UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria Meccatronica

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Tesi di Laurea

PHYSICS INFORMED NEURAL NETWORK PER SISTEMI MECCANICI

Relatore: BOSCARIOL Paolo

Candidato: Alessandro MERLIN

Anno Accademico 2023-2024

Indice

Capitolo 1: Introduzione

L'ingegneria meccatronica rappresenta una delle discipline fondamentali dell'ingegneria, con un impatto trasversale in molteplici settori chiave dell'industria e della tecnologia moderna. Dai sistemi di produzione industriale alle macchine di precisione, dall'energia alle infrastrutture, l'ingegneria è al centro dello sviluppo e del progresso della nostra società. Tuttavia, con l'aumentare della complessità dei sistemi meccanici e delle sfide ingegneristiche, emergono nuove necessità di approcci innovativi e soluzioni avanzate.

In questo contesto, l'intelligenza artificiale (AI) e il machine learning hanno assunto un ruolo sempre più rilevante nell'ambito dell'ingegneria meccanica. Tra le molte tecniche di machine learning, le reti neurali profonde (DNN) si distinguono per la loro capacità di apprendere rappresentazioni complesse dei dati e di estrarre pattern significativi da grandi dataset. Questa capacità le rende particolarmente adatte per una vasta gamma di applicazioni, dalle previsioni di guasti alla manutenzione predittiva, dal controllo di processo alla progettazione ottimizzata.

L'obiettivo di questa tesi è esplorare e dimostrare il potenziale delle reti neurali profonde nell'ottimizzazione dei sistemi meccanici attraverso due applicazioni specifiche. Attraverso un'analisi dettagliata e sperimentazioni empiriche, si cerca di valutare l'efficacia delle DNN e di fornire insight preziosi per futuri sviluppi nell'ambito dell'ingegneria meccatronica.

La tesi è strutturata nel seguente modo: nel secondo capitolo, verrà fornita una panoramica dei fondamenti teorici delle reti neurali profonde, delineando le loro caratteristiche principali. Nel terzo capitolo, si esaminerà il contesto e le applicazioni delle reti neurali nei sistemi meccanici, evidenziando le sfide e le opportunità presenti in questo campo. Nel quarto capitolo, verrà spiegato il concetto di PINN e verranno esposte le applicazioni di questa tecnologia. I capitoli cinque e sei saranno dedicati alle due applicazioni specifiche, dove saranno esposti il contesto dell'applicazione, i dettagli della metodologia adottata e i risultati ottenuti. Infine, nel capitolo sette, verranno discussi i risultati delle analisi, confrontando le prestazioni dei modelli sviluppati e fornendo raccomandazioni per sviluppi futuri nell'ambito delle reti neurali per sistemi meccanici.

Capitolo 2: Fondamenti teorici delle reti neurali profonde

Le reti neurali profonde (DNN) rappresentano una classe di modelli di intelligenza artificiale ispirati al funzionamento del cervello umano, caratterizzati da una struttura complessa di più strati di neuroni artificiali. Questo capitolo si propone di fornire una panoramica approfondita dei principali concetti e delle caratteristiche delle reti neurali profonde.

2.1 Concetti fondamentali delle reti neurali

Introduzione al concetto di neurone artificiale:

Un neurone artificiale è l'unità base di una rete neurale, ispirato al funzionamento dei neuroni biologici nel cervello umano. Ogni neurone artificiale riceve input da altre unità o direttamente dai dati di input e produce un output sulla base di una funzione di attivazione.

Funzionamento dei percettroni:

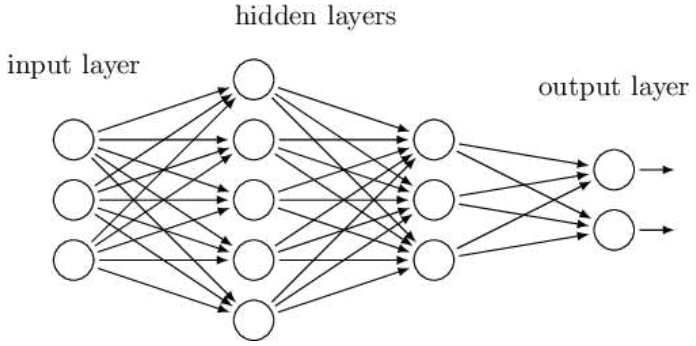
Il percettrone è una semplice forma di neurone artificiale che esegue una combinazione lineare dei suoi input, applica una funzione di attivazione e produce un output. La funzione di attivazione può essere di tipo binario o continua e determina se il neurone sarà attivato o meno in base al valore del suo input.

Una rete neurale profonda è composta da diversi strati di neuroni artificiali, ognuno dei quali ha un ruolo specifico nel processo di apprendimento e di elaborazione dei dati. I principali strati di una rete neurale profonda includono:

Strato di input: Questo strato riceve i dati di input grezzi o pre-elaborati e trasmette l'informazione ai livelli successivi della rete.

Strati nascosti: Questi strati intermedi contengono neuroni che eseguono operazioni di trasformazione e astrazione dei dati, apprendendo rappresentazioni sempre più complesse dei dati di input.

Strato di output: Questo strato produce l'output finale della rete neurale, che può essere una classificazione, una previsione o un'altra forma di risultato desiderato.



2.2 Architetture delle reti neurali profonde

Le reti neurali profonde possono assumere diverse architetture, ognuna delle quali ha caratteristiche specifiche che le rendono adatte a determinati compiti e contesti. In questo capitolo, esamineremo quattro delle principali architetture di reti neurali profonde: le reti neurali feedforward, le reti neurali completamente connesse, le reti neurali convoluzionali (CNN) e le reti neurali ricorrenti (RNN).

Reti neurali feedforward:

Le reti neurali feedforward sono un tipo di architettura di rete neurale in cui l'informazione fluisce in una direzione, dall'input verso l'output, senza cicli o connessioni retroattive. Questo tipo di architettura è caratterizzato da uno o più strati nascosti tra lo strato di input e lo strato di output, dove ogni neurone è connesso a tutti i neuroni dello strato successivo.

Reti neurali completamente connesse:

Le reti neurali completamente connesse sono il tipo più semplice di architettura di rete neurale profonda, in cui ogni neurone in uno strato è connesso a tutti i neuroni nello strato successivo. Questa struttura permette alla rete di apprendere relazioni complesse tra le feature di input, ma può essere soggetta a problemi di overfitting, soprattutto con dataset di grandi dimensioni. Le reti completamente connesse sono comunemente utilizzate in applicazioni di classificazione e regressione dove la struttura spaziale dei dati non è rilevante.

Reti neurali convoluzionali (CNN):

Le reti neurali convoluzionali sono progettate per lavorare con dati bidimensionali, come immagini, e sfruttano la struttura spaziale dei dati per apprendere pattern locali. Le CNN sono caratterizzate da strati di convoluzione, pooling e completamente connessi, che permettono loro di riconoscere pattern gerarchici su scale diverse. Questa architettura è particolarmente efficace nell'elaborazione di immagini e video, dove la località e la gerarchia delle feature sono cruciali.

Reti neurali ricorrenti (RNN):

Le reti neurali ricorrenti sono progettate per lavorare con dati sequenziali, come sequenze di testo, audio o serie temporali. A differenza delle reti neurali feedforward, le RNN contengono connessioni retroattive che consentono loro di memorizzare informazioni sullo stato precedente e di elaborare sequenze di lunghezza variabile. Questa architettura è ampiamente utilizzata in applicazioni di linguaggio naturale, traduzione automatica, analisi di serie temporali e molto altro.

2.3 Funzioni di attivazione e strumenti di ottimizzazione

Funzioni di attivazione:

Le funzioni di attivazione sono elementi chiave all'interno dei neuroni artificiali che introducono non linearità nel processo di apprendimento delle reti neurali. Queste funzioni determinano se un neurone sarà attivato o meno in base al valore del suo input. Alcune delle funzioni di attivazione più comuni includono la funzione sigmoidea, la funzione tangente iperbolica (tanh) e la funzione di attivazione lineare rettificata (ReLU). La ReLU è particolarmente popolare nelle reti neurali profonde poiché risolve problemi di scomparsa del gradiente e accelera il processo di apprendimento.

Algoritmi di ottimizzazione:

Gli algoritmi di ottimizzazione sono utilizzati per addestrare le reti neurali profonde, ovvero per regolare i pesi dei neuroni in modo che la rete possa apprendere dai dati e migliorare le sue prestazioni nel compito assegnato. Uno degli algoritmi di ottimizzazione più utilizzati è il gradiente discendente, che aggiorna i pesi della rete in base al gradiente della funzione di perdita rispetto ai pesi stessi. Questo processo viene iterato fino a quando la rete raggiunge una condizione di convergenza o un numero massimo di iterazioni. Varianti ottimizzate del gradiente discendente includono il gradiente discendente stocastico (SGD), l'Adaptive Moment Estimation (Adam), il Root Mean Square Propagation (RMSProp), tra gli altri. Queste varianti aggiungono meccanismi di adattamento del tasso di apprendimento o dell'aggiornamento dei pesi per migliorare l'efficienza e la stabilità del processo di apprendimento.

2.4 Tecniche di regolarizzazione e prevenzione dell'overfitting

Le reti neurali profonde possono essere soggette all'overfitting, un fenomeno in cui il modello impara troppo bene dai dati di addestramento e quindi si comporta male su dati non visti durante il test. Le tecniche di regolarizzazione sono utilizzate per contrastare questo problema, limitando la complessità del modello e migliorando la sua capacità di generalizzazione su dati non osservati. Alcune delle tecniche di regolarizzazione più comuni utilizzate nelle reti neurali profonde includono la regolarizzazione L1 e L2, il dropout e la normalizzazione del batch.

Regolarizzazione L1 e L2:

La regolarizzazione L1 e L2 sono tecniche utilizzate per limitare la complessità del modello aggiungendo un termine di regolarizzazione alla funzione di costo durante l'addestramento della rete. Il termine di regolarizzazione penalizza i pesi dei neuroni, impedendo loro di raggiungere valori troppo grandi. La regolarizzazione L1 aggiunge il valore assoluto dei pesi come termine di regolarizzazione, mentre la regolarizzazione L2 aggiunge il quadrato dei pesi. Queste tecniche aiutano a prevenire l'overfitting regolando la complessità del modello.

