POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Data-driven system identification of PMSM control using SINDy and Neural Network

MODELLING FROM MEASUREMENTS - COURSE PROJECT

Authors: **Matteo Deponti & Juri Tessaro**

11th October 2024

Contents

Contents ..... **iii**

A Problem statement **5**

A.1. System definition 5

A.2. SINDy 7

B SINDy – Application to the PMSM system **9**

B.1. Configuration and simulation setup 9

B.1.1. Simulation 1: no noise 11

B.1.2. Simulation 2: noisy data 12

B.1.3. Simulation 3: PCA based pre-processing of noisy data 14

B.1.4. Modified SINDy: sparsity known a priori 16

C Neural Network **19**

C.1. Objective 19

C.2. Model Implementation 20

C.2.1. Data Generation for Training 20

C.2.2. Neural Network Creation and Training 20

C.3. Prediction and Validation 21

D Conclusion and future developments **23**

1 Problem statement

1.1 System definition

The system consists in a three-phase Permanent Magnet Synchronous Motor (PMSM) fed by a Voltage Source Converter (VSC): this represents a typical experimental setup. The controller consists of:

- An outer close-loop speed PI regulator: it receives as input
- An inner close-loop dq-current PI regulator,

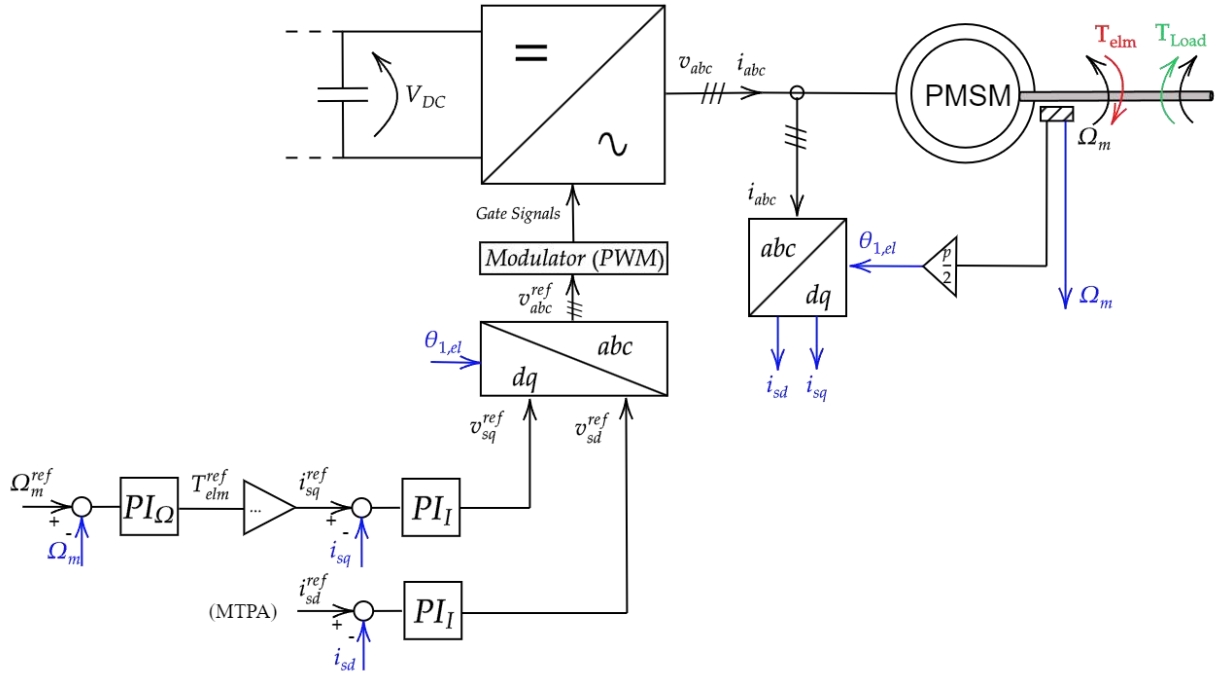

Fig. 1. block diagram for PMSM control

The mechanical model of the system is given by the Newton's law of motion:

$$J_m \frac{d}{dt}\Omega_m = T_{elm} - T_{load}$$

Where Ω_m is the rotor mechanical speed, T_{load} is a given load torque and T_{elm} is the electromagnetic torque produced by the PMSM. The electrical model is represented by the dq stator voltage equations:

$$\begin{cases} v_{sd}^* = R_s i_{sd}^* + L_d \dfrac{d}{dt} i_{sd}^* - pp L_q \Omega_m i_{sq}^* \\ v_{sq}^* = R_s i_{sq}^* + L_q \dfrac{d}{dt} i_{sq}^* + pp L_d \Omega_m i_{sd}^* + pp \Omega_m \hat{\Psi}_{PM} \\ T_{elm} = \dfrac{3}{2} pp \left(\hat{\Psi}_{PM} i_{sq} + (L_d - L_q) i_{sd} i_{sq} \right) \end{cases}$$

By merging the two model in a unique system, and rearranging the terms accordingly, the following nonlinear system of first order differential equations can be obtained:

$$\begin{cases} \dfrac{d}{dt} i_{sd}^* = -\dfrac{1}{L_d} R_s i_{sd}^* + \dfrac{1}{L_d} pp L_q \Omega_m i_{sq}^* + \dfrac{1}{L_d} v_{sd}^* \\ \dfrac{d}{dt} i_{sq}^* = -\dfrac{1}{L_q} R_s i_{sq}^* - \dfrac{1}{L_q} pp L_d \Omega_m i_{sd}^* - \dfrac{1}{L_q} pp \Omega_m \hat{\Psi}_{PM} + \dfrac{1}{L_q} v_{sq}^* \\ \dfrac{d}{dt} \Omega_m = \dfrac{1}{J_m} \dfrac{3}{2} pp \left(\hat{\Psi}_{PM} i_{sq}^* + (L_d - L_q) i_{sd}^* i_{sq}^* \right) - \dfrac{1}{J_m} T_{load} \end{cases}$$

