



**Politecnico
di Torino**

Master's Thesis in Mechanical Engineering
A.Y. 2025 / 2026

Physics informed LSTM network for hydraulic servo valve Digital Twin modeling

Supervisor:

De Martin Andrea

Co – supervisors:

Guida Roberto
Sorli Massimo

Candidate:

Sveglia Federico – 320124

Table of Contents

1	<i>Introduction</i>	3
1.1	Scope of the thesis	3
1.2	Motivation.....	4
1.3	Servosystems for flight control.....	7
2	<i>Neural Network approach</i>	26
2.1	Time series regression problem	26
2.2	Feedforward Neural Network architecture – The Forward Step	26
2.3	1 st order backpropagation of a FCNN – time independent backward step	30
2.4	Recurrent Neural Network and BPTT	32
2.5	Long-Short Term Memory architecture.....	36
3	<i>Physics Informed Neural Networks (PINNs)</i>	39
3.1	The flaw of a purely data driven approach	39
3.2	Physical description of the real system.....	39
3.4	Physical law implementation	61
4	<i>Hyperparameters search</i>	65
4.1	Introduction to hyperparameters.....	65
4.2	Variable learning rate and Adam optimizer.....	71
4.2	Bayesian optimization for hyperparameters search	75
5	<i>Model integration in MATLAB environment</i>	80
5.1	<i>dlarrays</i> – MATLAB smart vectors	80
5.3	Sensor data pre-processing	81
5.4	Training data analysis	93
5.5	Result analysis and limitations	94
6	<i>Conclusions</i>	98
6.1	Final considerations	98
6.2	Future work.....	98
	<i>References</i>	99

1 Introduction

1.1 Scope of the thesis

In modern industry and aerospace, the application of predictive maintenance has become a critical priority. Prognostics and Health Management (PHM) is the field of research that applies this core concept, seeking to anticipate components failures and estimating the Remaining Useful Life (RUL) of a mechanical system, thereby optimizing maintenance schedules and preventing catastrophic failures.

The integration of Physics-Informed Neural Networks (PINNs) into Prognostics and Health Management (PHM) and digital twin modeling is a rapidly growing field.

At the core of many advanced PHM strategies is the creation of Digital Twin — a virtual replica of a physical system that mirrors the real time operational state and functionalities. However, constructing a highly accurate digital twin for complex mechatronic systems presents significant logistical and computational challenges [19], [1].

Traditionally, data-driven digital twins rely heavily on high fidelity data streams from the real-world physical assets. This is often unfeasible because physical and operational constraint frequently make it impossible to place sensors exactly where they are needed. Specifically, invasive sensor placement can:

- Require halting the operation of vehicle, aircraft or industrial machinery, leading to unacceptable downtime.
- Incur in prohibitive operative costs associated with lost operational hours.
- Compromise or even destroy the structural integrity and normal functionality of the mechanical system itself.

Consequently, engineers are often left with sparse data extracted only from non-invasive sensors or existing Electronic Control Units (ECUs) or Programmable Logic Controllers (PLCs), which is rarely sufficient to implement a complete traditional digital twin using 1-D simulation, CFD or even FEM finite elements simulation. This might not be due to the scarcity of available data streams produced by sensors, but more related to the knowledge of the system itself.

These conventional simulation models have severe limitations when applied to real-world operational environments. They require exhaustive knowledge of the system's geometry, material properties, and boundary conditions. If the mechanical system is a legacy asset, highly degraded, or manufactured by a third party with proprietary constraints, the necessary baseline data to build an accurate FEM or CFD model is simply unavailable. Furthermore, these conventional models are sometimes computationally heavy, rendering them impractical for the real-time monitoring required by a digital twin. To bridge the gap between pure data-driven models that lack sufficient training data and pure physics-based models that lack complete system parameters, this thesis investigates the application of **Physics-Informed Neural Networks (PINNs)**.

PINNs embed known physical laws, expressed as ordinary or partial differential equations, directly into the loss function of the neural network. By doing so, the model is constrained to produce predictions that respect the underlying physics of the mechanical system. This hybrid approach drastically reduces the dependency on large, high-fidelity datasets.

Constructing a virtual clone of a specific mechanical system could be a challenge because the data required for this purpose might not match what is actually available from sensors; this could be labeled as a “Correlation Problem”, data stream from PLCs and ECUs could be very consistent, measurement accuracy might be really good, but they do not reproduce the complete physics

behind a complex mechanical system. Furthermore, they could be correlated only using a simple physical principle, which is insufficient for the level of detail required by traditional simulation modes.

In successive chapters, we will demonstrate how a PINN might be the useful instrument in this case, because they combine a solid framework to solve a regression problem to correlate sparse data from sensors, with a simple, yet sufficient, physical law that express the general principle that measured physical quantities must follow when the system is functional.

1.2 Motivation

In a recent study published by Taimun et. Al. in 2026 [19], the use of Machine Learning and Deep Learning based PHM strategies is analyzed. Traditional preventive maintenance strategies often consist of fixed interval inspections that don't consider operational conditions. They are labor intensive, and early stages faults are difficult to detect until they are large enough to cause machine failure. This will directly translate into maintenance costs and downtimes causing very high losses especially in process industry.

The article proposes a framework that makes use of Neural Networks as an instrument to construct a Digital Twin of the system, for the estimation of the Remaining Useful Life (RUL) with a superior accuracy when compared with more standard methods. In this case, the study focuses on a manufacturing system for textile industry.

The main challenge that a Digital Twin has to face when used for PHM purposes is State Synchronization. This is the capability of a virtual model to process a large amount of data consisting of live telemetry from the sensors and continuously update its current state, defined by the authors as:

$$S(t) = f(S(t-1), X(t), U(t))$$

Every state $S(t)$ of the digital twin is a function of the previous state $S(t-1)$, sensor data stream $X(t)$ and the operational commands $U(t)$ sent to the real machinery.

This Digital Twin must overcome three main challenges:

- **Synchronization Lag:** we will completely rely on a consistent and low latency data stream of measured physical quantities and operational commands logs, transmitted in real time to the model. If the latency between physical data collection and the virtual update is too high, the digital twin generates predictions based on outdated information.
- **Data Quality and Preprocessing:** raw sensor signals always contain environmental and operational noise. Before the data is synchronized with the Digital Twin, it must pass through noise filtering and normalization steps to improve the data quality. The data from sensor and control channels is often normalized, to avoid the scaling effect due to the different orders of magnitude (for example a position signal expressing millimeters will be very different from a pressure signal in pascals coming from a hydraulic servo valve).
- **Computational Cost:** this is where DL based methods shine as a modeling choice. Maintaining state synchronization requires recalculating the system's physics at high frequencies. Continuously updating a traditional Finite Element Method (FEM) or

Computational Fluid Dynamics (CFD) mesh to match real-time sensor inputs is computationally prohibitive. A trained PINN, however, can execute these complex, non-linear physical recalculations in fractions of a second, rendering real-time synchronization achievable, even when only poor computational power is available.

Table 1: PHM framework summary, adapted from Taimun et. al. research, 2026. [19]

Framework Layer	Description	Role
Data Acquisition	Connected sensors, PLCs, ECUs, data stream.	State and command logs in real time. Physical quantities monitoring.
Digital Twin	Simulation asset, trained model performing evaluation or inference.	State Synchronization of the system.
... DL or ML Based Fault Detection ...		
Predictive analytics and support to decisions (RUL estimation and decisions)		

The layers related to the RUL estimation and decision making to overcome any development of an early fault are beyond the scope of this thesis. We will only focus on Digital Twin modelling. Besides that, by comparing the detection accuracy, implementing a Digital Twin Method outperforms every other strategy.

Detection accuracy is the ability of a predictive maintenance strategy to correctly identify the faulty behavior of a system and do not trigger false alarms when the actual fault is not present. It is defined as follows:

$$A_d = \frac{TP + TN}{TP + TN + FP + FN}$$

True positives correspond to the case when the strategy correctly detects a real fault, while true negatives means that the system is working properly and there are no false alarms. The summation of true negatives, true positives, false negatives and false positives correspond to all the population. A_d is in fact the probability of the strategy to catch a faulty behavior only when there is one in the real system.

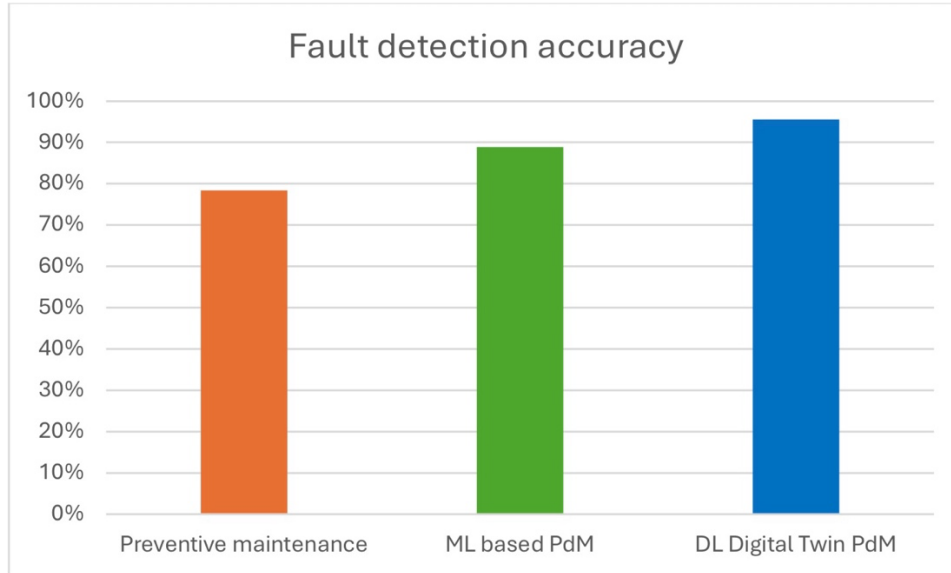


Figure 1: Fault detection accuracy performances of the 3 methods. – Adapted from Taimun et. al. research, 2026. [19]

Another relevant advantage of this predictive maintenance strategy using digital twins is being able to carefully plan maintenances to further reduce unplanned downtimes and money losses. It is known that system availability is a performance indicator of production equipment:

$$A_s = \frac{MTBF}{MTTR + MTBF}$$

It takes the mean time between failures (measured in uptime hours) and divides it for the entire system uptime plus the downtime due to maintenances (so a theoretical uptime where the system never fails, 100% availability).