Dropout:

Il dropout è una tecnica di regolarizzazione utilizzata per prevenire l'overfitting disattivando casualmente un insieme di neuroni durante ciascuna iterazione dell'addestramento. Questo impedisce alle reti neurali di diventare dipendenti da specifiche combinazioni di neuroni e aumenta la robustezza del modello. Durante il test, tutti i neuroni sono attivi, ma i pesi sono ridotti in base alla probabilità di dropout per compensare l'attivazione casuale durante l'addestramento.

Normalizzazione del batch:

La normalizzazione del batch è una tecnica utilizzata per migliorare la stabilità e l'efficienza dell'addestramento delle reti neurali profonde. Consiste nel normalizzare i valori di input di ogni mini-batch durante l'addestramento, in modo che abbiano una media zero e una deviazione standard unitaria. Questo aiuta a evitare problemi di saturazione del gradiente e accelerare il processo di convergenza.

Queste tecniche di regolarizzazione svolgono un ruolo cruciale nel garantire che le reti neurali profonde siano in grado di generalizzare bene su nuovi dati e di evitare l'overfitting, migliorando così le loro prestazioni e la loro robustezza nell'ambito dell'ingegneria meccanica e oltre.

Capitolo 3: *Applicazioni delle reti neurali profonde nell'ingegneria meccanica*

Le reti neurali profonde hanno dimostrato di essere estremamente versatili e utili in una vasta gamma di applicazioni nell'ambito dell'ingegneria meccanica. Questo capitolo esplorerà diverse di queste applicazioni, evidenziando come le reti neurali profonde possano essere utilizzate per migliorare l'efficienza, l'affidabilità e le prestazioni dei sistemi meccanici.

Manutenzione predittiva:

Una delle applicazioni principali delle reti neurali profonde nell'ingegneria meccanica è la manutenzione predittiva. Utilizzando dati di sensori e di monitoraggio delle condizioni, le reti neurali possono essere addestrate per prevedere guasti e malfunzionamenti dei componenti meccanici prima che si verifichino effettivamente. Questo permette una pianificazione più efficiente della manutenzione e riduce i costi associati a fermi macchina non pianificati.

Diagnosi di guasti:

Le reti neurali profonde possono essere impiegate per diagnosticare guasti nei sistemi meccanici analizzando segnali di sensori, vibrazioni o altre informazioni diagnostiche. Le reti neurali possono apprendere a riconoscere modelli associati a guasti specifici e a identificare anomalie nei dati operativi, consentendo interventi tempestivi per prevenire danni o malfunzionamenti più gravi.

Controllo di processo:

Le reti neurali profonde possono essere utilizzate per migliorare il controllo dei processi industriali nei settori manifatturiero, chimico, energetico e altri. Le reti neurali possono apprendere modelli complessi di comportamento del processo e predire in modo accurato le risposte del sistema a diverse condizioni operative. Ciò consente un controllo più preciso e adattabile dei processi, migliorando l'efficienza e la qualità dei prodotti.

Progettazione ottimizzata:

Le reti neurali profonde possono essere impiegate per ottimizzare il processo di progettazione dei componenti meccanici. Attraverso tecniche di ottimizzazione basate su reti neurali, è possibile esplorare rapidamente il vasto spazio delle possibili configurazioni e identificare soluzioni ottimali o near-optimali per criteri specifici, come peso ridotto, resistenza migliorata o performance ottimizzate.

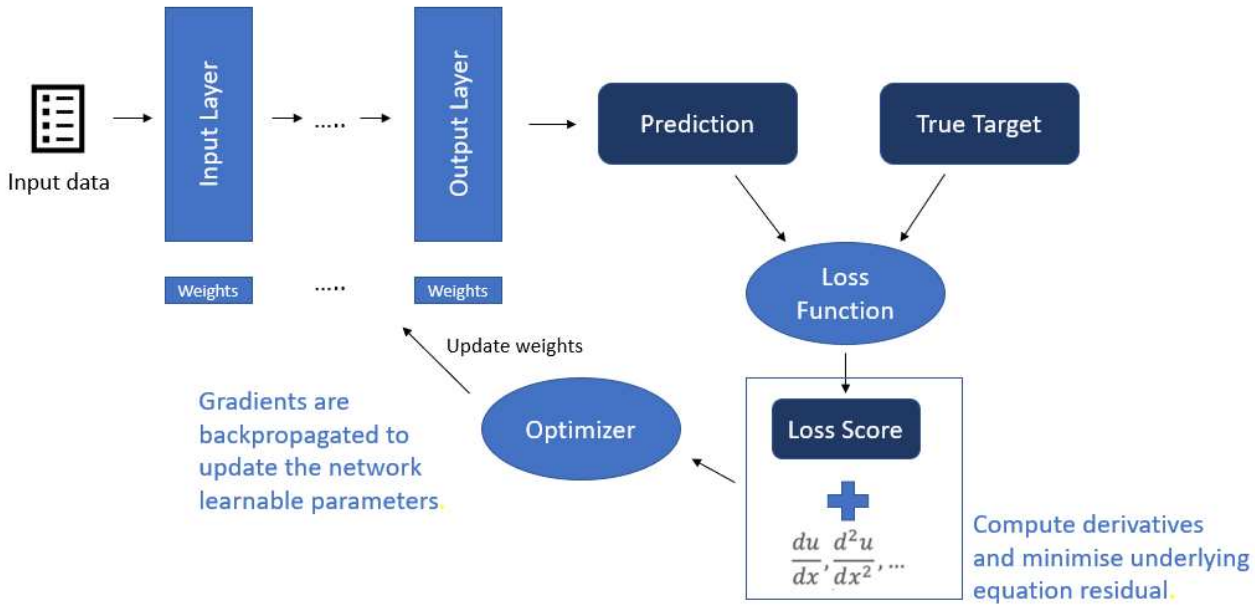
Queste sono solo alcune delle molte applicazioni delle reti neurali profonde nell'ambito dell'ingegneria meccanica. La loro flessibilità e potenza analitica offrono numerose opportunità per migliorare i processi e i prodotti nell'industria meccanica, contribuendo così a un progresso continuo e a una maggiore efficienza nel settore.

Capitolo 4: Physics Informed Neural Network (PINN)

Le Physics-Informed Neural Networks (PINN) rappresentano un'avanzata metodologia che combina i principi della fisica con le capacità di apprendimento delle reti neurali artificiali. Questo capitolo esplorerà in dettaglio il concetto di PINN, illustrando perché sono diventate uno strumento prezioso nel mondo dell'ingegneria moderna.

4.1 Concetto di PINN

Le PINN sono progettate per risolvere problemi in cui è necessario integrare le leggi della fisica all'interno del processo di apprendimento delle reti neurali. Questo significa che anziché affidarsi esclusivamente ai dati, come fanno le reti neurali tradizionali, le PINN incorporano conoscenze fisiche preesistenti nei loro modelli, garantendo così soluzioni consistenti con le leggi della fisica. Sono basate su un'idea concettualmente semplice: aggiungere le equazioni differenziali che descrivono un dato sistema fisico nel calcolo delle perdite (loss) durante la fase di apprendimento. Il processo di addestramento è iterativo. Ad ogni passaggio, i dati del set di addestramento vengono forniti in ingresso e passati attraverso i diversi strati della rete neurale fino a raggiungere quello di output, dove viene generata una predizione. Questa predizione viene confrontata con il valore atteso per l'input specifico, e viene calcolata la perdita (loss), utilizzando la funzione di perdita scelta per l'addestramento della rete. Questa perdita viene quindi utilizzata da un ottimizzatore per aggiornare i pesi della rete, al fine di ridurre la perdita e migliorare le prestazioni nella successiva iterazione. Ciò che distingue le PINN è il metodo di calcolo della perdita: oltre ai dati sperimentali, viene calcolata anche considerando le equazioni differenziali che descrivono il sistema in esame. Di conseguenza, la rete neurale, imparando a risolvere tali equazioni, acquisisce una comprensione scientifica del problema affrontato. Si può quindi descrivere il processo di training di una PINN nel seguente modo:



4.2 Utilità delle PINN nel mondo odierno

Le PINN offrono una serie di vantaggi e applicazioni significative nel panorama dell'ingegneria moderna:

Integrazione della conoscenza fisica:

Uno dei principali vantaggi delle PINN è la loro capacità di integrare conoscenze fisiche preesistenti nei processi di apprendimento automatico. Questo consente di sviluppare modelli più accurati e affidabili, soprattutto in scenari in cui i dati sono limitati o rumorosi.

Riduzione della dipendenza dai dati:

Le PINN riducono la dipendenza esclusiva dai dati di addestramento, consentendo di ottenere risultati affidabili anche con set di dati limitati o di bassa qualità. Questo è particolarmente utile in situazioni in cui la raccolta di dati è costosa, difficile o pericolosa.

Applicazioni in problemi complessi:

Le PINN trovano ampio impiego nella risoluzione di problemi complessi e multidimensionali, come la simulazione e la previsione di fenomeni fisici in ambienti reali. Questo include la modellazione di flussi fluidodinamici, la previsione del comportamento strutturale dei materiali e la progettazione ottimizzata di sistemi complessi.

Efficienza computazionale:

Nonostante la loro complessità concettuale, le PINN possono essere implementate in modo efficiente e scalabile, offrendo tempi di calcolo ragionevoli anche per problemi di grandi dimensioni. Questo le rende una scelta attraente per applicazioni in tempo reale e su larga scala.

Contributi all'innovazione tecnologica:

L'adozione delle PINN sta contribuendo significativamente all'innovazione tecnologica in una vasta gamma di settori, dall'ingegneria aerospaziale all'energia rinnovabile, dalla medicina alle scienze ambientali. Le soluzioni fornite dalle PINN stanno consentendo di superare sfide complesse e di sviluppare tecnologie più avanzate e sostenibili.

In sintesi, le PINN rappresentano una poderosa sintesi tra i principi della fisica e le capacità di apprendimento delle reti neurali, offrendo soluzioni innovative e affidabili per una vasta gamma di problemi ingegneristici. La loro crescente adozione e diffusione nel mondo odierno promette di rivoluzionare numerosi settori e di guidare l'innovazione tecnologica verso nuovi orizzonti.

