Linearizzazione e predizione con reti neurali ricorrenti

Corso di Laurea in Fisica

Anno Accademico 2024/2025

Laureando: Niccolò De Poli

Matricola: 2075457

Relatore: Dr. Michele Allegra

Abstract

Il Reservoir Computing (RC) è una tecnica di apprendimento automatico utilizzata per apprendere mappe ingresso-uscita in presenza di correlazione temporale. Nelle architetture di tipo RC, un layer di input è connesso linearmente a un layer ricorrente, non lineare e ad alta dimensionalità (detto reservoir), il quale è a sua volta connesso linearmente a un layer di uscita. La caratteristica peculiare del RC è che solo il layer di uscita viene addestrato, mediante una semplice regressione lineare, mentre il reservoir rimane non addestrato. L'utilizzo del RC per apprendere e prevedere il comportamento di sistemi dinamici è assodato. Un recente studio ha dimostrato che anche un reservoir lineare è sufficiente per questo compito. In realtà, il RC lineare non solo fornisce previsioni accurate, ma permette anche una linearizzazione implicita del sistema dinamico. In questa tesi, lo studente implementerà da capo una rete RC lineare. La rete sarà inizialmente addestrata su un semplice sistema linearizzabile e, successivamente, verrà applicata a dati reali provenienti da esperimenti di neuroimaging.

Contents

1 Introduzione **1**

2 Metodi **3**
2.1 Rete neurale 3
2.2 RNN 3
2.3 RC - Reservoir Computing 4
2.4 Copia sistema non lineare con RNNs lineari 6
2.5 Operatore di Koopman 8
2.6 Test R^2 9
2.7 Matrici di corelazione funzionale FC 9

3 Simulazioni **11**
3.1 Metodo di Eulero 11
3.1.1 Implementazione metodo di Eulero 12
3.2 Metodo di Runge-Kutta 14
3.2.1 Implementazione metodo di Runge-Kutta 15
3.3 Confronto tra i due metodi 17

4 Implementazione di un 'digital twin' tramite una rete neurale **19**
4.1 Applicazione alle simulazioni numeriche 19
4.2 Applicazione a dati reali 23

1 Introduzione

Lo studio dei sistemi dinamici complessi riveste un ruolo centrale nella ricerca scientifica contemporanea. Negli ultimi decenni, è emersa una vasta gamma di metodologie basate sullo studio di dati provenienti da sistemi di crescente complessità. Tra queste metodologie, le tecniche di apprendimento automatico sono diventate sempre più popolari grazie alla loro efficacia nel modellare insiemi complessi di dati e nel fornire predizioni accurate. Sebbene i modelli siano molto accurati e precisi, spesso mancano di interpretabilità, perdendo in chiarezza e la comprensibilità.
Per affrontare questa limitazione, la teoria di Koopman offre un quadro teorico prezioso (Fig. (1)). Mappando sistemi non lineari in spazi di dimensione superiore riesce a fornire una visione più chiara della dinamica del sistema. Questo approccio prevede lo sviluppo di metodi mirati a calcolare rappresentazioni lineari di dimensioni finite di sistemi non lineari all'interno di uno spazio funzionale. Una volta inferito un sistema lineare equivalente, l'interpretabilità diventa immediata, permettendo di comprendere le scale temporali e i cambiamenti dei sistemi osservati.

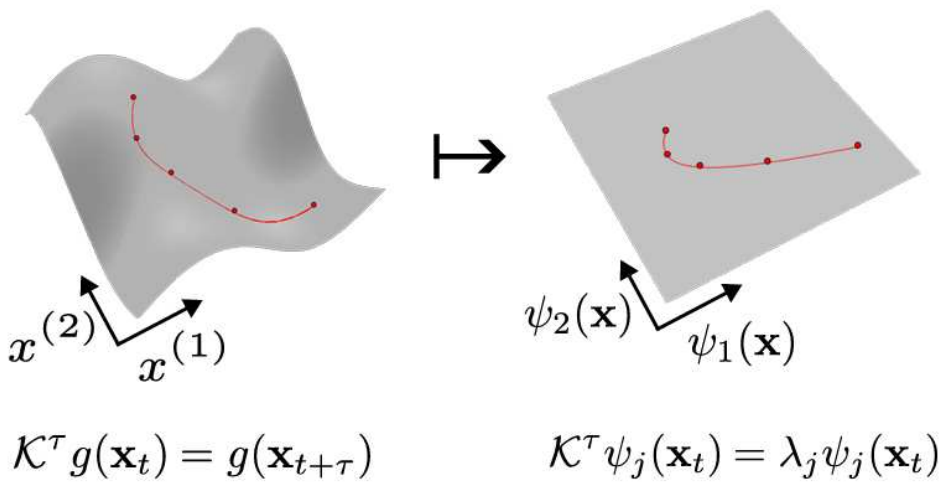


Figure 1: La dinamica non lineare può essere proiettata su una varietà lineare. Nello spazio funzionale infinito-dimensionale degli osservabili di sistema, le autofunzioni dell'operatore di Koopman forniscono una base per ridurre dinamiche non lineari a processi lineari semplificati.

Il Reservoir Computing (RC) è un approccio comune di apprendimento automatico (Fig. 2). Utilizza reti neurali ricorrenti (*recurrent neural networks*, RNN) ad alta dimensionalità con accoppiamenti interni fissati, che trasformano in modo non lineare le serie temporali ricevute in ingresso. Ciò porta a una rappresentazione non lineare ad alta dimensionalità del segnale in ingresso, "leggendo" la quale, tramite un proiezione lineare, si può in linea di principio prevedere campioni futuri della stessa serie temporale. La forza di questo approccio risiede nel fatto che la proiezione giusta può essere semplicemente ottenuta tramite una regressione lineare. Concettualmente, le traiettorie non lineari descritte dal sistema in ingresso su una *varietà*[1] basso-dimensionale vengono mappate in traiettorie

[1]Una *varietà* è uno spazio topologico che localmente è omeomorfo a uno spazio euclideo n-dimensionale ($\mathbb{R}^n$), ma che globalmente può avere proprietà geometriche differenti. Ad esempio le varietà localmente

lineari descritte dalla rete su una varietà alto dimensionale, e vice versa (Fig. (2)).

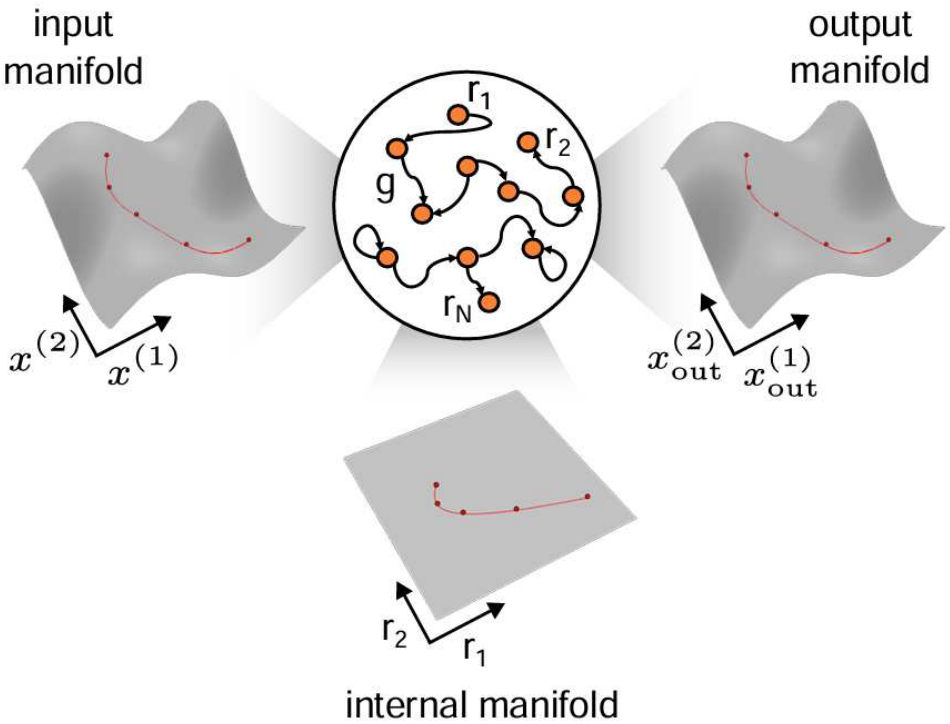


Figure 2: Dopo la fase di apprendimento, i pesi di lettura $\boldsymbol{W}^{out}$ vengono impostati per riprodurre le osservabili del sistema fornite come input. La rete risultante è composta da unità lineari (lRNN) e la sua dinamica autonoma post-apprendimento racchiude una rappresentazione lineare del sistema originale.

Questa soluzione ricorda l'approccio di codifica efficiente adottato dalle reti neuronali biologiche nel cervello. Non sorprende, quindi, che le RNN siano state ampiamente studiate in neuroscienze.
Disporre di strumenti in grado di prevedere osservazioni future di un sistema dinamico utilizzando osservazioni passate come input permette di costruire una replica dello stesso sistema. Infatti, le osservazioni predette possono essere reinserite come input, dando vita a un modello generativo autonomo. Sia la teoria di Koopman che gli approcci RC hanno il potenziale per generare tale "copia digitale" (digital twin), producendo serie temporali statisticamente equivalenti a quelle generate dal sistema originale. Tali copie digitali possono essere strumenti efficaci per analizzare e simulare le loro controparti fisiche. Esempi di successo includono copie digitali dell'attività cerebrale progettate per assistere i neurochirurghi nel trattamento di pazienti con epilessia farmaco-resistente. Inoltre, questi strumenti possono fornire una piattaforma complementare per progettare stimolazioni da testare *in silico*[2], portando infine allo sviluppo di sistemi di controllo per la neuroriabilitazione [1,2,3].

omeomorfe alla retta sono le curve, mentre quelle localmente omeomorfe al piano sono le superfici.

[2] *In silico* definisce la simulazione matematica al computer per riprodurre fenomeni e comportamenti di sistemi biologici, ad esempio di cellule od organi.

2 Metodi

2.1 Rete neurale

Nel cervello, l'azione coordinata dei neuroni consente di identificare fenomeni, valutare opzioni e trarre conclusioni. Una rete neurale (neural network) è un modello computazionale ispirato al funzionamento del cervello, utilizzato nell'intelligenza artificiale e nell'apprendimento automatico per risolvere problemi complessi. Una rete neurale è un grafo diretto di neuroni connessi fra di loro e organizzati in strati (layers) (Fig. (3)):

- Input layer: Riceve i dati grezzi.
- Hidden layers: Strati intermedi che estraggono caratteristiche astratte.
- Output layer: Produce il risultato finale

I layers sono i "mattoni" fondamentali di una rete neurale, organizzati in una struttura gerarchica per elaborare i dati in modo incrementale. Ogni strato riceve input dallo strato precedente, applica trasformazioni (lineari e non lineari) e passa l'output allo strato successivo.
Le reti neurali possono essere classificate in diversi tipi in base al loro scopo:

- Reti neurali feed forward (FFNN, Feed Forward Neural Networks), utilizzate per l'elaborazione di dati non sequenziali, ad esempio nel riconoscimento delle immagini
- Reti neurali ricorrenti (RNN, Recurrent Neural Networks), utilizzate principalmente quando si utilizzano serie temporali per fare previsioni sui risultati futuri [2, 4].

2.2 RNN

Una Recurrent Neural Network (RNN) è un'architettura di rete neurale artificiale specializzata nell'elaborazione di dati sequenziali, caratterizzata da connessioni cicliche che permettono il mantenimento di uno stato interno (hidden state) (Fig. (3)). Mentre le reti di deep learning tradizionali presuppongono che le coppie input-output siano indipendenti, nelle reti neurali ricorrenti l'output dipende dalla sequenza di input precedenti. Infatti, le RNN hanno una forma di "memoria interna" che permette loro di utilizzare informazioni provenienti da passaggi precedenti nella sequenza per influenzare l'output corrente.
Data una sequenza di input $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$, una RNN aggiorna il suo stato nascosto h_t al tempo t secondo l'equazione:

$$h_t = \sigma(\mathbf{W}_h h_{t-1} + \mathbf{W}_x x_t + \mathbf{b}), \tag{1}$$

dove:

- $\mathbf{W}_h$ e $\mathbf{W}_x$ sono matrici di pesi apprendibili che sono riutilizzate per ogni passo temporale
- $\mathbf{b}$ è un vettore di bias
- σ è una funzione di attivazione non lineare

L'output y_t al tempo t è calcolato come:

$$y_t = \mathbf{W}_y h_t + \mathbf{b}_y. \tag{2}$$

Le RNN standard soffrono però del problema del *vanishing gradient*, che limita la capacità di apprendere dipendenze a lungo termine tramite algoritmi di discesa del gradiente. Per risolvere questo problema, sono state sviluppate architetture avanzate come LSTM (Long Short-Term Memory) e GRU (Gated Recurrent Unit). Le principali applicazioni delle RNN si hanno nell'elaborazione del Linguaggio Naturale (Natural language Processing, NLP), nell'analisi di serie temporali, nel riconoscimento vocale e nella generazione di testo e musica [2, 4].