To get all the variables of the same order of magnitude, the system is normalized with respect to given base quantities, defined as:

- $V_b = \sqrt{2} V_n, \;\; I_b = \sqrt{2} I_n, \;\; Z_b = \frac{V_b}{I_b}$;
- $\omega_n^e = pp * \Omega_{m,n} = \omega_b$;
- $L_b = \frac{Z_b}{\omega_b}$;
- $\Psi_b = \frac{V_b}{\omega_b}$
- $T_b = \frac{3}{2} pp \Psi_b I_b$;

Then applying:

$$\begin{cases} \dfrac{1}{V_b} \dfrac{d}{dt} i_{sd}^* = \dfrac{1}{V_b} \left(-\dfrac{1}{L_d} R_s i_{sd}^* + \dfrac{1}{L_d} pp L_q \Omega_m i_{sq}^* + \dfrac{1}{L_d} v_{sd}^* \right) \\ \dfrac{1}{V_b} \dfrac{d}{dt} i_{sq}^* = \dfrac{1}{V_b} \left(-\dfrac{1}{L_q} R_s i_{sq}^* - \dfrac{1}{L_q} pp L_d \Omega_m i_{sd}^* - \dfrac{1}{L_q} pp \Omega_m \hat{\Psi}_{PM} + \dfrac{1}{L_q} v_{sq}^* \right) \\ \dfrac{1}{T_b} \dfrac{d}{dt} \Omega_m = \dfrac{1}{T_b} \left(\dfrac{1}{J_m} \dfrac{3}{2} pp \left(\hat{\Psi}_{PM} i_{sq}^* + (L_d - L_q) i_{sd}^* i_{sq}^* \right) - \dfrac{1}{J_m} T_{load} \right) \end{cases}$$

And considering:

- $r_s = \frac{R_s}{Z_b}$, $l_d = \frac{L_d}{L_b}$, $l_q = \frac{L_q}{L_b}$, $\psi_{pm} = \frac{\hat{\Psi}_{PM}}{\Psi_b}$;
- $H_m = \frac{1}{2} \frac{J_m \Omega_{m,n}^2}{T_b \Omega_{m,n}}$;
- $v_{sd} = \frac{v_{sd}^*}{V_b}$, $v_{sq} = \frac{v_{sq}^*}{V_b}$, $i_{sd} = \frac{i_{sd}^*}{I_b}$, $i_{sq} = \frac{i_{sq}^*}{I_b}$;
- $T_l = \frac{T_{Load}}{T_b} \omega_m = \frac{\Omega_m}{\Omega_{m,n}}$;

The following system in per-unit is obtained:

$$\begin{cases} \frac{d}{dt} i_{sd} = -r_s \frac{\omega_b}{l_d} i_{sd} + \frac{\omega_b}{l_d} l_q \omega_m i_{sq} + \frac{\omega_b}{l_d} v_{sd} \\ \frac{d}{dt} i_{sq} = -r_s \frac{\omega_b}{l_q} i_{sq} - \frac{\omega_b}{l_q} l_d \omega_m i_{sd} - \frac{\omega_b}{l_q} \psi_{pm} \omega_m + \frac{\omega_b}{l_q} v_{sd} \\ \frac{d}{dt} \omega_m = \frac{1}{2H_m} \psi_{pm} i_{sq} + \frac{I_b}{2H_m} (l_d - l_q) i_{sd} i_{sq} - \frac{1}{2H_m} T_l \end{cases}$$

Which can be seen as:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$$

Where $\boldsymbol{x} = [i_{sd}, i_{sq}, \Omega_m]^T$ are the system state variables, $\boldsymbol{u} = [v_{sd}, v_{sq}, T_l]^T$ are the system inputs. For this system, the parameters are listed in Table 1.

Table 1: PMSM system parameters

Parameter	**Value**	**Parameter**	**Value**
V_n [V], I_n [A], pp	190, 10.2, 5	R_s [Ω], $L_d = L_q$ [H], $\widehat{\Psi}_{PM}$ [Wb]	$0.201, 4.4 * 10^{-3}, \sqrt{2} * 0.08$
$\Omega_{m,n}$ [rpm], J_m [kg*m^2]	$4500, 9.8 * 10^{-3}$		

1.2 SINDy

Let's consider the system dynamics problem expressed as:

$$\dot{\boldsymbol{X}} = \boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi}$$

where $\dot{\boldsymbol{X}}$ represents the data of the time derivative of the state variables $\boldsymbol{x}$ in time, $\boldsymbol{X}$ is the time-series of the state variables measurements $\boldsymbol{x}$, $\boldsymbol{\Theta}(\boldsymbol{X})$ is the matrix of base functions (nonlinear), and $\boldsymbol{\Xi}$ is the vector of coefficients describing the model.

In the context of sparse regression, the objective is to find the coefficients $\boldsymbol{\Xi}$ that best describe the dynamics by minimizing the following problem:

$$\xi_k = \text{argmin}_{\xi_k'} \left\| \dot{\boldsymbol{X}} - \boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi} \right\|_2 + \lambda \|\xi_k'\|_1$$

In this case, a combination of the L_2norm (squared error) and the L_1-norm (regularization) is used to obtain a sparse solution. The L_1-norm acts to penalize small coefficients and thus induces sparsity through a proper choice of $\lambda \in R$, as in the Lasso method.

In presence of input variables, the problem transforms to:

$$\xi_k = \text{argmin}_{\xi_k'} \left\| \dot{\boldsymbol{X}} - \boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{U})\boldsymbol{\Xi} \right\|_2 + \lambda \|\xi_k'\|_1$$

2 SINDy – Application to the PMSM system

In this chapter, SINDy is used to retrieve the machine parameters directly from measurements, so to reconstruct the system dynamics through easily implementable experimental tests. Dynamic Mode Decomposition (DMD) could not be applied to this case, being the system nonlinear. On the other hand, SINDy represents a proper tool to deal with the system nonlinearity, according to the library of candidate function which must be defined. SINDy algorithm was built in Matlab.

