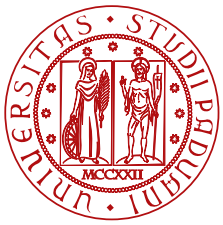

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

Identificazione sparsa di sistemi dinamici non lineari

Relatore
Prof. Pillonetto Gianluigi

Laureando
Fuso Alex

ANNO ACCADEMICO 2023-2024

Data di laurea 11/07/2024

Abstract

In questa tesi di laurea viene studiato l'algoritmo SINDy (Sparse Identification of Nonlinear Dynamics) per la stima di sistemi dinamici non lineari. Questo approccio è particolarmente utile per sistemi con un numero ristretto di termini importanti nella loro equazione di stato, che si traduce in una rappresentazione sparsa della dinamica. La stima di sistemi dinamici non lineari è un problema di grande rilevanza in diversi campi scientifici e ingegneristici. Molti sistemi fisici, come il sistema di Lorenz, sono intrinsecamente non lineari e la loro dinamica è governata da equazioni differenziali non lineari. La stima precisa di tali sistemi è fondamentale per la loro simulazione, controllo e previsione. Oltre all'implementazione dell'algoritmo SINDy su diversi sistemi modello, la tesi sviluppa una variante dell'algoritmo SINDy senza stima del parametro di sparsità. I risultati ottenuti dimostrano l'efficacia dell'algoritmo SINDy per la stima di sistemi dinamici non lineari. La variante dell'algoritmo SINDy senza stima del parametro di sparsità si dimostra un metodo promettente per la stima di sistemi con un numero elevato di variabili.

Indice

1 Introduzione **1**

2 Stima parametrica **3**

2.1 Modelli e stima 3

2.2 Stimatori ai minimi quadrati 4

2.3 Proprietà degli stimatori 5

2.4 Stimatori unbiased 6

2.4.1 Selezione del modello 7

2.4.2 Stima complessità del modello tramite cross validation (CV) 8

2.4.3 Stima complessità del modello tramite prediction sum of squares (PRESS) 8

2.4.4 Stima complessità del modello tramite generalized cross validation (GCV) 9

2.4.5 Stima complessità del modello tramite SURE 9

3 Identificazione sparsa di sistemi dinamici non lineari **11**

3.1 Identificazione sparsa 11

3.1.1 Algoritmo SINDy 13

3.1.2 Stima complessità del modello tramite CV 13

3.1.3 Nuova versione di SINDy senza scelta di λ 14

3.1.4 Scelta parametro di sparsità tramite GCV 16

3.1.5 Versione di SINDy senza scelta di λ utilizzando GCV 16

3.1.6 Scelta del parametro di sparsità tramite SURE 17

3.1.7 Versione di SINDy senza scelta di λ utilizzando SURE 17

4 Esempi numerici **19**

4.1 Attrattore di Lorenz 19

4.1.1 Stima parametri con CV 27

4.1.2 Stima parametri senza λ con CV 27

4.1.3 Stima parametri con GCV 29

4.1.4 Stima parametri senza λ con GCV 32

4.1.5 Stima parametri con SURE 34
4.1.6 Stima parametri senza λ con SURE 36
4.2 Risposta impulsiva su FIR 38
4.2.1 Stima parametri con CV 40
4.2.2 Stima parametri senza λ con CV 46
4.2.3 Stima parametri con GCV 46
4.2.4 Stima parametri senza λ con GCV 48
4.2.5 Stima parametri con SURE 52
4.2.6 Stima parametri senza λ con SURE 52

5 Conclusione **57**

A Appendice **59**
A.0.1 Funzione sparse_representation 59
A.0.2 Simulazione con attrattore di Lorenz 60
A.0.3 Funzione per generare dati del sistema di Lorenz 63
A.0.4 Funzione per generare i mononiali 64
A.0.5 Funzione che genera la sequenza dei monomi successivi in grevlex . . 65
A.0.6 Funzione per calcolo del prossimo monomio dell'ordine grevlex 69
A.0.7 Funzione SINDy con CV 72
A.0.8 Funzione SINDy con GCV 72
A.0.9 Funzione SINDy con SURE 73
A.0.10 Funzione sparse_representation senza λ con CV 73
A.0.11 Funzione sparse_representation senza λ con GCV 74
A.0.12 Funzione sparse_representation senza λ con SURE 76
A.0.13 Simulazione con risposta impulsiva 77

Elenco delle figure

4.1 Schema dell'algoritmo SINDy, applicato al sistema di Lorenz 20
4.2 Simulazione del sistema di Lorenz 23
4.3 Simulazione del sistema di Lorenz con pochi campioni a disposizione 30
4.4 Confronto tra il rumore generato e l'uscita filtrata dal filtro passa-basso nel caso di 500 campioni e 50 incognite 40
4.5 Confronto tra il rumore generato e l'uscita filtrata dal filtro passa-basso nel caso di 500 campioni e 100 incognite 41
4.6 Confronto tra risposta impulsiva reale e stima LS nel caso di 500 campioni e 50 incognite 42
4.7 Confronto tra risposta impulsiva reale e stima LS nel caso di 500 campioni e 100 incognite 43
4.8 Confronto tra risposta impulsiva reale e stima CV nel caso di 500 campioni e 50 incognite 44
4.9 Confronto tra risposta impulsiva reale e stima CV nel caso di 500 campioni e 100 incognite 45
4.10 Confronto tra risposta impulsiva reale e stima CV senza λ nel caso di 500 campioni e 50 incognite 46
4.11 Confronto tra risposta impulsiva reale e stima CV senza λ nel caso di 500 campioni e 100 incognite 47
4.12 Confronto tra risposta impulsiva reale e stima GCV nel caso di 500 campioni e 50 incognite 48
4.13 Confronto tra risposta impulsiva reale e stima GCV nel caso di 500 campioni e 100 incognite 49
4.14 Confronto tra risposta impulsiva reale e stima con GCV senza λ nel caso di 500 campioni e 50 incognite 50
4.15 Confronto tra risposta impulsiva reale e stima con GCV senza λ nel caso di 500 campioni e 100 incognite 51

4.16 Confronto tra risposta impulsiva reale e stima con SURE nel caso di 500 campioni e 50 incognite 52
4.17 Confronto tra risposta impulsiva reale e stima con SURE nel caso di 500 campioni e 100 incognite 53
4.18 Confronto tra risposta impulsiva reale e stima con SURE senza λ nel caso di 500 campioni e 50 incognite 54
4.19 Confronto tra risposta impulsiva reale e stima con SURE senza λ nel caso di 500 campioni e 100 incognite 55

Capitolo 1

Introduzione

La stima di sistemi dinamici non lineari è un problema di grande rilevanza in diversi campi scientifici e ingegneristici. Molti sistemi fisici, come il sistema di Lorenz, sono intrinsecamente non lineari e la loro dinamica è governata da equazioni differenziali non lineari. La stima precisa di tali sistemi è fondamentale per la loro simulazione, controllo e previsione.
In questa tesi di laurea sarà studiato l'algoritmo SINDy (Sparse Identification of Nonlinear Dynamics) per la stima di sistemi dinamici non lineari. Questo approccio è particolarmente utile per sistemi con un numero ristretto di termini importanti nella loro equazione di stato, che si traduce in una rappresentazione sparsa della dinamica. Nel capitolo 2 verranno spiegati i concetti di stima e modelli: stima parametrica; da dati osservati, stimatori ai minimi quadrati, stimatori unbiased non affetti da errori sistematici e tecniche di stima della dimensionalità del modello come cross-validazione e SURE, tecniche di stima della dimensionalità del modello: tecniche per determinare il numero di variabili necessarie per una buona rappresentazione del sistema. La scelta di queste tecniche sarà motivata e discussa in dettaglio. Nel capitolo 3 verrà illustrata la stima sparsa di sistemi dinamici non lineari tramite l'algoritmo SINDy e la sua variante senza stima del parametro di sparsità. L'algoritmo SINDy è un metodo recente e promettente per la stima di sistemi dinamici non lineari. La sua capacità di ottenere rappresentazioni sparse della dinamica lo rende particolarmente interessante per sistemi con un elevato numero di variabili. Sarà inoltre valutata l'efficacia dell'algoritmo SINDy per la stima di sistemi dinamici non lineari e come contributo verrà discussa una variante dell'algoritmo SINDy senza stima del parametro di sparsità. Verranno confrontate le prestazioni delle due varianti dell'algoritmo SINDy su diversi sistemi modello. Infine nel capitolo 4 verranno trattati esempi numerici sul sistema di Lorenz e sulla risposta impulsiva su sistema FIR. L'algoritmo SINDy si dimostrerà un metodo efficace per la stima di sistemi dinamici non lineari e verrà presentato un approccio alternativo evitando la selezione di questo parametro. La variante dell'algoritmo SINDy senza stima del parametro di sparsità si dimostrerà un metodo promettente per la stima di sistemi con un elevato

numero di variabili. In Appendice A tutto il codice matlab utilizzato, diviso per function. Viene seguito l'ordine di esposizione dei contenuti delle tesi.

Capitolo 2

Stima parametrica

Questo capitolo introduce gli aspetti teorici fondamentali per la risoluzione del problema della stima parametrica. Verrà ilLustrato il concetto di stimatore, delle sue proprietà e della selezione del modello con varie tecniche.

2.1 Modelli e stima

In ingegneria dell'automazione, i modelli sono strumenti fondamentali per la progettazione, la realizzazione e la regolazione di sistemi dinamici [1] [2] [3] [4] [5]. Un modello di un sistema dinamico è una descrizione matematica della sua evoluzione nel tempo. I modelli possono essere utilizzati per comprendere il comportamento del sistema, per prevedere le sue risposte a varie condizioni di ingresso, e per sviluppare algoritmi di controllo che consentano al sistema di raggiungere un obiettivo desiderato. I modelli di sistemi dinamici possono essere classificati in due categorie principali:

- Modelli deterministici: sono modelli che descrivono il comportamento del sistema in modo deterministico, ovvero senza considerare la presenza di rumore o disturbi.

- Modelli probabilistici: sono modelli che descrivono il comportamento del sistema in modo probabilistico, ovvero tenendo conto della presenza di rumore o disturbi.

I modelli deterministici possono essere ulteriormente classificati in due categorie principali:

- Modelli lineari, sono modelli in cui le relazioni tra le variabili del sistema sono descritte da equazioni lineari.

- Modelli non lineari, sono modelli in cui le relazioni tra le variabili del sistema sono descritte da equazioni non lineari.

Il problema della stima è quello di determinare il valore attuale o futuro di una variabile del sistema, sulla base di misurazioni effettuate in passato. La stima è un'attività fondamentale in molti campi dell'ingegneria, tra cui l'automazione, la robotica, la navigazione e la medicina. In particolare, la stima dello stato è il problema di determinare il valore attuale del vettore di stato di un sistema dinamico, sulla base di misurazioni delle sue uscite. La stima dello stato è un problema fondamentale in ingegneria dell'automazione, in quanto consente di implementare algoritmi di controllo che tengano conto delle condizioni attuali del sistema [6].

2.2 Stimatori ai minimi quadrati

L'approccio classico alla stima di sistemi dinamici è quello di utilizzare un modello del sistema per generare delle previsioni sulle sue uscite [7]. Queste previsioni vengono poi utilizzate per stimare il valore attuale o futuro delle variabili del sistema.
Dato y un vettore aleatorio di dimensione n, in generale il problema della stima parametrica è formalizzato , assegnando una famiglia di densità di probabilità candidate per y, $p(y|\theta)$ (o $p_\theta(y)$, per ogni evento $A \subseteq \mathbb{R}^n$ si ha

$$\mathcal{P}(y \in A|\theta) = \int_A p_\theta(y)\, dy \tag{2.1}$$

con quindi un risultato in funzione di θ.
L'obiettivo è quello di usare l'informazione contenuta nelle realizzazione di y per determinare il valore di θ.
Viene definita come $p(y|\theta)$ la densità congiunta di y e statistica una funzione misurabile che mappa $\mathbb{R}^n$ (spazio delle misure) in $\mathbb{R}^m$ (spazio che contiene Θ) che non dipende da Θ. La principale assunzione da fare è che la classe di densità $p(y|\theta)$ contiene la stima vera definita con θ_0; così è quindi possibile definire uno stimatore $\hat{\theta}(y)$ come una qualunque statistica di argomento y e con valori in Θ dove il valore assunto dallo stimatore in corrispondenza delle realizzazione di y fornisce la stima di θ_0.
Supponendo di aver fissato la seguente struttura

$$y_i = g_i(\theta) + e_i \quad i = 1, \ldots, n \tag{2.2}$$

un approccio per creare uno stimatore consiste nel determinare quel θ che descrive i dati osservati y; un primo esempio è quello di utilizzare uno stimatore ai minimi quadrati (least squares, LS) che per ogni osservazione y mappa θ^{LS}, definendo così lo stimatore ai minimi quadrati del parametro θ basato sulle misure y, $\theta^{LS}(y)$

$$\theta^{LS}(y) = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - g_i(\theta))^2 \tag{2.3}$$

si noti che la formulazione non utilizza la densità $p(y|\theta)$ [8]. In generale le g_i potrebbero essere funzioni che rendono difficile il calcolo di $\theta^{LS}(y)$ e dato che la stima potrebbe non essere disponibile in forma chiusa, si rende necessario ricorrere a tecniche iterative di ottimizzazione. Per esempio scegliendo g_i come funzioni lineari in θ, l'equazione 2.2 diventa

$$y = \Phi\theta + E \tag{2.4}$$

che porta ad avere una struttura lineare in θ dove la stima ai minimi quadrati assume la seguente forma

$$\theta^{LS}(y) = \underset{\theta \in \Theta}{\operatorname{argmin}} |||y - \Phi\theta||^2 \tag{2.5}$$

che viene detto stimatore ai minimi quadrati lineari. E' possibile dimostrare che le equzioni ammettono sempre almeno una soluzione unica e la condizione necessaria e sufficiente che questo accada è che le colonne di Φ siano linearmente indipendenti quindi la soluzione diventa

$$\theta^{LS}(y) = (\Phi^T\Phi)^{-1}\Phi^T y \tag{2.6}$$

2.3 Proprietà degli stimatori

E' necessario definire un metodo con cui confrontare diversi stimatori e quindi introdurre concetti di distanza tra il vettore aleatorio $\hat{\theta}$ e quello deterministico θ [9]. Per esempio con l'utilizzo della norma euclidea è possibile usare l'errore quadratico medio, mean squared error (MSE) di uno stimatore $\hat{\theta}$ di un paramentro θ

$$MSE_{\hat{\theta}}(\theta) = \mathcal{E}_\theta(||\hat{\theta} - \theta||^2) \tag{2.7}$$

in questo modo è possibile scomporre l'errore di ricostruzione in bias e varianza

$$MSE_{\hat{\theta}} = Tr\left[\mathcal{V}_\theta(\hat{\theta})\right] + \sum_{i=1}^{n} (Bias_{\theta_i}(\hat{\theta}_i))^2 \tag{2.8}$$

dove la parte di bias della componente i-esima è definito come

$$Bias_{\hat{\theta}_i} = \mathcal{E}_\theta \hat{\theta}_i - \theta_i \tag{2.9}$$

Lo stimatore $\hat{\theta}_1$ è detto ammissibile se non è possibile trovare un altro stimatore $\hat{\theta}_2$ e un insieme $A \subseteq \Theta$ con $A \neq \emptyset$ con

$$\begin{cases} MSE_{\hat{\theta}_2} \leq MSE_{\hat{\theta}_1} & \theta \in \Theta \setminus A \\ MSE_{\hat{\theta}_2} < MSE_{\hat{\theta}_1} & \theta \in A \end{cases} \tag{2.10}$$

negli altri casi $\hat{\theta}_1$ è detto inammisibile.
In generale uno stimatore dovrebbe sempre essere scartato in quanto esiste sempre uno stimatore migliore, con un errore di ricostruzione più basso.

2.4 Stimatori unbiased

Per ricavare lo stimatore ottimo si va a restringere la ricerca alle classi di stimatore definiti unbiased, ovvero imponendo che la media dello stimatore $\hat{\theta}$ sia pari al valore vero θ_0 [10] [11] . In particolare lo stimatore $\hat{\theta}$ è detto unbiased se

$$\mathcal{E}_\theta \hat{\theta} = \theta \qquad\qquad \forall \theta \in \Theta$$

dove quindi il bias è nullo e dove di conseguenza l'errore di ricostruzione

$$MSE_{\hat{\theta}} = Tr\Big[\mathcal{V}_\theta(\hat{\theta})\Big] \qquad\qquad \hat{\theta} \text{ unbiased}$$

è solo il termine di varianza. Uno stimatore è detto unbiased a minima varianza di errore (UMVE) se per ogni altro stimatore unbiased $\bar{\theta}$

$$Tr\Big[\mathcal{V}_\theta(\hat{\theta})\Big] \leq Tr\Big[\mathcal{V}_\theta(\bar{\theta})\Big] \qquad\qquad \forall \theta \in \Theta$$

Definendo di seguito $\hat{\theta}_n$ come uno stimatore in funzione del numero delle misure si può dire che è asintoticamente unbiased se:

$$\lim_{n\to\infty} \mathcal{E}_\theta \hat{\theta}_n = \theta \qquad\qquad \forall \theta \in \Theta$$

dove quindi è desiderata la convergenza dello stimatore ad un valore vero θ_0 che è incognito e per cui si richiede che $\hat{\theta}_n$ si concentri attorno a θ per ogni $\theta \in \Theta$.

Questa proprietà è detta consistenza e in generale uno stimatore è consistente se si ha che:

$$\lim_{n\to\infty} \mathcal{P}(|\hat{\theta}_n - \theta| < \varepsilon) = 1 \qquad \forall\theta \in \Theta \qquad \forall\varepsilon > 0$$

e quindi definita la consistenza ed è possibile anche indicare una condizione sufficente per la consistenza, dove se uno stimatore $\hat{\theta}_n$ è asintoticamente unbiased e se $\lim_{n\to\infty} \mathcal{V}_\theta(\hat{\theta}_n) = 0$ allora lo stimatore è consistente.
Lo stimantore è definito come asintoticamnete efficiente se

$$\sqrt{n}(\hat{\theta}_n - \theta) \to \mathcal{N}(0, I(\theta)^{-1}) \text{ in distribuzione con } n \to \infty \text{ e } \forall\theta \tag{2.11}$$

con $I(\theta)$ matrice di Fisher.
Risulta quindi che lo stimatore a massima verosomiglianza è consistente e asintoticamente efficiente [12] .

