

**UNIVERSITà** **DEGLI STUDI DI MESSINA**

**DIPARTIMENTO DI INGEGNERIA**

CORSO DI LAUREA TRIENNALEIN

INGEGNERIA ELETTRONICA E INFORMATICA

**TITOLO**

Applicazione delle tecniche DMD e SINDy

per la creazione di modelli di ordine ridotto

Tesi di laurea di:

**Antonino Biancuzzo**

Relatore:

**Chiar.mo Prof. Luca Patanè**

**Anno Accademico 2021/2022**

**Indice**

[1 Introduzione 3](#_Toc109463127)

[2 Richiami della teoria del DMD e del SINDy 4](#_Toc109463128)

[2.1 Dynamic Mode Decomposition 4](#_Toc109463129)

[2.1.1 Versioni del DMD e campi d’utilizzo 7](#_Toc109463130)

[2.1.2 Architettura 9](#_Toc109463131)

[2.1.3 Algoritmo 12](#_Toc109463132)

[2.1.4 DMDc – Dynamic Mode Decomposition con controllo 16](#_Toc109463133)

[2.1.5 DMD con la matrice di Hankel 20](#_Toc109463134)

[2.2 SINDy 24](#_Toc109463135)

[2.2.1 Sistema tempo continuo 24](#_Toc109463136)

[2.2.2 Sistema tempo discreto 25](#_Toc109463137)

[2.2.3 Algoritmo 25](#_Toc109463138)

[2.2.4 SYNDy con il controllo (SYNDy-MPC) 31](#_Toc109463139)

[2.3 Sistema di Lorenz 34](#_Toc109463140)

[2.3.1 Sistema di Lorenz di base 34](#_Toc109463141)

[2.3.2 Sistema di Lorenz con forzante 35](#_Toc109463142)

[3 DMD: casi di studio 37](#_Toc109463143)

[3.1 DMD nativo con matrice d’ingresso diagonale 37](#_Toc109463144)

[3.2 Hankel DMD con matrice d’ingresso diagonale 40](#_Toc109463145)

[3.3 Hankel DMD con sistema di Lorenz 43](#_Toc109463146)

[3.4 DMDc con matrice d’ingresso diagonale 46](#_Toc109463147)

[3.5 Hankel DMDc con matrice d’ingresso diagonale 50](#_Toc109463148)

[3.6 Hankel DMDc con sistema di Lorenz 53](#_Toc109463149)

[3.7 DMDc con processo industriale 57](#_Toc109463150)

[3.8 Hankel DMDc con processo industriale 60](#_Toc109463151)

[4 SINDy: casi di studio 66](#_Toc109463152)

[4.1 SINDy nativo con sistema di Lorenz 66](#_Toc109463153)

[4.2 SINDy con controllo con sistema di Lorenz 69](#_Toc109463154)

[5 Conclusioni 77](#_Toc109463155)

[6 Bibliografia 79](#_Toc109463156)

# Introduzione

Negli ultimi anni l’interesse nei confronti dei metodi data-driven è aumentato sempre più e sta portando ad un cambiamento nella visione all’interno del mondo della scienza ingegneristica. Infatti, con i sistemi data-driven si ha una disponibilità senza precedenti di misurazioni ad alta fedeltà provenienti da serie temporali, simulazioni numeriche e dati sperimentali. Quando i dati sono utilizzati insieme ad algoritmi efficienti e tecniche di machine learning, si possono estrarre dei modelli spaziotemporali. Un’importante implicazione che deriva da queste tecniche di modellizzazione data-driven è che si può ottenere un’ampia conoscenza dei processi fondamentali e anche migliorare la capacità nella predizione, stima e controllo di sistemi complessi. L’abilità nell’ottenere tutta questa conoscenza dai dati è stata chiamata *quarto paradigma della scoperta scientifica* [8].

Il primo punto su cui si focalizza questa tesi è il metodo del Dynamic Mode Decomposition (DMD). DMD è una tecnica di decomposizione matriciale altamente versatile e si basa sulla potenza della Singular Value Decomposition (SVD). È un potente strumento di analisi che permette di ottenere le strutture spaziali coerenti associate con le componenti spettrali temporali [5], quindi è molto utile nella ricostruzione di sistemi complessi. Questa tecnica può anche essere usata per predire le serie temporali.

Il secondo punto su cui si focalizza l’elaborato di tesi è l’algoritmo Sparse Identification of Nonlinear Dynamics (SINDy), che combina le tecnologie sparse lifting e machine learning con le dinamiche non lineari. Possiamo vederlo come un’evoluzione del DMD nel caso in cui non si abbiano abbastanza dati e relazioni tra i dati per ricavarne un modello accettabile. Si applica il metodo Sequential Threshold Least Squares (STLS), algoritmo in grado di creare un’enorme insieme di regressori e selezionare quelli più importanti in base al valore dato dalla soglia preimpostata, alla funzione libreria, composta da un insieme di funzioni candidate, per approssimare l’equazione dinamica del sistema non lineare in esame.

L’algoritmo prevede l’utilizzo della Singular Value Decomposition (SVD) e in questo modo si trovano le derivate parziali. Si può applicare il SINDy a dati con rumore, dati da sistemi stocastici o dati da sistemi ad alte dimensioni.

Con dati campionati con una frequenza sufficiente, anche se molto rumorosi, si possono spesso usare delle tecniche di de-noising per calcolare le derivate parziali.

Per quanto riguarda la funzione libreria, si può sfruttare la simmetria e la fisica per creare la libreria che si preferisce e lo si può fare anche con il controllo (SINDy with control).

# Richiami della teoria del DMD e del SINDy

Al giorno d’oggi ci sono due direzioni popolari nella ricerca dei metodi di modellizzazione dinamica guidata dai dati:

* Equation free modeling reserch, che è rappresentata dalla tecnologia di decomposizione modale. In questo caso i dati sono usati per invertire i modelli a basso ordine per ottenere le caratteristiche dinamiche a lungo termine del sistema. Il metodo di ricerca principale è la Modal Decomposition Technology, la quale decompone i dati originati in modalità sovrapposte con caratteristiche dinamiche diverse, ma ha delle lacune per quanto riguarda l’interpretabilità fisica. Alcune di queste tecniche includono: Empirical Mode Decomposition (EDM), Variatonal Mode Decompositon (VDM), Proper Orthogonal Decomposition (POD) e Dynamic Mode Decomposition (DMD), ovviamente con le varie estensioni.
* Lo studio della ricostruzione di equazioni dinamiche dai dati tramite l’identificazione di sistemi dinamici non lineari. Questo studio si basa sull’uso di metodi di intelligenza artificiale come il machine learning o il deep learning.

## Dynamic Mode Decomposition

È un metodo di regressione che viene utilizzato per ricavare il sistema dinamico lineare più adatto dai dati misurati nel tempo, anche se le dinamiche sono non lineari.

La modellizzazione data-driven e il controllo dei sistemi complessi è un campo in rapida evoluzione con grande potenziale in ambito ingegneristico, biologico e fisico [8]. I moderni sistemi di interesse, come un fluido turbolente o una rete di neuroni, possono essere caratterizzati come sistemi dinamici non lineari di alto rango sia nello spazio che nel tempo. Nonostante siano complessi, molti di questi sistemi evolvono in un attrattore dalle basse dimensioni che potrebbe essere caratterizzato da strutture spaziotemporali coerenti, per cui ne segue che la Dynamic Mode Decomposition (DMD) è una nuova e potente tecnica per la scoperta dei sistemi dinamici provenienti da dati ad alte dimensioni.

Il suo crescente successo deriva dal fatto che è un metodo equation-free e data-driven, in grado di fornire un’accurata decomposizione di un sistema complesso in strutture spaziotemporali coerenti che potrebbero essere usate per predire lo stato futuro nel breve periodo e per il controllo. Più in generale, il DMD ha guadagnato popolarità velocemente da quando è stata evidenziata la sua connessione con l’operatore di Koopman. Lo sviluppo del DMD è dovuto anche alla concorrente crescita della scienza dei dati, che comprende un ampio assortimento di tecniche, dal machine learning e la regressione statistica alla computer vision e compressed sensing. Un’ulteriore motivo per cui il DMD viene ampiamente utilizzato è dato dal fatto che permetta di catturare le componenti in frequenza che descrivono la crescita o il decadimento e le oscillazioni nei dati mentre riesce a predire il sistema durante lo stadio di ricostruzione.

Il DMD è basato unicamente sui dati raccolti come output da un sistema, del quale non si conosce analiticamente la dinamica, quindi è basato sul metodo Proper Orthogonal Decomposition (POD), che utilizza una tecnica computazionalmente efficiente: la Singular Value Decomposition (SVD) [2]. Il POD, in breve è un metodo numerico che estrae la struttura coerente del campo di flusso. Viene usata per costruire un metodo di calcolo per il modello di ordine ridotto e questo modello di ordine ridotto può mantenere un certo grado di accuratezza. In sostanza riduce il consumo computazionale dell’algoritmo. Una sua evoluzione è la Spectral Proper Orthogonal Decomposition (SPOD) e la sua funzionalità sta nel fatto che converte il dominio del tempo in quello della frequenza ed attua una decomposizione fino ad ottenere un serie di modi (autovettori) che mantengono l’ortogonalità spaziale e temporale.

Il DMD fornisce una decomposizione modale dove ogni modo (autovettore) è formato da strutture spazialmente correlate che hanno lo stesso comportamento lineare nel tempo. In questo modo, il DMD oltre a fornire una riduzione della dimensionalità dei modi del sistema, fornisce anche un modello che mostra come questi modi evolvono nel tempo.

I modi spaziali correlati sono anche associati con una data frequenza temporale.

I vantaggi di questo metodo sono che è molto semplice da eseguire e non fa quasi nessuna ipotesi sul sistema sottostante, mentre il costo dell’algoritmo è la Singular Value Decomposition (SVD) dalla matrice degli snapshot costruita dai dati xk.

In generale può essere visto come un algoritmo che svolge tre compiti fondamentali:

1. **Diagnostica**. L’algoritmo estrae la caratteristiche spaziotemporali di basso rango chiave di molti sistemi ad alte dimensioni, permettendo di ottenere risultati interpretabili fisicamente in termini di strutture spaziali e delle loro risposte temporali associate.
2. **Stima dello stato e predizione dello stato futuro**. Usa le strutture spaziotemporali che sono dominanti nei dati per costruire modelli dinamici dei processi osservati. È un compito difficile a causa del fatto che il DMD sia limitato nella costruzione del sistema dinamico lineare più adatto dal sistema dinamico non lineare che genera i dati. L’obiettivo, quindi, è anticipare lo stato del sistema in un regime dove non sono fatte misurazioni. Ci sono numerose strategie chiave, come ad esempio il campionamento intelligente dei dati, che permette al DMD di essere efficiente nella generazione di modelli dinamici lineari utili. Questo approccio della generazione del modello può eventualmente essere usata per la predizione dello stato futuro dei sistemi dinamici.
3. **Controllo**. Usando un modello dinamico lineare per predire il futuro di un sistema dinamico non lineare, è ragionevole aspettarsi che ci sia solo una finestra limitata, nel tempo, nel futuro, dove il modello approssimato e quello reale vanno effettivamente d’accordo. La speranza è che questa finestra di predizione accurata sia lunga abbastanza per permettere che il controllo sia in grado di influenzare lo stato futuro del sistema. Il DMD permette inoltre di ridurre i disturbi di rumore migliorando l’attenuazione del rumore.

Usando questa tecnica bisognerebbe essere sempre consapevoli dell’uso e dell’esito atteso. Inoltre, il successo dell’algoritmo dipenderà ampiamente su quale dei compiti sopra citati si sta cercando di ottenere un risultato.

### Versioni del DMD e campi d’utilizzo

* **Dynamic Mode Decomposition (DMD)**: è la versione base dell’algoritmo e fornisce una riduzione della dimensionalità dell’insieme degli autovettori del sistema e riesce a dare un modello per come questi modi dovrebbero evolvere nel tempo [2]. Proprio per il fatto che sia la base di tutti i successivi algoritmi, permette il proprio utilizzo in quasi tutte le successive condizioni, ma non con un’ottimizzazione eccellente in tutti i campi. Inizialmente è stato usato per studiare sistemi dinamici come la turbolenza di un fluido o sistemi epidemiologici.
* **DMD multi-resolution (mrDMD)**: viene applicato a sistemi dinamici altamente dimensionali e complessi che mostrano dinamiche multi-scala sia nel tempo che nello spazio con fenomeni transitori o intermittenti e tutto questo viene fatto grazie alla trasformata di Gabor. Il DMD di base non riesce a catturare accuratamente le dinamiche transitorie del sistema e, in questo caso, entra in gioco l’mrDMD, che è anche in grado di distinguere 2 o più oggetti in movimento ad una velocità diversa estraendo modi veloci e lenti nel caso di due soli oggetti [8]. Ha un’efficienza di esecuzione maggiore rispetto al DMD di base grazie al campionamento dei sottoinsiemi. È stato utilizzato nella predizione dei fenomeni climatici globali dato che riesce a scomporre le dinamiche in scale dei tempi differenti, isolando i transitori e i modelli intermittenti [1]. Utilizza pochi parametri, infatti, ad ogni livello di decomposizione, il numero dei dati campionati in ogni finestra di campionamento potrebbe essere un sottoinsieme dell’intero set di dati. Questo sottoinsieme di dati incrementa l’efficienza dell’esecuzione dell’algoritmo [8].
* **Compressed DMD**: questa versione permette di passare facilmente da un sistema con la totalità delle misurazioni ad uno con compressed measurements di più facile eseguibilità e si può, inoltre, ricostruire tutto il set di misurazioni tramite l’utilizzo del Compressed-Sensing DMD [2]. Al posto di collezionare dati altamente dimensionali solo per comprimerli e scartarne la maggior parte, questo algoritmo colleziona poche misurazioni compresse o random per poi dedurne la rappresentazione sparsa. Viene usato quando è possibile ricostruire il segnale completo con altra probabilità usando algoritmi convessi.
* **Exstended DMD (eDMD)**: è stato sviluppato per ridurre il costo della valutazione dell’operatore di Koopman quando il numero degli istanti di campionamento del sistema è molto più alto rispetto al numero di variabili [8] e anche perché il DMD di base usa delle misurazioni lineari del sistema che, generalmente, non sono ricche abbastanza da caratterizzare fenomeni non lineari. Quindi questo algoritmo cerca di ottenere il modello lineare del sistema anche tramite l’utilizzo di misurazioni non lineari [1,2]. È stato usato inizialmente per simulare le dinamiche molecolari con un’ampia separazione delle scale temporali e poi nell’analisi delle componenti indipendenti con ritardo temporale.
* **DMD with Hankel matrix**: il DMD estrae i modi dominanti presenti nei segnali misurati per predirne il comportamento futuro, ma questo è possibile solo se il set di segnali linearmente indipendente contenente i modi dominanti ha una cardinalità maggiore o uguale a questi modi, altrimenti il DMD restituisce un risultato errato [5]. In questo caso viene usata la tecnica che comporta l’utilizzo della matrice di Hankel che permette di aumentare artificialmente il numero di segnali misurati disponendoli all’interno di questa matrice. Si incrementano le dimensioni del sottospazio dei dati fino a quando non vengono contenuti i modi principali. Questo lo si fa usando delle coordinate ritardate di un istante temporale per ogni riga successiva della matrice e, così, si permette al DMD di predire lo stato anche dei sistemi con un piccolo numero di variabili o con dei dati non sufficienti. In [5] il DMD con Hankel viene usato per fare una predizione del numero di vendite di una catena di negozi nel lungo periodo ed è stato confrontato con altri algoritmi di predizione usati nello stesso campo: ANN1, ANN10 e ANN50 e si è notato che il DMD con questo approccio ha avuto prestazioni e risultati migliori rispetto a quest’ultimi. In [6] l’Hankel DMD viene usato per fare l’analisi spettrale della locomozione umana e si è derivata un’ottima approssimazione tra il sistema reale e il sistema approssimato dall’algoritmo, soprattutto nell’estrare le proprietà dinamiche delle strutture di coordinazione al fine di ottenere informazioni dinamiche con un significato, fisico come ad esempio l’angolo del piede durante una camminata. Il tutto permetterà di aumentare la chiarezza di questo algoritmo nel campo delle dinamiche sottostanti al modello studiato. Per decenni è stato usato nell’identificazione del sistema, soprattutto nell’ERA e nell’SSA, ma anche nel campo della neuroscienza; inoltre l’utilizzo del DMD con la matrice di Hankel è stato molto importante per sistemi con effetti di memoria a lungo termine [1].
* **DMD with control (DMDc)**: è l’algoritmo di base che, però, utilizza sia le misurazioni del sistema che un controllo applicato esternamente per scoprire le dinamiche sottostanti del sistema. Può anche estrarre le caratteristiche input-output, in modo da permettere la costruzione di un insieme di modelli dall’ordine ridotto che possono essere usati per predire e modellizzare controllori per sistemi complessi altamente dimensionali. Viene usato per sviluppare modelli per le previsioni future e strategie di controllo per guidare il sistema verso lo stato desiderato. È stato utilizzato al fine di ridurre gli sforzi durante le malattie infettive e all’interno dei sistemi di distribuzione [8].

### Architettura

Iniziamo col considerare dei dati collezionati da un sistema dinamico

|  |  |  |
| --- | --- | --- |
|  |  | (2.1) |

dove x(t) è un vettore che rappresenta lo stato del sistema dinamico al tempo *t,* *µ* contiene i parametri del sistema e *f(·)* rappresenta le dinamiche. Lo stato *x* è tipicamente abbastanza grande, con una dimensione n >> 1.

La dinamica tempo continuo può anche essere rappresentata nel tempo discreto

|  |  |  |
| --- | --- | --- |
|  | xk + 1 = F(xk) | (2.2) |

e in generale è non lineare, anche nel caso in cui F non sia nota esplicitamente. Questo viene fatto campionando il segnale continuo ogni istante di tempo . Il DMD ha l’obiettivo di approssimare la dinamica con un modello lineare tempo invariante:

|  |  |  |
| --- | --- | --- |
|  | xk + 1 = Axk | (2.3) |

attraverso uno studio dei dati prodotti dal modello tramite esperimenti o simulazioni.

La struttura del DMD è equation-free e le dinamiche potrebbero essere sconosciute, quindi le misurazioni dai dati del sistema sono usate per approssimare le dinamiche e predire lo stato futuro.

La ricostruzione del segnale originale nel tempo continuo si realizza tramite la seguente formula

|  |  |  |
| --- | --- | --- |
|  |  | (2.4) |

dove **b** sono i coefficienti delle condizioni iniziali x1 in modo tale da ottenere che , **Φ** sono i modi del sistema e **Λ** è la matrice diagonale degli autovalori discreti del sistema.

Il primo passo da fare è collezionare un numero di snapshots dello stato del sistema ad un intervallo di tempo dato . La matrice generata potrebbe essere del tipo:

|  |  |  |
| --- | --- | --- |
|  |  | (2.5) |

dove *n* rappresenta il numero delle variabili del sistema e *m* rappresenta il numero degli snapshots presi.

Dopo aver collezionato gli snapshots all’interno della matrice x, i dati vengono immagazzinati in due matrici X e X’:

|  |  |  |
| --- | --- | --- |
|  |  | (2.6) |
|  |  | (2.7) |

Assumendo che gli snapshots siano calcolati in *m* intervalli di tempo: all’interno di saranno presenti i primi m-1 snapshots della matrice x (*x1 – xm-1*), mentre all’interno di viene eseguito uno shift a destra e quindi sono presenti gli snapshots che vanno dal secondo all’ultimo di quelli presenti all’interno della matrice x (*x2* – *xm*), quindi questa matrice non è altro che la stessa ma shiftata di un istante di tempo in avanti.

Ricaviamo una relazione tra queste due matrici nel tempo e la dipendenza è data dalla matrice A, chiamata Matrice di Stato, contenente gli autovalori e gli autovettori del sistema in esame:

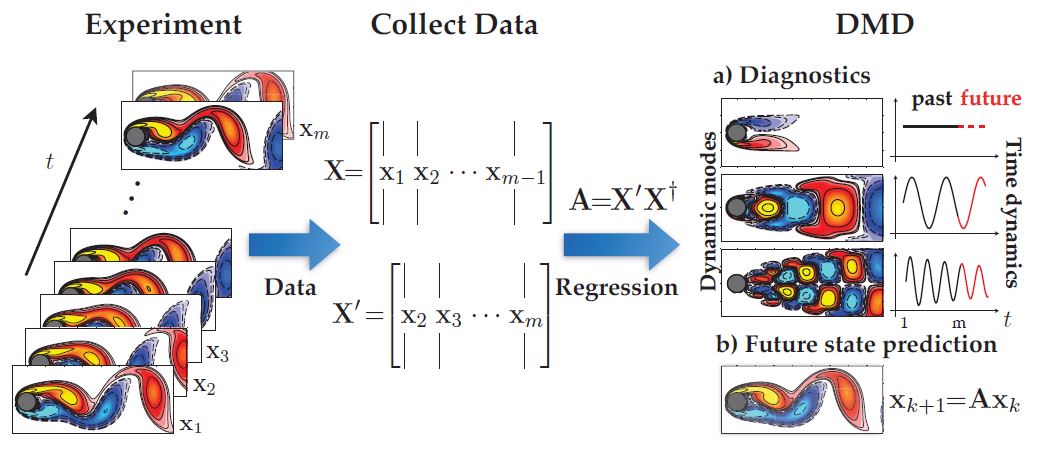
|  |  |  |
| --- | --- | --- |
|  |  | (2.8) |

Questa matrice rappresenta l’operatore lineare più adatto per ottenere un sistema dinamico lineare che indirizzi al meglio le misurazioni degli snapshots dei dati nel tempo [2]. La matrice di stato possiamo calcolarla tramite la seguente relazione:

|  |  |  |
| --- | --- | --- |
|  |  | (2.9) |

dove è la pseudoinversa di Monroe-Penrose.

Per sistemi molto grandi con vettore di stato x , bisogna considerare che la matrice A possiede n2 elementi e quindi potrebbe essere troppo grande e risultare poco pratica da calcolare; addirittura alcune volte la matrice potrebbe avere una dimensione così tanto elevata da rendere impossibile il suo immagazzinamento in memoria. Per questo è necessario fornire un’autodecomposizione a basso rango della matrice , caratteristica che il DMD è in grado di rispettare. Per ottenere questo risultato i dati misurati vengono proiettati in un sottospazio dal basso rango tramite la SVD e poi il calcolo continua con la matrice di basso rango . e possiedono gli stessi autovalori e autovettori, quindi il DMD può lavorare sulla matrice dal basso rango appena ricavata e ricostruirli senza calcolare esplicitamente l’operatore . Uno schematico del DMD è proposto nella Figura 2.1.



**Figura 2.1** Schematico del DMD [8]

DMD in breve: dopo che sono stati collezionati gli snapshots dei dati nelle matrici X (2.6) e X’ (2.7), il DMD esegue una regressione per trovare l’operatore lineare A tale che A . Successivamente i modi del sistema, cioè gli autovalori e gli autovettori, sono calcolati tramite la matrice di rango ridotto e da qui si può partire per ottenere il risultato richiesto dal DMD: la ricostruzione del sistema originale e/o la predizione futura dello stato e dei modi del sistema.

### Algoritmo

Come già discusso in precedenza, quando la il numero di variabili n è molto elevato, potrebbe essere impossibile calcolare e di conseguenza lavorare sulla matrice di stato . Invece il DMD riduce di rango la matrice utilizzata tramite la POD realizzando .

1. Calcolo della Singular Value Decomposition di X:

|  |  |  |
| --- | --- | --- |
|  |  | (2.10) |

dove , \* rappresenta la coniugata trasposta, , e . In questo caso r rappresenta il rango dell’approssimazione della SVD ridotta a *X* [8]*.*

Viene eseguita la SVD per ridurre il rango della matrice e viene calcolata con la specifica ‘econ’ che rappresenta una riduzione rispetto alla SVD nativa, la quale altrimenti potrebbe produrre matrici dal rango del milione per miliardi. Produce tre matrici: U contiene i vettori singolari di sinistra, è la matrice dei valori singolari e V sono i vettori singolari di destra. La riduzione del rango viene ultimata tramite l’utilizzo di un valore di troncamento r che rappresenta il numero di valori singolari utilizzati. Riducendo la dimensionalità del sistema da n (spesso milioni o miliardi) a r (decine o centinaia) si fornisce una predizione e una stima del sistema più veloce e a bassa latenza.

Il vantaggio dell’approccio con la SVD è che attenua il rumore/disturbo nei dati e compensa i problemi derivati dal troncamento numerico.

1. Calcolo dell’operatore lineare di rango ridotto

|  |  |  |
| --- | --- | --- |
|  |  | (2.11) |

dove , cioè è la proiezione della matrice completa A nei modi POD. L’osservazione chiave in questo passaggio è data dal fatto che abbia gli stessi autovalori non nulli della matrice , quindi lavorando con la matrice ridotta non abbiamo nessuna perdita rispetto a lavorare con la matrice di stato .

1. Calcolo della decomposizione spettrale di :

|  |  |  |
| --- | --- | --- |
|  |  | (2.12) |

In questo passaggio si determinano gli autovalori e gli autovettori del sistema. Le colonne di W sono autovettori e Λ è una matrice diagonale contente gli autovalori

1. Ricostruzione dei modi della matrice di stato A:

|  |  |  |
| --- | --- | --- |
|  |  | (2.13) |

Adesso che è stato completato il procedimento nativo del DMD, all’occorrenza, è possibile generare una predizione futura del sistema. Prima di tutto, se il sistema originale fosse tempo continuo, bisognerebbe trasformare gli autovalori da tempo discreto a tempo continuo tramite la relazione:

|  |  |  |
| --- | --- | --- |
|  | k = ln()/ | (2.14) |

dove rappresenta il tempo di campionamento.

Adesso possiamo realizzare la vera e propria approssimazione dello stato futuro del sistema:

|  |  |  |
| --- | --- | --- |
|  |  | (2.15) |

dove è l’ampiezza iniziale di ogni modo, è la matrice le quali colonne sono gli autovettori e è la matrice diagonale contenente gli autovalori del sistema. Gli autovettori hanno la stessa dimensione dello stato x e **b** è il vettore dei coefficienti .

Adesso calcoliamo i valori del coefficiente Considerando le condizioni iniziali x1 che si hanno al tempo t1 = 0, possiamo calcolarli tramite la relazione x1 = . La matrice degli autovettori normalmente non quadrata e quindi possiamo ottenere il vettore **b** tramite la seguente relazione:

|  |  |  |
| --- | --- | --- |
|  |  | (2.16) |

Un esempio di codice scritto in MATLAB è presentato nell’algoritmo 2.1.

|  |
| --- |
| function [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(x,dt,lorenz)  if (lorenz==0)  thresh = 1e-10;  else thresh = 1e-6;  end  %% Matrix  X = x(:,1:end-1);  Xprime = x(:,2:end);  %% DMD  [U,Sigma,V] = svd(X,'econ'); % Step 1 con econ riduce il  % rango della matrice  r = length(find(diag(Sigma)>thresh)); % truncation value  Ur = U(:,1:r); % left singular vector matrix  Sigmar = Sigma(1:r,1:r); % singular values matrix  Vr = V(:,1:r); % right singular vector  % matrix  Atilde = Ur'\*Xprime\*Vr/Sigmar; % Step 2. It's the best-fit  % linear model  [W,Lambda] = eig(Atilde); % Step 3. They are the  % eigenvectors and the  % eigenvalues  Phi = Xprime\*(Vr/Sigmar)\*W;  lambda = diag(Lambda); % discrete-time eigenvalues  omega = log(lambda)/dt; % continuous-time eigenvalues  x1 = X(:, 1);  b = pinv(Phi)\*x1; % initial amplitude of each  % mode vector |

**Algoritmo 2.1** - Algoritmo DMD

All’inizio dell’algoritmo viene scelta la soglia in base al fatto che si stia eseguendo il DMD sul sistema di Lorenz o su un sistema con matrice diagonale. Successivamente vengono create le matrici e viene eseguito il DMD.