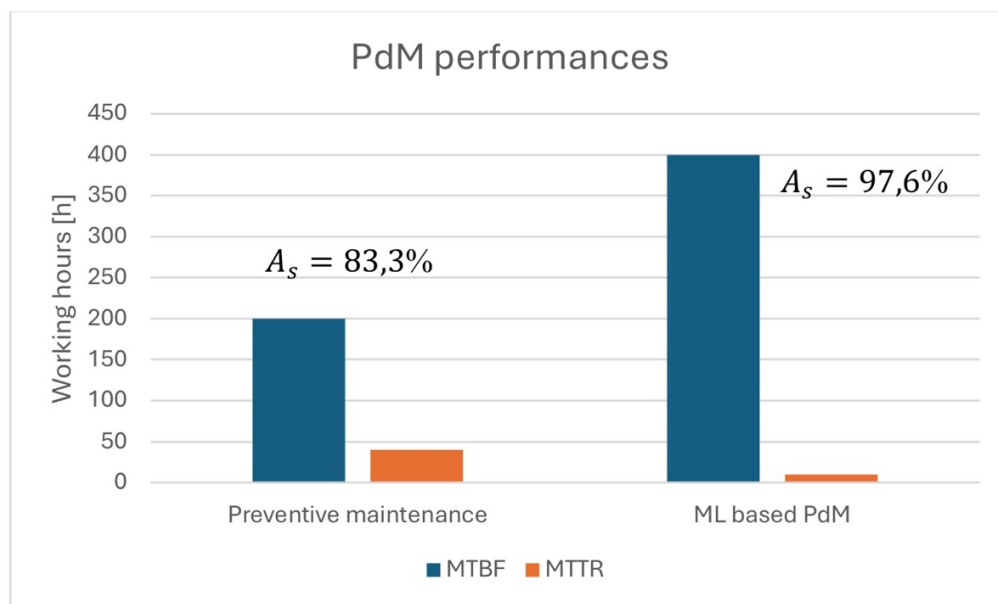


Figure 2: maintenance strategy performances. – Adapted from Taimun et. al. research, 2026. [19]

1.3 Servosystems for flight control

1.3.1 Introduction to Flight Controls

As aircraft became more advanced, control surfaces were increasingly actuated controlled by computers, through fiber optic cables or traditional buses. This system, known as “fly-by-wire,” replaces the physical connection between pilot commands and flight control surfaces with an electrical interface. Additionally, in larger and faster aircraft, controls are enhanced by hydraulically or electrically actuated systems. In both fly-by-wire and boosted controls, the sensation of the control reaction is conveyed to the pilot through simulated means.

Flight controls of an aircraft are separated into 2 categories, primary and secondary controls. The first are controls used to safely maneuver the aircraft during flight, while the second consist of the secondary control system and grand stability to the aircraft while also relieving the pilot from excessive inputs.

1.3.2 Primary Controls

Aircraft control systems are designed to perform adequately at higher air speeds providing a natural feel with a rapid response to control applications.

An actuation of a primary control (ailerons, elevator or rudder) changes the pressure field around the airfoil, this will generate a different aerodynamic lift and drag configuration and allows the pilot to control the aircraft about the 3 axes of rotation.

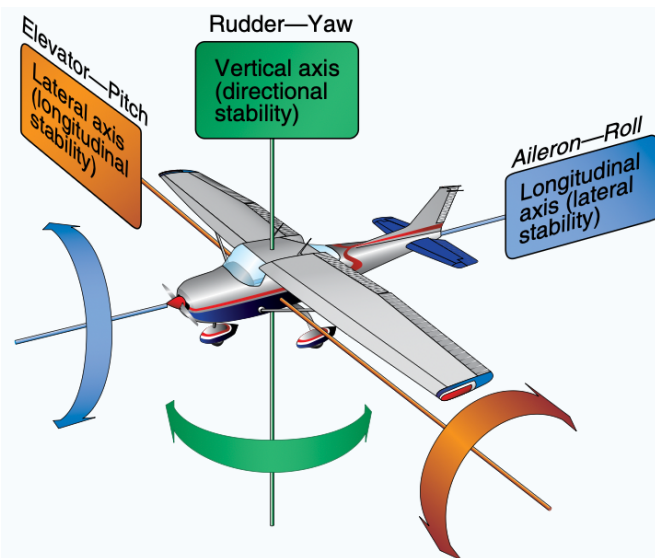


Figure 3: the 3 main axes of rotation of an aircraft.

In normal conditions, an aircraft control system should limit the maximum of deflection of control surfaces, to avoid overstress on the servo mechanisms from excessive feedback forces.

AILERONS

Ailerons control the roll around the longitudinal axis. They are mounted on the outbound trailing edge of every wing, and they actuate in opposite direction to generate *differential lift* resulting in

a rolling torque on the aircraft. This means that if the left aileron is deflected downward, it generates more lift than in its resting position, while the right aileron is deflected upward generating less lift.

It is also possible to demonstrate that ailerons can produce unwanted *differential drag* resulting in the phenomenon of *Adverse Yaw*, especially in aircraft with long wingspan.

To avoid the condition where a wing tip stalls during a roll, at near stalling speed the downward aileron at maximum deflection has strictly limited effect in producing more lift due to wing geometry. A first approach consisted of adopting *differential ailerons control*, as shown in Fig. 2 (US patent 1,565,097 – [22]).

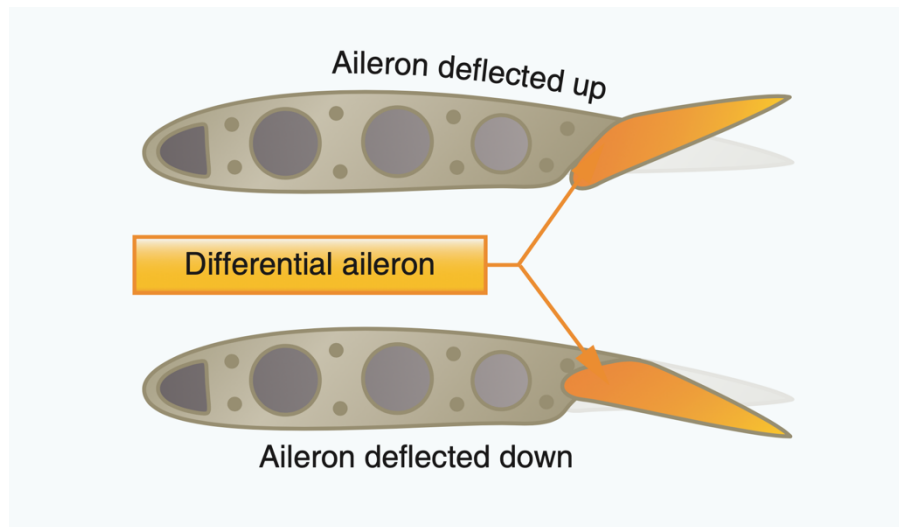


Figure 4: differential aileron control.

By Federal Aviation Administration - Pilot's Handbook of Aeronautical Knowledge, page 6-4, Public Domain.

In this situation, we recover some of the differential lift by increasing deflection on the upward aileron. This creates higher drag and Adverse Yaw.

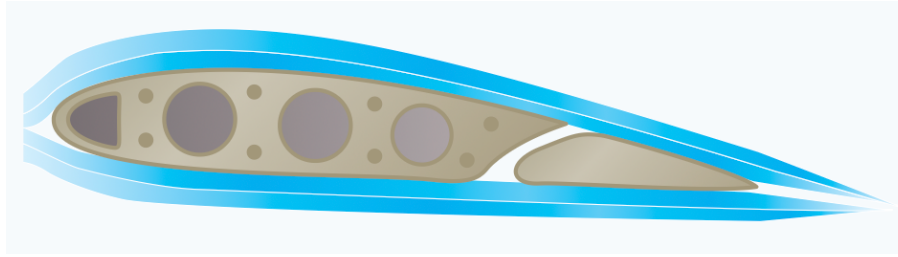


Figure 5: neutral Frise type aileron.

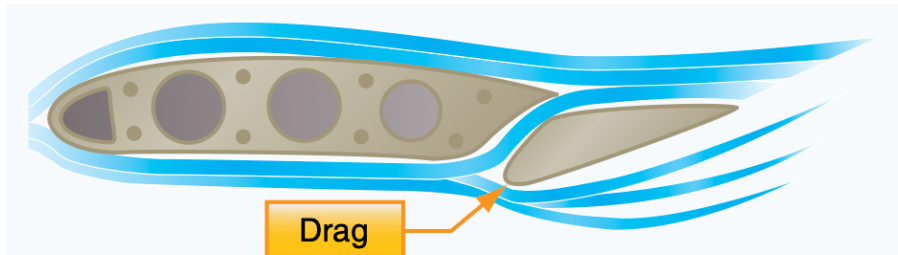


Figure 6: upward Frise type aileron.