2.1 Configuration and simulation setup

The idea is to set a procedure reproducible in a laboratory, where a PMSM is available but for which there is little or no knowledge on its parameters. Without this piece of information, it would be difficult to design a suitable control scheme, for instance the PI regulators shown in Fig. 1. Moreover, one could operate this test to calibrate and assess the change in time of the machine parameters.

To investigate the system dynamics, the PMSM is excited through specific dq voltage and load torque input profiles; to help the SINDy algorithm to recognize the right input-state relationships, the following functions were adopted:

- $v_{sd} = \widehat{V_d} \sin\left(2\pi f_d - \frac{\pi}{2}\right)$;
- $v_{sq} = V_{q0} + \widehat{V_q} \sin(2\pi f_q)$;
  - $f_d \neq f_q$ in order to assist the algorithm in finding the right correlations;
- $T_l = T_{l0} + k_f * \omega_m^2$
  - This represents a realistic scenario, in which friction also acts on the system: it was supposed to be function of the squared of the speed.

The input parameters are listed in Table 2.

Table 2: input variables parameters used in the simulation

Parameter	**Value**	**Parameter**	**Value**
$\boldsymbol{f_d}$ [Hz], $\boldsymbol{f_q}$ [Hz]	5, 0.1	T_{l0} [p.u.], k_f [p.u.]	0.08, 0.05
$\boldsymbol{\widehat{V_d}}$, $\boldsymbol{\widehat{V_q}}$, $\boldsymbol{V_{q0}}$ [p.u.]	0.010, 0.025, 0.050		

Let us consider all the state variables $\boldsymbol{x} = [i_{sd}, i_{sq}, \Omega_m]^T$ as measured and the inputs $\boldsymbol{u} = [v_{sd}, v_{sq}, T_l]^T$ as known. The library of candidate function $\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{U})$ is built by analyzing the system of nonlinear differential equations provided previously; let us define the data time-series, obtained by measuring the state variables with a frequency $f_{sampling} = \frac{1}{t_{step}} = 10^5$ [Hz]:

- $\boldsymbol{X} = \begin{bmatrix} i_{sd}(1) & i_{sq}(1) & \omega_m(1) \\ i_{sd}(2) & i_{sq}(2) & \omega_m(2) \\ \dots & \dots & \dots \\ i_{sd}(n) & i_{sq}(n) & \omega_m(n) \end{bmatrix} = [\boldsymbol{i_{sd}} \quad \boldsymbol{i_{sq}} \quad \boldsymbol{\omega_m}] \in \boldsymbol{R}^{n\,x\,3}$;
- $\boldsymbol{U} = \begin{bmatrix} v_{sd}(1) & v_{sq}(1) & T_l(1) \\ v_{sd}(2) & v_{sq}(2) & T_l(2) \\ \dots & \dots & \dots \\ v_{sd}(n) & v_{sq}(n) & T_l(n) \end{bmatrix} \in \boldsymbol{R}^{n\,x\,3}$;

The following library of candidate functions is proposed:

$$\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{U}) = [\boldsymbol{i_{sd}} \quad \boldsymbol{i_{sq}} \quad \boldsymbol{\omega_m} \quad \boldsymbol{i_{sd}} * \boldsymbol{\omega_m} \quad \boldsymbol{i_{sq}} * \boldsymbol{\omega_m} \quad \boldsymbol{i_{sd}} * \boldsymbol{i_{sq}} \quad \boldsymbol{v_{sd}} \quad \boldsymbol{v_{sq}} \quad \boldsymbol{T_l}]$$

The vector of time derivatives is calculated by computing the discrete derivative for each state, given by the generic formula:

$$\frac{dx}{dt} \approx \frac{\Delta x}{t_{step}} = \frac{x(k) - x(k-1)}{t_{step}}$$

thus, resulting in:

$$\dot{\boldsymbol{X}} = \frac{1}{t_{step}} \begin{bmatrix} i_{sd}(1) & i_{sq}(1) & \omega_m(1) \\ i_{sd}(2) - i_{sd}(1) & i_{sq}(2) - i_{sq}(1) & \omega_m(2) - \omega_m(1) \\ \dots & \dots & \dots \\ i_{sd}(n) - i_{sd}(n-1) & i_{sq}(n) - i_{sq}(n-1) & \omega_m(n) - \omega(n-1) \end{bmatrix}$$

SINDy algorithm is performed through the following Matlab function, which gets as inputs $\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{U})$, $\dot{\boldsymbol{X}}$, λ and the number of measurements, and gives as output the vector of coefficients $\boldsymbol{\Xi}$.

```
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
end
```

2.1.1 Simulation 1: no noise

Let us suppose here that no noise is present in the measurements. The test is carried out for a period $T_{simulation} = 15$ [s]; the state variables and inputs are plotted in Fig. 2.

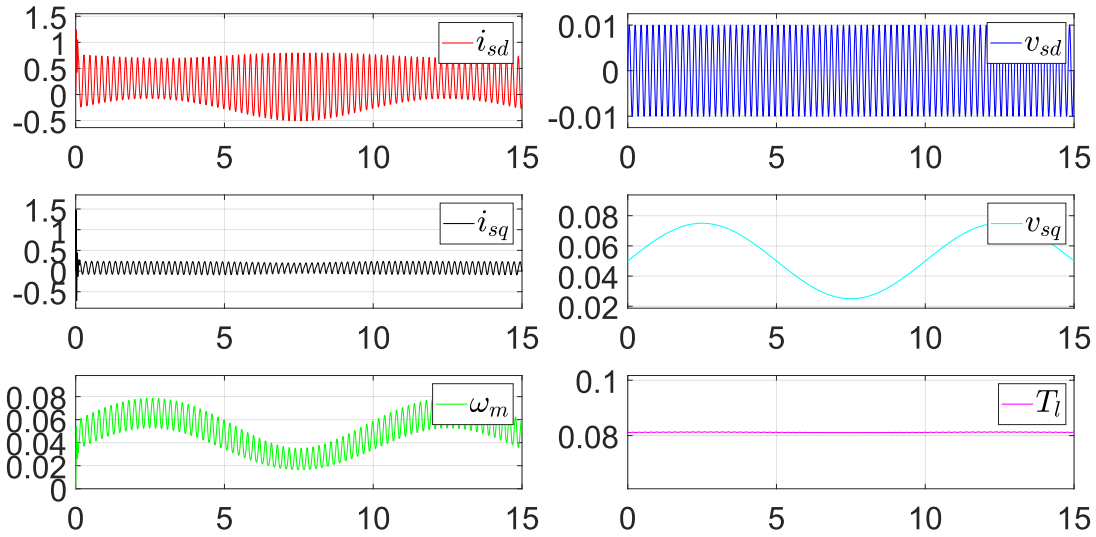

Fig. 2. Inputs and state variables

As a starting point, SINDy is performed with the sparsity promoting parameter $\lambda = 0.1$; the result is:

Result 2.1-1

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 2.7*iq - 2.7*Tl (SINDy)
-
```

As can be seen, SINDy is able to properly reconstruct the system dynamics in case of clean measurements.

By looking at Matlab function *sparsifyDynamics*, the lambda coefficient filters out all the terms less than $\lambda = 0.1$. This enhances the sparsity of the reconstructed system, but care must be taken. In the original system, 2.7 is the lowest coefficient in value, as can be seen in "*Eq*3". If for example λ is set to 5, hence greater than 2.7, the following result is obtained:

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 0.0 (SINDy)
-
```

As expected, the algorithm cancels those parameters less than λ, therefore is not able to capture the dynamics of "*Eq3*".

2.1.2 Simulation 2: noisy data

In real experiments, measurements are affected by noises, hence:

- A normally distributed random signal with zero mean and a standard deviation of $\sigma_I = \frac{0.05}{I_b}$ is added to i_{sd} and i_{sq} measurements;
- A normally distributed random signal with zero mean and a standard deviation of $\sigma_\Omega = \frac{1}{\Omega_{m,n}}$ is added to ω_m measurement.

The inputs are considered not to be affected by noises. The noisy state variables are plotted in Fig. 3.

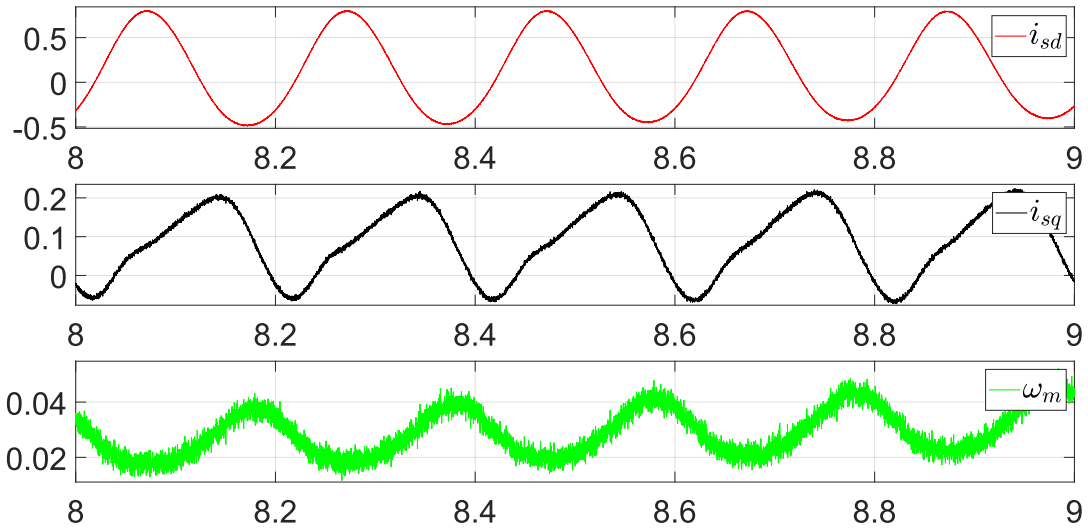

Fig. 3. Noisy States variables

As can be seen, the noise is more important on ω_m measurement, as this variable keeps low in magnitude throughout the acquisition time.

As a starting point, SINDy is performed with the sparsity promoting parameter $\lambda = 0.1$; the result is:

Result 2.1-2

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 46.0*id - 2.1*iq - 0.47*Tl + 180.0*vq - 170.0*wm + 2.8*id*iq - 100.0*id*wm +
2.3e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 0.16*id - 23.0*Tl + 26.0*iq + 280.0*vd + 1.1e+3*vq - 1.1e+3*wm - 45.0*id*iq - 500.0*id*wm
- 270.0*iq*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 2.7*iq - 2.7*Tl + 0.17*vd + 0.58*vq - 0.52*wm - 0.37*id*wm - 0.24*iq*wm (SINDy)
-
```

By setting $\lambda = 1$, so promoting more sparsity in the reconstruction, one gets:

Result 2.1-3

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 2.2*iq - 46.0*id + 180.0*vq - 170.0*wm + 2.8*id*iq - 100.0*id*wm +
2.3e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 26.0*iq - 23.0*Tl + 290.0*vd + 1.1e+3*vq - 1.1e+3*wm - 45.0*id*iq - 490.0*id*wm -
270.0*iq*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 2.7*iq - 2.7*Tl (SINDy)
-
```

Before commenting on the results, let us investigate the algorithm performance setting $\sigma_\Omega = 0$, as to have a clean speed measurement. Keeping $\lambda = 1$, the following result is obtained:

Result 2.1-4

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 46.0*id + 1.8*vq - 1.4*wm - 1.9*id*wm + 2.3e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 11.0*vd - 45.0*iq + 4.2e+3*vq - 4.2e+3*wm - 2.3e+3*id*wm + 5.8*iq*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 2.7*iq - 2.7*Tl (SINDy)
-
```

Some considerations can be made:

- the low value of λ results in a less sparse system identification, as expected: however, care must be taken when choosing λ too high, as this can hide some important system dynamics;

- the presence of noisy data strongly affects the algorithm performance: one could mitigate the problem by changing λ, up to some extent, but it is fundamental to have a proper pre-process or filtering of the data.

In the next, different methods to reduce noises impact are investigated.

2.1.3 Simulation 3: PCA based pre-processing of noisy data

An attempt to filter the data is represented by the Principal Component Analysis (PCA). The PCA is essentially obtained by applying a Singular Value Decomposition (SVD) to a mean-subtracted dataset. The data are projected onto a new coordinate system determined by Principal Components that are orthogonal to each other, maximising the correlation with measurements.

In this case, we are considering the following dataset:

$$\boldsymbol{Y} = [\boldsymbol{X}, \boldsymbol{U}] = \in R^{n\,x\,6}$$

where m is the number of samples/snapshots. Let us define with $\overline{\boldsymbol{Y}} \in R^{1\,x\,6}$:

$$\overline{\boldsymbol{Y}} = \begin{bmatrix} \overline{\iota_{sd}} & \overline{\iota_{sq}} & \overline{\omega_m} & \overline{v_{sd}} & \overline{v_{sq}} & \overline{T_l} \end{bmatrix}$$

the column vector containing the mean of the measurements, so of the n samples. Then, B can be defined as:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{Y}(1) - \overline{\boldsymbol{Y}} \\ \boldsymbol{Y}(2) - \overline{\boldsymbol{Y}} \\ \cdots \\ \boldsymbol{Y}(n) - \overline{\boldsymbol{Y}} \end{bmatrix} \in R^{n\,x\,6}$$

The economy SVD is applied to the mean-subtracted dataset $\boldsymbol{B}$, resulting in:

$$\boldsymbol{B} \approx \boldsymbol{U_{svd}} \boldsymbol{\Sigma} \boldsymbol{V^T}$$

Where $\boldsymbol{U_{svd}} \in R^{n\,x\,6}$ and $\boldsymbol{V} \in R^{6\,x\,6}$ are the left and right singular eigenvectors, $\boldsymbol{\Sigma} \in R^{6\,x\,6}$ is a diagonal matrix with the singular values, which capture the variance of the data in the principal directions.

The idea is to create a low-rank r approximation of the data in which only the principal directions with the highest variances are considered, so getting:

$$\widetilde{\boldsymbol{B}} = \widetilde{\boldsymbol{U_{svd}}} \widetilde{\boldsymbol{\Sigma}} \widetilde{\boldsymbol{V}}^{\boldsymbol{T}}$$

Where $\widetilde{\boldsymbol{U_{svd}}} \in R^{n\,x\,r}$, $\widetilde{\boldsymbol{V}} \in R^{r\,x\,r}$ and $\widetilde{\boldsymbol{\Sigma}} \in R^{r\,x\,r}$; the rank dimension is selected by including only the principal components with per cent variance higher than 1%. The per cent variances are shown in Fig. 4, and it's defined as:

$$\sigma_{\%,i} = \frac{\sigma_i}{\sum_{j=1}^{6} \sigma_j} * 100$$

Where σ_i is the principal component i, i.e. the i-element of the diagonal of $\boldsymbol{\Sigma}$.

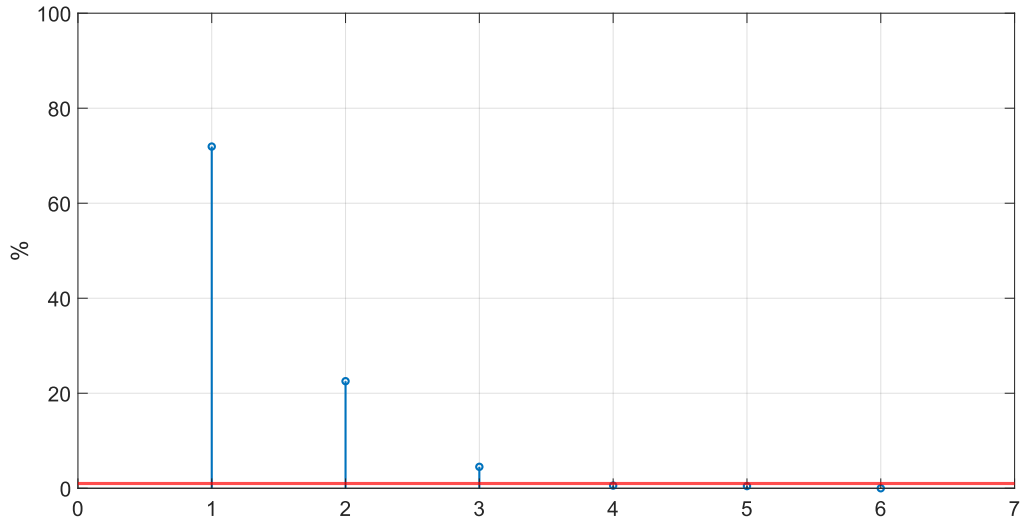

Fig. 4. Percent variances

As can be seen, the last three principal components have a percent variance lower than the threshold, thus $r = 3$ is selected. Thus the dataset $\boldsymbol{Y}$ is reconstructed first calculating $\widetilde{\boldsymbol{B}}$, then retrieving:

$$\widetilde{\boldsymbol{Y}} = \widetilde{\boldsymbol{B}} + \overline{\boldsymbol{Y}} = [\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{U}}]$$

Since the input dataset $\boldsymbol{U}$ is not affected by noises, its low-rank representation $\widetilde{\boldsymbol{U}}$ won't be used in the SINDy algorithm, which is then set to work on the reconstructed state variables dataset $\widetilde{\boldsymbol{X}}$ and on the former input dataset $\boldsymbol{U}$.

In Fig. 5 the noisy and PCA-filtered state variable measurements are shown.

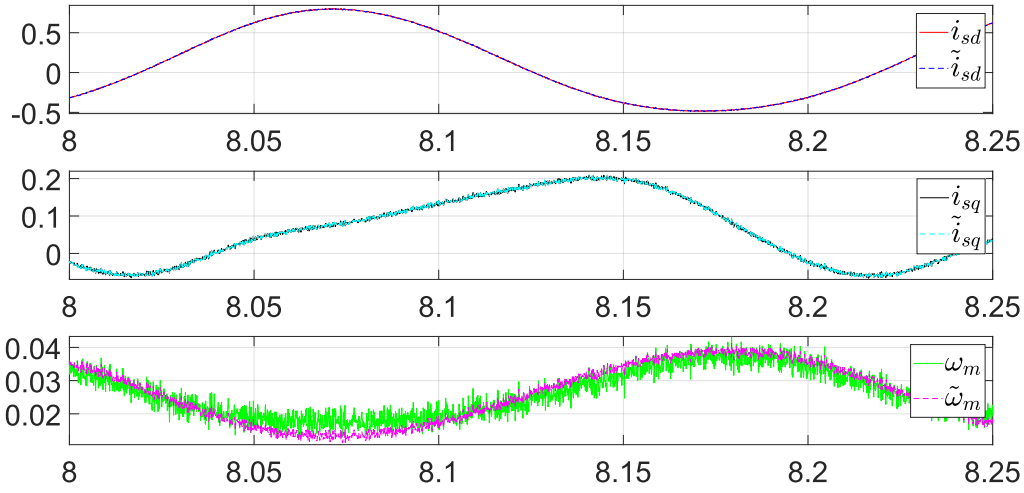

Fig. 5. State variable measurements PCA-filtered

While PCA-based filtering has little impact on low-noisy data, so on i_{sq} and i_{sd}, it shows good filtering capabilities on ω_m data, the mostly affected by noise being low in magnitude.

By setting $\lambda = 0.1$, the SINDy algorithm is applied to both noisy and filtered dataset, so to $[\widetilde{\boldsymbol{X}}, \boldsymbol{U}]$ and $[\boldsymbol{X}, \boldsymbol{U}]$; the following results are obtained:

- Noisy data $[\boldsymbol{X}, \boldsymbol{U}]$:

Result 2.1-5

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 46.0*id - 2.1*iq - 0.47*Tl + 180.0*vq - 170.0*wm + 2.8*id*iq - 100.0*id*wm +
2.3e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 0.16*id - 23.0*Tl + 26.0*iq + 280.0*vd + 1.1e+3*vq - 1.1e+3*wm - 45.0*id*iq - 500.0*id*wm
- 270.0*iq*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 2.7*iq - 2.7*Tl + 0.17*vd + 0.58*vq - 0.52*wm - 0.37*id*wm - 0.24*iq*wm (SINDy)
-
```