Per quanto riguarda la ricostruzione, una volta ottenuti tutti i dati necessari per realizzarla, viene eseguito un procedimento Matlab dove vengono ripetuti i calcoli teorici sopra indicati per l’approssimazione dello stato futuro. Di seguito è riportato un esempio di algoritmo di ricostruzione scritto in MATLAB all’interno dell’algoritmo 2.2.

|  |
| --- |
| %% DMD reconstruction  time\_dynamics = zeros(r, length(t));  for iter = 1:length(t)  time\_dynamics(:,iter) = (b.\*exp(omega\*t(iter)));  end  Xdmd = real(Phi \* time\_dynamics); |

**Algoritmo 2.2** - Algoritmo di ricostruzione DMD

In tutte le versioni del DMD si ha sempre l’utilizzo di ‘r’, cioè il valore di troncamento durante la SVD. Questo valore è molto importante innanzitutto perché permette la riduzione della dimensionalità del sistema, ma assume una rilevanza diversa in base alla versione dell’algoritmo: nelle versioni dove la matrice dei dati rimane invariata rispetto alle misurazioni eseguite sui dati, il valore di ‘r’ è molto vicino rispetto al valore delle variabili totali del sistema, quindi se il sistema è dell’ordine della decine di variabili, si potrebbe anche pensare di eseguire il DMD senza applicare questo troncamento. Nel caso in cui si stesse utilizzando l’Hankel DMD o un suo derivato, il numero di variabili sarebbe molto maggiore rispetto a quelle d’ingresso e, in questo modo, aumenterebbe di molto la complessità di esecuzione dell’algoritmo, quindi il valore della ‘r’ diviene di primaria importanza per riportare il sistema ad una complessità di esecuzione accettabile. Per trovare il valore corretto di ‘r’ viene usata usa soglia che può variare a seconda dei dati in ingresso e del sistema approssimato in ricostruzione. Ad esempio, per il sistema di Lorenz è stata usata una soglia del valore di , mentre i sistemi con matrici diagonali è stata usata una soglia del valore di .

### DMDc – Dynamic Mode Decomposition con controllo

La tecnica Dynamic Mode Decomposition con il controllo, anche chiamata DMDc, utilizza sia le misurazioni del sistema che un controllo applicato esternamente per scoprire le dinamiche sottostanti del sistema, quindi al sistema originario viene applicata una forzante esterna durante il periodo di osservazione. Può anche estrarre le caratteristiche input-output, in modo da permettere la costruzione di un modello dall’ordine ridotto che può essere usato per predire e modellizzare controllori per sistemi complessi ad alte dimensioni. Inoltre eredita tutte le caratteristiche vantaggiose del DMD: opera su dati campionati e gestisce efficientemente misurazioni altamente dimensionali, connettendo i dati misurati all’analisi del sistema dinamico non lineare [8].

L’obiettivo principale dell’algoritmo è quello di sviluppare modelli per le previsioni future e strategie di controllo per guidare il sistema verso uno stato desiderato [8], quindi il DMDc cerca di caratterizzare la relazione che si ha tra le tre misure fondamentali in gioco con questa tecnica: la misura corrente xk, lo stato futuro xk+1 e il controllo uk. La relazione tra queste tre misure può essere descritta tramite il seguente modello del sistema lineare:

|  |  |  |
| --- | --- | --- |
|  |  | (2.17) |

Dunque possiamo dire che questo sistema è dipendente dall’ingresso x, ma anche dal controllo u: .

Gli input del sistema sono:

* La matrice delle misurazioni dei primi m-1 istanti di tempo X

|  |  |  |
| --- | --- | --- |
|  |  | (2.18) |

* La matrice X shiftata di un istante di tempo in avanti X’

|  |  |  |
| --- | --- | --- |
|  |  | (2.19) |

* La matrice delle misurazioni del controllo Y

|  |  |  |
| --- | --- | --- |
|  |  | (2.20) |

X e X’ sono gli input originali del DMD, e a loro si aggiunge Y che aiuterà a interpretare e distinguere l’attuazione del controllo B e l’effettiva dinamica del sistema A.

È possibile riscrivere il modello lineare in termini di queste matrici dei dati:

|  |  |  |
| --- | --- | --- |
|  |  | (2.21) |

A rappresenta la matrice di stato presente anche nel DMD nativo e non è altro che l’evoluzione del sistema non forzato, mentre l’operatore B caratterizza l’impatto del controllo uk nei confronti dello stato futuro xk+1.

Il DMDc utilizza le tre matrici dei dati , e per trovare la migliore approssimazione per gli operatori e [8].

Possiamo avere due casi:

1. B è conosciuto

In questo caso sappiamo come stiamo influenzando il sistema e quindi ci ritroviamo nella situazione più semplice.

Per trovare l’operatore , in questo caso, basta sottrare a l’operatore moltiplicato la matrice dei dati del controllo :

|  |  |  |
| --- | --- | --- |
|  |  | (2.22) |

In questo modo è come se avessimo una nuova matrice e può essere eseguita tutta la procedura di un normale DMD. Nel caso in cui A fosse di alte dimensioni, si potrebbe fare il calcolo per gli autovalori e gli autovettori principali di tramite una rappresentazione ridotta

1. B è sconosciuto

Viene riformulato il problema e vengono create due matrici aumentate: G = [A B], dove , *n* è la dimensione dello stato x e *q* la dimensione dell’ingresso u. G rappresenta la matrice degli operatori aumentata e , dove che rappresenta la matrice dei dati aumentata.

In questo modo ci ritroviamo la seguente relazione:

|  |  |  |
| --- | --- | --- |
|  |  | (2.23) |

Viene eseguito l’algoritmo DMD su questa struttura e viene usata simultaneamente una regressione data-driven per risolvere la matrice concatenata formata da A e B. Questo comporta una SVD della matrice conosciuta Ω facendo una pseudo inversa e riordinando le parti generate al fine di ricavare le due matrici A e B.

Se non si tiene conto dei dati di controllo, verrà eseguito erroneamente il DMD nativo e si otterrà una matrice di stato A contaminata dagli effetti del controllo non considerato. Per ottenere più informazioni e distinguere le due matrici, oltre al controllo originario bisogna mettere un kick, cioè un segnale di rumore . Senza l’errore non si potrebbe capire se il sistema è senza forzanti o possiede un controllo in runtime.

I modelli generati tramite il DMDc, basati su misure lineari o non lineari di un sistema, sono stati recentemente usati con il modello di controllo predittivo (MPC) per ottenere un controllo migliorato dei sistemi non lineari. Sono state notate prestazioni sorprendentemente buone, anche con sistemi fortemente non lineari.

Un esempio di codice di DMD col controllo è mostrato nei prossimi algoritmi. Nell’algoritmo 2.3 viene dato uno sguardo alla creazione dei sistemi di ingresso e di controllo e alla costruzione delle matrici , e , che nell’algoritmo chiamiamo Ups.

|  |
| --- |
| g = transpose(15\*cos(5\*tspan-0.25)); % control  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)System\_v2(t,x,A,B,g,tspan),tspan,x0,options);  %% construction of the snapshot matrices  X = transpose(x(1:end-1,:));  Xprime = transpose(x(2:end,:));  Ups = transpose(g(1:end-1,:));  Omega = [X;Ups];  OmegaOne = Omega(:,1:end-1);  OmegaPrime = Omega(:,2:end); |

**Algoritmo 2.3** -Creazione delle matrici X, X’ e Ups

Vengono derivate le matrici OmegaOne e OmegaPrime, ma in realtà sono le X e Xprime con l’aggiunta del controllo.

Nel secondo algoritmo, il 2.4, si calcolano le SVD delle matrici di input Omega e di uscita X’ e viene inoltre calcolata l’approssimazione dell’operatore G.

|  |
| --- |
| %% compute the svd of the input space Omega  [~,Ur,Sigmar,Vr] = DMDc(Omega,l);  %% compute the svd of the output space X'  [~,Uhat,Sighat,Vhat] = DMDc(Xprime,l);  %% compute the approximation of the operator G = [A B]  nn = size(X,1); % length of the first dimension of X  q = size(Ups,1); % length of the first dimension of Ups  U\_1 = Ur(1:nn,:);  U\_2 = Ur(nn+q:nn+q,:); |

**Algoritmo 2.4** -SVD di Omega e X’ e approssimazione di G

Successivamente viene eseguita la ricostruzione del sistema, come visibile nell’algoritmo 2.5.

|  |
| --- |
| %% Evoluzione sistema discreto  approxAd=(Xprime)\*Vr\*inv(Sigmar)\*U\_1';  approxBd=(Xprime)\*Vr\*inv(Sigmar)\*U\_2';  approxX(:,1)= b;  for k=1:1000  approxX(:,k+1)=approxAd\*approxX(:,k)+approxBd\*g(k);  end |

**Algoritmo 2.5** – Ricostruzione del sistema

### DMD con la matrice di Hankel

Il DMD e l’eDMD sono algoritmi basati sulla disponibilità di misurazioni dell’intero stato del sistema, che, nella maggior parte delle situazioni, si traducono in un sistema altamente dimensionale. Tuttavia, spesso ci si ritrova nella situazione in cui sono disponibili solo delle informazioni/misurazioni parziali da cui si ottiene che alcune variabili saranno nascoste o latenti. In questa situazione si possono usare delle misurazioni ritardate nel tempo del sistema per creare un vettore di stato aumentato, il quale, algoritmicamente parlando, si trasforma nella creazione del DMD con Hankel [1].

Il DMD riesce a modellare e a predire le serie temporali ed è un metodo data-driven che scompone il sistema dato in ingresso nei suoi autovettori, ma assume che le serie temporali siano generate da sistemi LTI autonomi con molte variabili di stato, quindi i dati dovrebbero essere abbastanza ricchi da permettere un’approssimazione ottimale delle autofunzioni. Nei casi in cui i sistemi sono formati da poche variabili di stato, l’algoritmo di base commette errori proprio in questa approssimazione perché la dimensione delle matrici è troppo piccola per permettere una giusta approssimazione [6].

Utilizzando la matrice di Hankel prima di eseguire il DMD si creano nuove variabili di stato shiftate nel tempo. Questa strategia aumenta l’ordine del sistema di partenza e il DMD riesce a catturare molti più modi. L’uso del DMD con la matrice di Hankel migliora significativamente l’accuratezza delle predizioni a lungo termine dell’algoritmo rispetto a quelle fatte col DMD nativo riuscendo praticamente a duplicare il numero dei modi stimati [5].

Questa variante è stata anche usata per sviluppare una vista alternativa delle analisi di Koopman. Questo approccio è stato chiamato Hankel alternative view of Koopman (HAVOK) e consente di stimare i modelli lineari che catturano le dinamiche periodiche o quasi periodiche dei sistemi non lineari in una maniera quasi perfetta.

Di base, il DMD estrae i modi dominanti presenti nel segnale preso in esame per predire il suo comportamento futuro. Tuttavia, questo è possibile solo se l’insieme di variabili linearmente indipendenti contenente i modi dominanti ha una cardinalità maggiore o uguale rispetto al numero di questi modi. In caso contrario il DMD fallirà e restituirà dei risultati inaccurati.

Il DMD è pensato per lavorare con matrici dal basso rango. Se il numero di righe è molto superiore al numero delle colonne (), le matrici e avranno un rango basso e il DMD potrà lavorare senza problemi. In questo caso le variabili si potranno diffondere lungo tutti i modi dominanti e il rango della matrice sarà uguale al numero di questi modi. D’altro canto, ci sono alcuni insiemi di dati con dipendenze lineari all’interno delle proprie variabili misurate che gli impediranno di espandersi su tutti i modi dominanti. In queste situazioni il rango della matrice è troppo basso e il DMD non riesce a identificare correttamente le caratteristiche del sistema in esame.

La soluzione per questo problema è quella di incrementare il rango della matrice dei dati aggiungendo delle coordinate ritardate nel tempo prima di eseguire il DMD. Questo viene fatto riarrangiando l’insieme dei dati nella matrice di Hankel:

|  |  |  |
| --- | --- | --- |
|  |  | (2.24) |

Questo approccio rimodella i dati di input del sistema creando nuove variabili di stato. Queste ultime permettono di trovare la miglior matrice dal basso rango mantenendo una struttura abbastanza lineare. Nonostante ciò, l’introduzione di queste nuove variabili viene fatta a spesa di una diminuzione del numero di campioni presi dall’insieme di dati. Il numero di queste nuove variabili, cioè il numero di righe della matrice di Hankel, deve essere bilanciato tra l’abilità nel trovare i modi dominanti e l’accuratezza nella stima del modello futuro.

Adesso vedremo un algoritmo il quale riporta l’implementazione della matrice di Hankel per la creazione della tecnica DMDc.

Subito dopo la generazione dei dati e delle condizioni iniziali, si passa alla creazione della matrice di Hankel, come visibile nell’algoritmo 2.6:

|  |
| --- |
| %% Hankel matrix  x = x.';  n = 3000; % number of rows  m = length(tspan)-n; % number of columns  index1 = 1:n;  index2 = n:n+m-1;  X = []; Xprime=[];  for ir = 1:size(x,1)    % Hankel blocks ()  c = x(ir,index1).'; r = x(ir,index2);  H = hankel(c,r).';  c = x(ir,index1+1).'; r = x(ir,index2+1);  UH= hankel(c,r).';    X=[X,H]; Xprime=[Xprime,UH];  end  X=X';Xprime=Xprime'; |

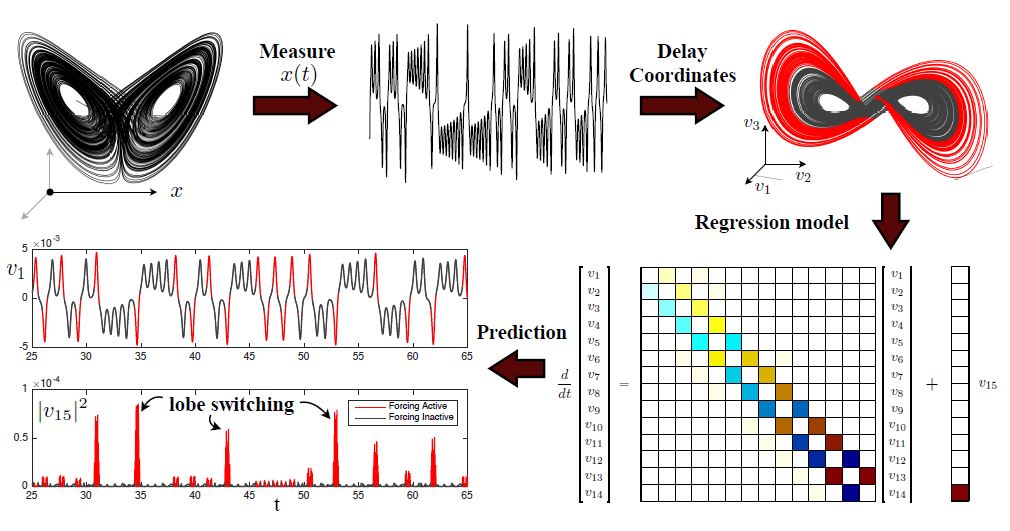
**Algoritmo 2.6** – Creazione della matrice di Hankel

In questo algoritmo notiamo che inizialmente viene trasposta la matrice dei dati perché, per eseguire il blocco di Hankel, bisogna avere le varie variabili disposte lungo le righe e non lungo le colonne e l’evoluzione delle misurazioni nelle colonne [9]; inoltre è possibile vedere l’indice n che rappresenta il numero di righe della matrice di Hankel. Nell’esempio è posto uguale a *3000* perché nel DMD serve lavorare con n >> m [5], ma in questo momento la matrice dei dati è stata trasposta quindi bisogna invertire la relazione. Alla fine della costruzione della matrice di Hankel, le matrici delle misurazioni vengono trasposte una seconda volta per tornare alle condizioni iniziali e poter eseguire il DMD.

Dopo aver rimodellato i dati all’interno della matrice di Hankel, viene eseguito l’algoritmo DMD base andandosi a ricavare gli autovettori del sistema in esame e potendo così essere in grado di eseguire la ricostruzione.

Come sappiamo, il DMD è uno strumento utilizzato per ricavare il sistema dinamico lineare più adatto dai dati misurati nel tempo e può anche essere usato per predire le serie temporali abbastanza periodiche tramite la stima dei suoi modi dominanti, ma può stimare solo i modi attraversati dalle variabili prese in esame. Se questi modi sono in una quantità limitata e/o insufficiente, il DMD non riesce a stimare correttamente i modi e le predizioni risultano molto inaccurate. Il numero di variabili può essere incrementato artificialmente tramite l’utilizzo della matrice di Hankel, che riesce ad aumentare la dimensione del sottospazio diffondendo queste variabili fino a quando non entrano in contatto con tutti i modi principali. In questo modo il DMD diviene uno strumento efficace per la previsione delle serie temporali anche disponendo di un numero ridotto di dati in ingresso e quindi di variabili.

La Figura 2.2 mostra, in breve, il funzionamento totale del DMD with Hankel.



**Figura 2.2** – Schematico DMD with Hankel

## SINDy

Prima della sua introduzione, la forma del modello di un sistema era vincolata alla conoscenza a priori delle equazioni che lo governavano o venivano ottimizzati vari parametri per adattare i dati al modello. Con l’introduzione del DMD si ottenevano modelli con l’approssimazione lineare più adatta, ma allo stesso tempo, identificare la struttura non lineare e i parametri del modello dai dati era considerata una sfida molto più stimolante [2]. I modelli creati dal SINDy sono parsimoniosi e identificano i termini più importanti che servono per spiegare il modello tramite le misurazioni effettuate [7].

### Sistema tempo continuo

È stato creato l’algoritmo SINDy o Sparse Identification of Nonlinear Dynamics proprio per identificare i sistemi dinamici non lineari e i parametri del modello utilizzando i dati ottenuti, basandosi sul fatto che molti sistemi hanno pochi termini nelle equazioni che governano il sistema [7]. Questo algoritmo va oltre la ricerca combinatoria attraverso tutte le possibili strutture del modello, diventata intrattabile con dati e modelli sempre più complessi, giocando sul fatto che molti sistemi dinamici del tipo:

|  |  |  |
| --- | --- | --- |
|  |  | (2.25) |

hanno dinamiche *f* con pochi termini attivi in tutto il proprio spazio. Per esempio, parlando del sistema di Lorenz, esso ha solo pochi termini lineari e quadratici in ogni equazione [2].

A questo punto si cerca di approssimare le dinamiche *f* tramite un modello lineare generalizzato:

|  |  |  |
| --- | --- | --- |
|  |  | (2.26) |

Si ha l’obiettivo di avere il minor numero di termini non nulli possibile all’interno di . Dopo avere derivato questo modello lineare è possibile usare la regressione sparsa per trovare i termini di rilievo che sono attivi nelle dinamiche; questo metodo riduce il numero di termini presenti all’interno delle dinamiche e quindi è molto adatto a problemi di grandi dimensioni.

Al fine di determinare le dinamiche *f* dai dati, le misurazione vengono collezionate campionando nel tempo lo stato x e le sue derivate , arrangiandole all’interno di due matrici: la matrice dei dati campionati nel tempo e quella delle derivate temporali , partendo dalla (2.25):

|  |  |  |
| --- | --- | --- |
|  |  | (2.27) |
|  |  | (2.28) |

Successivamente si costruisce la libreria delle funzioni candidate :

|  |  |  |
| --- | --- | --- |
|  |  | (2.29) |

dove ogni colonna di questa libreria rappresenta una funzione candidata. Si ha una grande libertà nella scelta degli ingressi non lineari per questa libreria. Un consiglio che viene sempre dato è quello di testare diverse librerie e usare la sparsità e l’accuratezza del modello risultante come uno strumento di diagnostica per l’ottenimento della combinazione di ingressi per la libreria migliore per rappresentare il sistema in esame [3].

### Sistema tempo discreto

È anche possibile usare il SINDy per identificare sistemi tempo discreto:

|  |  |  |
| --- | --- | --- |
|  |  | (2.30) |

come fatto nel DMD, per evitare di dover lavorare con le derivate. A questo punto viene costruita una libreria delle funzioni non lineari candidate , la (2.29).

In genere, il contenuto di questa libreria è limitato solo dalla propria immaginazione [2].

### Algoritmo

Adesso si può rappresentare il sistema dinamico in (2.25) sottoforma della matrici :

|  |  |  |
| --- | --- | --- |
|  |  | (2.32) |

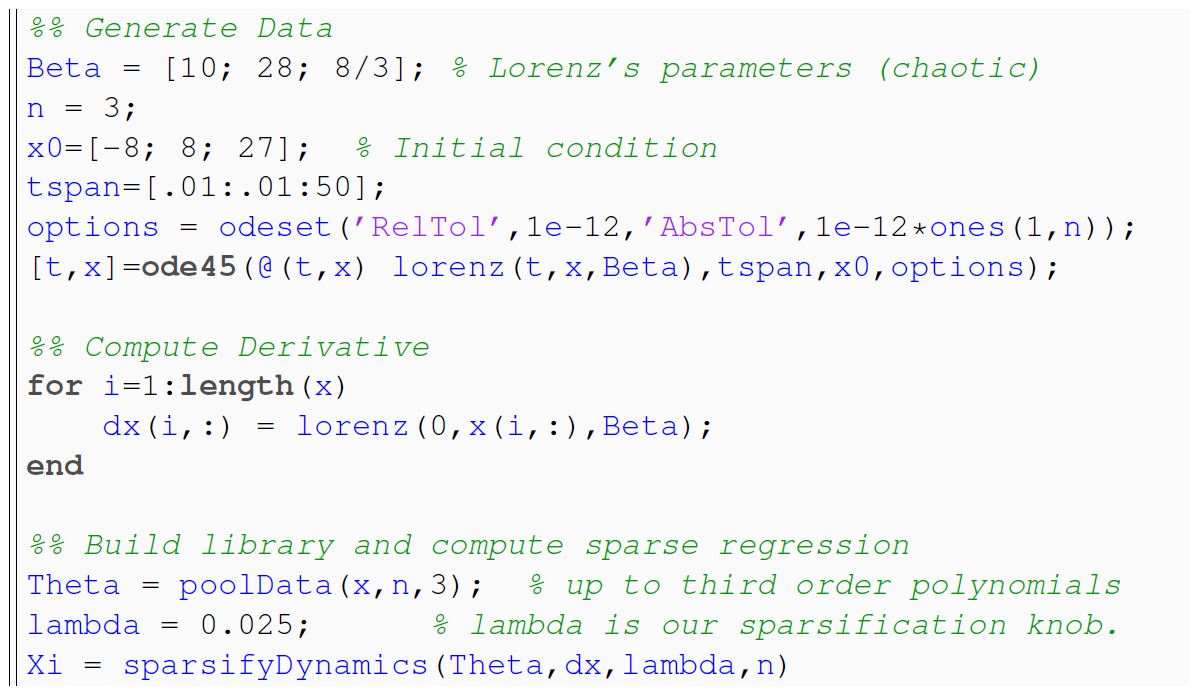
Ogni colonna di rappresenta un vettore di coefficienti che determinano i termini attivi nella k-esima riga. Prima di calcolare , in base al rumore presente nel sistema e nei dati misurati, potrebbe essere necessario filtrare le matrici e , usando, ad esempio, una soglia ottimale per la determinazione dei valori singolari.

Un modello ottimale potrebbe essere identificato usando una regressione sparsa e una di quelle più utilizzate con il SINDy è la Sequential Thresholded Least-squares (STLS). Questo algoritmo migliora la robustezza numerica dell’identificazione del modello nel caso in cui si abbiano rumori sovradeterminati. Successivamente viene proposto il codice per eseguire l’algoritmo STLS usato all’interno del SINDy, visibile nell’algoritmo 2.7.

|  |
| --- |
| function Xi = sparsifyDynamics(Theta,dXdt,lambda,n)  % compute Sparse regression: sequential least squares  Xi = Theta\dXdt; % initial guess: Least-squares  % lambda is our sparsification knob.  for k=1:10  smallinds = (abs(Xi)<lambda); % find small coefficients  Xi(smallinds)=0; % and threshold  for ind = 1:n % n is state dimension  biginds = ~smallinds(:,ind);  % Regress dynamics onto remaining terms to find sparse Xi  Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);  end  end |

**Algoritmo 2.7** – Algoritmo Sequential Thresholded Least-squares [2]

Nell’esempio seguente si può notare come il SINDy viene utilizzato per scoprire le equazioni del sistema di Lorenz partendo dai dati. Il codice presente all’interno dell’algoritmo 2.8 prima genera i dati che servono a ricreare il sistema di Lorenz, poi calcola le derivate del sistema, crea la libreria delle funzioni candidate e poi esegue la sparse regression:



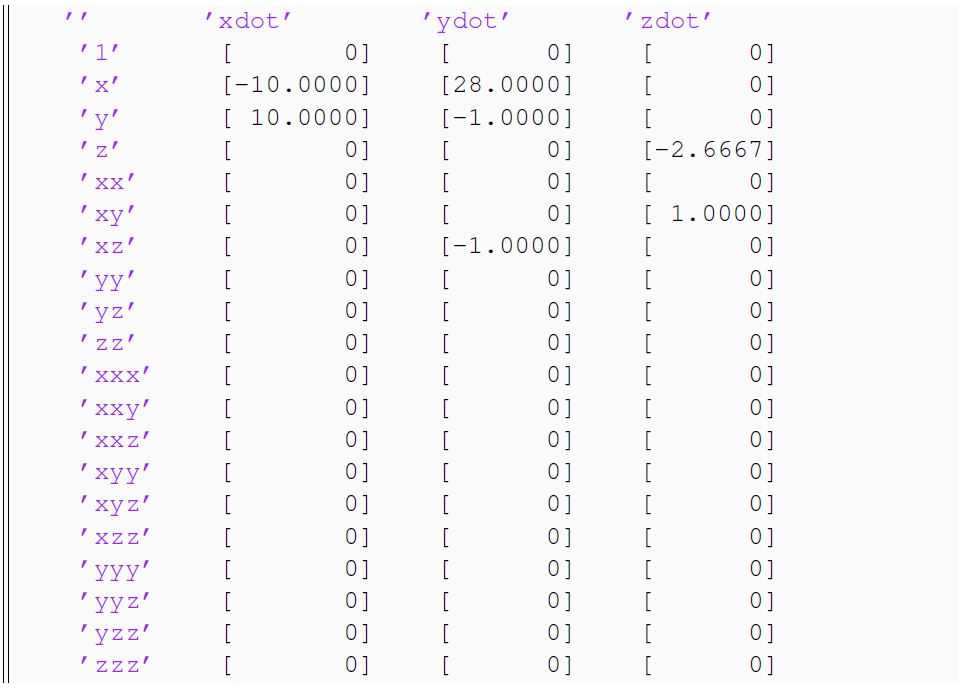
**Algoritmo 2.8** – SYNDy utilizzato per identificare il sistema di Lorenz dai dati [2]

Un’altra funzione molto importante presente all’interno del SINDy è la poolData, la quale permette di generare la libreria delle funzioni candidate. In questa funzione l’utente che sta utilizzando il SINDy può dare libero sfogo alla propria creatività in senso di tipologia di funzione che vuole inserire all’interno del proprio sistema, partendo da quelle polinomiali di qualsiasi ordine voluto, passando per le funzioni sinusoidali e inserendo qualsiasi altro tipo di funzione desiderata. Questa libreria potrebbe essere costruita solamente tramite le misurazioni fatte sul sistema in esame. Un esempio di come viene scritta la poolData è presente all’interno degli algoritmi 2.9:

|  |
| --- |
| function yout = poolData(yin,nVars,polyorder,usesine)  n = size(yin,1);  ind = 1;  % poly order 0  yout(:,ind) = ones(n,1);  ind = ind+1;  % poly order 1  for i=1:nVars  yout(:,ind) = yin(:,i);  ind = ind+1;  end  if(polyorder>=2)  % poly order 2  for i=1:nVars  for j=i:nVars  yout(:,ind) = yin(:,i).\*yin(:,j);  ind = ind+1;  end  end  end  if(polyorder>=3)  % poly order 3  for i=1:nVars  for j=i:nVars  for k=j:nVars  yout(:,ind) = yin(:,i).\*yin(:,j).\*yin(:,k);  ind = ind+1;  end  end  end  end  if(polyorder>=4)  % poly order 4  for i=1:nVars  for j=i:nVars  for k=j:nVars  for l=k:nVars  yout(:,ind) = yin(:,i).\*yin(:,j).\*yin(:,k).\*yin(:,l);  ind = ind+1;  end  end  end  end  end  if(polyorder>=5)  % poly order 5  for i=1:nVars  for j=i:nVars  for k=j:nVars  for l=k:nVars  for m=l:nVars  yout(:,ind) = yin(:,i).\*yin(:,j).\*yin(:,k).\*yin(:,l).\*yin(:,m);  ind = ind+1;  end  end  end  end  end  end  if(usesine)  n1 = size(yin,2);  for l=1:n1  for k=1:10;  yout = [yout sin(k\*yin(:,l)) cos(k\*yin(:,l))];  end  end  end |

**Algoritmo 2.9** – Creazione della funzione poolData [2]

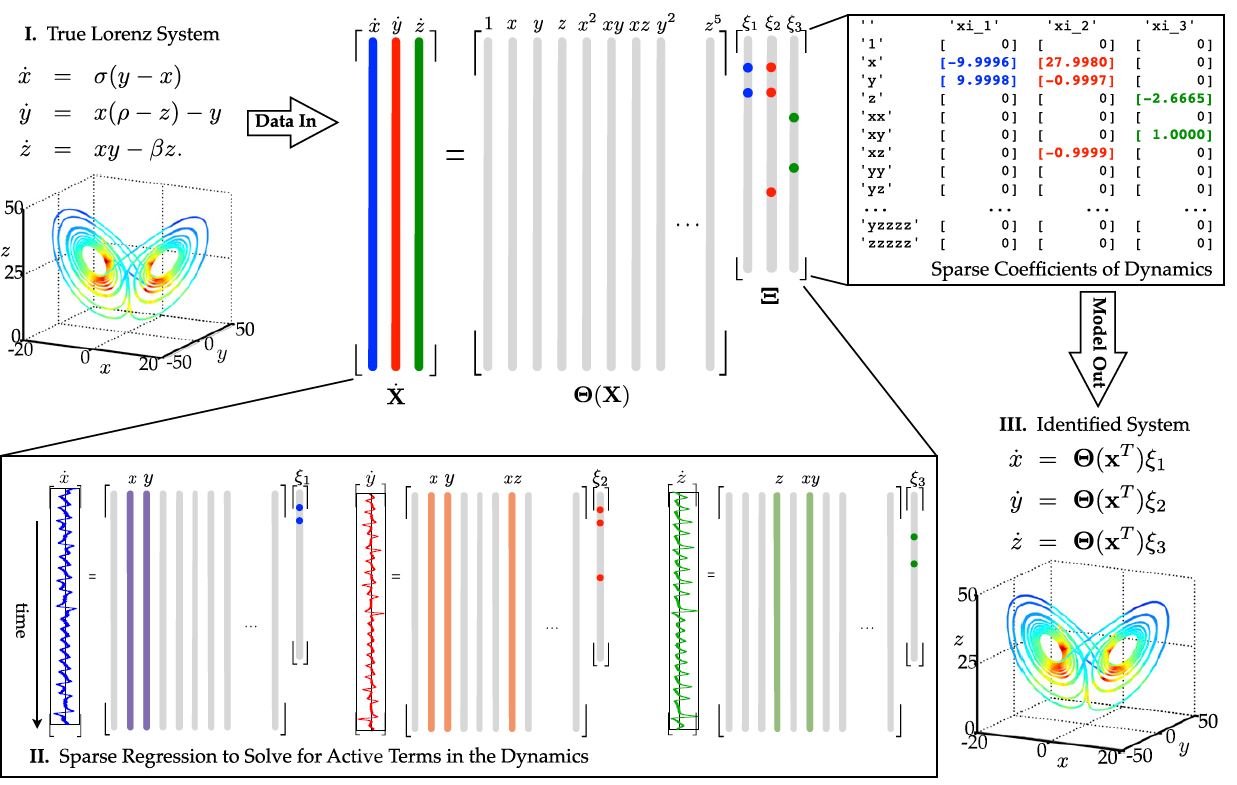
Infine, come uscita, l’algoritmo SINDy di base restituisce la matrice sparsa dei coefficienti , di cui ne è riportato un esempio, relativo al sistema di Lorenz, nell’algoritmo 2.10.