2.4.1 Selezione del modello

Si intende di seguito affrontare il tema della selezione del modello usando il principio della parismonia dove considerando due modelli lineari che condividono la stessa matrice di regressione Φ e lo stesso θ incognito

$$y = \Phi\theta + e \tag{2.12}$$

e

$$z = \Phi\theta + v \tag{2.13}$$

con e e v vettori della stessa dimensione n con media nulla e varianza σ^2 [13].
Dato uno stimatore ai minimi quadrati $\theta\hat{(y)}$ di θ si vuole calcolare

$$\mathcal{E}(||z - \hat{y}||^2) \tag{2.14}$$

dove $\hat{y} = \Phi\hat{\theta}$ e quindi

$$\begin{aligned} \hat{y} &= \Phi(\Phi^T\Phi)^{-1}\Phi^T y \\ &= \Phi(\Phi^T\Phi)^{-1}\Phi^T(\Phi\theta + e) \\ &= \Phi\theta + He \end{aligned} \tag{2.15}$$

dove H è definita come $H = \Phi(\Phi^T\Phi)^{-1}\Phi^T$ e quindi

$$\begin{aligned}\mathcal{E}(||z-\hat{y}||^2) &= \mathcal{E}||\Phi\theta + v - \Phi\theta - He||^2 \\ &= \mathcal{E}||v - He||^2 \\ &= \mathcal{E}||v||^2 + \mathcal{E}||He||^2 \\ &= n\sigma^2 + \sigma^2 Tr(HH^T) \\ &= n\sigma^2 + \sigma^2 Tr(H) \\ &= (n + dim(\theta))\sigma^2\end{aligned} \tag{2.16}$$

2.4.2 Stima complessità del modello tramite cross validation (CV)

Di seguito si presenta un metodo per la selezione dell'ordine di modello, la cross validazione (CV) che è uno dei metodi più utilizzati per la selezione di un modello [14] [15] [16].
Ci sono varie versioni di questo metodo e di seguito viene trattata la hold-out che consiste nel dividere i dati in due sottoinsiemi disgiunti chiamati training set e validation set. Nel training set ($\mathcal{T}$) sono presenti $n-k$ elementi e verrà usato per stimare i modelli mentre il validation set ($\mathcal{V}$) che contiene i k elementi per testare i modelli e scegliere il migliore in base a quello che minimizza l'errore.
Per ogni struttura del modello $\mathcal{M}(\theta)$ ipotizzata, $\mathcal{T}$ serve a determinare θ con per esempio il criterio della massima verosomiglianza.
Una volta ottenuto il modello ipotizzato $\mathcal{M}(\hat{\theta})$ si ottengono predizioni sulle uscite in $\mathcal{T}$ che alimentano $\mathcal{M}(\hat{\theta})$ così da ottenere le predizioni $\hat{y}$ e generare un indice di capacità di generalizzazione

$$CV = ||y - \hat{y}||^2 \tag{2.17}$$

2.4.3 Stima complessità del modello tramite prediction sum of squares (PRESS)

Un'alternativa è PRESS (prediction sum of squares) che consiste nel fissare $k = 1$, rendendo il set $\mathcal{V}$ composto da un elemento per poi calcolare la capacità di predizione del modello utilizzando le possibili n partizioni [16] [17].
Nel caso di regressione lineare è possibile quindi definire i residui

$$r = y - \hat{y} = y - \Phi(\Phi^T\Phi)^{-1}\Phi^T y \tag{2.18}$$

e di conseguenza si ha

$$PRESS = \sum_{i=1}^{n} \frac{r_i^2}{(1-h_i)^2} \tag{2.19}$$

dove gli h_i sono gli elementi sulla diagonale di

$$H = \Phi(\Phi^T\Phi)^{-1}\Phi^T \tag{2.20}$$

2.4.4 Stima complessità del modello tramite generalized cross validation (GCV)

Un'approssimazione di PRESS è il metodo generalized-cross-validation (GCV) [18].
Per poterlo definire è necessario riprendere il vettore dei residui 2.18 e successivamente si ha

$$GCV = \sum_{i=1}^{n} \frac{r_i^2}{(1-h_i)^2} \tag{2.21}$$

dove gli h_i sono definiti come

$$h_i \approx \frac{Tr(H)}{n} = \frac{Tr(\Phi(\Phi^T\Phi)^{-1}\Phi^T)}{n} = \frac{dim(\theta)}{n} \tag{2.22}$$

che porta quindi l'indice a essere

$$GCV = \frac{||r||^2}{(1-\frac{dim(\theta)}{n})^2} \tag{2.23}$$

2.4.5 Stima complessità del modello tramite SURE

La seguente tecnica per la stima non necessita di dividere y e l'unica ipotesi per la generazione dei dati e che il sistema sia del seguente tipo

$$y_i = \mu_i + e_i \qquad\qquad i = 1, \ldots, n$$

dove gli errori sono a media nulla e varianza σ^2.
Fissata la struttura $\mathcal{M}(\theta)$ e una regola per determinare y e θ si possono calcolare gli stimatori $\hat{y}_i$ e $\hat{\mu}_i$. Se quindi si utilizza una struttura del tipo $y = \Phi\theta + e$ il predittore ai minimi quadrati risulta $\hat{y} = \Phi(\Phi^T\Phi)^{-1}\Phi^T y$.

La stima dell'errore di predizione SURE (Stein's unbiased risk estimation) è definita come

$$SURE = ||y - \hat{y}||^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{y}_i}{i} \tag{2.24}$$
$$= ||y - \hat{y}||^2 + 2\sigma^2 dim(\theta)$$

dove l'ultima equazione è valida nel caso di regressione lineare risolta tramite i minimi quadrati [19].
La soluzione proposta ha validità anche per rumori non Gaussiani e per quanto riguarda la varianza, intesa come incognita, è possibile utilizzare un suo stimatore unbiased come per esempio il seguente

$$\hat{\sigma}^2 = \frac{||y - \Phi\hat{\theta}^{LS}||^2}{n - m} \tag{2.25}$$

che in caso di rumore Guassiano è proporzionale a una χ^2 con $n - m$ gradi di libertà

Capitolo 3

Identificazione sparsa di sistemi dinamici non lineari

3.1 Identificazione sparsa

In questo lavoro di tesi viene implementanto SINDy [20] [21]–[24]. Nel lavoro viene sfruttato il fatto che la maggior parte dei sistemi fisici hanno solo pochi termini rilevanti che definiscono la dinamica, rendendo sparse le equazioni che la governano. Qui vengono considerati i sistemi dinamici della forma

$$\dot{x}(t) = f(x(t)) \tag{3.1}$$

dove il vettore $x(t) \in \mathbb{R}^n$ è lo stato di un sistema al tempo t, e la funzione $f(x(t))$ rappresenta i vincoli che definiscono le equazioni della dinamica del sistema. L'osservazione principale è che per molti sistemi di interesse, la funzione f è costituita da pochi termini rilevanti, il che la rende sparsa.
L'identificazione del modello sparso promuove modelli non complessi in grado però di descrivere i dati.
Per determinare la funzione f dai dati, raccogliamo dati dello stato $x(t)$ (o tramite delle simulazioni) e ne viene calcolata la derivata $\dot{x}(t)$ o approssimata stimata da $x(t)$. I dati sono campionati ai tempi $t_1, t_2, \ldots, t_m$ e disposti in due matrici:

$$X = \begin{bmatrix} x^T(t_1) \\ x^T(t_2) \\ \vdots \\ x^T(t_m) \end{bmatrix} = \begin{bmatrix} x_1^T(t_1) & x_2^T(t_1) & \dots & x_n^T(t_1) \\ x_1^T(t_2) & x_2^T(t_2) & \dots & x_n^T(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1^T(t_m) & x_2^T(t_m) & \dots & x_n^T(t_m) \end{bmatrix} \tag{3.2}$$

$$\dot{X} = \begin{bmatrix} \dot{x}^T(t_1) \\ \dot{x}^T(t_2) \\ \vdots \\ \dot{x}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x}_1^T(t_1) & \dot{x}_2^T(t_1) & \dots & \dot{x}_n^T(t_1) \\ \dot{x}_1^T(t_2) & \dot{x}_2^T(t_2) & \dots & \dot{x}_n^T(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1^T(t_m) & \dot{x}_2^T(t_m) & \dots & \dot{x}_n^T(t_m) \end{bmatrix} \tag{3.3}$$

Successivamente, viene costruita $\Theta(X)$ che rappresenta candidati di funzioni non lineari delle colonne di X. $\Theta(X)$ sarà formata dal termine costante e da tutti i monomiali di grando r:

$$\Theta(X) = \begin{bmatrix} | & | & | & | \\ 1 & X & X^{P_2} & \dots \\ | & | & | & | \end{bmatrix} \tag{3.4}$$

Ogni colonna di $\Theta(X)$ rappresenta una funzione candidata per il lato destro dell'Equazione 3.1. C'è un'enorme libertà nella scelta in questa matrice di non linearità e si può come primo tentativo impostare un problema di regressione per determinare i vettori sparsi di coefficienti $\Xi = \begin{bmatrix} \xi_1 & \xi_2 & \dots & \xi_n \end{bmatrix}$ che determinano quali non linearità sono attive:

$$\dot{X} = \Theta(X)\Xi \tag{3.5}$$

Ogni colonna ξ_k di Ξ è un vettore sparso di coefficienti che determinano quali termini sono attivi sul lato destro per una delle equazioni di riga $\dot{x}_K = f_k(x)$ nell'Eq. 3.1. Una volta che Ξ è stato determinato, un modello di ciascuna riga delle equazioni che governano può essere costruito come segue:

$$\dot{x}_K = f_k(x) = \Theta(x^T)\xi_k \tag{3.6}$$

dove $\Theta(x^T)$ è il vettore delle funzioni degli elementi di x, non $\Theta(X)$ che è la matrice dei dati. Possiamo risolvere per Ξ nell'Eq. 3.1 usando la regressione sparsa.
Realisticamente, spesso è disponibile solo X e $\dot{X}$ deve essere approssimato numericamente; quindi in X e $\dot{X}$ è presente del rumore, quindi l'Eq. 3.1 diventa

$$Y = \Theta(X)\Xi + \eta Z \tag{3.7}$$

dove Z è modellato come una matrice di elementi Gaussiani indipendenti distribuiti con media zero e varianza η. Il seguente algoritmo 4.1 si basa su LASSO [25], una regressione regolarizzata-l_1 che promuove la sparsità; ne vengono migliorate le prestazioni in caso di grandi dataset.

3.1.1 Algoritmo SINDy

Vi sono molti algoritmi per determinare le soluzioni sparse Ξ al problema di regressione in Eq. 3.1. Ogni colonna dell'Eq. 3.1 richiede un problema di ottimizzazione distinto per trovare il vettore sparso dei coefficienti ξ_k per il k^i elemento riga dell'equazione.
Per questo lavoro di tesi, la matrice $\Theta(X)$ ha dimensioni $m \times p$, dove p è il numero di funzioni non lineari candidate, e dove $m \gg p$ poiché ci sono più campioni di dati rispetto alle funzioni non lineari candidate. Spesso sono disponibili solo i dati X mentre $\dot{X}$ deve essere approssimato numericamente, come negli esempi seguenti. Pertanto, sia X che $\dot{X}$ avranno del rumore in modo che l'Eq. 3.1 non valga esattamente avendo quindi Y dati rumorosi 3.7.
E' stato implementato l'algoritmo A.0.1 che inizia con una soluzione dei minimi quadrati per Ξ per poi venire scartati tutti i coefficienti che sono inferiori a un certo valore di λ. Una volta che vengono identificati tutti gli indici dei restanti coefficienti diversi da zero, si ottiene un'altra soluzione dei minimi quadrati per Ξ sugli indici rimanenti.
Questi nuovi coefficienti sono nuovamente valutati attraverso il parametro λ e la procedura continua fino a quando i coefficenti diversi da zero convergono.
Questo algoritmo è efficiente dal punto di vista computazionale e converge rapidamente in un soluzione sparsa in un numero ridotto di iterazioni [26]. L'algoritmo beneficia anche della semplicità, con un singolo parametro λ necessario per determinare il grado di sparsità in Ξ e si dimostra robusto al rumore, anche quando le derivate devono essere approssimate da dati rumorosi.

3.1.2 Stima complessità del modello tramite CV

Dato che il parametro λ non è noto a priori, un punto di difficoltà è la scelta di tale parametro; è stato quindi implementato l'utilizzo di una griglia di λ, un Λ contente una serie di λ candidati e ne è stato scelto quello che minimizza l'errore tramite il metodo della cross-validazione presentanto in 2.4.2.
L'idea è quella di dividere i dati in un set di training e uno di validation; l'algoritmo va a considerare per ogni valore di λ nel vettore Λ delle possibile scelte. Viene calcolata una rappresentazione sparsa del modello utilizzando i dati di training con il valore corrente di λ. Viene successivamente calcolato l'errore di convalida utilizzando i dati di validazione e la rappresentazione sparsa corrente e al termine del controllo di tutti vari λ ne viene individuato il valore di λ ottimo che minimizza l'errore di convalida. La funzione implementata $SINDY\_CV$ (A.0.7) effettua una scelta del modello tramite CV per individuare il parametro di regolarizzazione ottimo (λ^*). Riceve come input i dati di training (Θ_{train} e Y_{train}) e di validazione (Θ_{val} e Y_{val}), insieme a un vettore di possibili valori per il parametro di regolarizzazione (Λ). L'algoritmo itera su ogni valore di λ nel vettore Λ. Ad ogni iterazione, viene chiamata

la funzione $sparse\_representation$ (A.0.1) per calcolare una rappresentazione sparsa del modello utilizzando i dati di training e il valore corrente di λ. Vengono inoltre ottenuti gli indici $biginds$ e $smallinds$ che identificano i coefficienti del modello e rispettivamente quelli che vengono considerati($biginds$) e quelli esclusi($smallinds$). Successivamente, viene calcolato l'errore di convalida utilizzando i dati di validazione e la rappresentazione sparsa corrente. Al termine del ciclo, viene individuato il valore di λ (λ^*) che minimizza l'errore di convalida. Infine, la funzione richiama nuovamente $sparse\_representation$ utilizzando Θ_{train}, Y_{train} e il λ^* ottenuto per calcolare la rappresentazione sparsa finale del modello e gli indici associati ai coefficienti. Questo approccio consente di selezionare il parametro di regolarizzazione che bilancia meglio la complessità del modello e la sua capacità di generalizzare su dati non visti durante il training. Il risultato è quindi una tabella con coefficienti di regressione sparsa.

3.1.3 Nuova versione di SINDy senza scelta di λ

L'apporto del lavoro di tesi è stato quello di implementare un nuovo algoritmo senza la necessità di dover scegliere un parametro di sparsità λ in modo da migliorare la prima stima fatta dai minimi quadrati (la scelta del modello migliore è presentata con CV e successivamente in GCV e SURE). In questa sezione si è tentato un approccio che non necessita del parametro di sparsità λ così da evitare una prima fase di analisi per capire quali valori delle soluzioni dei coefficienti trovati scartare. L'idea è quella di iterare per il numero di incognite e di rimuovere la componente con valore assoluto minimo. Vengono quindi divisi i dati in training e validation e con la parte di training viene eseguito l'algoritmo e poi grazie ai dati di validation viene calcolato l'errore e così via per ogni iterazione per incognita.
La funzione implementata quindi è sparse_representation_CV_no_lambda (presentata in appendice A.0.10) e prende quattro argomenti in ingresso:

- Θ_{train}: Matrice dei monomi di training
- Y_{train}: Vettore delle approssimazioni delle derivate dello stato per il training
- Θ_{val}: Valori di Θ usati per la convalida
- Y_{val}: Vettore delle approssimazioni delle derivate dello stato per la convalida

La funzione restituisce quattro elementi:

- Ξ: Rappresentazione dei coefficienti del sistema
- $biginds$: Indici degli elementi di Ξ considerati significativi (non azzerati)

- $smallinds$: Indici degli elementi di Ξ azzerati durante il processo di sparsità

- Ξ_{LS}: primo tentativo con i minimi quadrati, che verrà confrontato con il migliore risultato

L'algoritmo implementa una regressione ai minini quadrati per ottenere una rappresentazione di Ξ.
Vengono calcolati i coefficienti di regressione minimi quadrati tramite $\frac{\Theta}{Y}$.
La funzione esegue un ciclo per iterare su ogni colonna di Ξ e all'interno del ciclo viene creata una maschera logica $mask_1$ per identificare gli elementi nulli correnti in Ξ.
Viene calcolato il valore assoluto minimo degli elementi non nulli in Ξ e salvato in $minAbsValue$. Successivamente si crea un'altra maschera logica $mask_2$ per identificare gli elementi in Ξ con valore assoluto uguale a $minAbsValue$ e la maschera $smallinds$ viene aggiornata per includere sia gli elementi nulli originali che quelli identificati da $minAbsValue$.
Gli elementi di Ξ selezionati da $smallinds$ vengono impostati a zero così da escludere queste componenti e dopo aver iterato su tutte le colonne di Ξ, la funzione esegue un altro ciclo per affinare la rappresentazione sparsa; per ogni dimensione del segnale (n), viene creata una maschera logica $biginds$ che è l'elemento negativo di $smallinds$ per la colonna corrente (questa maschera identifica gli elementi non nulli in Ξ).
Si esegue una regressione di Y sulla colonna corrente di Θ utilizzando solo gli elementi non nulli identificati da $biginds$, il risultato viene utilizzato per aggiornare la colonna corrispondente in Ξ.
Vengono successivamente identificati i coefficienti con valore assoluto piccolo o nullo in Ξ tramite una maschera di coefficienti e si individua il valore assoluto minimo diverso da zero dei coefficienti di Ξ. Vengono poi salvati gli elementi che si sono di volta in volta tolti nella matrice $smallinds$.
Gli elementi in Ξ individuati da $smallinds$ vengono azzerati per ottenere poi una stima più accurata.
Vengono calcolati gli indici degli elementi rimanenti in Ξ tramite $biginds$ e si esegue una regressione sui termini rimanenti per trovare un nuovo valore sparso di Ξ per la dimensione di stato corrente. L'errore di convalida viene calcolato e salvato in $err(k)$ e viene selezionato il miglior modello ripristinando i valori iniziali di Ξ e viene individuata l'iterazione con il minor errore di convalida. La funzione restituisce la rappresentazione finale Ξ ottenuta dall'iterazione con il minor errore di convalida, insieme agli indici degli elementi significativi ($biginds$) e azzerati ($smallinds$).

3.1.4 Scelta parametro di sparsità tramite GCV

Nel caso di matrice Θ malcondizionata, usare tutti i dati per la stimare i parametri risulta particolarmente vantaggioso.
Il seguente metodo di scelta del parametro di sparsità λ sfrutta tutti i dati e non divide in training e validation, utilizza quindi la tecnica di GCV indicata nel capitolo 2.4.4. La funzione $SINDY\_GCV$ è (definita in appendice A.0.8) simile a $SINDY\_CV$ e impiega anch'essa la convalida per individuare il parametro di regolarizzazione ottimo (λ^*) per un modello a rappresentazione sparsa. Assume come input i dati complessivi (Θ e Y) e un vettore di possibili valori per λ.
Come per $SINDY\_CV$, si itera su ogni valore di λ nel vettore Λ e ad ogni iterazione, viene chiamata $sparse\_representation$ per ottenere la rappresentazione sparsa del modello e gli indici $biginds$ e $smallinds$. In seguito, viene calcolato l'errore tra i dati target Y e la predizione del modello Ξ, normalizzato per tenere conto della dimensione dei dati e del numero di termini del modello.
Al termine del ciclo, viene individuato il valore di λ (λ^*) che minimizza l'errore di convalida GCV. Infine, la funzione richiama $sparse\_representation$ con λ^* per ottenere la rappresentazione sparsa finale del modello e gli indici associati ai coefficienti. Questo approccio mira a bilanciare la complessità del modello e la sua aderenza ai dati.

3.1.5 Versione di SINDy senza scelta di λ utilizzando GCV

Come per cross validazione è stato implementato l'algoritmo SINDy senza parametro di sparsità ma con scelta del modello tramite GCV. La funzione è $sparse\_representation\_GCV\_no\_lambda$ (definita in A.0.11) e restituisce una rappresentazione sparsa di Ξ utilizzando i monomiali Θ, i dati con rumore Y e un parametro di sparsità interno.
La funzione necessita dei seguenti ingressi:

- Θ: Matrice ($m \times d$) contenente i monomiali, dove m è il numero di monomiali e d è la dimensione dello stato del sistema
- Y: Matrice ($n \times d$) contenente i dati delle uscite rumorse dove n è il numero di campioni e d è la dimensione dello stato del sistema

Successivamente la funzione restituisce:

- Ξ: Matrice ($m \times d$) contenente la rappresentazione sparsa dei coefficienti

- $biginds$: Matrice logica ($m \times d$) che indica gli elementi di Ξ mantenuti durante la sparsità
- $smallinds$: Matrice logica ($m \times d$) che indica gli elementi di Ξ impostati a zero durante la sparsità e quindi scartati dall'analisi; sono gli elementi che di fatto non contribuiscono alla descrizione del sistema
- Ξ_{LS}: Matrice ($m \times d$) contenente la soluzione iniziale dei coefficienti calcolata mediante regressione ai minini quadrati

La funzione $sparse\_representation\_GCV\_no\_lambda$ determina Ξ che viene inizializzata con la soluzione LS, calcolando $\Xi = \frac{\Theta}{Y}$. Successivamente viene eseguito il ciclo per un numero massimo di iterazioni $m \times n$ (che sono tutte le possibili incognite, in modo tale da togliere tutte le variabili a fine ciclo). La funzione utilizza GCV per selezionare la soluzione sparsa con l'errore di ricostruzione minimo.