- Filtered data $[\widetilde{\boldsymbol{X}}, \boldsymbol{U}]$

Result 2.1-6

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 1.6*Tl - 44.0*id - 9.5*iq + 4.2e+3*vd + 42.0*vq - 36.0*wm + 15.0*id*iq - 71.0*id*wm +
2.4e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 76.0*Tl - 38.0*id - 3.5*iq + 390.0*vd + 2.4e+3*vq - 2.5e+3*wm - 45.0*id*iq - 370.0*id*wm
+ 180.0*iq*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 0.78*id - 1.7*Tl + 1.4*iq - 55.0*vd - 14.0*vq + 15.0*wm + 3.1*id*wm - 32.0*iq*wm (SINDy)
-
```

As can be seen, significant improvements are obtained in "*Eq.2*", where all the terms got closer to the true value, except for the "*id*wm*" one. On the other hand, a worse and less sparse identification is obtained for "*Eq.3*".

2.1.4 Modified SINDy: sparsity known a priori

If the structure of the model is already known, that is, if we know which bases should be zero and which should not, the problem can be simplified by defining proper library of candidate functions for each state equation. In this way, the knowledge of the system dynamics can be exploited: this represents a common case when dealing with physical systems. For example, it is known that the load torque input T_l is not present in the dq stator equations.

For example, let us suppose the sparse structure of the bases $\boldsymbol{\Theta}$ is supposed to be:

$$\boldsymbol{i_{sd} \rightarrow \Theta_1 = [i_{sd} \quad 0 \quad 0 \quad i_{sd} * i_{sq} \quad 0 \quad \omega_m * i_{sq} \quad v_{sd} \quad 0 \quad 0]}$$
$$\boldsymbol{i_{sq} \rightarrow \Theta_2 = [0 \quad i_{sq} \quad \omega_m \quad i_{sd} * i_{sq} \quad \omega_m * i_{sd} \quad 0 \quad 0 \quad v_{sq} \quad 0]}$$
$$\boldsymbol{i_{sq} \rightarrow \Theta_3 = [0 \quad i_{sq} \quad \omega_m \quad i_{sd} * i_{sq} \quad 0 \quad 0 \quad 0 \quad 0 \quad T_l]}$$

In this case, the identification reduces to solving the problem:

$$\xi_k = \text{argmin}_{\xi_k'} \|\dot{\boldsymbol{x}}_{\boldsymbol{t}} - \boldsymbol{\Theta_i}(\boldsymbol{X})\boldsymbol{\Xi_i}\|_2 + \lambda\|\xi_k'\|_1$$

Using the STLS algorithm with prior knowledge of the non-zero bases, the resulting model for the system dynamics is as follows:

$$\begin{cases} \frac{d}{dt} i_d = 4200\ v_d - 46\ i_d + 2400\ \omega_m i_q \\ \frac{d}{dt} i_q = 4200\ v_q - 46\ i_q - 2300\ \omega_m i_d - 4200\ \omega_m \\ \frac{d}{dt} \omega_m = 2.7\ i_q - 0.22\ C_m \end{cases}$$

This model was obtained from the SINDy method applied to the system, considering the known sparse structure for the calculation of the coefficients.