**Algoritmo 2.10** – Esempio di uscita dell’algoritmo SINDy [2]

Dunque il risultato del SINDy è un modello parsimonioso che include al suo interno solo i termini di maggiore importanza richiesti per spiegare il comportamento del sistema preso in esame. La procedura della sparse regression svolta dal SINDy è una procedura convessa, ma è l’unica fattibile visto che un approccio alternativo consisterebbe in una ricerca a forza bruta attraverso tutte le varie forme di modello candidate unite tra loro in modo combinatorio che risulta intrattabile. Questa ricerca combinatoria viene quindi oltrepassata tramite un’ottimizzazione convessa e il machine learning. Un punto a favore di questo algoritmo è che ha l’abilità di identificare il modello richiesto che contiene solo i termini non lineari richiesti con un’enorme accuratezza, evitando il sovraffollamento dei dati. Inoltre, è interessante notare che nei sistemi discreti, quando la libreria delle funzioni candidate è composta solo da termini lineari e se si elimina il termine che determina la sparsità dell’algoritmo, il SINDy si riduce alla Dynamic Mode Decomposition [2].

Successivamente viene mostrato lo schematico dell’algoritmo SINDy tramite la figura 2.3.



**Figura 2.3** – Schematico del SINDy che lavora sul sistema di Lorenz. Vengono selezionati i modelli parsimoniosi dalla libreria delle funzioni non lineari candidate tramite la sparse regression. [2] I dati vengono selezionati dal sistema, poi viene creata la libreria delle funzioni candidate e poi viene eseguita la sparse regression per trovare i termini più rilevanti per spiegare le dinamiche del sistema in esame. [3]

Le relazioni delle dinamiche non lineari con le strutture coerenti dominanti la si identifica dalle equazioni che governano il sistema. Molti di questi sistemi, nel mondo reale, sono ad alte dimensioni, ma evolvono in un attrattore e questo rende le dimensioni reali del sistema molto più piccole e aumenta l’usabilità del SINDy come algoritmo per la determinazione del sistema partendo dai dati.

### SYNDy con il controllo (SYNDy-MPC)

Viene esteso l’algoritmo SINDy per includere gli effetti del controllo e dimostrare l’abilità di questi modelli per migliorare le prestazioni dei dati presi con rumore. MPC, anche detto Model Predictive Control, è un metodo avanzato per il controllo del modello in esame mentre vengono imposti dei vincoli, il quale può essere applicato sia al DMD che al SINDy. È stato notato che il SINDy-MPC ha prestazioni migliori, richiede meno dati ed è più efficiente e robusto dal punto di vista computazionale rispetto agli algoritmi non lineari. Questo lo rende ottimo per l’esecuzione in real-time in risposta ai cambiamenti rapidi di un sistema. Mostra delle prestazioni migliori anche nell’esecuzione di modelli data-driven lineari, nonostante questi modelli possano fornire un ripiego fino a quando non sono disponibili abbastanza dati per utilizzare il SINDy nativo [7]. Il SINDy col controllo mostra anche un’accuratezza nella predizione e delle prestazioni relative al controllo migliori rispetto ai modelli non lineari, soprattutto per quantità di dati ridotte o moderate che contengono, però, anche del rumore. Inoltre, i modelli ricreati tramite questo algoritmo sono meno costosi da eseguire rispetto sempre all’utilizzo di un modello non lineare, e questo permette l’utilizzo del SINDy-MPC per applicazioni in tempo reale.

#### Sistema tempo continuo

Questo algoritmo è basato sulle stesse assunzioni del SINDy nativo, infatti il sistema dinamico non lineare in ingresso

|  |  |  |
| --- | --- | --- |
|  |  | (2.33) |

dove è lo stato, è il controllo in ingresso, ha le dinamiche che hanno, anche in questo caso, solo pochi termini attivi. Il SINDy è stato rapidamente generalizzato per includere l’attuazione, cioè il controllo, e questo ha richiesto l’ampliamento della libreria delle funzioni candidate che è diventata per includere anche il controllo. Le funzioni al suo interno adesso possono includere anche termini non lineari incrociati tra lo stato e il controllo. Vengono campionati sia lo stato che il segnale d’ingresso lungo *m* istanti di campionamento e poi arrangiati nelle seguenti due matrici

|  |  |  |
| --- | --- | --- |
|  | e | (2.34) |

Mentre la libreria delle funzioni non lineari candidate viene rappresentata tramite la seguente struttura

|  |  |  |
| --- | --- | --- |
|  |  | (2.35) |

dove definisce il vettore di tutte le combinazioni di prodotti delle variabili di e .

Nonostante in questo modo ci siano delle righe ripetute all’interno di , l’implementazione viene ridotta fino a delle combinazioni uniche nella pratica. È di cruciale importanza per la riuscita dell’algoritmo che venga generata una libreria delle funzioni candidate adatta al sistema in esame [7]. Una strategia ottimale per la creazione della libreria perfetta la si ha partendo da una scelta di base, ad esempio le funzioni polinomiali, e andando via via ad incrementarne la complessità aggiungendo altri termini, come le funzioni trigonometriche.

Adesso il sistema può essere scritto come

|  |  |  |
| --- | --- | --- |
|  |  | (2.36) |

Se le derivate temporali non vengono misurate direttamente, vengono calcolate tramite differenziazione numerica o approssimazione usando dei filtri, nel caso in cui i dati sono corrotti da del rumore. Essendo che i coefficienti sono sparsi per molti sistemi dinamici, viene eseguita la sparse regression per identificare quali di questi corrispondono alle poche non linearità presenti nella libreria della funzioni candidate.

#### Sistema tempo discreto

Nell’algoritmo di base è mostrato che è possibile identificare i modelli tempo discreto della forma

|  |  |  |
| --- | --- | --- |
|  |  | (2.37) |

Invece di calcolare le derivate, con questo genere di sistemi viene creata la matrice con le colonne di portate in avanti di un istante temporale: .

Le dinamiche possono essere riscritte nel modo seguente:

|  |  |  |
| --- | --- | --- |
|  |  | (2.38) |

Nel DMD viene effettuata una regressione simile per identificare il modello lineare tempo discreto che mappa in : . Grazie a questo, è possibile dichiarare che il SINDy si riduce al DMD se viene formulato con un sistema tempo discreto, con la libreria delle funzioni candidate contenente solo termini lineari e con il termine che determina la sparsità nullo [7].

Se l’input corrisponde al controllo di feedback, in modo tale da avere , è impossibile distinguere l’effetto del controllo dal termine di feedback interno all’interno del sistema dinamico e la regressione del SINDy-MPC diventa mal condizionata. In questo caso il controllo potrebbe essere identificato come funzione dello stato.

|  |  |  |
| --- | --- | --- |
|  |  | (2.39) |

I modelli non lineari basati sul machine learning sono sempre più usati, ma spesso queste tecniche fanno affidamento all’accesso ad un enorme mole di dati, hanno un’abilità limitata nella generalizzazione di quest’ultimi, non sono sempre pronti a comprendere dei vincoli fisici e presentano una computazione molto costosa dal punti di vista del tempo e del consumo. Al contrario, i modelli ottenuti utilizzando DMD con controllo e SINDy con controllo hanno delle prestazioni molto superiori ai primi e hanno un tempo di esecuzione molto più basso rispetto ai modelli non lineari.

## Sistema di Lorenz

Durante i vari esempi e le numerose prove, svoltesi sugli algoritmi DMD e SINDy e relative varianti, è stato usato un sistema in particolare per verificare il funzionamento degli algoritmi e quello in questione è il sistema di Lorenz, che si sviluppa dall’omonimo attrattore. È stato utilizzato in due versioni, quella di base e quella con forzante in ingresso.

### Sistema di Lorenz di base

Il sistema di Lorenz è stato scoperto nel 1963 ed è stato il primo esempio di sistema formato da equazioni differenziali in grado di generare un comportamento caotico. Quindi sostanzialmente è un sistema caotico numerico con un chiaro significato fisico.

Il sistema è formato da tre equazioni differenziali del primo ordine e si presenta nel seguente modo:

|  |  |  |
| --- | --- | --- |
|  |  | (2.40) |

dove , e sono i parametri caotici del sistema di Lorenz.

Il funzionamento di questo sistema è quello di descrivere un andamento chiamato a “farfalla” attorno all’attrattore del sistema, che è un punto verso il quale evolve il sistema dopo un tempo abbastanza lungo, partendo da delle precise condizioni iniziali. Questo comportamento aiuta a capire se il sistema approssimato in seguito all’utilizzo di DMD o SINDy risulta corretto o contiene degli errori.

Nel seguente algoritmo 2.11 è presente il codice per ricreare il sistema di Lorenz usando MATLAB:

|  |
| --- |
| function dx = lorenz(t,x,Beta)  %% Lorenz equations  dx = [  Beta(1)\*(x(2)-x(1));  x(1)\*(Beta(2)-x(3)) - x(2);  x(1)\*x(2) - Beta(3)\*x(3);  ];  end |

**Algoritmo 2.11** - Sistema di Lorenz

Nelle prove sono stati usati [10; 28; 8/3] come parametri e [0; 1; 20] come condizioni iniziali.

### Sistema di Lorenz con forzante

I sistemi come quello di Lorenz sono molto utilizzati per la crittografia e possono aumentare la sicurezza nelle comunicazioni multimediali, ma per assicurare questi pregi è buona norma utilizzare delle varianti che consentano il controllo. Una di queste è il sistema di Lorenz autonomo 4D basato su un approccio con controllo [4]. Il controllo è una funzione sinusoidale introdotta nella prima equazione del sistema di Lorenz di base. Il sistema in questione è formato dalle tre equazioni seguenti:

|  |  |  |
| --- | --- | --- |
|  |  | (2.41) |

dove , e sono sempre i parametri caotici del sistema di Lorenz e è la funzione sinusoidale che svolge il ruolo di ingresso di controllo; rappresenta l’ampiezza e la frequenza della funzione. Considerando il sistema può essere riscritto nel modo seguente:

|  |  |  |
| --- | --- | --- |
|  |  | (2.42) |

Questo sistema, dal punto di vista implementativo, l’accuratezza della regolazione in frequenza solitamente è maggiore rispetto a quella del controllo dei parametri e questo lo rende più conveniente per le regolazioni dei parametri all’interno dei circuiti [4].

Nel seguente algoritmo 2.12 viene mostrato il codice per la realizzazione pratica del sistema di Lorenz con controllo:

|  |
| --- |
| function dx = lorenz\_c(t,x,Beta,g,gt,a)  %% Lorenz equations  g = interp1(gt,g,t);  dx = [  -Beta(1)\*(x(1)-x(2)-(a\*g));  x(1)\*(Beta(2)-x(3)) - x(2);  x(1)\*x(2) - Beta(3)\*x(3);  ];  end |

**Algoritmo 2.12** - Sistema di Lorenz con controllo

All’inizio di questo codice viene eseguita la funzione di interpolazione che serve al fine di creare il controllo sinusoidale. Successivamente sono riportate le equazioni differenziali del sistema con l’aggiunta del controllo. Nelle prove sono stati usati [8; 35; 8/3] come parametri, [0; 1; 20] come condizioni iniziali, rad/s.

In questo sistema e in generale, in tutti i sistemi con delle funzioni sinusoidali, viene usata la funzione “interp1” nativa di MATLAB per interpolare l’insieme dei dati con l’insieme dei tempi. In questo modo si crea la funzione (sinusoidale) continua da due insiemi finiti.

# DMD: casi di studio

Di seguito saranno esposte vari applicazioni del DMD nelle sue diverse versioni: nativo, con il controllo e con la matrice di Hankel.

## DMD nativo con matrice d’ingresso diagonale

Inizialmente bisogna dare le condizioni iniziali del sistema e tutti i parametri per la creazione della matrice diagonale, la quale sarà l’ingresso del nostro sistema. Inoltre vengono disposti i dati nelle matrici X e X’ ed eseguiamo il DMD, che è stato inserito in tutti i successivi algoritmi sottoforma di funzione per snellire il codice. Sarà presente in due versioni e in questo primo codice verrà usato il DMD di base. Il tutto lo si può vedere nell’algoritmo 3.1, mentre il DMD usato è presente nell’algoritmo 3.2.

|  |
| --- |
| clear all, close all  l = 0; % Lorenz variable  params = [-1 -2 -3 -4 -5 -20 -30 -40 -50 ];  x0 = ones(1,9); % initial condition  dt = 0.001;  tspan = 0:dt:10-dt; % time span  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)diagonal\_v1(t,x,params),tspan,x0,options);  %% Matrix creation  X = transpose(x(1:end-1,:));  Xprime = transpose(x(2:end,:));  %% DMD  [Phi,b,omega,r] = DMD(X,Xprime,dt,l); |

**Algoritmo 3.1** – Creazione variabili e sistema ed esecuzione del DMD

|  |
| --- |
| function [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(X,Xprime,dt,lorenz)  if (lorenz==0)  thresh = 1e-10;  else thresh = 1e-6;  end  [U,Sigma,V] = svd(X,'econ'); % Step 1 con econ riduce il  % rango della matrice  r = length(find(diag(Sigma)>thresh)); % truncation value  Ur = U(:,1:r); % left singular vector matrix  Sigmar = Sigma(1:r,1:r); % singular values matrix  Vr = V(:,1:r); % right singular vector  % matrix  Atilde = Ur'\*Xprime\*Vr/Sigmar; % Step 2. It's the best-fit  % linear model  [W,Lambda] = eig(Atilde); % Step 3. They are the  % eigenvectors and the  % eigenvalues  Phi = Xprime\*(Vr/Sigmar)\*W;  lambda = diag(Lambda); % discrete-time eigenvalues  omega = log(lambda)/dt; % continuous-time eigenvalues  x1 = X(:, 1);  b = pinv(Phi)\*x1; % initial amplitude of each  % mode vector |

**Algoritmo 3.2** – Esecuzione del DMD

Dopo l’esecuzione dell’algoritmo, viene eseguita la ricostruzione, visibile nell’algoritmo 3.3, per andare a fare il confronto tra il sistema reale e quello approssimato, Figura 3.1, e visualizzare l’errore presente in ogni variabile tra i due sistemi, Figura 3.2.

|  |
| --- |
| %% DMD reconstruction  time\_dynamics = zeros(r, length(t));  for iter = 1:length(t)  time\_dynamics(:,iter) = (b.\*exp(omega\*t(iter)));  end  Xdmd = real(Phi \* time\_dynamics); |

**Algoritmo 3.3** – Ricostruzione del sistema approssimato

Tramite l’ultima riga di codice della ricostruzione si ritorna ad un sistema comprendente un numero di variabili uguali a quelle in ingresso.

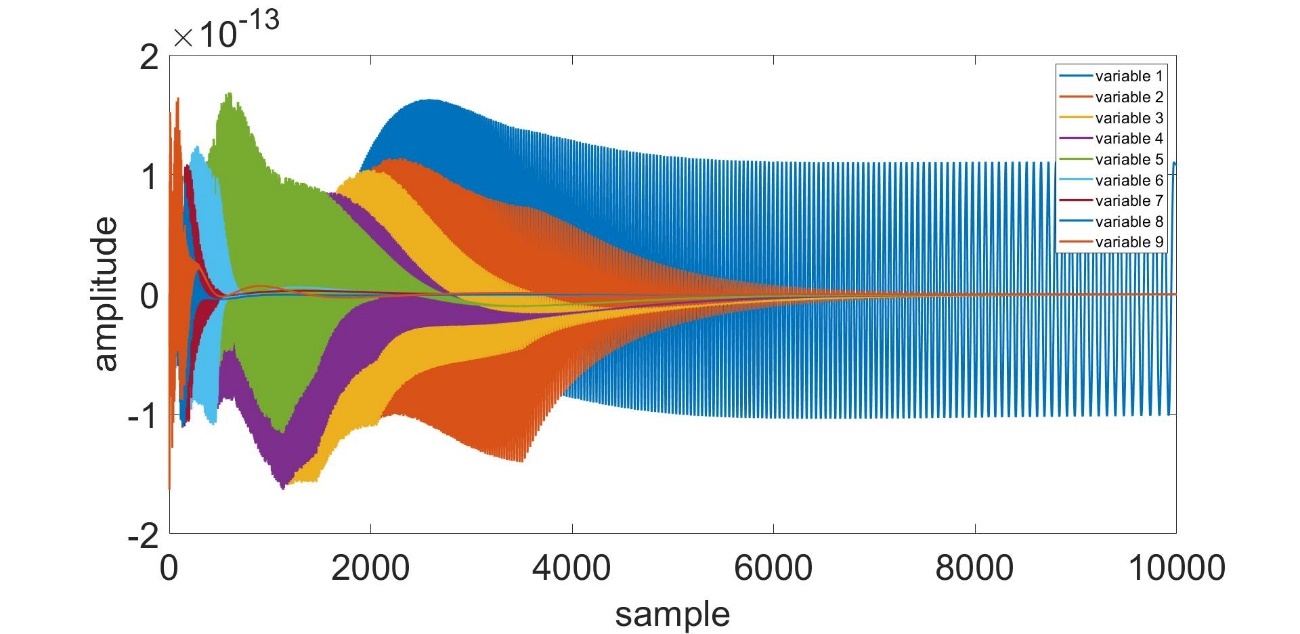
Il codice per la generazione del sistema è quello presente nell’algoritmo 3.4:

|  |
| --- |
| function dx = diagonal\_v1(t,x,A)  %% Lorenz equations  dx = [  A(1)\*x(1);  A(2)\*x(2);  A(3)\*x(3);  A(4)\*x(4);  A(5)\*x(5);  A(6)\*x(6);  A(7)\*x(7);  A(8)\*x(8);  A(9)\*x(9);  ]; |

**Algoritmo 3.4** – Generazione del sistema



**Figura 3.1** – Confronto tra il sistema reale e approssimato



**Figura 3.2** – Errore di ricostruzione presente in ogni variabile

## Hankel DMD con matrice d’ingresso diagonale

All’inizio dell’algoritmo avviene la dichiarazione delle variabili e la creazione del sistema da attenzionare. Subito dopo viene modellata la matrice dei dati in ingresso per trasformarla nella matrice di Hankel, quindi con il ritardo temporale. Tutto questo viene attenzionato nell’algoritmo 3.5.

|  |
| --- |
| clear all, close all  l = 0; % Lorenz variable  params = [-1 -2 -3 -4 -5 -20 -30 -40 -50];  x0 = ones(1,9); % initial condition  dt = 0.001;  tspan = 0:dt:5-dt; % 5.000 time span  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)diagonal\_v1(t,x,params),tspan,x0,options);  %% hankel matrix  x = x.';  n = 20; % number of rows  m = length(tspan)-n; % number of columns  index1 = 1:n;  index2 = n:n+m-1;  X = []; Xprime=[];  for ir = 1:size(x,1)    % Hankel blocks ()  c = x(ir,index1).'; r = x(ir,index2);  H = hankel(c,r).';  c = x(ir,index1+1).'; r = x(ir,index2+1);  UH= hankel(c,r).';    X=[X,H]; Xprime=[Xprime,UH];  end  X=X';Xprime=Xprime'; |

**Algoritmo 3.5** – Creazione variabili, sistema e matrice di Hankel

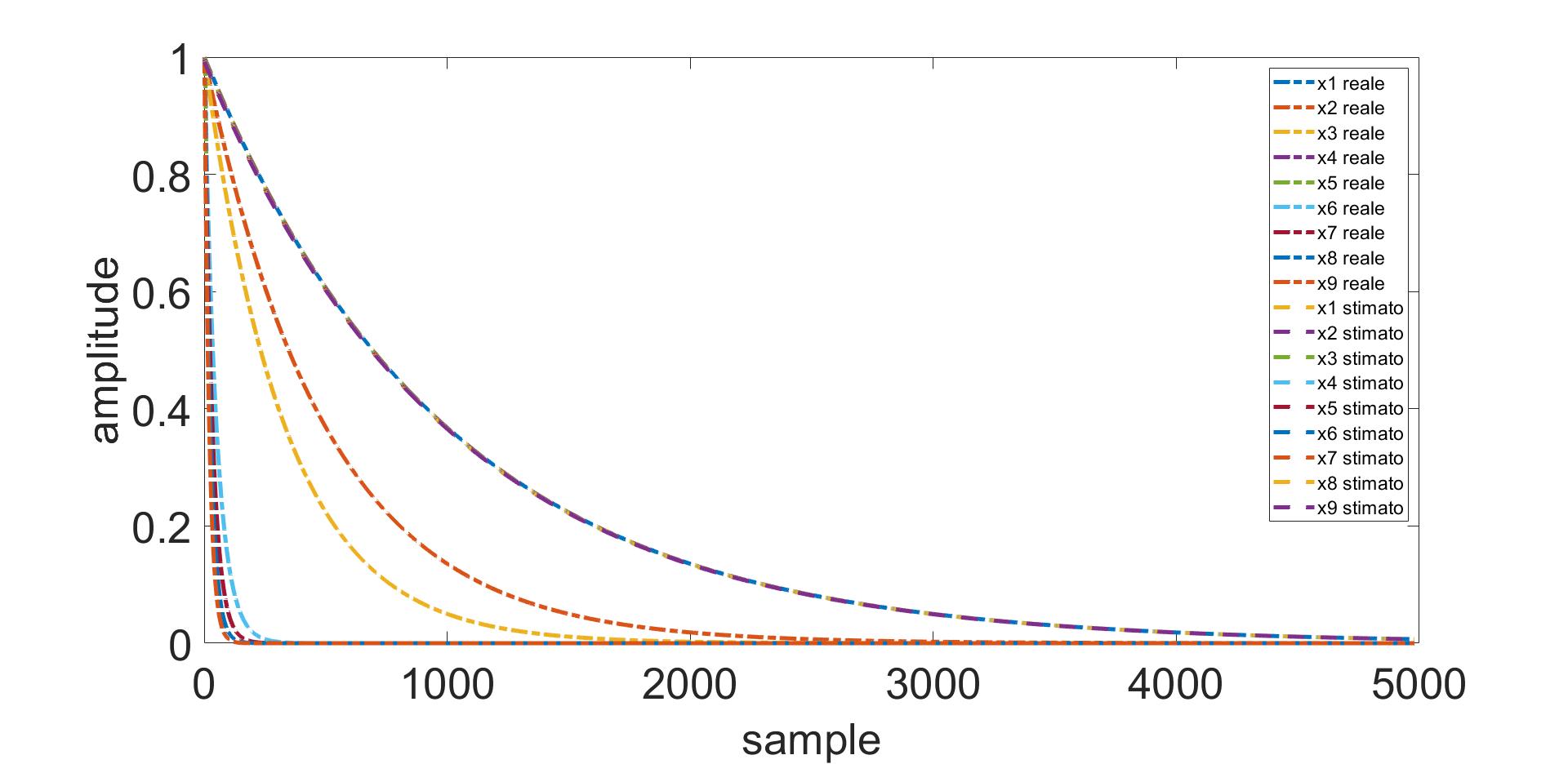
In questo caso di studio sono state utilizzate 20 righe per la matrice di Hankel, ma questo 20 rappresenta il numero di righe per variabile che saranno presenti all’interno della matrice; infatti le matrici e sono composte da 180 righe ciascuna, questo a causa delle 20 righe e delle 9 variabili in ingresso. Subito dopo viene eseguito il DMD nativo, ma questa volta le due matrici X e X’ sono ricavate direttamente dalla struttura di Hankel, e poi viene ricostruito il sistema approssimato. Il procedimento è visibile all’interno dell’algoritmo 3.6.

|  |
| --- |
| %% DMD of x  [Phi,b,omega,r] = DMD(X,Xprime,dt,l);  %% DMD reconstruction  time\_dynamics = zeros(r, m);  for iter = 1:m  time\_dynamics(:,iter) = (b.\*exp(omega\*t(iter)));  end  Xdmd = real(Phi \* time\_dynamics);  Xdmdf = Xdmd(1:n:end,:); |

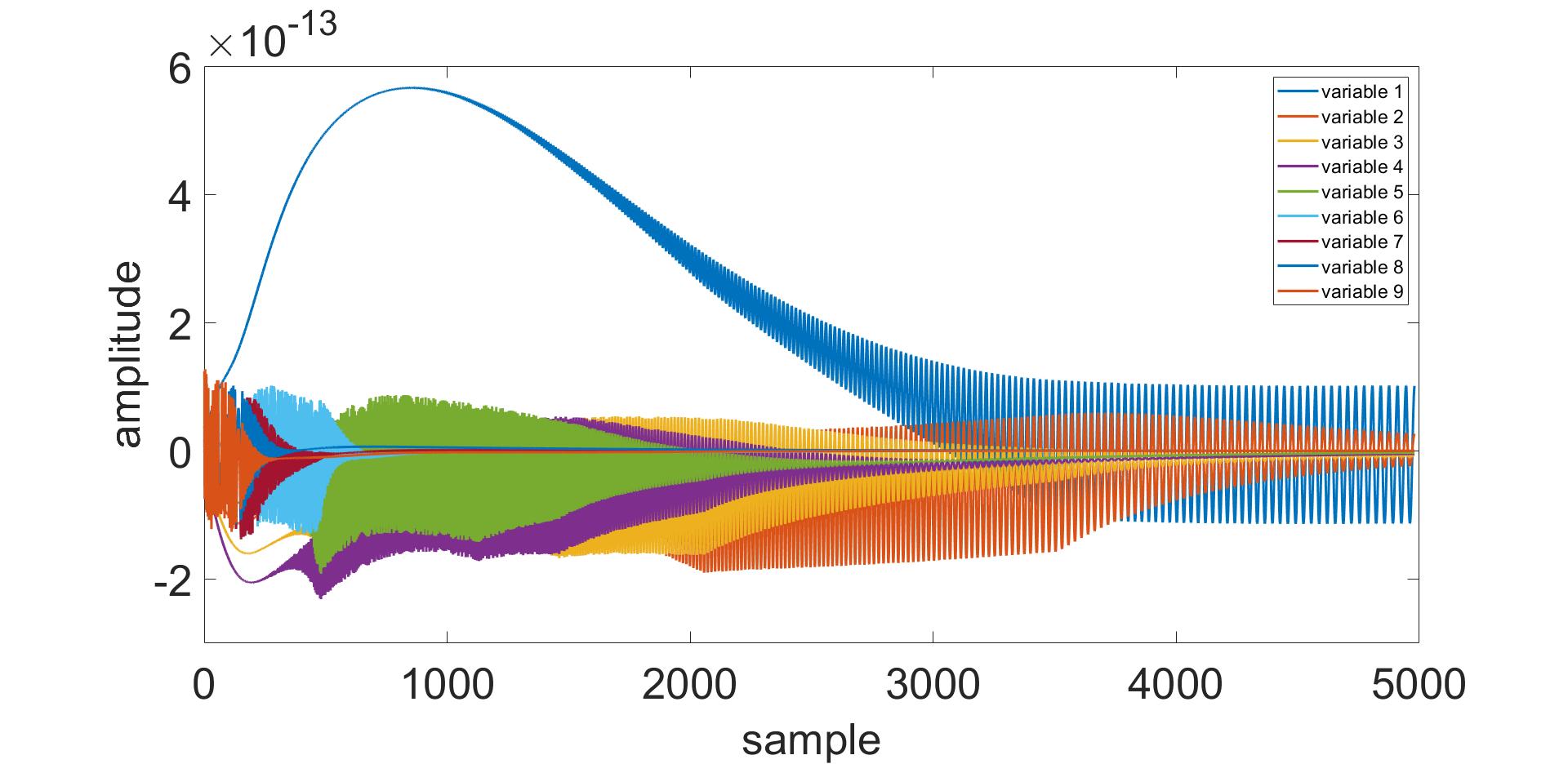
**Algoritmo 3.6** – Esecuzione DMD e ricostruzione sistema approssimato

Alla fine del codice le variabili approssimate vengono ridotte per risultare uguali a quelle del sistema reale. Questa riduzione è dovuta all’introduzione della matrice di Hankel all’interno dell’algoritmo.