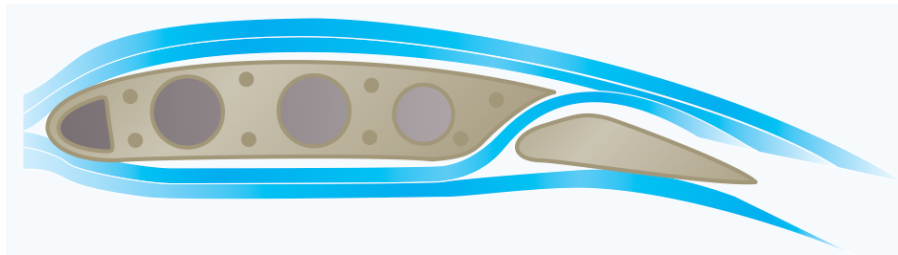


Figure 7: downward Frise type aileron.

In Figg. 2-4 it is detailed the increased drag that a Frise – type aileron produces in lower amount with a downward configuration. This design, developed by Eng. Leslie George Frise (1897 – 1979), adopts an aileron pivoted at about 25% to 30% of chord length of a wing, in proximity to its bottom surface. When a Frise type aileron is upturned, the leading edge protrudes under the wing, its drag helps the trailing edge to move up.

It creates a vane between itself and the rest of the wing. The high-pressure channel from beneath the wing is partially deflected into this vane and against the upturned aileron towards the trailing edge. This creates a downward force on the wing in a more efficient manner with less drag, countering the adverse yaw effect.

Although the main technique to counter adverse yaw is through rudder input.

ELEVATOR

The elevator controls pitch rotation of the aircraft about the lateral axis.

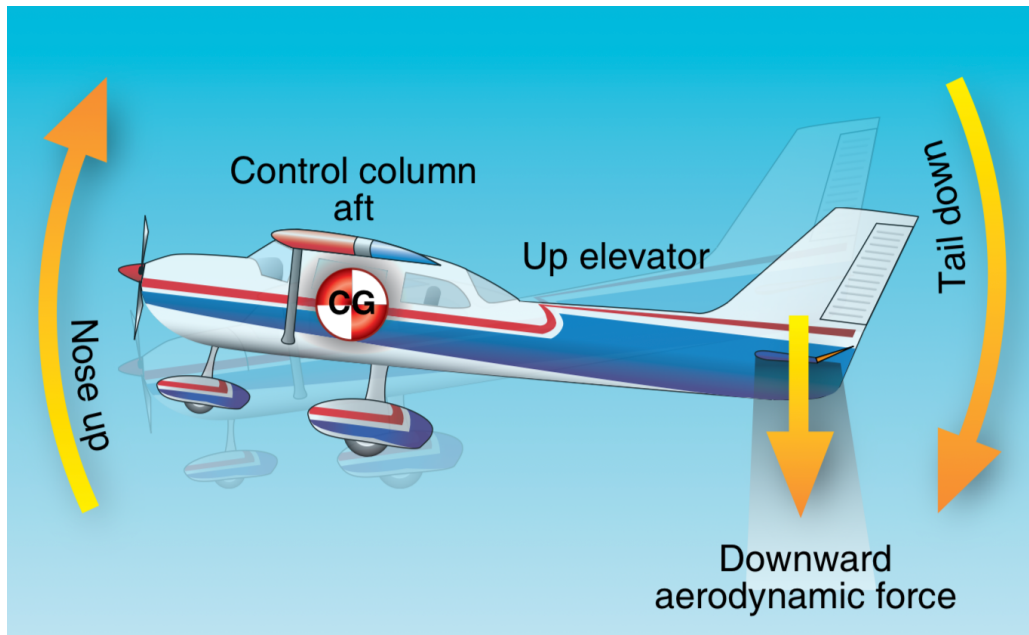


Figure 8: upward elevator effect on the aircraft. By Federal Aviation Administration - Pilot's Handbook of Aeronautical Knowledge, page 6-5, Public Domain.

This is an *elevator up* command; it creates a downward force on the tail rotating the aircraft as shown in Fig. 6. The pitching movement occurs about the center of gravity (CG), but the actual center of rotation of the aircraft varies with flying conditions.

Opposite behavior happens during *elevator down* command.

Elevator control surfaces may be installed away from aerodynamically polluted air flow (like downwash effect) from propellers. For example, in T-Tail configuration. Seaplanes adopt this tail design to keep control surface as far away from water as possible.

This control should have enough authority to keep the aircraft nose up when landing, granting sufficient (negative) lift force with low air speed.

RUDDER

The rudder controls the aircraft rotation about its vertical axis. In traditional aircrafts, it is usually controlled by the left and right rudder pedals. When pushing the *right* pedal, the rudder control surface rotates to the right, generating an opposing force on the tail that moves it left, effectively yawing the aircraft with nose to the right. This is the main control used to counter adverse yaw generated by ailerons.

1.3.3 Secondary Controls

Secondary controls surfaces consist of flaps, leading-edge devices, trim systems and wing spoilers. Flaps and spoilers are also known as *high-lift devices*.

FLAPS

These are the most common trailing edge device used on aircrafts. They are hinged at the trailing edge of a wing, and they can greatly increase both lift and drag for any angle of attack (AOA or α) or retracted during high cruising air speed. They divide in 4 types, plain, split, slotted and fowler.

Plain flaps are the simplest of them all, it increases the wing camber significantly increasing the lift coefficient C_L . At the same time it increases drag and recovers pressure in proximity of the wing trailing edge, generating a pitch down moment. They are mostly used on aircraft with thin wings, where is not possible to slot a complex actuator mechanism.

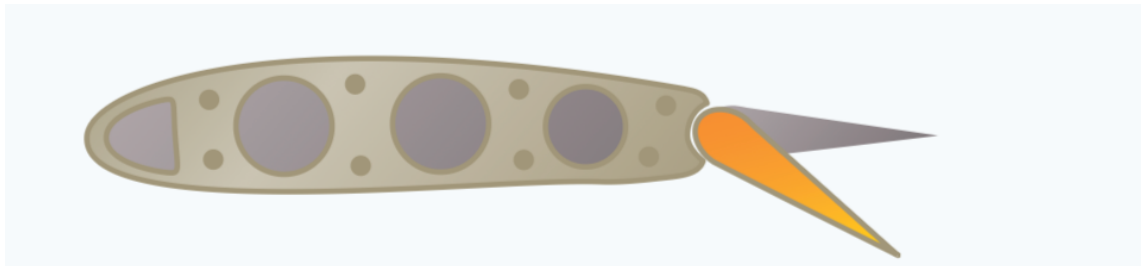


Figure 9: scheme of a plain flap – Pilot's Handbook of Aeronautical Knowledge, page 6-9, Public Domain.

Split flaps produce additional lift when compared to plain flaps, but at the cost of greater drag. They are deflected from wing lower surface.

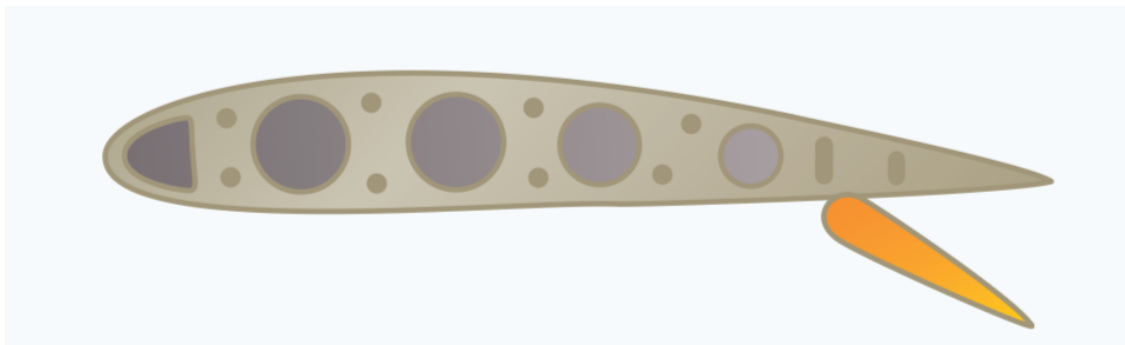


Figure 10: scheme of a split flap – Pilot's Handbook of Aeronautical Knowledge, page 6-9, Public Domain.

Slotted flaps are the most common on larger aircrafts, they allow the formation of a vane between the flap leading edge and the flap well on the wing, this allows part of the high-pressure stream from beneath the wing to pass through this vane and attach to the flap top surface. This delays pressure recovery from flow separation and additionally increases C_L compared to other types.

Known variations to slotted flaps are double, or even triple slotted flaps. They provide even more lift, and additional vanes to delay flow separation and pressure recovery on top of the wings.

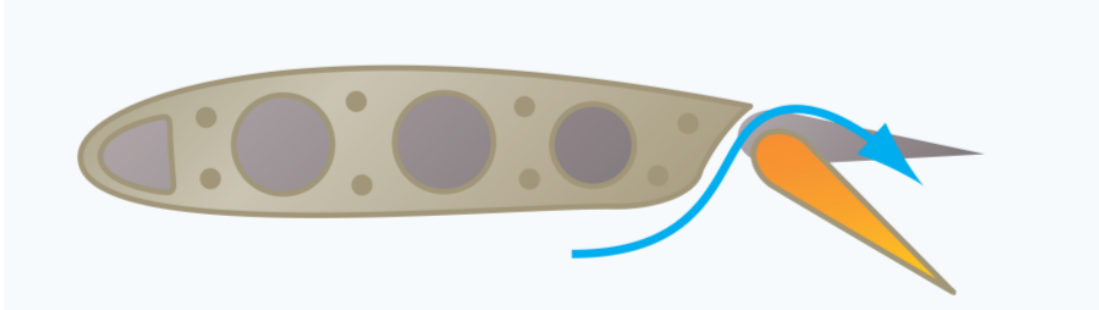


Figure 11: scheme of a single slotted flap – Pilot's Handbook of Aeronautical Knowledge, page 6-9, Public Domain.

Fowler flaps are a subtype of slotted flaps that account for increased drag problem with a different working mechanism that increases wing surface. The fowler flap does not rotate in the first actuation stage; it instead slides backwards on a track to increase wing's camber and consequently lift coefficient. This effectively minimizes drag. Fowler flaps can rotate in the second stage of actuation, further increasing C_L at the cost of drag force.

