ , \tag{2.21}$$
$$\sigma_{xy} = \frac{E}{1-\nu^2}\varepsilon_{xy} \ ,$$

dove E denota il modulo di Young mentre ν il coefficiente di Poisson, σ_{ij} per $i,j \in \{x,y\}$ è la componente i,j -esima del tensore degli sforzi mentre ε_{ij} del tensore di deformazione. Quest'ultimo termine può essere descritto come

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right) \ , \tag{2.22}$$

per $i,j \in \{x,y\}$, dove u_i rappresenta la derivata parziale di u fatta rispetto a i. In generale, il modello lineare di elasticità include un bilancio dei momenti

$$\sigma_{ij,j} + f_i = 0 \ , \tag{2.23}$$

dove f_i per $i \in \{i,j\}$ sono forze di campo. Al fine dell'addestramento di una PINN, è necessario descrivere l'impiego della Jacobiana per determinare le derivate. Infatti, per definizione le deformazioni risultano essere la derivata parziale rispetto ad una direzione dello spostamento. Ciò può essere fatto tramite metodologie di differenziazione automatica (AD), integrate negli ambienti di machine learning **[8]**. L'AD non è una differenziazione numerica, la quale rappresenta una approssimazione alle differenze finite impiegando il valore della funzione nello stesso punto. Ad esempio, definendo $f\colon \mathbb{R}^n \to \mathbb{R}$, allora il gradiente può essere determinato come

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_m}\right) \ con \ \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x)}{h} \ per \ i = 1, \dots, n \ , \tag{2.24}$$

dove e_i è l' i-esimo vettore unitario e $h > 0$ piccolo step. L'approssimazione numerica (*Eq. 2.24*) delle derivate è intrinsecamente mal condizionata e instabile, eccetto metodi a variabili complesse applicate ad un piccolo set di funzioni olomorfe (funzione definita su un sottoinsieme aperto del piano dei numeri complessi $\mathbb{C}$ con valori in $\mathbb{C}$ che è differenziabile in senso complesso in ogni punto del dominio). Varie tecniche sono state sviluppate per mitigare l'errore di approssimazione in una differenziazione numerica, ad esempio l'approssimazione alle differenze finite centrali

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x - he_i)}{2h} + O(h^2) \ , \tag{2.25}$$

dove l'errore al primo ordine viene cancellato, spostando l'errore di troncamento dal primo al secondo ordine in h. L'AD non è nemmeno una differenziazione simbolica, la quale rappresenta la manipolazione delle espressioni ottenute dalle derivate, ottenute applicando trasformazioni rappresentate dalle seguenti regole

$$\frac{d}{dx}\big(f(x)+g(x)\big) \to \frac{d}{dx}f(x)+\frac{d}{dx}g(x)\ ,$$
$$\frac{d}{dx}\big(f(x)g(x)\big) \to \left(\frac{d}{dx}f(x)\right)g(x)+f(x)\left(\frac{d}{dx}g(x)\right)\ . \tag{2.26}$$

Quando le formule sono rappresentate come strutture dati, differenziare simbolicamente è un processo perfettamente meccanicistico, considerato soggetto ad automazione meccanica anche all'inizio stesso del calcolo. Nell'ottimizzazione, le derivate simboliche possono fornire preziose informazioni sulla struttura del dominio del problema e, in alcuni casi, produrre soluzioni analitiche degli estremi che possono eliminare la necessità di calcolare la derivata su tutto il dominio. D'altra parte, le derivate simboliche non si prestano ad un calcolo efficiente in tempo reale dei valori delle derivate, poiché possono diventare esponenzialmente più grandi; l'AD può essere considerata come l'esecuzione di un'interpretazione non standard di un programma informatico, in cui tale interpretazione comporta l'incremento del calcolo standard con il calcolo di varie derivate.

Come descritto in seguito, per l'applicazione di interesse è necessario applicare anche una parametrizzazione dei domini fisici. Tale aggiunta complica i calcoli numerici. Infatti, le composizioni di un insieme finito di operazioni elementari per le quali le derivate sono note e combinando le derivate delle operazioni costituenti attraverso la regola della catena si ottiene la derivata complessiva; per ottenere la derivata fisica dello spostamento, si richiama la regola della catena. Ad esempio $\hat{\varepsilon}_{xx}$ può essere determinato come

$$\hat{\varepsilon}_{xx} = \frac{\partial \hat{u}_x}{\partial x} = \frac{\partial u_x\big(\xi(x,y),\eta(x,y)\big)}{\partial x} = \frac{\partial u_x(\xi,\eta)}{\partial \xi}\frac{\partial \xi}{\partial x}+\frac{\partial u_x(\xi,\eta)}{\partial \eta}\frac{\partial \eta}{\partial y}\ . \tag{2.27}$$

La differenziazione automatica viene utilizzata per ottenere la derivata parziale di u_x sia rispetto a ξ, sia rispetto a η dell'*Eq. 2.27*. Comunque, le derivate parziali delle coordinate parametriche rispetto alle coordinate fisiche sono specifiche geometriche e devono essere fornite alla PINN durante la fase di addestramento poiché date dal mapping tra dominio fisico e parametrico, e quindi disponibili. In pratica, questo significa importare la Jacobiana inversa ai punti di collocazione di tutte le geometrie di addestramento

$$J^{-1} = \begin{pmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y} \end{pmatrix}\ . \tag{2.28}$$

Viene impiegata solo in fase di addestramento (e solo per PINN) perché le derivate fisiche delle distribuzioni previste dalla rete devono essere calcolate solo per determinare la perdita. Quando si applica la rete addestrata per fare previsioni, invece, una PINN produce direttamente le distribuzioni fisiche.

2.2. Contesto meccanico dell'ottimizzazione di forma

Dopo aver introdotto il funzionamento delle Physics-Informed Neural Networks e il loro impiego per la risoluzione di equazioni differenziali alle derivate parziali, è necessario definire il problema meccanico al quale la rete viene applicata. In particolare, si considera un problema di elasticità lineare, nel quale il comportamento della struttura è descritto mediante il campo di spostamento e dalle corrispondenti grandezze di tensione e deformazione. La formulazione del problema meccanico permette quindi di definire le equazioni e le condizioni che devono essere rispettate dalla soluzione della rete durante l'addestramento. Il sistema meccanico può essere descritto tramite la letteratura di **Allaire G et al [9]**. Una struttura meccanica nella sua configurazione di riferimento è descritta dalla forma $\Omega \subset \mathbb{R}^d$ (con d pari a 2 oppure 3), che è un dominio lipschitziano (sottoinsieme aperto e connesso di uno spazio euclideo la cui frontiera è "sufficientemente regolare"), il cui confine è scomposto in the pezzi disgiunti: $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma$, dove:

- La struttura è attaccata alla regione $\Gamma_D \neq \emptyset$,
- I carichi g: $\Gamma_D \to \mathbb{R}^d$ sono applicati su Γ_N,
- La regione rimanente Γ non è sottoposta ai carichi.

Inoltre, Ω è soggetta alle forze di campo (i.e. forza di gravità) $f : \Omega \to \mathbb{R}^d$ (*Figura 2*). In questa situazione, lo spostamento $u_\Omega : \Omega \to \mathbb{R}^d$ è un'unica soluzione al sistema lineare dell'elasticità

$$\begin{cases} -div\big(\sigma(u_\Omega)\big) = f & in\ \Omega, \\ u_\Omega = 0 & su\ \Gamma_D, \\ \sigma(u_\Omega)n = g & su\ \Gamma_N, \\ \sigma(u_\Omega)n = 0 & su\ \Gamma. \end{cases} \tag{2.29}$$

Qui, $\sigma(u) : \Omega \to \mathbb{R}^{d\times d}$ è il tensore degli stress introdotti dal campo degli spostamenti $u : \Omega \to \mathbb{R}^d$: per qualsiasi vettore unitario $\kappa \in \mathbb{R}^d$, il vettore $\sigma(u)(x)\kappa$ è la forza risentita dalla superficie ortogonale a κ in una porzione piccola di materiale intorno a x. Il tensore degli stress $\sigma(u)$ è legato al tensore di deformazione

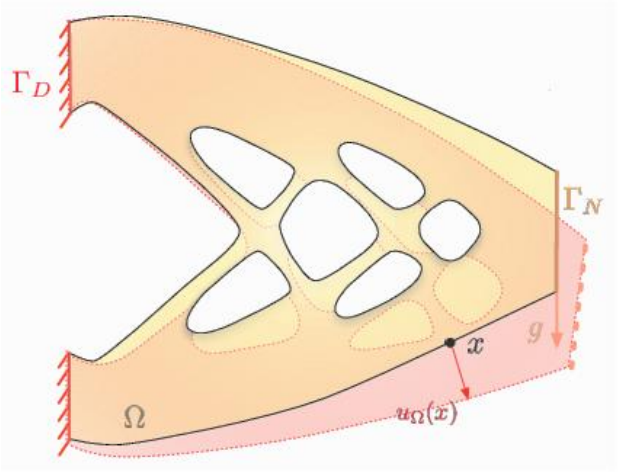


Figura 2 - Esempio di struttura elastica: trave a mensola. L'area gialla rappresenta la configurazione di riferimento, mentre la rossa la configurazione a seguito del carico applicato. [9]

$e(u) := \frac{1}{2}(\nabla u + \nabla u^T)$ mediante la legge di Hooke

$$\sigma(u) = Ae(u) \ ,$$
$$Ae = 2\mu e + \ \lambda tr(e)I \ . \tag{2.30}$$

I è la matrice identità $d \times d$, mentre λ e μ sono i parametri di Lamé del materiale che costituisce la struttura, i quali devono soddisfare $\mu > 0$ e $\lambda + 2\mu/d > 0$. Spesso sono definiti come

$$\mu = \frac{E}{2(1+\nu)} \ ,$$
$$\lambda = \frac{E\nu}{(1+\nu)(1+\nu(1-d))} \ . \tag{2.31}$$

Le forze di campo f appartengono allo spazio $L^2(\Omega)^d$ di funzioni integrali su Ω, e $g \in L^2(\Gamma_N)^d$. Lo spostamento u_Ω è un elemento di $H^1_{\Gamma_D}(\Omega)^d$, dove

$$H^1_{\Gamma_D}(\Omega) := \{u \in H^1(\Omega), u = 0 \ su \ \Gamma_D\} \ . \tag{2.32}$$

Così facendo, il sistema (*Eq. 2.29*) può essere inteso come

$$Cerca \ u_\Omega \in H^1_{\Gamma_D}(\Omega)^d \ t.c. \forall v \in H^1_{\Gamma_D}(\Omega)^d, \int_\Omega Ae(u_\Omega) : e(v) \ dx = \int_\Omega f \cdot v \ dx + \int_{\Gamma_N} g \cdot v \ ds \ , \tag{2.33}$$

la cui relazione deriva della teoria standard di Lax-Milgram. In generale:

- Dal punto di vista della modellizzazione: difficoltà nel descrivere il problema fisico in gioco mediante un modello che sia rilevante (quindi sufficientemente complicato), ma trattabile (cioè abbastanza semplice).
- Dal punto di vista teorico: spesso, le forme ottimali non esistono e i problemi di ottimizzazione della forma godono al massimo di ottimi locali.
- Dal punto di vista sia teorico che numerico: la variabile di ottimizzazione è il dominio. Da qui la necessità di un mezzo per differenziare le funzioni che dipendono dal dominio e, prima ancora, di parametrizzare le forme e le loro variazioni.
- Dal lato numerico: difficoltà nel rappresentare le forme e le loro evoluzioni; i problemi di ottimizzazione della forma possono essere molto sensibili e possono essere completamente dominati da errori di discretizzazione.

Dopo aver descritto le relazioni che descrivono lo spostamento subito dal sistema meccanico, ora si va ad analizzare il modo in qui si arriva ad ottenere la configurazione ottimale. Come tecnica principale si utilizza il metodo di discesa del gradiente. Si immagina di avere una forma iniziale Ω. A questo dominio si applica adesso una deformazione θ. Sotto questa deformazione, ogni punto x della struttura si sposta in un nuovo punto $x + \theta(x)$. Si indica con Ω_θ la nuova forma ottenuta. Indicando con $\mathcal{J}(\Omega) \in \mathbb{R}$ una funzione scalare, essa è definita

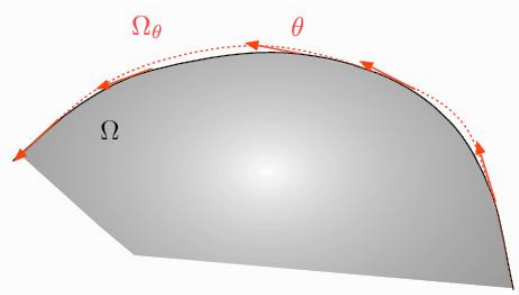


Figura 3 - Al primo ordine, da un campo vettore tangente θ ($\theta \cdot n = 0$) ci si aspetta che $\mathcal{J}'(\Omega)(\theta) = 0$ [9].

la derivata di forma di Ω se è differenziabile tramite derivata di Fréchet all'istante iniziale, applicando così la seguente espansione

$$\mathcal{J}(\Omega_\theta) = \mathcal{J}(\Omega) + \mathcal{J}'(\Omega)(\theta) + o(\|\theta\|)\,, \tag{2.34}$$

con $\mathcal{J}'(\Omega)(\theta)$ derivata di $\mathcal{J}$ in Ω. L'idea è che la derivata di forma $\mathcal{J}'(\Omega)(\theta)$ di una funzione regolare $\mathcal{J}(\Omega)$ dipende esclusivamente dalla componente normale $\theta \cdot n$ del vettore deformazione θ (*Figura 3*). Attualmente, la derivata di forma di 'molte' funzioni obiettivo può essere espressa nella forma

$$\mathcal{J}'(\Omega)(\theta) = \int_{\partial\Omega} v_\Omega(\theta \cdot n)\, ds\ , \tag{2.35}$$

dove $v_\Omega : \partial\Omega \to \mathbb{R}$ è un campo scalare che dipende da $\mathcal{J}$ e dalla forma corrente Ω. Questa struttura si presta al calcolo mediante discesa del gradiente: infatti, ponendo $\theta = -tv_\Omega n$, per un piccolo step $t > 0$

$$\mathcal{J}(\Omega_{t\theta}) = \mathcal{J}(\Omega) - t\int_{\partial\Omega} v_\Omega^2\, ds + o(t) < \mathcal{J}(\Omega)\ . \tag{2.36}$$

Esistono una serie di tecniche generali impiegate per l'ottimizzazione di forma, però molte di esse sono onerose dal punto di vista del calcolo numerico, in particolare per il calcolo delle derivate. Esiste un metodo più semplice, che consente di determinare le derivate di forma: il metodo di Cèa (*Figura 4*) consiste nell'esprimere $\mathcal{J}(\Omega)$ come un punto di sella della Lagrangiana

$$\mathcal{L}(\Omega, u, p) = \int_\Omega j(u)dx + \int_\Omega (-\Delta u - f)p\, dx\ , \tag{2.37}$$

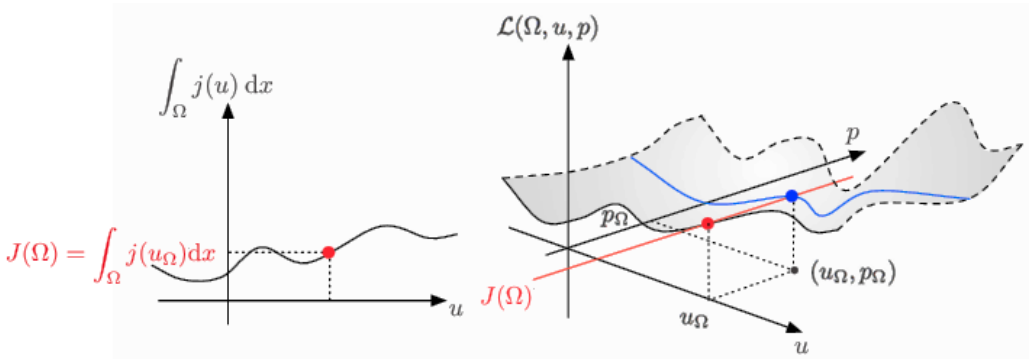


Figura 4 - Il metodo di Céa. [9]

dove le variabili Ω, u, p sono indipendenti. Il primo integrale caratterizza la funzione obiettivo, mentre il secondo consente di rispettare i vincoli imposti mediante penalizzazione con il moltiplicatore di Lagrange p. Questo metodo è formale: in particolare, si assume che $\Omega \longmapsto u_\Omega$ sia differenziabile.