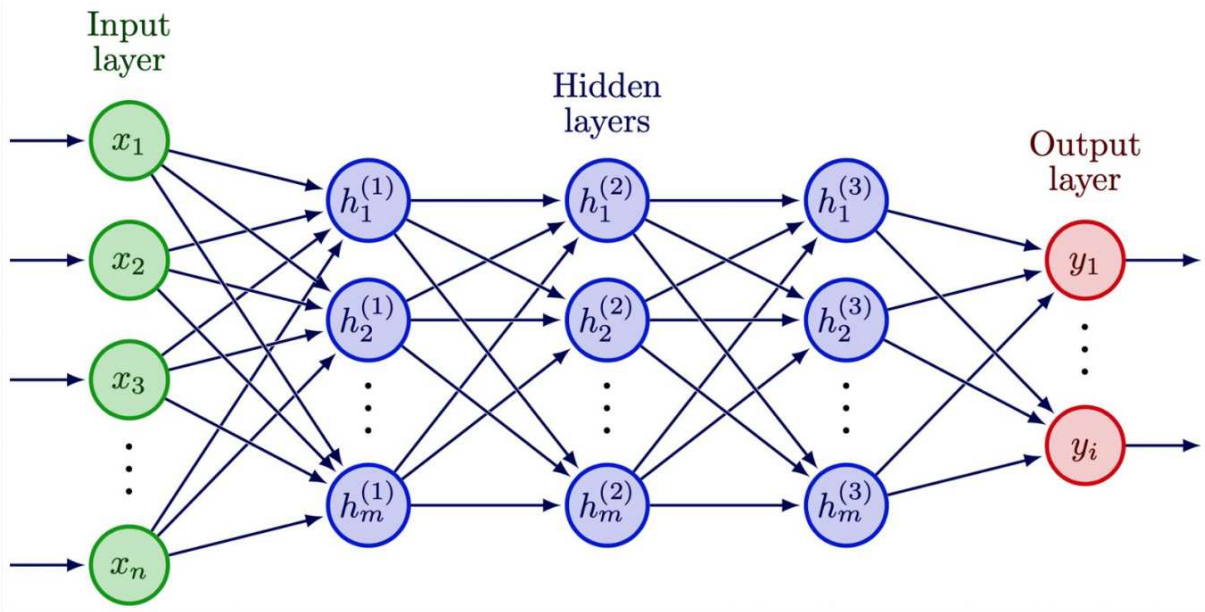


Figure 3: Schema Recurrent Neural Network

2.3 RC - Reservoir Computing

L'approccio del reservoir computing utilizza una rete neurale ricorrente (RNN) con accoppiamenti interni fissi e casuali, detto reservoir, il cui stato viene propagato a un layer di unità di "lettura" (readout). In questo contesto, la proprietà di "echo state" emerge spontaneamente dalla dinamica collettiva della RNN.
Assumendo che le condizioni standard di compattezza (gli spazi di inuput e degli stati sono chiusi e limitati) siano soddisfatte e che la rete non abbia connessioni di feedback in uscita (gli output non vengono ri-iniettati nel reservoir), la rete possiede "stati eco" (*echo states*) se lo stato della rete $\boldsymbol{x}(n)$ è univocamente determinato da qualsiasi sequenza di input sinistra-infinita $\boldsymbol{u}^{-\infty}$. Più precisamente:

- Per ogni sequenza di input $\boldsymbol{u}(n-1), \boldsymbol{u}(n) \in U^{-\mathbb{N}}$,
- Per tutte le coppie di sequenze di stati $\boldsymbol{x}(n-1), \boldsymbol{x}(n) e \ldots, \boldsymbol{x}'(n-1), \boldsymbol{x}'(n) \in A^{-\mathbb{N}}$,

con $\boldsymbol{x}(i) = T(\boldsymbol{x}(i-1), \boldsymbol{u}(i)) e \boldsymbol{x}'(i) = T(\boldsymbol{x}'(i-1), \boldsymbol{u}(i))$, ossia due sequenze diverse di stati (condizioni iniziali diverse) che evolovono processando la stessa sequenza di input, vale:

$\boldsymbol{x}(n) = \boldsymbol{x}'(n)$.
Equivalentemente esistono "funzioni eco di input" $\boldsymbol{E} = (e_1, \ldots, e_N)$, con:

$$e_i \colon U^{-\mathbb{N}} \to \mathbb{R},$$

tali che per ogni storia di input sinistra-infinita $\ldots, \boldsymbol{u}(n-1), \boldsymbol{u}(n) \in U^{-\mathbb{N}}$, lo stato corrente della rete è: $\boldsymbol{x}(n) = \boldsymbol{E}(\ldots, \boldsymbol{u}(n-1), \boldsymbol{u}(n))$ [6].
In altre parole, non importa da quale stato iniziale si parta: dopo aver processato una lunga sequenza di input, lo stato interno $\boldsymbol{x}(n)$ dipende solo dagli input passati $\boldsymbol{u}(n-1)$, $\boldsymbol{u}(n)$ e non dallo stato iniziale. Questo è fondamentale perché significa che la rete "dimentica" le condizioni iniziali e risponde in modo consistente agli input.
Le unità che compongono la RNN modellano aggregati neuronali omogenei e locali di una rete corticale.
Ogni unità j della rete $j \in [1, N] \subset \mathbb{Z}$ ha uno stato di attività $r_j(t)$ che evolve nel tempo secondo:

$$\tau \dot{r}_j = \Phi(h_j) - r_j, \tag{3}$$

dove τ è la costante di decadimento e $\Phi(h_j)$ è la funzione di attivazione, identica per tutte le unità. Qui si considera il caso limite di accoppiamento ricorrente debole, dove la funzione di attivazione può essere approssimata a una funzione lineare: $\Phi(h_j) = h_j$.
L'input sinaptico h_j è la somma pesata:

$$h_j(t) = \sum_{k=1}^{N} W_{jk} r_k(t) + h_j^{\text{in}}(t),$$

dove W_{jk} sono gli elementi della matrice sinaptica $W \in \mathbb{R}^{N \times N}$ (accoppiamenti interni) e $h_j^{\text{in}}(t)$ è l'input esterno ricevuto dall'unità j.
Nel reservoir computing, l'input esterno è guidato dall'osservabile ω del sistema analizzato, inserito nella rete tramite accoppiamenti sinaptici casuali: $h_j^{\text{in}}(t) = W_j^{\text{in}} \omega(t)$.
Grazie all'alta dimensionalità dello spazio degli stati della RNN, una semplice trasformazione lineare rappresentata da $\boldsymbol{W}^{\text{out}}$ è spesso sufficiente per mappare accuratamente lo stato della RNN in una serie temporale funzionalmente legata a quella di input.
Dati $\boldsymbol{R} \in \mathbb{R}^{N \times T}$ (dove la riga k-esima rappresenta la serie temporale dell'unità k-esima durante un periodo di apprendimento di T passi) e $\boldsymbol{\Omega} \in \mathbb{R}^{M \times T}$ (le cui righe contengono gli M osservabili $\omega_k(t)$ da replicare), la mappatura lineare è calcolata con una regressione *ridge*.
La regressione ridge è una variante della regressione lineare che introduce una penalizzazione quadratica (norma L_2) sui coefficienti per ridurre la loro varianza. Il modello ottimizza la seguente funzione di costo:

$$J(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \beta \|\mathbf{w}\|_2^2 = \mathit{Errore\ quadratico} + \beta \cdot \mathit{Penalizzazione\ dei\ coefficienti},$$

dove $\mathbf{y}$ è il vettore delle osservazioni, $\mathbf{X}$ è la matrice delle variabili predittive, $\mathbf{w}$ è il vettore dei coefficienti, e l'iperparametro $\beta \ll 1$ controlla l'intensità della regolarizzazione.
La soluzione in forma chiusa è:

$$\mathbf{w}_{\text{ridge}} = (\mathbf{X}^T \mathbf{X} + \beta \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y},$$

con $\mathbf{I}$ matrice identità. La regolarizzazione evita di incappare nella non-invertibilità di $\mathbf{X}^T \mathbf{X}$ quando le variabili sono correate [7].

Per il caso in esame, si ha

$$\boldsymbol{W}^{\text{out}} = \boldsymbol{\Omega}\boldsymbol{R}^T(\boldsymbol{R}\boldsymbol{R}^T + \beta\boldsymbol{I})^{-1}, \tag{4}$$

Se l'output target (desiderato) coincide con l'input esterno $\omega(t)$, la mappatura risultante può simulare la stimolazione esterna e prevederne l'evoluzione. In particolare:

$$W^{\text{in}}\omega(t) \approx W^{\text{in}}W^{\text{out}}r(t),$$

porta al sistema autonomo:

$$\tau\dot{\boldsymbol{r}} = \boldsymbol{\Phi}[\widetilde{\boldsymbol{W}}\boldsymbol{r})] - \boldsymbol{r},$$

dove $\widetilde{\boldsymbol{W}} = \boldsymbol{W} + \boldsymbol{W}^{\text{in}}\boldsymbol{W}^{\text{out}}$ è una matrice sinaptica aggiornata.

2.4 Copia sistema non lineare con RNNs lineari

Si consideri un reservoir computing (RC) di una rete neurale ricorrente (RNN) con N unità lineari (lRNN). Lo stato della lRNN è il vettore $\mathbf{r}$(t), gli elementi del quale $r_i(t)$ rappresentano le attività di ciascuna unità. $\mathbf{r}$(t) evolve secondo l' equazione lineare

$$\tau\frac{d\boldsymbol{r}(t)}{dt} = \boldsymbol{W}\boldsymbol{r}(t) + \boldsymbol{W}^{in}\boldsymbol{w}(t) - \boldsymbol{r}(t) \tag{5}$$

La matrice sinaptica $\boldsymbol{W} \in \mathbb{R}^{N\times N}$ controlla l'input ricorrente $\boldsymbol{W}\boldsymbol{r}$, che, insieme ai contributi provenienti dall'ambiente esterno $\boldsymbol{W}^{\text{in}}\boldsymbol{\omega}$(t), costituisce l'input totale $\boldsymbol{h}$ ricevuto da ciascuna unità. In generale, queste correnti (l'attivazione temporale dei neuroni nel reservoir, quindi gli stati) vengono trasformate in modo non lineare da una funzione di guadagno ingresso-uscita $\Phi(\boldsymbol{h})$, che in questo caso assumiamo essere $\Phi(\boldsymbol{h}) = \boldsymbol{h}$ - condizione tipicamente associata a fluttuazioni perturbative delle attività delle unità attorno allo stato di quiete.
Le unità della rete ricevono un input esterno derivante dagli M osservabili ω_i(t) dei sistemi sotto esame, mediato dalla matrice sinaptica $\boldsymbol{W}^{\text{in}} \in \mathbb{R}^{N\times M}$.
Da Eq. (5), lo stato della rete a un qualunque tempo t risulta quindi essere

$$\boldsymbol{r}(t) = \int_{-\infty}^{t} e^{(\boldsymbol{W}-\boldsymbol{I})(t-t')/\tau}\boldsymbol{W}^{\text{in}}\boldsymbol{\omega}(t')\,dt' \tag{6}$$

con $\boldsymbol{I} \in \mathbb{R}^{N\times N}$ matrice identità.
Se l'lRNN manifesta la proprietà detta *sincronizzazione generalizzata / generalized synchronization*[3] in risposta all'input $\boldsymbol{W}^{\text{in}}\boldsymbol{\omega}$(t), allora $\boldsymbol{r}$(t) descrive efficacemente la dinamica dell'intero sistema osservato nel suo spazio degli stati. Ciò avviene perché l'lRNN integra "l'*eco*"[4] delle osservazioni passate, implementando un *embedding* [5] dimensionale a *la Takens*[6]. In questo contesto di Reservoir Computing , per N sufficientemente grande, esiste

[3] *sincronizzazione generalizzata / generalized synchronization* significa che esiste una relazione funzionale (statica) tra gli stati di due sistemi [5].