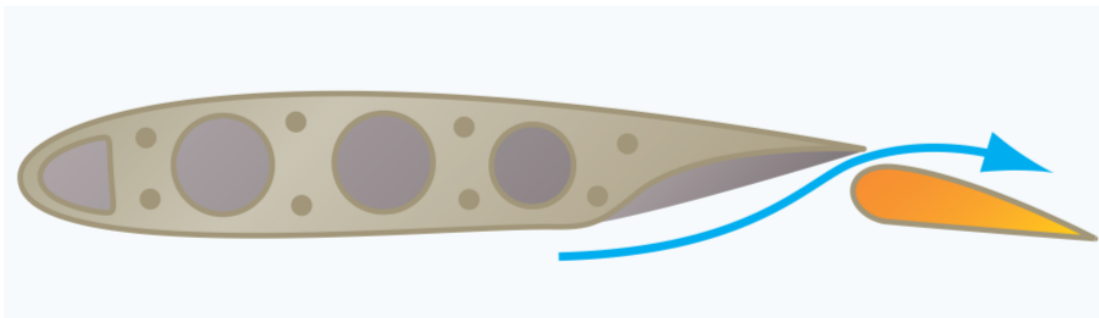


Figure 12: scheme of a fowler flap – Pilot's Handbook of Aeronautical Knowledge, page 6-9, Public Domain.

Other leading-edge devices exist with the role of increasing wing camber, partially or completely detaching the flow from wings to generate pressure recovery on bottom surface. They are common on large commercial aircraft or faster fighter jets and used during maneuvers at higher angles of attack.

SPOILERS

Spoilers are high drag devices most often used in controlled descending maneuvers during a landing process. Their role is to *spoil* the airflow on top surface of wings, decreasing lift with flow pressure recovery. The increase in aerodynamic drag force is wanted in this case to slow down the aircraft. They can be used for roll control as well, to turn left, for example, the left-wing spoiler could be raised. This will cause that left wing to lose lift, rolling the aircraft, and the increase of drag is sufficient to generate the wanted yaw rotation of a banking turn.

Spoilers are also used to optimize weight transfers on the landing gear wheels improving brake effectiveness, by partially destroying lift on the selected wing.

TRIM SYSTEMS

Aircrafts are designed to be operated with a limited combination of primary and secondary controls. Trim systems are adopted to relieve the pilot of a constant pressure on flight controls, minimizing the pilot workload. They divide in trim tabs, balance tabs, and adjustable stabilizers.

Trim tabs, mostly used for small aircraft, consist of a single tab attached to the trailing edge of an elevator. They provide a biased control of a primary surface.

In the example of an elevator, the final trim tab could be positioned completely downturned to force a nose-down pitch to the aircraft. In this case, any input to the primary elevator could be used to finely control the descent maneuver during a landing procedure. On the contrary, an upturned trim tab could be used during a climb.

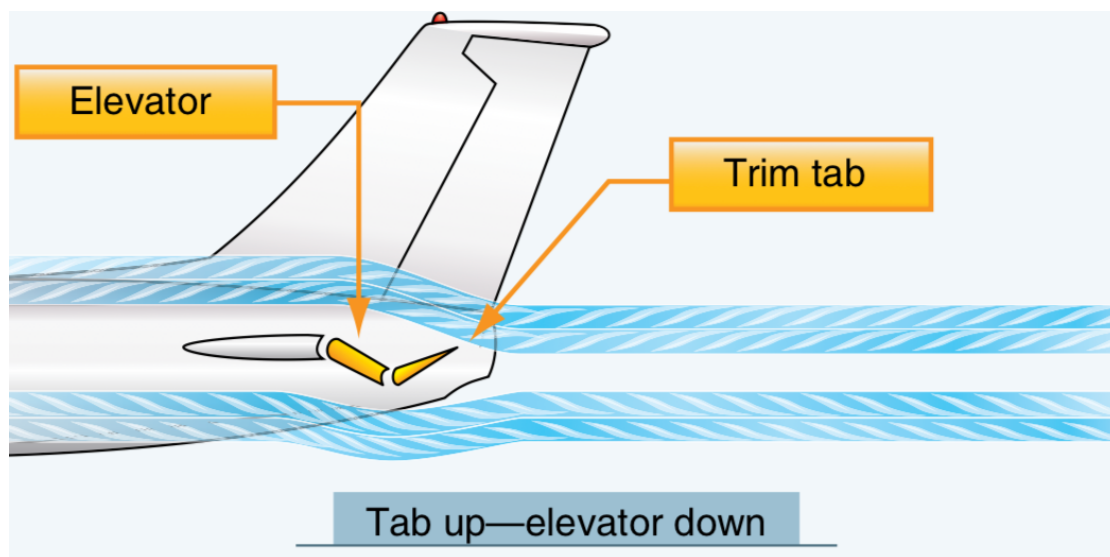


Figure 13: example of a trim tab upturned, while a primary elevator down command is executed – Pilot's Handbook of Aeronautical Knowledge, page 6-11, Public Domain.

Balance tabs are used to relief the pilot or the servo system actuating the primary control surface, from excessive load. They could be analogous to trim tabs but always moving to the opposite direction of the primary control surface.

Using pressure recovery of air striking the main surface, they help with its rotation using air rather than pure mechanical or hydraulic force.

Adjustable stabilizers are used in some aircraft to provide a bias correction to a primary surface. The trim force is generated by an internal mechanism that changes the camber by rotating the first section of the stabilator surface, where the leading edge is located.

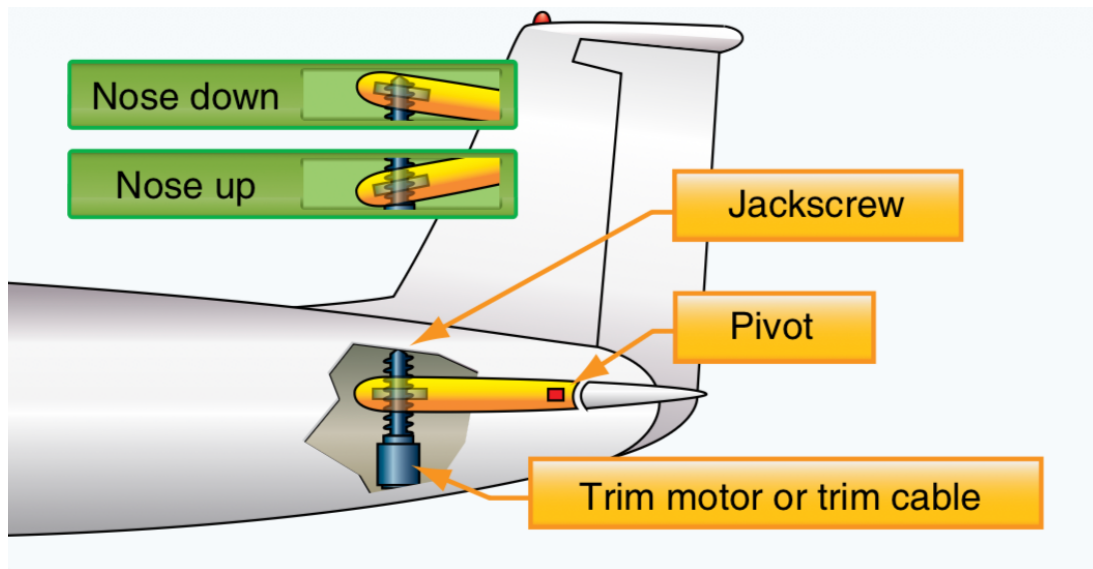


Figure 14: adjustable stabilizer simplified scheme, used on a jet transport aircraft – *Pilot's Handbook of Aeronautical Knowledge*, page 6-12, Public Domain.

1.3.4 Auxiliary servosystems – fly by wire

In research from 2017, by Jean-Charles Maré and Jian FU [11], signal-by-wire and power-by-wire servo systems are reviewed for applications declined to more electric aircraft.

An aircraft is equipped by multiple users of auxiliary non propulsive power, supplied by hydraulic, electric and pneumatic power networks. Their main source comes from turbine inside the propellers or a ram air turbine using air at opposite relative speed to generate power.

Actuators serve flight control surfaces, landing gears and other hardware utilities, and on large commercial aircraft they may require up to hundreds of kW of power, depending on mission profile.

Hydraulic servo actuation has been an established technology for over 50 years, and their use has been evolved extensively with closed loop servo systems for primary flight controls or landing gear actuation. The first main reason is related to pilot muscular power that have been reached quickly with the increase in speed and size of aircrafts. The second is to decrease the pilot intellectual workload, mainly with autopilot functions, control compensations and a more general increase in flight efficiency and performance.

The introduction of flight computer control (FCC) allowed more advanced implementation of control laws of servosystems. The evolution from mechanical controls to full electric command took several years to fully establish. Thanks to the introduction of reliable servo valves a progressive transition to signal-by-wire was enabled as summarized in Table 1.

Table 2: evolution of fly-by-wire controls on European commercial aircrafts. Adapted from J. C. Maré and J. FU, 2017 [11].

Years: First Flight EIS (entry into service)	Aircraft Model	Relative advancement (from previous model)
1969 1975	Sud Aviation – British Aircraft Corporation / Concorde	Analog fly-by-wire with mechanical backup. Analog electrical signaling from cockpit to actuators.
1972 1974	Airbus / A 300 B	Analog fly-by-wire for non- essential functions, analog signaling from cockpit to actuators. Position servo control from a primitive FCC in the cockpit.
1982 1983	Airbus / A 310	24 fly-by-wire actuators controlled by 5 computers. Roll control only through spoilers instead of low-speed ailerons. Electrically signaled trims for rudder and ailerons.
1987 1988	Airbus / A 320	Fly-by-wire on 3 axes, 7 FCC computers. Mechanical signaling between rudder actuators and yaw damper, used to prevent yaw-roll oscillations (Dutch Roll) during cruise. Introduction of the side stick directly talking with FCC (via FbW) for a more streamlined control.
1991 1993	Airbus / A 330 – A 340	5 digital computers Any mechanical signaling for rudder control has been removed and replaced by an electrical analog backup control module (BCM).
2005 2007	Airbus / A 380	6 digital computers. Removal of mechanical backup signaling for trim stabilizers in favor of full fly- by-wire control.
2013 2015	Airbus / A 350	6 digital computers. Position controlled servo systems for actuators.