A questo punto, è possibile passare all'algoritmo di discesa del gradiente: partendo da una forma iniziale Ω^0, per $n = 0, \ldots$, fino a convergenza:

- Calcola lo stato u_{Ω^n} (e possibilmente quello aggiunto p_{Ω^n}) del sistema differenziale considerato su Ω^n.
- Calcola il gradiente di forma $\mathcal{J}'(\Omega^n)$ grazie alla *Eq. 2.35* e deduce una direzione di discesa θ^n per la funzione costo (che in questo caso descrive la forma della struttura).
- Calcola la forma allo step successivo in accordo con il campo di spostamento per un piccolo pseudo-time step τ^n, in modo da ottenere $\Omega^{n+1} = (I + \tau^n \theta^n)(\Omega^n)$.

Ogni forma è rappresentata mediante una mesh simpliciale $\mathcal{T}^n$ (cioè mesh composte da triangoli in due dimensioni oppure tetraedri in tre dimensioni). Il metodo agli elementi finiti viene impiegato per generare la mesh $\mathcal{T}^n$. La direzione di discesa θ^n è successivamente calcolata utilizzando la formula teorica per la derivata di $\mathcal{J}(\Omega)$. La modifica della mesh $\Omega^n \longmapsto \Omega^{n+1}$ è ottenuta spingendo i nodi (*Figura 5*) di $\mathcal{T}^n$ lungo $\tau^n \theta^n$, in modo da ottenere una nuova mesh $\mathcal{T}^{n+1}$.

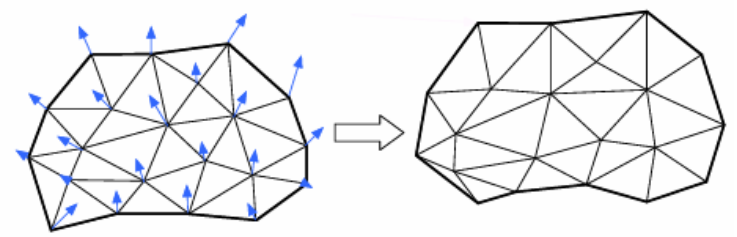


Figura 5 - Deformazione della mesh tramite riallocazione dei nodi per definire la posizione finale. [9]

Nell'affrontare questa tematica, vi possono presentare alcuni problemi e difficoltà:

- Difficoltà nella deformazione della mesh: quando la forma è esplicitamente modificata, è necessario aggiornare la mesh ad ogni step $\Omega^n \longmapsto (I + \tau^n \theta^n)(\Omega^n) = \Omega^{n+1}$. La nuova mesh ottenuta $\mathcal{T}^{n+1}$ è ottenuta riallocando ogni nodo $x \in \mathcal{T}^n$ a $x + \tau^n \theta^n$. Questo potrebbe comportare delle difficoltà, il particolare la possibilità di ottenere delle inversioni degli elementi, invalidando la mesh (*Figura 6*). Per questa ragione, il metodo di deformazione è preferibile applicarlo a seguito di 'piccoli' spostamenti.

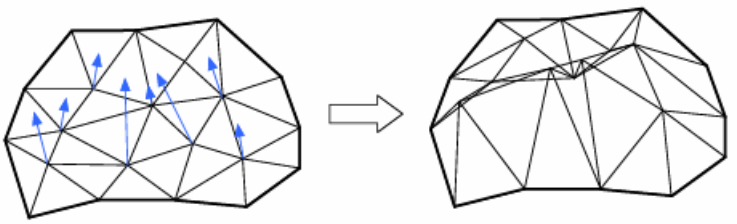


Figura 6 - Configurazione invalida a seguito dell'inversione degli elementi. [9]

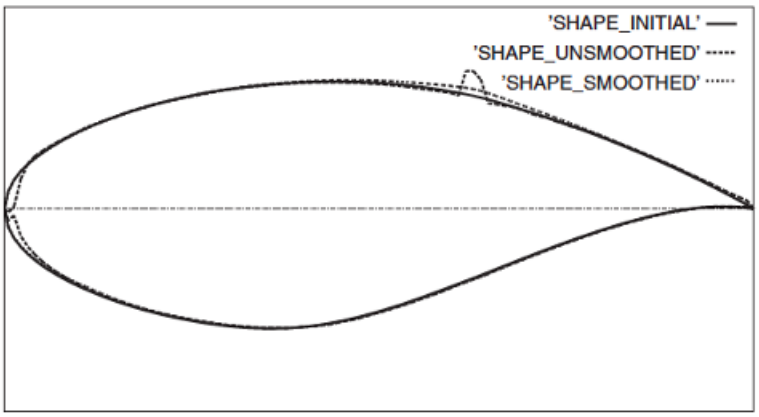


Figura 7 - Irregolarità della derivata di forma in un problema di minimizzazione del drag di un profilo alare. [10]

- Velocità di estensione: una direzione di discesa $\theta = -v_\Omega n$ a partire da Ω è dedotta dalla derivata di forma di $\mathcal{J}(\Omega)$, calcolata tramite l'*Eq. 2.35*. Per molte ragioni, è cruciale estendere θ nel modo più 'intelligente' possibile. Dunque, la 'naturale' estensione della formula $\theta = -v_\Omega n$, che è valida solo per $\partial\Omega$ potrebbe non essere una 'buona' scelta.
- Velocità di regolarizzazione: prendendo $\theta = -v_\Omega n$ su $\partial\Omega$ potrebbe produrre una irregolarità (*Figura 7*), per via di artefatti numerici che si presentano durante l'analisi FEM e per via di mancata regolarità di $\mathcal{J}'(\Omega)$. Nella pratica, è spesso necessario lisciare la direzione di discesa affinché le forme considerate rimangano regolari **[10]**.
- Impiegare un parametro di regolazione: si suppone di estendere il campo scalare $v_\Omega: \partial\Omega \to \mathbb{R}$ fino a Ω. L'idea è quella di sostituire il prodotto interno 'naturale' su $L^2(\partial\Omega)$ con un prodotto interno più regolare su Ω. Ad esempio, si cerca di estendere/regolarizzare il campo scalare V come

$$Cerca\, V \in H^1(\Omega)\ \ t.c.\ \forall w \in H^1(\Omega)\ \ \alpha \int_\Omega \nabla V \cdot \nabla w\, dx + \int_\Omega Vw\, dx = \int_{\partial\Omega} \mathrm{v}w\, ds\ \ . \tag{2.38}$$

  Il parametro di regolazione α controlla il bilancio tra la fedeltà di V a v_Ω e l'intensità di smoothing. Il campo scalare risultante V è intrinsecamente definito su Ω ed è più regolare di v_Ω.

La scelta di un metodo numerico per l'ottimizzazione di forma deve raggiungere un compromesso tra accuratezza numerica e robustezza: più accurata è la rappresentazione dei confini, più accurate sono le analisi meccaniche effettuate sulle forme (calcolo di u_{Ω^n}, p_{Ω^n}, ecc.) e più accurato è il calcolo delle direzioni di discesa. Però, è più laborioso e soggetto a errori al passaggio tra le forme $\Omega^n \mapsto \Omega^{n+1}$; l'ottimizzazione di forma presenta dei legami con la topologia del sistema. Infatti, nel momento in qui il modello numerico modifica la forma di partenza al fine di cercare un ottimo, potrebbe involontariamente modificare la struttura interna. Dunque, altro fattore importante da considerare è la possibilità di ottenere una variazione in volume della struttura (*Figura 8*). Infatti, se non imponesse variazione di volume nulla il modello analitico tenderebbe a generare configurazioni dove la struttura risulta completamente piena (e ciò è dovuto al fatto che applicando la teoria dell'elasticità, riempiendo completamente la struttura si ottiene la massima rigidezza). Partendo dalla

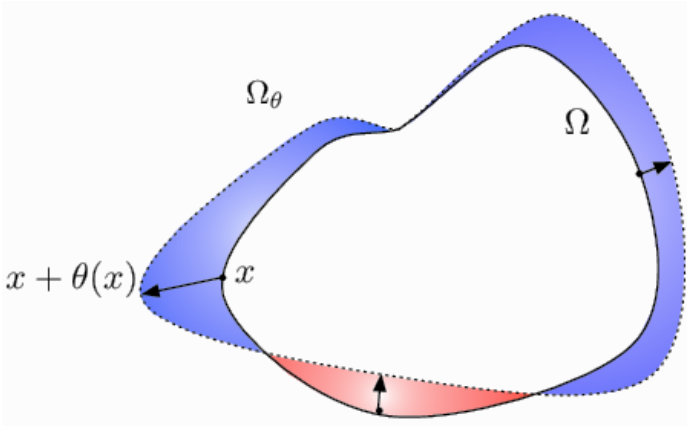


Figura 8 - Se il modello, per modificare la forma, tende ad aumentare il volume (regione blu), allora necessariamente alcune zone devono subire una contrazione (regione rossa). [9]

definizione di volume, si determina la variazione di volume e come step successivo si applica il teorema della divergenza

$$V(\Omega) = \int_\Omega 1 \, dx \quad \Longrightarrow \quad V'(\Omega)(\theta) = \int_{\partial\Omega} \theta \cdot n \, ds = \int_\Omega div(\theta) \, dx \quad , \tag{2.39}$$

se la divergenza del campo di deformazione è nulla, allora la struttura con la configurazione variata dal modello presenterà lo stesso volume di partenza.

2.3. Parametrizzazione mediante NURBS

Come evidenziato nel paragrafo precedente, nell'ottimizzazione di forma la variabile di progetto non è costituita direttamente dai parametri del modello fisico, ma è rappresentata dalla geometria del dominio. Ne consegue che, per poter applicare in modo efficace una procedura iterativa di ottimizzazione, è necessario disporre di una rappresentazione parametrica della forma, che permetta di descrivere le variazioni geometriche attraverso un numero limitato di parametri. In questo modo, la modifica della geometria può essere incorporata direttamente all'interno della procedura di ottimizzazione, evitando di dover intervenire esplicitamente sulla rappresentazione del dominio fisico ad ogni iterazione. Nel presente lavoro, tale rappresentazione viene ottenuta mediante l'utilizzo di un dominio parametrico e della parametrizzazione Non Uniform Rational Basis-Splines (NURBS). Questa scelta consente di descrivere diverse configurazioni geometriche attraverso la variazione dei parametri associati alla rappresentazione NURBS, mantenendo invece fissa la distribuzione dei punti nel dominio parametrico. Tale caratteristica risulta particolarmente utile nel contesto delle reti neurali considerate, poiché permette di fornire alla rete, oltre alle coordinate parametriche dei punti di collocazione, anche i parametri che caratterizzano la geometria del dominio. Il modello ha come obiettivo l'ottenimento del profilo ottimale per il foro della piastra. Il codice impiegato non effettua quindi la modifica del profilo direttamente nel dominio fisico, ma opera attraverso la parametrizzazione del dominio mediante NURBS.