[4] *eco* definisce che lo stato corrente della rete è determinato da una qualsiasi sequenza degli input passati [6].

[5] *embedding* è la mappatura di oggetti complessi (come ad esmepio parole, stati di una rete) in vettori numerici in uno spazio continuo

[6] *Takens* Dato un sistema dinamico con uno spazio degli stati sconosciuto, ma osservabile tramite una serie temporale, è possibile ricostruire un embedding topologicamente equivalente dello stato originale usando ritardi temporali della serie osservata [10].

una semplice trasformazione lineare in grado di mappare approssimativamente lo stato della rete a un output desiderato.

$$\boldsymbol{w}(t) \approx \boldsymbol{W}^{\text{out}}\boldsymbol{r}(t) = \int_0^\infty \boldsymbol{G}(s)\boldsymbol{w}(t-s)ds \tag{7}$$

con pesi sinaptici $\boldsymbol{W}^{\text{out}} \in \mathbb{R}^{M\times N}$ calcolati mediante una regressione lineare.
I kernel lineari $\boldsymbol{G}(s) = \boldsymbol{W}^{\text{out}}e^{(\boldsymbol{W}-\boldsymbol{I}(s)/\tau}\boldsymbol{W}^{\text{in}} \in \mathbb{R}^{M\times M}$ (cioè le funzioni di Green) risultanti dall'Eq. (6) mostrano la natura autoregressiva dell'approccio RC in questo lRNN. Questo concetto può essere chiarito ricorrendo alla decomposizione spettrale della matrice sinaptica $\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$. $\boldsymbol{\Lambda}$ è la matrice diagonale degli autovalori ($\Lambda_{ii} = \lambda_i$) e gli autovettori sono le colonne di $\boldsymbol{Q}$: $\boldsymbol{W}|Q_i\rangle = \lambda_i\,|Q_i\rangle$. Applicando questa decomposizione all'espressione del kernel risulta $\boldsymbol{G}(s) = \sum_{n=1}^{N}\boldsymbol{J}_n e^{(\lambda_n-1)s/\tau}$, ossia la sovrapposizione di N modi esponenziali moltiplicati per $\boldsymbol{J}_n$, matrici di rango 1 definite come $\boldsymbol{J}_n = \boldsymbol{W}^{\text{out}}\,|Q_n\rangle\,\langle Q_n^{-1}|\,\boldsymbol{W}^{\text{in}}$, con $\langle Q_n^{-1}|$ l'n-esima riga della matrice degli autovettori inversa.
Per ottenere un kernel regolare e compatto, il raggio spettrale $\rho = \max_{n\in[1,N]}\text{Re}\lambda_n$ di $\boldsymbol{W}$ deve essere minore di 1. Sotto questa condizione, l'Eq. (7) descrive un modello autoregressivo (AR) a tempo continuo di ordine N (numero di modi esponenziali in $\boldsymbol{G}(s)$).
Questa relazione tra RC con lRNN e modelli AR generalizza risultati parziali ottenuti nel dominio del tempo discreto. L'ordine del modello AR equivalente può essere molto minore di N poiché molti $\boldsymbol{J}_n$ sono solitamente prossimi a 0.
Se gli osservabili $\boldsymbol{\omega}$(t) forniti in input vengono sostituiti con quelli ricostruiti dall'Eq. (7), la lRNN segue la dinamica autonoma

$$\tau\frac{d\boldsymbol{r}}{dt} = \tilde{\boldsymbol{W}}\boldsymbol{r} - \boldsymbol{r} \tag{8}$$

dove $\widetilde{\boldsymbol{W}} = \boldsymbol{W} + \boldsymbol{W}^{\text{in}}\boldsymbol{W}^{\text{out}}$ è la matrice sinaptica aggiornata (cioè "appresa").
Se la rete è in grado di predire correttamente il futuro dell'output target, l'Eq. (8) fornisce una rappresentazione lineare del sistema sotto analisi. Basandosi, come sopra, sulla decomposizione in autovalori $\widetilde{\boldsymbol{W}} = \widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{\Lambda}}\widetilde{\boldsymbol{Q}}^{-1}$, le proiezioni $\boldsymbol{v}(t) = \widetilde{\boldsymbol{Q}}^{-1}\mathbf{r}$(t) risultano evolvere nel tempo come un insieme di variabili disaccoppiate:

$$v_k(t) = v_k(0)e^{(\tilde{\lambda}_k-1)t/\tau} \quad \forall k \in [1, N]. \tag{9}$$

Analogamente alla decomposizione di Koopman, una combinazione lineare appropriata di questi modi implementa una decomposizione efficace della serie temporale $\boldsymbol{\omega}$(t):

$$\boldsymbol{\omega}(t) = \boldsymbol{W}^{\text{out}}\mathbf{r}(t) \equiv \boldsymbol{\Xi}\boldsymbol{v}(t), \tag{10}$$

con $\boldsymbol{\Xi} = \boldsymbol{W}^{\text{out}}\widetilde{Q}$. La trasformata di Laplace dell'osservabile ricostruito avrà quindi N poli $\sigma_k = (\tilde{\lambda}_k - 1)/\tau$ con i relativi residui, permettendoci di rappresentare le proprietà dinamiche del "gemello digitale" (cioè l'lRNN autonomo), e collegandolo ai modelli AR e alle approssimazioni finite dell'operatore di Koopman. Infatti, il k-esimo residuo è proporzionale a $\Xi_{jk}v_k(0)$ e fornisce, in valore assoluto, il contributo (la rilevanza) del k-esimo modo nella descrizione dell'osservabile ω_j, dato lo stato iniziale della rete $\mathbf{r}(0)$.
Come prova per la teoria sviluppata sopra, viene considerato il seguente sistema non lineare:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{C_x x} \\ \dot{\boldsymbol{y}} = \boldsymbol{C_y y} + \boldsymbol{C_{yx} x} \odot \boldsymbol{x} \end{cases} \tag{11}$$

scegliendo $\boldsymbol{C_x}$ e $\boldsymbol{C_{yx}}$ come matrici diagonali -**I**, questo sistema può essere mappato su uno lineare introducendo un nuovo insieme di osservabili $\boldsymbol{z} = \boldsymbol{x} \odot \boldsymbol{x}$. In questo caso, il relativo generatore infinitesimale degli operatori di Koopman è equivalente alla matrice finita $\boldsymbol{K}$ con blocchi diagonali $\boldsymbol{C_x}$, $2\boldsymbol{C_x}$, $\boldsymbol{C_y}$.
Il sistema derivante da Eq. (11) applicando la sostituzione risulta essere:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{C_x x} \\ \dot{\boldsymbol{y}} = \boldsymbol{C_y y} + \boldsymbol{C_{yx} z} \\ \dot{\boldsymbol{z}} = 2\boldsymbol{C_x z} \end{cases} \tag{12}$$

È stato quindi addestrato un lRNN a replicare questo sistema a partire dagli osservabili $\boldsymbol{\omega} = [\boldsymbol{x}; \boldsymbol{y}]$ con $\boldsymbol{x} \in \mathbb{R}^5$ e $\boldsymbol{y} \in \mathbb{R}^{10}$ [1].

2.5 Operatore di Koopman

L'operatore di Koopman a tempo discreto $K^1 \colon F \to F$ è un operatore lineare definito in uno spazio infinito-dimensionale F di osservabili dello stato del sistema . L'azione dell'operatore di Koopman su un'osservabile g è descritta dall'equazione:

$$K_1 g(\boldsymbol{x}_t) = g(\boldsymbol{x}_{t+1}). \tag{13}$$

Qui, $\boldsymbol{x}$ rappresenta un punto nello spazio degli stati. Le autofunzioni dell'operatore di Koopman sono esse stesse potenziali osservabili del sistema e hanno la proprietà peculiare di evolversi linearmente nel tempo:

$$K^1 \psi_k(\boldsymbol{x}_t) = \psi_k(\boldsymbol{x}_{t+1}) = \lambda_k^1 \psi_k(\boldsymbol{x}_t). \tag{14}$$

Queste autofunzioni $\psi_k(\boldsymbol{x}_t)$ costituiscono una base dello spazio F, permettendo di rappresentare tutte le possibili osservabili del sistema come una combinazione lineare di funzioni che evolvono linearmente.
In generale, esiste una famiglia di operatori di Koopman, K_t, che fa avanzare una funzione nel tempo di un intervallo t:

$$K^t g(\boldsymbol{x}) = g(S^t(\boldsymbol{x})), \tag{15}$$

dove S^t denota l'operatore di flusso temporale, con $\boldsymbol{x}(t) = S^t[\boldsymbol{x}(0)]$.
Il generatore infinitesimale della famiglia di operatori di Koopman

$$\mathcal{L} g = \lim_{t \to 0} \frac{K^t g - g}{t}$$

è l'operatore di Lie, che valuta la variazione temporale delle osservabili:

$$\mathcal{L} g(\boldsymbol{x}(t)) = \frac{d}{dt} g(\boldsymbol{x}(t)).$$

La famiglia di operatori di Koopman può quindi essere espressa in termini di questo generatore come $K^t = e^{\mathcal{L}t}$ [1].

2.6 Test R^2

R^2 o coefficiente di determinazione, è una misura statistica che valuta la bontà dell'adattamento di un modello ai dati osservati in ingresso. Nell' apprendimento automatico, i modelli di regressione puntano a prevedere un valore numerico continuo basato sui dati in ingresso. Una volta addestrati, è importante valutarne le prestazioni e determinare quanto bene catturano i pattern dei dati, compito svolto da R^2. Un valore R^2 pari a 1 (o 100%) implica che il modello predice perfettamente la variabile dipendente, al contrario, 0 (o 0%) indica che il modello non riesce a spiegare alcuna variabilità nella variabile dipendente. Per calcolare R^2, viene confrontata la somma delle differenze al quadrato tra i valori osservati e i valori previsti dal modello (SSR) con la somma totale delle differenze quadrate tra i valori osservati e la loro media (SST), con formula:

$$R^2 = 1 - \frac{SSR}{SST} \tag{13}$$

Per capire se il modello ha subito overfitting analizzando l'$\mathbf{R^2}$, è necessario separare il set dei dati in ingresso in *training set* e *test set*, calcolare $\mathbf{R^2}$ su ognuno e confrontarli.
Se l'$\mathbf{R^2}$ sul training set è molto alto (ad esempio superiore a 0,.) mentre l'$\mathbf{R^2}$ sul test set è significativamente più basso (con un divario maggiore di 0.1-0.2 punti), questo è un chiaro indicatore di overfitting. Il modello ha memorizzato i dati di training invece di apprendere pattern generalizzabili, mostrando così ottime performance sui dati noti ma scarse capacità predittive su nuovi dati. Un $\mathbf{R^2}$ di test molto inferiore a quello di training segnala che il modello si adatta anche al rumore e ai dettagli specifici del dataset di addestramento, compromettendo la sua efficacia su dati non visti.

2.7 Matrici di corelazione funzionale FC

Una matrice FC si calcola partendo da due variabili neurali X e Y, dalle quali viene ricavato:

$$r_{XY} = \frac{\sum_{i=1}^{n}(X_i - \mu_X)(Y_i - \mu_Y)}{\sqrt{\sum_{i=1}^{n}(X_i - \mu_X)^2}\sqrt{\sum_{i=1}^{n}(Y_i - \mu_Y)^2}} \tag{14}$$

dove:

- X_i, Y_i = valori al timepoint i
- μ_X, μ_Y = valori medi
- n = numero di timepoints

creando quindi la matrice FC:

$$FC = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mm} \end{bmatrix}$$

la quale soddisfa le seguenti proprietà:

$$\begin{aligned} \text{Simmetria:} \quad & r_{ij} = r_{ji} \\ \text{Diagonale:} \quad & r_{ii} = 1 \\ \text{Range:} \quad & -1 \leq r_{ij} \leq 1 \end{aligned}$$

Le matrici di correlazione funzionale, di un sitema complesso, sono necessarie per mapparne e quantificarne le relazioni dinamiche e la connettività, rappresentandone l'attività coordinata.
A differenza di quanto il nome potrebbe suggerire, non analizzano funzioni matematiche, ma l'indipendenza statistica tra le varie componenti del sistema mentre questo è in funzione. Ogni elemento assume un valore compreso tra -1 e 1:

- 1: correlazione perfetta positiva: aree che si "accendono" insieme.
- -1: correlazione perfetta negativa (anticorrelazione): aree che si "spengono" a vicenda.
- 0: nessuna correlazione.