At each step, the use of electrical signaling saved significant amount of mass [11]. Total removal of mechanical signaling mechanisms was first established in military aviation, except for most helicopters, with only few adoptions of full FbW controls with zero mechanical backups [18].

1.3.6 Position controlled hydraulic servo system

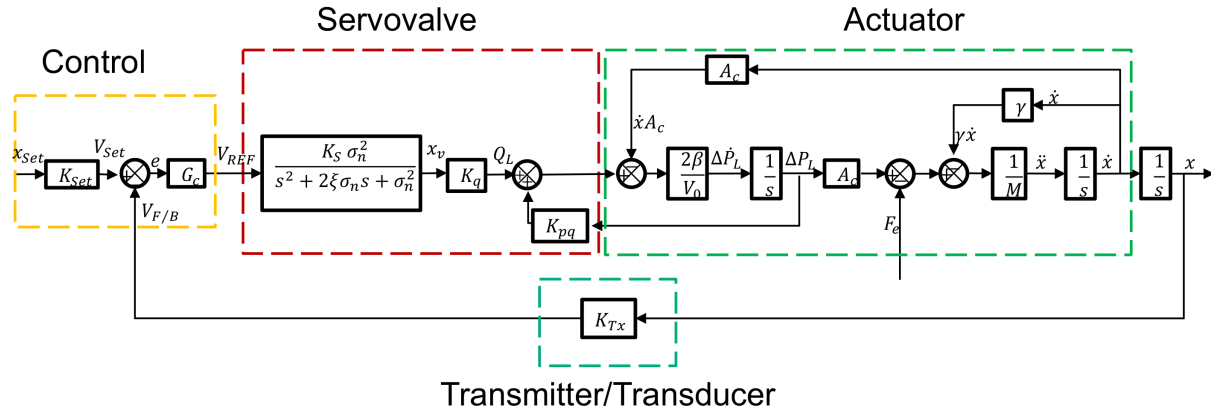


Figure 15: block diagram of a position controlled EHSA.

CONTROL BLOCK

In this simplified diagram it is visible the position controlled EHSA entire architecture. On the left, the *Control* block governs the reference signal V_{ref} , result of the error between the set command from the cockpit, transmitted via fly-by-wire, x_{set} and the error signal obtained measuring the final effective position on the feedback loop.

This is a *zero-order* model because the final control gain G_C is proportional, no integration, nor differentiation is present. A more realistic implementation shall include at least integrative or even a derivative control step to increase stability to low and high frequency error variation. More likely, when dealing with a hydraulic servo system, it is generally adopted a PI control, or a PID with a heavily lowpass-filtered derivative term to not amplify chatters and pressure transients.

The PID control is a complete error compensation strategy that accounts for the 3 main characteristics of a servo system.

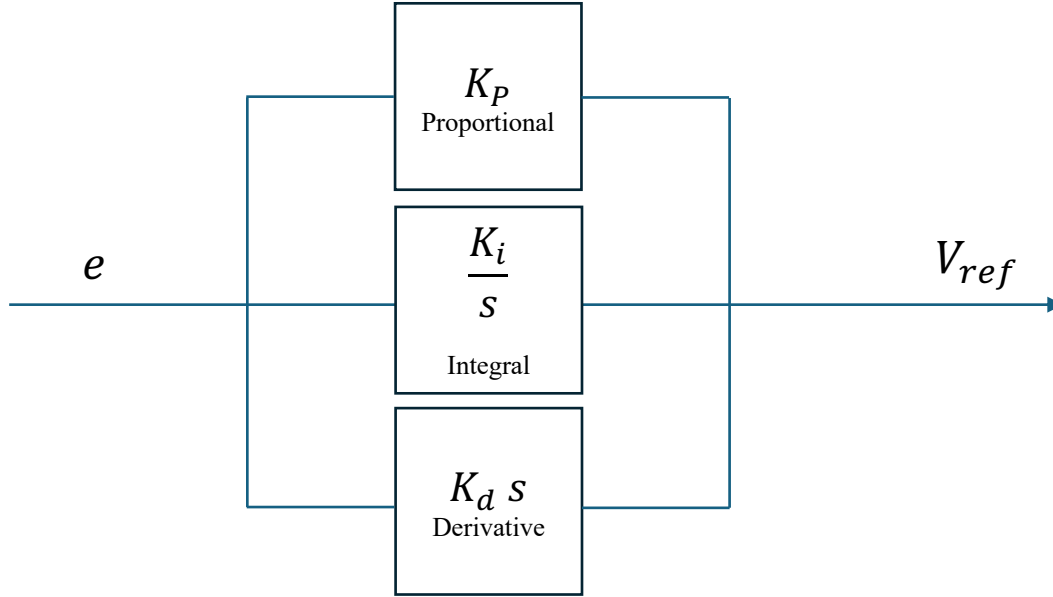


Figure 16: block diagram of a PID control.

The error signal $e = V_{ste} - V_{fb}$ is processed in parallel by each block of the PID control, then contributions are summed together to obtain V_{ref} . The equation is as follows:

$$V_{ref} = e \cdot K_p + e \cdot \frac{K_i}{s} + e \cdot K_d s = e \cdot PID$$

This can be rewritten as a sum of 3 transfer functions to obtain PID. This control strategy is the composition of PI and PD controls; this result is important while performing the frequency tuning.

The PI control stand for proportional and integrative:

$$V_{ref}(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t) dt$$

The Laplace transform of this equation is:

$$V_{ref} = e \cdot \left(K_p + \frac{K_i}{s} \right)$$

We can also write just the PI transfer function:

$$PI = \frac{K_p}{s} \left(s + \frac{1}{T_i} \right)$$

$$T_i = K_p / K_i$$

The variable T_i gives a quick representation of the *cutoff frequency* at which the integral part stops functioning and the error is only gained by the proportional constant K_P .

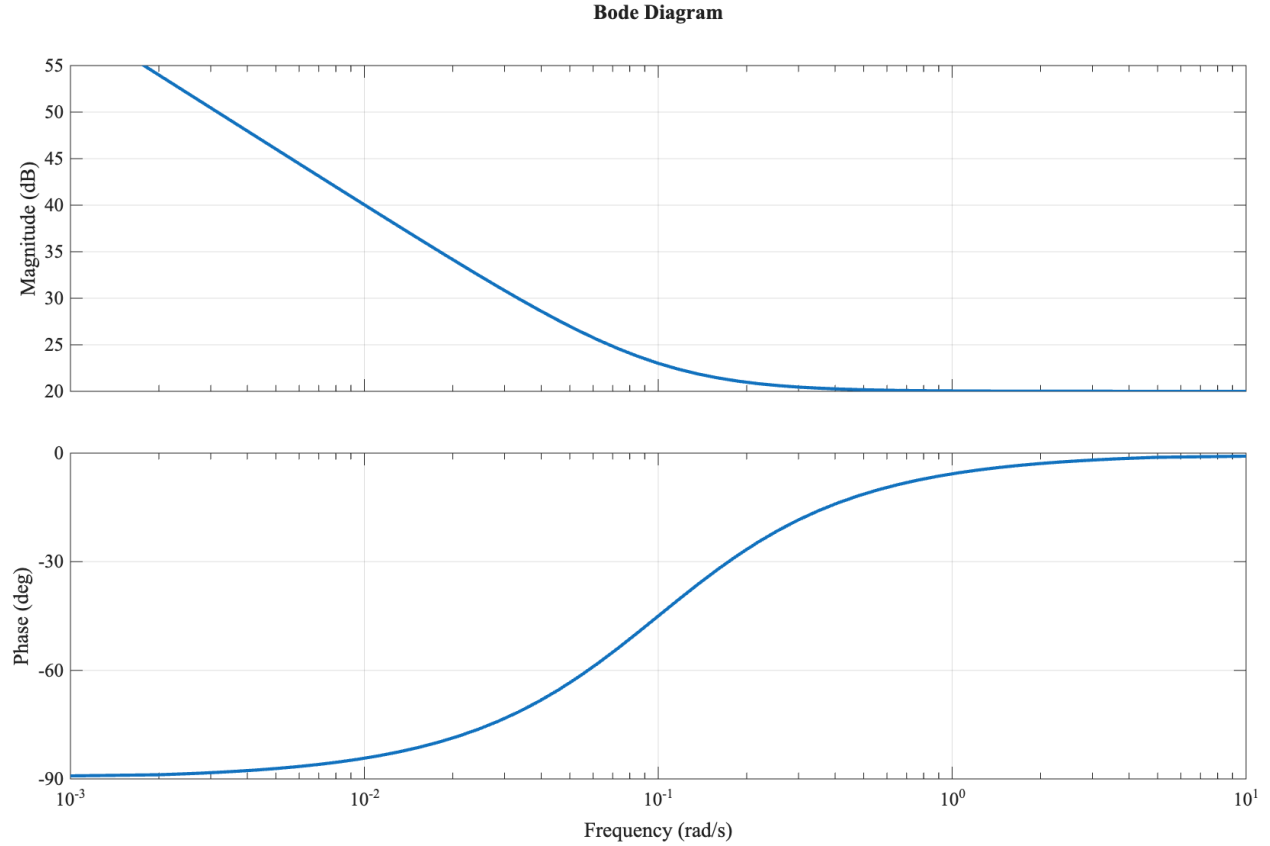


Figure 17: bode diagram of a PI control. $K_i = 1$, $K_p = 10$, $\omega = 0,1$ rad/s.