Capitolo 5: PINN Moto di un proiettile

CODICE DI ADDESTRAMENTO E DEFINIZIONE DELLA RETE:

```
clc; clear all; close all;

% Parametri iniziali
mu = rand * 0.1; % Genera un valore casuale tra 0 e 0.1
num_samples = 200;

% Genera dati di addestramento
P0 = zeros(num_samples, 2); % Inizializza P0 con tutte le X = 0
P0(:, 2) = rand(num_samples, 1) * 10; % Assegna valori casuali alle Y
V0 = rand(num_samples, 2) * 10; % Velocità iniziali casuali
T = linspace(0, 1, num_samples); % Intervallo di tempo

% Simula le traiettorie per i dati di addestramento
X_train = zeros(num_samples, length(T));
Y_train = zeros(num_samples, length(T));
Vx_train = zeros(num_samples, length(T));
Vy_train = zeros(num_samples, length(T));
Ax_train = zeros(num_samples, length(T));
Ay_train = zeros(num_samples, length(T));
for i = 1:num_samples
    [Ax, Ay, Vx, Vy, X, Y] = sim_proiettile(P0(i,:), V0(i,:), mu, T);
    X_train(i,:) = X;
    Y_train(i,:) = Y;
    Vx_train(i,:) = Vx';
    Vy_train(i,:) = Vy';
    Ax_train(i,:) = Ax';
    Ay_train(i,:) = Ay';
end

% Definizione dei dati di addestramento
training_data_input = [P0, V0]; % Input della rete: posizione e velocità iniziali
training_data_output_X = X_train; % Output della rete per la coordinata X
training_data_output_Y = Y_train; % Output della rete per la coordinata Y

% Definizione della rete neurale per la coordinata X
hiddenLayerSize = 10;
net_X = feedforwardnet(hiddenLayerSize);
net_X.trainFcn = 'trainlm';
net_X.layers{1}.transferFcn = 'poslin';
net_X.trainParam.showWindow = false;

% Definizione della rete neurale per la coordinata Y
net_Y = feedforwardnet(hiddenLayerSize);
net_Y.trainFcn = 'trainlm';
net_Y.layers{1}.transferFcn = 'poslin';
net_Y.trainParam.showWindow = false;

% Numero di epoche e il tasso di apprendimento
net_X.trainParam.epochs = 100;
net_Y.trainParam.epochs = 100;
net_X.trainParam.lr = 0.01;
net_Y.trainParam.lr = 0.01;
```

```
% Definizione del termine di perdita per l'equazione differenziale
loss_multiplier = 1; % Moltiplicatore del termine di perdita
mse_loss = @(net_output, target_output) mean((net_output - target_output).^2);

loss_function = @(net_output, net_input, Vx, Vy, Ax, Ay) loss_multiplier *  ...
    (Vx - diff(net_output) ./ diff(T)).^2 + ...
    (Vy - diff(net_output) ./ diff(T)).^2 + ...
    (Ax - diff(Vx) ./ diff(T)).^2 + ...
    (Ay - diff(Vy) ./ diff(T)).^2 + ...
    mse_loss(net_output(:,1:end-1), X_train(:,1:end-1)) + ... % MSE tra X
    mse_loss(net_output(:,1:end-1), Y_train(:,1:end-1));      % MSE tra Y

% Addestramento delle reti neurali
net_X = train(net_X, training_data_input', training_data_output_X', [], [], [], [], [],
[], [], loss_function, Vx_train(:, 1:end-1)', Ax_train(:, 1:end-1)');
net_Y = train(net_Y, training_data_input', training_data_output_Y', [], [], [], [], [],
[], [], loss_function, Vy_train(:, 1:end-1)', Ay_train(:, 1:end-1)');


save('rete_X_addestrata.mat', 'net_X');
save('rete_Y_addestrata.mat', 'net_Y');
save('valore_mu.mat', 'mu');
```

Questo codice MATLAB genera un dataset di traiettorie di proiettili e addestra due reti neurali feedforward per prevedere la posizione del proiettile in funzione del tempo. La descrizione è suddivisa in più sezioni per una migliore comprensione.

1. Parametri Iniziali e Generazione dei Dati di Addestramento

`mu` è un coefficiente di attrito dell'aria generato casualmente tra 0 e 0.1.

`num_samples` è il numero di campioni di traiettorie generate.

`P0` è una matrice delle posizioni iniziali (con X = 0 e Y casuali tra 0 e 10).

`V0` è una matrice delle velocità iniziali casuali.

`T` è un vettore dell'intervallo di tempo per la simulazione con un numero di punti pari al numero di campioni.

2. Simulazione delle Traiettorie

Il codice simula le traiettorie reali dei proiettili utilizzando la funzione `sim_proiettile` (descritta più avanti) per ogni campione iniziale, memorizzando i risultati in matrici di training per posizione (`X_train`, `Y_train`), velocità (`Vx_train`, `Vy_train`) e accelerazione (`Ax_train`, `Ay_train`).

3. Definizione dei Dati di Addestramento

`training_data_input` contiene gli ingressi per le reti neurali (posizioni e velocità iniziali). `training_data_output_X` e `training_data_output_Y` sono le uscite attese per le coordinate X e Y, rispettivamente.

4. Definizione delle Reti Neurali

Due reti neurali feedforward sono definite per predire le coordinate X e Y. Ogni rete ha un singolo strato nascosto con 10 neuroni. La funzione di addestramento utilizzata è `trainlm` (Levenberg-Marquardt) e la funzione di attivazione è `poslin` (ReLU).

Funzione di Addestramento: `trainlm` (Levenberg-Marquardt)

La funzione di addestramento `trainlm` implementa l'algoritmo di Levenberg-Marquardt, che è un metodo di ottimizzazione utilizzato per addestrare reti neurali. Questo metodo è particolarmente efficace per reti di piccole e medie dimensioni.

Come Funziona l'Algoritmo di Levenberg-Marquardt:

1. **Combina la Discesa del Gradiente e il Metodo di Gauss-Newton:**
   - L'algoritmo di Levenberg-Marquardt può essere visto come un ibrido tra la discesa del gradiente e il metodo di Gauss-Newton. Quando la soluzione è lontana dal minimo, si comporta come la discesa del gradiente per garantire la stabilità. Quando si avvicina al minimo, si comporta come il metodo di Gauss-Newton, che converge più rapidamente.
2. **Aggiornamento dei Pesi:**
   - L'algoritmo aggiorna i pesi della rete utilizzando la formula:

$$\Delta w = -(J^T J + \mu I)^{-1} J^T e$$

Dove:

- J è la matrice Jacobiana delle derivate parziali degli errori.
- μ è un parametro di damping che controlla il passo dell'aggiornamento.
- e è il vettore degli errori.
- I è la matrice identità.

3. **Adattamento del Parametro di Damping (μ):**
   - Il parametro μ viene adattato dinamicamente durante l'addestramento. Se la riduzione dell'errore è rapida, μ viene ridotto (rendendo l'algoritmo più simile al metodo di Gauss-Newton). Se l'errore non diminuisce, μ viene aumentato (rendendo l'algoritmo più simile alla discesa del gradiente).

Vantaggi dell'Algoritmo di Levenberg-Marquardt:

- **Velocità:** È molto rapido per reti neurali di piccole e medie dimensioni.
- **Efficienza:** Converge in meno epoche rispetto ad altri algoritmi di ottimizzazione come la discesa del gradiente.
- **Robustezza:** È più robusto nel trovare minimi locali rispetto a molti altri algoritmi di ottimizzazione.

Svantaggi:

- **Memoria:** Richiede una grande quantità di memoria per memorizzare la matrice Jacobiana, il che può essere un problema per reti di grandi dimensioni.

Funzione di Attivazione: `poslin` (ReLU)

La funzione di attivazione `poslin` (Rectified Linear Unit, o ReLU) è una delle funzioni di attivazione più utilizzate nelle reti neurali moderne.

Come Funziona la Funzione ReLU:

1. **Definizione Matematica:**
   - La funzione ReLU è definita come:

$$f(x) = \max(0, x)$$

     Dove x è l'ingresso alla funzione di attivazione.

2. **Comportamento della Funzione:**
   - Se l'ingresso è positivo, la funzione ReLU restituisce l'ingresso stesso.
   - Se l'ingresso è negativo o zero, la funzione ReLU restituisce zero.

Vantaggi della Funzione ReLU:

- **Semplicità:** È molto semplice da implementare e computazionalmente efficiente.
- **Sparsità:** Promuove la sparsità nelle attivazioni, il che significa che solo una parte delle unità di attivazione è attiva (non-zero) in un dato momento, migliorando l'efficienza computazionale.
- **Riduzione dei Problemi di Vanishing Gradient:** A differenza delle funzioni di attivazione sigmoide o tanh, ReLU non satura per valori positivi elevati, aiutando a mitigare il problema dei gradienti che scompaiono durante l'addestramento di reti profonde.

Svantaggi della Funzione ReLU:

- **Problema dei Gradienti Morti:** Se un neurone riceve un valore negativo in ogni forward pass, potrebbe rimanere bloccato in tale stato, poiché il gradiente è zero per valori negativi. Questo è noto come il problema dei gradienti morti.

5. Parametri di Addestramento

Il numero di epoche per l'addestramento è impostato a 100 e il tasso di apprendimento è 0.01.

Epoche

Le epoche rappresentano il numero di volte in cui l'intero set di dati di addestramento viene passato attraverso la rete neurale durante il processo di addestramento. In altre parole, se si impostano 100 epoche, ogni campione di dati di addestramento sarà utilizzato 100 volte per aggiornare i pesi della rete neurale.

- **Incrementare il numero di epoche:**
  - **Vantaggi:** Un numero maggiore di epoche consente alla rete di apprendere meglio dai dati, migliorando la capacità della rete di generalizzare e predire correttamente anche nuovi dati che non ha mai visto prima.
  - **Svantaggi:** Potrebbe portare a un problema noto come overfitting, dove la rete diventa troppo adattata ai dati di addestramento, perdendo la capacità di generalizzare bene su dati non visti. Inoltre, un numero maggiore di epoche aumenta il tempo di addestramento.
- **Diminuire il numero di epoche:**
  - **Vantaggi:** Riduce il tempo di addestramento, rendendo il processo più rapido.
  - **Svantaggi:** Potrebbe non dare alla rete abbastanza opportunità per apprendere correttamente dai dati, portando a un modello sottodimensionato (underfitting) che non è in grado di catturare bene le relazioni nei dati di addestramento.