2.3.1. Algoritmo NURBS

La parametrizzazione può essere ottenuta nel seguente modo: sia $\{B_{i,p}\}\ per\ i = 0, \dots, n$ la funzione B-Spline di grado p definita sul vettore dei nodi $k_\xi = \{\xi_0, \xi_1, \dots, \xi_{n+p+1}\}$. Per una generica i-esima B-Spline, si ha

$$B_{i,p}(\xi) = \frac{(\xi - \xi_i)B_{i,p-1}(\xi)}{\xi_{i+p} - \xi_i} - \frac{(\xi_{i+p+1} - \xi)B_{i+1,p-1}(\xi)}{\xi_{i+p+1} - \xi_{i+1}} \ ,$$
$$B_{0,p}(\xi) = \begin{cases} 1 & \xi_i \le \xi \le \xi_i \\ 0 & altrove \end{cases} \ . \tag{2.40}$$

La derivata della B-Spline è data da

$$B'_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi) \ . \tag{2.41}$$

Una curva NURBS di grado p può essere definita come

$$\boldsymbol{x}(\xi) = \frac{\sum_{i=0}^{n} B_{i,p}(\xi) w_i P_i}{\sum_{j=0}^{n} B_{j,p}(\xi) w_j} \ , \tag{2.42}$$

dove $\{P_i\}\ per\ i = 0, \dots, n$ rappresenta le coordinate dei punti di controllo, $w_i\ per\ i = 0, \dots, n$ i corrispondenti pesi e $\xi \in [0, 1]$ la variabile parametrica. L'*Eq. 2.42* può essere riscritta come

$$\boldsymbol{x}(\xi) = \sum_{i=0}^{n} R_{i,p}(\xi) P_i \ , \tag{2.43}$$

dove $R_{i,p}(\xi)$ viene definito come

$$R_{i,p}(\xi) = \frac{B_{i,p}(\xi) w_i}{\sum_{j=0}^{n} B_{j,p}(\xi) w_j} \ , \tag{2.44}$$

Dunque, derivando l'*Eq. 2.43* si ottiene

$$\boldsymbol{x}_\xi(\xi) = \sum_{i=0}^{n} R'_{i,p}(\xi) P_i \ , \tag{2.45}$$

dove $R'_{i,p}(\xi)$ è la derivata di $R_{i,p}(\xi)$

$$R'_{i,p}(\xi) = \frac{B_{i,p}(\xi) w_i}{\sum_{j=0}^{n} B_{j,p}(\xi) w_j} - \frac{B_{i,p}(\xi) w_i (\sum_{j=0}^{n} B'_{j,p}(\xi) w_j)}{(\sum_{j=0}^{n} B_{j,p}(\xi) w_j)^2} \ . \tag{2.46}$$

Similmente, è possibile definire una superficie NURBS di grado p rispetto a ξ e di grado q rispetto a η

$$\boldsymbol{x}(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,p}(\xi) R_{j,q}(\xi) P_{ij} \ , \tag{2.47}$$

dove i punti di controllo $\{P_{ij}\}$ nella forma $\{(n+1)\times(m+1)\}$ sono associati i pesi $\{w_{ij}\}$. Finalmente, $\{R_{i,p}(\xi)\}$ e $\{R_{j,q}(\eta)\}$ sono le funzioni NURBS basate sulle funzioni B-Spline, dove quest'ultime sono definite dal noto vettore $k_\xi = \{\xi_0, \xi_1, \dots, \xi_{n+p+1}\}$ e $k_\eta = \{\eta_0, \eta_1, \dots, \eta_{m+q+1}\}$. Dunque, le derivate delle NURBS sono

$$\begin{aligned} \boldsymbol{x}_\xi(\xi) &= \sum_{i=0}^{n}\sum_{j=0}^{m} R'_{i,p}(\xi)R_{j,q}(\eta)P_{ij} \quad , \\ \boldsymbol{x}_\eta(\eta) &= \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,p}(\xi)R'_{j,q}(\eta)P_{ij} \quad . \end{aligned} \tag{2.48}$$

Una delle proprietà più interessanti della parametrizzazione tramite NURBS è la possibilità di andare a generare una serie vasta di geometrie con un numero piccolo di punti di controllo. Ciò consente di andare a limitare il costo computazionale del calcolo senza andare a limitare la precisione del modello.

FORMULAZIONE DEL PROBLEMA

In questo elaborato viene scelta come geometria di riferimento una piastra forata. Data la simmetria rispetto ad entrambi gli assi, allora è possibile semplificare il problema andando a considerare solamente un quarto di piastra. Ridurre il dominio computazionale mantenendo le condizioni di simmetria appropriate sul bordo consentirà di diminuire il costo computazionale; dunque, l'obiettivo è quello di andare ad ottimizzare il profilo del foro (*Figura 9*) partendo da una generica geometria di partenza.

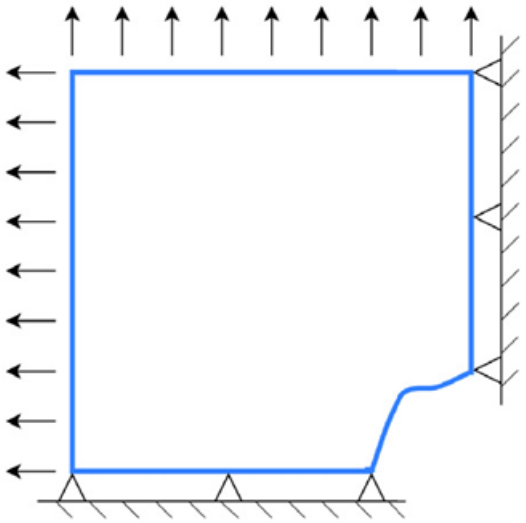


Figura 9 - Schema della piastra forata da ottimizzare. [11]

Seguendo la letteratura di **Mezzadri F et al [11]**, scelto un dominio fisico generico Ω con coordinate (x, y), esso può essere mappato su un dominio parametrico $\widehat{\Omega}$, con coordinate parametriche (ξ, η). La mappatura tra Ω e $\widehat{\Omega}$ viene costruita tramite NURBS. La mappatura geometrica tra i domini (*Figura 10*) si basa sulle coordinate di un insieme di punti di controllo, indicati con P_{ij}. Lavorando nel dominio parametrico, le coordinate parametriche dei punti di collocazione non cambieranno nemmeno quando il dominio fisico viene modificato. Supponendo di avere accesso alla soluzione $u^*(\xi, \eta)$ di un'equazione differenziale alle derivate parziale in questi punti di collocazione per una o più forme di addestramento, è possibile addestrare una rete neurale in modo da essere un modello surrogato della soluzione della PDE su diversi domini. Gli input di tale rete neurale sono le coordinate parametriche (ξ, η) insieme ai parametri di forma che caratterizzano la forma dei domini. I dati di addestramento sono costituiti dalla soluzione $u^*(\xi, \eta)$ della PDE nei domini di addestramento. L'utilizzo della parametrizzazione NURBS è comodo per generare i dati tramite analisi isogeometrica (IGA). Con queste informazioni, è possibile formulare una funzione di perdita NN (completamente basata sui dati di addestramento) o una funzione di perdita PINN (in cui sono inclusi anche termini basati sulla fisica). La rete neurale addestrata può prevedere una soluzione dell'equazione differenziale per domini appartenenti alla famiglia di geometrie descritta dalla parametrizzazione adottata quando la rete viene alimentata con un insieme di coordinate (ξ, η) e con i parametri di forma, come punti di controllo

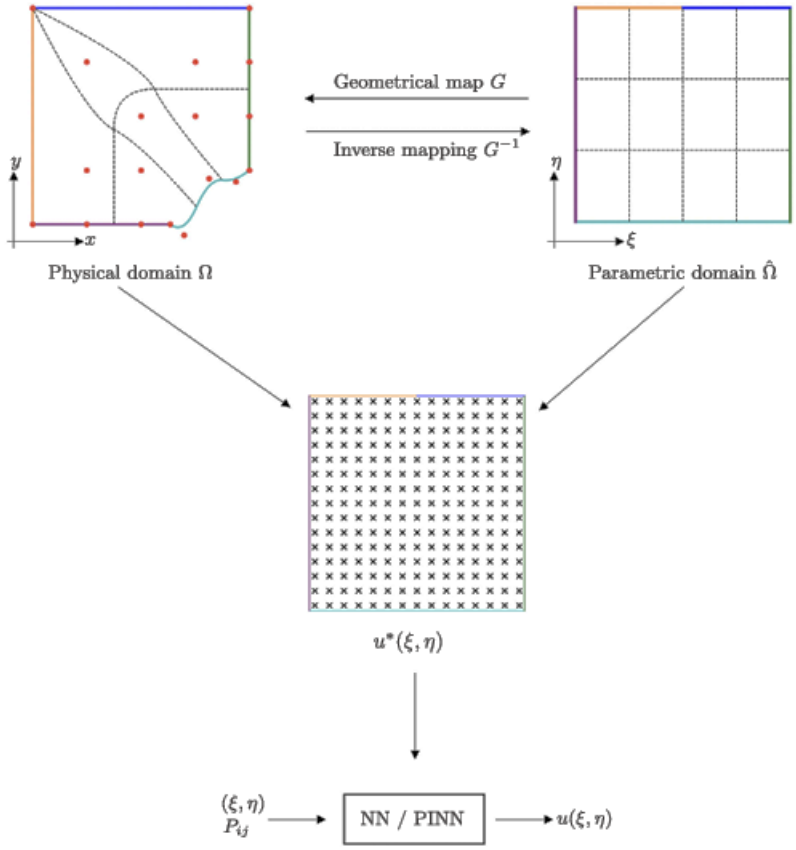


Figura 10 - Confronto tra dominio fisico e dominio parametrico. [11]

NURBS. Per dimostrare l'efficacia di questo approccio, vengono affrontati problemi di elasticità lineare definiti su domini di diversa complessità.

A questo punto, dopo aver definito i dati necessari per l'addestramento, il prossimo step è quello di definire il modello che verrà successivamente analizzato e simulato. Si ricorda che l'obiettivo del modello è la minimizzazione della funzione $\mathcal{L}_{PINN}$, applicando all'*Eq. 2.19* si ottiene

$$\mathcal{L}_{PINN} = \mathcal{L}_{data} + \mathcal{L}_{PDE} \ , \tag{3.1}$$

dove i termini che costituiscono la loss function sono dati da

$$\begin{aligned} \mathcal{L}_{data} = \ & |\hat{u}_x - \hat{u}_x^*| + |\hat{u}_y - \hat{u}_y^*| + |\hat{\sigma}_{xx} - \hat{\sigma}_{xx}^*| + |\hat{\sigma}_{yy} - \hat{\sigma}_{yy}^*| + |\hat{\sigma}_{xy} - \hat{\sigma}_{xy}^*| \\ \mathcal{L}_{PDE} = \ & |\hat{\sigma}_{xx,x} - \hat{\sigma}_{xy,y}^* - f_x^*| + |\hat{\sigma}_{xy,x} - \hat{\sigma}_{yy,y}^* - f_y^*| + \\ & + \left|\left(\hat{\varepsilon}_{xx} + \nu\hat{\varepsilon}_{yy}\right)\frac{E}{1-\nu^2} - \hat{\sigma}_{xx}\right| + \left|\left(\hat{\varepsilon}_{yy} + \nu\hat{\varepsilon}_{xx}\right)\frac{E}{1-\nu^2} - \hat{\sigma}_{yy}\right| + \left|\frac{E}{1-\nu^2}\hat{\varepsilon}_{xy} - \hat{\sigma}_{xy}\right| \end{aligned} \ . \tag{3.2}$$

Le grandezze di interesse in uscita dal modello sono gli spostamenti $\hat{u}$ e le tensioni $\hat{\sigma}$ (*Figura 11*): tali sono in funzione delle coordinate parametriche (ξ,η), ma grazie alle curve NURBS viste nel *Paragrafo 2.3*, il modello sarà in grado di modificare la forma nel dominio fisico in modo da definirne quella ottimale. Una volta addestrata, la rete neurale viene utilizzata all'interno della procedura di ottimizzazione di forma come modello surrogato della soluzione del problema. Per ogni configurazione geometrica, definita dai parametri di forma associati ai punti di controllo NURBS, la rete fornisce la previsione degli spostamenti e delle tensioni nel

dominio considerato. A partire da tali grandezze è quindi possibile determinare la compliance della struttura, che viene utilizzata come funzione obiettivo della procedura di ottimizzazione. I parametri di forma vengono successivamente modificati dall'algoritmo di ottimizzazione e la rete viene nuovamente impiegata per prevedere la risposta della nuova geometria. Il procedimento viene ripetuto iterativamente fino al raggiungimento del criterio di arresto, ottenendo così il profilo ottimizzato del foro.

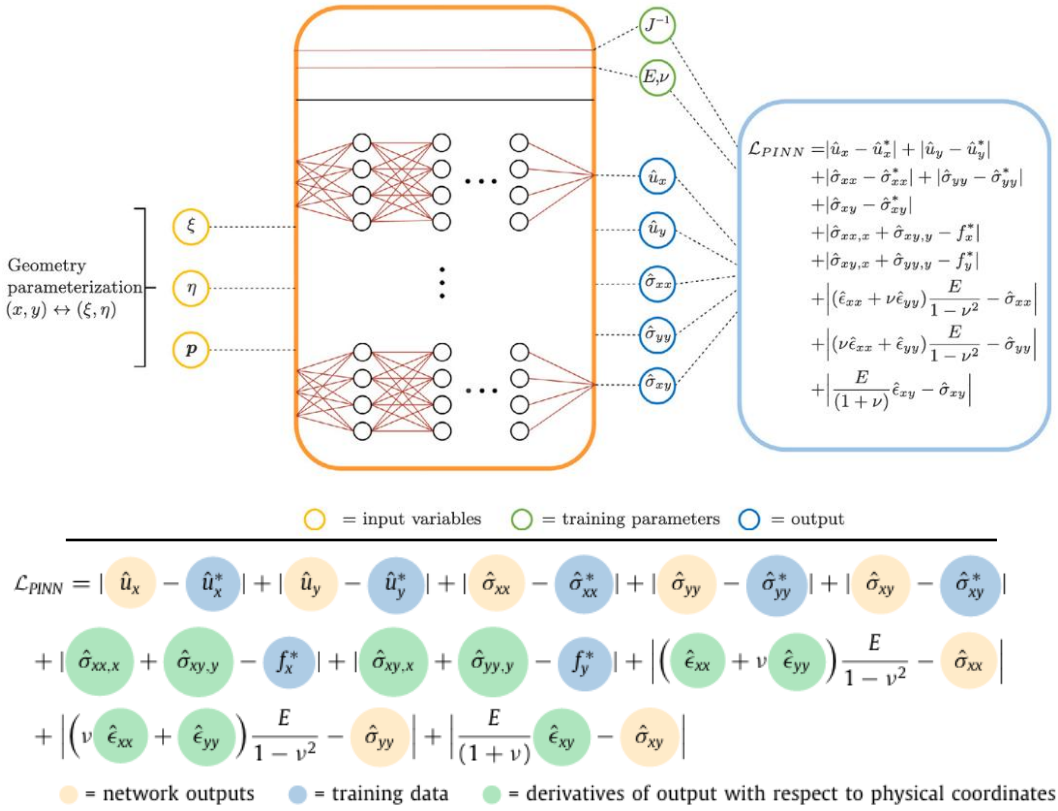


Figura 11 - Physics Informed Neural Network. [11]

ESPERIMENTI NUMERICI

Le simulazioni sono state condotte tramite VivoBook ASUS, con un processore 11th Gen Intel® Core (Tm) i7-1165G7 @ 2.80 GHz e 16 GB di RAM. Come linguaggio di programmazione è stato impiegato Python: linguaggio informatico moderno, utilizzato in ambito delle reti neurali e/o data analysis, caratterizzato da dinamicità, semplicità e flessibilità. Python è un linguaggio di scripting avanzato che viene utilizzato con successo per integrare componenti software di grandi dimensioni. È presente su diverse piattaforme, prodotti middleware e ambiti applicativi. Esistono diversi esempi di applicazioni che inizialmente erano scritte interamente in C++ e a cui Python è stato aggiunto come linguaggio di estensione, e in cui, ad ogni nuova versione, la percentuale di codice scritto in Python è aumentata, migliorando al contempo le prestazioni, la funzionalità e l'affidabilità complessive dell'applicazione **[12]**. Vengono impiegate due librerie di riferimento: la prima è TensorFlow. È una libreria open-source per il calcolo numerico, il machine learning su larga scala, il deep learning e altre attività di analisi statistica e predittiva. Questo tipo di tecnologia consente l'implementazione di modelli di machine learning più semplice e veloce per gli sviluppatori, assistendoli nel processo di acquisizione dei dati, nella formulazione di previsioni su larga scala e nel successivo affinamento dei risultati. Questo significa che è possibile addestrare e implementare un modello di machine learning rapidamente, indipendentemente dal linguaggio o dalla piattaforma che utilizzi **[13]**. La seconda libreria di riferimento è SciANN: utilizza i pacchetti di deep learning ampiamente diffusi Tensorflow e Keras per costruire reti neurali profonde e modelli di ottimizzazione, ereditando così molte delle funzionalità di Keras, come l'ottimizzazione batch e il riutilizzo del modello per il transfer learning. SciANN è progettato per astrarre la costruzione di reti neurali per calcoli scientifici e la soluzione e scoperta di equazioni differenziali parziali (PDE) utilizzando l'architettura delle reti PINN, fornendo quindi la flessibilità necessaria per impostare forme funzionali complesse **[14]**. Come interfaccia per gli addestramenti e le verifiche è stato impiegato Anaconda: programma open-source di Python ampiamente utilizzata, pensata per la scienza dei dati, l'apprendimento automatico e l'intelligenza artificiale. Semplifica la gestione e l'installazione dei pacchetti, integrando Python con centinaia di librerie di dati preinstallate e includendo Conda, un potente ambiente virtuale e gestore di pacchetti (*Tabella 3* in appendice).