A questo punto viene fatto il confronto tra il sistema reale e quello approssimato, Figura 3.3, e viene visualizzato l’errore presente in ogni variabile tra i due sistemi, Figura 3.4.

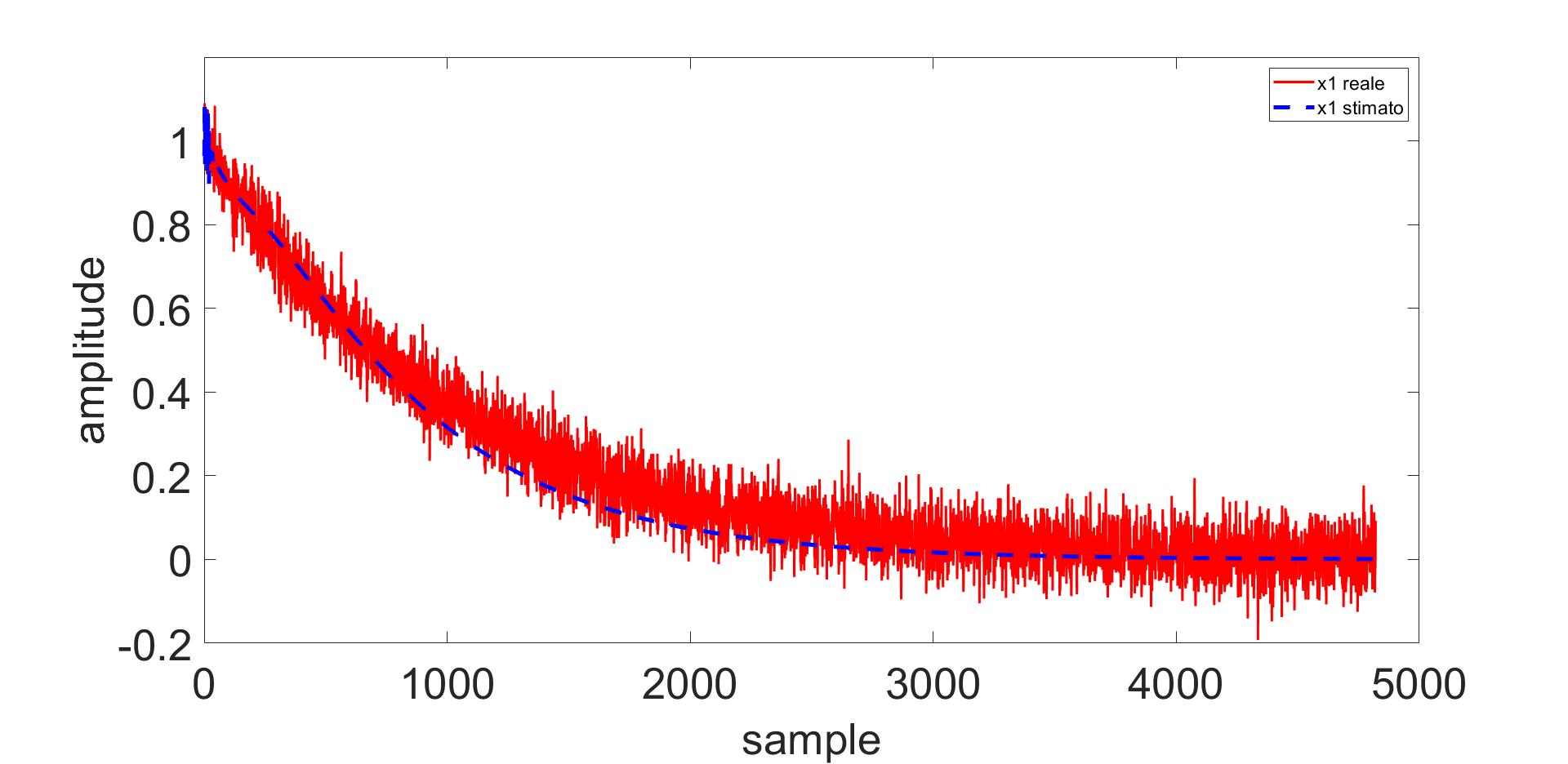


**Figura 3.3** – Confronto tra il sistema reale e approssimato



**Figura 3.4** – Errore di ricostruzione presente in ogni variabile

Successivamente viene mostrato come, aggiungendo del rumore al segnale d’ingresso, il DMD con Hankel riesce ad ottenere un segnale approssimato pari a quello originale libero dal rumore in Figura 3.5.



**Figura 3.5** – Confronto dei sistema con rumore in ingresso

Il rumore è stato aggiunto tramite il seguente comando:

|  |  |  |
| --- | --- | --- |
|  | x = awgn(x,10,'measured'); | (3.1) |

Con questa espressione è stato aggiunto del rumore Gaussiano al segnale di partenza e 10 rappresenta il rapporto segnale-rumore in dB. ‘measured’ serve a generare un livello di rumore appropriato basato sul rapporto segnale-rumore.

## Hankel DMD con sistema di Lorenz

All’interno del terzo esempio di algoritmo relativo al DMD viene creato un codice per lavorare con il sistema di Lorenz e, per farlo, viene usato l’Hankel DMD.

Inizialmente avviene la dichiarazione delle variabili e la creazione del sistema di Lorenz. Successivamente vengono rimodellati i dati per creare la matrice di Hankel. Tutto questo è visibile nell’algoritmo 3.7.

|  |
| --- |
| clear all, close all  %% Dichiarazione variabili  l = 1; % Lorenz variable  Beta = [10; 28; 8/3]; % chaotic values  x0 = [0; 1; 20]; % initial condition  dt = 0.001;  tspan = 0:dt:15-dt; % time span  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)lorenz(t,x,Beta),tspan,x0,options);  %% Hankel matrix  x = x.';  n = 3000; % number of rows  m = length(tspan)-n; % number of columns  index1 = 1:n;  index2 = n:n+m-1;  X = []; Xprime=[];  for ir = 1:size(x,1)    % Hankel blocks ()  c = x(ir,index1).'; r = x(ir,index2);  H = hankel(c,r).';  c = x(ir,index1+1).'; r = x(ir,index2+1);  UH= hankel(c,r).';    X=[X,H]; Xprime=[Xprime,UH];  end  X=X';Xprime=Xprime'; |

**Algoritmo 3.7** - Dichiarazione variabili e creazione della matrice di Hankel

Prima di creare le matrici, i dati vengono trasposti e poi vengono dichiarate le righe e le colonne della matrice di Hankel, che nel caso del sistema di Lorenz sono rispettivamente 3000 e 6000. Alla fine di questo processo, i dati vengono trasposti una seconda volta per poter eseguire il DMD. Il sistema di lorenz utilizzando in questo codice è quello dell’algoritmo 2.12.

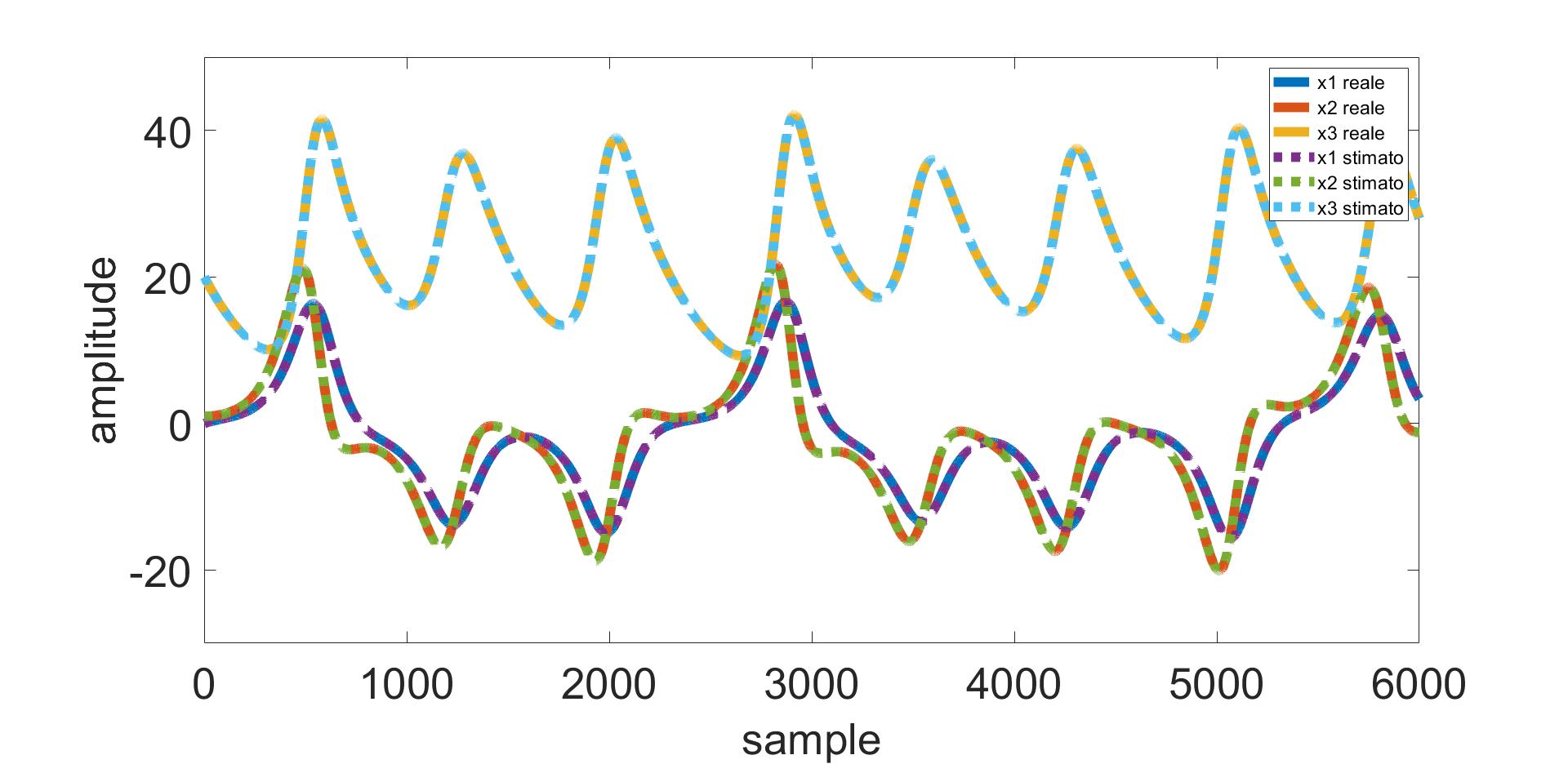
Come passaggio successivo si ha la computazione del DMD e infine la ricostruzione del segnale approssimato. Questi due step si trovano all’interno dell’algoritmo 3.8.

|  |
| --- |
| %% DMD of x  [Phi,b,omega,r] = DMD(X,Xprime,dt,l);  %% DMD reconstruction  time\_dynamics = zeros(r, m);  for iter = 1:m  time\_dynamics(:,iter) = (b.\*exp(omega\*t(iter)));  end  Xdmd =real(Phi \* time\_dynamics);  Xdmdf = Xdmd(1:n:end,:); |

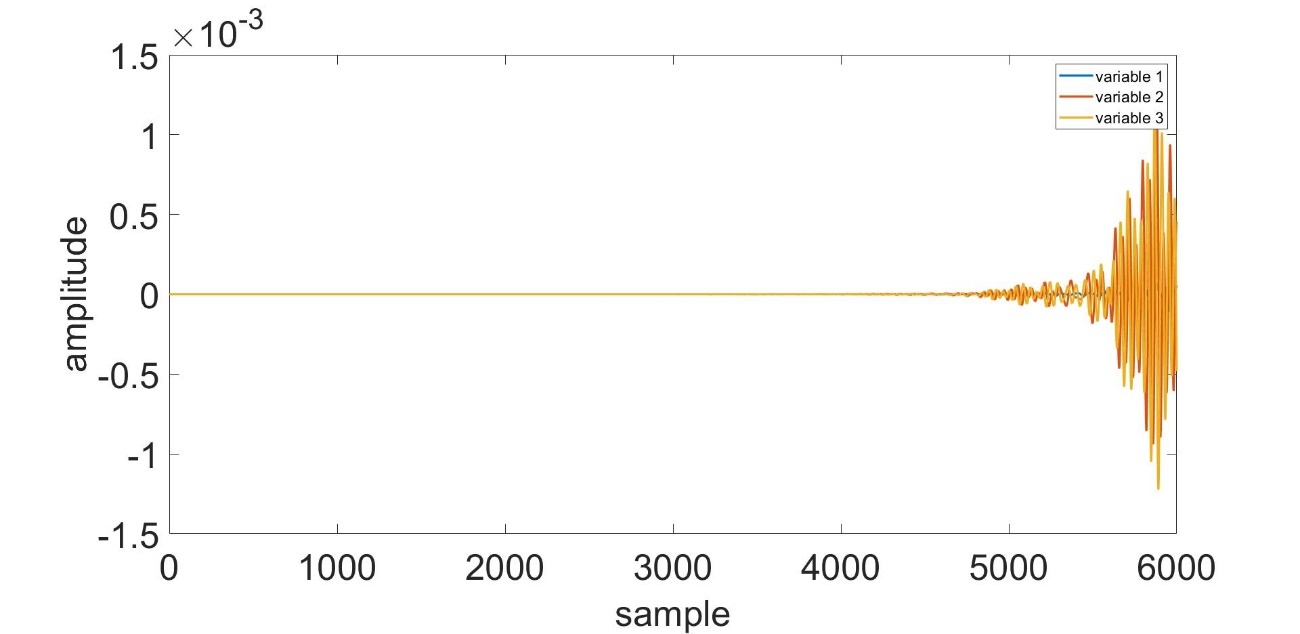
**Algoritmo 3.8** - Esecuzione DMD e ricostruzione sistema approssimato

Alla fine del codice le variabili approssimate vengono ridotte per risultare uguali a quelle del sistema reale. Questa riduzione è dovuta all’introduzione della matrice di Hankel all’interno dell’algoritmo. Con una soglia di è stato ottenuto un valore di r pari a 344.

A questo punto viene fatto il confronto tra il sistema reale e quello approssimato, Figura 3.6, e viene visualizzato l’errore presente in ogni variabile tra i due sistemi, Figura 3.7.



**Figura 3.6** – Confronto tra il sistema reale e approssimato



**Figura 3.7** – Errore di ricostruzione per le tre variabili di stato

## DMDc con matrice d’ingresso diagonale

Con il quarto caso viene introdotto il controllo all’interno dell’algoritmo DMD.

Come prima operazione vengono dichiarate tutte le variabili che serviranno all’interno del codice, viene realizzato l’ingresso sinusoidale di controllo, viene generato il sistema tramite l’utilizzo di ode45 e infine i dati vengono immagazzinati all’interno della matrici che serviranno al DMD. Tutto questo è presente nell’algoritmo 3.8.

|  |
| --- |
| clear all, close all  l = 0; % Lorenz variable  dt = 0.01;  tspan = 0:dt:10-dt;  A = [-1 0 0 0 0; 0 -2 0 0 0; 0 0 -3 0 0;  0 50 0 -100 0; 0 0 10 0 -100];  B = [1 2 3 4 5]';  x0 = [1 .5 -1 2 3]; % initial condition  g = transpose(15\*cos(5\*tspan-0.25)); % control  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)System\_v2(t,x,A,B,g,tspan),tspan,x0,options);  %% construction of the snapshot matrices  X = transpose(x(1:end-1,:));  Xprime = transpose(x(2:end,:));  Ups = transpose(g(1:end-1,:));  Omega = [X;Ups];  OmegaOne = Omega(:,1:end-1);  OmegaPrime = Omega(:,2:end); |

**Algoritmo 3.8** – Dichiarazione variabili, creazione sistema e matrici

Gli ultimi passaggi di questo codice consistono in un ridimensionamento della matrice Omega per la creazione di due matrici che hanno lo stesso scopo di X e Xprime, ma con all’interno anche la variabile di controllo.

Nei passaggi successivi viene eseguito il DMD sulla matrice Omega dove viene eseguito il DMD nativo per ottenere anche la matrice Phi che servirà per la ricostruzione del sistema approssimato. Poi si la ricostruzione del sistema. Questo lo si può vedere all’interno dell’algoritmo 3.9.

|  |
| --- |
| %% compute the svd of the input space Omega  [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(OmegaOne,OmegaPrime,dt,l);  %% DMD reconstruction  time\_dynamics = zeros(r, length(t));  for iter = 1:length(t)  time\_dynamics(:,iter) = (b.\*exp(omega\*t(iter)));  end  Xdmd = real(Phi \* time\_dynamics);  Xdmdf = Xdmd(1:5,:); |

**Algoritmo 3.9** – SVD di Omega e ricostruzione del sistema

Nell’ultima riga di questo codice vengono prese solo le prime 5 variabili del sistema approssimato rispetto alle 6 totali perché l’ultima rappresenta il controllo.

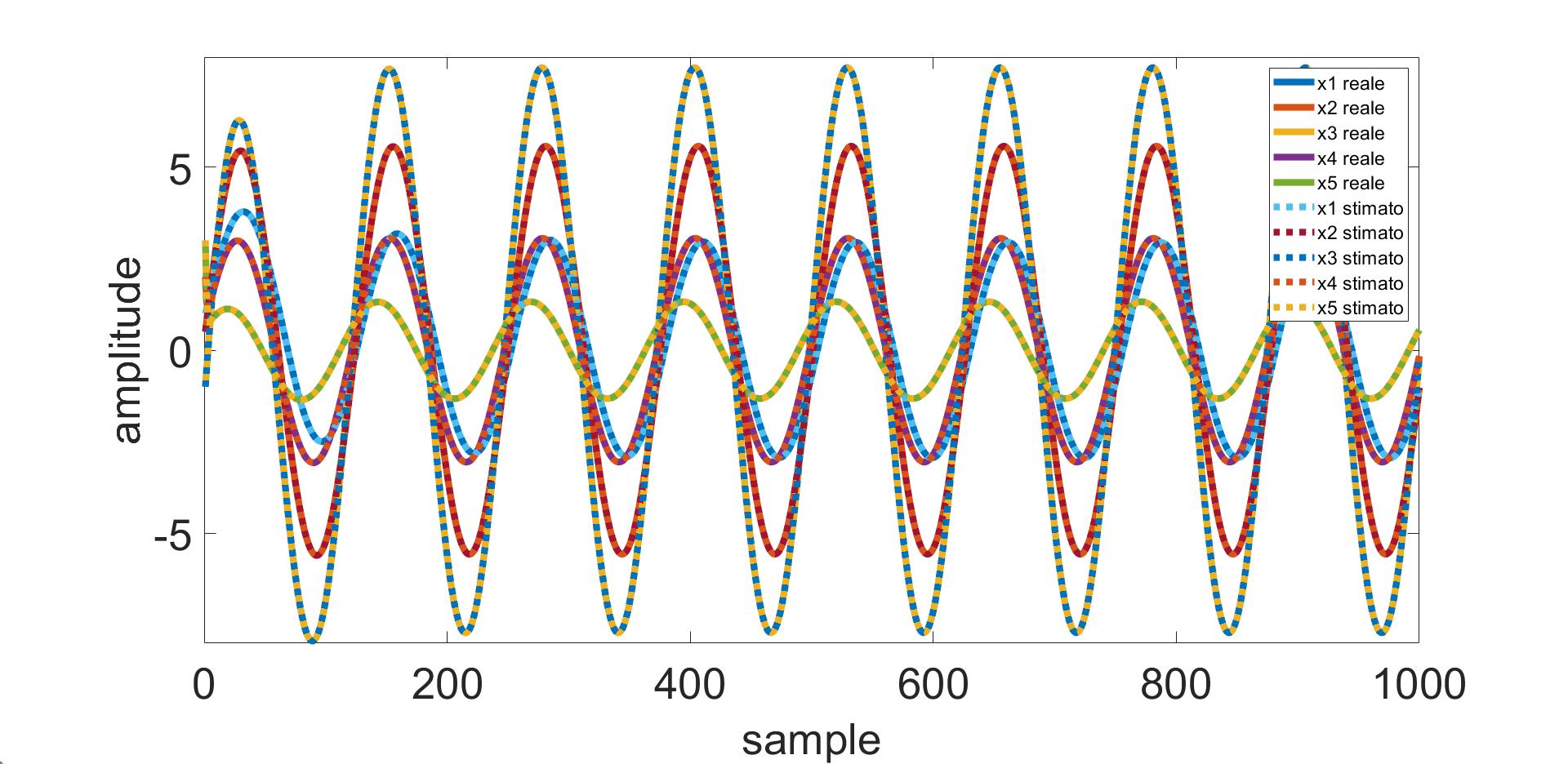
Nel seguente algoritmo, il 3.10, viene riportato i codice del DMD utilizzato.

|  |
| --- |
| function [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(X,Xprime,dt,lorenz)  if (lorenz==0)  thresh = 1e-10;  else thresh = 1e-1;  end  [U,Sigma,V] = svd(X,'econ'); % Step 1 con econ riduce il  % rango della matrice  r = length(find(diag(Sigma)>thresh)); % truncation value  Ur = U(:,1:r); % left singular vector matrix  Sigmar = Sigma(1:r,1:r); % singular values matrix  Vr = V(:,1:r); % right singular vector  % matrix  Atilde = Ur'\*Xprime\*Vr/Sigmar; % Step 2. It's the best-fit  % linear model  [W,Lambda] = eig(Atilde); % Step 3. They are the  % eigenvectors and the  % eigenvalues  Phi = Xprime\*(Vr/Sigmar)\*W;  lambda = diag(Lambda); % discrete-time eigenvalues  omega = log(lambda)/dt; % continuous-time eigenvalues  x1 = X(:, 1);  b = pinv(Phi)\*x1; % initial amplitude of each  % mode vector |

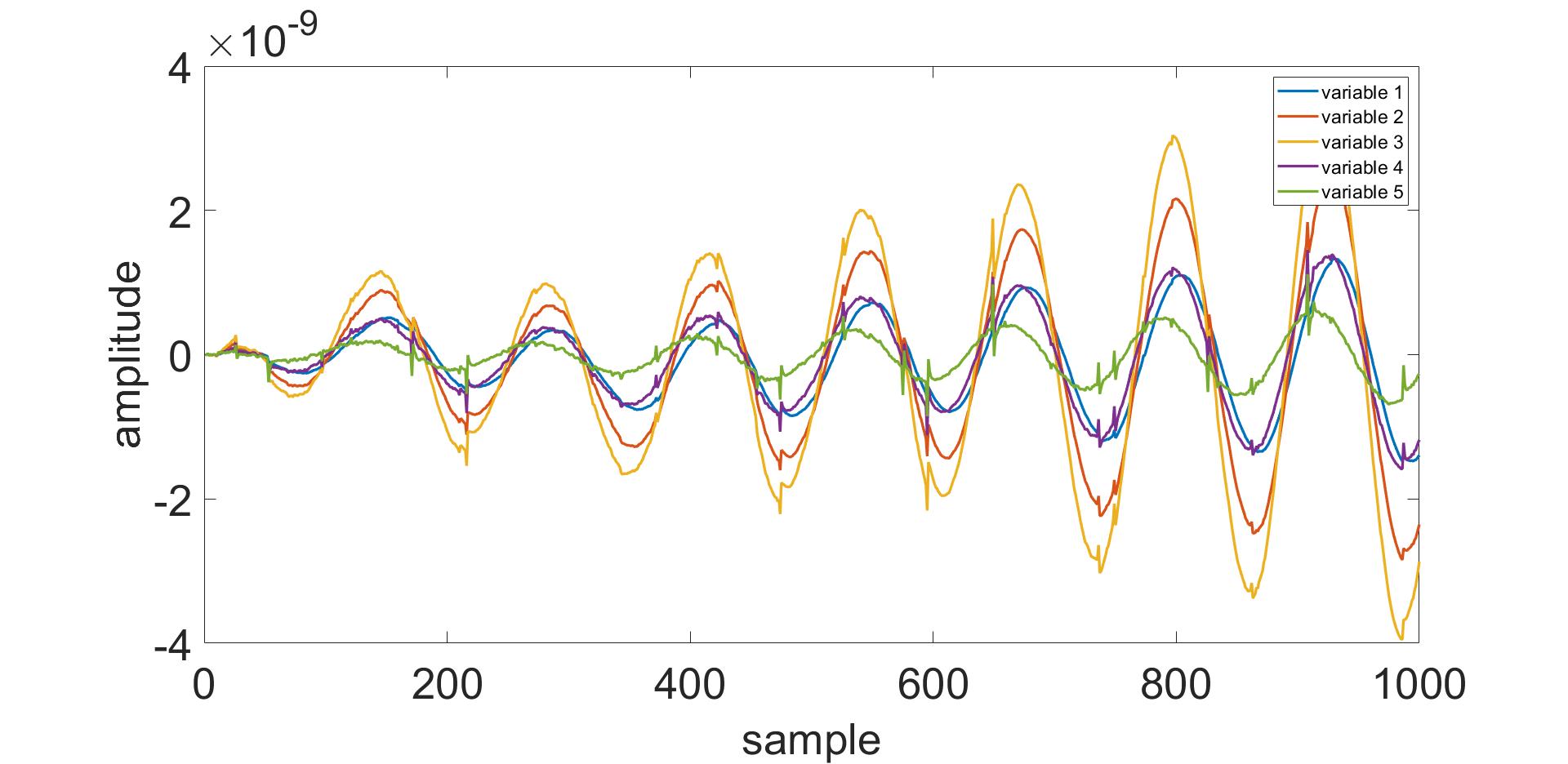
**Algoritmo 3.10** – Algoritmo DMD

Nel codice è presente la variabile lorenz che permette al sistema di impostare autonomamente la soglia che verrà usata per il calcolo della ‘r’ e nel caso di studio corrente è stato ottenuto un valore di r pari a 6.