3.1.6 Scelta del parametro di sparsità tramite SURE

Come per il capitolo 3.1.4 di seguito si espone il metodo della scelta del modello SURE come indicato nel capitolo 2.4.5 che come detto può essere utile in cui la Θ è malcodizionata.
La funzione $SINDY\_SURE$ è un'altra tecnica di selezione del parametro di regolarizzazione basata su un criterio di predizione dell'errore di Squared Unbiased Risk Estimate (SURE). Richiede come input i dati complessivi Θ e Y, un vettore di possibili valori per λ e la varianza del rumore nei dati σ_y^2 che, come indicato nel capitolo 2.4.5, è stata stimata con uno stimatore unbiased.
Similmente alle funzioni precedenti, itera su ogni valore di λ del vettore Λ. Ad ogni iterazione, viene chiamata $sparse\_representation$ per ottenere la rappresentazione sparsa del modello e gli indici $biginds$ e $smallinds$. Successivamente, viene calcolato l'errore.
Al termine del ciclo, viene individuato il valore di λ (λ^*) che minimizza l'errore SURE. Infine, la funzione richiama $sparse\_representation$ con λ^* per ottenere la rappresentazione sparsa finale del modello e gli indici associati ai coefficienti.

3.1.7 Versione di SINDy senza scelta di λ utilizzando SURE

La funzione $sparse\_representation\_SURE\_no\_lambda$ (A.0.12) è molto simile a $sparse\_representation\_GCV\_no\_lambda$ e ha l'obiettivo di trovare una rappresentazione sparsa di Ξ utilizzando i monomiali Θ, i dati con rumore Y e un parametro di rumore σ^2 senza però ricorerre alla scelta di λ. La funzione richiede in ingresso:

- Θ: Matrice ($m \times d$) contenente i monomiali candidati, dove m è il numero di monomiali e d è la dimensione dello stato del sistema
- Y: Matrice ($n \times d$) contenente le uscite rumorose del sistema di Lorenz, dove n è il numero di campioni e d è la dimensione dello stato del sistema
- σ^2: Varianza del rumore presente nei dati Y

In uscita la funzione restituisce:

- Ξ: Matrice ($m \times d$) contenente la rappresentazione sparsa dei coefficienti
- $biginds$: Matrice logica ($m \times d$) che indica gli elementi di Ξ mantenuti durante la sparsità
- smallinds: Matrice logica ($m \times d$) che indica gli elementi di Ξ impostati a zero e quindi scartati
- Ξ_{LS}: Matrice ($m \times d$) contenente la soluzione iniziale dei coefficienti calcolata mediante stimatore ai minini quadrati

La funzione determina Ξ e viene inizializzata con la soluzione ai minimi quadrati, calcolando $\Xi_{LS} = \frac{\Theta}{Y}$. A differenza di GCV, qui il criterio di selezione utilizza SURE.
L'errore di ricostruzione viene calcolato per ogni iterazione come: $e = ||Y - \Theta\Xi||^2 + 2\sigma^2 m$
Il termine $2\sigma^2 m$ rappresenta una penalizzazione basata su SURE che tiene conto della complessità del modello (numero di termini in Θ). Viene quindi rieseguito nuovamente l'algoritmo utilizzando la soluzione con l'errore minimo come punto di partenza fermandosi all'indice con errore minimo, trovando quindi la soluzione che minimizza l'errore.

Capitolo 4

Esempi numerici

I metodi descritti nel Capitolo 3 per identificare le equazioni dai dati sono ora mostrati in due esempi di sistemi di cui sono riportati i risultati. Nel primo esempio, è stato studiato l'attrattore di Lorenz dove l'algoritmo di identificazione sparsa riproduce accuratamente la forma delle equazioni che governano la dinamica dell'attrattore. Il secondo esempio consiste nello stimare la risposta all'impulso g^0 FIR di una dinamica di un sistema a tempo discreto, lineare e causale, a partire da dati di uscita rumorosi.

4.1 Attrattore di Lorenz

I test di identificazione del sistema dinamico non lineare sono stati fatti simulando l'Attrattore di Lorenz che ha un comportamento caotico, il che significa che è altamente sensibile alle condizioni iniziali e le sue traiettorie nello spazio delle fasi sono imprevedibili a lungo termine[27]. Questa complessità sfida gli algoritmi di identificazione del modello a catturare correttamente la dinamica del sistema ed è di interesse provare a verificare se l'algoritmo SINDy sia in grado di riscostruire correttamente il sistema. Lorenz è formato da un sistema di tre equazioni differenziali del primo ordine:

$$\begin{cases} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - xz - y \\ \dot{z} &= xy - \beta z \end{cases} \tag{4.1}$$

dove σ è il numero di Prandt e ρ è il numero di Rayleigh. σ , ρ e β sono maggiori di 0, ma nella maggior parte dei casi $\sigma = 10$ e $\beta = \frac{8}{3}$, mentre ρ è variabile. Sebbene queste equazioni diano origine a dinamiche ricche e caotiche che si evolvono su un attrattore, ci sono solo pochi termini che ne identificano la dinamica rendendola sparsa. La Figura 4.1 mostra uno schema di come i dati vengono raccolti per questo esempio e quanto sono sparse le dinamiche identificate. Per

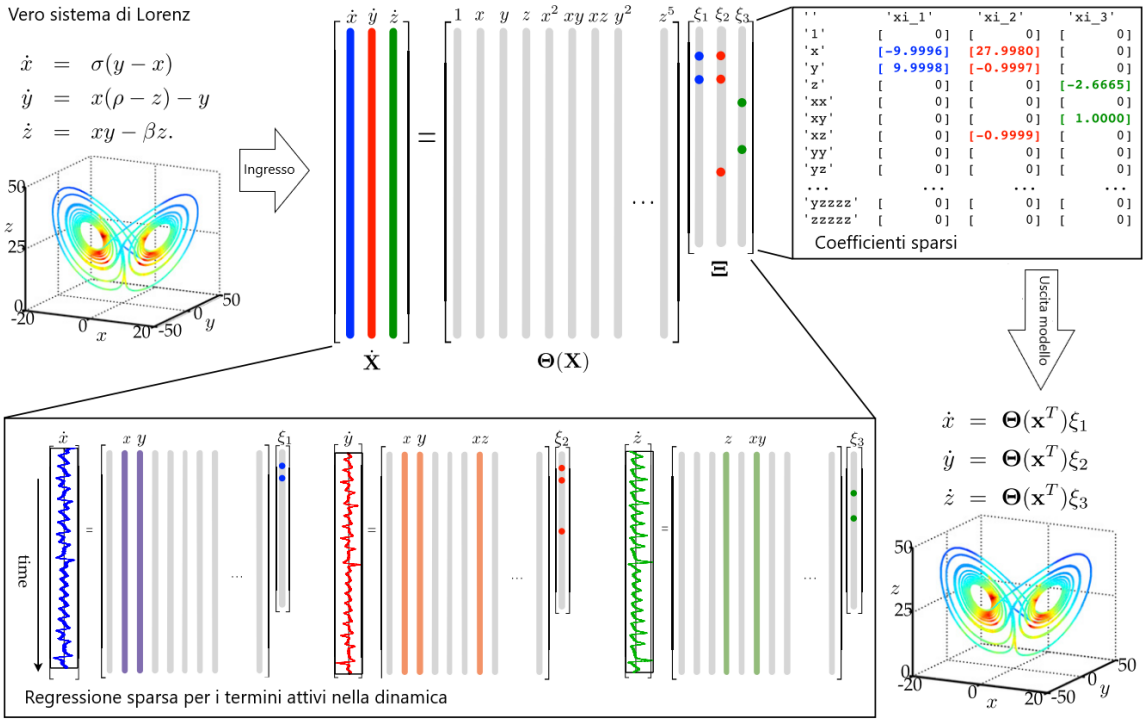

Figura 4.1: Schema dell'algoritmo SINDy, applicato al sistema di Lorenz

questo esempio, i dati vengono raccolti per il sistema Lorenz e vengono salvati in due matrici di dati X e $\dot{X}$, dove ogni riga di X è un'istantanea dello stato x nel tempo e ogni riga di $\dot{X}$ è un' istantanea della derivata temporale dello stato x nel tempo. Qui, la dinamica è identificata nello spazio dei monomi $\Theta(X)$ in (x, y, z) fino al un ordine di grado r

$$\Theta(X) = \begin{bmatrix} | & | & | & | & | & | & | \\ z(t) & y(t) & x(t) & z(t)^2 & y(t)z(t) & \dots & x(t)^r \\ | & | & | & | & | & | & | \end{bmatrix} \tag{4.2}$$

Ogni colonna di $\Theta(X)$ rappresenta una funzione candidata per il lato destro dell'equazione 3.1. Poiché solo alcuni di questi termini sono attivi in ogni f, risolviamo il problema della regressione sparsa nell'equazione 3.5 per determinare i vettori sparsi dei coefficienti $\Xi = \begin{bmatrix} \xi_1 & \xi_2 & \dots & \xi_n \end{bmatrix}$ che determinano quali termini sono attivi nelle dinamiche. Questo è illustrato in Figura 4.1, dove si trovano vettori sparsi ξ_k che rappresentano la derivata $\dot{x}^k$ come combinazione lineare del minor numero di termini in $\Theta(X)$. Come mostrato in Figura 4.1, il modello sparso riproduce fedelmente la dinamica dell'attrattore da misure di traiettoria caotica. Il codice MATLAB presentato in appendice A.0.2 implementa l'algoritmo SINDy (Sparse Identification of Nonlinear Dynamics) per l'approssimazione di sistemi dinamici non lineari a partire da dati osservati. L'algoritmo si basa sulla regressione lineare sparsa per identificare

una rappresentazione del sistema in termini di monomi di grado specificato.
Nella prima parte del codice vengono definiti i parametri del sistema e del dataset da generare:

- t_0 e t_max: tempo iniziale e finale della simulazione
- σ, ρ, β: parametri del sistema di Lorenz
- σ_y^2: varianza del rumore bianco aggiunto alle uscite
- r: grado dei monomi utilizzati per l'espansione polinomiale
- $initial\_condition$: condizioni iniziali del sistema di Lorenz
- $num\_samples$: numero di campioni da generare

Questo permette di modificare la simulazione intervenendo sull'errore e sul numero di campioni ed è fondamentale per mettere a confronto varie situazioni e verificare la bontà dell'algortmo implementato. Verranno quindi valorizzati i dati iniziali per generare varie casistiche come dati molto rumorosi, o Θ malcodinzionata. Inoltre come indicato nella capitolo è fondamentale definire Λ che è il vettore di valori di λ da testare per la regressione sparsa (spaziati logaritmicamente da -2 a 5 in base 10).
Successivamente vengono generati i dati del sistema di Lorenz tramite la funzione $generate\_lorenz\_dataset$, presentata in appendice A.0.3, che genera dati dal sistema di Lorenz con l'aggiunta di rumore bianco.
Questo permette di simulare situazioni in cui i dati a disposizione non sono esattamente quelli di sistema a causa di possibili errori dati dai sensori di misurazione.
Il codice MATLAB presentato in appendice A.0.3 genera quindi set di dati simulando il sistema caotico di Lorenz e aggiungendo rumore. Vengono presi diversi parametri in ingresso e vengono restituite diverse uscite:

- Y: Segnale con rumore (derivata approssimata dello stato del sistema di Lorenz)
- X: Stato del sistema di Lorenz nel tempo
- t: vettore temporale
- $\frac{dx(t)}{dt}$: Derivata dello stato del sistema di Lorenz (senza rumore)

I parametri in ingresso (passati da A.0.2 che esegue tutte le principali funzioni) sono:

- $initial\_condition$: Condizioni iniziali del sistema di Lorenz (che è un vettore 1×3)

- t_0: Tempo iniziale della simulazione
- t_{max}: Tempo massimo della simulazione
- σ: Parametro di Lorenz (σ)
- ρ: Parametro di Lorenz (ρ)
- β: Parametro di Lorenz (β)
- σ_y^2: Varianza del rumore bianco aggiunto alle uscite
- $num\_samples$: Numero desiderato di campioni, che permette di generare un numero di campioni preciso

Per la generazione dei dati viene utilizzato il solver $ode45$, per permettere, dato un sistema di equazioni differenziali di Lorenz, di risolvere le equazioni differenziali con le condizioni iniziali specificate e restituire i valori di x dello stato del sistema.
Viene successivamente effettuato un downsampling dei dati per ottenere il numero desiderato di campioni ($num\_samples$) e vengono mantenuti solo i campioni a intervalli regolari nel tempo t e nello stato x.
Viene calcolata la derivata dello stato del sistema di Lorenz ($\frac{dx(t)}{dt}$) per ogni campione che rappresenta il segnale senza rumore e viene successivamente generata una matrice di rumore bianco casuale di varianza definita (σ_y^2) e aggiunto alla derivata. Il risultato, Y rappresenta il segnale con rumore che approssima il calcolo della derivata.
Successivamente la traiettoria del sistema di Lorenz viene visualizzata in 3D usando plot3 come si può vedere nella Figura 4.2.

La funzione $get\_regression\_matrix$, definita in A.0.4 crea una matrice di regressione Θ contenente termini monomiali di grado r.
Restituisce:

- $train\_idx$: indici per i dati di training
- val_idx: indici per i dati di validazione
- Θ: matrice di regressione
- $column\_labels$: etichette per i termini monomiali (nomi delle colonne di Θ)

Questa funzione serve a calcolare i monomi di un vettore x elevato a un grado r e a creare una matrice di regressione. Inoltre fornisce per gli indici di training e validation quale monomio

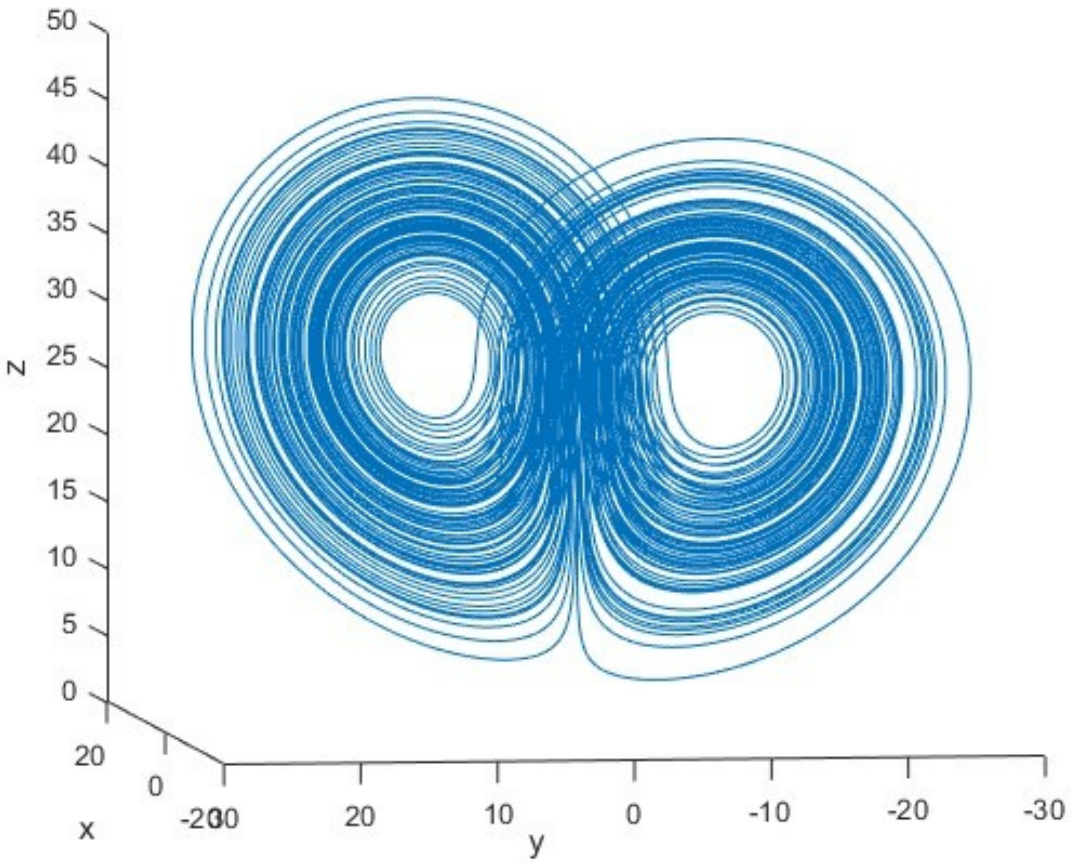

Figura 4.2: Simulazione del sistema di Lorenz

corrisponde a una colonna specifica (come indicato nel capitolo 2.4.2, Θ e Y vengono divisi in set di training e validation usando $train_{idx}$ e val_{idx}).
Vengono quindi considerati in input:

- x: Un vettore di dati di ingresso
- r: Il grado a cui elevare i valori di x

I valori di output della funzione sono

- $train\_idx$: indici per i dati di training
- val_idx: indici per i dati di validazione
- $monomials$: La matrice di regressione contenente i monomi calcolati
- $column\_labels$: Un array di celle contenente etichette per ogni colonna della matrice di regressione, che indicano il monomio corrispondente

Viene quindi considerata la dimensione dello spazio di stato (numero di elementi in x), n e il numero di campioni in x, t.
Viene quindi inizializzata la matrice $monomials$ che verrà utilizzata per memorizzare i monomi calcolati. La sua dimensione è t righe (per ogni campione) e $nchoosek(n + r, r)$ colonne (per tutti i possibili monomi con grado r). Viene anche inizializzato il vettore $column\_labels$ che conterrà le etichette delle colonne e definito $degree$ come il grado di elevazione (r) e inizializzato exp il vettore che verrà utilizzato per memorizzare gli esponenti dei monomi durante l'iterazione.
Viene utilizzato un ciclo $while(true)$ per iterare su tutte le combinazioni di esponenti possibili fino a raggiungere il grado massimo r per ogni elemento in x.
All'interno del ciclo, un ulteriore ciclo for itera su ogni elemento k in x.
Per ogni elemento k, il monomio corrente in $monomials$ viene calcolato elevando l'elemento $x(i, k)$ all'esponente $exp(k)$ e moltiplicandolo per il monomio corrente.
L'etichetta della colonna viene creata usando la funzione $sprintf$ per formattare gli esponenti in una stringa separata da virgole.
Questa stringa viene poi aggiunta all'array $column\_labels$.
Il ciclo $while$ termina quando l'esponente del primo elemento $exp(1)$ raggiunge il grado massimo n_2, che è il grado di elevazione. La funzione $mono\_between\_next\_grevlex$, presentata in appendice A.0.5 genera la sequenza dei monomi successivi in ordine lessicografico inverso graduato(grevlex), tenendo conto di un grado totale che deve essere compreso tra due valori specifici e utilizza come variabili:

- M: la dimensione dello spazio, ovvero il numero di variabili
- N_1: il grado minimo consentito, deve essere non negativo e minore o uguale a N_2
- N_2: il grado massimo consentito, deve essere non negativo
- X: un array di interi di dimensione M che rappresenta il monomio corrente

Per iniziare la sequenza, è necessario impostare X come un array riempito di zeri eccetto per l'ultima componente, la quale deve essere impostata a N_1. Come output la funzione restituisce X che è l'array che viene aggiornato con il monomio successivo nella sequenza. La funzione controlla che N_1 e N_2 siano validi, ovvero non negativi e con N_2 maggiore o uguale a N_1 e si verifica che il grado totale del monomio corrente (X) sia compreso tra N_1 e N_2. Se N_2 è uguale a 0, significa che è stato raggiunto l'ultimo elemento della sequenza e la funzione termina. Se il primo elemento di X è uguale a N_2, allora significa che si è arrivati alla fine di una sottosequenza con grado totale fissato e in questo caso il primo elemento viene azzerato e l'ultimo elemento viene impostato a N_1 per ricominciare la generazione di monomi con il grado totale minimo consentito. Altrimenti, viene chiamata la funzione $mono\_next\_grevlex$, presentata in appendice per ottenere il prossimo monomio secondo l'ordine grevless lessicografico, mantenendo lo stesso grado totale [28]. La funzione $mono\_next\_grevlex$ serve per calcolare il prossimo monomio dell'ordine grevlex. Per l'esecuzione delle funzione vengono richiesti i senguenti parametri in ingresso:

- M: la dimensione dello spazio che rappresenta il numero di variabili presenti nel monomio.
- X: un array di interi di dimensione M che rappresenta il monomio corrente.