Lo scopo principale è quindi quello di identificare pattern di sincronizzazione, per rivelare quali parti del sistema comunicano e lavorano insieme, confrontando stati diversi e permettendo quindi di distinguere il funzionamento di un cervello sano rispetto a uno affetto da una patologia neurologica.

3 Simulazioni

In informatica, i metodi numerici sono spesso preferiti alle soluzioni esatte per risolvere equazioni differenziali per diverse ragioni fondamentali. Molte equazioni differenziali, specialmente quelle non lineari o a coefficienti variabili, non ammettono una soluzione esatta. Inoltre, anche quando una soluzione analitica esiste, spesso è espressa in termini di funzioni speciali che devono comunque essere valutate numericamente. In questi casi, l'alternativa più efficiente è l'uso di metodi numerici. Infatti, metodi numerici ottimizzati, rispetto al calcolo esatto, possono essere più veloci nel produrre una soluzione sufficientemente accurata. Inoltre, in problemi pratici, le condizioni ai bordi possono essere irregolari o note solo numericamente, rendendoli più flessibili coerenti con la natura approssimata del problema. Quindi, i metodi numerici sacrificano la precisione teorica per guadagnare generalità, efficienza e applicabilità a problemi reali. L'accuratezza può essere controllata e bilanciata con le esigenze e le risorse computazionali disponibili.
In questa tesi sono stati applicati due metodi numerici per risolvere Eq. (11): il metodo di Eulero e il metodo di Runge-Kutta.

3.1 Metodo di Eulero

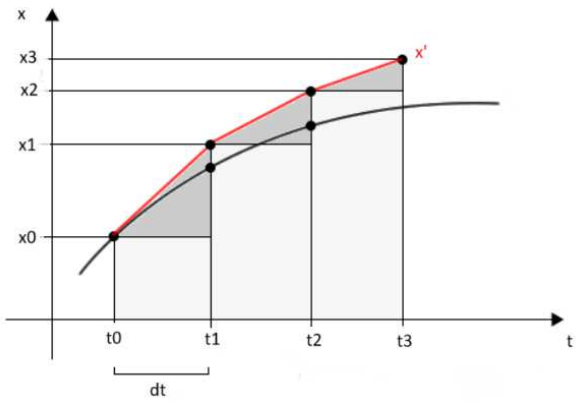


Figure 4: Grafico qualitativo metodo di Eulero

Il concetto fondamentale su cui si basa il seguente metodo è un principio molto semplice. Supponiamo che una particella si muova in modo tale che, al tempo t_0, la sua posizione sia x_0 e che, in quell'istante, la velocità sia nota e pari a v_0. Il principio essenziale è che, in un intervallo di tempo sufficientemente "*breve*"[7] lo spostamento sarà approssimativamente uguale alla variazione temporale moltiplicata per v_0.
Se il moto della particella è governato da un'equazione differenziale, il valore di v_0 sarà noto in funzione di t_0 e x_0. Pertanto, dati t_0 e x_0, la soluzione in t_1 (assunto vicino a t_0) può essere calcolata come:

$$x_1 = x_0 + (t_1 - t_0)v_0 \tag{15}$$

dove v_0 dipende unicamente dai valori noti t_0, x_0 (Fig. (4)).
Assumendo che v_1, calcolato dall'equazione differenziale usando t_1 e x_1, sia sufficientemente accurato, si può procedere con un secondo passo per determinare x_2, approssimazione della soluzione in t_2, mediante la formula:

[7] *breve* determina un intervallo ti tempo in cui la velocità non cambia significatamente rispetto a v_0.

$$x_2 = x_1 + (t_2 - t_1)v_1. \tag{16}$$

Una sequenza di approssimazioni x_1, x_2, x_3, ... per la soluzione dell'equazione differenziale nei punti t_1, t_2, t_3, ... consentirà di ottenere risultati accettabili anche a tempi sempre più distanti dai dati iniziali.
L'interpretazione del metodo di Eulero è più ampia della semplice descrizione del moto di una particella lungo una linea. La variabile dipendente x può anche essere vettoriale come nel caso di Eq. (11), interpretato come un insieme di componenti scalari $x_1, x_2, ..., x_N$:

$$\boldsymbol{x(t)} = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_N(t) \end{bmatrix}.$$

L'equazione differenziale, insieme ai dati iniziali che determinano l'evoluzione delle componenti di $\boldsymbol{x}$ al variare della variabile temporale, può essere scritta nella forma:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}(t)), \qquad \boldsymbol{x}(t_0) = x_0 \tag{17}$$

Tuttavia, è spesso conveniente esprimere le singole componenti di $\boldsymbol{f}$ come funzioni scalari di t e del vettore $\boldsymbol{x}(t)$ (o equivalentemente, delle sue componenti). Analogamente, i dati iniziali possono essere specificati per ogni componente $x_{1_0}, x_{2_0}, ..., x_{N_0}$ di $\boldsymbol{x_0}$.

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \\ \vdots \\ \dot{x_N}(t) \end{bmatrix} = \boldsymbol{f}(t, x_1(t), x_2(t), \ldots, x_N(t))$$

$$\boldsymbol{x}(t_0) = \begin{bmatrix} x_{1_0} \\ x_{2_0} \\ \vdots \\ x_{N_0} \end{bmatrix}$$

Quindi per semplificare e generalizzare le Eq. (15) e Eq. (16), riscrivo $(t_N - t_{N-1}) = \delta t$, e quindi N-esimo passo iterativo come:

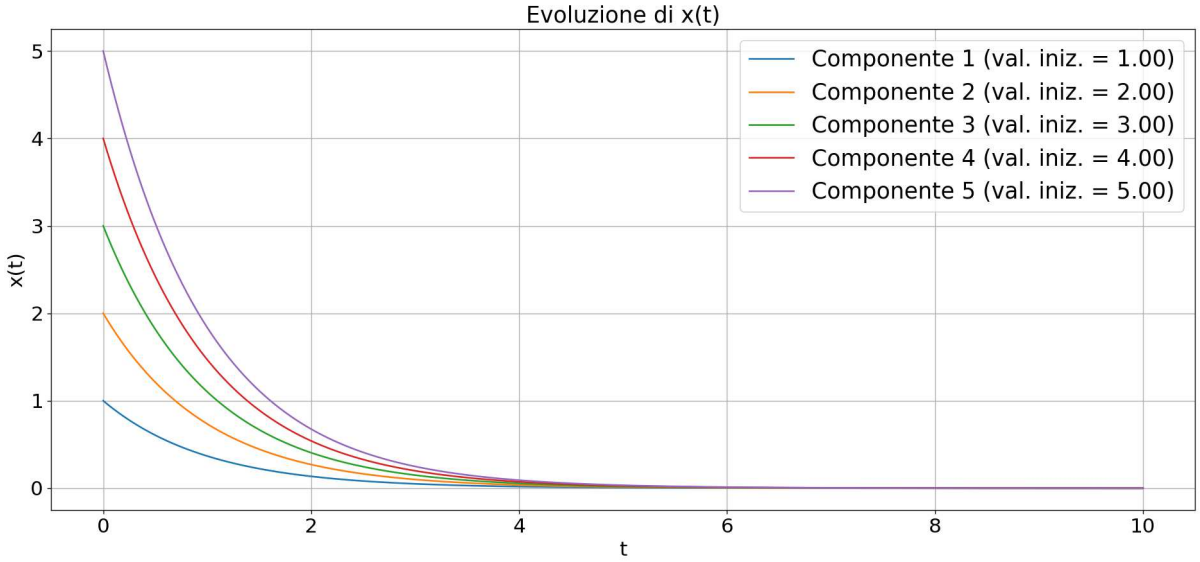
$$\boldsymbol{x}_N = \boldsymbol{x}_{N-1} + \delta t \dot{\boldsymbol{x}}_{N-1} \tag{18}$$

Questo metodo però, per essere accurato, richiede δt molto piccoli. Il metodo di Eulero semplice si ottiene utilizzando un solo termine della serie di Taylor per $\boldsymbol{x}(t)$ espansa intorno a $t = t_0$. Il metodo di Eulero modificato può essere derivato utilizzando due termini: $\boldsymbol{x}(t_0 + \delta t) = \boldsymbol{x}(t_0) + \dot{\boldsymbol{x}}(t_0)\delta t + \ddot{\boldsymbol{x}}(t_0)\frac{\delta t^2}{2}$, se sostituiamo la derivata seconda con un'approssimazione alle differenze all'indietro, $\boldsymbol{x}(t_0 + \delta t) = \boldsymbol{x}(t_0) + \frac{\dot{\boldsymbol{x}}(t_0) + \dot{\boldsymbol{x}}(t_0 + \delta t)}{2}\delta t$ otteniamo la formula del metodo modificato che ha una maggiore precisione [8].

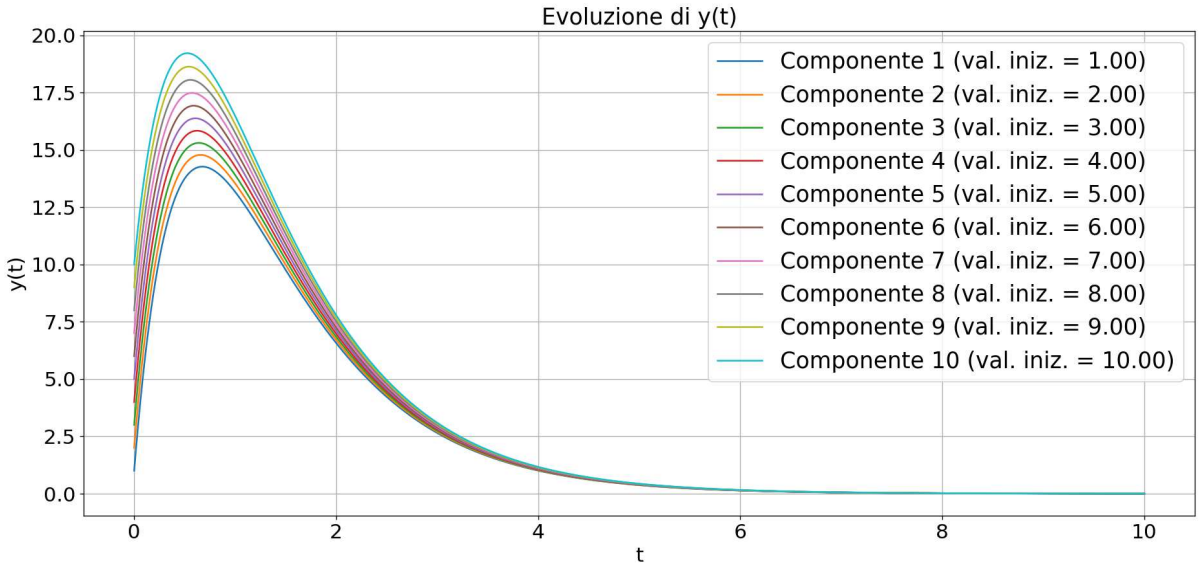
3.1.1 Implementazione metodo di Eulero

È stata implementata Eq. (11) con un programma in Python secondo il metodo di Eulero descritto nella sezione (3.1).

In particolare è stato deciso un $\Delta t = 10$, con δt =0.001, quindi un totale di 10 000 iterazioni.
Per facilitare la visualizzazione dell'andamento delle varie componenti è stato deciso di dargli valori crescenti da 1 a 10.
I grafici risultanti sono:



(a)



(b)

Figure 5: Soluzioni di Eulero: (a) varibile $\boldsymbol{x}$; (b) variabile $\boldsymbol{y}$.