4.1. Addestramento

La prima fase dell'addestramento è incentrata su una rete neurale semplice, ovvero senza le informazioni fisiche. Sono state condotte una serie di simulazioni in modo da analizzare l'influenza della variazione della profondità, a parità di numero di neuroni per strato; tutti i risultati ottenuti provengono da addestramenti in cui vengono utilizzate 100 forme di addestramento (di 500 totali), in ognuna noti 100×100 punti, con un numero massimo di iterazioni pari a 1200; analizzare la profondità di rete è fondamentale per due aspetti principali: in primis, più strati alla rete garantiscono di migliorare sensibilmente la precisione del calcolo delle derivate, in secundis aumentare gli strati comporta un aumento del tempo di addestramento e della probabilità di ottenere

overfitting. L'overfitting è un elemento da evitare in quanto la rete potrebbe dare luogo a risultati meno precisi rispetto a quelli ottenuti in fase di addestramento. In altre parole, la rete risulterebbe addestrata talmente bene rispetto alle geometrie di addestramento che in fase di verifica la geometria ottimizzata da luogo ad una precisione inferiore; in questa parte, le reti addestrate presentano tutte un numero di neuroni per strato pari a 20. Analizzando la *Figura 12*, si può vedere che all'aumentare del numero di strati aumenta in modo "quasi" lineare il tempo necessario per effettuare l'addestramento, questo in accordo con l'*Eq 2.9*. Seppur una considerazione semplice, il tempo necessario per un addestramento è un aspetto critico: infatti, aumentare il numero di strati consente di aumentare la precisione del calcolo delle derivate, ma deve essere giustificato dal risultato ottenuto. Spendere una quantità di tempo elevata ad addestrare una rete per limare la precisione del codice, per questa casistica, porta solo ad una spesa di tempo ed energia non necessaria. Analizzando solo il tempo, una scelta consapevole è quella di impostare una rete con un numero di strati compresa tra quattro e sei, dove ci si può aspettare un risultato sufficientemente accurato a fronte di un tempo di addestramento che può essere leggermente superiore al doppio del primo caso. Uno step successivo è quello di valutare la loss function: in questo caso è possibile subito notare la forte diminuzione del valore all'aumentare del numero di strati, andando oltre il 97% di diminuzione oltre i quattro strati. Questo risultato è interessante, dato che in questo modo non si ha la necessità di appesantire la rete neurale per ottenere un risultato migliore, quindi risparmiando potenza di calcolo. Considerando che lo studio è incentrato su una rete superficiale, molto impiegata per la risoluzione delle equazioni differenziali alle derivate parziali, dai dati ottenuti è possibile decidere di optare per una rete con un numero di strati compreso tra quattro e sei: rispetto al caso con un solo strato, il tempo necessario per l'addestramento oscilla tra 50% e 100% rispetto al primo caso, ma con una diminuzione della loss function compresa tra 97% e 98%. Andare oltre significa aumentare significativamente il tempo di addestramento (si ricorda avere una crescita "quasi" lineare), con una diminuzione particolarmente ridotta della loss function; dall'addestramento è interessante valutare i valori ottenuti della loss function relativi agli spostamenti e agli elementi che compongono la matrice di deformazione: infatti, al calare del valore la rete è in grado di approssimare al meglio lo spostamento e deformazione rispetto alla soluzione.

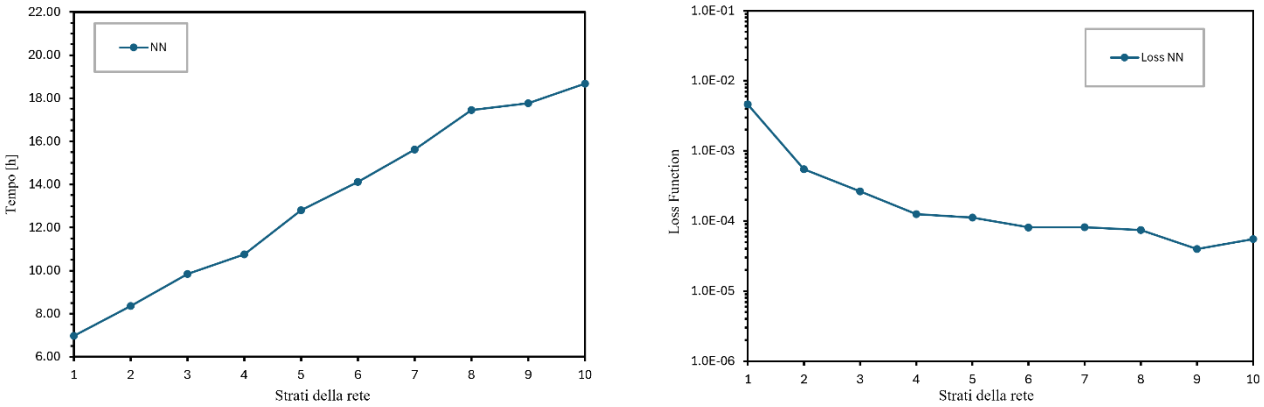


Figura 12 - Tempo di Addestramento (a sinistra) e Loss Function (a destra), rete NN

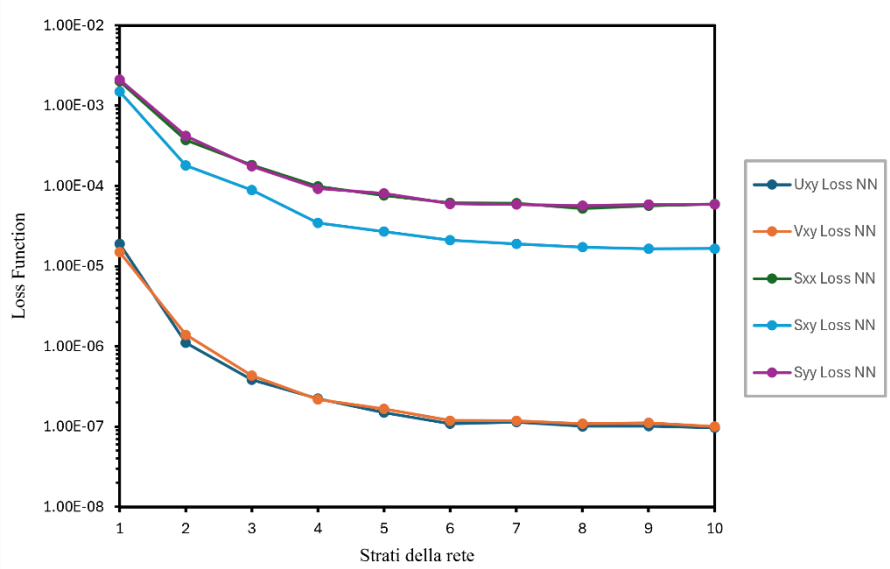


Figura 13 - Loss Function Spostamenti e Tensore di Defromazione

Analizzando la *Figura 13*, è apprezzabile una diminuzione di tutti i termini della rete (il grafico è in scala semi-logaritmica). Si nota un calo sensibile dei temini fino ad una rete con quattro strati, perdendo circa due ordini di grandezza. Dopodiché, le reti successive presentano una struttura che si, riescono ad aumentare la precisione di calcolo, ma non diminuiscono sensibilmente il valore della loss. Dunque, a conferma di ciò che è stato detto precedentemente, analizzando i dati ottenuti in fase di addestramento, per un problema di ottimizzazione di forma come quello analizzato impiegare una rete neurale con un numero di strati superiore a sei può portare alla generazione di reti che non limitano notevolmente il valore della loss (fondamentale per capire quanto la rete si avvicina alla soluzione di riferimento), seppur spendendo una mole piuttosto generosa di tempo ed energia necessaria per l'addestramento. Questa affermazione, però, dovrà essere valutata in seconda sede, dopo che le reti neurali saranno sottoposte a verifica. Successivamente, se i risultati ottenuti in fase di verifica si discostano da quelli analizzati in addestramento, allora la rete di riferimento sarà data dal compromesso dei due risultati; il passo successivo è quello di confrontare le reti NN con quelle aventi informazioni sulla fisica. Innanzitutto, quello che ci si aspetta è un aumento sia del tempo impiegato per l'addestramento, sia della loss function. In entrambi casi, l'elemento che produce questo effetto è il residuo della PDE. Nel primo caso, la rete PINN non usa le differenze finite, ma la differenziazione automatica, valutando il residuo della PDE in un numero elevato di punti del dominio, per assicurare che la fisica sia rispettata ovunque e non solo dove sono disponibili i dati. Elaborare una mole elevata dei punti geometrici ad ogni iterazione aumenta notevolmente il carico di lavoro. Nel secondo caso, per il calcolo della loss function per PINN si somma a tutti i residui anche quello della PDE, termine che ovviamente non è presente per NN (*Eq 2.11)*. Il vantaggio di impiegare una rete con l'informazione fisica si manifesta quando non si ha la disponibilità un numero sufficiente di geometrie di addestramento o che siano imperfette, ma la fisica del problema è ben nota per via delle PDE. Poiché le leggi della fisica sono valide ovunque, essa mantiene la sua validità e accuratezza anche quando deve estrapolare informazioni in zone del dominio mai viste prima. Contrariamente, se i dati di addestramento sono sufficientemente elevati, la NN potrebbe risultare accurata quanto la PINN. Inoltre, in questo caso si potrebbe

avere un effetto ridondante del temine fisico, rendendo così troppo costoso addestrare una rete PINN a fronte di un risultato potenzialmente peggiore rispetto alla rete priva dell'informazioni fisiche. Dalla *Figura 14* è possibile notare come il tempo impiegato per l'addestramento per la PINN cresce più velocemente rispetto al tempo impiegato per la NN, a parità di numero di strati. Analogamente la loss function risulta quasi traslata verso l'alto, questo dovuto alla presenza del residuo della PDE. Interessante è l'analisi della loss per quanto riguarda gli spostamenti e le deformazioni. Per quanto riguarda le componenti del tensore di deformazione, sia le NN che le PINN mostrano tendenza di convergenza sovrapponibili, raggiungendo un valore compreso tra 10^{-5} e 10^{-4}. Tale sovrapposizione si spiega considerando che la PINN integra direttamente nel processo di ottimizzazione i residui dell'equazione differenziale dell'elasticità lineare. Poiché le tensioni sono legate al campo di spostamento tramite le derivate prime, la regolarizzazione fisica esercitata dalla PDE garantisce che le derivate calcolate dalla PINN approssimino le deformazioni con una precisione paragonabile a quella del modello NN. A differenza delle deformazioni, le componenti di spostamento mostrano una variazione quasi costante, attribuibile alla diversa struttura della funzione costo. Nella PINN, si introduce un vincolo di ammissibilità fisica globale che bilancia l'ottimizzazione sui nodi. L'integrazione spaziale necessaria per risalire dalle derivate agli spostamenti assoluti rende la PINN lievemente meno rigida nella ricostruzione del valore puntuale del campo di spostamento, favorendo tuttavia una soluzione globale fisicamente coerente e priva di sovrastima delle rigidezze globali (*Codice* PINN in appendice).