```
function Xi = sparsifyDynamics_JT(Theta,Theta1,Theta2,Theta3,dXdt,lambda,n)
    % Compute Sparse regression: sequential least squares
    Xi(:,1) = Theta1\dXdt(:,1); % Initial guess: Least-squares
    Xi(:,2) = Theta2\dXdt(:,2); % Initial guess: Least-squares
    Xi(:,3) = Theta3\dXdt(:,3); % 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
end
```

The result of SINDy by setting $\lambda = 0.1$ and using the noisy data, as in Fig. 3, is:

Result 2.1-7

```
-
Eq1: 4.2e+3*vd - 46.0*id + 2.4e+3*iq*wm
Eq1: 4.2e+3*vd - 45.0*id + 0.25*id*iq + 2.3e+3*iq*wm (SINDy)
-
Eq2: 4.2e+3*vq - 46.0*iq - 4.2e+3*wm - 2.4e+3*id*wm
Eq2: 1.2e+3*vq - 1.6*iq - 1.2e+3*wm - 36.0*id*iq - 460.0*id*wm (SINDy)
-
Eq3: 2.7*iq - 2.7*Tl
Eq3: 2.7*iq - 2.7*Tl (SINDy)
-
```

Still, SINDy has problems in reconstructing the dynamics of i_{sq}, but the identification has significantly improved with respect to the original SINDy, i.e. ***Result 2.1-2***. Of course, one must know the dynamics of the system before applying SINDy, so this might not be appliable in discovering new systems and variable correlations.

3 Time-Stepper Neural Network

Neural networks are computational models inspired by the structure and function of the human brain. They are used to solve complex regression and classification problems due to their ability to learn nonlinear relationships between inputs and outputs. In this project, a feedforward neural network, consisting of multiple layers of neurons, was used to learn the dynamic behaviour of the PMSM. The Time-Stepper NN algorithm was built in Matlab.

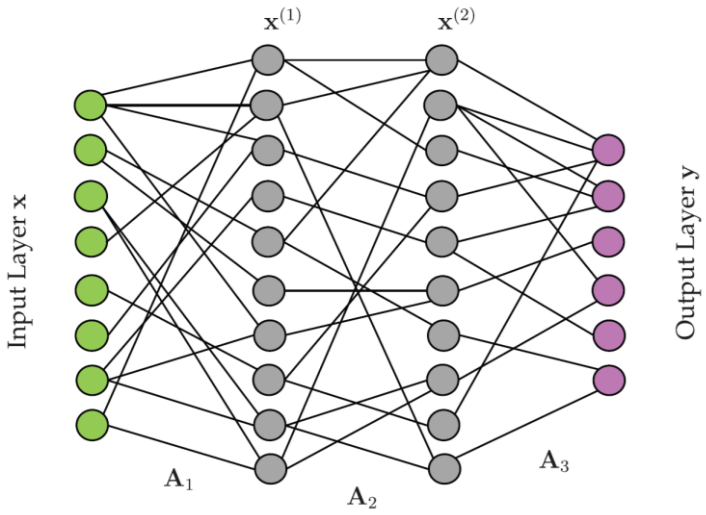

Fig. 6. Example of Neural Network

3.1 Objective

The goal of the project is to train a neural network to predict the future states of a PMSM based on the current conditions, using a dataset generated through simulations.

Both the state variables (such as motor currents and speed) and the control variables (such as the applied voltages and torque) are used as inputs,

$$input = [i_d(k) \quad i_q(k) \quad \omega_m(k) \quad v_d(k) \quad v_q(k) \quad C_m(k)]$$

while the future state variables represent the output.

$$output = [i_d(k+1) \quad i_q(k+1) \quad \omega_m(k+1)]$$

This allows the neural network to learn how the system evolves over time based on both the internal state and external control actions.

3.2 Model Implementation

The same Simulink model was used to simulate the motor's behaviour. The simulation was performed over a 10-second time interval, and the following variables were collected:

- Voltages (v_{sd}, v_{sq})
- Currents (i_{sd}, i_{sq})
- Speed (ω_m)
- Torque (T_l)

The variables are referred to the per-unit dynamic system presented in Section 1.1.

3.2.1 Data Generation for Training

To train the neural network, input and output data were generated through five different trajectories Fig. 7. The reference variables for each trajectory were varied to explore different motor behaviours.

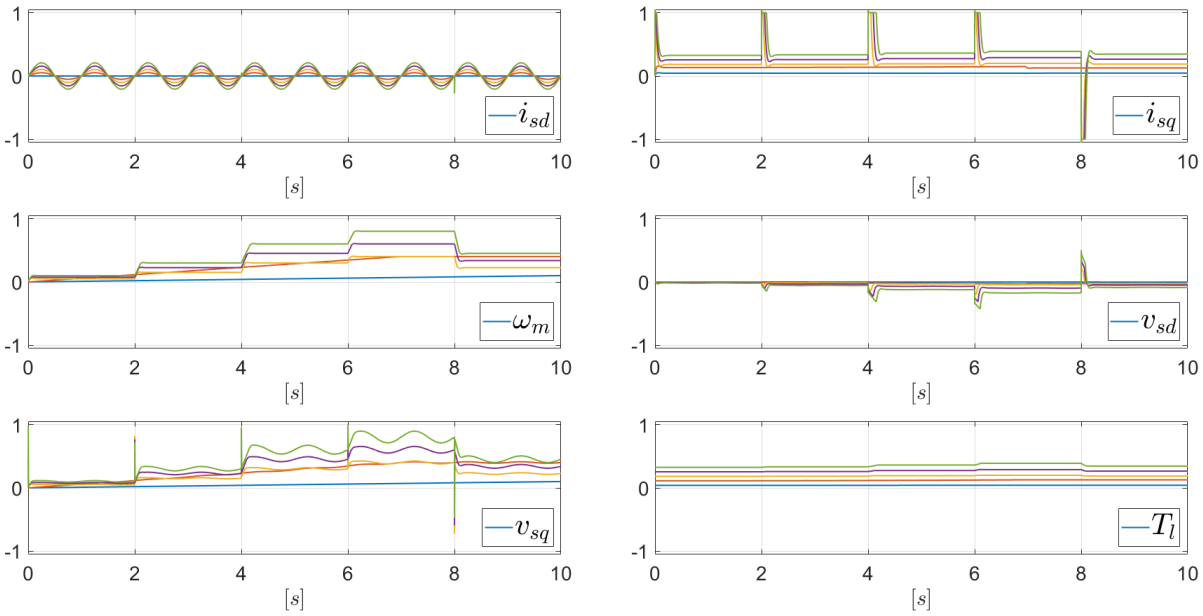

Fig. 7. Trajectory to train NN

The collected data were organized into matrices, where each row represents an observation, and the columns contain the state variables and inputs.

3.2.2 Neural Network Creation and Training

A feedforward neural network with 3 hidden layers, each containing 10 neurons, was created. The activation functions used were:

- Logsig: for the first two layers

- ReLU: for the third layer

The training parameters were set to run up to 50 epochs, using the backpropagation algorithm. The network was trained using the generated input and output data. A threshold on the Mean Square Error (MSE) was set to 1e-6, so to stop the training when this result was reached.