Una volta eseguito tutto il codice, viene fatto un confronto tra le variabili del sistema reale e quelle generate dal DMDc, visibile nella Figura 3.8, e viene fatto anche il calcolo dell’errore su ogni singola variabile, presente all’interno della Figura 3.9.



**Figura 3.8** – Confronto tra il sistema reale e quello approssimato



**Figura 3.9** – Errore di ricostruzione per ogni variabile di stato

Guardando la Figura 3.9 si può notare che l’errore tende ad aumentare all’aumentare del tempo, quindi probabilmente le variabili approssimate tenderebbero ad allontanarsi dalla traiettoria del sistema reale e per ovviare a questo problema, viene introdotto il DMDc che utilizza la matrice di Hankel, presente nel prossimo caso di studio.

## Hankel DMDc con matrice d’ingresso diagonale

Con questo studio vengono uniti i codici del DMD con controllo e dell’Hankel DMD per creare un codice migliore dal punto di vista computazionale.

Inizialmente vengono dichiarate le variabili utili all’interno del codice, viene realizzato l’ingresso sinusoidale di controllo, viene creato il sistema tramite l’utilizzo di ode45 e infine viene fatta la trasposizione dei dati all’interno della matrice di Hankel. Tutto questo è presente nell’algoritmo 3.11.

|  |
| --- |
| clear all, close all  l = 0; % Lorenz variable  dt = 0.01;  tspan = 0:dt:10-dt;  A = [-1 0 0 0 0; 0 -2 0 0 0; 0 0 -3 0 0; % Parameter  0 0 0 -100 0; 0 0 0 0 -100];  B = [1 2 3 4 5]'; % Parameter  x0 = [1 .5 -1 2 3]; % Initial condition  g = transpose(3\*sin(tspan-0.25));  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)diagonal\_v3(t,x,A,B,g,tspan),tspan,x0,options);  %% hankel matrix  x = x.';  n = 20; % number of rows  m = length(tspan)-n; % number of columns  index1 = 1:n;  index2 = n:n+m-1;  X = []; Xprime=[];  for ir = 1:size(x,1)    % Hankel blocks ()  c = x(ir,index1).'; r = x(ir,index2);  H = hankel(c,r).';  c = x(ir,index1+1).'; r = x(ir,index2+1);  UH= hankel(c,r).';    X=[X,H]; Xprime=[Xprime,UH];  end  X=X';Xprime=Xprime'; |

**Algoritmo 3.11** - Dichiarazione variabili, creazione sistema e generazione matrice di Hankel

Prima di creare la matrice di Hankel vengono trasposti i dati in ingresso per la lavorazione e subito dopo il completamento del procedimento viene fatta una seconda trasposizione per riportare i dati in una forma che permetta l’esecuzione del DMD. Infatti il passaggio successivo è quello della creazione delle matrici per il DMD e successivamente avviene l’esecuzione vera e propria del DMD; tutto presente all’interno dell’algoritmo 3.12.

|  |
| --- |
| %% construction of the snapshot matrices  Ups = g(1:size(X,2),:)';  Omega = [X;Ups];  Omega1 = Omega(:,1:end-1);  OmegaPrime = Omega(:,2:end);  %% compute the svd of the input space Omega  [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(Omega1,OmegaPrime,dt,l); |

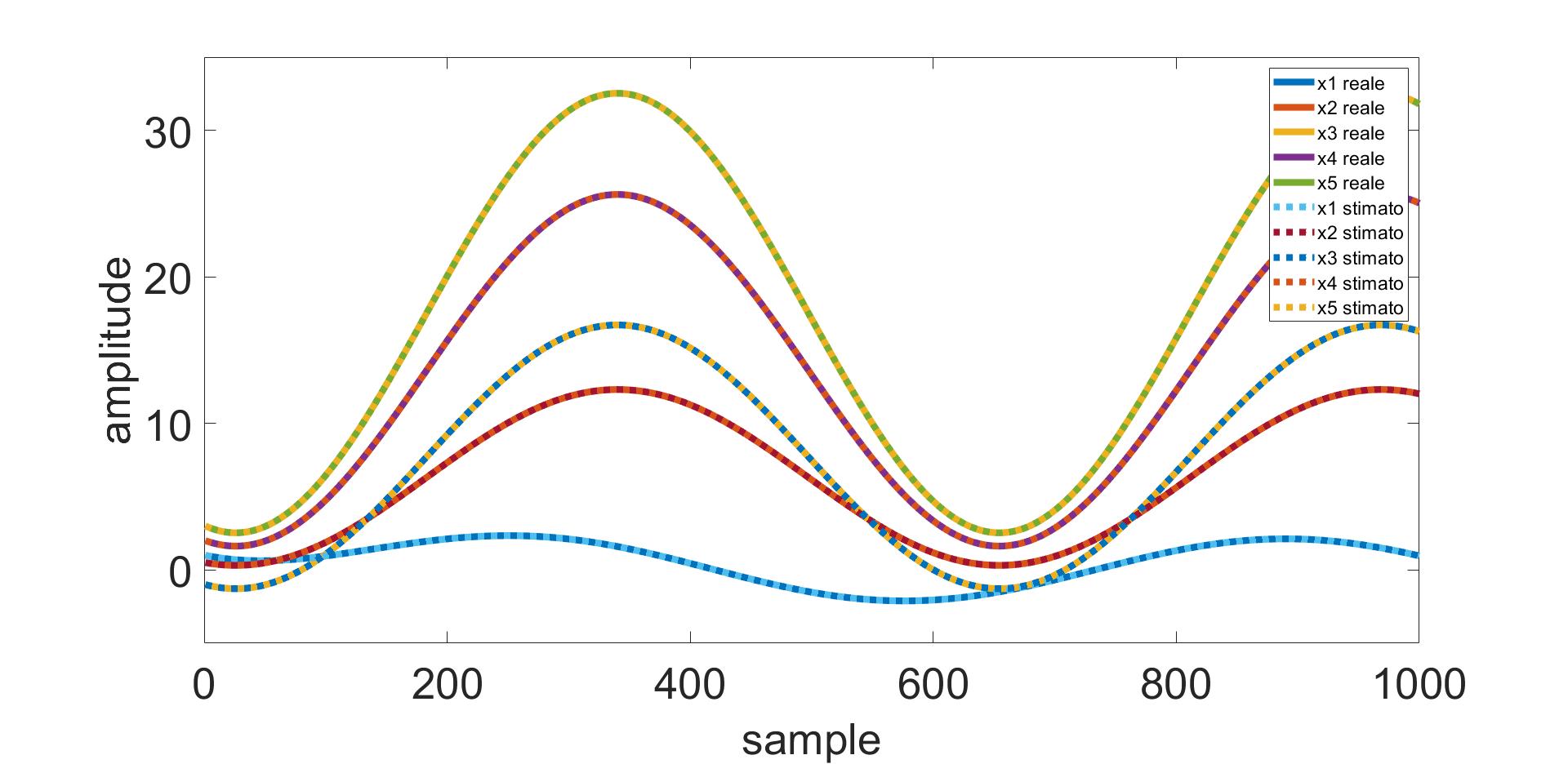
**Algoritmo 3.12** – Costruzione delle matrici e SVD di Omega

È stato ottenuto un valore di r pari a 209. Infine avviene la ricostruzione del sistema, visibile nell’algoritmo 3.13.

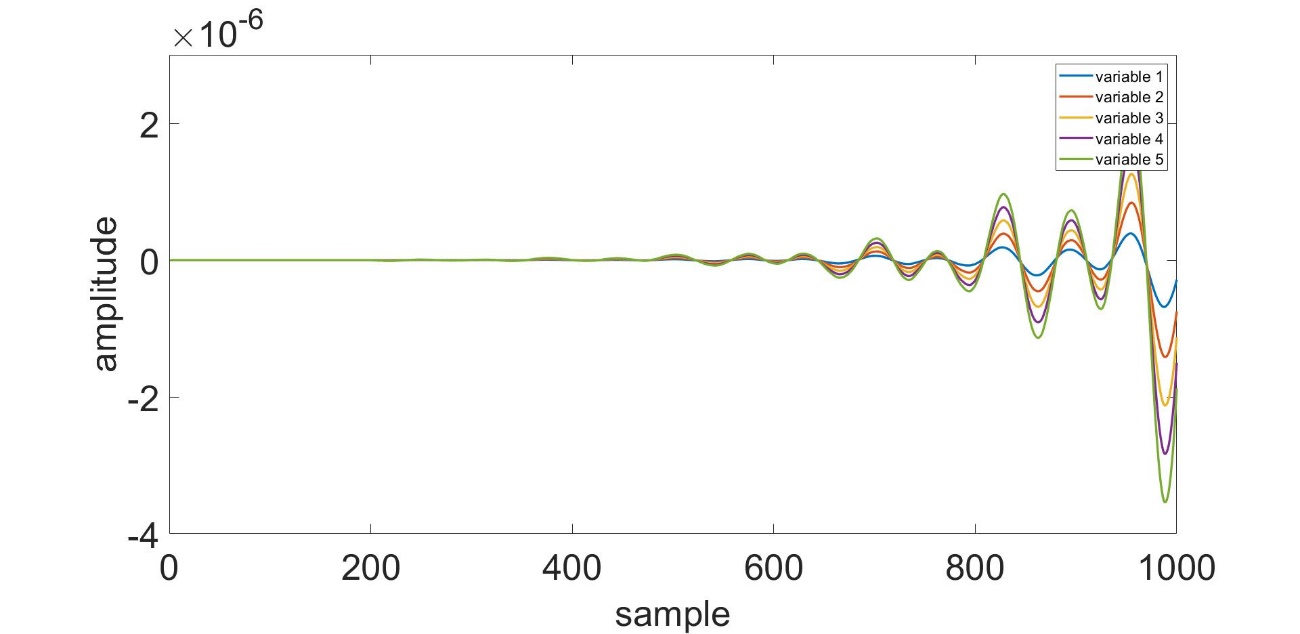
|  |
| --- |
| %% DMD reconstruction  time\_dynamics = zeros(r, 5000);  for iter = 1:5000%length(t)  time\_dynamics(:,iter) = (b.\*exp(omega\*t1(iter)));  end  Xdmd = real(Phi \* time\_dynamics);  Xdmdf = Xdmd(1:n:end-1,:); |

**Algoritmo 3.13** – Ricostruzione del sistema

Alla fine dell’algoritmo 3.13 le variabili del sistema ricostruito vengono riportate al numero iniziale, poiché in seguito all’aggiunta della variabile di controllo e all’utilizzo di Hankel erano aumentate notevolmente. Una volta eseguito interamente l’algoritmo si può passare al controllo dell’errore, il quale è visibile nelle prossime due figure, 3.10 e 3.11, dove viene mostrato il confronto tra il sistema reale e quello stimato nella prima e l’errore di ogni variabile approssimata rispetto alla corrispettiva reale nella seconda.



**Figura 3.10** – Confronto tra sistema reale e approssimato



**Figura 3.11** - Errore di ricostruzione per le cinque variabili di stato

Come si vede in Figura 3.11, si ha un errore sempre che aumenta con l’aumentare del tempo, ma la sua crescita adesso è molto minore rispetto al caso di studio in cui veniva utilizzato solo il DMDc senza la matrice di Hankel.

## Hankel DMDc con sistema di Lorenz

Con questo algoritmo è stato studiato il sistema di Lorenz utilizzando la tecnica DMD con l’aiuto del controllo e della matrice di Hankel. Verrà utilizzato il sistema di Lorenz con controllo, il cui codice è presente nell’algoritmo 2.13.

Prima di tutto vengono dichiarate le variabili del sistema e viene creato il controllo sinusoidale che poi sarà dato in ingresso al sistema di Lorenz controllato simulato tramite l’utilizzo di ode45. Dopo la fase iniziale, i dati vengono trasposti per permettere il modellamento di questi ultimi da parte delle matrici di Hankel e, alla fine del processo, vengono nuovamente trasposti per permettere l’esecuzione del DMD. Il tutto è mostrato nell’algoritmo 3.14.

|  |
| --- |
| clear all, close all  l = 1; % Lorenz variable  Beta = [8; 35; 8/3]; % chaotic values  x0 = [0; 1; 20]; % initial condition  dt = 0.001;  tspan = 0:dt:15-dt; % time span  omeg = 4.5; % input pulsation  a = 6; % input amplitude  g = (sin(omeg\*tspan));  options = odeset('RelTol',1e-12,'AbsTol',1e-12); % ode options  [t,x] = ode45(@(t,x)lorenz\_c(t,x,Beta,g,tspan,a),tspan,x0,options);  %% hankel matrix  x = x.';  n = 3000; % number of rows  m = length(tspan)-n; % number of columns  index1 = 1:n;  index2 = n:n+m-1;  X = []; Xprime=[];  for ir = 1:size(x,1)    % Hankel blocks ()  c = x(ir,index1).'; r = x(ir,index2);  H = hankel(c,r).';  c = x(ir,index1+1).'; r = x(ir,index2+1);  UH= hankel(c,r).';    X=[X,H]; Xprime=[Xprime,UH];  end  X=X';Xprime=Xprime'; |

**Algoritmo 3.14** – Dichiarazione variabili, creazione sistema e matrici di Hankel

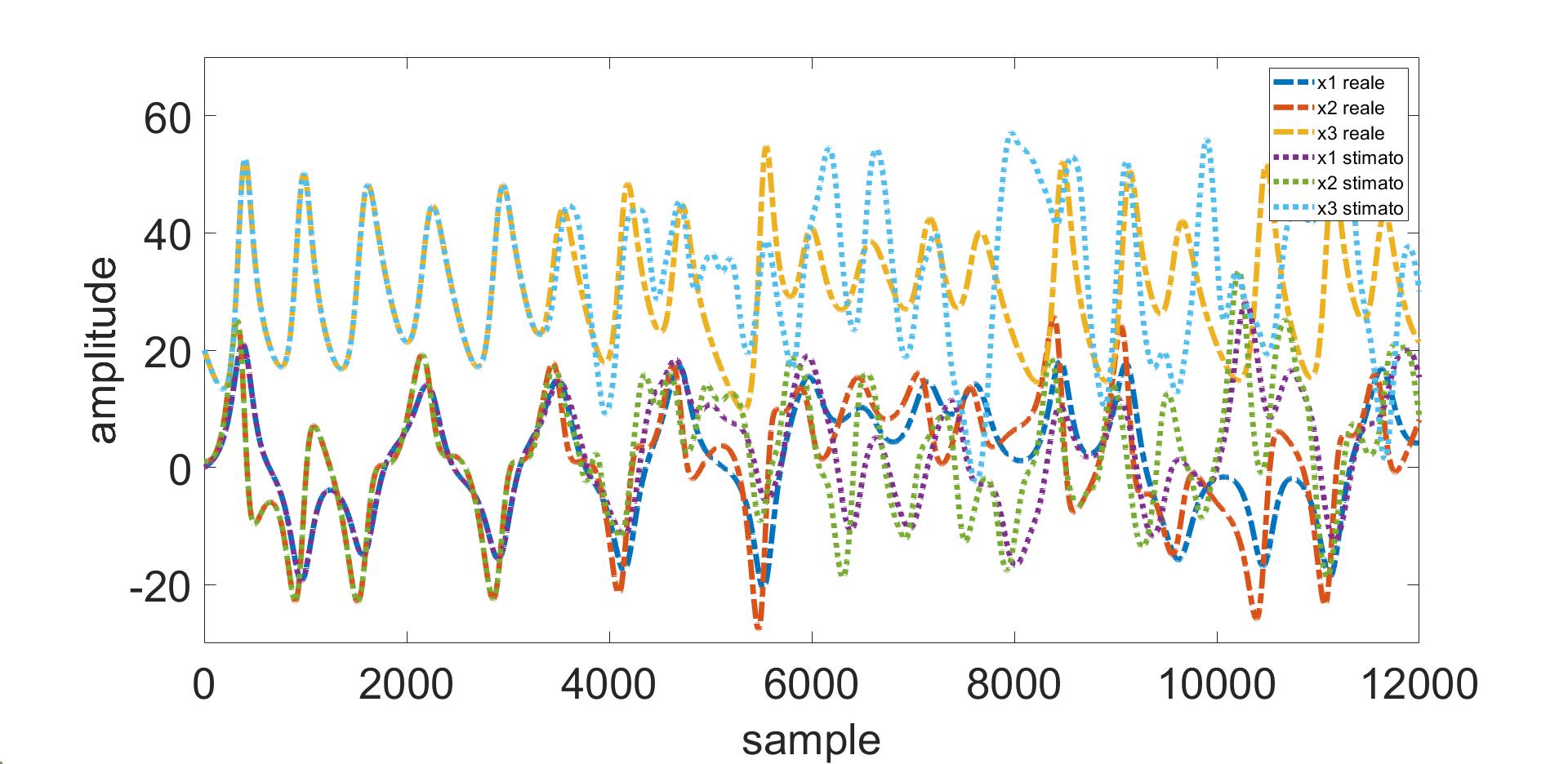
Come passaggio successivo si ha la costruzione delle matrici dei dati che serviranno nell’esecuzione del DMD, anche se viene creata solo la matrice del controllo perché X e Xprime sono già state create da Hankel e poi viene eseguita la ricostruzione del sistema approssimato. Questa parte di codice viene mostrata nell’algoritmo 3.15.

|  |
| --- |
| %% construction of the snapshot matrices  Ups = (g(:,1:size(X,2)));  Omega = [X;Ups];  Omega1 = Omega(:,1:end-1);  OmegaPrime = Omega(:,2:end);  %% compute the svd of the input space Omega  [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(Omega1,OmegaPrime,dt,l);  %% DMD reconstruction  time\_dynamics = zeros(r, m);  for iter = 1:m  time\_dynamics(:,iter) = (b.\*exp(omega\*t(iter)));  end  Xdmd = real(Phi \* time\_dynamics);  Xdmdf = Xdmd(1:n:end-1,:); |

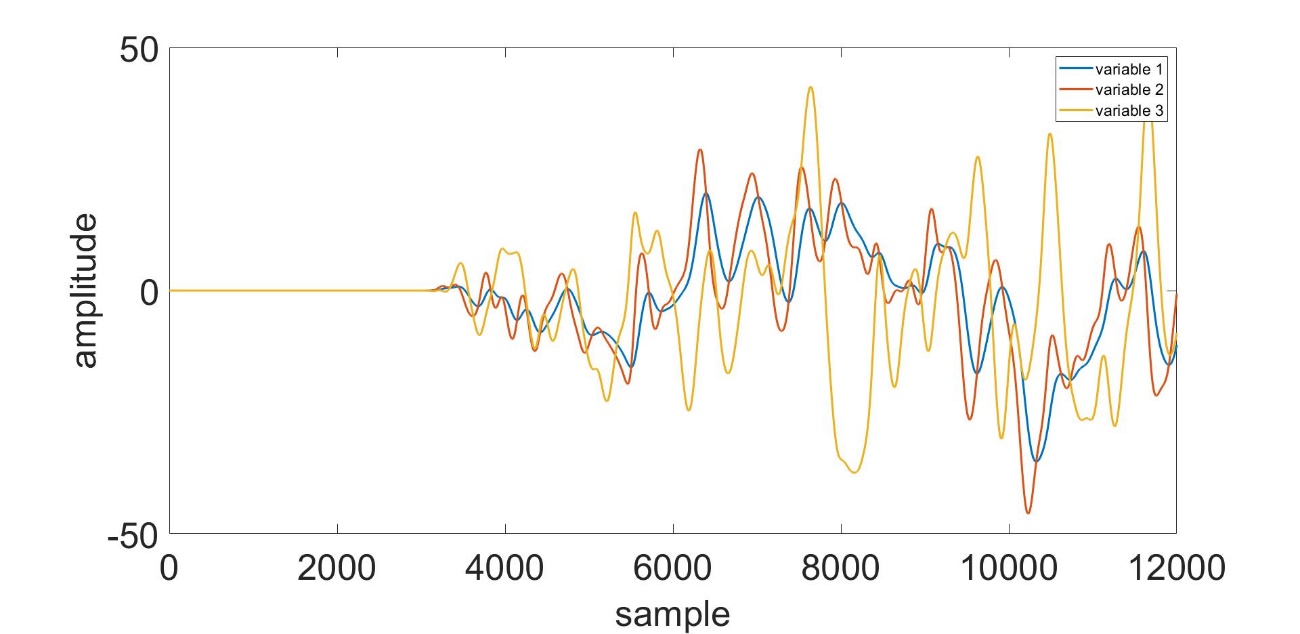
**Algoritmo 3.15** – Creazione delle matrici dei dati, SVD di Omega e ricostruzione del sistema

Alla fine dell’algoritmo 3.15 notiamo che, anche qui, le variabili sono state ridotte a seguito dell’introduzione del controllo e della matrice di Hankel, da cui ne è derivato un aumento e un successivo decremento.

Eseguito l’algoritmo, adesso vengono mostrati i risultati del sistema andando a fare il confronto tra il sistema reale e quello approssimato in Figura 3.12 e viene mostrato anche l’errore presente in ogni variabile stimata rispetto alle variabili del sistema reale in Figura 3.13.



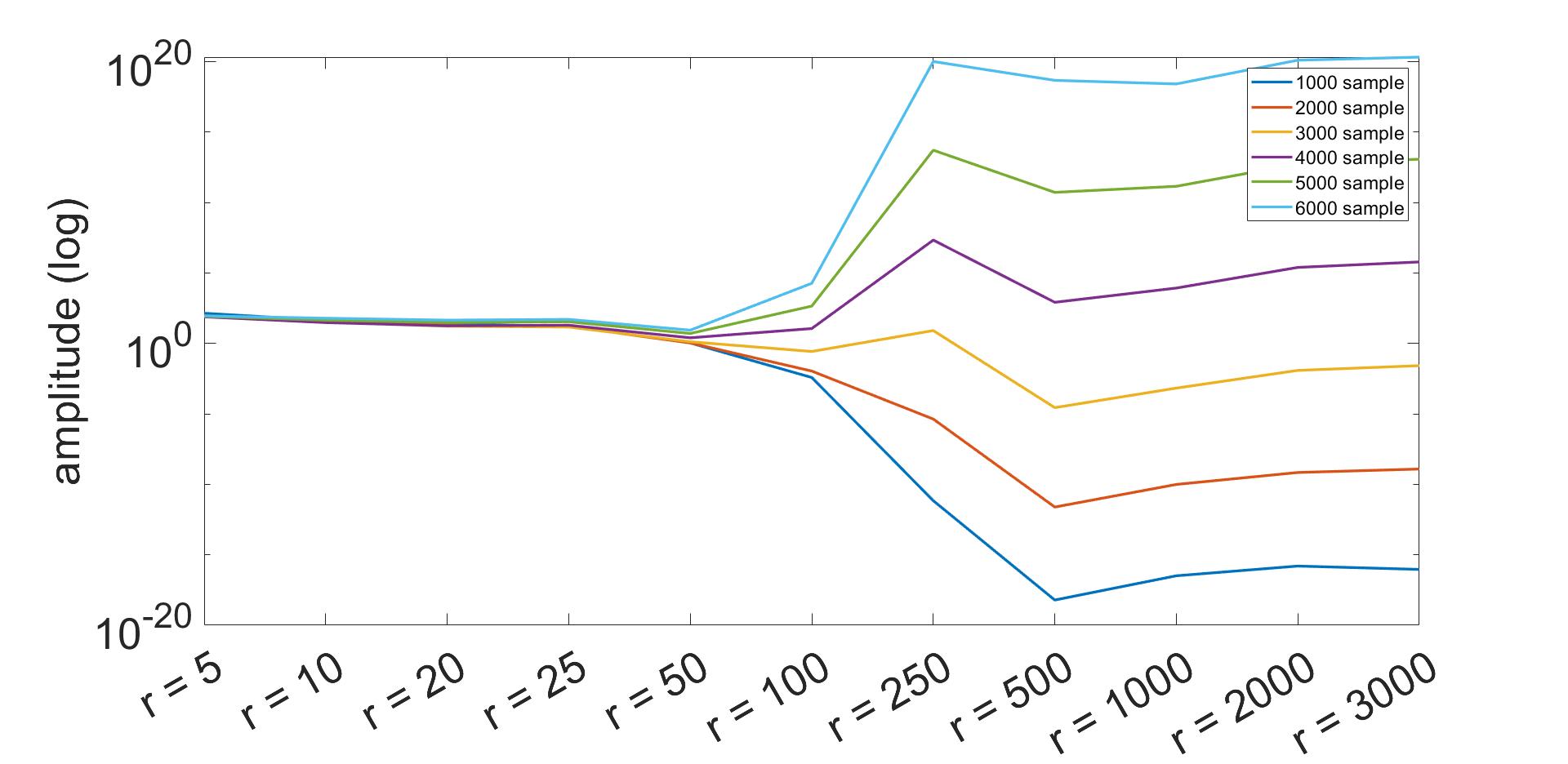
**Figura 3.12** – Confronto tra sistema reale e sistema approssimato con r = 335



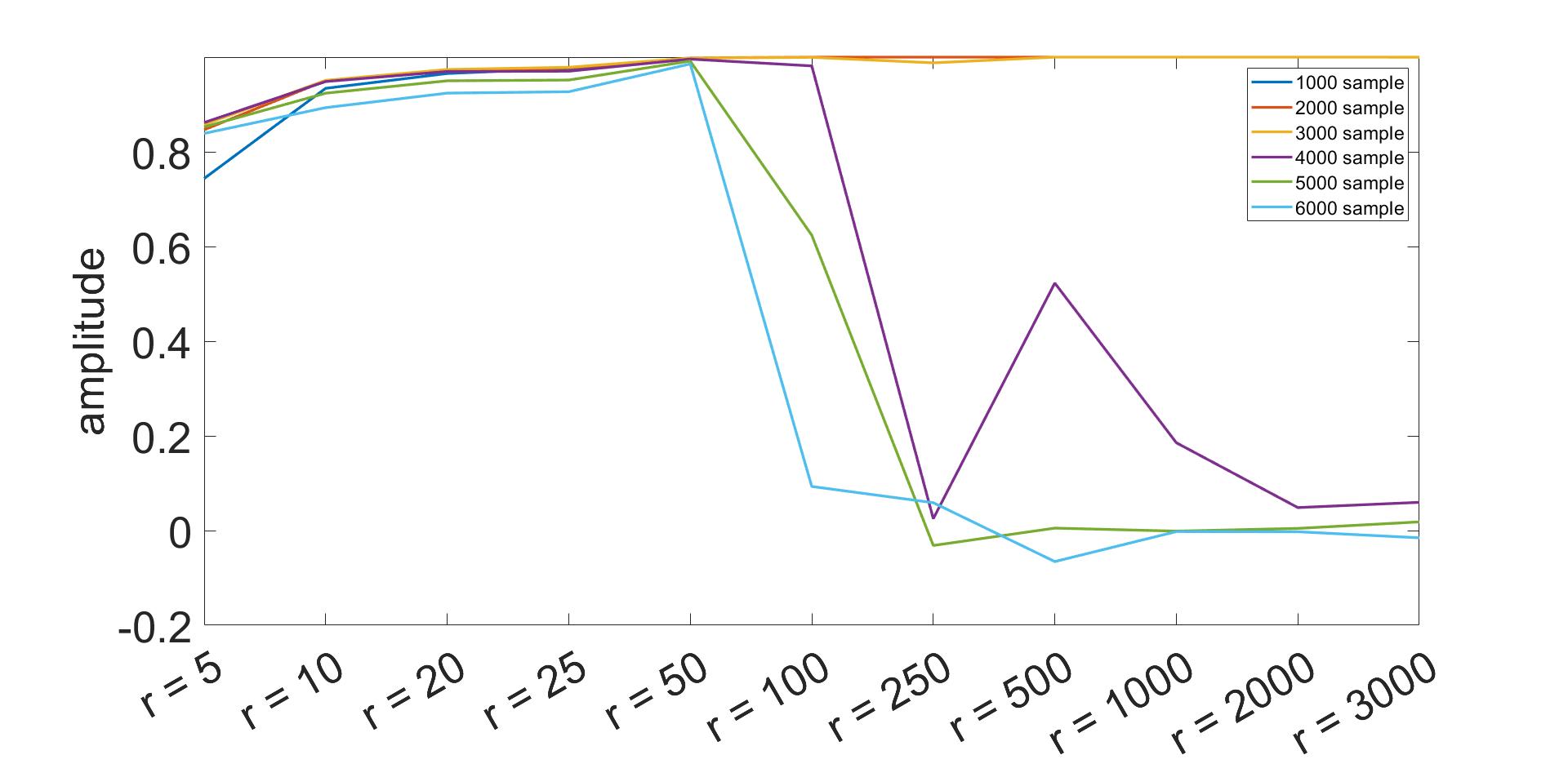
**Figura 3.13** – Errore di ricostruzione per le tre variabili di stato con r = 335

Il fatto che il sistema inizi ad avere un errore non trascurabile arrivato a circa 3000 istanti di tempo è un chiaro segnale che il DMD, sistema di approssimazione lineare, non basta per ricostruire un sistema non lineare e caotico come quello di Lorenz, anche con l’ausilio di Hankel e del controllo.