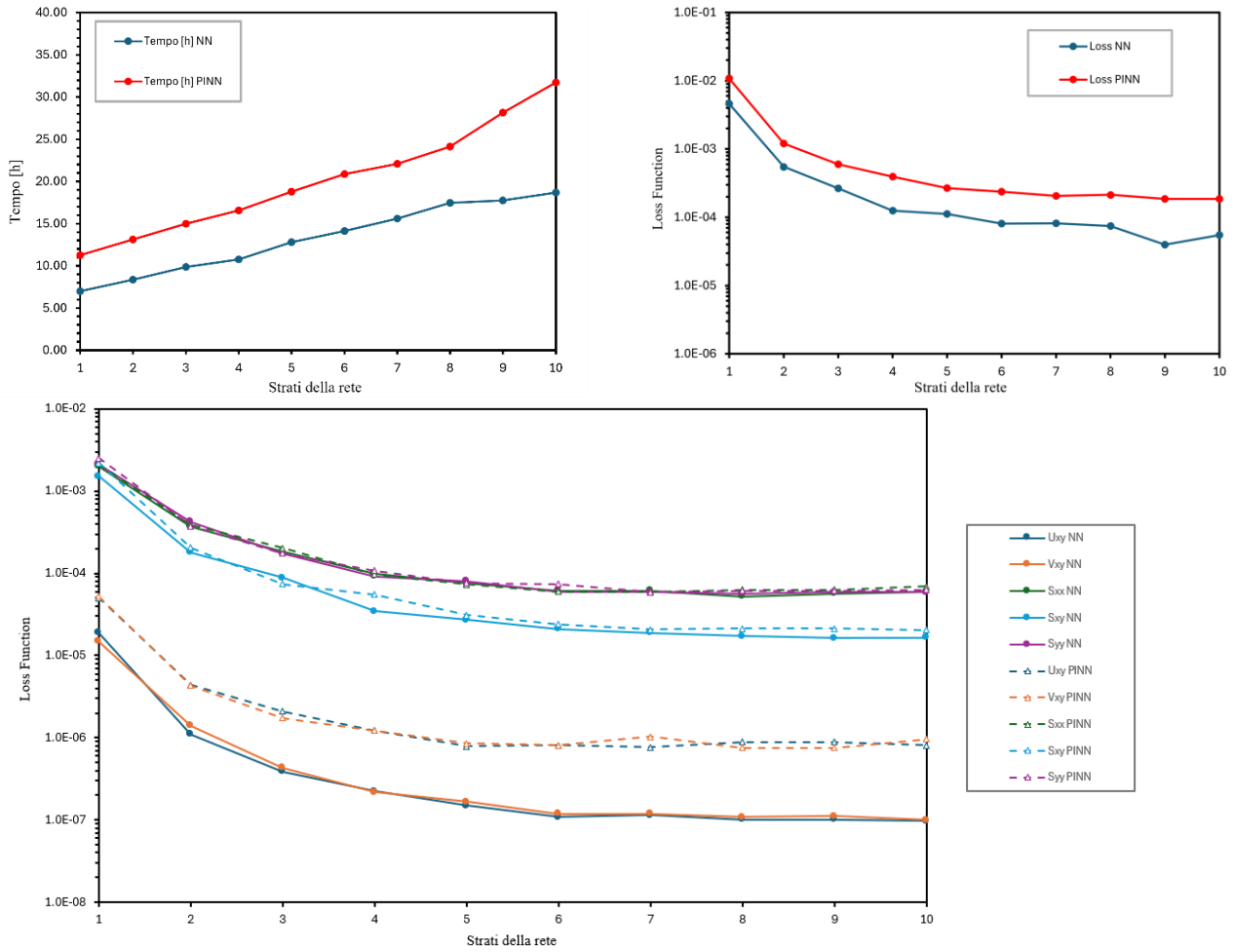


Figura 14 - Confronto Tempi e Loss tra PINN e NN

4.2. Validazione

Un importante parametro da valutare tra PINN e NN è l'accuratezza del modello. Per valutarla, vengono impiegati due temini di errore, e_d definito come la somma degli errori provenienti dal data-set, mentre e_r dalla somma di tutti i residui della loss senza forze di campo

$$\begin{aligned} e_d &= |\hat{u}_x - \hat{u}_x{}^*| + |\hat{u}_y - \hat{u}_y{}^*| + |\hat{\sigma}_{xx} - \hat{\sigma}_{xx}{}^*| + |\hat{\sigma}_{xy} - \hat{\sigma}_{xy}{}^*| + |\hat{\sigma}_{yy} - \hat{\sigma}_{yy}{}^*| \\ e_r &= \left|\left(\hat{\varepsilon}_{xx} + \nu\hat{\varepsilon}_{yy}\right)\frac{E}{1-\upsilon^2} - \hat{\sigma}_{xx}\right| + \left|\left(\nu\hat{\varepsilon}_{xx} + \hat{\varepsilon}_{yy}\right)\frac{E}{1-\upsilon^2} - \hat{\sigma}_{yy}\right| + \left|\frac{E}{1-\upsilon^2}\hat{\varepsilon}_{xy} - \hat{\sigma}_{xy}\right| \end{aligned} \quad . \tag{4.1}$$

Grazie a questi due termini, è stato possibile ottenere il grafico in *Figura 15*. Sia per PINN che per NN è stato determinato e_d ed e_r per 500 geometrie: da come è possibile notare, la rete con informazione sulla fisica risulta essere più accurata in fase di ottimizzazione, generando oltretutto un set di punti meno dispersivi. Questa differenza può essere enfatizzata nel caso in cui il data set di addestramento risulti essere composto da pochi dati, la PDE vincola il rispetto della fisica in tutti i punti della griglia, dando luogo ad un modello più accurato. È possibile anche confrontare le due reti in funzione del numero di geometrie di addestramento impiegate. Infatti, tali influenzano gli errori che possono essere commessi dalle reti stesse. Se il set di dati è caratterizzato da un numero elevato, il termine e_d potrebbe essere simile per entrambi le tipologie. Questo sempre per il fatto che una rete neurale semplice, dipendendo esclusivamente dal data-set di addestramento, può fornire risultati simili se non migliori di una rete con informazione fisica, in quanto risulterebbe sufficientemente addestrata e il vincolo della fisica non genererebbe dei miglioramenti in fase di ottimizzazione, con il rischio di ottenere un effetto ridondante al fine del risultato finale. Contrariamente, per e_r si dovrebbe assistere ad un miglioramento significativo, dovuto proprio al vincolo fisico imposto dall'equazione dell'elasticità lineare. Analizzando la *Figura 15*, si può vedere che l'errore dovuto ai parametri del data-set tende ad essere simile, a significare che il set di addestramento è caratterizzato da un numero elevato di parametri; invece, l'errore che caratterizza la fisica del problema invece risulta essere inferiore nel momento in cui si introduce la PDE all'interno del modello.

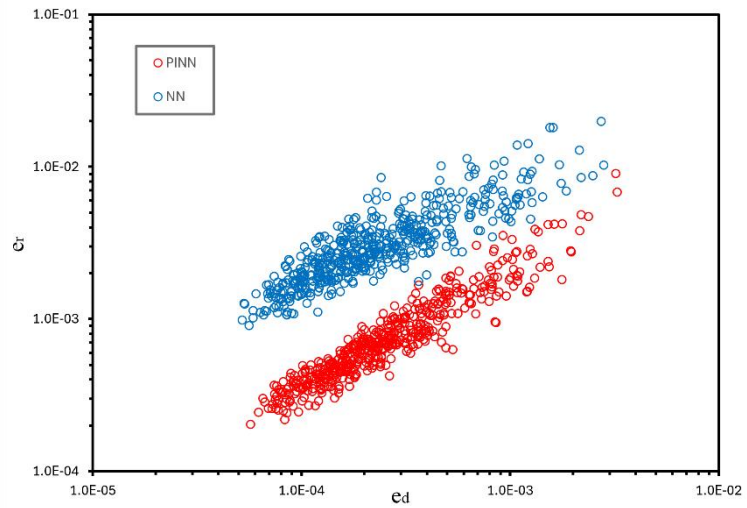


Figura 15 - Accuratezza di PINN e NN

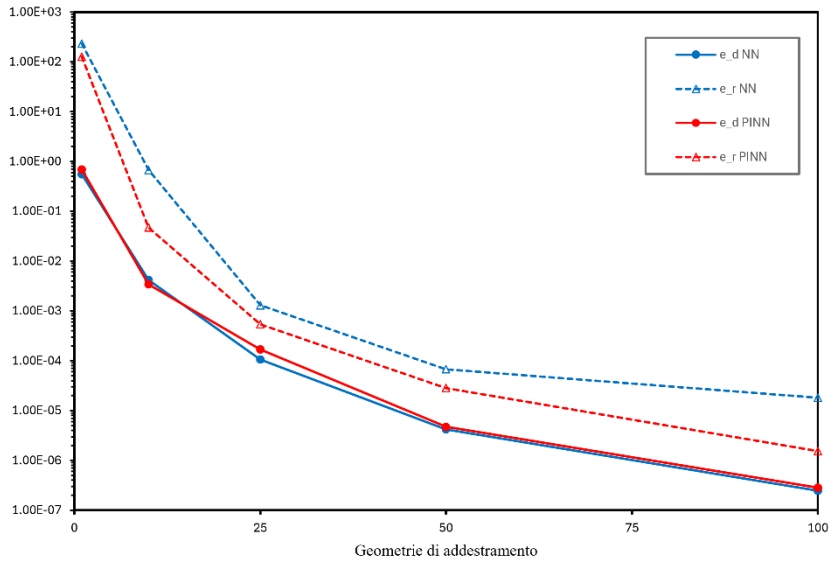


Figura 16 - Valutazione degli errori per PINN e NN a seconda del numero di geometrie di addestramento

I valori ottenuti in *Figura 16* sono stati calcolati applicando la seguente definizione

$$\begin{aligned} \bar{e}_d &= \frac{1}{n}\sum_{i=1}^{n} e_{d,i}^2 \\ \bar{e}_r &= \frac{1}{n}\sum_{i=1}^{n} e_{r,i}^2 \end{aligned} \quad . \tag{4.2}$$

Dato che le reti sono state addestrate con lo stesso data-set (sufficientemente ampio), l'errore $\bar{e}_d$ risulta simile per entrambe, a differenza di $\bar{e}_r$. Aumentando il numero di geometrie, PINN presenta un andamento migliore rispetto a NN: questo risultato è importante, in quanto il vincolo della fisica impostata in PINN fa sì che si generi un modello surrogato che fornisca un risultato più accurato. Impiegando cento geometrie di training, $\bar{e}_r$ per PINN vale $1.537 \cdot 10^{-6}$, mentre per NN si ottiene $1.815 \cdot 10^{-5}$. Un'ulteriore valutazione dell'errore viene effettuata visualizzando l'errore puntale commesso dal modello, analizzando sia gli spostamenti u_x e u_y sia le tensioni σ_{xx}, σ_{xy} e σ_{yy}. Dalla *Figura 17* è possibile intuire come, focalizzando sul profilo del foro di una generica geometria, sia NN che PINN generano un errore molto simile. Per le tensioni la differenza è quasi impercettibile. Questo motivo è dovuto all'elevato numero di geometrie in addestramento: infatti, si ricorda che una NN può fornire risultati simili se non migliori rispetto a PINN, anche senza vincolare quindi la fisica, se la rete risulta sufficientemente addestrata (nel caso, 100 geometrie di addestramento). Il problema sussiste quando il data-set di addestramento non è sufficientemente ampio oppure è costituito da dati imprecisi: è qui che PINN diventa più performante. Per capire meglio, sono stati ottenuti i medesimi grafici andando però ad impiegare una sola geometria di addestramento, estremizzando il concetto. Dalla *Figura 18* l'errore degli spostamenti u_x e u_y e delle tensioni σ_{xx}, σ_{xy} e σ_{yy} è inferiore per PINN, confermando che vincolare la fisica al problema quando si hanno poche informazioni è una scelta necessaria. Consente di diminuire gli errori, a fronte di un costo computazionale superiore. Da notare che aumentando le geometrie di addestramento da una a cento diminuisce di due ordini di grandezza l'errore commesso rispetto agli spostamenti.

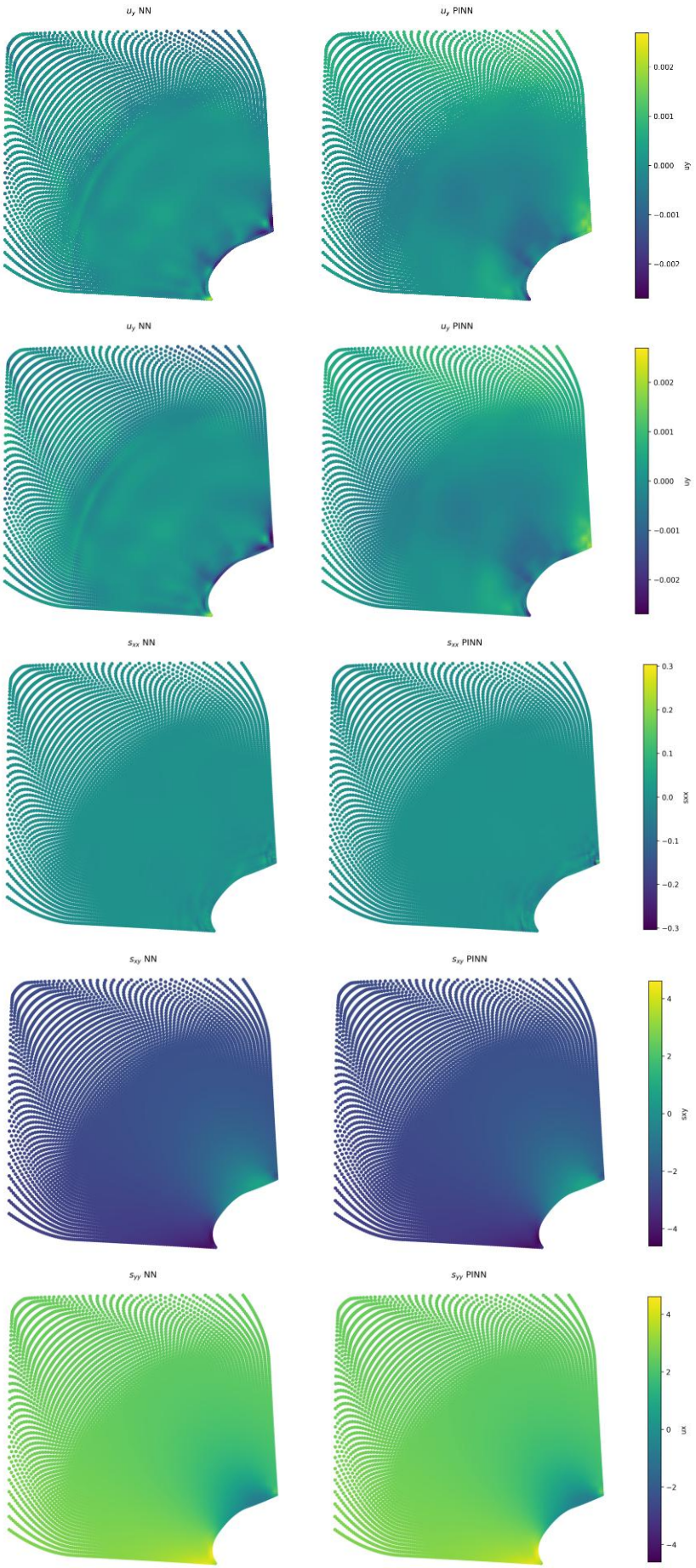


Figura 17 - Confronto errore spostamenti e tensioni per 100 geometrie di addestramento