In this diagram the bode plot of a PI control with $K_P = 10$ and $K_i = 1$. In this situation we obtain:

$$\omega_c = \frac{1}{T_i} = K_i/K_P = 0,1 \text{ rad/s}$$

If we increase K_i , we obtain both a stronger reaction to lower frequencies, and a higher *cutoff frequency*.

An integral control considers contribution from initial to the current time as the servo system continues to compensate the error. This is useful when feedback matches the set signal, in this case the error would be zero and the system stops reacting. In presence of an external constant disturbance, the error will immediately rise again if the servo system stops correcting, triggering another error signal and another correction. This is typical of a control strategy which is only proportional. An integrative strategy keeps the servo system reacting even if the error is zero, since the integral cannot disappear. The main disadvantage is the integral part is slow to react, since its phase starts from -90° , causing correction overshoots. To solve this problem, a derivative strategy should be adopted.

The second part of the PID is a PD, proportional and derivative control:

$$V_{ref}(t) = K_p \cdot e + \frac{de(t)}{dt}$$

Laplace transforming and just writing the transfer function we obtain:

$$PD = K_p + K_d s = K_p(1 + T_d s)$$

Again, in this case we obtain the derivative time constant T_d , whose inverse is the derivative *cutoff frequency* at which the derivative control comes into play and reacts to error fluctuations.

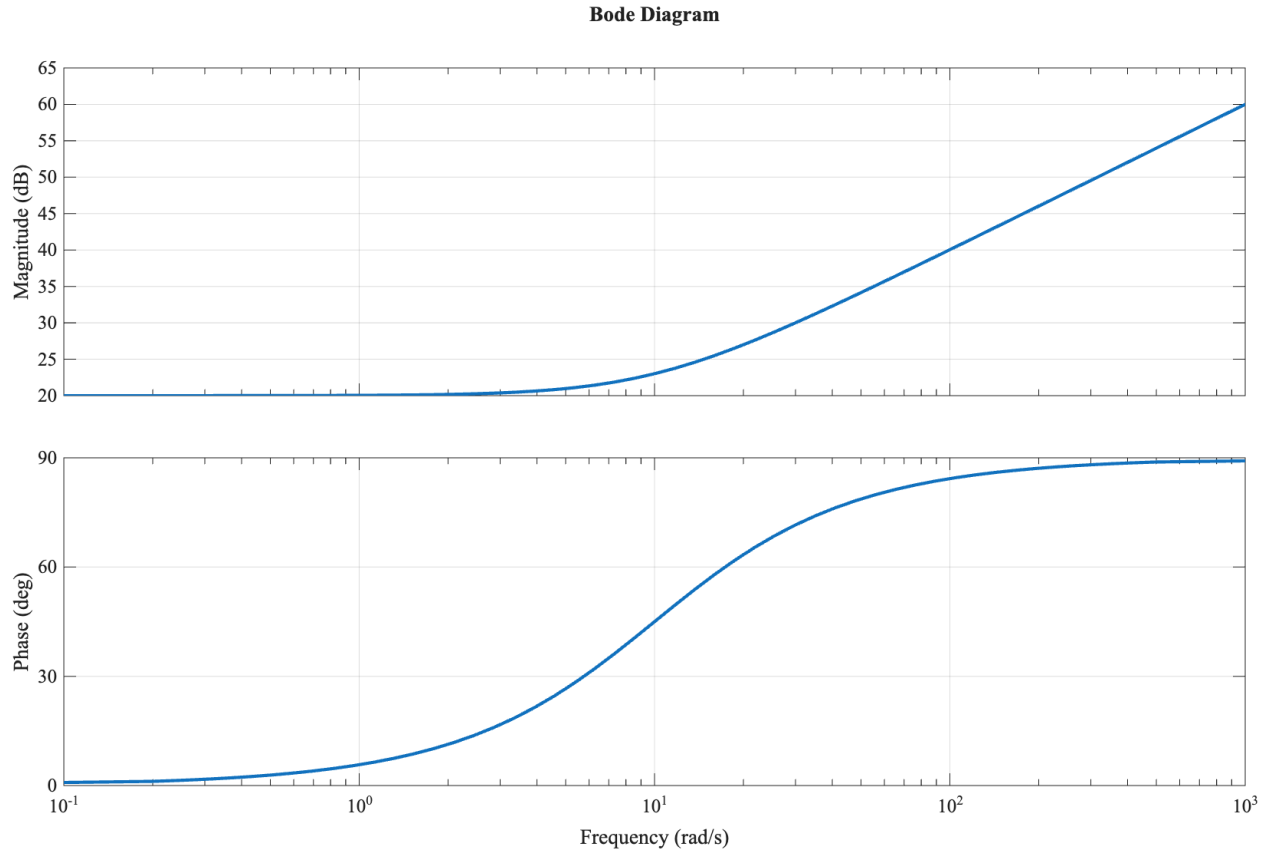


Figure 18: Bode plot of a PD control, $K_p = 10$ and $K_d = 1$, $\omega = 10$ rad/s.

We can compute the *cutoff frequency* as done before:

$$\omega_c = \frac{1}{T_d} = \frac{K_p}{K_d} = 10 \text{ rad/s}$$

The derivative control does not consider lower frequency content, but it aggressively compensates for rapid error oscillations. Since it reacts not to the error itself, but rather to its derivative, it has a $+90^\circ$ phase shift at regime.

The combination of these 2 control strategies forms the PID control, proportional, integrative and derivative.

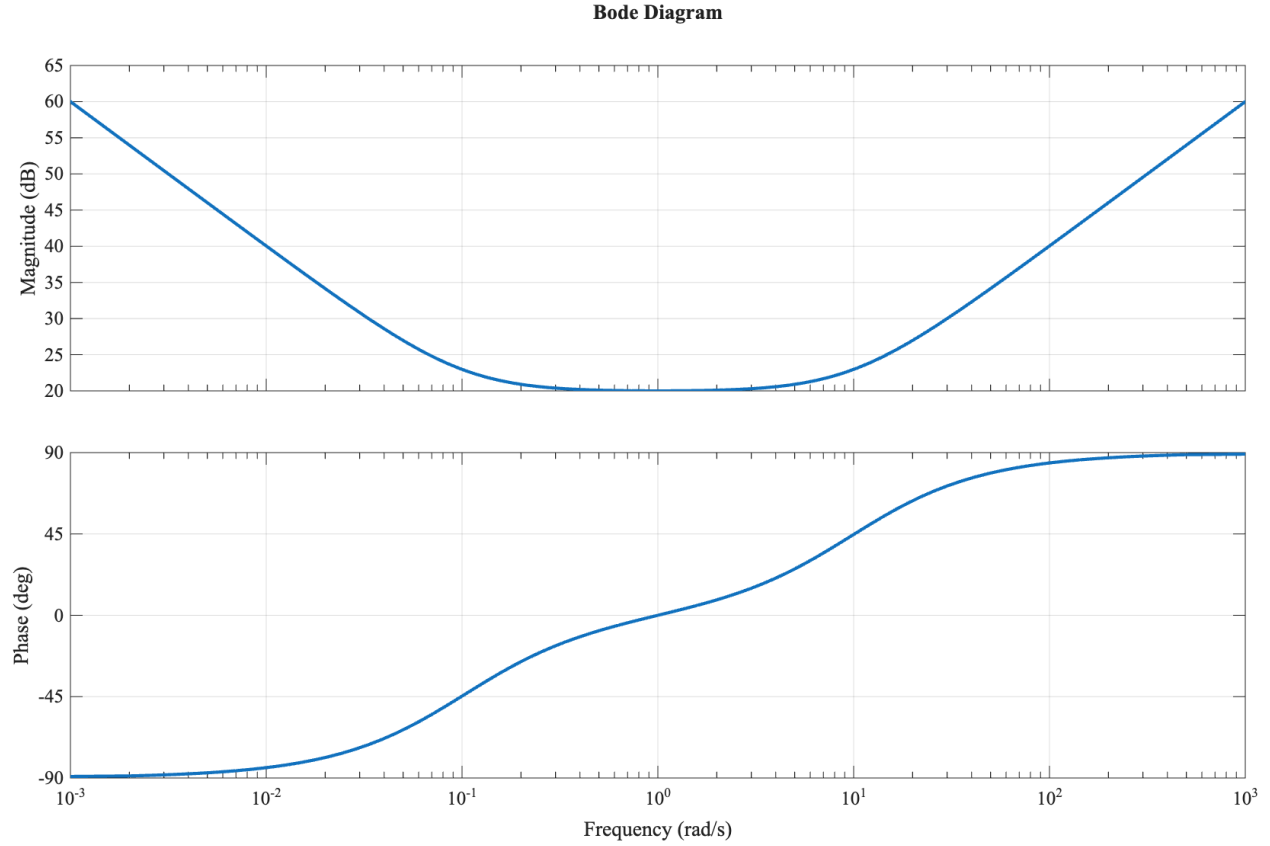


Figure 19: Bode plot of a PID control, $K_p = 10$, $K_i = K_d = 1$, $\omega_i = 0,1$ rad/s, $\omega_d = 10$ rad/s.

The tuning of a PID control is performed by accurately changing K_p , K_d and K_i constants.

SERVO VALVE BLOCK

The servo valve in this diagram is computed as a 2nd order dynamical system; this will be a similar solution to what adopted in Chapter 3, at least for the 2nd stage of the proportional SV.

This model represents the spool:

$$x_v = V_{ref} \cdot \frac{K_s \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2} = V_{ref} \cdot TF_{spool}$$

$$Q_L = x_v \cdot K_Q$$

The last proportional gain K_Q represents a zero-order electric equivalent fluid circuit analogous to that in Chapter 3.