All’interno di questo caso di studio è stata fatta un’ulteriore operazione, cioè sono stati confrontati il sistema reale e quello approssimato tramite l’errore generato dalle variabili e il coefficienti di correlazione al variare di r, cioè il valore di troncamento per la riduzione della SVD del DMD, tra 5 e 3000; più precisamente l’algoritmo è stato iterato 11 volte e i valori di r sono: 5,10,20,25,50,100,250,500,1000,2000,3000. Nella Figura 3.14 viene mostrato l’errore di ricostruzione presente per le variabili al variare di r, ma anche dei campioni prelevati. Lo stesso procedimento è stato fatto per il coefficiente di correlazione e i risultati sono presenti all’interno della Figura 3.15.



**Figura 3.14**  - Errore di ricostruzione al variare di r e dei sample



**Figura 3.15** – Coefficiente di correlazione al variare di r e dei sample

Visionando entrambe le figure si capisce che il valore di errore minore e il valore di coefficiente di correlazione maggiore lo si trova eseguendo l’algoritmo con un valore di troncamento pari a 50, che rappresenta il valore ottimale di troncamento per il sistema di Lorenz esposto all’interno di [9].

## DMDc con processo industriale

In questo caso di studio è stata verificata la bontà dell’algoritmo DMDc utilizzando dati provenienti da un processo industriale.

Come prima cosa vengono caricati i dati del sistema e del controllo in ingresso, poi vengono dichiarate le variabili che serviranno nel codice e vengono costruite le matrici dei dati per l’esecuzione del DMD. Questa parte di codice è mostrata nell’algoritmo 3.16.

|  |
| --- |
| clear all, close all  load SRU\_l2\_l4\_Dataset  Data.l4\_in\_n=zscore(l4\_in);  Data.l4\_out\_n=zscore(l4\_out);  %% Dichiarazione variabili  dt = 0.01; % Time step  l = 0; % Lorenz variable  tspan = 0:dt:110;  tr = size(Data.l4\_out\_n,1);  tspan = tspan(1:tr);  %% construction of the snapshot matrices  X = transpose(Data.l4\_out\_n(1:end-1,:));  Xprime = transpose(Data.l4\_out\_n(2:end,:));  Ups = transpose(Data.l4\_in\_n(1:end-1,:));  Omega = [X;Ups]; |

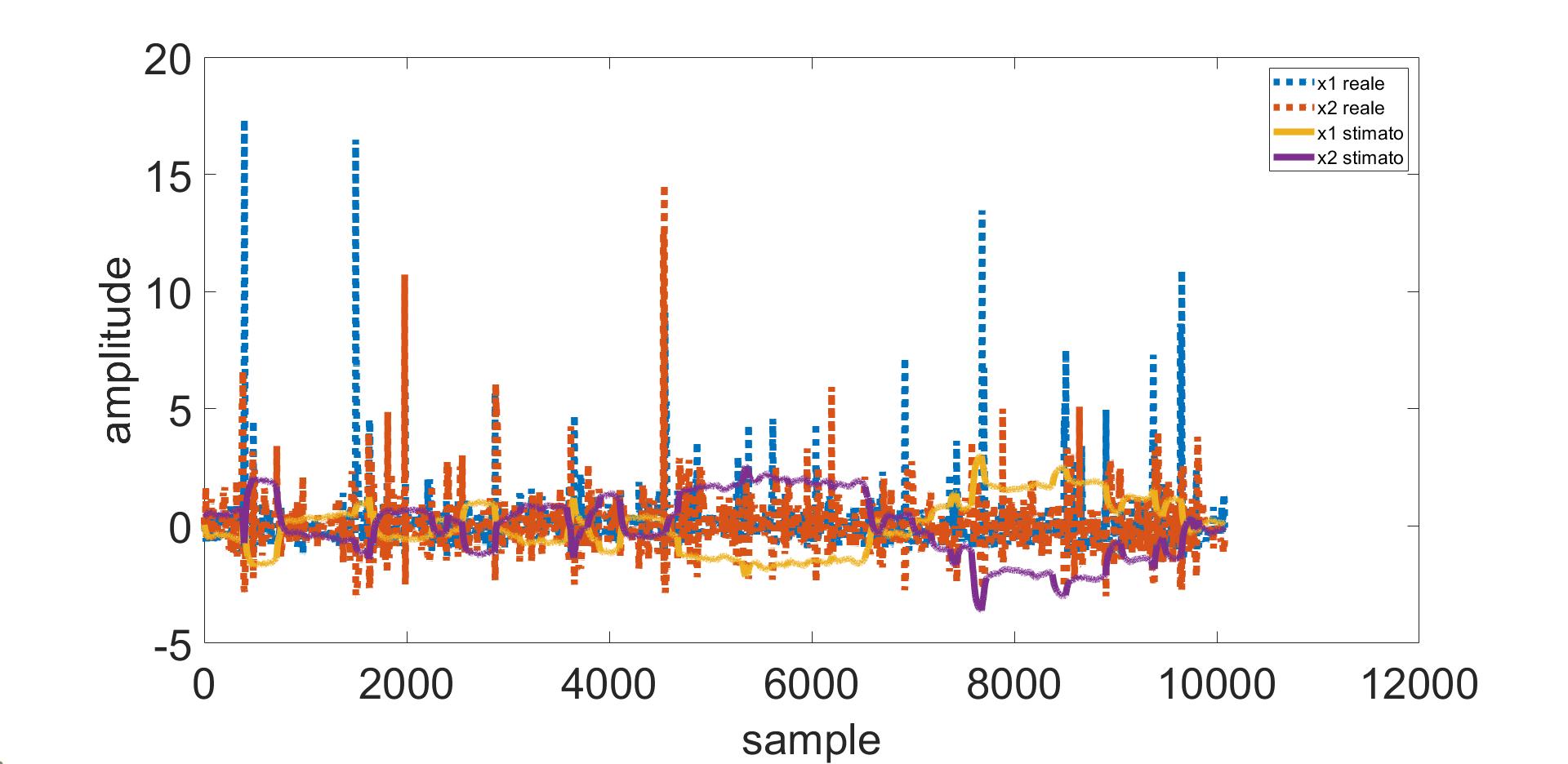
**Algoritmo 3.16** – Caricamento dati, dichiarazione variabili e creazione matrici

Successivamente viene calcolata la SVD delle matrici Omega e Xprime, viene approssimato l’operatore G e viene ricostruito il sistema stimato. Il codice di questi passaggi è presente nell’algoritmo 3.17.

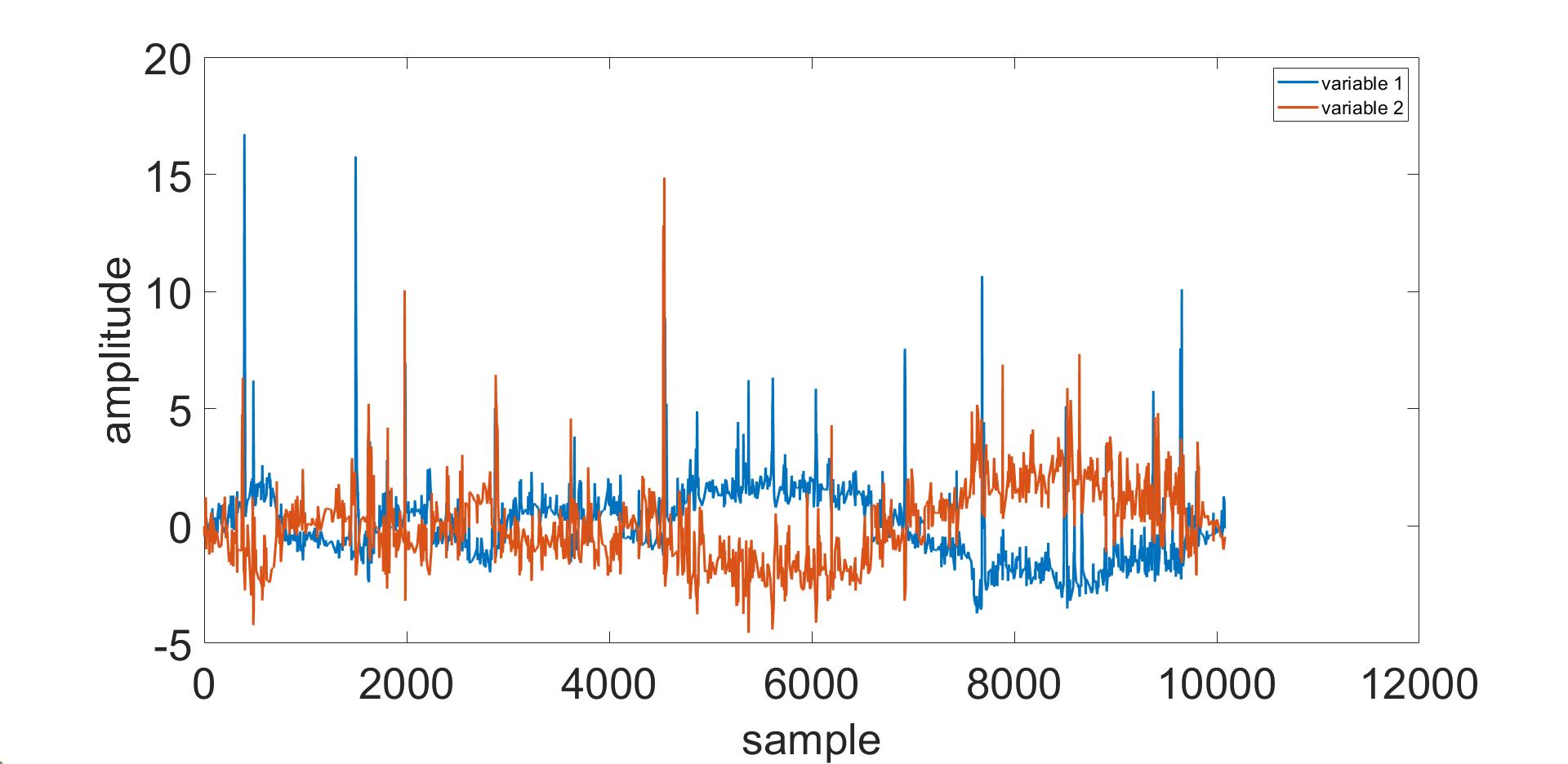
|  |
| --- |
| %% compute the svd of the input space Omega  [~,Ur,Sigmar,Vr] = DMDc(Omega,l);  %% compute the svd of the output space Xprime  [~,Uhat,Sighat,Vhat] = DMDc(Xprime,l);  %% compute the approximation of the operator G = [A B]  n = size(X,1); % length of the first dimension of X  q = size(Ups,1); % length of the first dimension of Ups  U\_1 = Ur(1:n,:);  U\_2 = Ur(n+q:n+q,:);  %% Evoluzione sistema discreto  approxAd = (Xprime)\*Vr\*inv(Sigmar)\*U\_1’;  approxBd = (Xprime)\*Vr\*inv(Sigmar)\*U\_2’;  approxX(:,1)=Data.l4\_out\_n(1,:);  for k=1:10080  approxX(:,k+1)=approxAd\*approxX(:,k)+approxBd\*Data.l4\_in\_n(k);  end |

**Algoritmo 3.17** – SVD di Omega e Xprime, approssimazione di G e ricostruzione del sistema

Eseguito l’algoritmo, adesso vengono mostrati i risultati del sistema andando a fare il confronto tra il sistema reale e quello approssimato in Figura 3.16 e viene mostrato anche l’errore presente in ogni variabile stimata rispetto alle variabili del sistema reale in Figura 3.17.



**Figura 3.16** – Confronto tra sistema reale e approssimato con r = 7



**Figura 3.17** – Errore di ricostruzione presente in ogni variabile con r = 7

Nella Figura 3.16 è molto chiaro il fatto che il DMDc non riesca a seguire a pieno l’andamento del sistema reale. Per questo deve essere provato l’Hankel DMDc per scoprire se viene ottenuta un’approssimazione migliore.

## Hankel DMDc con processo industriale

In questo caso di studio è stato verificato l’algoritmo Hankel DMDc utilizzando dati provenienti da un processo industriale.

Come prima cosa vengono caricati i dati del sistema e del controllo in ingresso, poi vengono dichiarate le variabili che serviranno nel codice e viene creata la matrice di Hankel. Questa parte di codice è mostrata nell’algoritmo 3.18.

|  |
| --- |
| clear all, close all  load SRU\_l2\_l4\_Dataset  Data.l4\_in\_n=zscore(l4\_in);  Data.l4\_out\_n=zscore(l4\_out);  %% Declaration variable  dt = 0.01; % Time step  l = 0;  tspan = 0:dt:110;  tr = size(Data.l4\_out\_n,1);  tspan = tspan(1:tr);  x0 = Data.l4\_out\_n(1,:);  %% hankel matrix  Data.l4\_out\_n = Data.l4\_out\_n.';  n = 2000; % number of rows  m = length(tspan)-n; % number of columns  index1 = 1:n;  index2 = n:n+m-1;  X = []; Xprime=[];  for ir = 1:size(Data.l4\_out\_n,1)    % Hankel blocks ()  c = Data.l4\_out\_n(ir,index1).'; r = Data.l4\_out\_n(ir,index2);  H = hankel(c,r).';  c = Data.l4\_out\_n(ir,index1+1).'; r = Data.l4\_out\_n(ir,index2+1);  UH= hankel(c,r).';    X=[X,H]; Xprime=[Xprime,UH];  end  X=X';Xprime=Xprime'; |

**Algoritmo 3.18** – Caricamento dati, dichiarazione variabili e creazione matrice di Hankel

Come in ogni algoritmo che sfrutta Hankel, prima della creazione della matrice di Hankel i dati sono stati trasposti e subito dopo la creazione è avvenuta una seconda trasposizione per permettere la corretta esecuzione del DMD.

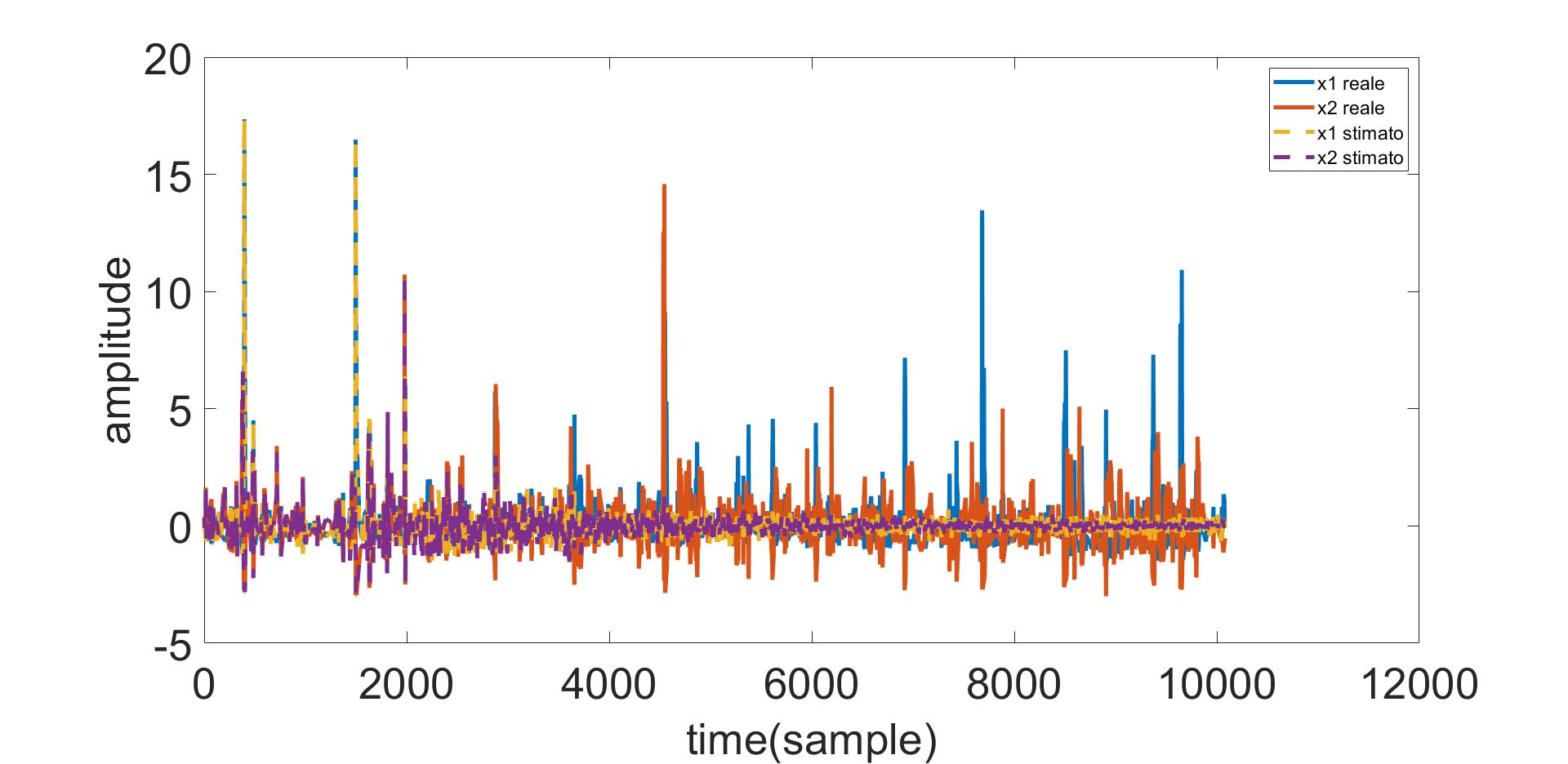
Successivamente vengono create le matrici per il DMD, viene calcolata la SVD sulla matrice Omega e infine viene eseguita la ricostruzione del sistema approssimato. Il codice di questa parte lo si trova all’interno dell’algoritmo 3.19.

|  |
| --- |
| %% construction of the snapshot matrices  Ups = Data.l4\_in\_n(1:size(X,2),:)';  Omega = [X;Ups];  Omega1 = Omega(:,1:end-1);  OmegaPrime = Omega(:,2:end);  %% compute the svd of the input space Omega  [Phi,b,omega,r,Ur,Sigmar,Vr] = DMD(Omega1,OmegaPrime,dt,l);  %% Recostruction  time\_dynamics = zeros(r, length(tspan));  for iter = 1:length(tspan)  time\_dynamics(:,iter) = (b.\*exp(omega\*tspan(iter)));  end  Xdmd = real(Phi \* time\_dynamics);  Xdmdf = [Xdmd(1,:)',Xdmd(n+1,:)']; |

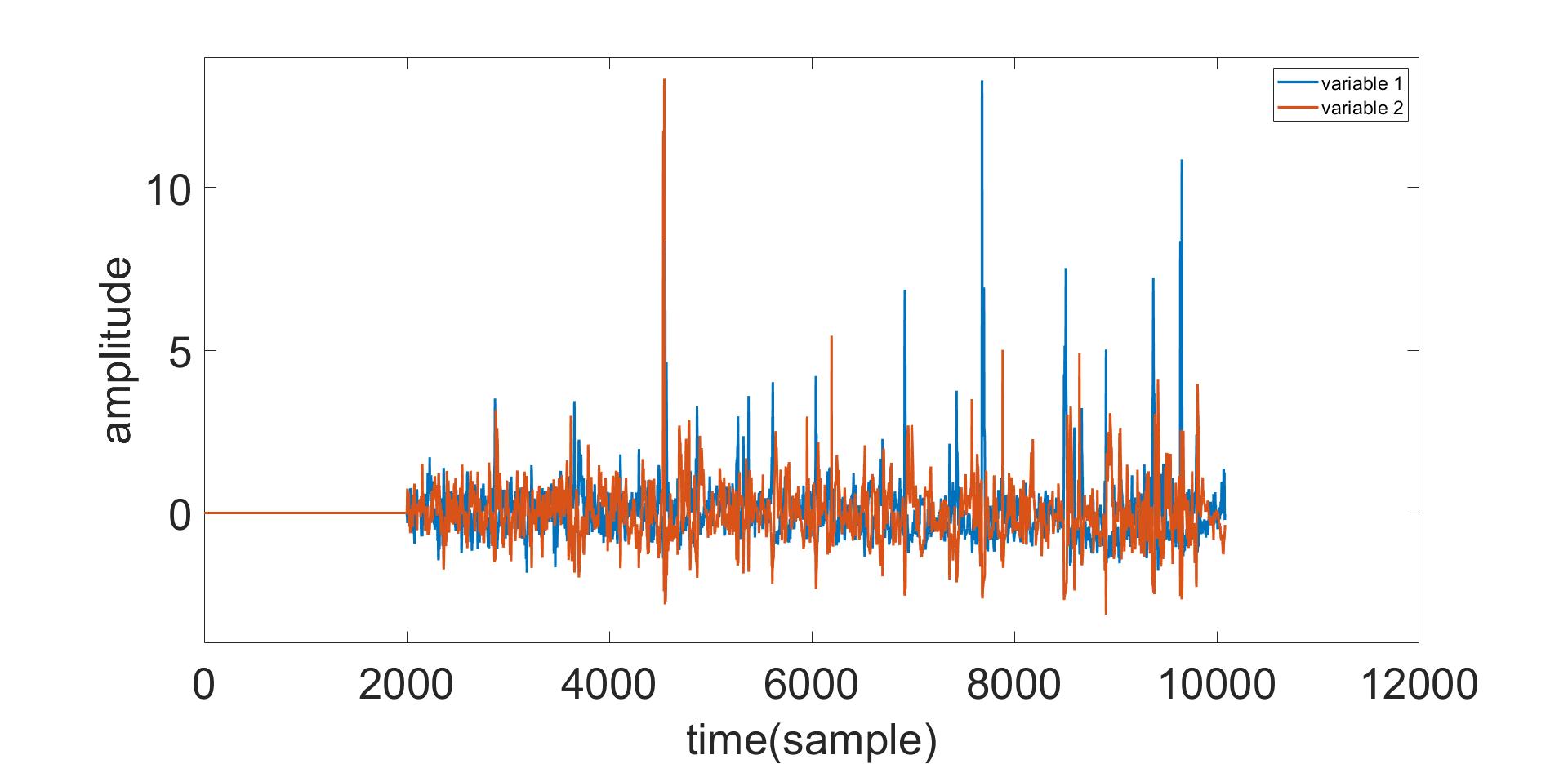
**Algoritmo 3.19** – Creazione matrici, calcolo SVD di Omega e ricostruzione sistema

Per il calcolo dell’errore, il risultato della ricostruzione è stato ridotto per equipararlo con la dimensione dei dati del processo industriale.

Eseguito l’algoritmo, adesso vengono mostrati i risultati del sistema andando a fare il confronto tra il sistema reale e quello approssimato in Figura 3.19 e viene mostrato anche l’errore presente in ogni variabile stimata rispetto alle variabili del sistema reale in Figura 3.19.