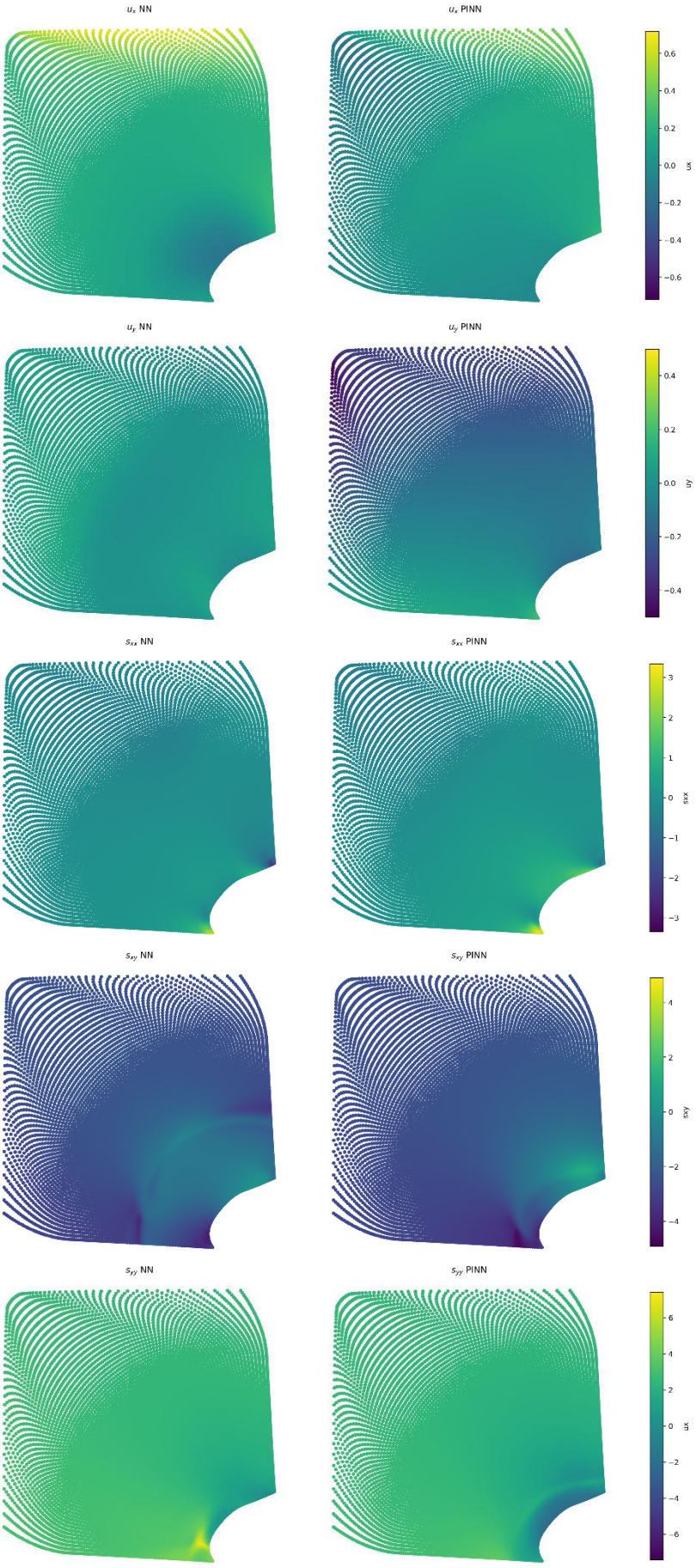


Figura 18 - Confronto errore spostamenti e tensioni per una geometria di addestramento

4.3. Ottimizzazione di forma

Sia per le NN che per le reti PINN sono state utilizzate 100 forme differenti caratterizzate dallo stesso volume, pari a 9600 (la figura per intera ha dimensioni 100 × 100, dunque con un volume massimo pari a 10000). Le forme del dataset sono state generate tramite analisi isogeometrica, impiegando cinque punti di controllo (otto parametri di forma) per la parametrizzazione tramite NURBS. I valori ottimizzati di Compliance e Volume sono stati ottenuti applicando un criterio di stop di 50 iterazioni o una tolleranza di 10^{-10}. Per entrambi vengono determinati gli errori relativi, dove per la compliance l'errore viene calcolato sia rispetto al valore ottimizzato tramite soluzione IGA che tramite soluzione analitica, mentre per il volume l'errore è calcolato rispetto al valore di riferimento (per questa ricerca è pari a 9600). Le relazioni sono le seguenti

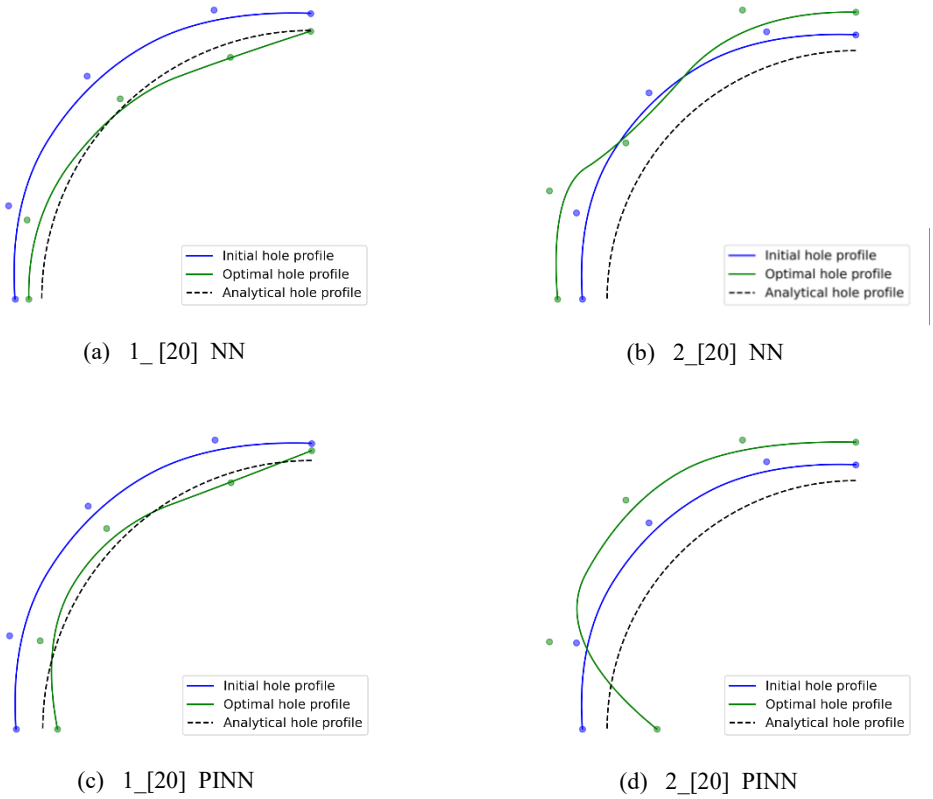
$$\begin{aligned} err_{C.NN-IGA} &= \left(\frac{C_{NN}}{C_{IGA}} - 1\right) \cdot 100 \\ err_{C.NN-Sol} &= \left(\frac{C_{NN}}{C_{Sol}} - 1\right) \cdot 100 \\ err_{V.NN-V} &= \left(\frac{V_{NN}}{V} - 1\right) \cdot 100 \end{aligned} \tag{4.3}$$

Per quanto riguarda PINN, cambia esclusivamente il pedice dei termini. La determinazione degli errori relativi è utile per stabilire quale rete presenta il miglior compromesso tra i risultati ottenuti in fase di addestramento e quelli ottenuti per l'ottimizzazione di forma. La rete selezionata sarà sottoposta ad una successiva fase di validazione. Vengono riportate in *Tabella 2* i valori di Compliance e Volume in funzione del numero di strati:

NN	**Compliance**							**Volume**			
N° strati	$C_{IC,NN}$	$C_{IC,IGA}$	C_{NN}	C_{IGA}	C_{Sol}	$err_{C.NN-IGA}$	$err_{C.NN-Sol}$	$V_{IC,NN}$	V_{NN}	V	$err_{V.NN-V}$
1	494.28	479.93	468.18	467.05	466.57	0.2410%	0.3445%	9501.34	9599.96	9600.00	-0.0004%
2	466.93		460.61	487.47		-5.5102%	-1.2773%		9451.39		-1.5480%
3	471.37		465.15	467.10		-0.4164%	-0.3035%		9599.77		-0.0024%
4	473.41		465.81	466.83		-0.2189%	-0.1638%		9599.78		-0.0023%
5	472.27		466.38	469.08		-0.5760%	-0.0405%		9599.81		-0.0020%
6	480.00		464.93	466.77		-0.3938%	-0.3516%		9599.89		-0.0012%
7	494.28		448.08	466.74		-3.9973%	-3.9625%		9599.93		-0.0007%
8	471.60		463.03	467.55		-0.9680%	-0.7593%		9599.63		-0.0038%
9	475.73		463.16	466.80		-0.7779%	-0.7301%		9599.85		-0.0015%
10	471.14		457.83	466.96		-1.9565%	-1.8740%		9599.95		-0.0005%
PINN	**Compliance**							**Volume**			
N° strati	$C_{IC,PINN}$	$C_{IC,IGA}$	CP_{PINN}	C_{IGA}	C_{Sol}	$err_{C.PINN-IGA}$	$err_{C.PINN-Sol}$	$V_{IC,PINN}$	V_{PINN}	V	$err_{V.PINN-V}$
1	481.2006	479.93	466.32	467.28	466.57	-0.2059%	-0.0542%	9501.34	9599.79	9600.00	-0.0022%
2	466.1224		459.37	492.06		-6.6441%	-1.5442%		9444.70		-1.6177%
3	483.366		466.38	466.89		-0.1088%	-0.0403%		9599.88		-0.0013%
4	477.45		466.67	466.82		-0.0330%	0.0206%		9599.87		-0.0013%
5	476.1112		465.64	467.18		-0.3310%	-0.1998%		9599.73		-0.0029%
6	478.5066		466.28	466.70		-0.0904%	-0.0631%		9599.73		-0.0028%
7	482.946		466.27	466.82		-0.1191%	-0.0648%		9599.86		-0.0014%
8	477.3399		466.97	466.84		0.0264%	0.0850%		9599.73		-0.0028%
9	478.2142		467.77	471.02		-0.6884%	0.2580%		9599.98		-0.0002%
10	475.9914		466.17	466.74		-0.1235%	-0.0867%		9599.98		-0.0002%

Tabella 2 - Compliance e Volume in funzione del numero di strati NN e PINN

Interessante notare come l'unica struttura che non è in grado di ottimizzare la geometria è quella composta con due strati. Analizzando la *Tabella 2*, si può vedere che ottimizzando con quella struttura neurale, il volume della figura tende a diminuire: infatti, la compliance è distante rispetto al valore determinato tramite IGA, ma non è sinonimo di calcolo andato in divergenza. La NN con sette strati presenta un errore relativo per la compliance molto elevato rispetto agli altri casi, ma con un errore relativo del volume ottimizzato che è quasi tre ordini di grandezza inferiore rispetto alla rete con due strati, stando a significare che la rete ha catturato correttamente l'andamento della fisica. Invece nel caso precedente la rete è incapace di seguire il problema, dovuto da una condizione di underfitting. Ad esempio, scegliendo un singolo strato, la rete risulta talmente semplice che applica una mappatura dei punti di addestramento diretta tra input e output, consentendo di forzare il profilo verso la soluzione analitica, ma non di generare una similitudine. Aumentando a due il numero di strati, la rete acquisisce una capacità superiore nell'andare ad approssimare le derivate, ma generando dei gradienti molto elevati. In fase di ottimizzazione, questi gradienti fanno si che si creino delle deformazioni errate. Dunque, questa struttura neurale rappresenta un punto di massima criticità per la convergenza, facendo sì che questa configurazione possa essere non considerata ai fini di una valutazione finale al fine di ottenere una rete ottimale per il problema (*Figura 19*). Aumentando ulteriormente il numero di strati (e dunque tre o superiore), i gradienti generati risultano notevolmente minori, garantendo una geometria del profilo ottimizzato più simile alla soluzione analitica. Lo stesso risultato è possibile estenderlo anche per le reti PINN. Il prossimo step è quello di scegliere quali tra le reti addestrate risulta fornisca il miglior compromesso tra i risultati ottenuti e i tempi di addestramento, oltre alla valutazione qualitativa del profilo ottenuto. Dalla *Tabella 2* è possibile