3.2 Metodo di Runge-Kutta

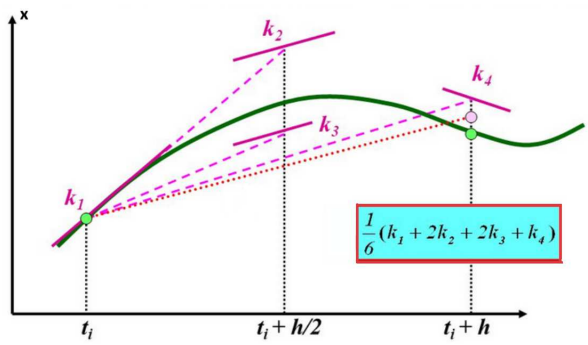


Figure 6: Grafico qualitativo metodo di Runge-Kutta

Due matematici tedeschi, Runge e Kutta, svilupparono algoritmi basati sull'uso di più di due termini della serie. Il metodo di Eulero modificato è un metodo di Runge-Kutta del secondo ordine. I metodi di Runge-Kutta del secondo ordine si ottengono utilizzando una media ponderata di due incrementi di $\boldsymbol{x}(\boldsymbol{t_0})$, k_1 e K_2. Per l'equazione $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}(t))$:

$$\begin{aligned} \boldsymbol{x}_{N+1} &= \boldsymbol{x}_N + aK_1 + bK_2 \\ K_1 &= \delta t f(t_N, \boldsymbol{x}_N) \\ K_2 &= \delta t \boldsymbol{f}(t_N + \alpha\delta t, \boldsymbol{x}_N + \beta K_1) \end{aligned} \tag{19}$$

Si possono considerare i valori K_1 e K_2 come stime della variazione di $\boldsymbol{x}$ quando t avanza di δt, poiché rappresentano il prodotto tra la variazione in t e il valore della pendenza della curva $\dot{\boldsymbol{x}}$.
I metodi di Runge-Kutta utilizzano sempre la stima semplice di Eulero come prima approssimazione di $\Delta\boldsymbol{x}$; l'altra stima viene ottenuta incrementando t e $\boldsymbol{x}$ rispettivamente di frazioni α e β di δt e della precedente stima $\Delta\boldsymbol{x}$ (ovvero k_1).
Il nostro problema consiste nello sviluppare uno schema per la scelta dei quattro parametri a, b, α e β. Questo viene fatto cercando di far coincidere il più possibile Eq. (17) con lo sviluppo in serie di Taylor, in cui le derivate di $\boldsymbol{x}$ sono espresse in termini di $\boldsymbol{f}$, partendo dalla relazione $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x})$,

$$\begin{aligned} \boldsymbol{x}_{N+1} &= \boldsymbol{x}_N + \boldsymbol{f}(t_N, \boldsymbol{x}_N)\delta t + \dot{\boldsymbol{f}}(t_N, \boldsymbol{x}_N)\frac{\delta t^2}{2} \\ &= \boldsymbol{x}_N + \boldsymbol{f}_N \delta t + (\boldsymbol{f}_t + \boldsymbol{f}_{\boldsymbol{x}}\boldsymbol{f})_N \frac{\delta t^2}{2} \end{aligned} \tag{20}$$

riscrivendo l'equazione (14) usando la definizione di K_1 e K_2 si ottiene

$$\begin{aligned} \boldsymbol{x}_{N+1} &= \boldsymbol{x}_N + a\boldsymbol{f}_N\delta t + b\delta t\boldsymbol{f}(t_N + \alpha\delta t, \boldsymbol{x}_N + \beta\delta t\boldsymbol{f}(t_N, \boldsymbol{x}_N)) \\ &\approx \boldsymbol{x}_N + a\boldsymbol{f}_N\delta t + b\delta t\left(\boldsymbol{f} + \alpha\delta t\boldsymbol{f}_t + \boldsymbol{f}_{\boldsymbol{x}}\beta\delta t\boldsymbol{f}\right)_N \\ &\approx \boldsymbol{x}_N + (a+b)\boldsymbol{f}_N\delta t + \delta t^2\left(\alpha b\boldsymbol{f}_t + \beta b\boldsymbol{f}_{\boldsymbol{x}}\boldsymbol{f}\right)_N \end{aligned} \tag{21}$$

Le equazioni (18) e (19) sono uguali se

$$a + b = 1, \quad \alpha b = \tfrac{1}{2}, \quad \beta b = \tfrac{1}{2}.$$

Si noti che solo tre equazioni devono essere soddisfatte dalle quattro incognite. Possiamo scegliere arbitrariamente un valore (con minime restrizioni), ottenendo così una famiglia di metodi del secondo ordine.

- Una scelta possibile è $a = 0$, $b = 1$; $\alpha = \frac{1}{2}$, $\beta = \frac{1}{2}$. Questa configurazione dà origine al *metodo del punto medio.*
- Un'altra opzione è $a = \frac{1}{2}$, $b = \frac{1}{2}$; $\alpha = 1$, $\beta = 1$, che corrisponde al metodo di Eulero modificato.

Tutte queste rappresentano casi particolari dei metodi di Runge-Kutta del secondo ordine. Nella simulazione è stato adottato il metodo di di Runge-Kutta del quarto ordine, i diversi termini si derivano seguendo un approccio simile a quello appena illustrato. La maggiore complessità deriva dalla necessità di confrontare i termini fino all'ordine δt^4, il che produce un sistema di 11 equazioni in 13 incognite. Questo sistema può essere risolto scegliendo arbitrariamente 2 incognite. La scelta più comune porta al seguente risultato:

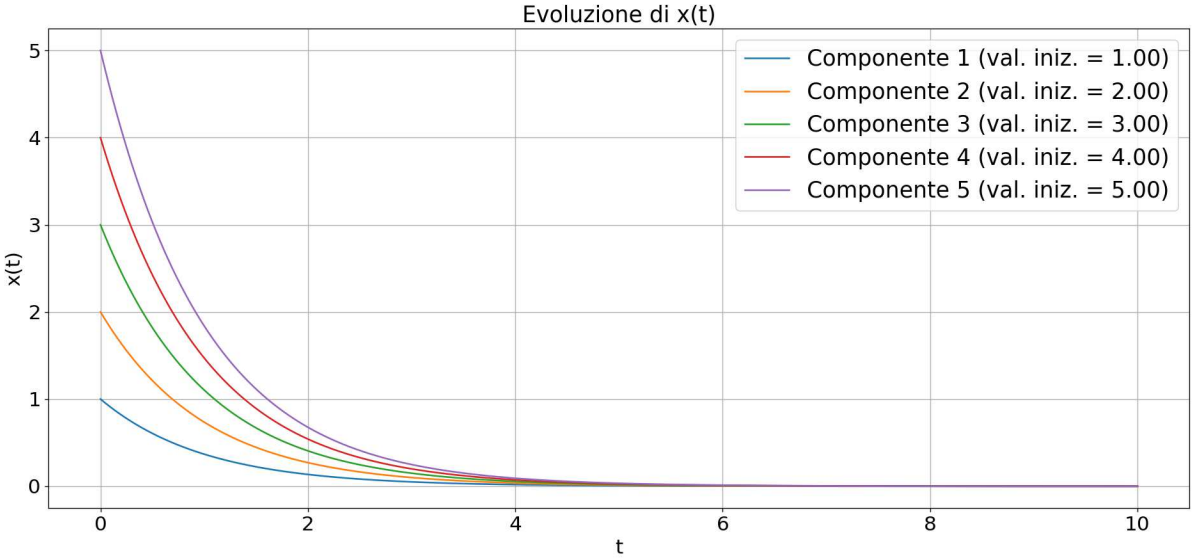
$$\begin{aligned}
\boldsymbol{x}_{N+1} &= \boldsymbol{x}_N + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\
K_1 &= \delta t \boldsymbol{f}(t_n, \boldsymbol{x}_N) \\
K_2 &= \delta t \boldsymbol{f}(t_n + \frac{1}{2}\delta t, \boldsymbol{x}_N + \frac{1}{2}K_1) \\
K_3 &= \delta t \boldsymbol{f}(t_n + \frac{1}{2}\delta t, \boldsymbol{x}_N + \frac{1}{2}K_2) \\
K_4 &= \delta t \boldsymbol{f}(t_n + \delta t, \boldsymbol{x}_N + K_3)
\end{aligned} \tag{22}$$

Il termine di errore locale per il metodo di Runge-Kutta del quarto ordine è $O(\delta t^5)$, mentre l'errore globale risulta essere $O(\delta t^4)$. Questo metodo è computazionalmente più efficiente del metodo di Eulero modificato perché, sebbene richieda più passaggi, consente di utilizzare passi molto più grandi a parità di accuratezza [9].

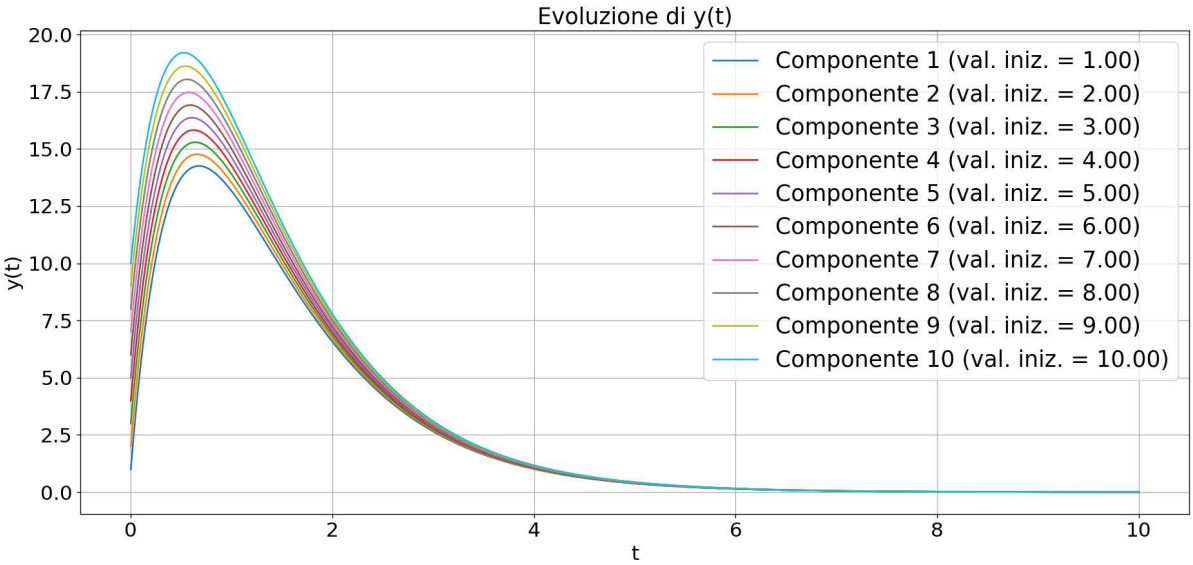
3.2.1 Implementazione metodo di Runge-Kutta

È stata implementata l' Eq. (11) con un programma in Python secondo il metodo di Runge-Kutta descritto nella sezione (3.2).
In particolare è stato deciso un $\Delta t = 10$, con δt =0.001, quindi un totale di 10 000 iterazioni.
Per facilitare la visualizzazione dell'andamento delle varie componenti è stato deciso di dargli valori crescenti da 1 a 10.

I grafici risultanti sono:



(a)



(b)

Figure 7: Soluzioni di Runge-Kutta: (a) varibile $\boldsymbol{x}$; (b) variabile $\boldsymbol{y}$.