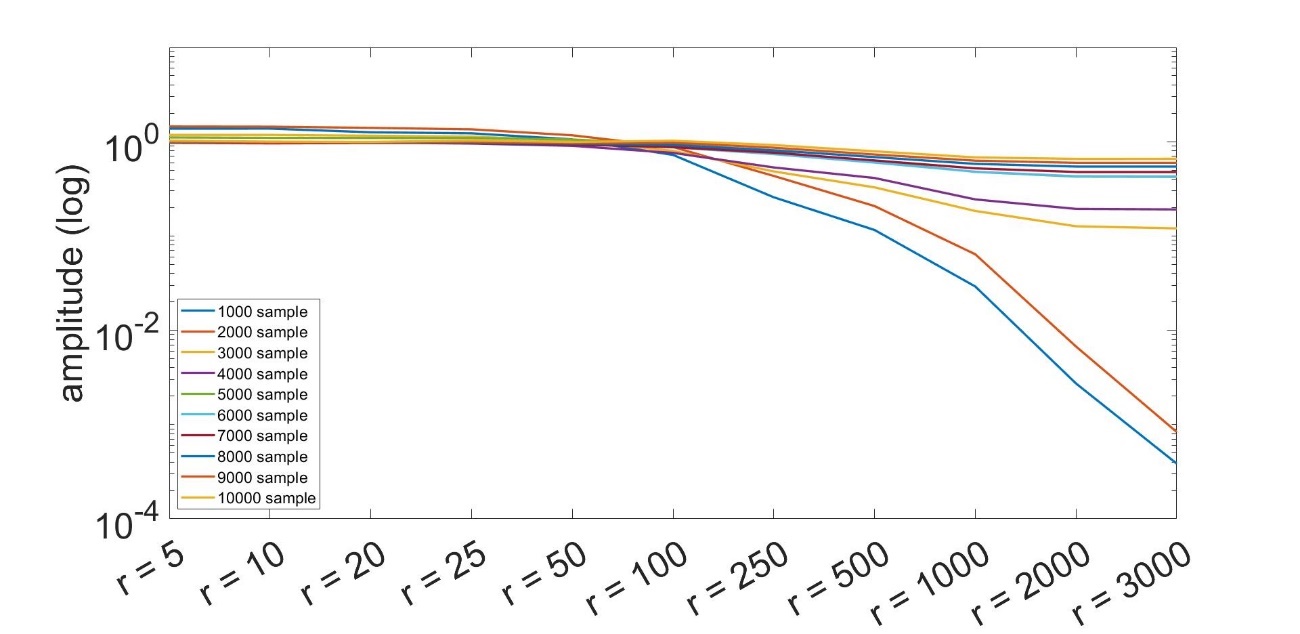
**Figura 3.18** – Confronto tra sistema reale e approssimato con r = 4005



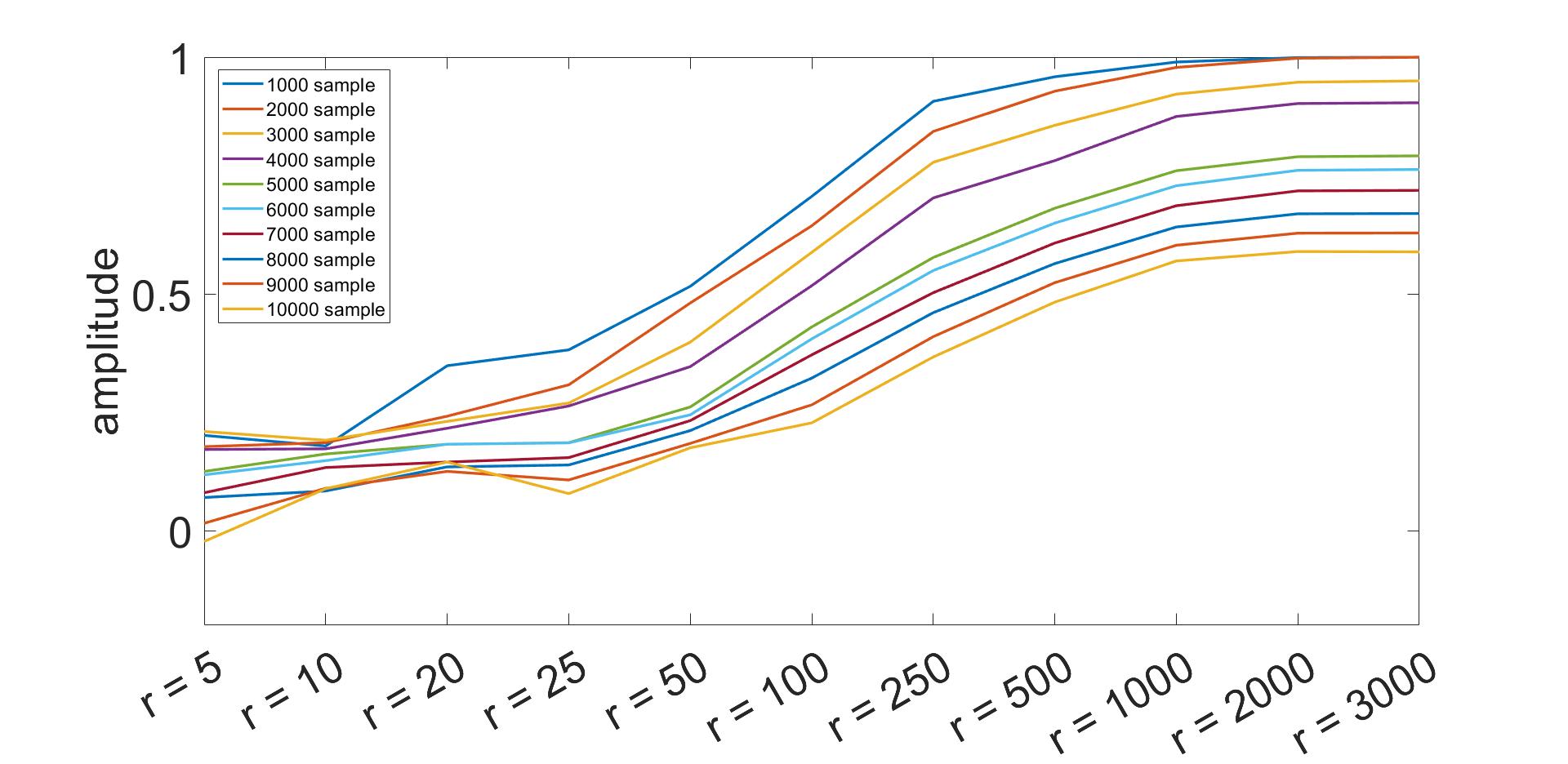
**Figura 3.19** – Errore di ricostruzione presente in ogni variabile con r = 4005

In questo caso l’Hankel DMDc riesce a costruire un sistema che segue perfettamente il sistema del processo industriale per i primi 2000 campioni, ma subito dopo avviene un aumento dell’errore che porta sempre più il sistema ricostruito a discostarsi dal sistema reale.

Anche all’interno di questo caso di studio è stata fatta un’ulteriore operazione, cioè sono stati confrontati i dati del processo industriale e quelli del sistema ricostruito tramite l’errore generato dalle variabili e i coefficienti di correlazione al variare di r, cioè il valore di troncamento per la riduzione della SVD del DMD, tra 5 e 3000; più precisamente l’algoritmo è stato iterato 11 volte e i valori di r sono: 5,10,20,25,50,100,250,500,1000,2000,3000. Nella Figura 3.20 viene mostrato l’errore di ricostruzione presente per le variabili al variare di r, ma anche dei campioni prelevati. Lo stesso procedimento è stato fatto per il coefficiente di correlazione e i risultati sono presenti all’interno della Figura 3.21.



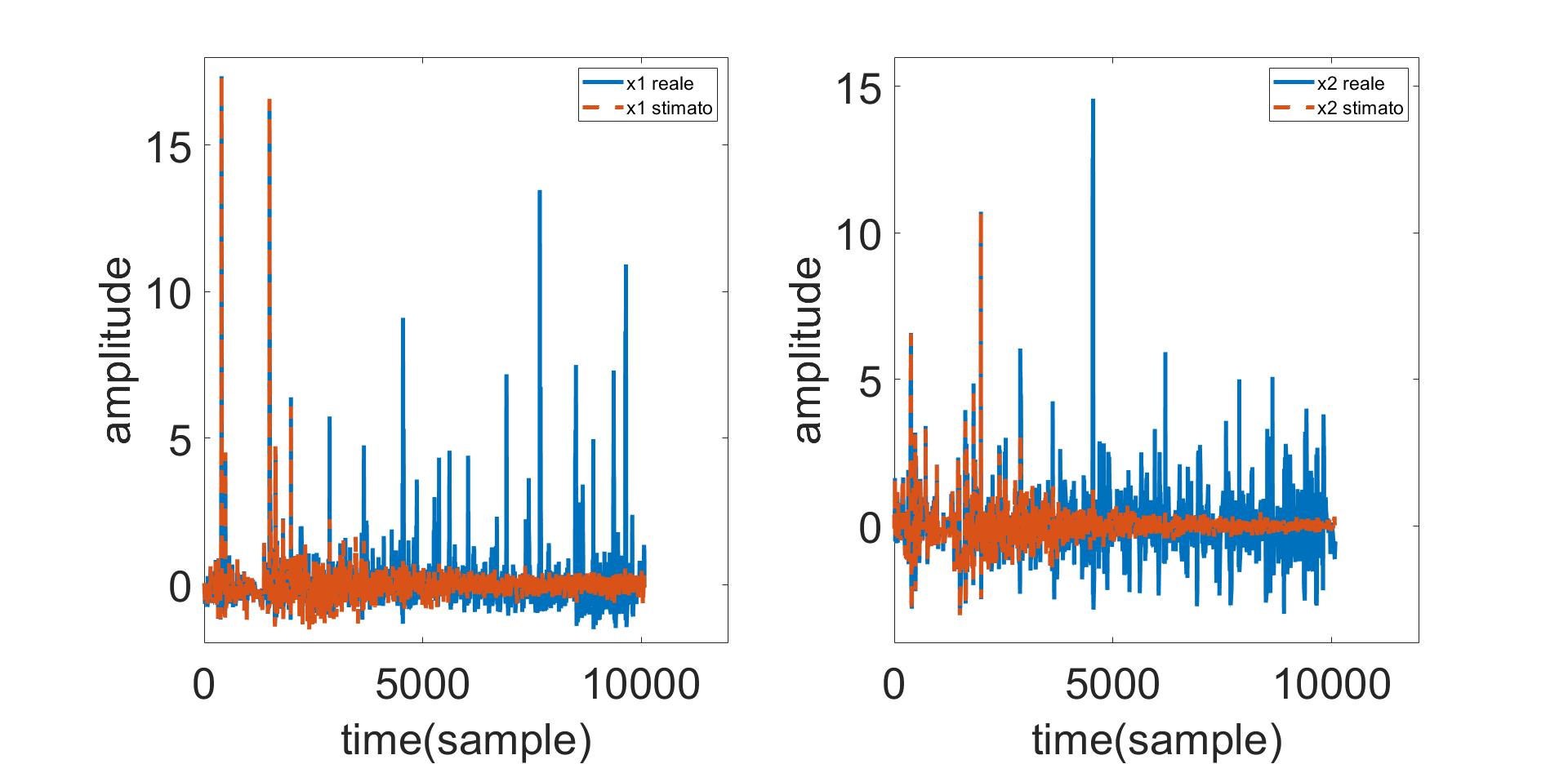
**Figura 3.20**  - Errore di ricostruzione al variare di r e dei sample



**Figura 3.21** – Coefficiente di correlazione al variare di r e dei sample

Nella Figura 3.20 si nota che, tralasciando l’errore presente solo nei primi 4000 campioni, l’errore totale rimane pressoché stabile per tutti i valori di r provati e, andando a guardare la Figura 3.22, si nota che il coefficiente è sempre maggiore all’aumentare del valore di r, per cui, sicuramente una condizione ottimale sarà quella di avere un r pari a 2000 o 3000. Se invece si tiene conto di tutti gli errori al variare dei sample presi in considerazione, una buona stima del segnale ricostruito la si ha per r pari a 100, che rappresenta l’ultimo valore prima che gli errori peggiorino drasticamente.

Di seguito viene mostrata la Figura 3.22 dove avviene il confronto, per r pari a 3000, tra ogni variabile del sistema reale e di quello approssimato.



**Figura 3.22** – Confronto tra sistema reale e approssimato con r pari a 3000

# SINDy: casi di studio

Di seguito saranno esposte vari applicazioni del SINDy nelle sue due versioni: nativo e con il controllo o MPC.

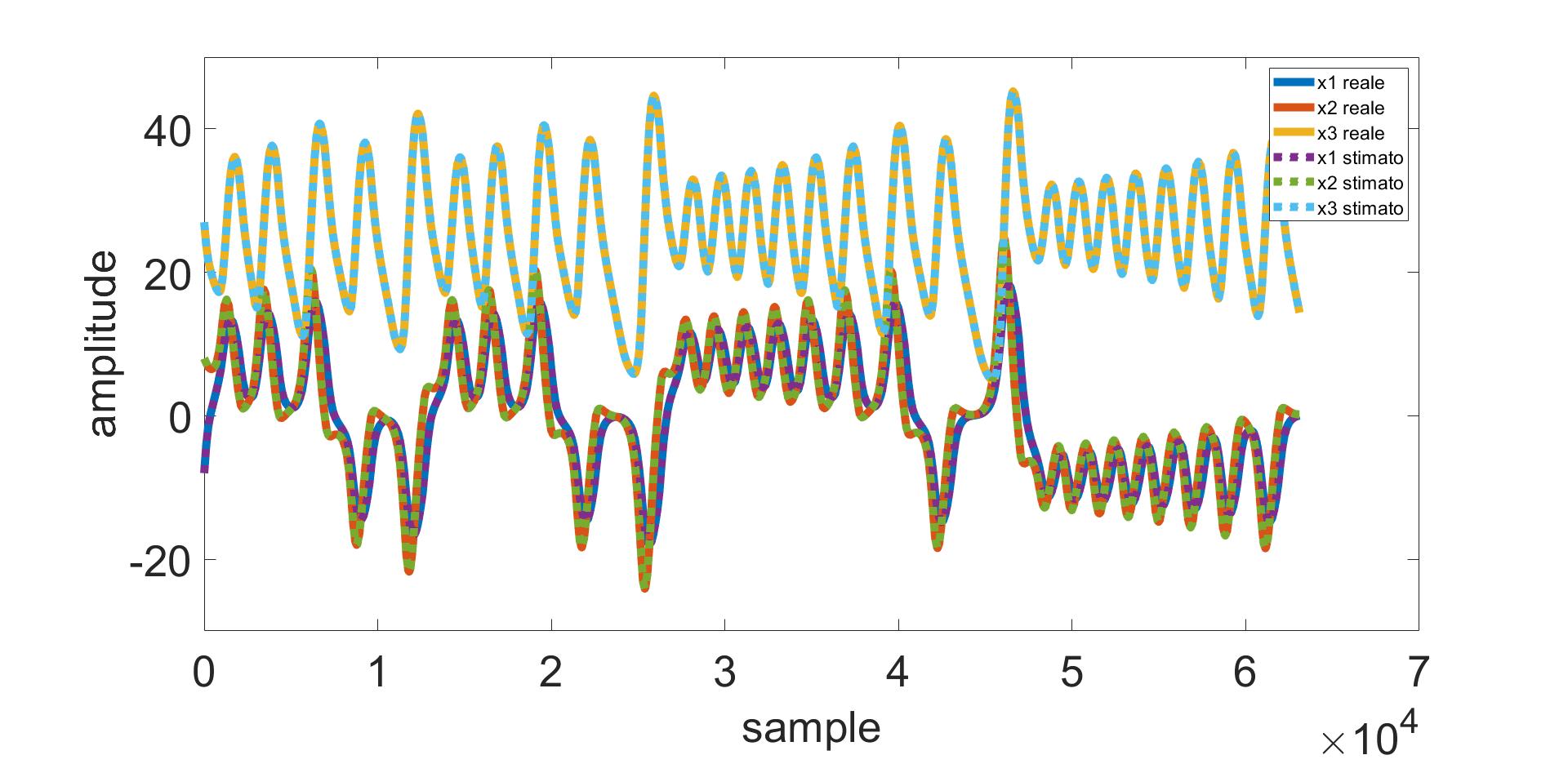
## SINDy nativo con sistema di Lorenz

In questo caso di studio viene computato il sistema di Lorenz utilizzando il SINDy nativo. Inizialmente vengono generati i dati del sistema di Lorenz e dichiarate le variabili che verranno utilizzate nel codice, poi si calcolano le derivate del sistema e da qui in poi ha inizio l’algoritmo in se, poiché viene generata la libreria delle funzioni non lineari candidate (in questo caso solo polinomiale in quanto il sistema di Lorenz è un sistema composto da solo equazioni differenziali polinomiali) tramite la funzione poolData presente nell’algoritmo 2.9, viene eseguito l’algoritmo di Sequential Threshold Least-Squares tramite la funzione sparsifyDynamics presente nell’algoritmo 2.7, che restituisce i valori che andranno a comporre la libreria dei termini utilizzati. Infine, dopo tutti questi passaggi, viene ricostruito il sistema utilizzando ode45 e la funzione nativa di SINDy chiamata sparseGalerkin. Il codice viene mostrato nell’algoritmo 4.1.

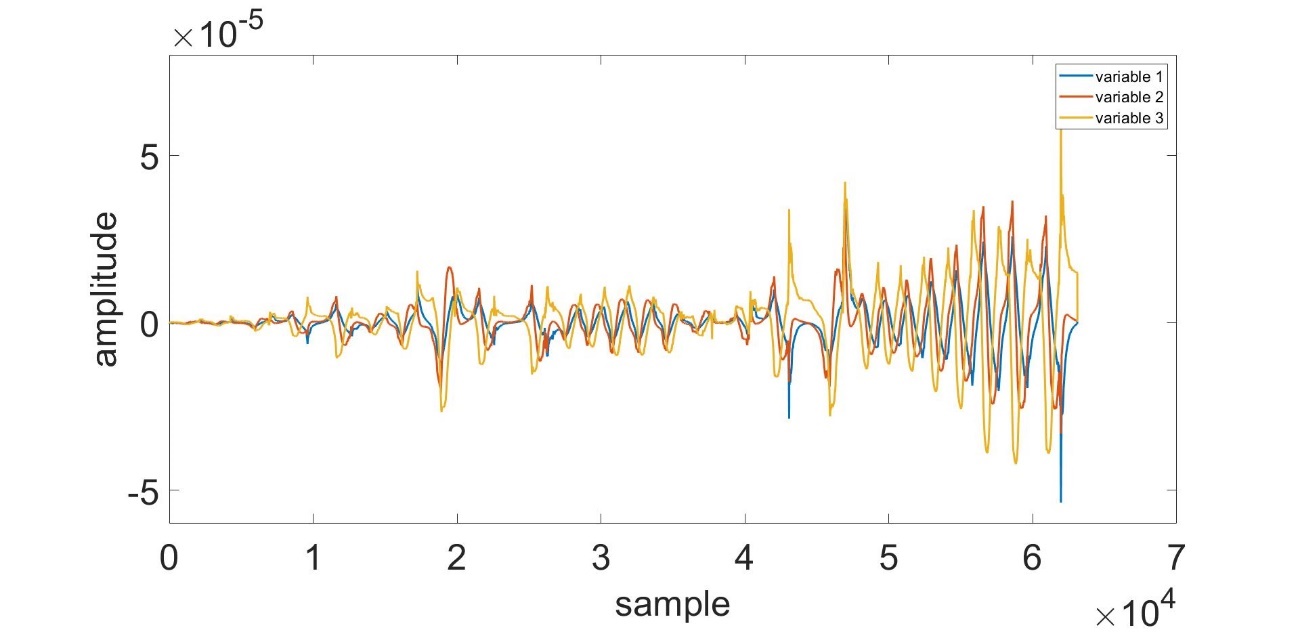
|  |
| --- |
| clear all, close all, clc  %% Generate Data  usesine = 0; % Using sine functions in library  Beta = [10; 28; 8/3]; % Lorenz's parameters (chaotic)  n = 3; % variables number  x0=[-8; 8; 27]; % Initial condition  dt = 0.01;  tspan=[0.01:dt:50];  options = odeset('RelTol',1e-12,'AbsTol',1e-12\*ones(1,n));  [t,x]=ode45(@(t,x) lorenz(t,x,Beta),tspan,x0,options);  %% Compute Derivative  for i=1:length(x)  dx(i,:) = lorenz(0,x(i,:),Beta);  end  %% Build library and compute sparse regression  Theta = poolData(x,n,4,usesine); % Generate library  lambda = 0.025; % sparsification knob  Xi = sparsifyDynamics(Theta,dx,lambda,n);  poolDataLIST({'x','y','z'},Xi,n,4);  %% Reconstruction  [tB,xB]=ode45(@(t,x)sparseGalerkin(t,x,Xi,4,usesine),tspan,x0,options); |

**Algoritmo 4.1** – SINDy nativo applicato al sistema di Lorenz

Di seguito viene riportato il confronto tra il sistema reale e quello approssimato, in Figura 4.1, e l’errore presente su ogni variabile del sistema approssimato rispetto a quella reale, in Figura 4.2.



**Figura 4.1** – Confronto tra sistema reale e approssimato



**Figura 4.2** – Errore di ricostruzione per le tre variabili di stato

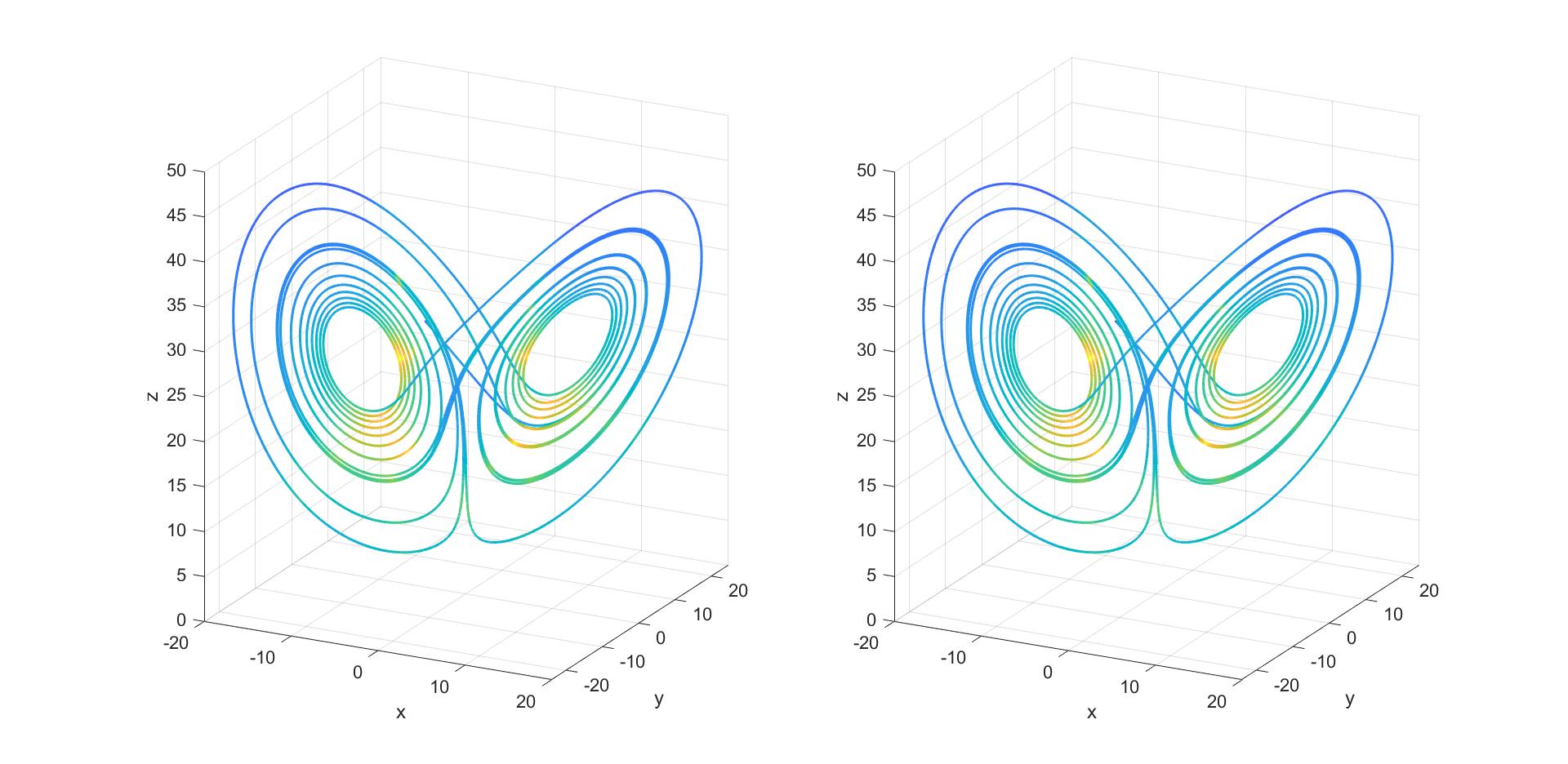
Adesso viene mostrato il confronto tra i parametri del sistema reale e quelli ottenuti dalla funzione *poolDataLIST*, questi ultimi presenti nella Tabella 4.1.

|  |
| --- |
| {0×0 char} {'xdot' } {'ydot' } {'zdot' }  {'1' } {[ 0]} {[ 0]} {[ 0]}  {'x' } {[-10.0000]} {[28.0000]} {[ 0]}  {'y' } {[ 10.0000]} {[-1.0000]} {[ 0]}  {'z' } {[ 0]} {[ 0]} {[-2.6667]} |

**Tabella 4.1** – Parametri del sistema approssimato

Per i parametri del sistema reale si può visionare l’algoritmo 4.1, e facendo un rapido confronto si denota che i valori sono totalmente identici.

Inoltre viene mostrata anche una rappresentazione 3D dei due sistemi, reale e stimato, in Figura 4.3.



**Figura 4.3** – Rappresentazione 3D dei due sistemi.

## SINDy con controllo con sistema di Lorenz

In questo caso di studio viene riprodotto il sistema di Lorenz con il controllo tramite l’utilizzo del SINDyc.

Inizialmente vengono dichiarate le variabili che saranno utilizzate all’interno del codice, viene simulato il sistema di Lorenz e vengono calcolare le sue derivate. Il tutto è mostrato all’interno dell’algoritmo 4.2.

|  |
| --- |
| clear all, close all, clc  %% Generate Data  usesine = 0; % sin knob  Beta = [10; 28; 8/3]; % Lorenz's parameters (chaotic)  n = 3; % variables number  polyorder = 3;  x0=[-8; 8; 27]; % Initial condition  dt = 0.001;  tspan=0:dt:15-dt;  omeg = 4.5; % input pulsation  a = 6; % input amplitude  g = (sin(omeg\*tspan));  options = odeset('RelTol',1e-12,'AbsTol',1e-12\*ones(1,n));  [t,x1] = ode45(@(t,x)lorenz\_c(t,x,Beta,g,tspan,a),tspan,x0,options);  figure, plot(x1(:,1),x1(:,3))  %% Compute true derivative  eps = 0.0; % Noise level  for i=1:length(x1)  dx(i,:) = lorenz\_d(0,x1(i,:),Beta,g(i),tspan,a); % Run Lorenz model to  % compute derivatives  end  dx(:,n+1) = 0\*dx(:,n); % Add additional column  % of zeros for  % derivatives of u  dx = dx + eps\*randn(size(dx)); % Add measurement noise  x1 = [x1 g']; % Stack state and input  % measurements |

**Algoritmo 4.2** – Dichiarazione variabili, simulazione sistema di Lorenz e calcolo derivate

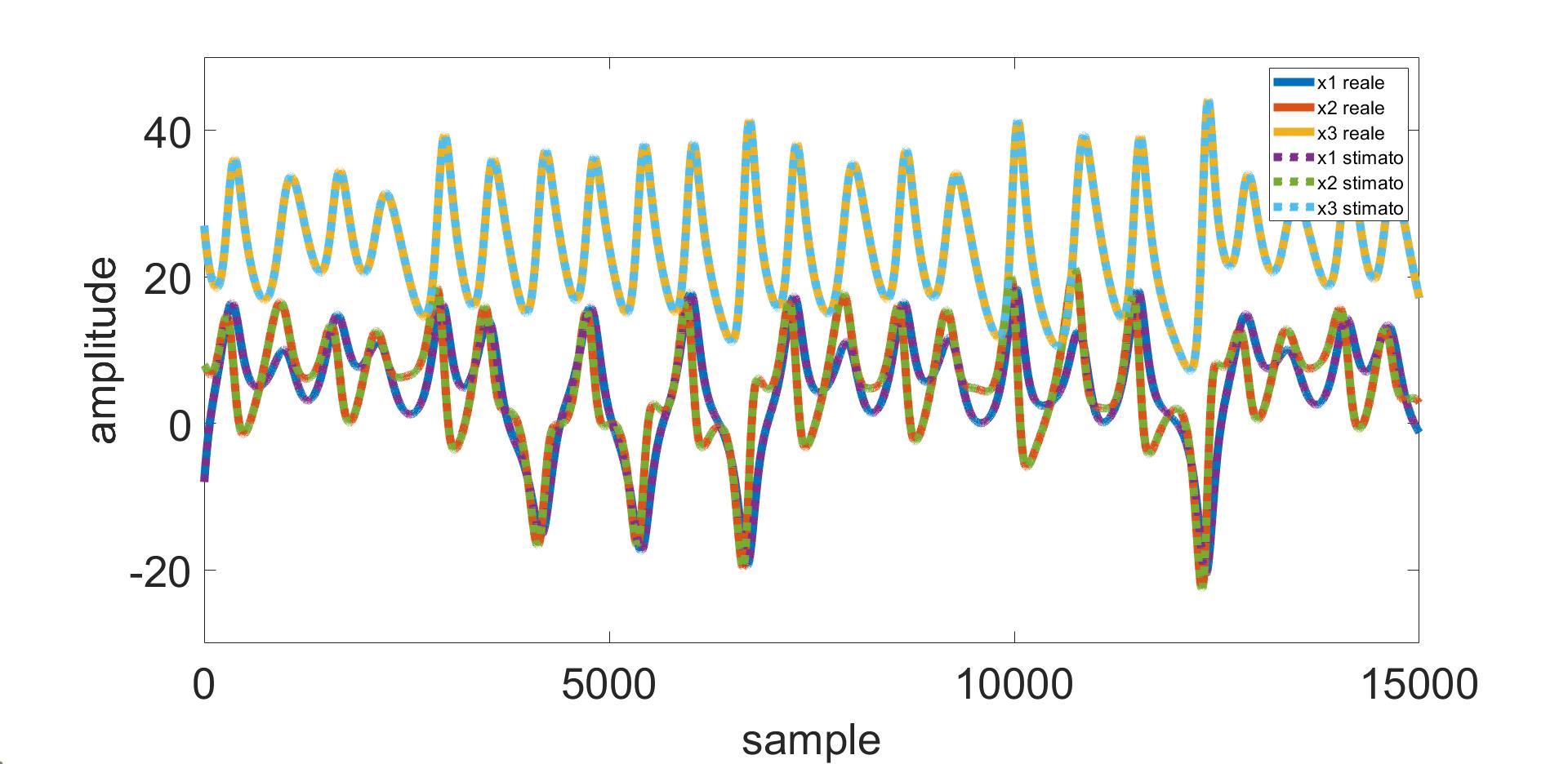
Adesso viene eseguito il SINDy andando, prima di tutto, a creare la libreria delle funzioni non lineari candidate aggiungendole una variabile a seguito dell’inserimento nel codice del controllo tramite la funzione poolData, presente nell’algoritmo 2.9, e poi viene eseguito l’algoritmo STLS tramite la funzione sparsifyDynamics, presente nell’algoritmo 2.7, che permette di trovare i termini attivi sparsi tra tutte le combinazioni delle funzioni presenti all’interno della libreria. Infine avviene la ricostruzione del sistema tramite l’utilizzo della funzione sparseGalerkinControl, la quale genera le variabili del sistema studiato. Questa parte del codice è presente nell’algoritmo 4.3.