In uscita la funzione restituisce l'array X aggiornato che rappresenta il prossimo monomio nell'ordine grevlex. La funzione itera attraverso gli elementi dell'array X e viene cercato il primo elemento diverso da zero. Una volta trovato questo elemento, la funzione esegue alcune operazioni per calcolare il prossimo monomio nell'ordine grevlex.
Una volta generata la matrice dei monomiali che rappresenta la matrice di regressione si è passati alla stima dei parametri di sparsità ed è stata fatta con tre approcci (come mostrato nel capitoli 2.4.2, 2.4.4 e 2.4.5) e poi è stata implementata la soluzione che non richiede il parametro di sparsità λ, anche qua con tre approcci (come mostrato nel capitoli 3.1.3, 3.1.5 e 3.1.7). Il primo approccio per cercare il parametro di sparsità ottimo è quello dell'utilizzo di CV. E' stata quindi implementata la funzione $SINDY\_CV$, discussa nel dettaglio in 3.1.2 esegue SINDy usando la regressione sparsa per ogni λ in Λ sui dati di training e valuta l'errore di validazione per ogni

modello sparso.
Viene restituita dalla funzione:

- λ^*_{CV}: λ ottimale che minimizza l'errore di validazione
- Ξ_{CV}: coefficienti di regressione sparsa per il λ^*_{CV}
- $bigind s_{CV}$: indici dei termini monomiali inclusi nel modello sparso
- $smallinds_{CV}$: indici dei termini monomiali esclusi dal modello sparso

Il successivo approccio per cercare il parametro di sparsità ottimo è quello dell'utilizzo di GCV. E' stata quindi implementata la funzione $SINDY\_GCV$ (simile a SINDY_CV) che esegue SINDy usando il criterio GCV per la selezione di λ che discussa nel dettaglio nel capitolo 3.1.4.
Viene restituita dalla funzione:

- λ^*_{GCV}: λ ottimale basato su GCV
- Ξ_{GCV}: coefficienti di regressione sparsa per λ^*_{GCV}
- $bigind s_{GCV}$: indici dei termini monomiali inclusi
- $smallinds_{GCV}$: indici dei termini monomiali esclusi

In seguito viene eseguito SINDy con il criterio SURE, tramite la funzione $SINDY\_SURE$ per la selezione λ, tenendo conto della varianza di Y, presentato nel capitolo 3.1.6.
Viene poi restituito:

- λ^*_{SURE}: λ ottimale basato su SURE
- Ξ_{SURE}: Coefficienti di regressione sparsa per λ^*_{SURE}
- $bigind s_{SURE}$: Indici dei termini monomiali inclusi
- $smallinds_{SURE}$: Indici dei termini monomiali esclusi
- err: Errore SURE

Vengono successivamente messi a confronto i λ ottimi ottenuti da $SINDY\_CV$, $SINDY\_GCV$ e $SINDY\_SURE$ e successivamente generati nomi di variabili $variableNames$ in base al numero di colonne in Ξ e vengono create le tabelle Ξ_{CV}_Table, Ξ_{GCV}_Table, Ξ_{SURE}_Table con i coefficienti di regressione sparsa corrispondenti ai λ^* ottimali ottenuti dai criteri SINDY_CV, SINDY_GCV e SINDY_SURE, rispettivamente.
In seguito è stato anche implentato il metodo per evitare di lavorare con il parametro di sparsità λ presentato nel capitolo 3.1.3 e si è dimostrato un metodo efficace per stimare il sistema di Lorenz utilizzando gli stessi criteri di scelta del modello CV, GCV e SURE presentati rispettivamente nei successivi capitoli 4.1.2, 4.1.4 e 4.1.6

4.1.1 Stima parametri con CV

E' stato eseguito l'algoritmo presentato nel capitolo 3.1.2 per il sistema di Lorenz.
Si può notare come l'argoritmo è stato in grado di ricostruire il Sistema di Lorenz dove nella prima colonna sono presenti i gradi dei monomiali e nelle altre il valore per le componenti nel sistema

coefficienti_stimati_CV	x_1	x_2	x_3
000	0	0	0
001	0	0	-2.6667
010	9.9999	-1.0000	0
100	-9.9999	28.0000	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	-1.0000	0
110	0	0	0.9999
200	0	0	0
003	0	0	0
012	0	0	0
021	0	0	0
030	0	0	0
102	0	0	0
111	0	0	0
120	0	0	0
201	0	0	0
210	0	0	0
300	0	0	0

4.1.2 Stima parametri senza λ con CV

Di seguito si mostrano i risultati dell'approccio indrodotto nel capitolo 3.1.3 in cui si è implementato l'algoritmo senza l'utilizzo di λ ma andando a rimuovere gli elementi in valore assoluto più piccoli. E' stata provata una situazione con molto rumore dove la stima ai minimi quadrati non riesce a dare risultati mentre la stima usando questo approccio innovativo che non necessita di λ, iterando per il numero di incognite e rimosse le componenti con valore assoluto minimo, riesce a trovare valori accettabili per stimare la dinamica del Sistema di Lorenz

coefficienti_stimati_LS	x_1	x_2	x_3
000	86.9428	204.3854	169.7598
001	-17.7229	-41.4828	-36.5050
010	12.1832	-17.9317	-10.3511
100	-5.3572	97.0660	49.9827
002	1.1451	2.6606	2.1229
011	-0.3627	-0.5274	-0.3397
020	-0.7791	1.7757	0.1613
101	-0.4247	-5.3769	-3.8295
110	3.5236	-1.5950	4.0935
200	-3.8858	-3.6388	-6.6810
003	-0.0235	-0.0537	-0.0414
012	0.0075	0.0539	0.0364
021	0.0312	0.0038	0.0325
030	-0.0225	-0.0770	-0.0475
102	0.0159	0.0677	0.0796
111	-0.1455	-0.1033	-0.1833
120	0.0758	0.4940	0.3139
201	0.1657	0.2376	0.2741
210	-0.0034	-0.6341	-0.3015
300	-0.0933	0.0644	-0.1441

mentre questo approccio è in grado di effettuare una corretta stima dei coefficienti

coefficienti_stimati_CV_no_λ	x_1	x_2	x_3
000	0	-0.2493	0
001	0	0	-2.6662
010	9.9886	-0.9831	0
100	-9.9995	28.1185	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	-1.0036	0
110	0	0	0.9973
200	0	0	0
003	0	0	0
012	0	0	0
021	0	0	0
030	0	0	0
102	0	0	0
111	0	0	0
120	0	0	0
201	0	0	0
210	0	0	0
300	0	0	0

4.1.3 Stima parametri con GCV

Si è provato a simulare con pochi campi a disposizione (20) e un tempo di simulazione di 50, così il sistema di Lorenz rilevato risulti essere quello in figura 4.3

dove i Coefficienti di regressione sparsa per CV non trovano un risultato mentre per GCV sì.

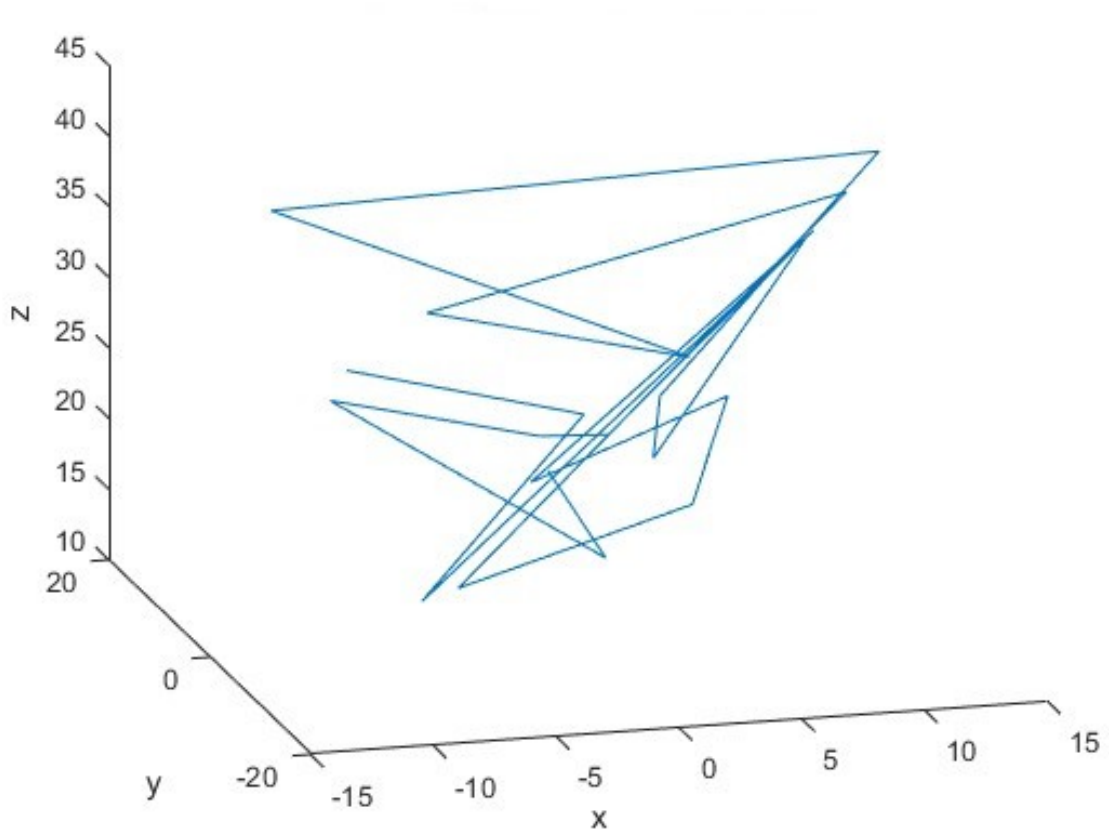

Figura 4.3: Simulazione del sistema di Lorenz con pochi campioni a disposizione

coefficienti_stimati_CV	x_1	x_2	x_3
000	0	0	0
001	0	0	0
010	0	0	0
100	0	0	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	0	0
110	0	0	0
200	0	0	0
003	0	0	0
012	0	0	0
021	0	0	0
030	0	0	0
102	0	0	0
111	0	0	0
120	0	0	0
201	0	0	0
210	0	0	0
300	0	0	0

coefficienti_stimati_GCV	x_1	x_2	x_3
000	0	0	-0.0100
001	0	0	-2.6663
010	9.9996	-0.9997	0
100	-9.9998	28.0012	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	-1.0000	0
110	0	0	1.0000
200	0	0	0
003	0	0	0
012	0	0	0
021	0	0	0
030	0	0	0
102	0	0	0
111	0	0	0
120	0	0	0
201	0	0	0
210	0	0	0
300	0	0	0

4.1.4 Stima parametri senza λ con GCV

A differenza del buon risultato visto nel capitolo 4.1.2, nel caso di utilizzo del nuovo algoritmo presentato in 3.1.3 in cui si evita l'utilizzo di λ ma con scelta del modello tramite la tecnica di GCV non porta a un miglioramento della soluzione rispetto al primo tentativo effettuato con i minimi quadrati. In particolare, solo pochi termini vengono scartati e rendendo molto sparsa la soluzione finale; sembra che la nuova tecnica del capitolo 3.1.5 abbia un risultato peggiore rispetto a quella in cui è presente una griglia di λ da scegliere per fare cut-off di coefficienti. Il test è stato effettuato con 15 campioni a disposizione e una varianza di rumore a 1.
Di seguito lista di risultati mettendo a confronto soluzione tramite CV, GCV, minimi quadrati e infine GCV senza l'utilizzo del λ.

coefficienti_stimati_CV	x_1	x_2	x_3
000	0	0	0
001	0	0	0
010	0	0	0
100	0	0	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	0	0
110	0	0	0
200	0	0	0

coefficienti_stimati_GCV	x_1	x_2	x_3
000	-0.1748	3.0608	6.1278
001	0	-0.2099	-0.2099
010	10.0602	10.0602	0.6024
100	0.6024	0.6024	0.6024
002	0	0	0.0232
011	0	0.0372	0.0123
020	0	0.02829	0
101	0	-1.0536	0.0281
110	0	-0.0696	1.0429
200	0	0.0612	-0.0790

coefficienti_stimati_LS	x_1	x_2	x_3
000	-0.1144	4.5689	6.0150
001	-0.0271	-0.3993	-3.4399
010	9.5714	-2.4527	0.6234
100	-8.9644	30.1029	-1.1812
002	0.0017	0.0056	0.0228
011	0.0207	0.0433	-0.0130
020	-0.0099	0.0321	-0.0006
101	-0.0342	-1.0597	0.0290
110	-0.0021	-0.0677	1.0435
200	-0.0009	0.0446	-0.0783

coefficienti_stimati_GCV_no_λ	**x_1**	**x_2**	**x_3**
000	-0.1144	4.5689	6.1278
001	-0.0271	-0.3993	6.1278
010	9.5714	-2.4527	0.6024
100	-8.9644	30.1029	-1.1541
002	0.0017	0.0056	0.0232
011	0.0207	0.0433	-0.0123
020	-0.0099	0.0321	0
101	-0.0342	-1.0597	0.0281
110	-0.0021	-0.0677	1.0429
200	-0.0009	0.0446	-0.0790

4.1.5 Stima parametri con SURE

Anche in questo caso si è provato a simulare pochi campi a disposizione (20) e un tempo di simulazione di 50 e dove i coefficienti di regressione sparsa per CV non trovano un risultato mentre per SURE sì

coefficienti_stimati_CV	x_1	x_2	x_3
000	0	0	0
001	0	0	0
010	0	0	0
100	0	0	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	0	0
110	0	0	0
200	0	0	0
003	0	0	0
012	0	0	0
021	0	0	0
030	0	0	0
102	0	0	0
111	0	0	0
120	0	0	0
201	0	0	0
210	0	0	0
300	0	0	0

coefficienti_stimati_SURE	x_1	x_2	x_3
000	0	0	-0.0100
001	0	0	-2.6663
010	9.9996	-0.9997	0
100	-9.9998	28.0012	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	-1.0000	0
110	0	0	1.0000
200	0	0	0
003	0	0	0
012	0	0	0
021	0	0	0
030	0	0	0
102	0	0	0
111	0	0	0
120	0	0	0
201	0	0	0
210	0	0	0
300	0	0	0

4.1.6 Stima parametri senza λ con SURE

Anche questo caso risponde con le stesse criticità riscontrate nel capitolo 4.1.4.
Di seguito lista di risultati mettendo a confronto soluzione tramite CV, SURE, minimi quadrati e infine SURE senza l'utilizzo del λ con 15 campioni a disposizione e una varianza di rumore a 1.

coefficienti_stimati_CV	x_1	x_2	x_3
000	0	0	0
001	0	0	0
010	0	0	0
100	0	0	0
002	0	0	0
011	0	0	0
020	0	0	0
101	0	0	0
110	0	0	0
200	0	0	0

coefficienti_stimati_SURE	x_1	x_2	x_3
000	1.4602	-6.7199	2.5004
001	-0.04557	0.8029	-3.0142
010	8.1931	-0.2437	0.0642
100	-7.0960	27.2298	-0.06833
002	0	-0.0243	0.0119
011	0.0542	-0.0299	0
020	0.0285	-0.0298	0.0129
101	-0.0828	-0.9722	0
110	-0.06589	0	1.0084
200	0.0289	0.0751	-0.0430

coefficienti_stimati_LS	x_1	x_2	x_3
000	-0.1522	-6.7381	2.0260
001	0.1569	0.8009	-2.96098
010	8.3242	-0.2086	0.22955
100	-7.2574	27.17627	-0.3321
002	-0.0060	-0.0241	0.0104
011	0.0477	-0.0308	-0.0057
020	0.0244	-0.0304	0.0103
101	-0.0763	-0.9708	0.0080
110	-0.0678	0.0020	1.0131
002	0.0467	0.0731	-0.0419

coefficienti_stimati_SURE_no_λ	x_1	x_2	x_3
000	-0.1522	-6.7199	2.0260
001	0.1569	0.8029	-2.96098
010	8.3242	-0.2437	0.22955
100	-7.2574	27.2298	-0.3321
002	-0.0060	-0.0243	0.0104
011	0.0477	-0.0299	-0.0057
020	0.0244	-0.0298	0.0103
101	-0.0763	-0.9722	0.0080
110	-0.0678	0	1.0131
002	0.0467	0.0751	-0.0419

4.2 Risposta impulsiva su FIR

E' stata anche implementata la possibilità di testare l'algoritmo su una risposta impulsiva di un sistema lineare tempo invariante[29].
In particolare l'obiettivo è quello di stimare la risposta all'impulso g_0 di una dinamica a tempo discreto, lineare e causale, a partire da dati di uscita rumorosi. Il modello di misure è

$$y(t) = \sum_{k=1}^{\infty} g_k^0 u(t-k) + e(t), t = 1, \dot{} N \tag{4.3}$$

dove t indica il tempo, l'intervallo di campionamento è un'unità di tempo per semplicità, g_k^0 indicano i coefficienti di risposta all'impulso, $u(t)$ è l'ingresso noto del sistema mentre $e(t)$ è il rumore. Per determinare la risposta all'impulso dalle misure di ingresso-uscita, ci si è chiesti come parametrizzare l'incognita g^0. L'approccio classico introduce una raccolta di risposte all'impulso modelli $g(\Xi)$, ciascuno parametrizzato da un diverso vettore Ξ.
In particolare, viene adottato un modello FIR di ordine m, cioè $g^k(\Xi) = \Xi^k$ per $k = 1, \ldots, m$ e zero altrove [30] [31]. Questo permette di riformulare la 4.3 come una regressione lineare, sovrapponiamo tutti gli elementi $y(t)$ ed $e(t)$ per formare i vettori Y ed E e ottenere il modello

$$Y = \Theta\Xi + E \tag{4.4}$$

con la matrice di regressione $\Theta \in \mathbb{R}^{N \times m}$ data da

$$\Theta = \begin{bmatrix} u(0) & u(-1) & u(-2) & \dots & u(-m+1) \\ u(1) & u(0) & u(-1) & \dots & u(-m) \\ \vdots & & & & \\ u(N-1) & u(N-2) & u(N-3) & \dots & u(N-m) \end{bmatrix} \tag{4.5}$$

e successivamente è stato utilizzato l'algoritmo SINDy per stimare Ξ. La risposta impulsiva incognita g^0 è definita con la seguente funzione di trasferimento:

$$\frac{(z+1)^2}{z(z-0.8)(z-0.7))} \tag{4.6}$$

viene stimata l'uscita del sistema corroto da rumore bianco Gaussiano $e(t)$. I dati arrivano dal sistema inizialmente a riposo e a $t = 0$ viene dato in ingresso il rumore bianco filtrato con un filtro passa basso

$$\frac{z}{z-0.99} \tag{4.7}$$

Viene successivamente generato un segnale di rumore bianco che viene filtrato con un filtro passa basso per ottenere il segnale di ingresso u.
Il coefficiente del filtro passa-basso influenza il condizionamento del problema; un valore di 0.99 potrebbe portare a problemi di malcondizionatamento, questo comportamento verrà sfruttato nella simulazione per testare i differenti approcci nella scelta dei modelli.
In seguito viene simulato il sistema con un'equazione differenziale lineare con ingresso u e successivamente si crea la matrice Θ, contenente ritardi del segnale u fino a un certo numero di colonne di Ξ definite, fruttando la funzione Toeplitz [32] che genera una matrice a diagonali costanti; in cui ogni diagonale discendente da sinistra a destra è costante dove quindi gli elementi sulla stessa diagonale discendente hanno lo stesso valore, dove l'elemento della diagonale principale, indicato con a_0, determina il valore di tutti gli elementi sulla diagonale discendente principale. In altre parole, tutti gli elementi $a_{i,i}$ saranno uguali a a_0. Per quanto riguarda i valori della prima riga (o della prima colonna) determinano i valori di tutti gli elementi al di fuori della diagonale principale; per ogni i e j con $i! = j$, l'elemento $a_{i,j}$ sarà uguale al valore a_{i-j} della prima riga (o a_{j-i} della prima colonna).
Come per la simulazione effettuata con il sistema di Lorenz vengono eseguite le funzioni $SINDY\_CV$, $SINDY\_GCV$ e $SINDY\_SURE$ per il calcolo di coefficienti di sparsità.
Successivamente è stato provato l'algoritmo presentato nei capitoli 3.1.3, 3.1.5 e 3.1.7 per verificare se in questo caso si riesce a ottenere dei miglioramenti rispetto al lavorare con un griglia di λ per la scelta del modello. I test sono stati fatti con una Θ malcodizionata, con quindi il coefficiente del filtro passa-basso a -0.99999, con 500 campioni e 50 incognite così da provare che la stima con CV non è la migliore delle possibili, infatti provando successivamente con GCV e