3.3 Confronto tra i due metodi

Per confrontare i 2 metodi è stato deciso di paragonarli rispetto alla soluzione esatta, solamente per il vettore $\boldsymbol{x}$. È stata implementata la prima delle 3 equazioni del sistema Eq. (11) con un programma in Python, prima con la soluzione esatta con equazione $e^{C_x t}\boldsymbol{x_0}$ e successivamente sia con il metodo di Eulero descritto nella sezione (3.1) che con il metodo di Runge-Kutta descritto nella sezione (3.2).
In particolare è stato deciso un $\Delta t = 10$, con δt =0.001, quindi un totale di 10 000 iterazioni.
Per facilitare la visualizzazione dell'andamento delle varie componenti è stato deciso di dargli valori crescenti da 1 a 5.
I grafici risultanti sono:

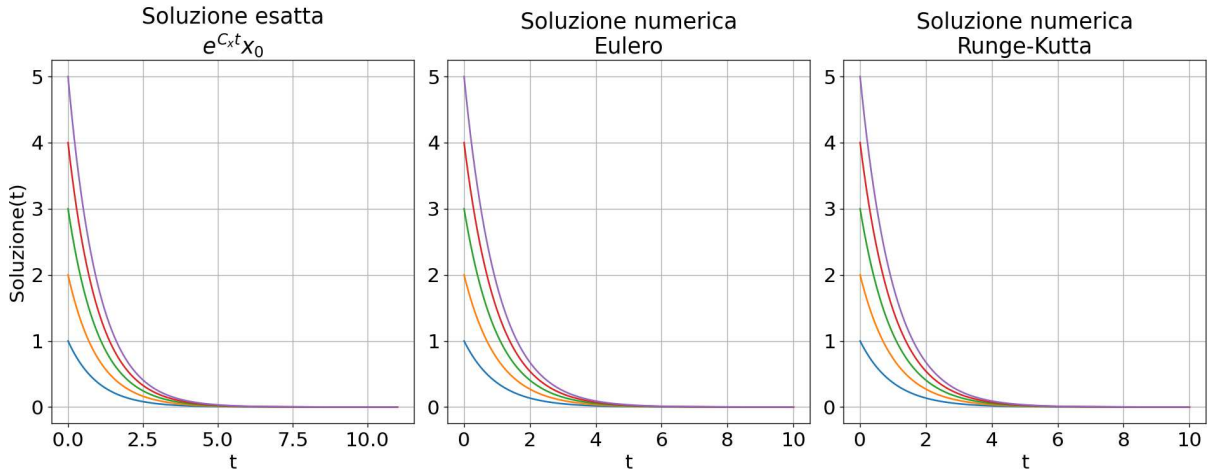


Figure 8: Confronto soluzione esatta e soluzioni numeriche

Come si nota dai i grafici le tre simulazioni risultano pressochè identiche.
Per evidenziare le differenze, sono stati calcolati i gli scarti in valore assolutro dei tra ciascono dei due metodi numerici e il metodo esatto.
I grafici risultano:

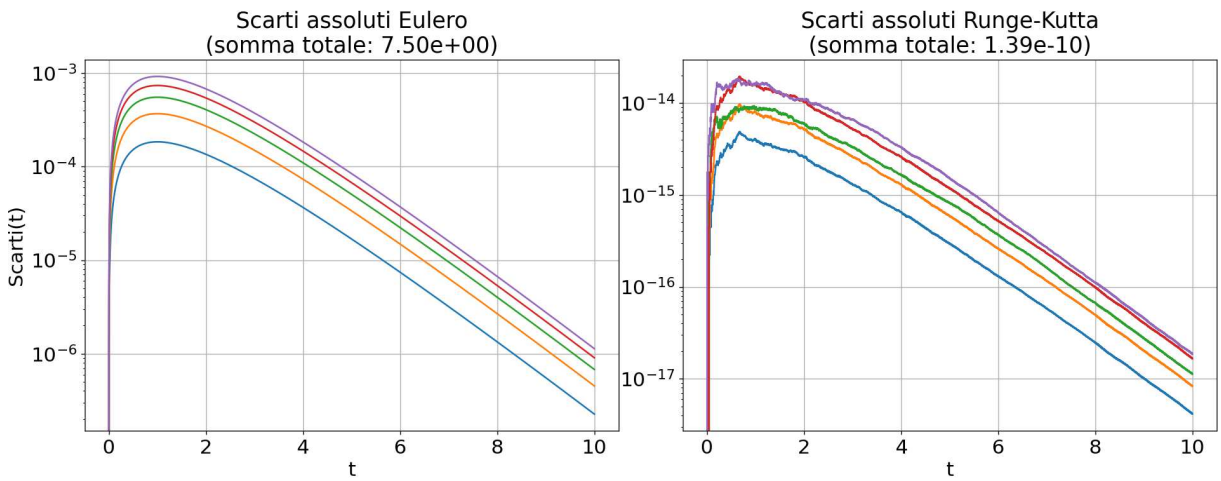


Figure 9: Confronto grafici scarti

È stato deciso di usare la scala logaritmica per l'asse delle ordinate per facilitare la visualizzazione degli ordini di grandezza.
L'andamento dei 2 grafici risulta molto simile, ma il metodo di Runge-Kutta, è molto più preciso rispetto al metodo di Eulero, la somma degli scarti infatti è di 10 ordini di grandezza inferiore. Per questo motivo come dati in ingresso per la simulazione successiva (sezione (4.1)) sono stati scelti i valori in uscita dalla simulazione con metodo di Runge-kutta.

4 Implementazione di un 'digital twin' tramite una rete neurale

4.1 Applicazione alle simulazioni numeriche

I dati in ingresso considerati questa sezione sono le simulazioni numeriche ottenute nella sezione (3.3)). In particolare, ci concentriamo sulel simulazioni ottenute con il metodo di Runge-Kutta (sezione (3.2)), dato che tale metodo ha mostrato una precisone ottimale. L'Eq. (5) è stata portata nella forma

$$\dot{\boldsymbol{r}}(t) = \frac{1}{\tau}(\boldsymbol{W}\boldsymbol{r}(t) + \boldsymbol{W}^{in}\boldsymbol{w}(t) - \boldsymbol{r}(t)) \tag{23}$$

ed è stata implementata con il metodo di Runge-Kutta per ottenere $\boldsymbol{r}(t)$.
I parametri della rete sono stati selezionati per ottimizzare l'accuratezza della predizione. È stato scelto un reservoir di 500 neuroni. Come matrice sinaptica iniziale $\boldsymbol{W}$ è stata scelta quella associata a una connettività ad anello (cioè $W_{i,i+1} = W_{N,1} = \rho$ sono gli unici elementi non nulli), nota per avere prestazioni ottimali nel caso di unità lineari. Il raggio spettrale ρ della matrice sinaptica, è impostato a 0.5 (Fig. (10)), e la costante di tempo τ è 1.0. La deviazione standard g^{in} della matrice di input $\boldsymbol{W}^{in}$ è fissata a 1.

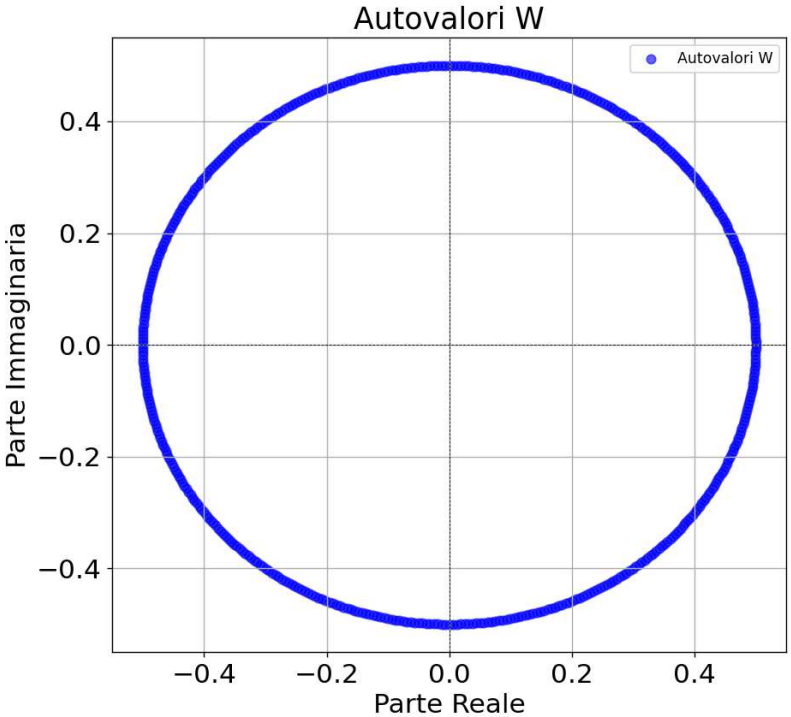
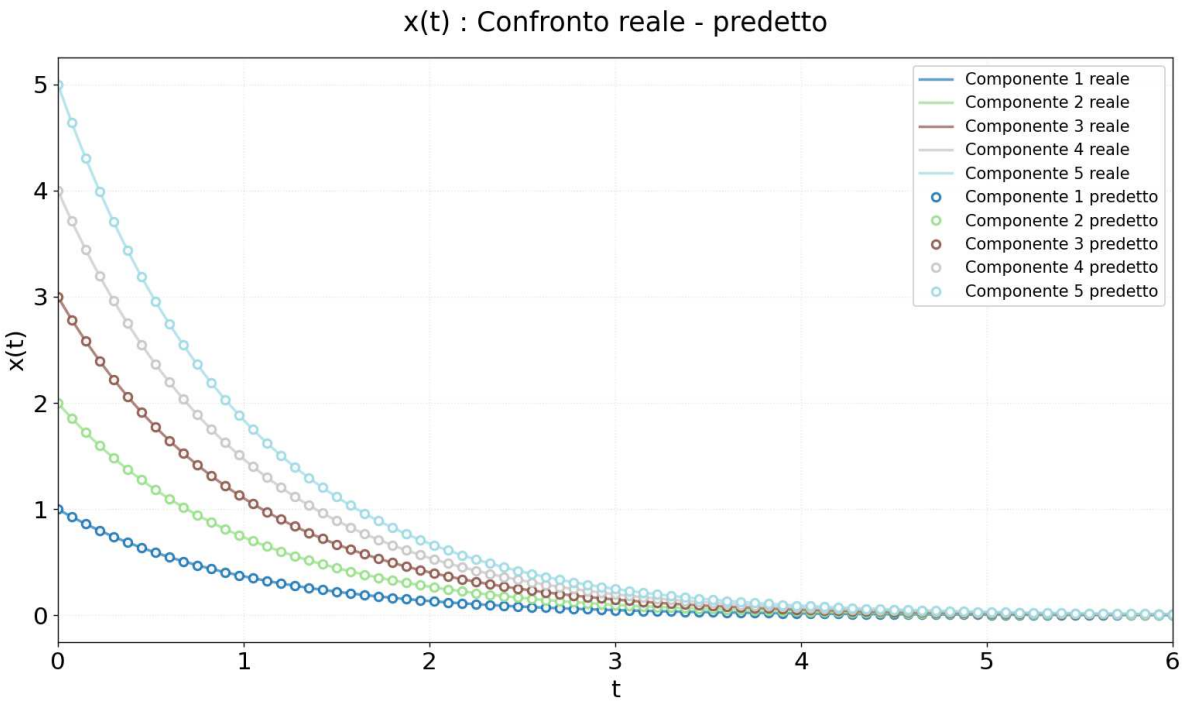
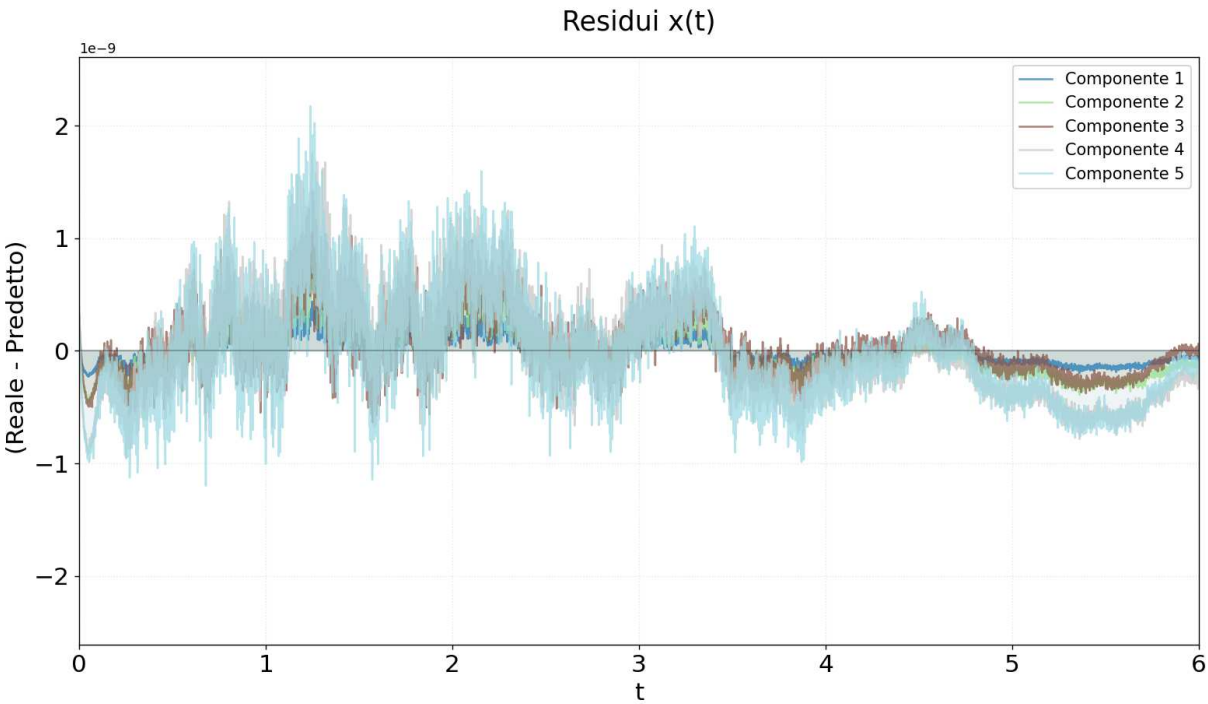


Figure 10: Grafico nel piano complesso degli autovalori della matrice sinaptica $\boldsymbol{W}$

È stato addestrato il modello con una regressione *Ridge* e in seguito, come output, sono stati predetti i dati in input con l'obiettivo di riprodurli fedelmente. In questo caso, trattandosi di predizione di equazioni differenziali lineari, l'iperparametro β, già incontrato nella sezione (2.3), è stato impostato a 0.
In seguito sono stati calcolati i residui tra i dati reali e predetti e l'R^2 (fare riferimento alla sezione (2.6)).