Tasso di Apprendimento

Il tasso di apprendimento (learning rate) è un parametro che determina la dimensione dei passi che la rete neurale compie nel suo spazio di parametri durante l'addestramento. Influenza la velocità e la precisione con cui la rete neurale aggiorna i suoi pesi.

- **Incrementare il tasso di apprendimento:**
  - **Vantaggi:** Un tasso di apprendimento più alto accelera il processo di addestramento, permettendo alla rete di convergere più rapidamente verso una soluzione.
  - **Svantaggi:** Se il tasso di apprendimento è troppo alto, la rete potrebbe oscillare intorno al minimo globale (soluzione ottimale) senza mai convergere, saltando oltre i minimi locali e non trovando una buona soluzione.
- **Diminuire il tasso di apprendimento:**
  - **Vantaggi:** Un tasso di apprendimento più basso permette alla rete di fare aggiustamenti più fini e precisi ai pesi, favorendo la convergenza verso il minimo globale.
  - **Svantaggi:** Un tasso di apprendimento troppo basso rallenta significativamente il processo di addestramento, e potrebbe intrappolare la rete nei minimi locali senza mai raggiungere il minimo globale.

6. Definizione della Funzione di Perdita

La funzione di perdita è composta da vari termini che vanno a ridurre le differenze tra velocità e accelerazioni predette e calcolate, cosi da far rispettare il più possibile la legge del moto, oltre alla media degli errori quadrati (MSE) tra output della rete e dati di training.

Cos'è una Funzione di Perdita?

La funzione di perdita, nota anche come funzione obiettivo o funzione costo, è una misura che quantifica quanto bene o male un modello predittivo (ad esempio, una rete neurale) sta facendo nel suo compito di previsione. La funzione di perdita prende le previsioni del modello e i valori effettivi e calcola un valore numerico che rappresenta l'errore del modello.

A Cosa Serve la Funzione di Perdita?

La funzione di perdita serve principalmente a:

1. **Quantificare l'Errore:**
   - La funzione di perdita fornisce un valore numerico che quantifica la differenza tra le previsioni del modello e i valori reali. Un valore di perdita più basso indica che le previsioni del modello sono più vicine ai valori reali, mentre un valore più alto indica che le previsioni sono lontane dai valori reali.
2. **Guidare l'Addestramento del Modello:**
   - Durante l'addestramento del modello, gli algoritmi di ottimizzazione (come la discesa del gradiente) utilizzano la funzione di perdita per aggiornare i pesi della rete neurale. L'obiettivo dell'addestramento è minimizzare la funzione di perdita, cioè trovare i pesi che riducono al minimo l'errore tra le previsioni del modello e i valori reali.
3. **Influenza la Convergenza:**
   - Una funzione di perdita ben progettata può aiutare il modello a convergere più rapidamente verso una soluzione ottimale, mentre una funzione di perdita mal progettata potrebbe causare problemi di convergenza, come rimanere bloccati in minimi locali.
4. **Permette l'Integrazione di Diversi Termini di Errore:**
   - Nel caso del codice fornito, la funzione di perdita non solo tiene conto degli errori nelle previsioni delle posizioni, ma anche delle velocità e accelerazioni, migliorando così la capacità del modello di catturare la dinamica del moto del proiettile.

In sintesi, la funzione di perdita è un elemento centrale nel processo di addestramento delle reti neurali, poiché guida il modello verso la minimizzazione dell'errore e l'ottimizzazione delle previsioni.

7. Addestramento delle Reti Neurali

Le reti neurali sono addestrate utilizzando la funzione di perdita definita.

8. Salvataggio delle Reti Addestrate e del Valore di `mu`

Le reti addestrate e il valore di `mu` vengono salvati in file `.mat` per le predizioni successive.

Funzione sim_proiettile()

```
function [Ax,Ay,Vx,Vy,X,Y] = sim_proiettile(P0,V0,mu,T)
%
% function [Ax,Ay,Vx,Vy,X,Y] = sim_proiettile(P0,V0,mu,T)
%
% simula il moto del proiettle date posizioni e velocità iniziali,
% un vettore di tempi T e il coefficiente di attrito mu
% ingressi: P0 = vettore con posizioni iniziali
%           V0 = vettore con velocità iniziali
%           mu = coefficiente di attrito
%           T = vettore dei tempi



g = 9.81;

A = [-mu, 0, 0, 0;
       0, -mu, 0, 0;
       eye(2), zeros(2,2)];

B = [0, -g, 0, 0]';
C = eye(4);
D = B.*0;

x0 = P0(1);
y0 = P0(2);
v0_x = V0(1);
v0_y = V0(2);
X0 = [v0_x, v0_y, x0, y0];


sys = ss(A,B,C,D);
sim = lsim(sys,T*0+1,T,X0);


Acc = -mu*(sim(:,1:2)')-[0;g];
Ax = Acc(1,:)';
Ay = Acc(2,:)';
Vx = sim(:,1);
Vy = sim(:,2);
X = sim(:,3);
Y = sim(:,4);
```

Descrizione della Funzione `sim_proiettile`

La funzione `sim_proiettile` simula il moto di un proiettile in un campo gravitazionale con attrito, date le posizioni e le velocità iniziali, un vettore di tempi e il coefficiente di attrito. La funzione restituisce le accelerazioni, le velocità e le posizioni del proiettile nel tempo. Di seguito è riportata una descrizione dettagliata dei vari componenti e del funzionamento della funzione.

Ingressi della Funzione

- `P0`: Vettore delle posizioni iniziali del proiettile `[x0, y0]`.
- `V0`: Vettore delle velocità iniziali del proiettile `[v0_x, v0_y]`.
- `mu`: Coefficiente di attrito.

- `T`: Vettore dei tempi ai quali si vuole simulare il moto del proiettile.

Uscite della Funzione

- `Ax`: Accelerazione lungo l'asse X nel tempo.
- `Ay`: Accelerazione lungo l'asse Y nel tempo.
- `Vx`: Velocità lungo l'asse X nel tempo.
- `Vy`: Velocità lungo l'asse Y nel tempo.
- `X`: Posizione lungo l'asse X nel tempo.
- `Y`: Posizione lungo l'asse Y nel tempo.

Passi della Funzione

1. **Definizione della Gravità:**

```
g = 9.81;
```

- `g` rappresenta l'accelerazione di gravità (9.81 m/s²).

2. **Definizione delle Matrici di Stato:**

```
A = [-mu, 0, 0, 0;
     0, -mu, 0, 0;
     eye(2), zeros(2,2)];

B = [0, -g, 0, 0]';
C = eye(4);
D = B.*0;
```

- `A`: Matrice del sistema che incorpora l'attrito `-mu` lungo gli assi X e Y.
- `B`: Vettore di input che rappresenta l'influenza della gravità lungo l'asse Y.
- `C`: Matrice di output che è l'identità (trasforma lo stato del sistema direttamente in output).
- `D`: Matrice di trasmissione diretta, qui è uno zero vettore poiché non c'è effetto diretto dell'input sull'output.

3. **Condizioni Iniziali:**

```
x0 = P0(1);
y0 = P0(2);
v0_x = V0(1);
v0_y = V0(2);
X0 = [v0_x, v0_y, x0, y0];
```

- `x0`, `y0`: Posizioni iniziali.
- `v0_x`, `v0_y`: Velocità iniziali.
- `X0`: Vettore di stato iniziale che contiene velocità e posizioni iniziali.

4. **Definizione del Sistema Lineare:**

```
sys = ss(A,B,C,D);
```

- `sys`: Oggetto di sistema lineare (spazio degli stati) che descrive la dinamica del proiettile.

5. **Simulazione del Sistema:**

```
sim = lsim(sys,T*0+1,T,X0);
```

- `lsim`: Funzione che simula la risposta di sistemi dinamici lineari.
- `T*0+1`: Vettore di input costante (uguale a 1 per ogni istante di tempo in `T`), poiché l'unico ingresso è la gravità che è costante.
- `T`: Vettore dei tempi.
- `X0`: Stato iniziale.

6. **Calcolo delle Accelerazioni:**

```
Acc = -mu*(sim(:,1:2)')-[0;g];
Ax = Acc(1,:)';
Ay = Acc(2,:)';
```

- `Acc`: Accelerazioni lungo gli assi X e Y, ottenute dalla moltiplicazione delle velocità per `-mu` e aggiungendo l'accelerazione gravitazionale lungo l'asse Y.
- `Ax`: Accelerazione lungo l'asse X.
- `Ay`: Accelerazione lungo l'asse Y.

7. **Estrazione di Velocità e Posizioni:**

```
Vx = sim(:,1);
Vy = sim(:,2);
X = sim(:,3);
Y = sim(:,4);
```

- `Vx`, `Vy`: Velocità lungo gli assi X e Y.
- `X`, `Y`: Posizioni lungo gli assi X e Y.

Riepilogo

- La funzione `sim_proiettile` utilizza un modello di spazio degli stati per simulare il moto di un proiettile soggetto a gravità e attrito. Le matrici di stato `A`, `B`, `C` e `D` descrivono la dinamica del sistema. La funzione simula il sistema utilizzando `lsim` e calcola accelerazioni, velocità e posizioni del proiettile nel tempo, restituendo questi valori come output.