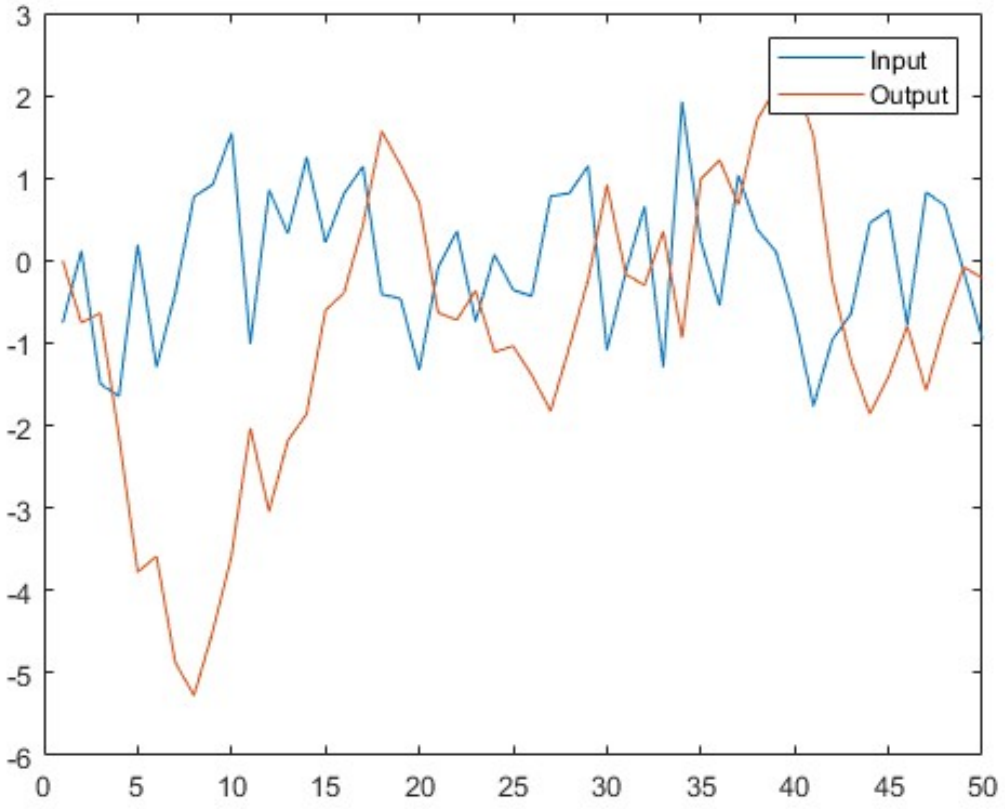

Figura 4.4: Confronto tra il rumore generato e l'uscita filtrata dal filtro passa-basso nel caso di 500 campioni e 50 incognite

SURE il fit risulterà più alto.

E' stata anche fatta un'altra simulazione con 500 dati e 100 incognite; ne verranno di seguito presentati i risultati nei due casi di studio. Di seguito anche l'immagine della stima effettuata con i minimi quadrati che risulta avere un fit 68.4 per il caso 500 campioni e 50 incognite, mentre è di 58.7 per il caso 500 campioni e 100 incognite.

Tali valori di fit miglioranno grazie alle stime presentate precedentemente.

4.2.1 Stima parametri con CV

Come per il sistema di Lorenz è stato testato l'algoritmo dove la scelta del parametro di sparsità λ è effettuata con CV come visibile nelle figure 4.8 e 4.9.

Il fit risulta 68.4 per il caso 500 campioni e 50 incognite mentre risuta 61.9 per il caso 500 campioni e 100 incognite.

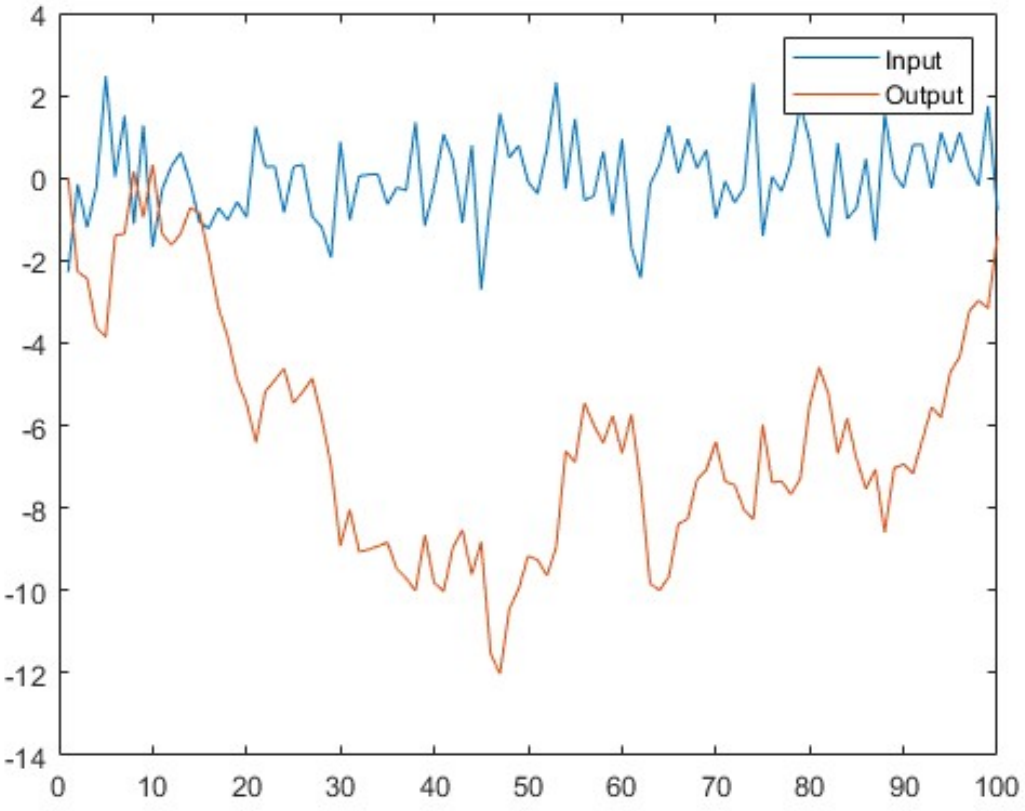

Figura 4.5: Confronto tra il rumore generato e l'uscita filtrata dal filtro passa-basso nel caso di 500 campioni e 100 incognite

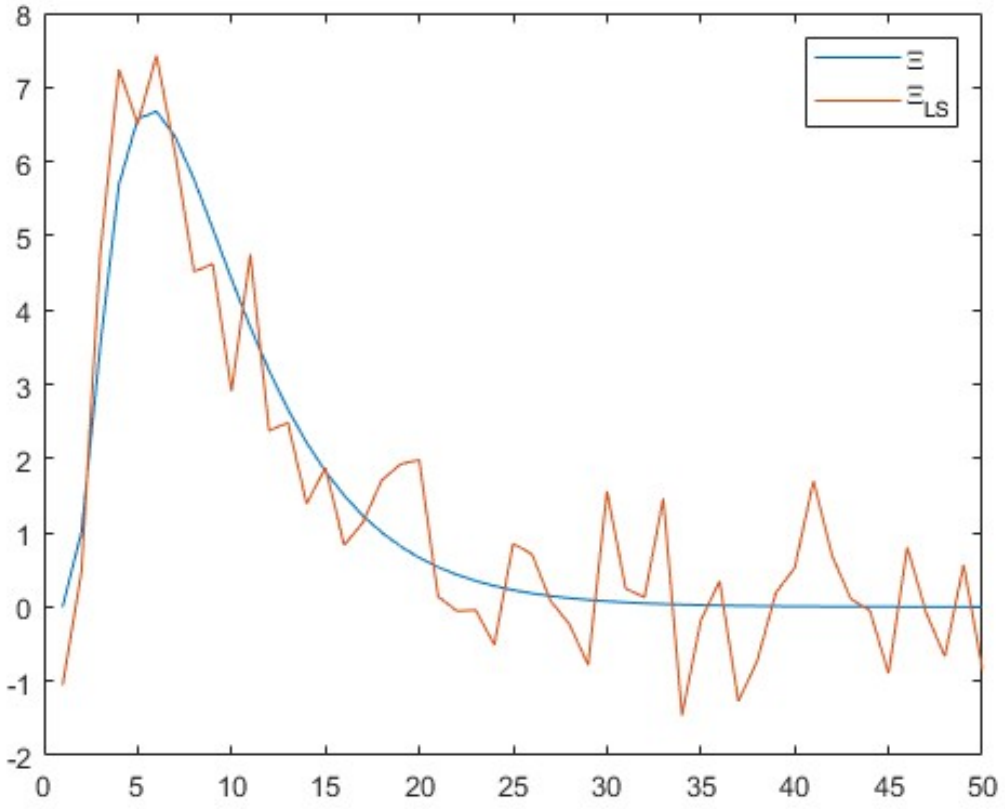

Figura 4.6: Confronto tra risposta impulsiva reale e stima LS nel caso di 500 campioni e 50 incognite

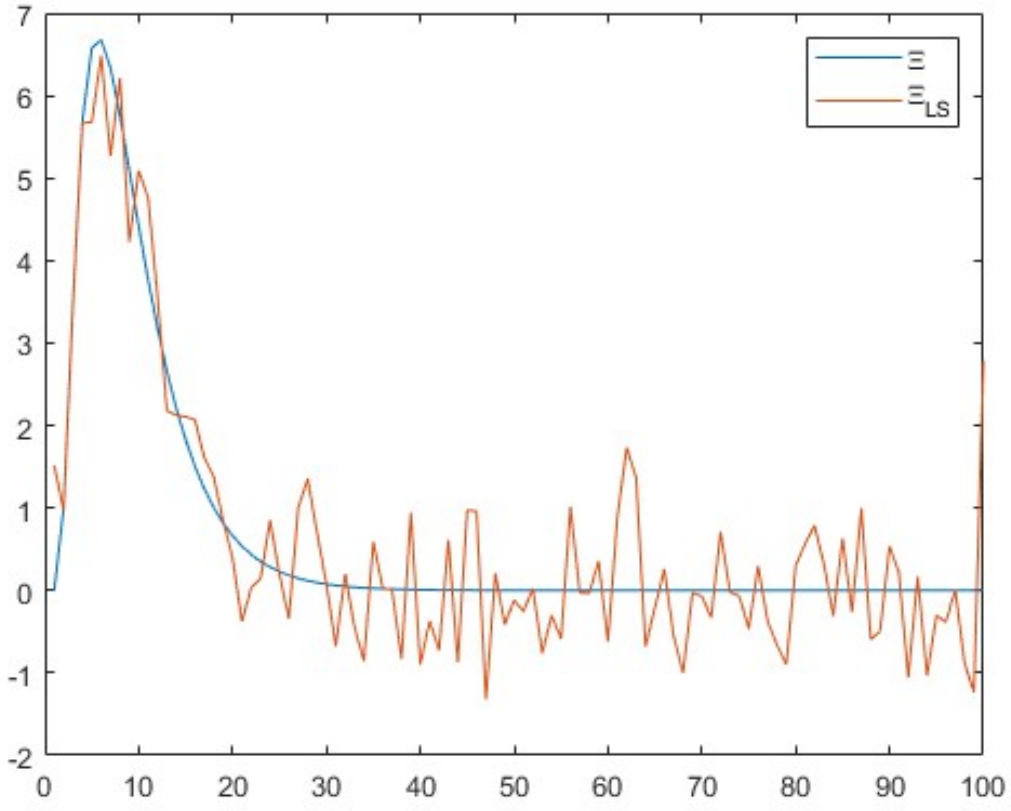

Figura 4.7: Confronto tra risposta impulsiva reale e stima LS nel caso di 500 campioni e 100 incognite

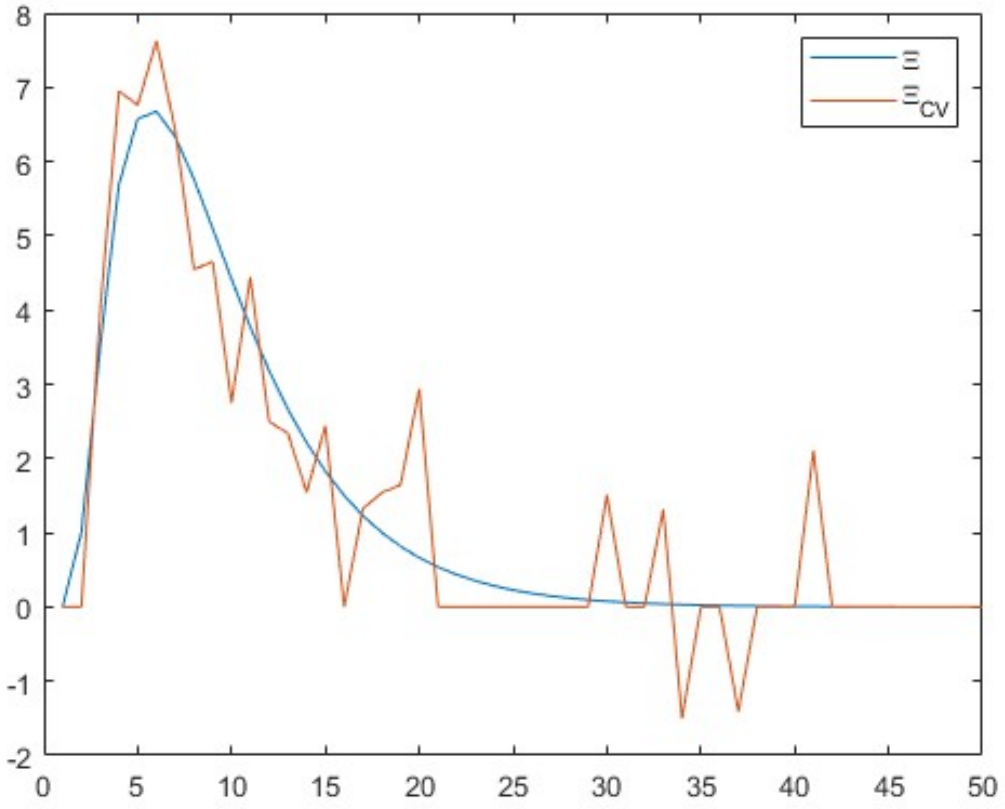

Figura 4.8: Confronto tra risposta impulsiva reale e stima CV nel caso di 500 campioni e 50 incognite

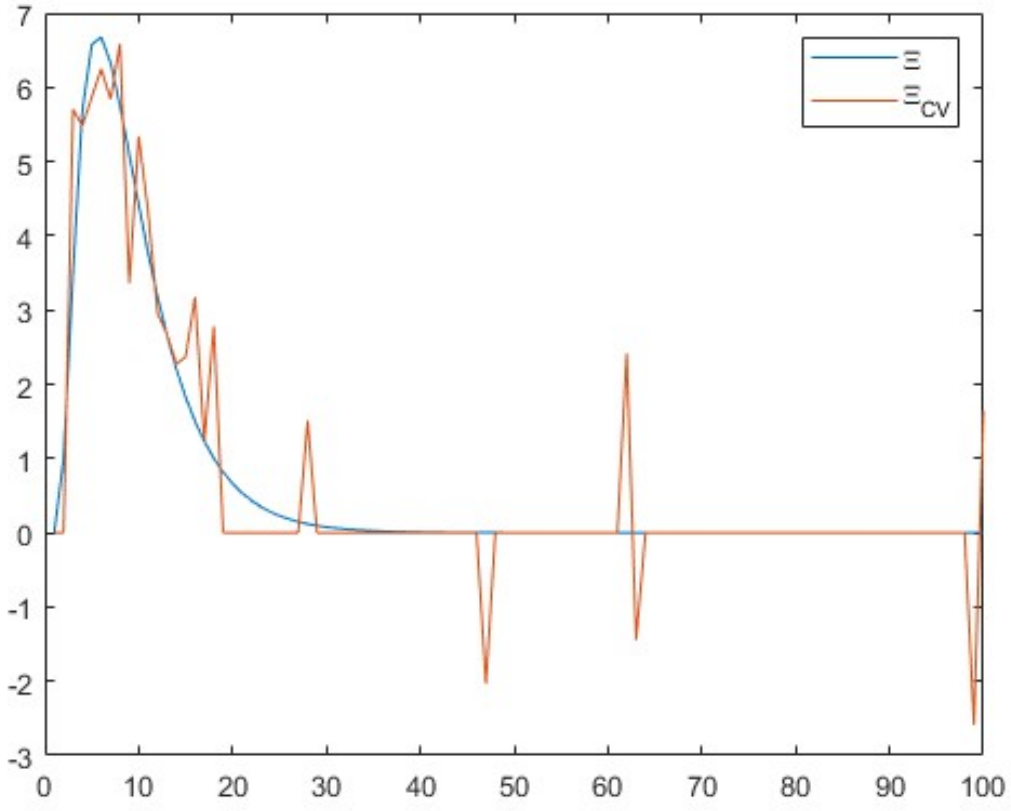

Figura 4.9: Confronto tra risposta impulsiva reale e stima CV nel caso di 500 campioni e 100 incognite

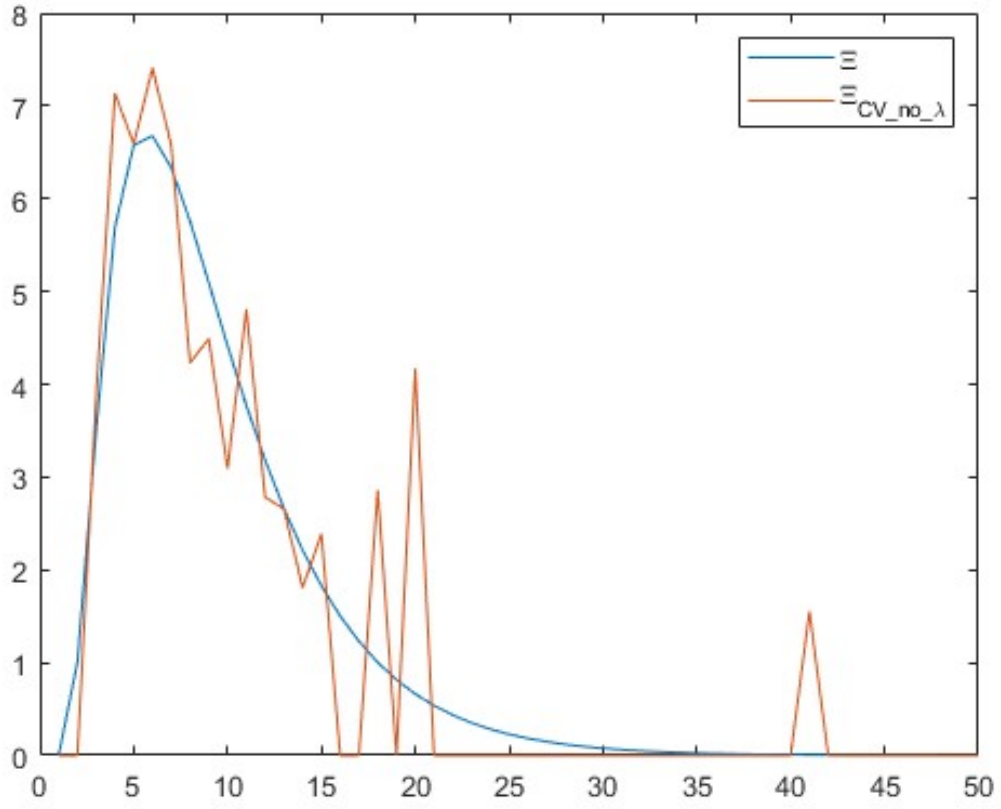

Figura 4.10: Confronto tra risposta impulsiva reale e stima CV senza λ nel caso di 500 campioni e 50 incognite

4.2.2 Stima parametri senza λ con CV

Come per il sistema di Lorenz è stato testato l'algoritmo senza λ con scelta del miglior modello con CV come visibile nelle figure 4.10 e 4.11. In questo caso l'algoritmo riesce ad togliere più coefficienti rispetto al metodo con la griglia dei λ rendendo più sparsa la soluzione finale.
Il fit risulta 69.1 per il caso 500 campioni e 50 incognite mentre risuta 65.2 per il caso 500 campioni e 100 incognite.