(a) 1_ [20] NN (b) 2_[20] NN

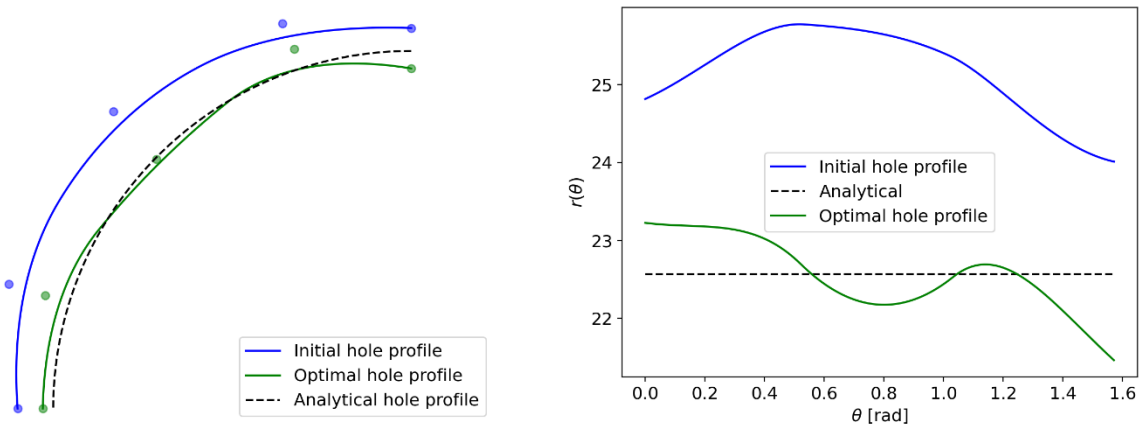
(c) 1_[20] PINN (d) 2_[20] PINN

Figura 19 - Confronto tra profili ottimizzati

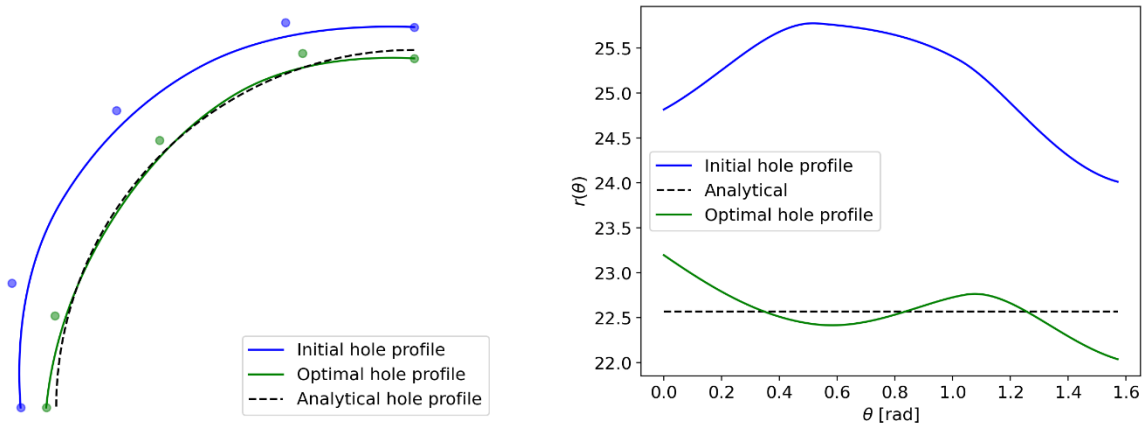
visualizzare l'errore relativo commesso dalla rete nel prevedere la compliance rispetto a quella calcolata rispetto a quella ottenuta tramite analisi isogeometrica IGA. Infatti, compliance IGA è essenziale per comprendere se la rete neurale è in grado di prevedere il profilo del foro: se il valore ottenuto è inferiore a quello iniziale (essendo la geometria iniziale uguale per tutte le reti, allora anche la compliance iniziale è uguale per tutte), allora è possibile affermare che il modello è predittivo. L'unica rete che non risulta essere utilizzabile è quella costituita da due strati, a conferma di ciò che è stato spiegato precedentemente. Innanzitutto, è interessante notare come l'errore relativo commesso dalle PINN sia inferiore rispetto a quelle delle NN. Seppur gli addestramenti siano stati effettuati con un numero consistente di geometrie, l'introduzione della PDE nella rete consente una previsione più accurata, a costo di un tempo di addestramento superiore. Infatti, prendendo come esempio una rete con sei strati, introducendo la PDE si ottiene una riduzione dell'errore di oltre il 77%, a fronte di un aumento del tempo di addestramento del 48% (passando da 14.12 h a 20.87 h). Al fine di ottenere la struttura impiegata in fase di validazione, vengono scelte le tre che hanno ottenuto l'errore relativo inferiore (in valore assoluto): in particolare, sono le strutture caratterizzate da quattro, sei ed otto strati di neuroni, dove quest'ultima ritorna il valore inferiore. Oltre la compliance, bisogna considerare anche come approssimano il volume finale rispetto alla soluzione (che corrisponde a 9600). La rete con quattro strati presenta un errore relativo inferiore rispetto alle altre due, 0.0013% rispetto a 0.0028% (in valore assoluto). Oltre ai profili ottenuti, per confrontare i risultati sono stati implementati dei grafici polari utili a comprendere quanto il profilo ottimizzato si discosta da quello analitico. Innanzitutto, si determina la soluzione analitica del problema

$$\pi r_{sol}^2 = 4(V_{tot} - V) \;\Rightarrow\; r_{sol} = \sqrt{\frac{4(V_{tot} - V)}{\pi}} = 22.57 \quad . \qquad (4.4)$$
$$V_{tot} = 10000, V = 9600$$

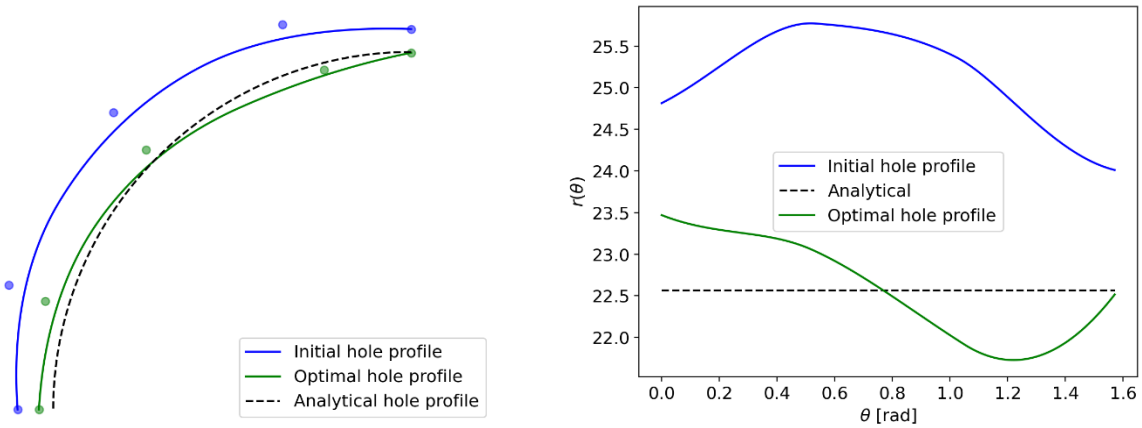
Analizzando il caso *a* (*Figura 20*), seppur fornisca i migliori risultati dal punto di vista dell'errore, il profilo ottimizzato presenta uno scostamento tra il massimo e il minimo raggio piuttosto notevole. Il raggio oscilla tra 21.46 e 23.23, il che significa un errore massimo relativo al raggio analitico pari a 4.89%. Per quanto riguarda il caso *b*, si può notare che la rete è in grado di far seguire il profilo rispetto a quello analitico in modo più preciso rispetto al caso precedente. Per questo caso, il raggio massimo è pari a 23.20, il raggio minimo invece 22.04, con un errore relativo pari a 2.78%, quasi la metà. Nell'ultimo caso, si ottiene un raggio massimo di 23.47 e uno minimo di 21.73, con un errore relativo massimo del 4.00%. A seguito dei valori riportati, la rete con sei strati risulta essere la migliore struttura nel fornire un profilo ottimizzato, oltretutto con un errore di calcolo contenuto. Analizzando i tempi di addestramento, un aumento del numero di strati da quattro a sei comporta un aumento del tempo di addestramento di circa il 26%, ma con una riduzione dell'errore massimo sul raggio del profilo del 44%, il che può giustificare l'utilizzo di questa struttura. La rete dotata di otto strati viene scartata in quanto non consente un miglioramento del profilo del foro, con un impiego del tempo per l'addestramento elevato (45% più elevato rispetto alla rete con quattro strati).



(a) 4_[20] PINN



(b) 6_[20] PINN



(c) 8_[20] PINN

Figura 20 - Profili ottimizzati con i relativi diagrammi polari

La *Figura 21* mostra la configurazione ottimizzata del quarto di piastra forata ottenuta al termine della procedura di ottimizzazione di forma. In particolare, il profilo del foro risultante dall'ottimizzazione numerica è rappresentato mediante una linea continua nera, mentre la linea tratteggiata rappresenta il corrispondente profilo analitico di riferimento. Il confronto tra le due geometrie permette di valutare la corrispondenza tra la soluzione ottenuta mediante il modello numerico e quella prevista dalla soluzione analitica. Come si può osservare, il profilo ottimizzato segue complessivamente l'andamento del profilo analitico, evidenziando una buona capacità della procedura di ottimizzazione di modificare la geometria iniziale in funzione dell'obiettivo considerato. Le eventuali differenze locali tra le due curve possono essere ricondotte alla discretizzazione della geometria mediante i punti di controllo e al processo numerico utilizzato per determinare i parametri di forma ottimali. Il confronto grafico costituisce pertanto un ulteriore elemento di valutazione della soluzione ottenuta, affiancando il confronto quantitativo dei valori della compliance e del volume riportato nelle analisi precedenti.

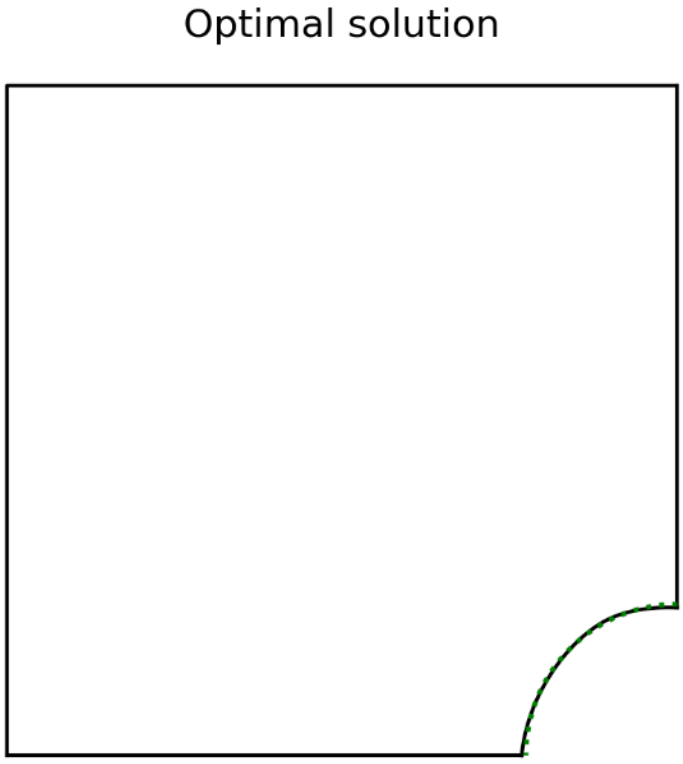


Figura 21 - Quarto di piastra con profilo ottimizzato mediante la rete PINN a sei strati. La linea continua nera rappresenta il profilo ottenuto dall'ottimizzazione, mentre la linea tratteggiata rappresenta il profilo analitico di riferimento.

CONCLUSIONI

In questo lavoro di tesi è stato sviluppato e validato un codice numerico in ambiente Python per la risoluzione di problemi di ottimizzazione di forma in ambito meccanico, integrando l'uso di reti neurali PINN e la parametrizzazione geometrica tramite curve NURBS. L'approccio proposto ha permesso di sostituire la risoluzione esplicita delle equazioni differenziali dell'elasticità lineare ad ogni iterazione dell'algoritmo di ottimizzazione con un modello surrogato basato su rete neurale, riducendo drasticamente il tempo computazionale richiesto nella fase di ricerca della forma ottimale. L'indagine svolta sul problema test della piastra forata ha consentito di evidenziare importanti proprietà relative al confronto tra reti puramente guidate dai dati (NN) e reti guidate dalla fisica (PINN):

1. Impatto del dataset e delle PINN: quando il dataset di addestramento è ridotto (es. poche geometrie disponibili), l'introduzione del residuo della PDE all'interno della loss function si rivela fondamentale. La rete PINN vincola il campo di spostamenti e tensioni al rispetto delle leggi dell'elasticità lineare, garantendo un errore di calcolo sensibilmente inferiore e una superiore capacità di generalizzazione rispetto a una rete NN convenzionale. All'aumentare del numero di geometrie di addestramento (fino a 100), la rete NN riduce progressivamente il proprio errore fino a raggiungere prestazioni confrontabili con la PINN, sebbene la PINN mantenga un errore relativo sulla *compliance* inferiore (riduzione dell'errore superiore al 77% per la configurazione a 6 strati).
2. Influenza della profondità della rete: l'analisi della profondità dell'architettura ha dimostrato che reti troppo superficiali, soffrono di gravi problemi di underfitting o instabilità dei gradienti, rendendo impossibile la corretta convergenza verso il profilo ottimale (con il caso a 2 strati che presenta la perdita del vincolo di volume). Al contrario, l'aumento eccessivo degli strati (es. 8 strati) porta a tempi di addestramento sproporzionati a fronte di un guadagno marginale o addirittura nullo in termini di accuratezza geometrica.
3. Configurazione Ottimale: la struttura di rete con 6 strati da 20 neuroni si è dimostrata il miglior compromesso tra costo computazionale e accuratezza del risultato. L'ottimizzazione del foro tramite questa configurazione PINN ha restituito un profilo con un errore relativo massimo sul raggio del 2,78% rispetto alla soluzione analitica e un controllo del vincolo di volume estremamente preciso (errore relativo dello 0,0028%).

APPENDICE

# Name	Version	# Name	Version	# Name	Version
_tflow_select	2.3.0	kiwisolver	1.4.7	pyparsing	3.3.2
abseil-cpp	20211102	latexcodec	3.0.1	pysocks	1.7.1
absl-py	2.1.0	libcurl	8.14.1	python	3.9.0
aiodns	3.5.0	libpng	1.6.56	python-dateutil	2.9.0.post0
aiohappyeyeballs	2.6.1	libprotobuf	3.20.3	python-flatbuffers	24.3.25
aiohttp	3.12.15	libssh2	1.10.0	pytz	2026.2
aiosignal	1.4.0	libuv	1.52.0	pyyaml	6.0.3
astunparse	1.6.3	libzlib	1.3.1	re2	2022.04.01
async-timeout	5.0.1	llvmlite	0.43.0	requests	2.32.5
attrs	25.4.0	lz4-c	1.9.4	requests-oauthlib	2.0.0
blas	1.0	m2w64-gcc-libgfortran	5.3.0	rsa	4.7.2
brotlicffi	1.0.9.2	m2w64-gcc-libs	5.3.0	sciann	0.7.0.1
c-ares	1.34.6	m2w64-gcc-libs-core	5.3.0	scikit-learn	1.6.1
ca-certificates	2026.5.14	m2w64-gmp	6.1.0	scipy	1.13.1
cachetools	5.5.1	m2w64-libwinpthread-git	5.0.0.4634.697f757	setuptools	80.9.0
certifi	2025.10.5	markdown	3.8	six	1.17.0
cffi	2.0.0	markupsafe	3.0.2	snappy	1.2.2
charset-normalizer	3.4.4	matplotlib	3.9.4	sqlite	3.51.2
click	8.1.8	mkl	2023.1.0	tbb	2021.8.0
colorama	0.4.6	mkl-service	2.4.0	tensorboard	2.10.0
contourpy	1.3.0	mkl_fft	1.3.11	tensorboard-data-server	0.6.1
cryptography	41.0.3	mkl_random	1.2.8	tensorboard-plugin-wit	1.8.1
cycler	0.12.1	mpi	1	tensorflow	2.10.0
flatbuffers	25.9.23	mpi4py	4.0.3	tensorflow-base	2.10.0
fonttools	4.60.2	msmpi	10.1.1	tensorflow-estimator	2.10.0
frozenlist	1.8.0	msys2-conda-epoch	20160418	termcolor	2.1.0
gast	0.4.0	multidict	6.7.0	threadpoolctl	3.6.0
geomdl	5.3.1	numba	0.60.0	typing-extensions	4.15.0
giflib	5.2.2	numpy	1.26.4	typing_extensions	4.15.0
google-auth	2.38.0	numpy-base	1.26.4	tzdata	2026.2
google-auth-oauthlib	0.4.4	oauthlib	3.3.1	ucrt	10.0.22621.0
google-pasta	0.2.0	openssl	1.1.1w	urllib3	2.5.0
grpc-cpp	1.48.2	opt_einsum	3.3.0	vc	14.3
grpcio	1.48.2	packaging	25	vc14_runtime	14.44.35208
h5py	3.14.0	pandas	2.3.3	vs2015_runtime	14.44.35208
hdf5	1.14.5	pillow	11.3.0	werkzeug	3.1.3
icc_rt	2022.1.0	pip	26.0.1	wheel	0.45.1
icu	58.2	propcache	0.3.1	win_inet_pton	1.1.0
idna	3.11	protobuf	3.20.3	winloop	0.2.2
importlib-metadata	8.5.0	pyasn1	0.6.1	wrapt	1.17.0
importlib-resources	6.5.2	pyasn1-modules	0.4.2	xz	5.8.2
intel-openmp	2023.1.0	pybind11-abi	5	yarl	1.18.0
joblib	1.5.3	pybtex	0.26.1	zipp	3.21.0
jpeg	9f	pycares	4.10.0	zlib	1.3.1
keras	2.10.0	pycparser	2.23	zstd	1.5.7
keras-preprocessing	1.1.2	pyopenssl	23.2.0		