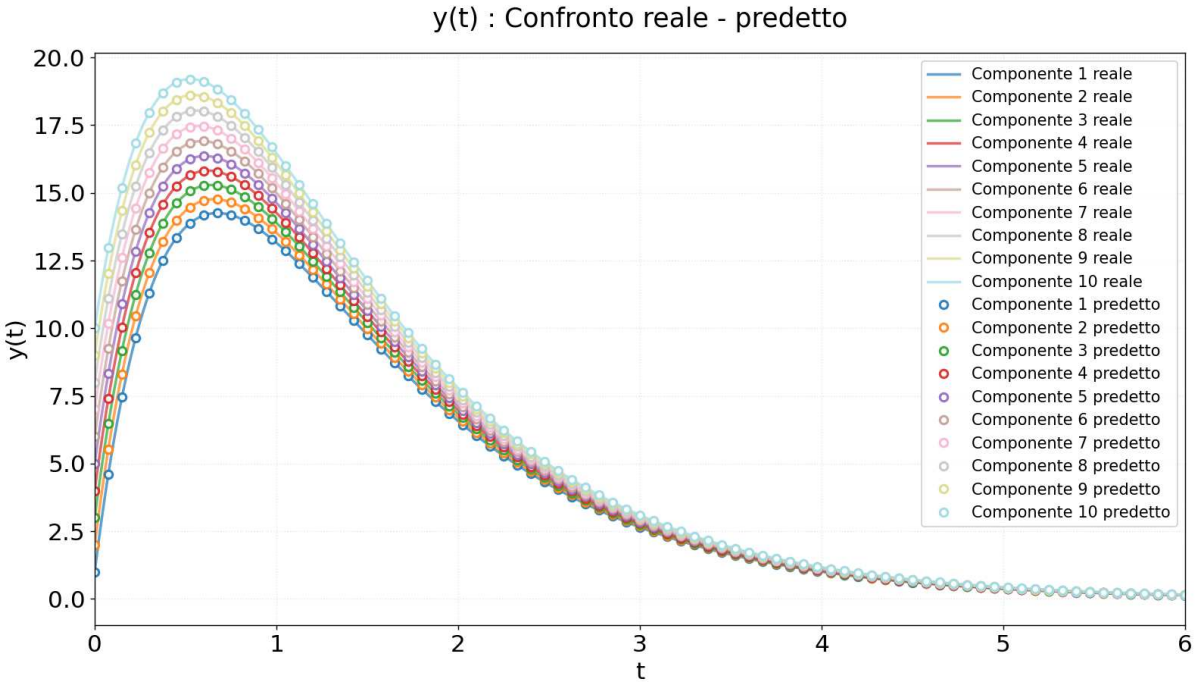


(a)

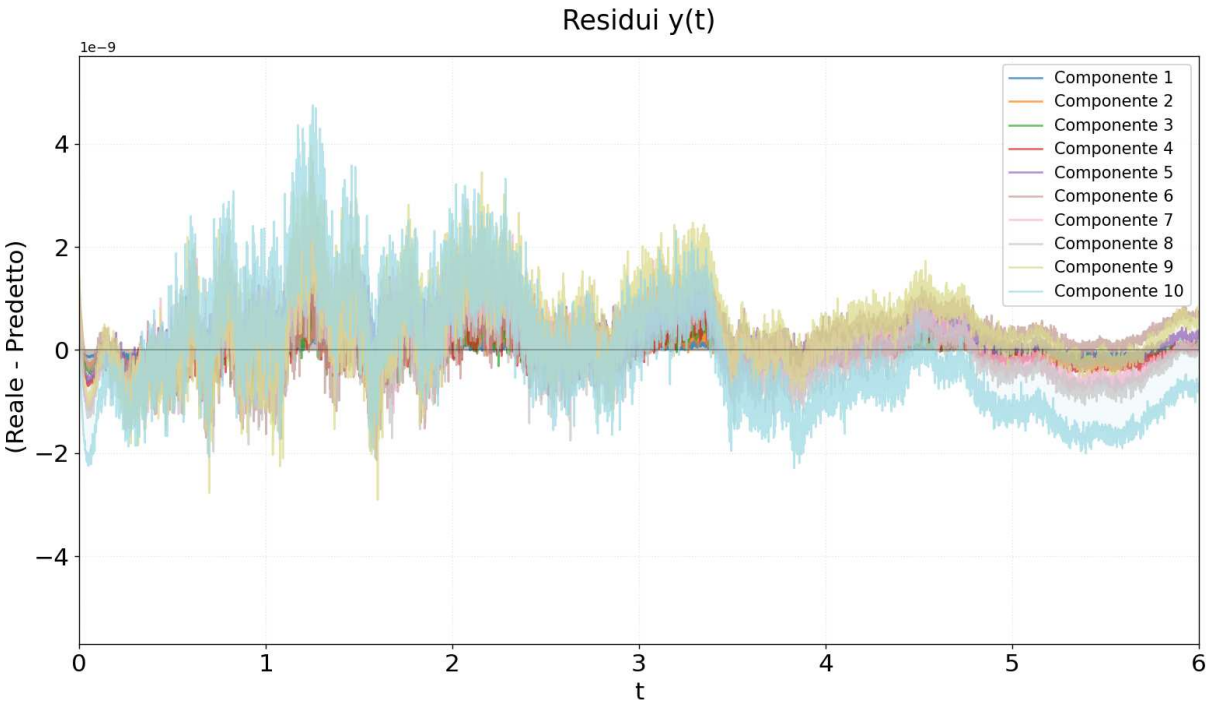


(b)

Figure 11: Predizione x: (a) Grafico confronto; (b) Residui dati reali-predetti.



(a)

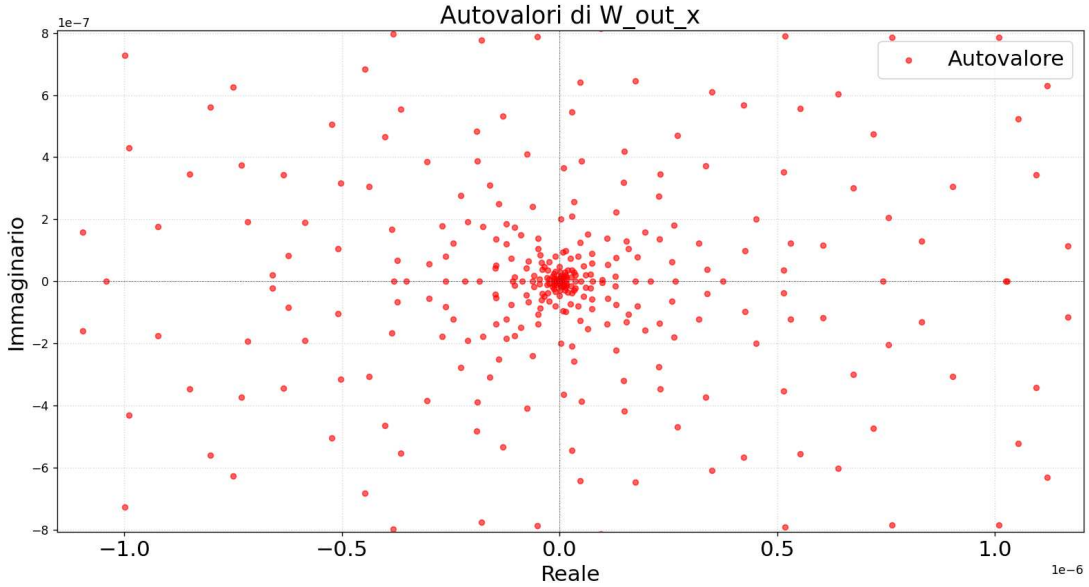


(b)

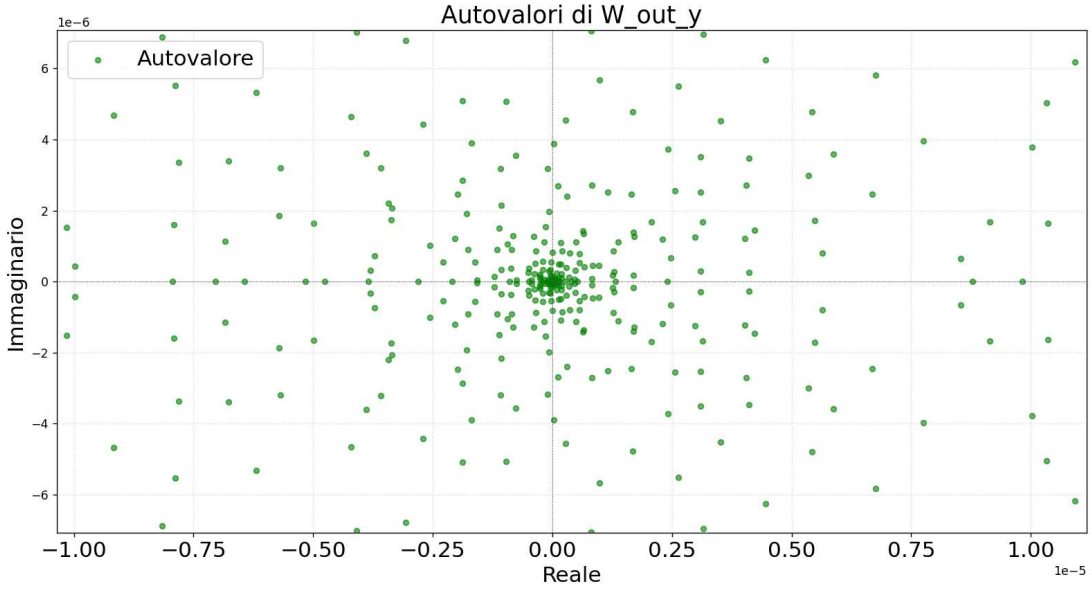
Figure 12: Predizione y: (a) Grafico confronto; (b) Residui dati reali-predetti.

Per favorire la leggibilità dei grafici è stato plottato un dato reale ogni 75 passi. Come si nota dai grafici (11a) e (12a) i dati predetti combaciano bene con i dati reali. In entrambi i casi (Fig. (11b) e Fig. (12b)) le differenze massime risultano dell'ordine di 10^{-9}. In particolare per $\boldsymbol{x}(t)$ la media dei residui risulta $-9.7 \cdot 10^{-11}$, mentre per $\boldsymbol{y}$(t) riuslta $3.3 \cdot 10^{-12}$.

È stato calcolato l'R^2 sia per $\boldsymbol{x}(t)$ che per $\boldsymbol{y}(t)$, il quale risulta 1.00 per entrambe le variabili. Questo significa che il modello predice esattamente i valori in ingresso.
Dai grafici (11b) (12b) si nota che i residui assumono un andamento oscillatorio. Questo può essere attribuito alla natura complessa degli autovalori delle matrici $\boldsymbol{W}^{\text{out}}$, ricavate secondo la relazione di Eq. (7). Gli autovalori sono mostrati in Fig. 13a, Fig. 13b.