4.2.3 Stima parametri con GCV

Come per il sistema di Lorenz è stato testato l'algoritmo dove la scelta di λ è effettuata con GCV come visibile nelle figure 4.12 e 4.13. Risulta essere più efficace di CV in quanto il sistema è molto malcondizionato e risulta quindi essere utile l'utilizzo di tutti i dati per la stima invece che una parte come avviene con CV.

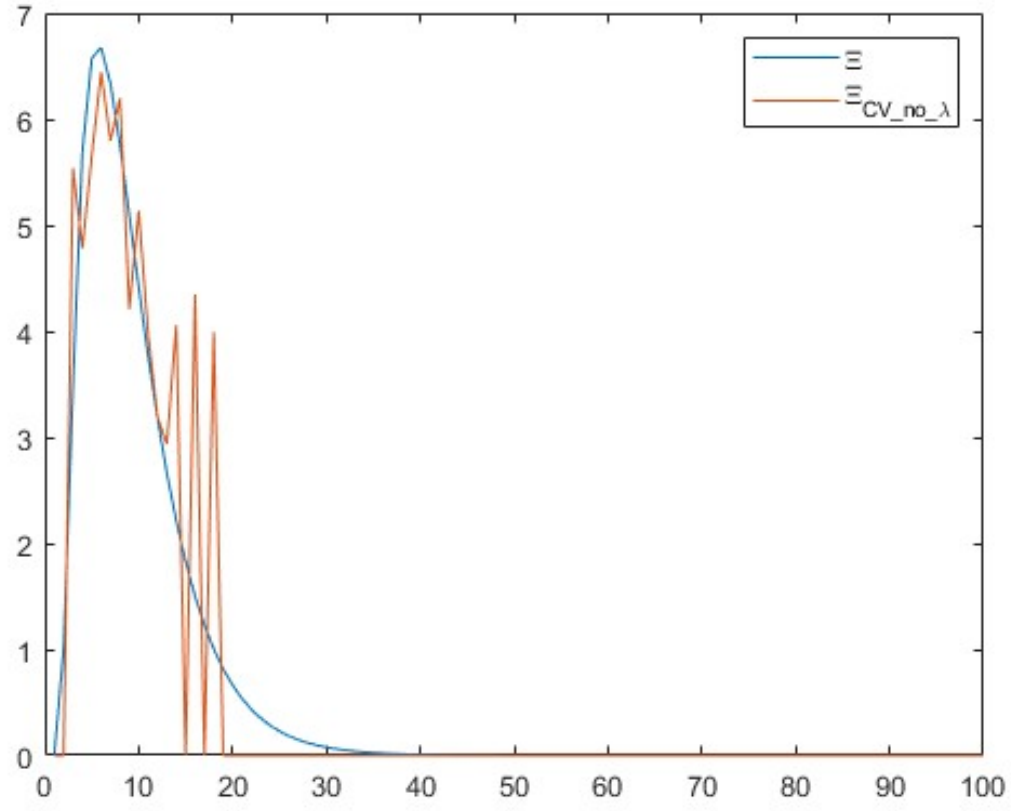

Figura 4.11: Confronto tra risposta impulsiva reale e stima CV senza λ nel caso di 500 campioni e 100 incognite

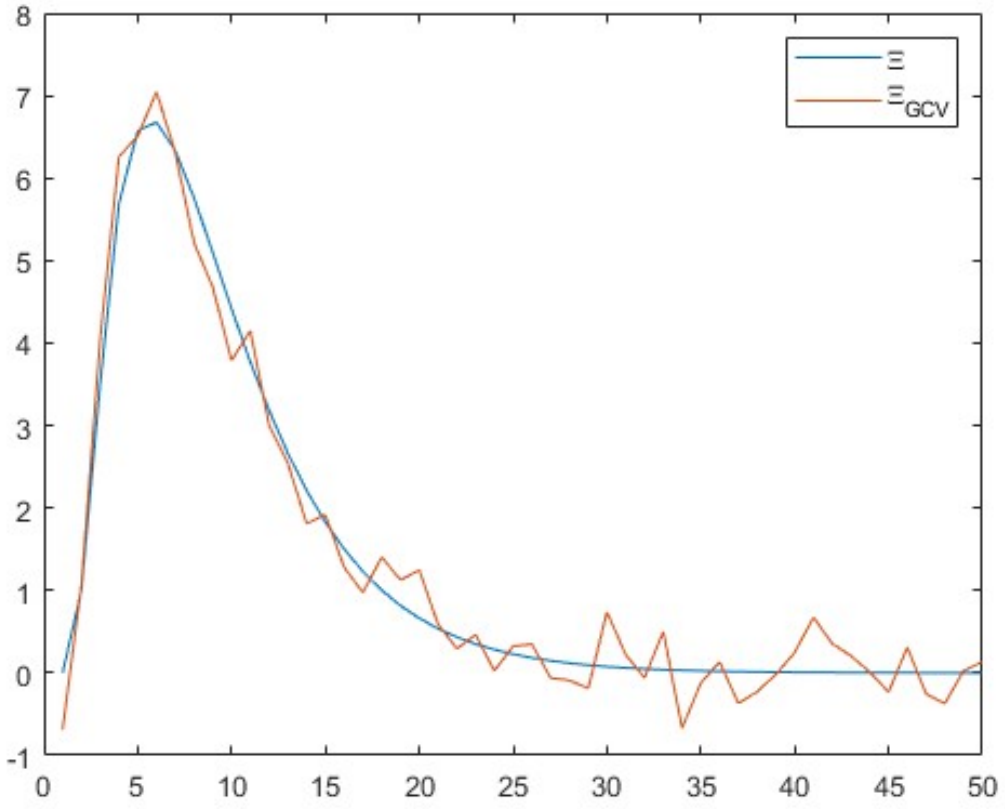

Figura 4.12: Confronto tra risposta impulsiva reale e stima GCV nel caso di 500 campioni e 50 incognite

Il fit risulta 86.4 per il caso 500 campioni e 50 incognite mentre risuta 82.9 per il caso 500 campioni e 100 incognite.

4.2.4 Stima parametri senza λ con GCV

E' stato testato l'algoritmo SINDy senza λ anche con GCV come visibile nelle figure 4.14 e 4.15.

Anche in questo caso la soluzione risulta meno sparsa e si trae un vantaggio nell'usare l'algorimo senza la scelta dei λ da una griglia.

Il fit risulta 86.4 per il caso 500 campioni e 50 incognite mentre risuta 82.9 per il caso 500 campioni e 100 incognite.

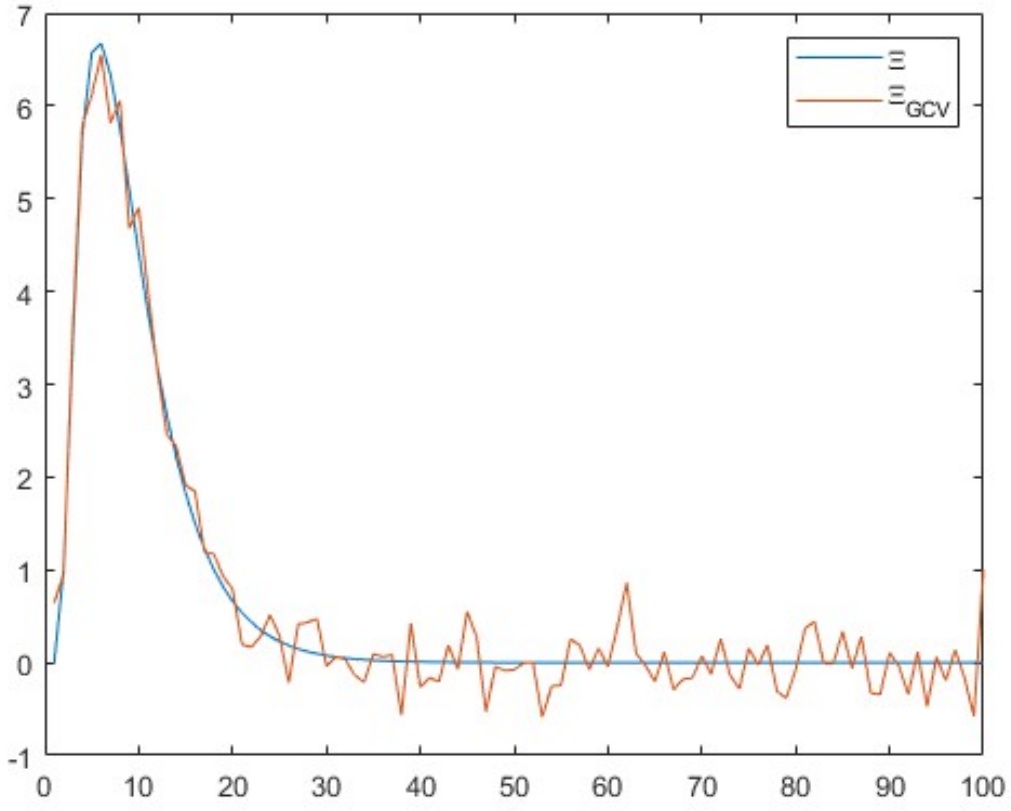

Figura 4.13: Confronto tra risposta impulsiva reale e stima GCV nel caso di 500 campioni e 100 incognite

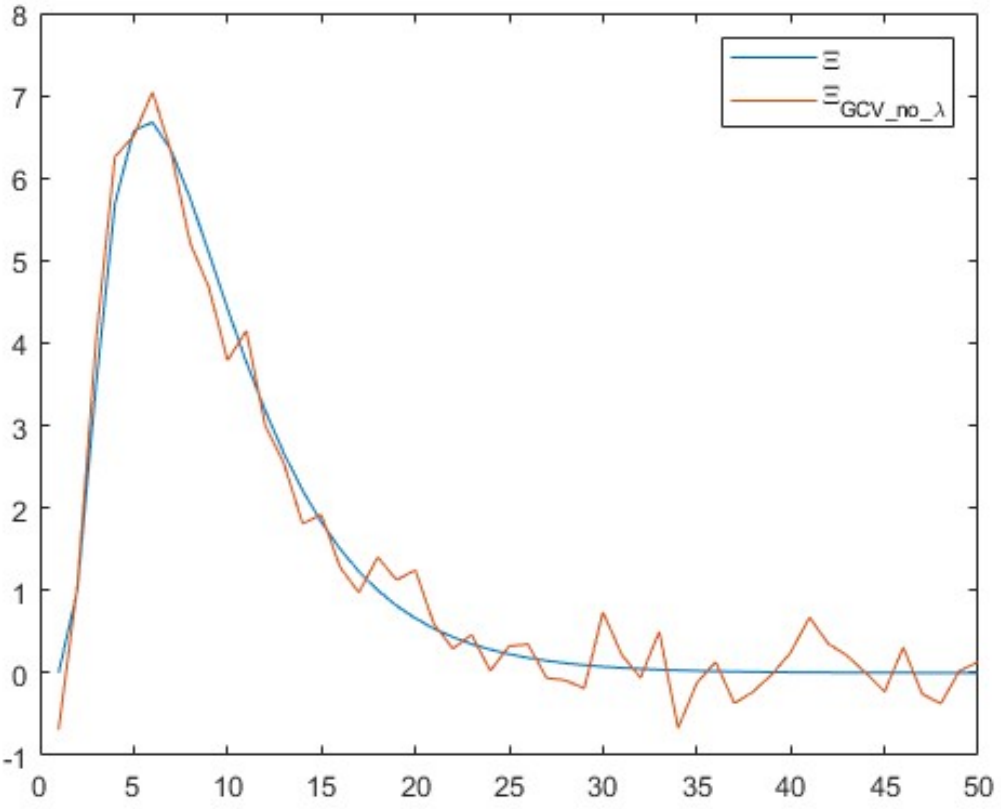

Figura 4.14: Confronto tra risposta impulsiva reale e stima con GCV senza λ nel caso di 500 campioni e 50 incognite

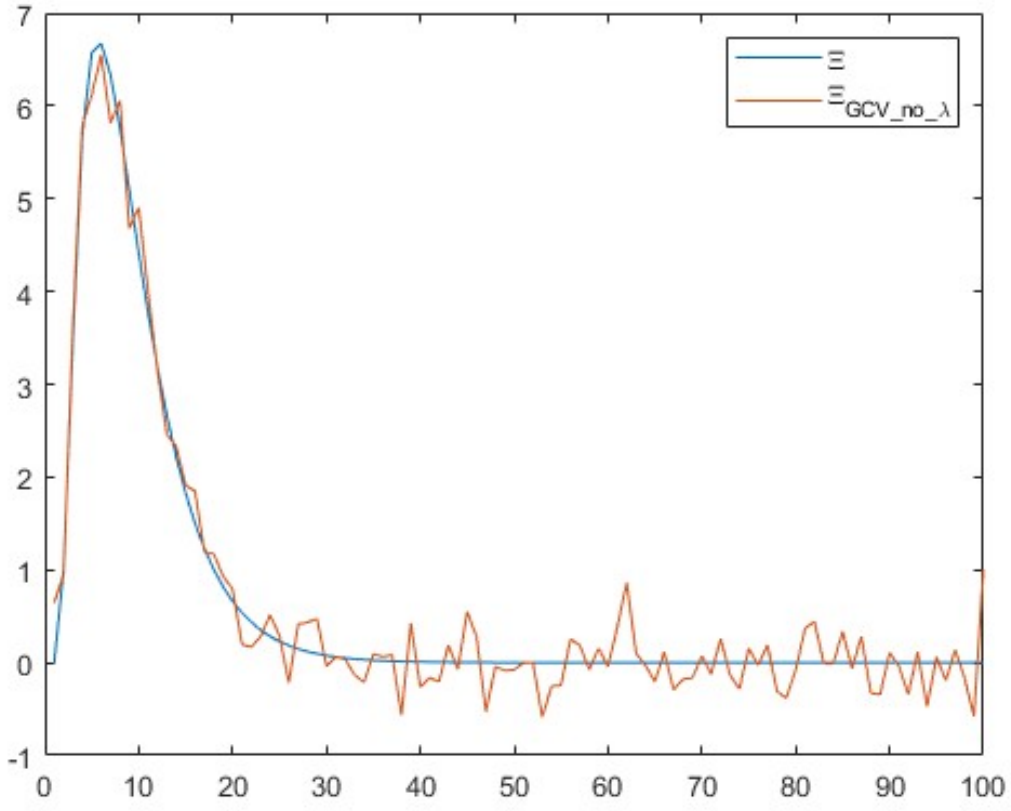

Figura 4.15: Confronto tra risposta impulsiva reale e stima con GCV senza λ nel caso di 500 campioni e 100 incognite

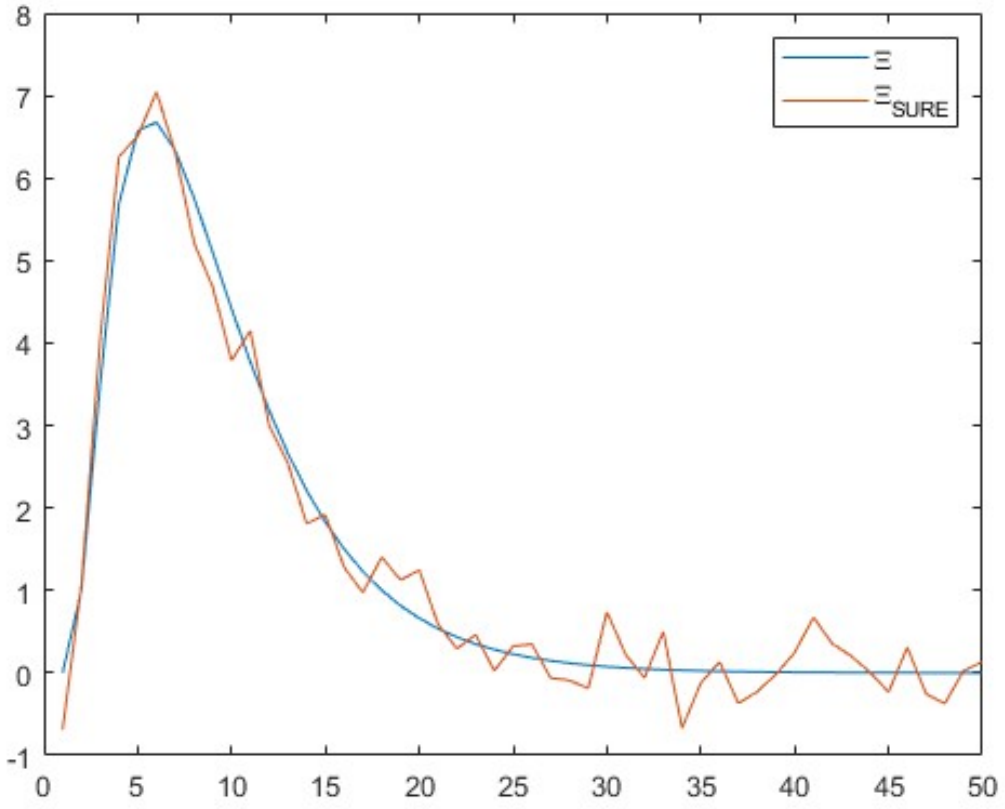

Figura 4.16: Confronto tra risposta impulsiva reale e stima con SURE nel caso di 500 campioni e 50 incognite

4.2.5 Stima parametri con SURE

Come per il sistema di Lorenz è stato testato l'algoritmo dove la scelta del parametro di sparsità λ è effettuata con SURE come visibile nelle figure 4.16 e 4.17.

Il fit risulta 86.4 per il caso 500 campioni e 50 incognite mentre risuta 82.9 per il caso 500 campioni e 100 incognite.

4.2.6 Stima parametri senza λ con SURE

E' stato testato l'algoritmo SINDy senza λ anche con SURE come visibile nelle figure 4.18 e 4.19.

Il fit risulta 86.4 per il caso 500 campioni e 50 incognite mentre risuta 83.0 per il caso 500 campioni e 100 incognite.