Notice that the total mass flow rate entering the actuator is:

$$Q_L - k_{pq}\Delta P_L = Q_1$$

This is the difference between the loading flow rate and what lost through a bypass orifice.

$$Q_{bp} = C_d A_{bp} \sqrt{\frac{2(\Delta P_L)}{\rho}} \quad 1.3.6.1$$

ACTUATOR BLOCK

Inside the HLA block, 2 sections can be identified. The first is a compressibility law managing flow rates and pressure differentials. The second is a mechanical dynamic equilibrium that computes HLA final position based on pressures and external forces.

Let's focus on the hydraulic first section:

$$\sum Q = \frac{V(t)}{\beta} \frac{dp}{dt} + \frac{dV}{dt}$$

If we write this compressibility law [4] for both chambers of the HLA and we subtract them, we obtain:

$$Q_{in} - \dot{x}A_c \cdot \frac{2\beta}{V_0} = \frac{d\Delta P_L}{dt}$$

By integration, we get the pressure differential used to compute the bypass flow rate and feedback it into previous servo valve block.

The last part is a second order representation of the HLA piston.

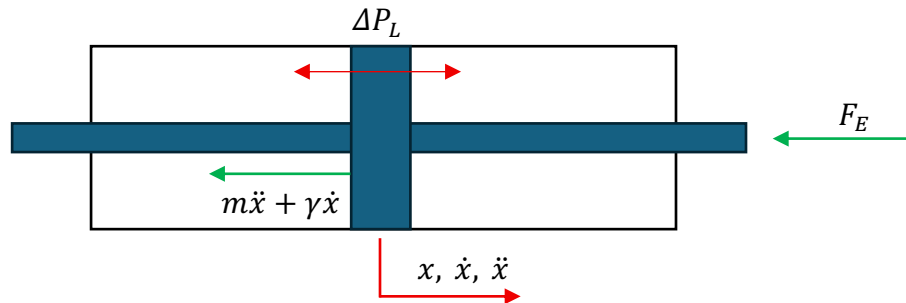


Figure 20: free body diagram of the HLA.

$$m\ddot{x} + \gamma\dot{x} = \Delta P_L - F_E$$

HLA output is the velocity signal $\dot{x}$, and through a final *free integral*, the HLA final position is computed. This physical quantity will be measured by a LVDT magnetic sensor, gained by K_{TX} , since the transmitter system is modelled as a zero-order, and fed all the way back into the main control block.

1.3.6 Force controlled hydraulic servo system

The force control is a variation of the position control, where external velocity is considered a disturbance and the system controls the force on a load cell. This model is used in the development of an iron bird, simulating the systems of the Alenia Aermacchi M-346 Master trainer jet, while also being adopted for the ASTIB test bench force-controlled HLA, simulating the landing gear wind disturbance.

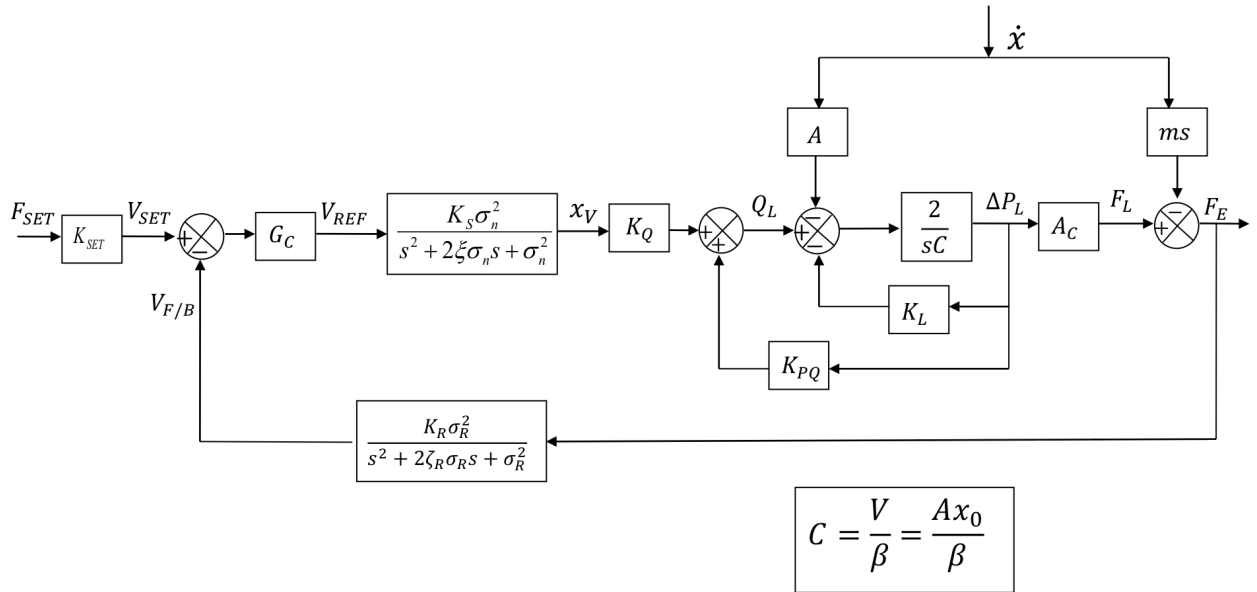


Figure 21: expanded block diagram of a force controlled servo system.

In this block diagram, the 2nd order transmitter model is immediately evident, representing the load cell, measuring the effective force fed back to the control section.

The first section is analogous to position controlled servo system, but now we have a set voltage representing a force and the final mechanical equation is a force balance. Speed measured by a transducer is a disturbance signal.

DYNAMICS EQUATIONS

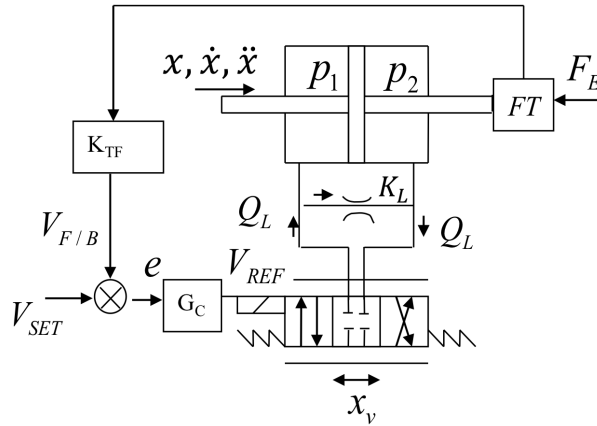


Figure 22: diagram of the servo valve - actuator control volume.

For the control block, it directly depends on adopted control strategy $G_C(s)$.

$$V_{set} = K_{set} F_{set}$$

$$V_{ref} = G_C(s)(V_{set} - V_{f/b})$$

The servo valve is now represented as a 2nd order, but in Chapter 3, the actual model is analyzed. The spool dynamic equation is the following:

$$\ddot{x}_v + 2\zeta\omega_n\dot{x}_v - \omega_n^2x_v = \omega_n^2K_S V_{ref}$$

An electric equivalent Wheatstone circuit with fluid resistances is used to compute flow rates:

$$Q_L = K_Q x_v + K_{PQ} \Delta P_L$$

The pressure differential ΔP_L across HLA chambers comes from the difference between 2 compressibility laws, each one written for a single chamber. This value is fed back to the Wheatstone circuit section.

$$Q_L - A_c \dot{x} = \frac{V}{2\beta} \frac{d}{dt} \Delta P_L + K_L \Delta P_L$$

$K_L \Delta P_L$ is the leaking bypass orifice flow rate, used to prevent frequent high frequency corrections by the servo valve.

Mechanical balance is written as follows:

$$F_L = A_c \Delta P_L$$

$$F_L - m\ddot{x} = F_{f/b}$$

A_C is the load section of the cylinder inside the HLA.

The feedback force is measured with a load cell (LC) modelled as a 2nd order system.

$$\ddot{V}_{f/b} + 2\zeta\omega_n\dot{V}_{f/b} + \omega_n^2 V_{f/b} = \omega_n^2 K_R F_{f/b}$$

This block's output is a feedback voltage signal $V_{f/b}$ entering the control block.

1.3.7 Electro-hydraulic servo actuator – examples

Examples of EHA in aerospace span from the actuation of primary and secondary control surfaces, to position controlled servo systems for landing gear.

In the Airbus A320 (Table 1), the motion about 3 axes is controlled via fly-by-wire, and the actuation stage is realized with an electrohydraulic servo actuator (EHSA).

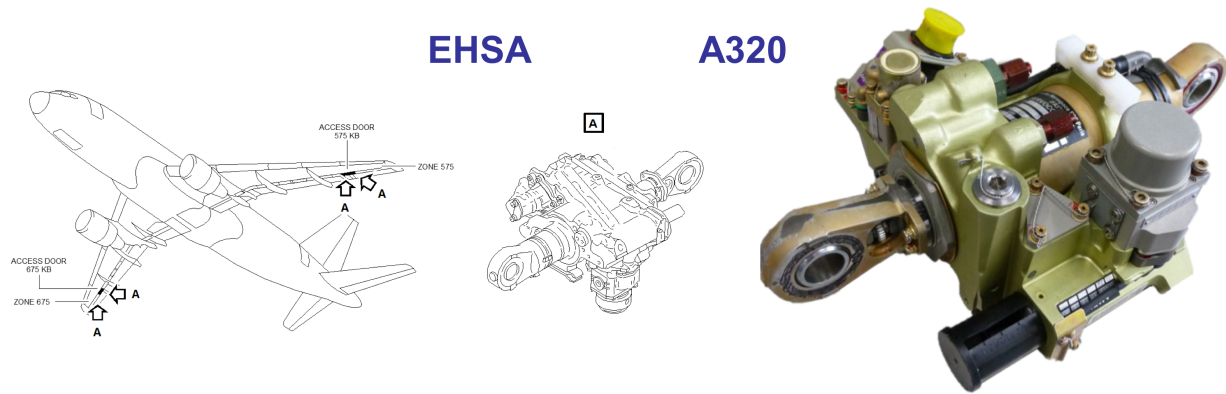


Figure 23: EHSA for Aileron and spoilers actuation on the Airbus A320.