Tabella 3 - Pacchetti impiegati per condurre le simulazioni (e le relative versioni)

```python
#import packages
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt
import sciann as sn
import pandas as pd
from numpy.random import seed
import random
import scipy
import os, sys, time
import shutil #to copy source file into result folder
import tensorflow as tf

#create output folder if it does not exist
if not os.path.isdir('./output'):
    os.mkdir('./output')

#delete content of output folder if it exists
for f in os.listdir('./output'):
    os.remove(os.path.join('./output', f))

wd = os.getcwd()    #get current working directory
shutil.copy(os.path.realpath(__file__), wd+'/output')    #copy source to output folder

#elasticity parameters
E = 210.
nu = 0.3

#import shape parameters
P = []
with open('./C1/CPs.txt', 'r') as arch:
    for line in arch:
        P.append(eval(line.rstrip()))

nr = np.shape(P)[0] #memorize number of rows in R_len
P_row = np.reshape(P,(8*nr),order='F') #write R on a single row

P1 = P_row[0:nr]         #vector containing values of first CP
P2 = P_row[nr:2*nr]      #vector containing values of second CP
P3 = P_row[2*nr:3*nr]      #vector containing values of third CP
P4 = P_row[3*nr:4*nr]      #vector containing values of fourth CP
P5 = P_row[4*nr:5*nr]      #vector containing values of fifth CP
P6 = P_row[5*nr:6*nr]      #vector containing values of sixth CP
P7 = P_row[6*nr:7*nr]      #vector containing values of seventh CP
P8 = P_row[7*nr:8*nr]      #vector containing values of eigth CP

P1 = (P1-np.min(P1))/(np.max(P1)-np.min(P1)) #normalize input P1
P2 = (P2-np.min(P2))/(np.max(P2)-np.min(P2)) #normalize input P2
P3 = (P3-np.min(P3))/(np.max(P3)-np.min(P3)) #normalize input P3
P4 = (P4-np.min(P4))/(np.max(P4)-np.min(P4)) #normalize input P4
P5 = (P5-np.min(P5))/(np.max(P5)-np.min(P5)) #normalize input P5
P6 = (P6-np.min(P6))/(np.max(P6)-np.min(P6)) #normalize input P6
P7 = (P7-np.min(P7))/(np.max(P7)-np.min(P7)) #normalize input P7
P8 = (P8-np.min(P8))/(np.max(P8)-np.min(P8)) #normalize input P8

#choose how many and which geometries use for training.
nds = 100 #number of training geometries
train_indx = np.arange(0,nds) #indices of training geometries. Cardinality must be nds
```

```
59
60  #build training dataset
61  num = str(train_indx[0]+1) #+1 because naming of datasets starts at 1, not 0
62  dataset1 = np.array(pd.read_csv('./C1/dataOmega'+num+'.csv')).astype(np.float32)
63  x_r = dataset1[:,0] #vector coordinates single geometry, used for dimensioning
64  N1 = np.size(x_r) #number of rows of a single geometry
65
66  dataset = dataset1 #initialize dataset
67  for i in range(1,nds):
68    num = str(train_indx[i]+1)
69
    dataset=np.append(dataset,np.array(pd.read_csv('./C1/dataOmega'+num+'.csv')).astype(np.float32))
70  dataset = np.reshape(dataset,(N1*nds,17))
71
72  #coordinates in parametric domain, do not change with physical shape
73  x_data = dataset[:,0]
74  y_data = dataset[:,1]
75  N = np.size(x_data)
76
77  #data on displacements and stresses
78  ux_data = dataset[:,4]
79  uy_data = dataset[:,5]
80  sigmaxx_data = dataset[:,9]
81  sigmayy_data = dataset[:,10]
82  sigmaxy_data = dataset[:,11]
83
84  #define body forces (if present)
85  fx_data = 0.
86  fy_data = 0.
87
88  #import data of inverse Jacobian
89  xix_data = dataset[:,12]
90  xiy_data = dataset[:,13]
91  etax_data = dataset[:,14]
92  etay_data = dataset[:,15]
93
94  #define shape data, i.e. training data of the control points of training geometries
95  A1_data = P1[train_indx[0]]*np.ones_like(x_r)
96  for i in range(1,nds):
97    A1_data = np.append(A1_data,P1[train_indx[i]]*np.ones_like(x_r))
98
99  A2_data = P2[train_indx[0]]*np.ones_like(x_r)
100 for i in range(1,nds):
101   A2_data = np.append(A2_data,P2[train_indx[i]]*np.ones_like(x_r))
102
103 A3_data = P3[train_indx[0]]*np.ones_like(x_r)
104 for i in range(1,nds):
105   A3_data = np.append(A3_data,P3[train_indx[i]]*np.ones_like(x_r))
106
107 A4_data = P4[train_indx[0]]*np.ones_like(x_r)
108 for i in range(1,nds):
109   A4_data = np.append(A4_data,P4[train_indx[i]]*np.ones_like(x_r))
110
111 A5_data = P5[train_indx[0]]*np.ones_like(x_r)
112 for i in range(1,nds):
113   A5_data = np.append(A5_data,P5[train_indx[i]]*np.ones_like(x_r))
114
115 A6_data = P6[train_indx[0]]*np.ones_like(x_r)
116 for i in range(1,nds):
```

```
    A6_data = np.append(A6_data,P6[train_indx[i]]*np.ones_like(x_r))

A7_data = P7[train_indx[0]]*np.ones_like(x_r)
for i in range(1,nds):
    A7_data = np.append(A7_data,P7[train_indx[i]]*np.ones_like(x_r))

A8_data = P8[train_indx[0]]*np.ones_like(x_r)
for i in range(1,nds):
    A8_data = np.append(A8_data,P8[train_indx[i]]*np.ones_like(x_r))

######################
# Neural Network Setup.
######################
dtype='float32'

#define the variables of the neural network
x = sn.Variable("x", dtype=dtype)
y = sn.Variable("y", dtype=dtype)
A1 = sn.Variable("A1", dtype=dtype)
A2 = sn.Variable("A2", dtype=dtype)
A3 = sn.Variable("A3", dtype=dtype)
A4 = sn.Variable("A4", dtype=dtype)
A5 = sn.Variable("A5", dtype=dtype)
A6 = sn.Variable("A6", dtype=dtype)
A7 = sn.Variable("A7", dtype=dtype)
A8 = sn.Variable("A8", dtype=dtype)
xix = sn.Variable("xix", dtype=dtype)
xiy = sn.Variable("xiy", dtype=dtype)
etax = sn.Variable("etax", dtype=dtype)
etay = sn.Variable("etay", dtype=dtype)

#define the functionals of the neural network
Uxy = sn.Functional("Uxy", [x, y, A1, A2, A3, A4, A5, A6, A7, A8], 5*[20], 'tanh')
Vxy = sn.Functional("Vxy", [x, y, A1, A2, A3, A4, A5, A6, A7, A8], 5*[20], 'tanh')
Sxx = sn.Functional("Sxx", [x, y, A1, A2, A3, A4, A5, A6, A7, A8], 5*[20], 'tanh')
Syy = sn.Functional("Syy", [x, y, A1, A2, A3, A4, A5, A6, A7, A8], 5*[20], 'tanh')
Sxy = sn.Functional("Sxy", [x, y, A1, A2, A3, A4, A5, A6, A7, A8], 5*[20], 'tanh')

#compute elasticity parameters
C11 = E/(1-nu**2)
C12 = nu*E/(1-nu**2)
C33 = E*(1-nu)/(2*(1-nu**2))

#compute deformations (using chain rule) for residual computations
Exx = sn.diff(Uxy, x)*xix + sn.diff(Uxy, y)*etax
Eyy = sn.diff(Vxy, x)*xiy + sn.diff(Vxy, y)*etay
Gxy = sn.diff(Uxy, x)*xiy + sn.diff(Uxy, y)*etay + sn.diff(Vxy, x)*xix + sn.diff(Vxy, y)*etax

#compute residuals
c1 = sn.Tie(Sxx, Exx*C11 + Eyy*C12)
c2 = sn.Tie(Syy, Eyy*C11 + Exx*C12)
c3 = sn.Tie(Sxy, Gxy*C33)

#assign data
Du = sn.Data(Uxy)
Dv = sn.Data(Vxy)
Ds1 = sn.Data(Sxx)
Ds2 = sn.Data(Sxy)
Ds3 = sn.Data(Syy)
```

```
176
177 targets = [Du, Dv, Ds1, Ds2, Ds3, c1, c2, c3]
178
179 #define scimodel
180 m = sn.SciModel([x, y, A1, A2, A3, A4, A5, A6, A7, A8, xix, xiy, etax, etay], targets)
181
182 #define residual data (zero for zero residual)
183 c1_data = np.zeros_like(y_data)
184 c2_data = np.zeros_like(y_data)
185 c3_data = np.zeros_like(y_data)
186
187 #train model
188 h = m.train([x_data, y_data, A1_data, A2_data, A3_data, A4_data, A5_data, A6_data, A7_data,
    A8_data, xix_data, xiy_data, etax_data, etay_data], [ux_data, uy_data, sigmaxx_data,
    sigmaxy_data, sigmayy_data, c1_data, c2_data, c3_data], batch_size=1000, epochs=1200,
    learning_rate = ([0, 300, 700, 1100], [0.001, 0.0005, 0.0002, 0.0001]), log_parameters={},
    adaptive_weights={'method':'NTK', 'freq': 100}, verbose = 2)
189
190
191 ##########################################################################
192 ##########                END OF PINN MODEL                  ############
193 ##########     GENEREATION AND PRINTING OF RESULTS BEGINS     ############
194 ##########################################################################
195
196 #save the weights of the trained PINN
197 m.save_weights('./output/weights_PINN.ckpt')
198 #copy output (if it was created) to output folder
199 #shutil.copy('./terminal_output.txt', './output/terminal_output.txt')
200
201 #save the plot of the total loss function
202 fig = plt.figure()
203 new_plot = fig.add_subplot(111)
204 new_plot.plot(h.history['loss'], label='loss')
205 plt.yscale('log')
206 plt.legend()
207 fig.savefig('./output/loss.png')
208
209 #save the plots of the partial loss functions for displacements and stresses
210 fig = plt.figure()
211 new_plot = fig.add_subplot(111)
212 new_plot.plot(h.history['Uxy_loss'], label='ux loss')
213 plt.yscale('log')
214 plt.legend()
215 fig.savefig('./output/ux_loss.png')
216
217 fig = plt.figure()
218 new_plot = fig.add_subplot(111)
219 new_plot.plot(h.history['Vxy_loss'], label='uy loss')
220 plt.yscale('log')
221 plt.legend()
222 fig.savefig('./output/uy_loss.png')
223
224 fig = plt.figure()
225 new_plot = fig.add_subplot(111)
226 new_plot.plot(h.history['Sxx_loss'], label='Sxx loss')
227 plt.yscale('log')
228 plt.legend()
229 fig.savefig('./output/Sxx_loss.png')
230
```

```
231  fig = plt.figure()
232  new_plot = fig.add_subplot(111)
233  new_plot.plot(h.history['Sxy_loss'], label='Sxy loss')
234  plt.yscale('log')
235  plt.legend()
236  fig.savefig('./output/Sxy_loss.png')
237
238  fig = plt.figure()
239  new_plot = fig.add_subplot(111)
240  new_plot.plot(h.history['Syy_loss'], label='Syy loss')
241  plt.yscale('log')
242  plt.legend()
243  fig.savefig('./output/Syy_loss.png')
244
245  #save the plot of the normalized total loss function
246  loss_array = np.array(h.history['loss'])
247  norm_loss = loss_array/loss_array[0]
248  fig = plt.figure()
249  new_plot = fig.add_subplot(111)
250  new_plot.plot(norm_loss, label='normalized loss')
251  plt.yscale('log')
252  plt.legend()
253  fig.savefig('./output/normalized_loss.png')
254  plt.close('all')
255
```

Codice PINN impiegato per l'addestramento

REFERENZE

[1] Sokolowski J, Zolesio JP: Introduction to Shape Optimization, Springer Science & Business Media, 2012.

[2] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019; 378: 686-707.

[3] Haghighat E, Raissi M, Moure A, Gomez H, Juanes R. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Comput Methods Appl Mech Engrg 2021; 379: 113741.

[4] Jin X, Cai S, Li H, Karniadakis GE. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. J Comput Phys 2021; 426(109951): 1-26

[5] Li L, Li Y, Du Q, Liu T, Xie Y. ReF-nets: Physics-informed neural network for Reynolds equation of gas bearing. Comput Methods Appl Mech Engrg 2022; 391(114524): 1-26.

[6] Mao Z, Jagtap AD, Karniadakis GE. Physics-informed neural networks for high-speed flows. Comput Methods Appl Mech Engrg 2020; 360(112789): 1-26.

[7] Peng W, Pu J, Chen Y. PINN deep learning method for the Chen-Lee-Liu equation: Rogue wave on the periodic background. Commun Nonlinear Sci Numer Simul 2022; 105(106067): 1-15.

[8] Baydin A, Pearlmutter B, Radul A, Siskind J: "Automatic Differentiation in Machine Learning: a Survey", Journal of Machine Learning Research 18, 2018: 1-43.

[9] Allaire G, Dapogny C, Jouve F: "Chapter 1 - Shape and topology optimization", Handbook of Numerical Analysis, Vol. 22, Elsevier, 2021.

[10] Mohammadi B, Pironneau O: Applied shape optimization for fluids, Vol. 28, Oxford University Press, 2001.

[11] Mezzadri F, Gasick J, Qian X: "A Framework for Physics-Informed Deep Learning Over Freeform Domains", Computer-Aided Design, Vol. 160, 2023.

[12] van Rossum G: Workshop on Compositional Software Architecture in Monterey, California, 1998.

[13] Abadi M et al: "TensorFlow: a system for large-scale machine learning", OSDI'16: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, USENIX Association, 2016.

[14] Haghighat E, Juanes R: SciANN: A Keras/Tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks, 2020.