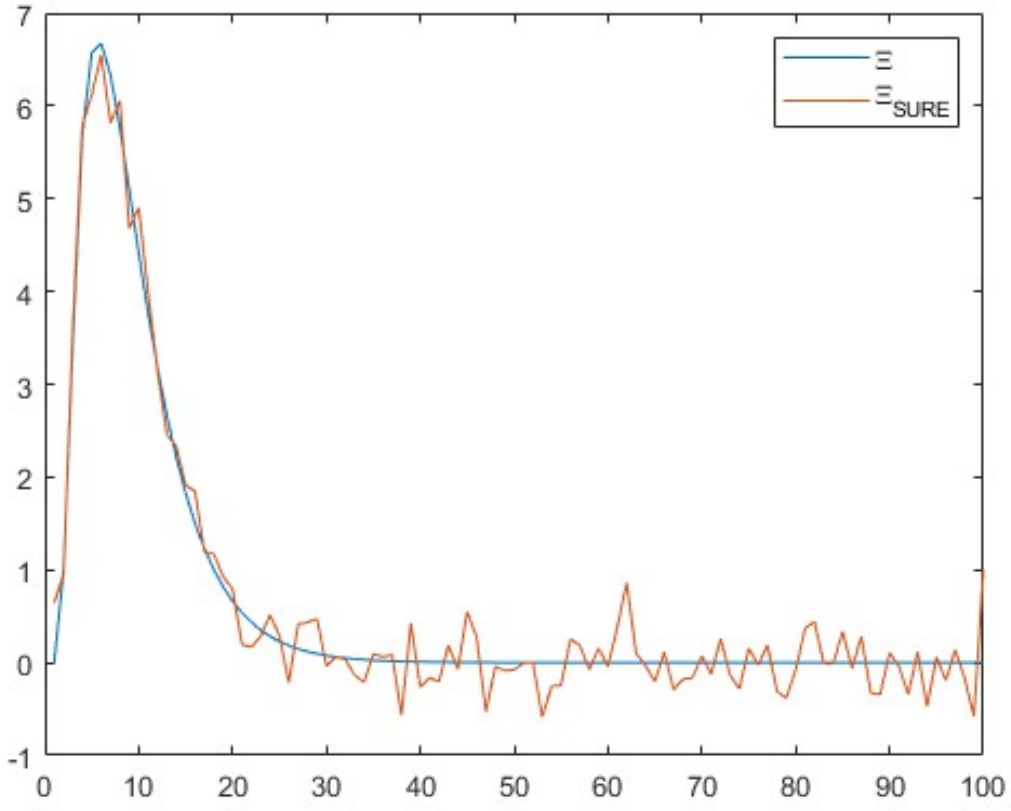

Figura 4.17: Confronto tra risposta impulsiva reale e stima con SURE nel caso di 500 campioni e 100 incognite

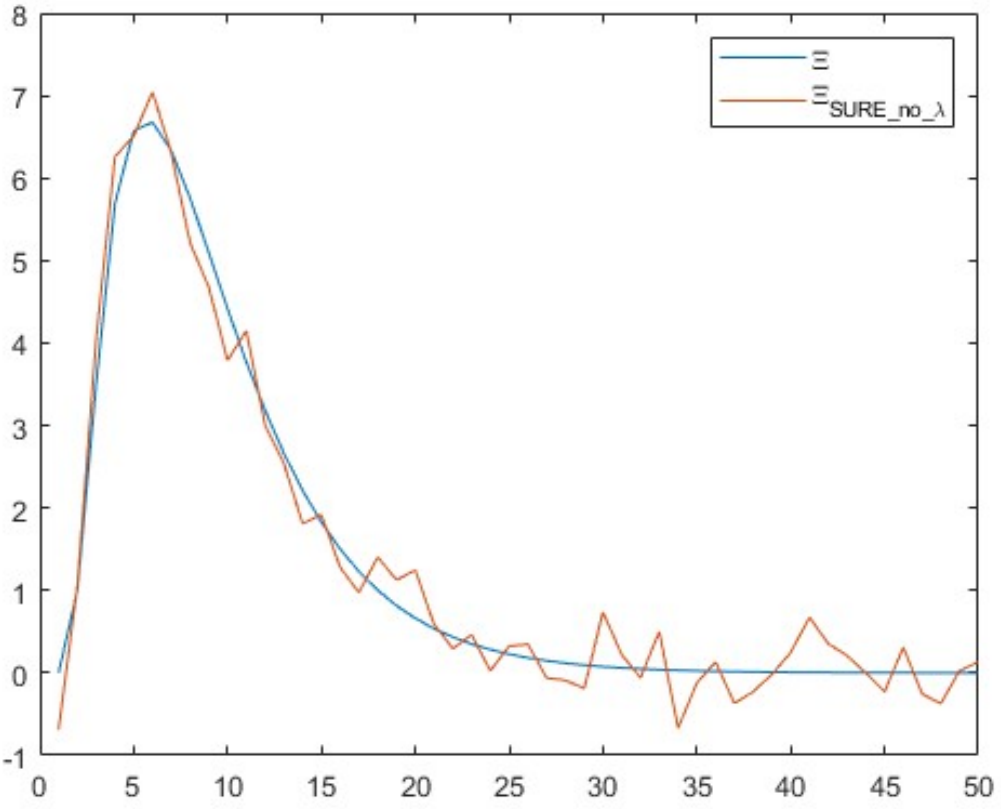

Figura 4.18: Confronto tra risposta impulsiva reale e stima con SURE senza λ nel caso di 500 campioni e 50 incognite

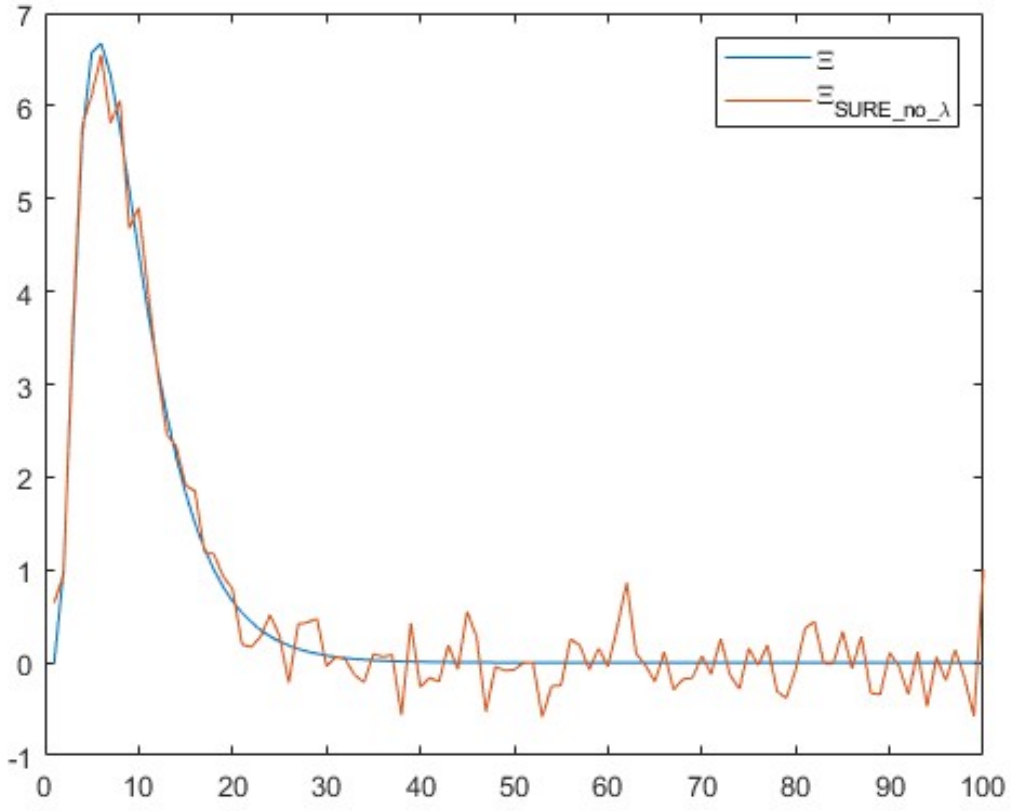

Figura 4.19: Confronto tra risposta impulsiva reale e stima con SURE senza λ nel caso di 500 campioni e 100 incognite

Capitolo 5

Conclusione

In questo lavoro di tesi di laurea si è studiato l'algoritmo SINDy (Sparse Identification of Nonlinear Dynamics) e esteso verso nuove direzioni per la stima di sistemi dinamici non lineari; è risultato utile per sistemi con un numero ristretto di termini importanti nella loro equazione di stato, con quindi una rappresentazione sparsa della dinamica. Sono stati spiegati i concetti di stima parametrica, stimatori ai minimi quadrati, stimatori unbiased e si sono viste tecniche di stima come cross-validazione e SURE.
E' stata successivamente illustrata la stima sparsa di sistemi dinamici non lineari tramite l'algoritmo SINDy ed è stata implementata la sua variante senza stima del parametro di sparsità. Questo metodo, nonostante sia meno oneroso in termini computazionali, mostra risultati sorprendetemente simili. E' stato fatto un confronto di prestazioni delle due varianti dell'algoritmo SINDy su diversi sistemi modello.
In particolare un aspetto importante è stato quello di sviluppare SINDy che utilizzi tutti i dati senza dividerli tra training e validazione. Infine si sono visti esempi numerici sul sistema di Lorenz e sulla risposta impulsiva su sistema FIR.
L'algoritmo SINDy si è dimostrato un metodo efficace per la stima di sistemi dinamici non lineari e la nuova nuova versione di SINDy senza stima del parametro di sparsità si è dimostrato un metodo promettente per la stima di sistemi con un elevato numero di variabili.
Gli sviluppi futuri di questo lavoro potrebbero essere quelli di capire come mai dal punto dal punto di vista teorico la variante di SINDy senza parametro di sparsità funziona bene.

Appendice A

Appendice

A.0.1 Funzione sparse_representation

```
function [Xi,biginds,smallinds] = sparse_representation(Theta,
    dXdt, lambda)
% funzione che restitusce sparse Xi passando Theta, i monomiali
    di
% training, dXdt (con rumore) e  parametro di sparsità
% lambda parametro di sparsità è scelto con CV e serve dividere
    i dati
n = size(dXdt,2);
%% calcolo regressione sparsa con LS
Xi = Theta\dXdt; % primo tentantivo: Least-squares
    for k=1:10
        smallinds = (abs(Xi)<lambda); % trova coefficenti
            piccoli
        Xi(smallinds)=0; %  threshold
        for ind = 1:n % n dimensione di stato
            biginds = ~smallinds(:,ind);
            % Diminuisce le dinamiche sui termini rimanenti per
                trovare Xi
            Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
        end
    end

end
```

A.0.2 Simulazione con attrattore di Lorenz

```
% Impostazioni
t0 = 0;
tmax = 100;
sigma = 10; %parametro sigma del sistema di lorenz
rho = 28;  %parametro rho del sistema di lorenz
beta = 8/3; %parametro beta del sistema di lorenz
noise_level = 0.01; %varianza errore uscita
r=3; %grado monomiali
initial_condition = [-8,7,27];

num_samples=100000;

% Generazione del dataset su un sistema di lorenz. l'uscita sono le derivate rumorose con
% varianza noise_level

[y, x, t, dxdt, snr] = generate_lorenz_dataset(initial_condition,t0, tmax, sigma, rho, beta, noise_level, num_samples);



% Plot the solution
plot3(x(:,1), x(:,2), x(:,3));
xlabel('x');
ylabel('y');
zlabel('z');
title('Sistema di Lorenz');

% Calcolo della matrice di regressione composta dai monomiali di grado r
[train_idx, val_idx,Theta,column_labels] = get_regression_matrix(x, r);

%creo tabella con label del monomiale
```

```
ThetaTable =
array2table(Theta,'VariableNames',column_labels);

%stimo varianza con stimatore unbiased
y_true_var = y(:,1);
S_true_var = lscov(Theta,y_true_var);
T_true_var = norm(y_true_var-Theta*S_true_var)^2 / ((length(
   y_true_var) - length(S_true_var)));
var_y = sqrt(T_true_var);









% divido theta in una parte di validation e una parte di
   trainig
ThetaTrain = Theta(train_idx, :);
ThetaVal = Theta(val_idx, :);
% divido le uscite rumorose in una parte di validation e una
   parte di trainig
yTrain = y(train_idx, :);
yVal = y(val_idx, :);
dxdtTrain = dxdt(train_idx, :);
dxdtVal = dxdt(val_idx, :);

% genero vari lambda da testare successivamente in sindy
Lambda = logspace(-2, 5,10);



% eseguo sindy, viene reiterata la sparse regression per i vari
    lambda e
% viene scelto il lambda ottimo che minimizza l'errore
   validazione
[lambdaopt, Xi,biginds,smallinds] = SINDY_CV(ThetaTrain,
   ThetaVal, yTrain,yVal,Lambda);
```

```
[Xi_no_lambda,biginds_no_lambda,smallinds_no_lambda,Xi_hLS] =
   SINDY_CV_no_lambda(ThetaTrain, ThetaVal, yTrain,yVal);
[lambaopt_gcv, Xi_gcv,biginds_gcv,smallinds_gcv] = SINDY_GCV(
   Theta, y, Lambda);
[Xi_gcv_no_lambda,biginds_gcv_no_lambda,smallinds_gcv_no_lambda
   ,Xi_LS] = SINDY_GCV_no_lambda(Theta, y);
[lambaopt_sure, Xi_sure,biginds_sure,smallinds_sure,err] =
   SINDY_SURE(Theta, y, Lambda,var_y);
[Xi_SURE_no_lambda,biginds_SURE_no_lambda,
   smallinds_SURE_no_lambda,Xi_LS] = SINDY_SURE_no_lambda(Theta
   , y,var_y);

sprintf("lambaopt_cv = %", lambdaopt);
sprintf("lambaopt_gcv = %", lambaopt_gcv);
sprintf("lambaopt_sure = %", lambaopt_sure);




% Generata nomi variabili
numColumns = size(Xi, 2);
variableNames = [];
for i = 1:numColumns
    variableNames{i} = sprintf('stima_coefficienti x_%d', i);
end


Xi_CV_Table = array2table(Xi,'VariableNames',variableNames,"
   RowNames",column_labels);
Xi_CV_no_lambda_Table = array2table(Xi_no_lambda,'VariableNames
   ',variableNames,"RowNames",column_labels);
Xi_hLS_Table = array2table(Xi_hLS,'VariableNames',variableNames
   ,"RowNames",column_labels);
Xi_GCV_Table = array2table(Xi_gcv,'VariableNames',variableNames
   ,"RowNames",column_labels);
```

```
86 Xi_GCV_no_lambda_Table = array2table(Xi_gcv_no_lambda,'
   VariableNames',variableNames,"RowNames",column_labels);
87 Xi_LS_Table = array2table(Xi_LS,'VariableNames',variableNames,"
   RowNames",column_labels);
88 Xi_SURE_Table = array2table(Xi_sure,'VariableNames',
   variableNames,"RowNames",column_labels);
89 Xi_SURE_no_lambda_Table = array2table(Xi_SURE_no_lambda,'
   VariableNames',variableNames,"RowNames",column_labels);
90 Xi_LS_Table = array2table(Xi_LS,'VariableNames',variableNames,"
   RowNames",column_labels);
91
92 %stima con tutti i dati e lambda e ottimo
93 [XiAll,bigindsAll,smallindsAll] = sparse_representation(Theta,
   y, lambdaopt);
94 XiTableAll = array2table(XiAll,'VariableNames',variableNames,"
   RowNames",column_labels);
```

A.0.3 Funzione per generare dati del sistema di Lorenz

```
1  function [y, x, t, dxdt, snr] = generate_lorenz_dataset(
   initial_condition, t0, tmax, sigma, rho, beta, noise_level,
   num_samples)
2
3  options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);
4  [t, x] = ode45(@(t, x) lorenz(t, x, sigma, rho, beta), [t0,
   tmax], initial_condition, options);
5
6  % Downsample
7  x = x(1:floor(end/num_samples):end, :);
8  t = t(1:floor(end/num_samples):end);
9
10 dxdt = zeros(size(x));
11 dxdt = [sigma*(x(:, 2) - x(:, 1)), rho * x(:, 1) - x(:, 2) - x
   (:, 1) .* x(:, 3), x(:, 1) .* x(:, 2) - beta * x(:, 3)];
12
13 % Genera rumore
```

```
noise = randn(size(dxdt));

% aggiunta rumore
y = dxdt + noise_level * noise;

% SNR
signal_power = norm(dxdt, 'fro')^2;
noise_power = norm(noise_level * noise, 'fro')^2;
snr = 10 * log10(signal_power / noise_power);

end
```

A.0.4 Funzione per generare i mononiali

```
function [train_idx, val_idx,monomials,column_labels] =
   get_regression_matrix(x, r)
%Calcolo monimiali del vettore x di grado r.
% Vengono restituti gli indici di training e validation e le
   label delle
% colonne per capire quale monomiale  sta in una certa colonna



% Dimensione dello spazio di stato
n = size(x, 2);
t = size(x, 1);

monomials = ones(t,nchoosek(n+r,r));
column_labels = cell(1, nchoosek(n+r,r));

degree=r;

% Iterate over all the possible exponents of the variables.
n1 = 0;
n2 = degree;
m = n;
```

```
21
22
23
24 exp = zeros(1,m);
25 exp(1,m) = n1;
26 i = 1;
27
28 while ( true )
29
30     for k=1:n
31         monomials(:,i) = monomials(:,i).*x(:,k).^exp(k);
32          column_label = sprintf('%d', exp);
33         % Add the string label to the cell array.
34         column_labels{i} = column_label;
35     end
36
37     if ( exp(1) == n2 )
38       break
39     end
40
41     exp = mono_between_next_grlex ( m, n1, n2, exp );
42     i = i + 1;
43
44 end
45
46
47
48 %divido dataset in traing e validation
49 [train_idx, val_idx] = dividerand(size(monomials, 1), 0.5, 0.5)
      ;
50
51
52 end
```

A.0.5 Funzione che genera la sequenza dei monomi successivi in grevlex

```
function x = mono_between_next_grevlex ( m, n1, n2, x )

%**************************************************80
%
%% mono_between_next_grevlex(): grevlex next monomial, degree
   between N1 and N2.
%
%  Discussion:
%
%    We consider all monomials in an M dimensional space, with
   total
%    degree N between N1 and N2, inclusive.
%
%    For example:
%
%    M = 3
%    N1 = 2
%    N2 = 3
%
%    #  X(1)  X(2)  X(3)  Degree
%      +-------------------------
%    1 |  0     0     2        2
%    2 |  0     1     1        2
%    3 |  1     0     1        2
%    4 |  0     2     0        2
%    5 |  1     1     0        2
%    6 |  2     0     0        2
%      |
%    7 |  0     0     3        3
%    8 |  0     1     2        3
%    9 |  1     0     2        3
%   10 |  0     2     1        3
%   11 |  1     1     1        3
%   12 |  2     0     1        3
%   13 |  0     3     0        3
%   14 |  1     2     0        3
```

```
%   15 |  2     1     0        3
%   16 |  3     0     0        3
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    09 September 2014
%
%  Author:
%
%    John Burkardt
%
%  Input:
%
%    integer M, the spatial dimension.
%
%    integer N1, N2, the minimum and maximum degrees.
%    0 <= N1 <= N2.
%
%    integer X(M), the current monomial.
%    To start the sequence, set X = [ 0, 0, ..., 0, N1 ].
%
%  Output:
%
%    integer X(M), the next monomial.
%    The last value in the sequence is X = [ N2, 0, ..., 0, 0
   ].
%
  if ( n1 < 0 )
    fprintf ( 1, '\n' );
    fprintf ( 1, 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!\n' )
       ;
    fprintf ( 1, '  N1 < 0.\n' );
```

```
69      error ( 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!' );
70    end
71
72    if ( n2 < n1 )
73      fprintf ( 1, '\n' );
74      fprintf ( 1, 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!\n' )
          ;
75      fprintf ( 1, '  N2 < N1.\n' );
76      error ( 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!' );
77    end
78
79    if ( sum ( x(1:m) ) < n1 )
80      fprintf ( 1, '\n' );
81      fprintf ( 1, 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!\n' )
          ;
82      fprintf ( 1, '  Input X sums to less than N1.\n' );
83      error ( 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!' );
84    end
85
86    if ( n2 < sum ( x(1:m) ) )
87      fprintf ( 1, '\n' );
88      fprintf ( 1, 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!\n' )
          ;
89      fprintf ( 1, '  Input X sums to more than N2.\n' );
90      error ( 'MONO_BETWEEN_NEXT_GREVLEX - Fatal error!' );
91    end
92
93    if ( n2 == 0 )
94      return
95    end
96
97    if ( x(1) == n2 )
98      x(1) = 0;
99      x(m) = n1;
100   else
101     x = mono_next_grevlex ( m, x );
```

```
  end

  return
end
```