|  |
| --- |
| %% Build library and compute sparse regression  Theta = poolData(x1,(n+1),polyorder,usesine); % Generate library  lambda = 1e-1; % sparsification  % hyperparameter  Xi = sparsifyDynamics(Theta,dx,lambda,(n+1));  poolDataLIST({'x','y','z','u'},Xi,(n+1),polyorder);  %% SINDy reconstruction  p.ahat = Xi(:,1:n);  p.polyorder = polyorder;  p.usesine = usesine;  options = odeset('RelTol',1e-12,'AbsTol',1e-12\*ones(1,n));  [~,xMT]=ode45(@(t,x)sparseGalerkinControl(t,x,g,tspan,p),tspan,x0,options); |

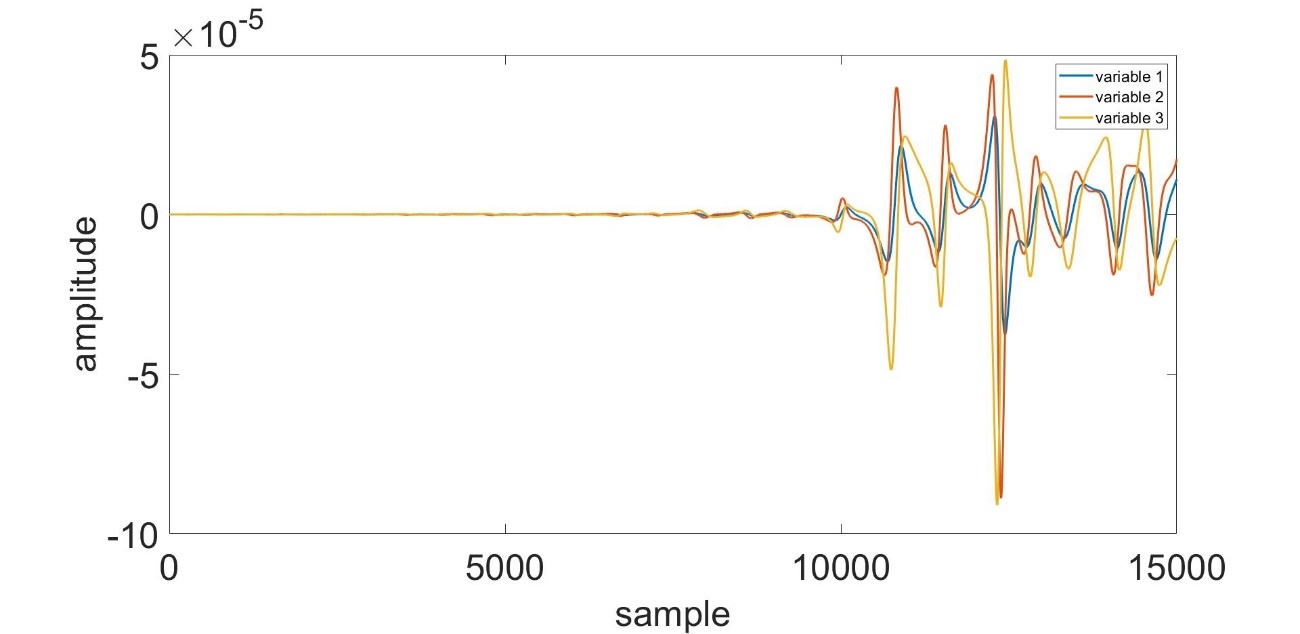
**Algoritmo 4.3** – Creazione libreria funzioni candidate, calcolo sparse regression e ricostruzione sistema

In questo caso, lavorando sempre con il sistema di Lorenz, è stato possibile ricostruire il sistema in modo ottimale con la libreria formata dalle sole funzioni non lineari polinomiali.

Adesso vengono mostrati i risultati dell’algoritmo, che consistono nella Figura 4.4, dove viene mostrato il confronto tra i due sistemi, e nella Figura 4.5, dove viene mostrato l’errore presente su ogni variabile del sistema stimato rispetto alle variabili del sistema reale.



**Figura 4.4** – Confronto tra sistema reale e approssimato



**Figura 4.5** – Errore di ricostruzione presente su ogni variabile

In questo caso si vede che il confronto risulta perfetto per tutta la durata dell’esecuzione dell’algoritmo, ma in realtà l’errore, dopo circa 10 mila istanti di tempo, inizia ad aumentare e questo probabilmente porterà ad un cambiamento dell’andamento del segnale ricostruito rispetto a quello reale aumentando ancora di più il tempo in cui viene eseguito l’algoritmo.

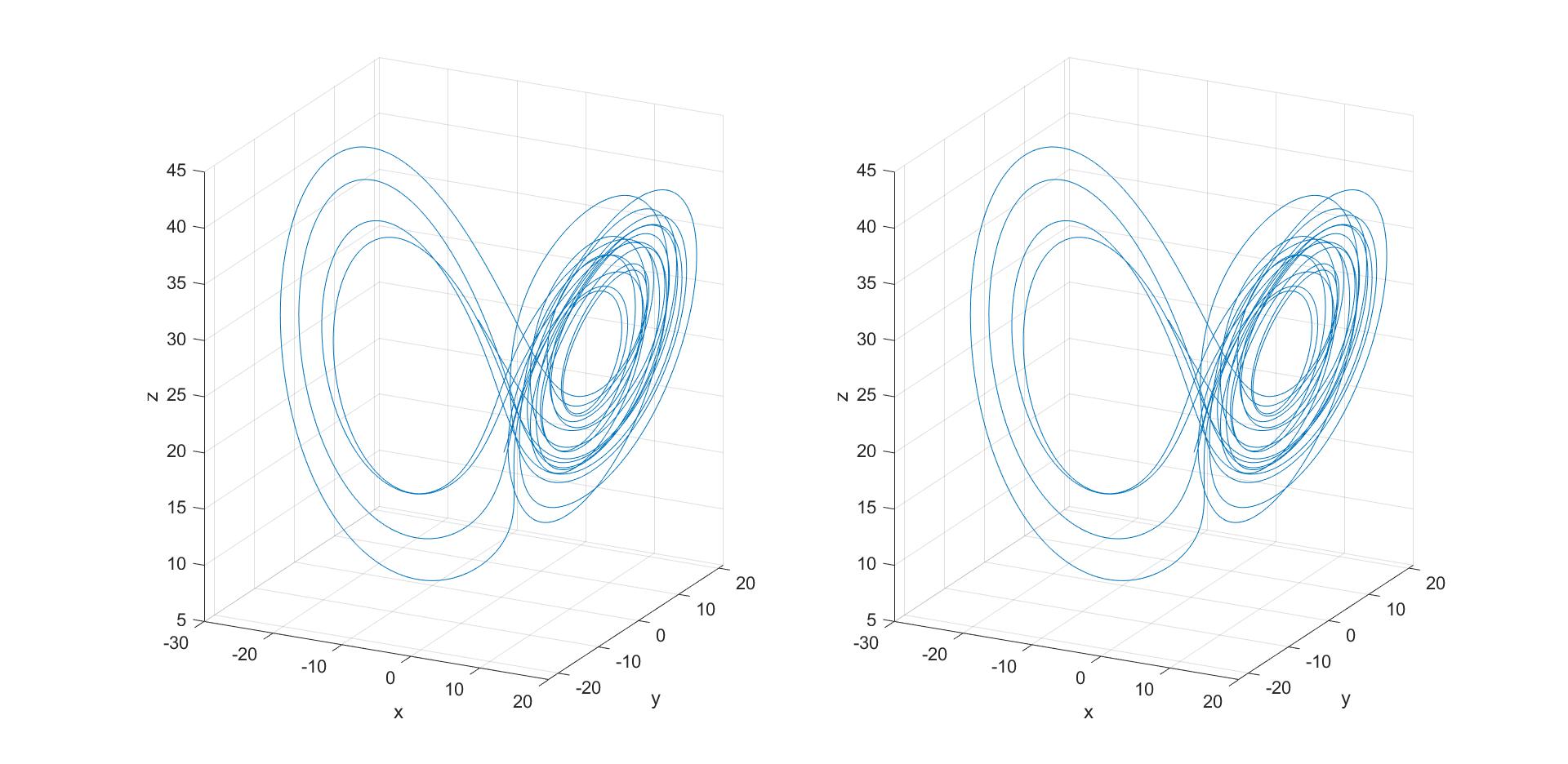
Per permettere la corretta esecuzione dell’algoritmo sono stati usati due codici leggermente differenti per la generazione del sistema reale e nel calcolo delle derivate: il codice per la generazione del sistema è quello presente nell’algoritmo 2.12, mentre quello per il calcolo delle derivate è il seguente.

|  |
| --- |
| function dx = lorenz\_d(t,x,Beta,g,gt,a)  %% Lorenz equations  dx = [  -Beta(1)\*(x(1)-x(2)-(a\*g));  x(1)\*(Beta(2)-x(3)) - x(2);  x(1)\*x(2) - Beta(3)\*x(3);  ];  end |

**Algoritmo 4.4** – Sistema di Lorenz per il calcolo delle derivate

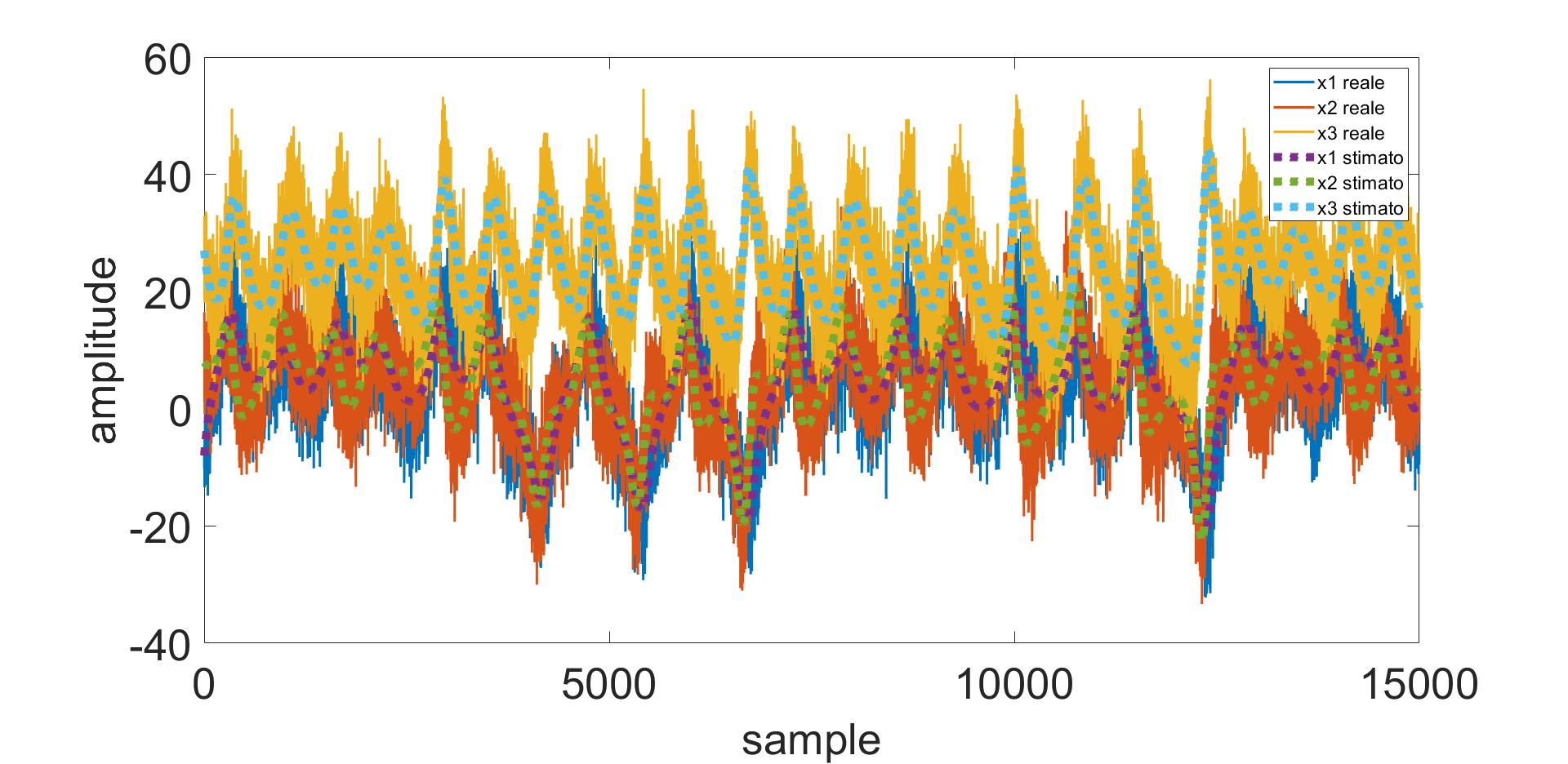
Questa variante è uguale al codice iniziale tranne che per la prima riga dove veniva fatta l’interpolazione della funzione sinusoidale, mentre durante il calcolo delle derivate questa funzione viene asserita all’interno dell’algoritmo 4.2.

Inoltre viene mostrata anche una rappresentazione 3D dei due sistemi, reale e approssimato, in Figura 4.6.



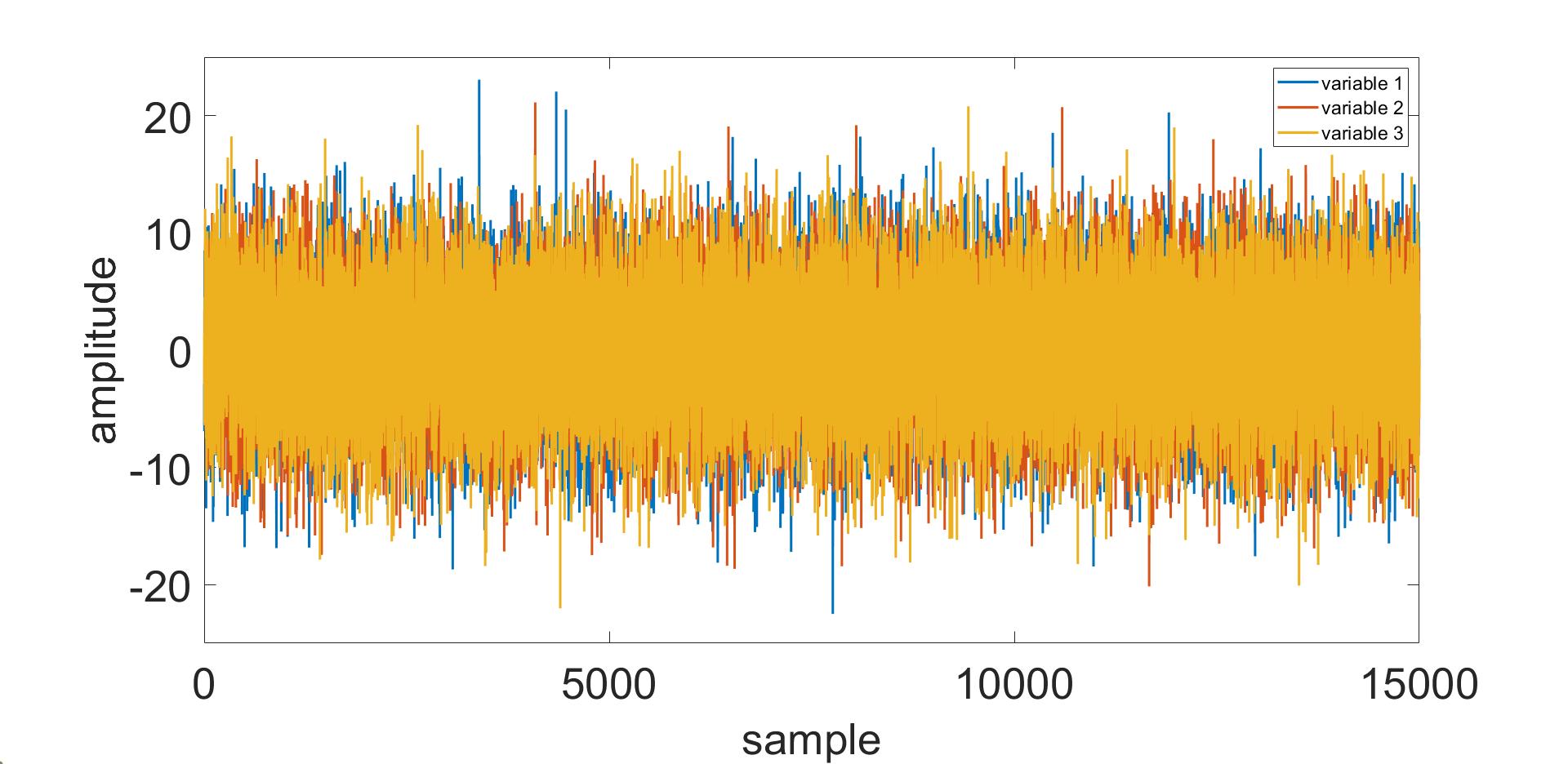
**Figura 4.6** – Plot 3d del sistema reale e di quello ricostruito

Successivamente è stato inserito un rumore, equivalente ad un rapporto S/N di 10 dB, al sistema reale per verificare la bontà dell’algoritmo ed è stato notato che il SINDy riesce perfettamente a trovare le dinamiche del sistema nonostante il rumore inserito. Questo viene mostrato all’interno della Figura 4.7.



**Figura 4.7 –** Confronto tra il sistema reale e quello approssimato con l’aggiunta di rumore

Inoltre viene mostrato l’errore che si ha tra le variabili reali e quelle approssimate nel caso dell’aggiunta di questo rumore, visibile in Figura 4.8:



**Figura 4.8** – Errore di ricostruzioni per ogni variabile del sistema

Adesso viene mostrato il confronto tra i parametri del sistema reale e quelli ottenuti dalla funzione *poolDataLIST*, questi ultimi presenti in Tabella 4.2.

|  |
| --- |
| {0×0 char} {'xdot' } {'ydot' } {'zdot' } {'udot'}  {'1' } {[ 0]} {[ 0]} {[ 0]} {[ 0]}  {'x' } {[-10.0000]} {[28.0000]} {[ 0]} {[ 0]}  {'y' } {[ 10.0000]} {[ -1]} {[ 0]} {[ 0]}  {'z' } {[ 0]} {[ 0]} {[-2.6667]} {[ 0]}  {'u' } {[ 60.0000]} {[ 0]} {[ 0]} {[ 0]}  {'xx' } {[ 0]} {[ 0]} {[ 0]} {[ 0]}  {'xy' } {[ 0]} {[ 0]} {[ 1.0000]} {[ 0]}  {'xz' } {[ 0]} {[ -1]} {[ 0]} {[ 0]} |

**Tabella 4.2** – Parametri del sistema approssimato

Per i parametri del sistema reale si può visionare l’algoritmo 4.2, e facendo un rapido confronto si denota che i valori sono totalmente identici.

Durante questo studio il SINDy è stato eseguito con come controllo, ma adesso viene mostrata una seconda esecuzione con solo come controllo usato per istruire il SINDy e il risultato che si otterrà sarà che il SINDy riuscirà a trovare il seno mancate.

La parte iniziale del codice è molto simile a quella originale, con l’unica modifica alla fine del calcolo delle derivate dove la matrice che formerà l’insieme dei dati per l’esecuzione del SINDy verrà completata con solo . Questo lo si può vedere all’interno dell’algoritmo 4.5.

|  |
| --- |
| clear all, close all, clc  %% Generate Data  usesine = 1; % sin knob  Beta = [10; 28; 8/3]; % Lorenz's parameters (chaotic)  n = 3; % variables number  polyorder = 4;  x0=[-8; 8; 27]; % Initial condition  dt = 0.001;  tspan=0:dt:15-dt;  omeg = 4.5;  a = 6;  g = (sin(omeg\*tspan));  options = odeset('RelTol',1e-12,'AbsTol',1e-12\*ones(1,n));  [t,x1] = ode45(@(t,x)lorenz\_c(t,x,Beta,g,tspan,a),tspan,x0,options);  %% Compute true derivative  eps = 0.0; % Noise level  for i=1:length(x1)  dx(i,:) = lorenz\_d(0,x1(i,:),Beta,g(i),tspan,a); % Run Lorenz model to  % compute derivatives  end  dx(:,n+1) = 0\*dx(:,n); % Add additional column  % of zeros for  % derivatives of u  dx = dx + eps\*randn(size(dx)); % Add measurement noise  x1 = [x1 (omeg\*tspan)']; % Stack state and input  % measurements |

**Algoritmo 4.5** - Dichiarazione variabili, simulazione sistema di Lorenz e calcolo derivate

La restante parte del codice è rimasta invariata. A questo punto viene fatto il confronto tra i parametri del sistema reale e quelli ottenuti dalla funzione *poolDataLIST*, con questi ultimi presenti in Tabella 4.3.

|  |
| --- |
| {0×0 char } {'xdot' } {'ydot' } {'zdot' } {'udot'}  {'1' } {[ 0]} {[ 0]} {[ 0]} {[ 0]}  {'x' } {[ -10]} {[ 28]} {[ 0]} {[ 0]}  {'y' } {[ 10]} {[-1.0000]} {[ 0]} {[ 0]}  {'z' } {[ 0]} {[ 0]} {[-2.6667]} {[ 0]}  {'u' } {[ 0]} {[ 0]} {[ 0]} {[ 0]}  {'xx' } {[ 0]} {[ 0]} {[ 0]} {[ 0]}  {'xy' } {[ 0]} {[ 0]} {[ 1.0000]} {[ 0]}  {'xz' } {[ 0]} {[-1.0000]} {[ 0]} {[ 0]}    {'sin(1\*yin4)' } {[60.0000]} {[ 0]} {[ 0]} {[ 0]} |

**Tabella 4.3** – Parametri del sistema approssimato

Per i parametri del sistema reale si può visionare l’algoritmo 4.5, e facendo un rapido confronto si denota che i valori sono totalmente identici.

# Conclusioni

In questo lavoro sono stati presi in esame due metodi, la Dynamic Mode Decomposition (DMD) e la Sparse Identification of Nonlinear Dynimics (SINDy), per la ricostruzione del sistema a partire dalle misurazioni prese su di esso. Sono stati usati inizialmente dei sistemi lineari con matrice diagonale per il debugging degli algoritmi e successivamente è stato usato un sistema non lineare composto da tre equazioni differenziali chiamato sistema di Lorenz. Per quanto riguarda il DMD sono stati utilizzati tre diverse versioni dell’algoritmo:

1. Il DMD nativo, che ha avuto dei problemi nella ricostruzione di sistemi più complessi come quello di Lorenz;
2. L’Hankel DMD, che è stato migliore nella stima delle variabili dei vari sistema, anche in presenza di dinamiche non lineari;
3. Il DMDc, il quale è stato usato per verificare i cambiamenti che si andavano a verificare sui vari sistemi dopo che si fosse applicato un segnale di controllo proveniente dall’esterno.

Queste versioni sono state prima studiate singolarmente, per poi essere nel caso del sistema di Lorenz.

Infine il DMD è stato usato anche per lo studio di un sistema reale, ma con l’algoritmo nativo non è stato possibile ottenere una ricostruzione accurata del sistema. Una sorte migliore è stata trovata con l’utilizzo dell’Hankel DMDc che è riuscito a ricostruire perfettamente il sistema nei primi campioni, ma successivamente è aumentato l’errore portando con se una divergenza tra sistema reale e approssimato.

Per quanto riguarda, invece, il SINDy, sono state usate due versioni:

1. Il SINDy nativo, il quale ha mostrato ottimi risultati con il sistema non lineare di Lorenz;
2. Il SINDyc, in grado di ricostruire il sistema, di Lorenz, con l’aggiunta del controllo.

Per quanto riguarda gli sviluppi futuri, gli algoritmi presentati verranno valutati su diverse categorie di modelli, analizzando le prestazioni e l’interpretabilità dei risultati ottenuti.

Verranno inoltre approfondite le relazioni tra i risultati e i parametri più rilevanti come il fattore di riduzione per il DMD e le librerie di funzioni per il SINDy.

Un ulteriore sviluppo futuro sia per il DMD che per il SINDy potrebbe essere la valutazione del filtraggio dei dati in ingresso tramite il seguente codice:

|  |
| --- |
| windowSize = 5;  b = (1/windowSize)\*ones(1,windowSize);  a = 1;  y = filter(b,a,x); |

dove vengono filtrati usando una funzione di trasferimento razionale definita da un numeratore e un denominatore di coefficienti b ed a.

# Bibliografia

[1] Brunton Steven L., Budisic Marko, Kaiser Eurica, Kutz Nathan J. *Modern Koopman Theory for Dynamical Systems,* s.l., SIAM, 2022

[2] Brunton Steven L., Kutz Nathan J. *Data Driven Science & Engineering - Machine Learning, Dynamical Systems, and Control,* Seattle, 2018, p.572

[3] Brunton Steven L., Proctor Joshua L., Kutz Nathan J. *Discovering governing equations from data by sparse identification of non-linear dynamical systems*, in “PNAS”, n. 15, 2016, 3932-3937

[4] Chunyan Han, Simin Yu, Guangyi Wang *A sinusoidally Driven Lorenz System and Circuit Implementation,* in “Hindawi”, n. 2015, 2015

[5]Filho Enio, dos Santos Paulo Lopes *A Dynamic Mode Decomposition approach with Hankel blocks to forecast multi-channel temporal series*, in “IEEE”, a. III, n.3,2019, 739-744

[6] Fujii Keisuke, Takeshi Naoya, Kibushi Benio, Kouzaki Motoki, Kawahara Yoshinobu *Data-driven spectral analysis for coordinative structures in periodic human* locomotion, in “Scientific reports”, a. IX, n.16755, 2019

[7] Kaiser Eurica, Kutz Nathan J., Brunton Steven L, *Sparse identification of nonlinear dynamics for model predictive control in the low-data limit*, in “The royal society”, a. CDLXXIV, n. 2219,2018

[8] Kutz Nathan J., Brunton Steven L., Brunton Bingni W., Proctor Joshua L. *Dynamic Mode Decomposition – Data-Driven Modeling of Complex Data*, Philadelphia, SIAM, 2016, p. 241

[9] Zhenglong Yin, Bo Fan, Zijing Ding, et al. *Comparative study of modal decomposition and dynamic equation reconstruction in data-driven* modeling, in “AIP”, a. XI, n. 085022, 2021

**Ringraziamenti**

*A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non esisterebbe nemmeno.*

*Ringrazio il mio relatore Patanè Luca, che in questi mesi di lavoro, ha saputo guidarmi, con suggerimenti pratici, nelle ricerche e nella stesura dell’elaborato.*

*Un ringraziamento speciale va ai colleghi, che hanno contribuito con le loro idee a dare un tocco di originalità a questo elaborato e ai miei amici per essere stati sempre presenti anche durante tutto il mio percorso di studi. Grazie per aver ascoltato i miei sfoghi, grazie per tutti i momenti di spensieratezza.*

*Infine ringrazio di cuore i miei genitori. Grazie per avermi sempre sostenuto e per avermi permesso di portare a termine gli studi universitari.*