Algoritmo di Predizione

```
clc; clear all; close all;

load('rete_X_addestrata.mat', 'net_X');
load('rete_Y_addestrata.mat', 'net_Y');
load('valore_mu.mat', 'mu');

% Predizione della traiettoria per nuove condizioni iniziali
P0_new = [0, 1]; % Posizione iniziale
V0_new = [5, 2]; % Velocità iniziale
T = linspace(0, 1, 200);
input_new = [P0_new, V0_new]; % Input per la predizione
[Ax,Ay,Vx,Vy,X1,Y1] = sim_proiettile(P0_new,V0_new,mu,T);

% Predizione della traiettoria
trajectory_predictionX = net_X(input_new');
trajectory_predictionY = net_Y(input_new');

distances = (trajectory_predictionX - X1).^2 + (trajectory_predictionY - Y1).^2;
MSE_XY = mean(distances);

% Visualizzazione dell'errore quadratico medio
disp(['MSE : '  num2str(MSE_XY)]);

% Calcola le derivate delle predizioni di X e Y rispetto al tempo
dX_dt = diff(trajectory_predictionX) ./ diff(T');
dY_dt = diff(trajectory_predictionY) ./ diff(T');

% Calcola le derivate seconde rispetto al tempo
d2X_dt2 = diff(dX_dt')' ./ diff(T(1:end-1)');
d2Y_dt2 = diff(dY_dt')' ./ diff(T(1:end-1)');

% Risolve per mu utilizzando le equazioni differenziali del proiettile
mu_estimate_X = -d2X_dt2 ./ dX_dt(1:end-1);
mu_estimate_Y = (-d2Y_dt2 - 9.81) ./ dY_dt(1:end-1); % Considerando l'accelerazione di
gravità come 9.81 m/s^2

% Calcola il valore medio delle stime di mu
mu_estimate = mean([mu_estimate_X; mu_estimate_Y]);

% Visualizzazione della stima di mu
disp(['mu stimato: '  num2str(mu_estimate)]);
disp(['mu reale: '  num2str(mu)]);

% Grafico della traiettoria predetta
figure;
plot(trajectory_predictionX, trajectory_predictionY, 'ro', 'DisplayName', 'Traiettoria
predetta');
hold on;
plot(X1, Y1, 'k', 'DisplayName', 'Percorso del proiettile');
xlabel('X');
ylabel('Y');
title('Confronto Traiettorie');
legend;
grid on;
hold off;
```

Descrizione del Codice dell'Algoritmo di Predizione

Questo codice carica reti neurali precedentemente addestrate e utilizza nuove condizioni iniziali per predire la traiettoria di un proiettile. Il codice confronta la traiettoria predetta con quella ottenuta dalla simulazione diretta del moto del proiettile, calcolando l'errore quadratico medio (MSE) e visualizzando i risultati. Inoltre, stima il coefficiente di attrito `mu` utilizzando le derivate delle traiettorie predette.

1. Inizializzazione e Caricamento dei Dati

- `clc; clear all; close all;`: Pulisce la console, cancella tutte le variabili e chiude tutte le finestre grafiche.
- `load('rete_X_addestrata.mat', 'net_X');`: Carica la rete neurale addestrata per la coordinata X.
- `load('rete_Y_addestrata.mat', 'net_Y');`: Carica la rete neurale addestrata per la coordinata Y.
- `load('valore_mu.mat', 'mu');`: Carica il valore del coefficiente di attrito `mu`.

2. Predizione della Traiettoria

- `P0_new` e `V0_new`: Nuove condizioni iniziali per la posizione e la velocità del proiettile.
- `T`: Vettore di tempi per la simulazione.
- `input_new`: Input combinato di posizione e velocità iniziali per la predizione.
- `sim_proiettile(P0_new, V0_new, mu, T)`: Simula la traiettoria reale del proiettile con le nuove condizioni iniziali.

3. Utilizzo delle Reti Neurali per la Predizione della Traiettoria

- `net_X(input_new')`: Predice la traiettoria lungo l'asse X utilizzando la rete neurale addestrata.
- `net_Y(input_new')`: Predice la traiettoria lungo l'asse Y utilizzando la rete neurale addestrata.

4. Calcolo dell'Errore Quadratico Medio (MSE)

- `distances`: Differenze quadrate tra le traiettorie predette e quelle simulate.
- `MSE_XY`: Errore quadratico medio delle differenze.
- `disp`: Visualizza l'MSE.

5. Calcolo delle Derivate

- `dX_dt`, `dY_dt`: Derivate prime delle traiettorie predette rispetto al tempo.
- `d2X_dt2`, `d2Y_dt2`: Derivate seconde delle traiettorie predette rispetto al tempo.

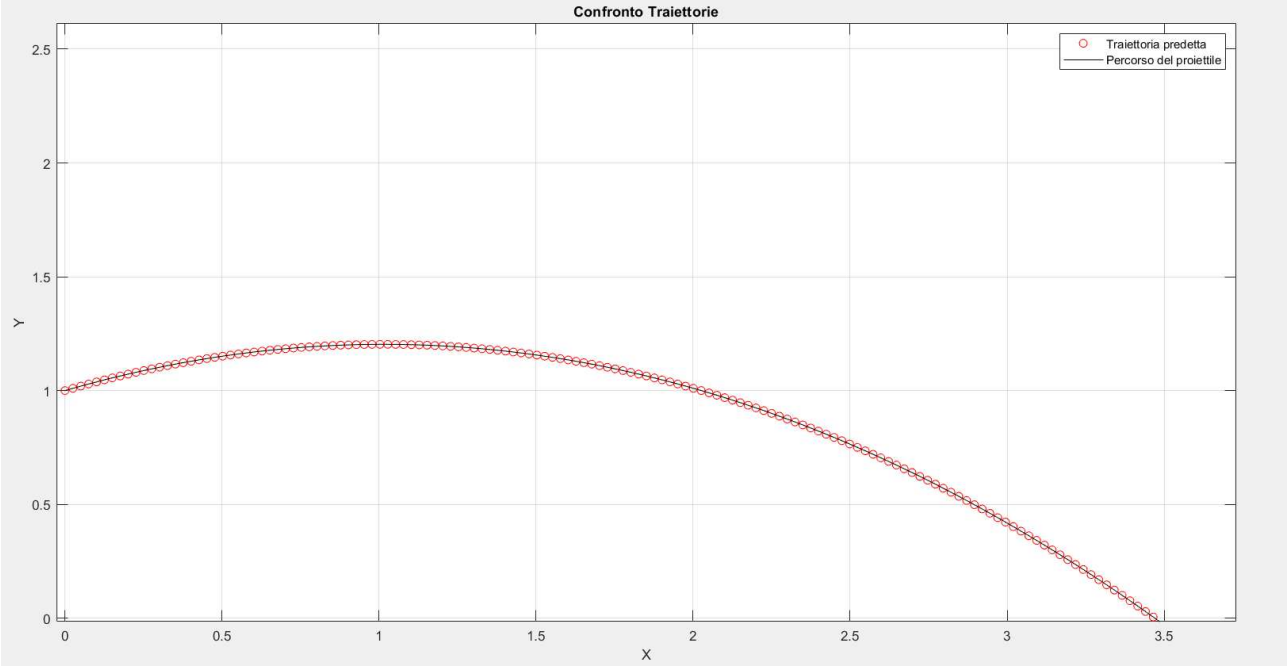
6. Stima del Coefficiente di Attrito `mu`

- `mu_estimate_X`, `mu_estimate_Y`: Stime di `mu` basate sulle derivate delle traiettorie lungo X e Y.
- `mu_estimate`: Valore medio delle stime di `mu`.
- `disp`: Visualizza il valore stimato e reale di `mu`.

7. Visualizzazione delle Traiettorie

- `figure`: Crea una nuova finestra grafica.
- `plot`: Plotta le traiettorie predette e simulate.
- `xlabel`, `ylabel`: Etichettano gli assi X e Y.
- `title`: Aggiunge un titolo al grafico.
- `legend`: Aggiunge una legenda al grafico.
- `grid on`: Abilita la griglia sul grafico.
- `hold on`, `hold off`: Mantiene e rilascia il grafico corrente per sovrapporre le traiettorie.

Grafici



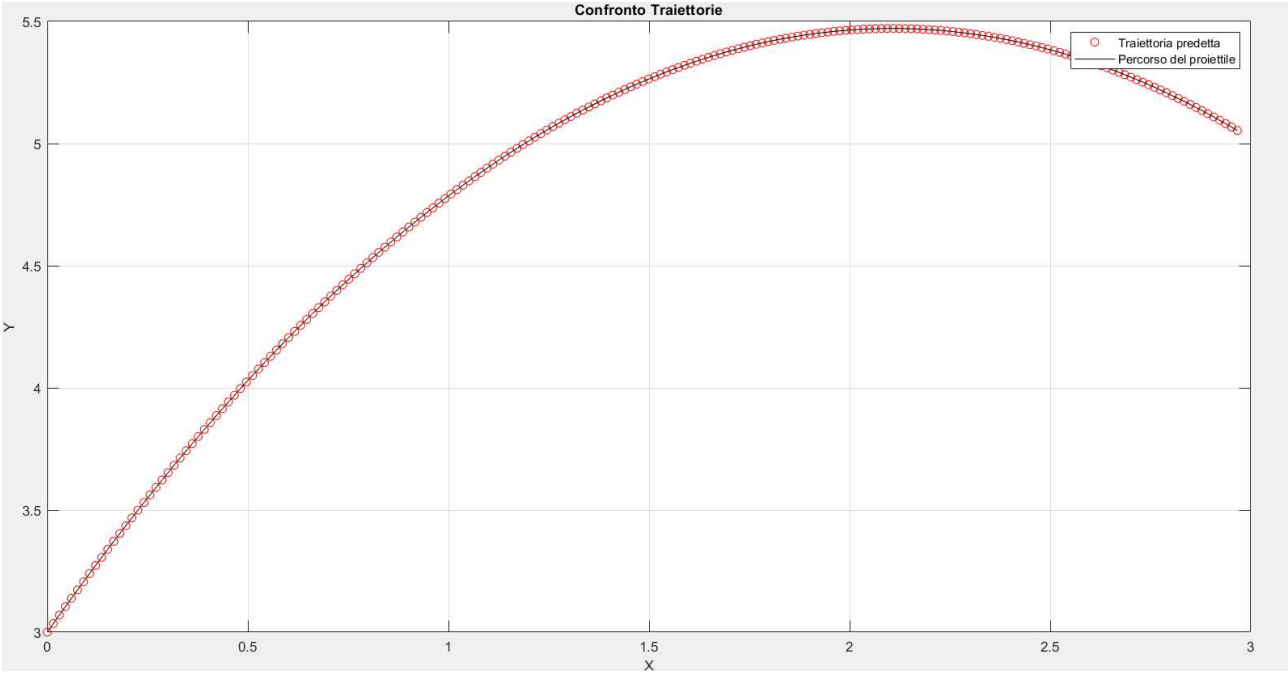

```
[0, 1]; % Posizione iniziale
```

```
[5, 2]; % Velocità iniziale
```