A.0.6 Funzione per calcolo del prossimo monomio dell'ordine grevlex

```
function x = mono_next_grevlex ( m, x )

%*****************************************************80
%
%% mono_next_grevlex(): grevlex next monomial.
%
%  Discussion:
%
%    For example:
%
%    M = 3
%
%    #  X(1)  X(2)  X(3)  Degree
%      +----------------------
%    1 |  0     0     0        0
%      |
%    2 |  0     0     1        1
%    3 |  0     1     0        1
%    4 |  1     0     0        1
%      |
%    5 |  0     0     2        2
%    6 |  0     1     1        2
%    7 |  1     0     1        2
%    8 |  0     2     0        2
%    9 |  1     1     0        2
%   10 |  2     0     0        2
%      |
%   11 |  0     0     3        3
%   12 |  0     1     2        3
```

```
%   13 |  1     0     2        3
%   14 |  0     2     1        3
%   15 |  1     1     1        3
%   16 |  2     0     1        3
%   17 |  0     3     0        3
%   18 |  1     2     0        3
%   19 |  2     1     0        3
%   20 |  3     0     0        3
%
%    Thanks to Stefan Klus for pointing out a discrepancy in a previous
%    version of this code, 05 February 2015.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    05 February 2015
%
%  Author:
%
%    John Burkardt
%
%  Input:
%
%    integer M, the spatial dimension.
%
%    integer X(M), the current monomial.
%    The first element in the sequence is X = [ 0, 0, ..., 0, 0 ].
%
%  Output:
%
%    integer X(M), the next monomial.
```

```
%
  if ( sum ( x(1:m) ) < 0 )
    fprintf ( 1, '\n' );
    fprintf ( 1, 'MONO_NEXT_GREVLEX - Fatal error!\n' );
    fprintf ( 1, '  Input X sums to less than 0.\n' );
    error ( 'MONO_NEXT_GREVLEX - Fatal error!' );
  end
%
%  Seek the first index 1 < I for which 0 < X(I).
%
  j = 1;

  for i = 2 : m
    if ( 0 < x(i) )
      j = i;
      break
    end
  end

  if ( j == 1 )
    t = x(1);
    x(1) = 0;
    x(m) = t + 1;
  elseif ( j < m )
    x(j) = x(j) - 1;
    t = x(1) + 1;
    x(1) = 0;
    x(j-1) = x(j-1) + t;
  elseif ( j == m )
    t = x(1);
    x(1) = 0;
    x(j-1) = t + 1;
    x(j) = x(j) - 1;
  end

  return
```

```
end
```

A.0.7 Funzione SINDy con CV

```
function [lambdaopt, Xi,biginds,smallinds] = SINDY_CV(
   ThetaTrain, ThetaVal, yTrain,yVal,Lambda)
% Funzione che integra sparse_representation per i vari lambda
   generati e per
% il migliore ricalcola sparse_representation trovando Xi
    for i = 1:size(Lambda,2)
        [Xi,biginds,smallinds] = sparse_representation(
           ThetaTrain, yTrain, Lambda(i));
        err(i) = norm(yVal-ThetaVal*Xi);
    end
    [q,w] = min(err);
    lambdaopt = Lambda(w);
    [Xi,biginds,smallinds] = sparse_representation(ThetaTrain,
       yTrain, lambdaopt);

end
```

A.0.8 Funzione SINDy con GCV

```
function [lambdaopt, Xi,biginds,smallinds] = SINDY_GCV(Theta, y
   ,Lambda)
  for i = 1:size(Lambda,2)
    [Xi,biginds,smallinds] = sparse_representation(Theta, y,
       Lambda(i));
    err(i) = norm(y-Theta*Xi)^2 / (size(y,1) - size(Theta,2));
  end
  [q,w] = min(err);
  lambdaopt = Lambda(w);
  [Xi,biginds,smallinds] = sparse_representation(Theta, y,
     lambdaopt);

end
```

A.0.9 Funzione SINDy con SURE

```
function [lambdaopt, Xi,biginds,smallinds, q,w] = SINDY_SURE(
   Theta, y, Lambda,var_y)
sigma = var_y;
    for i = 1:size(Lambda,2)
        [Xi,biginds,smallinds] = sparse_representation(Theta, y
           , Lambda(i));
        err(i) = norm(y-Theta*Xi)^2 + 2*sigma^2*size(Theta,2);
    end
    [q,w] = min(err);
    lambdaopt = Lambda(w);
    [Xi,biginds,smallinds] = sparse_representation(Theta, y,
       lambdaopt);

end
```

A.0.10 Funzione sparse_representation senza λ con CV

```
function [Xi,biginds,smallinds,Xi0] =
   sparse_representation_CV_no_lambda(Theta, dXdt, ThetaVal,
   yVal)
% funzione che restitusce sparse Xi passando Theta, i monomiali
    di
% training, Y
n = size(dXdt,2);
m = size(Theta,2);

Xi = Theta\dXdt;
Xi0 = Xi;
    for k=1:m*n
        mask1 = (abs(Xi)==0); % coefficienti già scartati
        % coefficienti da scartare
        minAbsValue = min(abs(Xi(Xi~=0)));
```

```
13
14          % maschera per coefficienti da scartare
15          mask2 = abs(Xi) == minAbsValue;
16
17          smallinds = mask1 | mask2;
18          Xi(smallinds)=0; %
19          for ind = 1:n
20              biginds = ~smallinds(:,ind);
21              Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
22          end
23          %calcolo errore con cv
24          err(k) = norm(yVal-ThetaVal*Xi);
25
26      end
27      Xi = Xi0;
28      [q,w] = min(err);
29      %si itera nuovamente fino all'indice che minimizza l'errore
30       for k=1:w
31          mask1 = (abs(Xi)==0);
32          minAbsValue = min(abs(Xi(Xi~=0)));
33          mask2 = abs(Xi) == minAbsValue;
34          smallinds = mask1 | mask2;
35          Xi(smallinds)=0;
36          for ind = 1:n
37              biginds = ~smallinds(:,ind);
38              Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
39          end
40
41      end
42
43  end
```

A.0.11 Funzione sparse_representation senza λ con GCV

```
1  function [Xi,biginds,smallinds,Xi0] =
       sparse_representation_GCV_no_lambda(Theta, dXdt)
```

```
% funzione che restitusce sparse Xi passando Theta, i monomiali di
% training, Y
n = size(dXdt,2);
m = size(Theta,2);
Xi = Theta\dXdt;
Xi0 = Xi;
    for k=1:m*n
        mask1 = (abs(Xi)==0);
        minAbsValue = min(abs(Xi(Xi~=0)));

        mask2 = abs(Xi) == minAbsValue;

        smallinds = mask1 | mask2;
        Xi(smallinds)=0;
        for ind = 1:n
            biginds = ~smallinds(:,ind);

            Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
        end
        %calcolo errore GCV
        err(k) = norm(dXdt-Theta*Xi)^2 / (size(dXdt,1) - size(
           Theta,2));

    end
    Xi = Xi0;
    [q,w] = min(err);

   for k=1:w
        mask1 = (abs(Xi)==0);
        minAbsValue = min(abs(Xi(Xi~=0)));
        mask2 = abs(Xi) == minAbsValue;

        smallinds = mask1 | mask2;
        Xi(smallinds)=0;
        for ind = 1:n
```

```
            biginds = ~smallinds(:,ind);
            Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
        end

    end

end
```

A.0.12 Funzione sparse_representation senza λ con SURE

```
function [Xi,biginds,smallinds,Xi0] =
   sparse_representation_SURE_no_lambda(Theta, dXdt,sigma)
% funzione che restitusce sparse Xi passando Theta, i monomiali
    di
% training, Y
n = size(dXdt,2);
m = size(Theta,2);
Xi = Theta\dXdt;
Xi0 = Xi;
    for k=1:m*n
        mask1 = (abs(Xi)==0);
        minAbsValue = min(abs(Xi(Xi~=0)));
        mask2 = abs(Xi) == minAbsValue;

        smallinds = mask1 | mask2;
        Xi(smallinds)=0;
        for ind = 1:n
            biginds = ~smallinds(:,ind);
            Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
        end
        %calcolo errore con SURE
        err(k) = norm(dXdt-Theta*Xi)^2 + 2*sigma^2*size(Theta
           ,2);

    end
    Xi = Xi0;
```

```
    [q,w] = min(err);

     for k=1:w
        mask1 = (abs(Xi)==0);
        minAbsValue = min(abs(Xi(Xi~=0)));
        mask2 = abs(Xi) == minAbsValue;

        smallinds = mask1 | mask2;
        Xi(smallinds)=0;
        for ind = 1:n
            biginds = ~smallinds(:,ind);
            Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
        end

    end

end
```

A.0.13 Simulazione con risposta impulsiva

```
close all
clear all
clc
numcampioni =500;
numcolonnetheta = 100;
coefflpf = -0.99999;

% rumore
noise = randn(numcampioni,1);

% filtro passa basso 1/(1 - 0.99z^-1)
u = filter([0 1], [1 coefflpf] , noise);

ingresso = zeros(size(u));

ingresso(1)=1;
```

```
17
18 Xi = filter([0 1 2 1], [1 -1.5 0.56 0] , ingresso);
19
20
21 theta = toeplitz(u);
22 theta = theta(:,1:numcolonnetheta);
23 %per normalizzare dimensioni Xi
24 Ti = toeplitz(ingresso);
25 Ti = Ti(:,1:numcolonnetheta);
26
27 Lambda = logspace(-7, 5 ,30);
28
29 y = filter([0 1 2 1], [1 -1.5 0.56 0] , u);
30
31 %calcolo varianza Y per SURE
32 S = lscov(theta,y);
33 Ta = norm(y-theta*S)^2 / ((length(y) - length(S)));
34 var_y = sqrt(Ta);
35
36 %inzio esecuzione sindy nei vari scenari
37 [lambaopt_sure, Xi_sure,biginds_sure,smallinds_sure, q,w] =
      SINDY_SURE(theta, y, Lambda,var_y);
38 [lambaopt_GCV, Xi_GCV,biginds_GCV,smallinds_GCV] = SINDY_GCV(
      theta, y, Lambda);
39 [lambdaopt_CV, Xi_CV,biginds_CV,smallinds_CV] = SINDY_CV(theta
      (1:numcampioni/2,:), theta(numcampioni/2+1:numcampioni,:), y
      (1:numcampioni/2,:),y(numcampioni/2 +1 :numcampioni,:),
      Lambda);
40 [Xi_CV_no_lambda,biginds_CV__no_lambda,smallinds_no_lambda,
      Xi_LS] = SINDY_CV_no_lambda(theta(1:numcampioni/2,:), theta(
      numcampioni/2+1:numcampioni,:), y(1:numcampioni/2,:),y(
      numcampioni/2 +1 :numcampioni,:));
41 [Xi_GCV_no_lambda,biginds_GCV_no_lambda,smallinds_GCV_no_lambda
      ] = SINDY_GCV_no_lambda(theta, y);
42 [Xi_SURE_no_lambda,biginds_SURE_no_lambda,
      smallinds_SURE_no_lambda] = SINDY_SURE_no_lambda(theta, y,
```

```
   var_y);

%plot

figure(1)
plot(noise)
hold on
plot(u)
xlim([0 numcolonnetheta])
legend('Input','Output')

figure(2)
plot(Xi)
hold on
plot(Xi_sure)
hold on
plot(Xi_CV)
hold on
plot(Xi_CV_no_lambda)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{SURE}','\Xi_{CV}','\Xi_{CV\_no\_\lambda}')

figure(3)
plot(Xi)
hold on
plot(Xi_sure)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{SURE}')

figure(4)
plot(Xi)
hold on
plot(Xi_CV)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{CV}')

```

```
figure(5)
plot(Xi)
hold on
plot(Xi_CV_no_lambda)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{CV\_no\_\lambda}')

figure(6)
plot(Xi)
hold on
plot(Xi_GCV)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{GCV}')

figure(7)
plot(Xi)
hold on
plot(Xi_GCV_no_lambda)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{GCV\_no\_\lambda}')

figure(8)
plot(Xi)
hold on
plot(Xi_SURE_no_lambda)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{SURE\_no\_\lambda}')

figure(9)
plot(Xi)
hold on
plot(Xi_LS)
xlim([0 numcolonnetheta])
legend('\Xi','\Xi_{LS}')

%calcolo fit
```

```
fit_cv_no_lambda = 100*(1-norm(Xi-Ti*Xi_CV_no_lambda)/norm(Xi))
    ;
fit_cv = 100*(1-norm(Xi-Ti*Xi_CV)/norm(Xi));
fit_ls = 100*(1-norm(Xi-Ti*Xi_LS)/norm(Xi));
fit_gcv_no_lambda = 100*(1-norm(Xi-Ti*Xi_GCV_no_lambda)/norm(Xi
    ));
fit_gcv = 100*(1-norm(Xi-Ti*Xi_GCV)/norm(Xi));
fit_sure_no_lambda = 100*(1-norm(Xi-Ti*Xi_SURE_no_lambda)/norm(
    Xi));
fit_sure = 100*(1-norm(Xi-Ti*Xi_sure)/norm(Xi));
```

Bibliografia

[1] A. C. Davison, *Statistical Models* (Cambridge Series in Statistical and Probabilistic Mathematics). Cambridge University Press, 2003.

[2] G. Picci e G. Picci, *Filtraggio statistico : Wiener, Levinson, Kalman e applicazioni / Giorgio Picci*, ita. Padova: Libreria Progetto, 2002, isbn: 97888733188X.

[3] T. J. Hastie, R. Tibshirani, J. Friedman e T. J. Hastie, *The elements of statistical learning : data mining, inference, and prediction / Trevor Hastie, Robert Tibshirani, Jerome Friedman* (Springer series in statistics), eng. New York [etc: Springer, 2001, isbn: 0387952845.

[4] S. Shalev-Shwartz, S. Ben-David e S. Shalev-Shwartz, *Understanding machine learning [risorsa elettronica] : from theory to algorithms / Shai Shalev-Shwartz, Shai Ben-David*, eng, Cambridge, 2014.

[5] K. P. Murphy e K. P. Murphy, *Machine learning [risorsa elettronica] : a probabilistic perspective / Kevin P. Murphy*, eng, Cambridge, 2012.

[6] F. Ettore, *Appunti di teoria dei sistemi / E. Fornasini*, ita. Padova: Progetto, 2014, isbn: 978-88-96477-32-8.

[7] M. Bisiacco, G. Pillonetto e M. Bisiacco, *Sistemi e modelli / Mauro Bisiacco, Gianluigi Pillonetto*, ita, Ristampa corretta. Bologna: Esculapio, 2015, isbn: 9788874888023.

[8] E. L. F. A. Charnes e P. L. Yu, «The Equivalence of Generalized Least Squares and Maximum Likelihood Estimates in the Exponential Family,» *Journal of the American Statistical Association*, vol. 71, n. 353, pp. 169–171, 1976. doi: 10.1080/01621459.1976.10481508. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1976.10481508. indirizzo: https://www.tandfonline.com/doi/abs/10.1080/01621459.1976.10481508.

[9] P. J. Bickel, K. A. Doksum e P. J. Bickel, *Mathematical statistics : basic ideas and selected topics / Peter J. Bickel, Kjell A. Doksum* (Holden-Day series in probability and statistics), eng. Oakland: Holden-Day, 1977, isbn: 0816207844.

[10] V. G. Voinov, M. S. Nikulin e V. G. Voinov, *Unbiased estimators and their applications / by V. G. Voinov and M. S. Nikulin*, eng. Dordrecht [etc: Kluwer, 1993.

[11] J. Pfanzagl, R. Hamboker, J. Pfanzagl e R. Hamboker, *Parametric statistical theory / Johann Pfanzagl ; with the assistance of R. Hamböker* (De Gruyter textbook), eng. Berlin New York: Walter de Gruyter, 1994, isbn: 3110140306.

[12] E. L. Lehmann e E. L. Lehmann, *Elements of large-sample theory / E. L. Lehmann* (Springer texts in statistics), eng. New York [etc: Springer, 1999, isbn: 0387985956.

[13] R. R. R. R. Hocking, *Methods and applications of linear models regression and the analysis of variance* (Wiley series in probability and statistics), eng, 2nd ed. Hoboken, N.J: Wiley-Interscience, 2003, isbn: 1-280-27269-4.

[14] C. Agostinelli e C. Agostinelli, *Robust model selection by cross-validation via weighted likelihood methodology / C. Agostinelli* (Redazioni provvisorie), eng. Padova Università degli studi: Dip. di Scienze Statistiche, 1999.

[15] S. M. Piryonesi e T. E. El-Diraby, «Data Analytics in Asset Management: Cost-Effective Prediction of the Pavement Condition Index,» *Journal of Infrastructure Systems*, vol. 26, p. 04 019 036, 2020. indirizzo: https://api.semanticscholar.org/CorpusID:213782055.

[16] D. M. Allen, «The Relationship between Variable Selection and Data Agumentation and a Method for Prediction,» *Technometrics*, vol. 16, n. 1, pp. 125–127, 1974, issn: 00401706. indirizzo: http://www.jstor.org/stable/1267500 (visitato il 08/05/2024).

[17] M. Stone, «Cross-Validatory Choice and Assessment of Statistical Predictions,» *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, n. 2, pp. 111–133, dic. 2018, issn: 0035-9246. doi: 10.1111/j.2517-6161.1974.tb00994.x. eprint: https://academic.oup.com/jrsssb/article-pdf/36/2/111/49096683/jrsssb_36_2_111.pdf. indirizzo: https://doi.org/10.1111/j.2517-6161.1974.tb00994.x.

[18] G. Golub, M. Heath e G. Wahba, «Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter,» *Technometrics*, vol. 21, pp. 215–223, mag. 1979. doi: 10.1080/00401706.1979.10489751.

[19] C. M. Stein, «Estimation of the Mean of a Multivariate Normal Distribution,» *The Annals of Statistics*, vol. 9, n. 6, pp. 1135 –1151, 1981. doi: 10.1214/aos/1176345632. indirizzo: https://doi.org/10.1214/aos/1176345632.

[20] S. L. Brunton, J. L. Proctor e J. N. Kutz, «Discovering governing equations from data by sparse identification of nonlinear dynamical systems,» *Proceedings of the National Academy of Sciences of the United States of America*, vol. 113, n. 15, pp. 3932–3937, 2016, issn: 00278424, 10916490. indirizzo: https://www.jstor.org/stable/26469234.

[21] J. Bongard e H. Lipson, «Automated reverse engineering of nonlinear dynamical systems,» *Proceedings of the National Academy of Sciences*, vol. 104, n. 24, pp. 9943–9948, 2007.

[22] S. L. Brunton, J. H. Tu, I. Bright e J. N. Kutz, «Compressive sensing and low-rank libraries for classification of bifurcation regimes in nonlinear dynamical systems,» *SIAM Journal on Applied Dynamical Systems*, vol. 13, n. 4, pp. 1716–1732, 2014.

[23] E. J. Candès, J. Romberg e T. Tao, «Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information,» *IEEE Transactions on Information Theory*, vol. 52, n. 2, pp. 489–509, 2006.

[24] R. Chartrand, «Numerical differentiation of noisy, nonsmooth data,» *ISRN Applied Mathematics*, vol. 2011, 2011.

[25] R. Tibshirani, «Regression Shrinkage and Selection Via the Lasso,» *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, n. 1, pp. 267–288, dic. 2018, issn: 0035-9246. doi: 10.1111/j.2517-6161.1996.tb02080.x. eprint: https://academic.oup.com/jrsssb/article-pdf/58/1/267/49098631/jrsssb_58_1_267.pdf. indirizzo: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

[26] L. Zhang e H. Schaeffer, «On the Convergence of the SINDy Algorithm,» *Multiscale Modeling & Simulation*, vol. 17, n. 3, pp. 948–972, 2019. doi: 10.1137/18M1189828. eprint: https://doi.org/10.1137/18M1189828. indirizzo: https://doi.org/10.1137/18M1189828.

[27] E. N. Lorenz, «Deterministic Nonperiodic Flow,» *Journal of Atmospheric Sciences*, vol. 20, n. 2, pp. 130 –141, 1963. doi: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2. indirizzo: https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml.

[28] D. A. Cox, *Ideals, Varieties, and Algorithms* (Undergraduate Texts in Mathematics). Springer, 2007, isbn: 978-0-387-21346-7.

[29] A. C. G. D. N. L. L. Gianluigi Pillonetto Tianshi Chen, *Regularized System Identification*. 2022.

[30] A. V. Oppenheim, A. S. Willsky, S. H. Nawab, A. V. Oppenheim e S. H. Nawab, *Signals systems / Alan V. Oppenheim, Alan S. Willsky ; with S. Hamid Nawab* (Prentice-Hall signal processing series), eng, 2. ed. Upper Saddle River: Prentice-Hall, 1997, isbn: 0136511759.

[31] L. R. Rabiner, B. Gold e L. R. Rabiner, *Theory and application of digital signal processing / Lawrence R. Rabiner, Bernard Gold*, eng. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975, isbn: 0139141014.

[32] R. M. Gray, *Toeplitz and Circulant Matrices: A Review*. 2006. doi: 10 . 1561 / 0100000006.