(a)



(b)

Figure 13: (a) Autovalori nel piano complesso di $\boldsymbol{W}_{\boldsymbol{x}}^{\text{out}}$; (b) Autovalori nel piano complesso di $\boldsymbol{W}_{\boldsymbol{y}}^{\text{out}}$.

4.2 Applicazione a dati reali

Il programma implementa un sistema di Reservoir Computing per analizzare e predire dati di attività cerebrale. I dati provengono dal lavoro di Kim et al. (2022), in cui è stata usata la risonanza magnetica funzionale per registrare l'attività cerebrale di 16 partecipanti, ciascuno dei quali è stato sottoposto a diversi (27) cicli di registrazione. In alcuni (20/27) cicli i partecipanti sono stati sottoposti a una sequenza di stimoli visivi ripartiti in diverse categorie (facce, scene, oggetti, etc.); nei rimanenti, sono rimasti a riposo (cioè in assenza di stimoli). Gli autori hanno individuato un insieme di 32 aree (prevantemente occipitali e parietali) che esibivano una risposta marcata a categorie specifiche di stimoli visivi. Per tutti i dettagli sull'esperimento nonché la pre-elaborazione dei dati si rimanda a Kim et al., 2022. I dati qui analizzati sono riferiti a una singola area cerebrale durante un singolo ciclo (300s di registrazione) a riposo di un singolo soggetto. La risoluzione temporale della registrazione era 1s, e l'area analizzata comprende un totale di 1108 vertici corticali: disponiamo pertanto di un insieme di 1108 serie temporali, ciascuna di 300 punti.
L'equazione dello stato e i parametri risultano invariati rispetto quanto riportato a inizio sezione (4.1). La serie temporale per tutte le variabili è stata divisa in 2 set:

- Training: primi 270 secondi.
- Predizione (test): ultimi 30 secondi.

È stato addestrato un modello tramite regressione Ridge che mappa gli stati del reservoir ai dati reali, l'iperparametro β (fare riferimento a sezione (2.3)) è stato impostato a 10^{-7} come in [1]. In seguito ne sono state valutate le performance con l'analisi dei residui e con R^2 (fare riferimento alla sezione (2.6)). In fine sono state calcolate le matrici di correlazione funzionale (FC) per studiare la connettività cerebrale (si faccia riferimento alla sezione (2.7)). I risultati dell'analisi sono riassunti nelle Fig.(14) e (15).

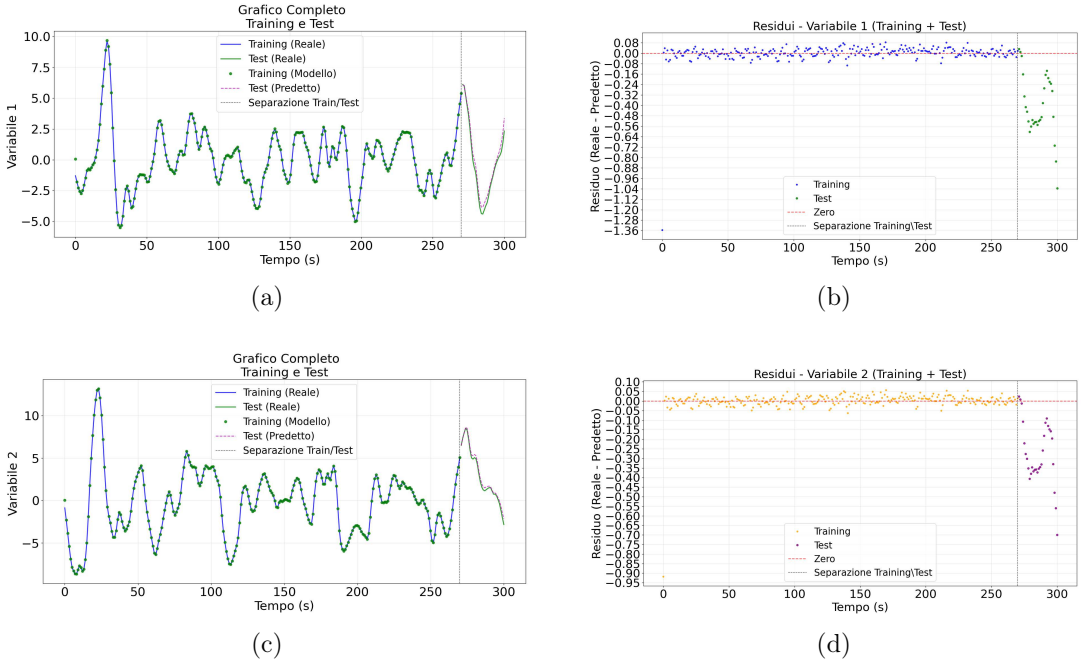


Figure 14: (a) Grafico sovrapposizione variabile 1 training+test a dati reali ; (b) Grafico residui variabile 1 training+test; (c) Grafico sovrapposizione variabile 2 training+test a dati reali; (d) Grafico residui variabile 1 training+test.

La Fig. (14) si riferisce solamente alle prime due variabili, ma rappresenta bene l'intero set di dati. Come si nota dalle Fig. (14a), Fig. (14c) il modello si è adattato molto bene ai dati reali. In Fig. (14b), Fig. (14d) sono mostrati i residui. Per quanto riguarda il set di training sembrano distribuiti in modo casuale intorno allo 0 (ad eccezione del residuo a $t = 0$, che è risente inevitabilmente della condizione iniziale del reservoir, posta nell'origine). La media generale dei residui del training risulta $3.9 \cdot 10^{-12}$. Il test R^2 dà come risultato 0.999, ossia il modello si adatta molto bene ai dati in ingresso.
Esaminando i dati di test, i residui in Fig. (14b), Fig. (14d) sono più irregolari rispetto a quelli riferiti al training, in particolare la maggior parte risulta negativa, quindi i dati predetti risultano maggiori rispetto a quelli di ingresso. In generale la loro media risulta -0.24, di molto maggiore rispetto a quella ottenuta in fase di training. Il test R^2 da come risultato 0.938, ossia il modello si adatta ancora bene ai dati in ingresso.
Paragonando il modello e le predizioni risulta che il fit degli ultimi 30 dati è peggiore rispetto a quello dei primi 270, come emege sia dai residui sia del valore di R^2. Nonostante questo, il valore di R^2 rimane vicino a 1, quindi il modello risulta valido anche in fase di predizione. Il valore di R^2 conferma che il modello non è stato overfittato. In caso contrario, il valore R^2 delle predizioni si sarebbe discostato più marcatamente da quello ottenuot in fase di training (differenza maggiore di 0.1).

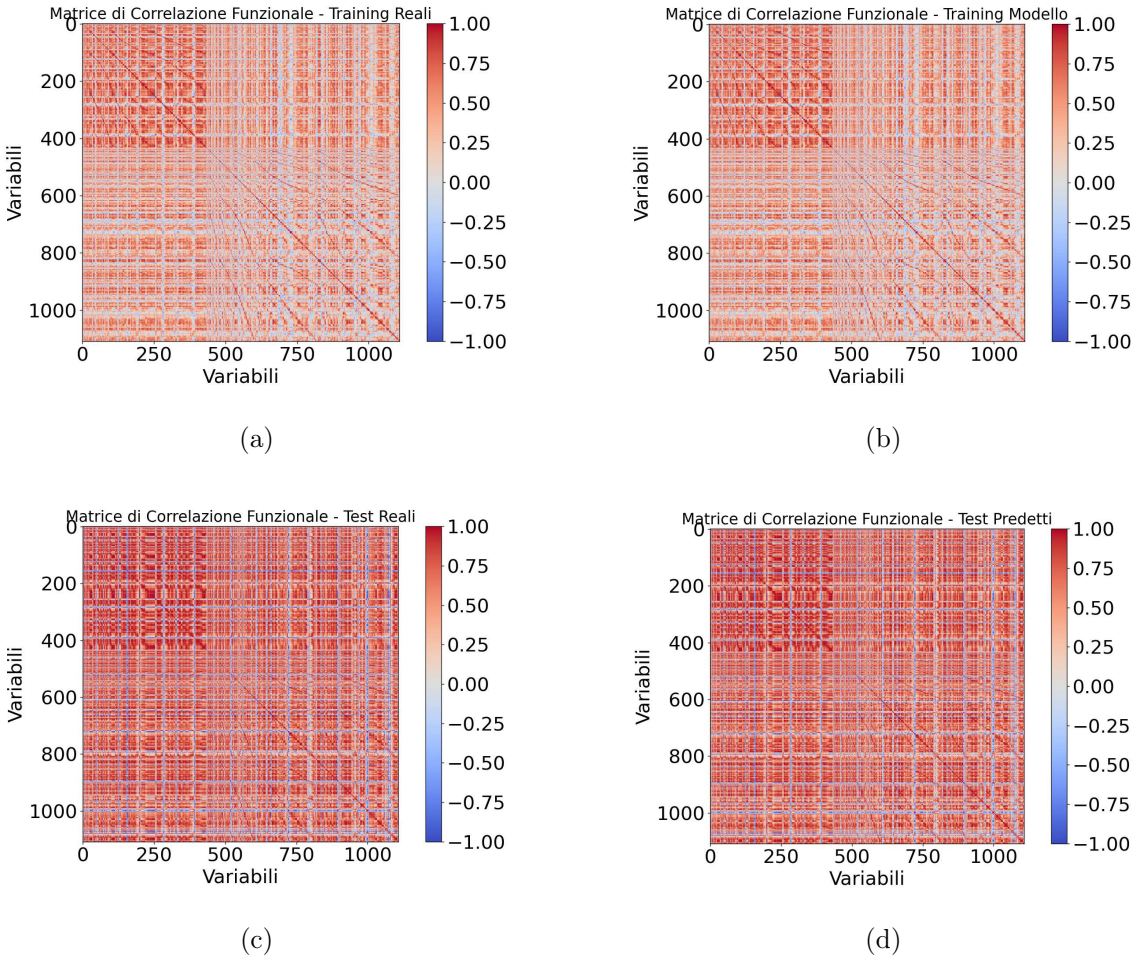


Figure 15: (a) Matrice di correlazione funzionale dati reali di training; (b) Matrice di correlazione funzionale dati modello; (c) Matrice di correlazione funzionale dati reali di test; (d) Matrice di correlazione funzionale dati predetti.

In Fig. (15) sono presenti quattro matrici di correlazione: la colonna di sinistra (Fig.(15a), Fig. (15c))rappresenta le matrici dei dati in ingresso, mentre la colonna di sinistra (Fig. (15b), Fig. (15d)) le matrici riferite ai dati in uscita. La riga superiore rappresenta il set di training e quella inferiore quello di test.
Le matrici di destra sono molto simili alle corrispettive matrici di sinistra, quondi la correlazione lineare tra tutte le coppie di variabili è ben preservata dal modello. In altre parole, il modello fornito dalla rete neurale non soltato predice accuratamente i segnali delle singole aree cerebrali, ma anche le relazioni funzionali tra di esse.

References

[1] Gabriele Di Antonio, Tommaso Gili, Andrea Gabrielli, Maurizio Mattia. *Linearizing and forecasting: a reservoir computing route to digital twins of the brain*, 22 Ottobre 2024.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning.*

[3] Bernard Mehlig. *Machine learning with neural networks.*

[4] Michael Nielsen. *Neural Networks and Deep Learning.*

[5] L. Kocarev, U. Parlitz. *Generalized Synchronization, Predictability, and Equivalence of Unidirectionally Coupled Dynamical Systems*, 11 Marzo 1996.

[6] Herbert Jaeger. *The "echo state" approach to analysing and training recurrent neural networks*, 26 Gennaio 2010.

[7] T. Hastie, R. Tibshirani, J. Friedman *The Elements of Statistical Learning.*

[8] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*, cap 2.

[9] Curtis F. Gerald, Patrick O. Wheatley. *Applied Numerical Analysis*, cap 6.3.

[10] Floris Takens. *Detecting strange attractors in turbulence.*

[11] D. Kim, T. Livne, N.V. Metcalf, M. Corbetta, & G. Shulman, *Spontaneously emerging patterns in human visual cortex and their functional connectivity are linked to the patterns evoked by visual stimuli.* Journal of neurophysiology, 124(5), 1343-1363 (2022).