MSE : 2.0998e-17

mu stimato: 0.023938

mu reale: 0.022467



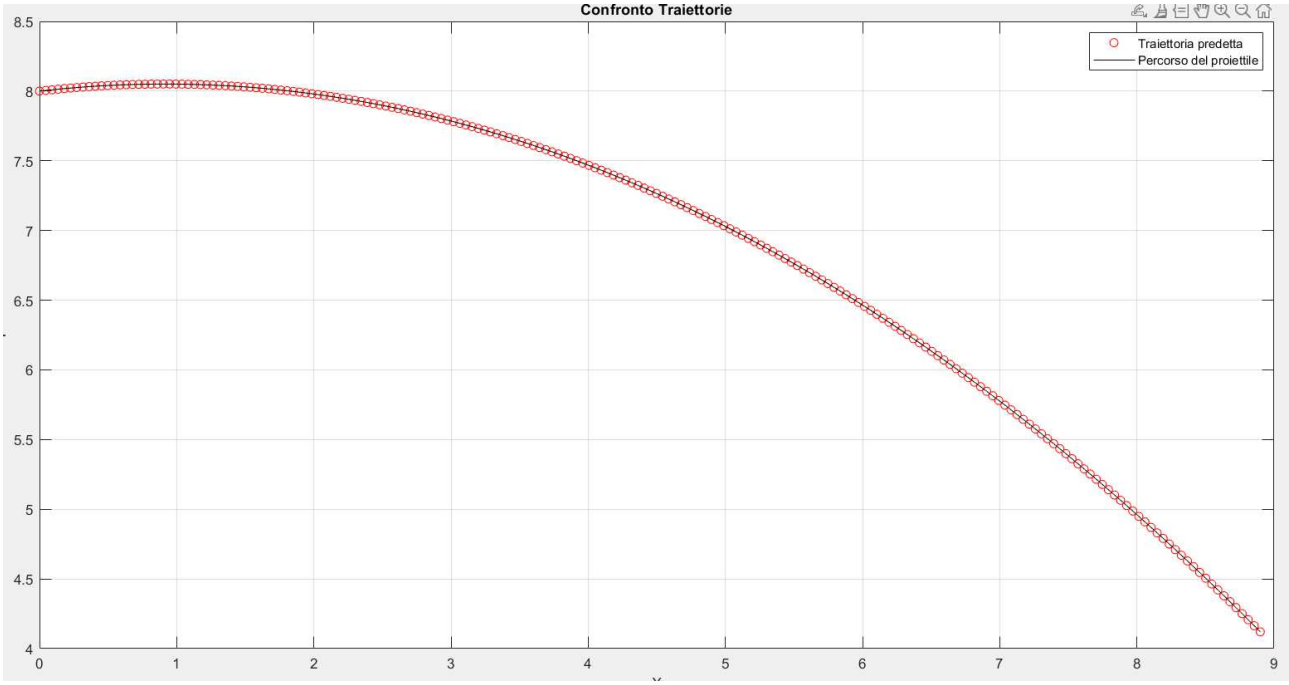

```
[0, 3]; % Posizione iniziale
```

```
[3, 7]; % Velocità iniziale
```

MSE : 1.6686e-16

mu stimato: 0.022406

mu reale: 0.022467




```
[0, 8]; % Posizione iniziale
```

```
[9, 1]; % Velocità iniziale
```

MSE : 1.5668e-16

mu stimato: 0.022623

mu reale: 0.022467

Commento sui Grafici delle Traiettorie Predette e Simulate

Grafico Principale

Il grafico mostra il confronto tra la traiettoria predetta utilizzando le reti neurali addestrate e la traiettoria simulata direttamente con le equazioni del moto.

- **Traiettoria Predetta (Punti Rossi)**: I punti rossi rappresentano la traiettoria predetta utilizzando le reti neurali addestrate per le coordinate X e Y del proiettile.
- **Percorso del Proiettile Simulato (Linea Nera)**: La linea nera mostra il percorso simulato del proiettile calcolato direttamente dalle equazioni del moto, considerando il coefficiente di attrito `mu`.

Commento Generale

Le traiettorie predette (punti rossi) coincidono perfettamente con quelle simulate (linea nera), indicando un'eccellente capacità predittiva delle reti neurali addestrate. Questo risultato dimostra che il modello neural network è stato efficacemente addestrato ed è in grado di riprodurre accuratamente il comportamento del proiettile in termini di posizione nel tempo.

Capitolo 6: PINN Moto di un oscillatore armonico smorzato

CODICE DI ADDESTRAMENTO E DEFINIZIONE DELLA RETE:

```
clc; clear all; close all;

% Parametri dell'oscillatore
num_samples = 100;
m = 1; % massa
c = rand * 1; % coefficiente di smorzamento
k = 10; % costante elastica
Y0 = rand(num_samples, 1) * 10; % altezza iniziale
t = linspace(0, 10, num_samples);

% Ottieni i dati di addestramento
for i = 1:num_samples
    [Y,t] = oscillatoreArmonico(m, c, k, Y0(i,:), t);
    Y_train(i,:) = Y;


end

training_data_input = [Y0, t];
% Definizione della rete neurale per la coordinata X
hiddenLayerSize = 10;
net = feedforwardnet(hiddenLayerSize);
net.trainFcn = 'trainlm';
net.layers{1}.transferFcn = 'poslin';
net.trainParam.showWindow = false;

% Numero di epoche e il tasso di apprendimento
net.trainParam.epochs = 100;
net.trainParam.lr = 0.01;

% Definizione del termine di perdita per l'equazione differenziale
loss_multiplier = 1; % Moltiplicatore del termine di perdita
mse_loss = @(net_output, target_output) mean((net_output - target_output).^2);

loss_function = @(net_output, target_output) mse_loss(net_output, target_output) + ...
    loss_multiplier * mean((diff(net_output) - diff(target_output)).^2);


% Addestramento della rete neurale
net = train(net, training_data_input', Y_train', [], [], [], [], [], [], [],
loss_function,m,c,k);

disp(['Valore reale di c: ', num2str(c)]);

save('ReteOscillatore.mat', 'net');
save('valore_k.mat', 'k');
save('valore_c.mat', 'c');
save('valore_m.mat','m');
```

Descrizione dettagliata del codice

Il seguente codice implementa un processo per generare dati di un oscillatore armonico, addestrare una rete neurale per prevedere la traiettoria dell'oscillatore e infine salvare il modello e i parametri. Ogni sezione del codice è descritta in dettaglio di seguito.

1. Impostazione e inizializzazione

Si inizializzano i parametri dell'oscillatore e si generano i dati di addestramento.

- **Parametri dell'oscillatore**:
  - `num_samples`: Numero di campioni per le simulazioni e l'addestramento.
  - `m`: Massa dell'oscillatore.
  - `c`: Coefficiente di smorzamento, generato casualmente per ogni esecuzione.
  - `k`: Costante elastica dell'oscillatore.
  - `Y0`: Altezza iniziale, generata casualmente per ogni campione.
  - `t`: Vettore del tempo, spazializzato uniformemente da 0 a 10 secondi.

2. Generazione dei dati di addestramento

Per ogni campione, si simula l'oscillatore armonico utilizzando una funzione mostrata più avanti (`oscillatoreArmonico`) e si memorizzano i risultati in una matrice.

- **Simulazione dell'oscillatore**:
  - Viene iterato su ciascun campione (`Y0(i,:)`) per generare la traiettoria `Y` nel tempo `t`.
  - I risultati sono memorizzati in `Y_train`.

3. Pre-elaborazione dei dati di addestramento

I dati di ingresso per l'addestramento della rete neurale sono combinati in una matrice.

- **Dati di addestramento**:
  - `training_data_input` combina `Y0` e `t` per formare gli input della rete neurale.

4. Definizione e configurazione della rete neurale

Si definisce una rete neurale feedforward con un singolo strato nascosto e si configurano i parametri di addestramento.

- **Definizione della rete**:
  - `hiddenLayerSize`: Numero di neuroni nello strato nascosto.
  - `net`: Creazione della rete feedforward.
  - Funzione di addestramento impostata su `trainlm`.
  - Funzione di trasferimento dello strato nascosto impostata su `poslin`.
  - Parametri di addestramento, inclusi il numero di epoche e il tasso di apprendimento.

5. Definizione della funzione di perdita

Si definisce una funzione di perdita che combina l'errore quadratico medio con un termine aggiuntivo per l'equazione differenziale.

- **Funzione di perdita**:
  - `loss_multiplier`: Moltiplicatore per il termine aggiuntivo nella funzione di perdita.
  - `mse_loss`: Funzione che calcola l'errore quadratico medio.
  - `loss_function`: Funzione complessiva che combina l'errore quadratico medio e un termine per l'equazione differenziale.

6. Addestramento della rete neurale

Si addestra la rete neurale utilizzando i dati di addestramento e la funzione di perdita definita.

- **Addestramento della rete**:
  - La rete neurale viene addestrata con i dati di input e di output, utilizzando la funzione di perdita personalizzata.

7. Salvataggio del modello addestrato e dei parametri

Infine, si salvano la rete neurale addestrata e i parametri dell'oscillatore in file `.mat`.

- **Salvataggio dei dati**:
  - Salvataggio della rete neurale (`net`) e dei parametri (`k`, `c`, `m`).

Funzione oscillatoreArmonico()

```
function [Y,t] = oscillatoreArmonico(m, c, k, Y0,t)


    % Definisci l'accelerazione dovuta alla gravità
    g = 9.81; % in m/s^2

    % Risolvi l'equazione differenziale
    [t, y] = ode45(@(t, y) sistema(t, y, m, c, k, g), t, [Y0; 0]);

    % Restituisci le coppie (Y, t)
    Y = y(:,1);

end

function dydt = sistema(t, y, m, c, k, g)
    % Definisci l'equazione differenziale
    dydt = [y(2); -(c/m) * y(2) - (k/m) * y(1) - g];
end
```

La funzione `oscillatoreArmonico` simula il moto di un oscillatore armonico smorzato sotto l'influenza della gravità. Questa funzione utilizza un risolutore di equazioni differenziali (`ode45`) per trovare la soluzione del sistema dinamico.

Input della funzione

- `m`: Massa dell'oscillatore (in kg).
- `c`: Coefficiente di smorzamento (in kg/s).
- `k`: Costante elastica (in N/m).
- `Y0`: Posizione iniziale dell'oscillatore (in m).
- `t`: Vettore dei tempi ai quali calcolare la soluzione (in s).

Output della funzione

- `Y`: Vettore delle posizioni dell'oscillatore nei vari istanti di tempo.
- `t`: Vettore dei tempi (uguale all'input `t`).