```
%% Get NN ready
% Define a target MSE error threshold for stopping training
target_mse = 1e-6;

% Update the training options to include the custom OutputFcn
options = trainingOptions('adam', ...
    'MaxEpochs', 8, ...
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 0.001, ...
    'Plots', 'training-progress', ...
    'Verbose', false, ...
    'OutputFcn', @(info) stopTrainingIfTargetMSEReached(info, target_mse));

layers = [
    featureInputLayer(6, 'Normalization', 'none')       % 6 input neurons
    fullyConnectedLayer(10)                   % 1° level FC with 10 neurons
    sigmoidLayer                              % activation function Sigmoid
    fullyConnectedLayer(10)                   % 2° level FC with 10 neurons
    reluLayer                                 % activation function ReLU
    fullyConnectedLayer(10)                   % 3° level FC with 10 neurons
    fullyConnectedLayer(3)                    % final level with 3 neurons
    regressionLayer];

% input & output data
input_tensor = input_data;
output_tensor = output_data;

%% NN training
net = trainNetwork(input_tensor, output_tensor, layers, options);
```

The training process used all the available epochs, struggling at reaching the threshold MSE.

3.3 Prediction and Validation

After training, the network was tested to evaluate its ability to predict the motor states. The predicted output was compared with the real data generated by the Simulink model, with another simulation, and graphs in Fig. 8 were used to visualize the correlation between predicted and real values.

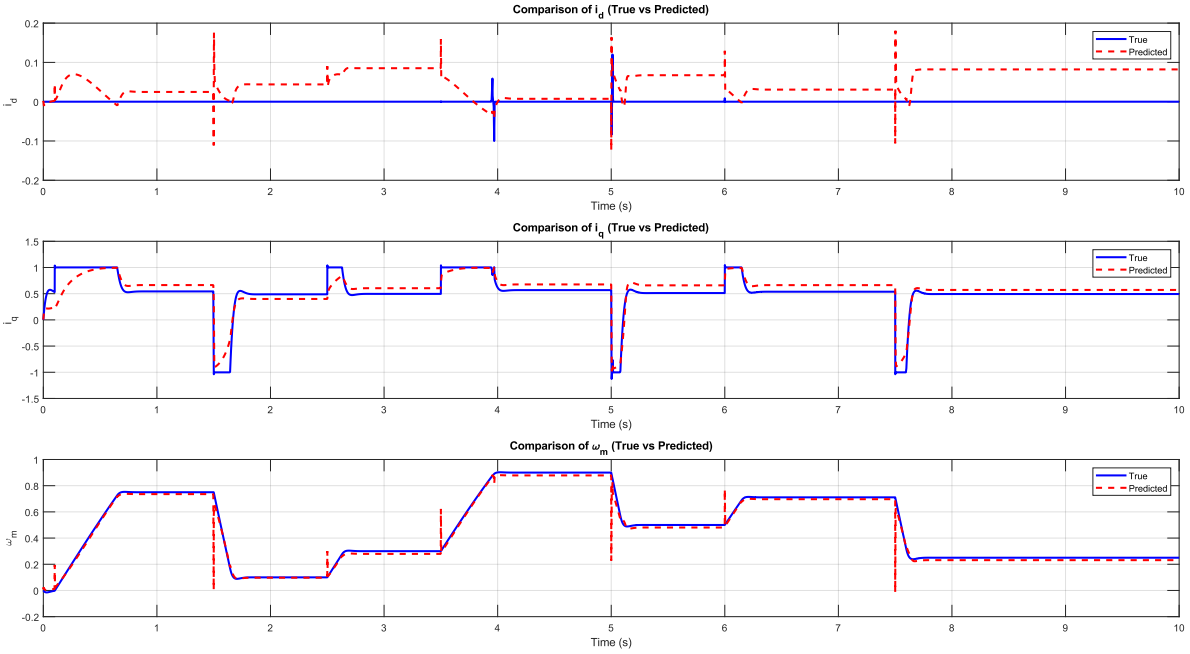

Fig. 8. Validation NN with Simulink

Simulations showed that the neural network was able to predict the values of the current i_{sd}, i_{sq}, and speed ω_m with good accuracy. The results were graphically represented to facilitate the comparison between the real and predicted values, showing good correlation, especially for the velocity values.

4 Conclusion and future developments

In this study, we explored two distinct approaches for the analysis of a permanent magnet synchronous motor (PMSM): SINDy and a Neural Network.

1. SINDy and Sensitivity to Noisy Data:

   The SINDy algorithm provided a good reconstruction of the system dynamics under ideal conditions, allowing for effective parameter estimation. However, accuracy significantly decreased with the introduction of noise, highlighting its sensitivity to imperfect data. This limitation poses a challenge for implementation in real-world scenarios, where data may be affected by various disturbances. Therefore, to make SINDy effective in practical contexts, it would be necessary to develop strategies that can manage data quality, as its current application appears less promising in the presence of noise.

2. Neural Network: Behaviour Prediction:

   In contrast, our objective with the neural network was not to estimate parameters but to predict the behaviour of the motor, i.e. to estimate the states at the next future step. Although we were unable to delve deeply into the workings, we proceeded through trial and error, achieving encouraging results. This empirical approach suggests that the neural network may tackle complex problems and handle noisy data more effectively than SINDy: analysis to assess this ability will be carried out. The robustness and relative ease of implementation of the neural network indicate that it could represent a promising avenue for future investigations.

3. Future Perspectives:

   Considering the results of the neural network, further testing will be beneficial, exploring advanced architectures and techniques that could enhance the prediction of motor behaviour. This represents an interesting direction for future research in the control of electric motors.