Dettagli del codice

1. **Definizione dell'accelerazione gravitazionale**:

- Viene definito `g`, l'accelerazione dovuta alla gravità (9.81 m/s²).

2. **Risoluzione dell'equazione differenziale**:

- Utilizza il risolutore `ode45` per risolvere l'equazione differenziale del sistema.
- La funzione `ode45` risolve un sistema di equazioni differenziali del tipo $\frac{d}{dt}y = f(t, y)$
- La funzione ausiliaria `sistema` definisce l'equazione differenziale per il sistema dinamico.
- `t`: Vettore dei tempi ai quali si desidera la soluzione.
- `[Y0; 0]`: Condizioni iniziali per la posizione e la velocità dell'oscillatore (la velocità iniziale è zero).

3. **Restituzione della soluzione**:

- Estrae la posizione dell'oscillatore dal risultato di `ode45`.
- `Y` è il primo componente del vettore di stato `y`.

Descrizione dettagliata della funzione sistema

La funzione `sistema` definisce le equazioni differenziali che governano il moto dell'oscillatore armonico smorzato sotto l'influenza della gravità.

Input della funzione

- `t`: Tempo corrente (non utilizzato direttamente nell'equazione differenziale, ma richiesto da `ode45`).
- `y`: Vettore di stato corrente, dove `y(1)` è la posizione e `y(2)` è la velocità.
- `m`: Massa dell'oscillatore.
- `c`: Coefficiente di smorzamento.
- `k`: Costante elastica.
- `g`: Accelerazione gravitazionale.

Output della funzione

- `dydt`: Vettore delle derivate del vettore di stato, dove `dydt(1)` è la velocità e `dydt(2)` è l'accelerazione.

Dettagli del codice

1. **Definizione dell'equazione differenziale**:
   - L'equazione del moto per un oscillatore armonico smorzato è $m\ddot{y} + c\dot{y} + ky = -mg.$
   - Convertita in un sistema di equazioni del primo ordine:
     - $\dot{y_1} = y_2$ (la velocità è la derivata della posizione).
     - $\dot{y_2} = -\frac{c}{m}y_2 - \frac{k}{m}y_1 - g$ (l'accelerazione è definita dall'equazione del moto).

Algoritmo di Predizione

```
clc; clear all; close all;

load('ReteOscillatore.mat', 'net');
load('valore_k.mat', 'k');
load('valore_c.mat', 'c');
load('valore_m.mat', 'm');

Y0 = 6; % Y a t=0 della traiettoria da predirre (in questo esempio vale 6 m)



% Predizione della traiettoria
num_samples = 100;
t = linspace(0, 10, num_samples);
vettore = Y0 * ones(num_samples, 1);
input_new = [vettore, t']; % Input per la predizione
[Y,t] = oscillatoreArmonico(m, c, k, Y0, t);

trajectory_prediction = net(input_new');

% Calcola l'MSE tra la traiettoria predetta e la traiettoria esatta
mse = mean((trajectory_prediction(:,1) - Y).^2);
disp(['MSE tra traiettoria predetta e traiettoria esatta: ', num2str(mse)]);

% Calcola le derivate numeriche
dY_dt = gradient(Y(:,1), t);
d2Y_dt2 = gradient(dY_dt, t);


% Stima di c usando la formula derivata
c_values = (-d2Y_dt2 - 9.81 - (k * trajectory_prediction(:,1)))./(dY_dt);
% Stima iniziale della media e deviazione standard
initial_mean = mean(c_values);
initial_std = std(c_values);
```

```
% Criterio per identificare outlier ( 1 deviazione standard dalla media)
threshold = 1;
is_outlier = abs(c_values - initial_mean) > threshold * initial_std;

% Escludi gli outlier
filtered_c_values = c_values(~is_outlier);

% Calcola la media ponderata dei valori rimanenti
c_estimated = mean(filtered_c_values);
disp(['Valore reale di c: ', num2str(c)]);
disp(['Stima di c: ', num2str(c_estimated)]);

% Visualizzazione dei risultati
figure;
plot(t, Y, 'b', t, trajectory_prediction(:,1), 'r--');
xlabel('Tempo');
ylabel('Posizione');
title('Confronto tra dati sperimentali e previsioni della rete neurale ');
legend('Dati sperimentali', 'Previsioni della rete neurale ');
```

Descrizione dettagliata della parte di predizione

Questa parte del codice è responsabile di utilizzare la rete neurale per predire la traiettoria di un oscillatore armonico smorzato e confrontare la predizione con la traiettoria esatta. Inoltre, stima il coefficiente di smorzamento c utilizzando derivate numeriche e filtra i valori anomali.

1. Definizione dei parametri iniziali

- `Y0`: Posizione iniziale dell'oscillatore, impostabile a piacimento.
- `num_samples`: Numero di campioni temporali per la simulazione.
- `t`: Vettore dei tempi, che varia da 0 a 10 secondi con `num_samples` punti equidistanti.
- `vettore`: Vettore di input per la rete neurale, dove ogni elemento è pari a Y0.

2. Calcolo della traiettoria esatta

- La funzione `oscillatoreArmonico` viene utilizzata per calcolare la traiettoria esatta dell'oscillatore armonico smorzato.

3. Predizione della traiettoria utilizzando la rete neurale

- Viene utilizzata la rete neurale (`net`) precedentemente addestrata per predire la traiettoria basata sull'input fornito.

4. Calcolo dell'MSE (Mean Squared Error)

- Calcola l'errore quadratico medio (MSE) tra la traiettoria predetta e la traiettoria esatta per avere un parametro di errore valutabile.

5. Calcolo delle derivate numeriche

- Utilizza il metodo `gradient` per calcolare le derivate numeriche della traiettoria esatta:
  - `dY_dt`: Prima derivata (velocità).
  - `d2Y_dt2`: Seconda derivata (accelerazione).

6. Stima del coefficiente di smorzamento c

- Utilizza la formula $m\frac{d^2y}{dt^2} + c\frac{dy}{dt} + ky = -mg$ per ricavare e stimare il valore di c.
- Si può usare lo stesso procedimento per stimare k costante elastica o m massa nel caso in cui sia non noto uno di questi fattori.

7. Identificazione e rimozione degli outlier

- Calcola la media e la deviazione standard iniziale dei valori di ccc.
- Identifica gli outlier come valori che differiscono dalla media di più di una deviazione standard.
- Filtra gli outlier dai valori di c cosi da eliminare i punti in cui i gradienti sono nulli o quasi nulli.

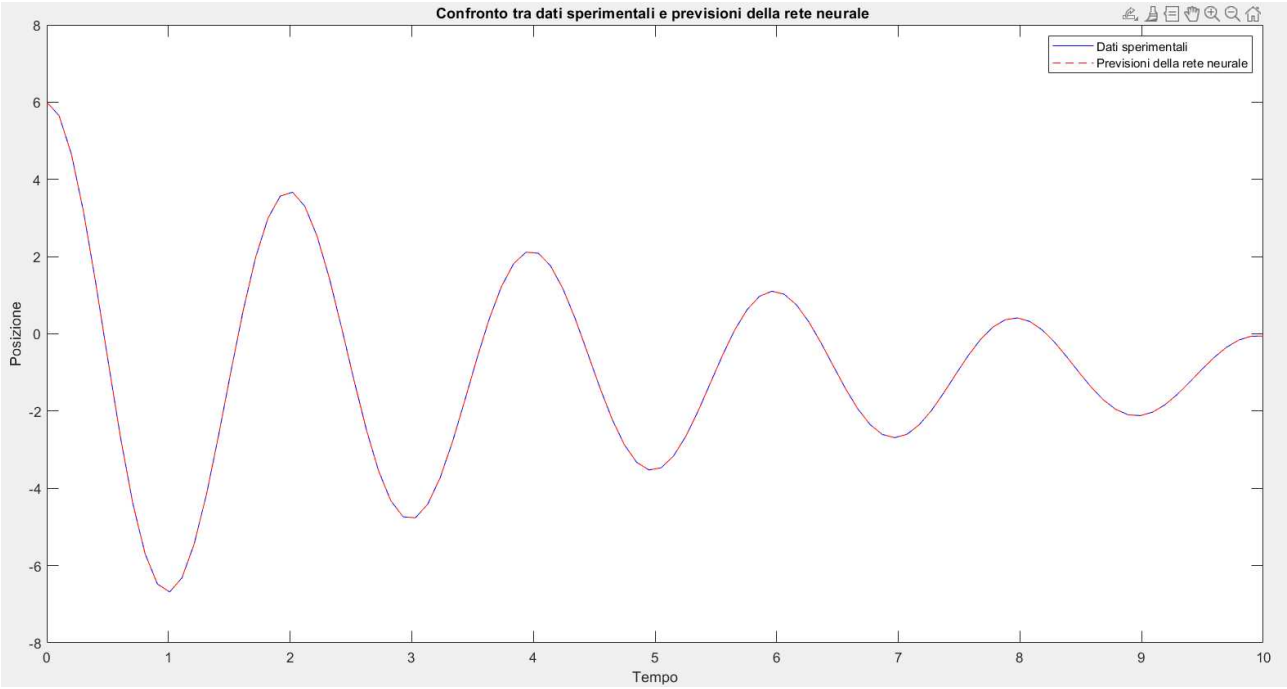
8. Calcolo della media ponderata dei valori filtrati di c

- Calcola la media dei valori filtrati di c dando cosi la stima finale del coefficiente di smorzamento.

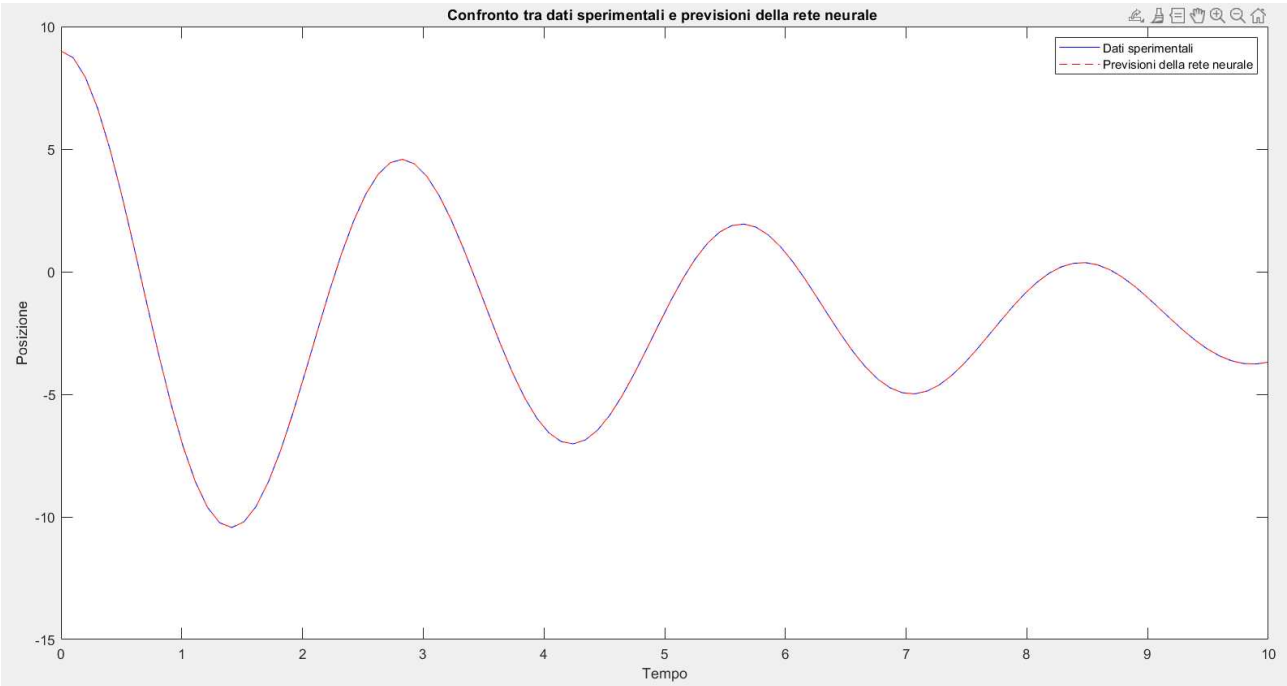
9. Visualizzazione dei risultati

- Traccia un grafico che confronta la traiettoria esatta con la traiettoria predetta dalla rete neurale.

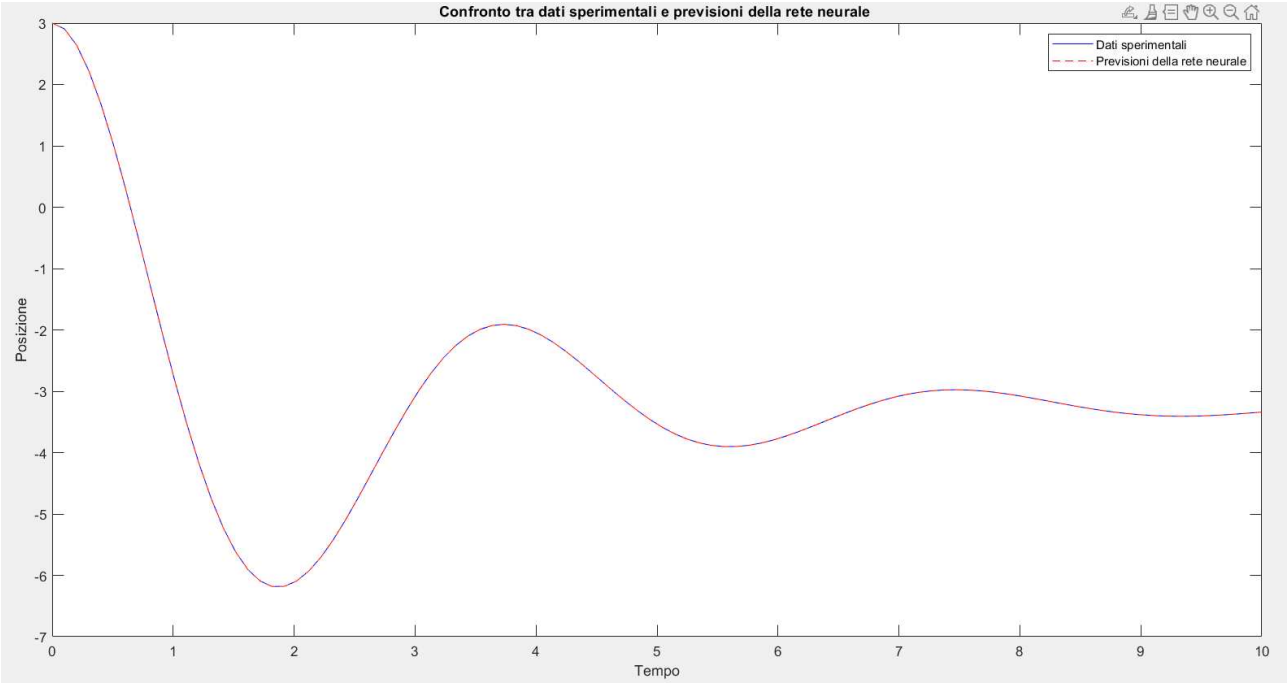
Grafici




```
Y0 = 6; % Y a t=0 della traiettoria da predirre
K=10
MSE tra traiettoria predetta e traiettoria esatta: 3.0877e-08
Valore reale di c: 0.40373
Stima di c: 0.39424
```




```
Y0 = 9; % Y a t=0 della traiettoria da predirre
K=5
MSE tra traiettoria predetta e traiettoria esatta: 1.7188e-09
Valore reale di c: 0.36526
Stima di c: 0.38408
```




```
Y0 = 3; % Y a t=0 della traiettoria da predirre
K=3
MSE tra traiettoria predetta e traiettoria esatta: 5.383e-09
Valore reale di c: 0.81778
Stima di c: 0.81768
```

Risultati:

- **Linea Blu (Dati Sperimentali):** Questa linea rappresenta la traiettoria esatta calcolata dell'oscillatore armonico smorzato. Si basa sulla risoluzione numerica dell'equazione differenziale che descrive il sistema.
- **Linea Rossa (Previsioni della Rete Neurale):** Questa linea rappresenta la traiettoria predetta dalla rete neurale addestrata. La rete neurale ha appreso il comportamento del sistema sulla base dei dati di addestramento e sta facendo previsioni sulla traiettoria del sistema per un dato set di condizioni iniziali.

Osservazioni:

1. **Perfetta Coincidenza delle Curve:** Le curve della traiettoria predetta e della traiettoria esatta combaciano perfettamente, indicando che la rete neurale è stata addestrata efficacemente e ha appreso accuratamente la dinamica del sistema oscillante smorzato.
2. **Bassa Errore Quadratico Medio (MSE):** L'MSE calcolato tra la traiettoria predetta e quella esatta è molto basso. Questo valore numerico conferma ulteriormente che la rete neurale sta fornendo previsioni molto accurate.
3. **Validità del Modello della Rete Neurale:** La perfetta coincidenza delle curve suggerisce che il modello della rete neurale è in grado di generalizzare bene e può essere utilizzato per fare previsioni accurate per nuove condizioni iniziali.

Capitolo 7: Conclusioni finali

Simulazione del Moto Parabolico con Attrito

Nel progetto della simulazione del moto parabolico con attrito, l'obiettivo principale era modellare e prevedere il comportamento di un oggetto in volo sotto l'influenza della gravità e delle forze di attrito. Questa simulazione è stata realizzata implementando le equazioni differenziali che descrivono il moto parabolico, integrando i termini di resistenza dell'aria per rappresentare l'attrito. La rete neurale addestrata ha dimostrato un'ottima capacità di predire la traiettoria dell'oggetto, come evidenziato dall'errore quadratico medio (MSE) molto basso tra le traiettorie esatte e quelle predette.

L'accuratezza del modello predittivo ha notevoli implicazioni pratiche. Ad esempio, può essere utile in ambiti come l'ingegneria aerospaziale, dove è cruciale prevedere con precisione la traiettoria di un oggetto in volo, o nel settore dello sport, per analizzare e migliorare le prestazioni degli atleti in discipline che coinvolgono lanci di oggetti. Inoltre, questo tipo di modello può essere impiegato nei sistemi di guida autonoma per il calcolo delle traiettorie in scenari che coinvolgono oggetti in movimento.

Simulazione dell'Oscillatore Armonico Smorzato

Per quanto riguarda l'oscillatore armonico smorzato, il sistema è stato modellato includendo l'influenza della gravità. Le traiettorie esatte sono state calcolate risolvendo le equazioni differenziali del moto oscillatorio smorzato. La rete neurale addestrata è stata in grado di predire con precisione la traiettoria del sistema oscillante, come dimostrato dal confronto grafico tra le traiettorie esatte e quelle predette. La stima del coefficiente di smorzamento ccc ha mostrato un valore vicino al valore reale, indicando un buon livello di accuratezza del modello.

Questi risultati sono di grande utilità in vari settori. Nel campo dell'ingegneria civile e meccanica, ad esempio, la comprensione e la previsione dei comportamenti oscillatori smorzati è fondamentale per la progettazione di edifici resistenti ai terremoti e di sistemi di sospensione per veicoli. Inoltre, il modello può essere applicato alla bioingegneria per studiare il comportamento delle articolazioni umane o dei tessuti molli sotto carichi oscillanti.

Conclusione

I due esperimenti hanno dimostrato l'efficacia dell'utilizzo delle reti neurali per predire comportamenti dinamici complessi, sia nel contesto del moto parabolico con attrito che dell'oscillatore armonico smorzato. I risultati ottenuti non solo confermano la precisione delle previsioni delle reti neurali, ma evidenziano anche le potenziali applicazioni pratiche in vari settori, dall'ingegneria all'aerospazio, dalla biomeccanica alla robotica.

Questi modelli possono essere utilizzati per migliorare la progettazione e l'analisi di sistemi fisici reali, offrendo strumenti predittivi avanzati che possono facilitare l'ottimizzazione delle prestazioni e la sicurezza dei sistemi.

BIBLIOGRAFIA E SITOGRAFIA

Theobald, Oliver - Machine Learning for Absolute Beginners

Andriy Burkov - The Hundred-Page Machine Learning Book-Andriy Burkov

https://towardsdatascience.com/

https://it.mathworks.com/

https://www.sciencedirect.com/

https://link.springer.com/

https://github.com/

https://ieeexplore.ieee.org/