This is a position-controlled servo system which uses a centralized hydraulic power rail as the main distribution network of the power fluid.

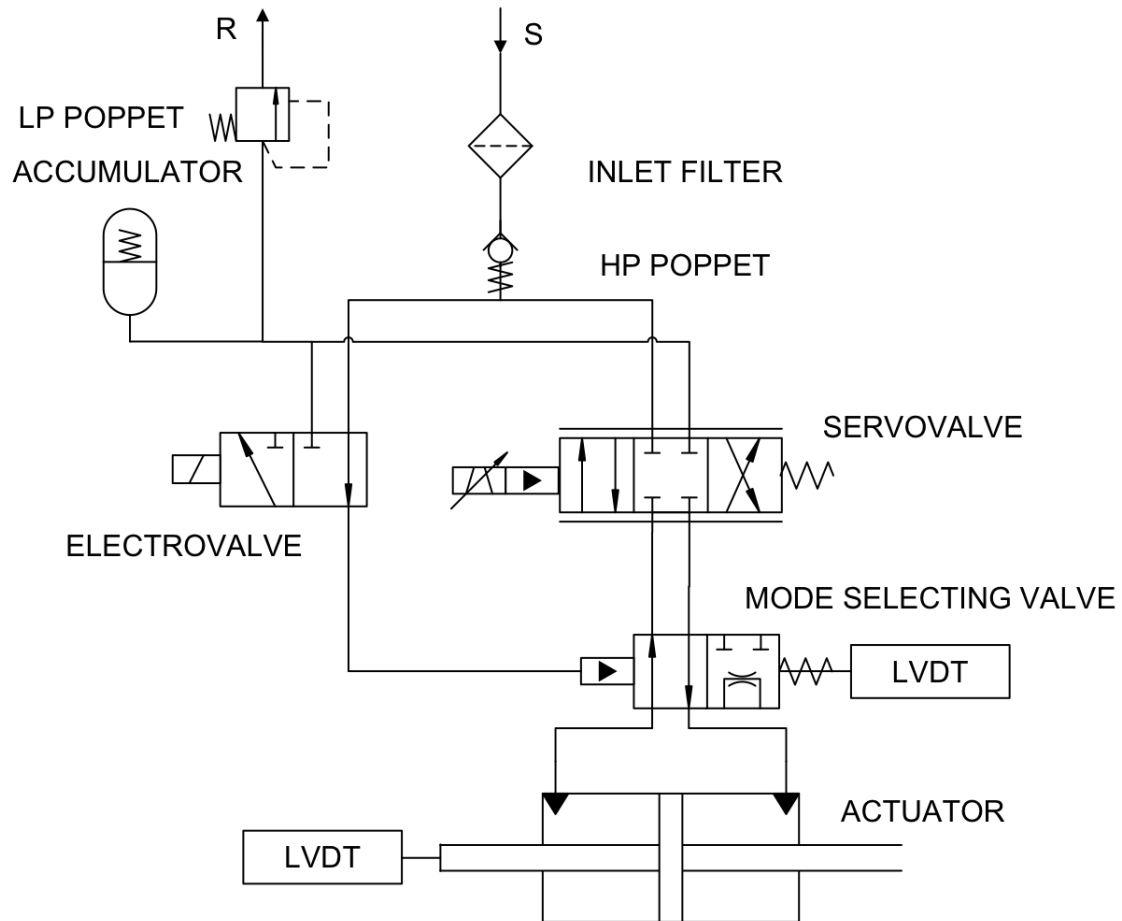


Figure 24: scheme of a position controlled EHSA on the Airbus A320.

The first electro valve on the upper left section of Fig. 16 governs the piloting stage of the mode selecting valve, this isn't a complete signal-by-wire configuration for every component of the Interface – Actuation stage of this servosystem. LVDT magnetic position sensors are used to control SV spool and EHSA piston positions.

2 Neural Network approach

2.1 Time series regression problem

Physics Informed Neural Networks (PINNs) are a specific category of neural networks that integrate physical knowledge along with the more standard data driven approach, using deep learning algorithms.

As mentioned before, physical knowledge is necessary to improve both generalization in the final trained model, and to grant convergence in scenarios when the architecture of the real-world system the network is reproducing isn't completely known in the first place.

When engineers are dealing with data streams coming from a transmitter subsystem, they usually manage time series. A series is a collection of points measured by mechanical or mechatronic system sensors, PLCs or other control electronics that can log a signal or measure a physical quantity. They may have a specific sampling frequency, so that every time they sample the quantity, a discrete series of measurements is produced.

$$Y = Y(t)$$

In the real world, a sample of the quantity at a given time t is not independent from all the other previous points in time. A standard feedforward Deep Neural Network (DNN) – take as an example a fully connected FCNN architecture, but this can be extended to others – can reproduce this quantity, using other generic ones as input $X(t)$ through a learning process that could be purely data driven or physical informed. The process of reproducing a curve expressed by the target quantity is called **regression problem**, a type of supervised learning when inputs and outputs are given, but the link between them is unknown. As we will later demonstrate, a time independent approach using backpropagation is insufficient to correctly represent temporal dependencies of a time series.

2.2 Feedforward Neural Network architecture – The Forward Step

Artificial Neural Networks fall into the family of machine learning models that replicate the behavior of biological neurons. ANNs are built of artificial neural units called *perceptron*, every one of them can be approximated using a mathematical function f_j that will be called *activation function* of the j -th unit. Its role is to produce an output h_j called *activation* typically between 0 and 1. This is the contribution of a single unit to the final output of the network.

A group of *perceptron* could be stacked to form a *layer*, units in the same layer are not usually interconnected between themselves, but they receive the output of units of a previous layer as an input and send their *activation* to the next. The size of a layer is directly proportional to the complexity of problems a network is used to solve.

The mathematical process at the foundation of an Artificial Neural Network is *activation function* composition, which consists of a composition of N nonlinear relations that maps an input vector x to the network output.

Network function:

$$f(x) = f^1(x) \circ f^2 \circ \dots \circ f^k \circ \dots \circ f^N$$

Here the input vector is only input of the first layer of neurons. Moreover, f^k represents an activation function if the k -th layer size is, in fact, 1. For every other case f^k should express a

vector of activation functions. In the end, for all layers except the 1st, f^k ($k > 1$), takes as an input the vector of *activations* $\mathbf{h}$ of a previous layer.

The term *hidden layer* is used in literature and scientific articles to define a layer with index $1 < k < N$, that is an internal layer of the neural network.

Every j -th *perceptron* in a generic k -th hidden layer is defined as follows:

$$f^k(h^{k-1}) = f^k(W^k h_j^k + b_j)$$

- k is the k -th layer index.
- W^k represent a weight matrix multiplying the activation vector. Components of this matrix are called parameters; they are used to produce the $net_{j,k}$ argument of the activation function.
- h_j^k is the activation vector.
- b_j is the bias vector, a deviation term added to net_j^k to better represent correlations with non-zero intercept.

This formulation of a single unit is typical of a Fully Connected (FCNN) architecture.

Examples of activation function are:

2.2.1 Sigmoid function

$$\sigma = \frac{1}{1 + e^{-x}}$$

Sigmoid function falls in the family of logistic functions, determining a S shaped growth two asymptotes. Usually adopted in networks solving classification problems. These problems are often called *logistic regressions*, even though they are not properly regressions, but a category of binary classifications with supervised learning processes.

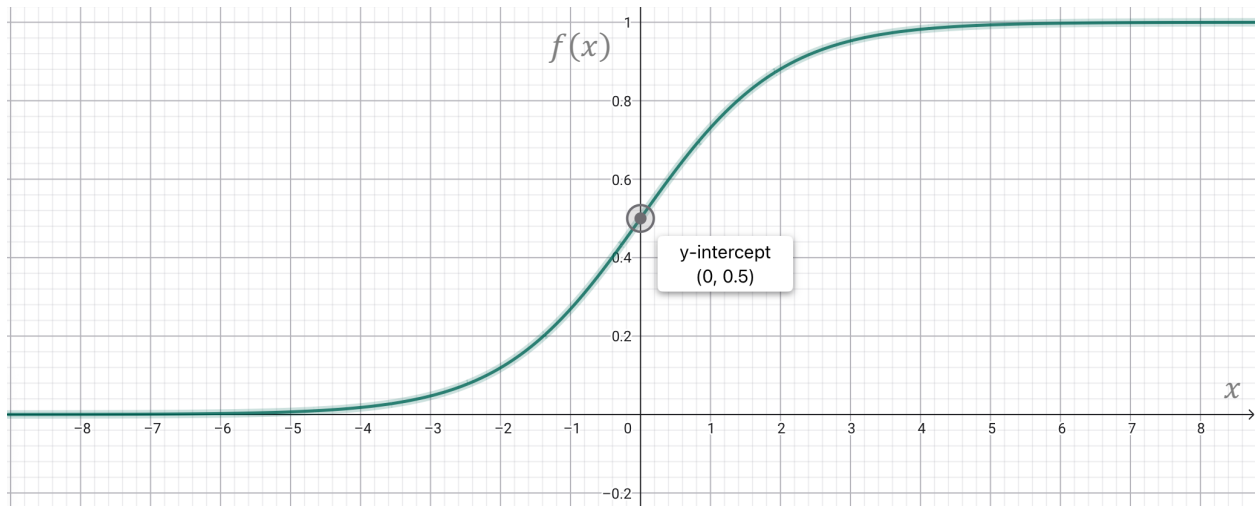


Figure 25: example of a sigmoid function.

2.2.2 SoftMax

$$\sigma : \mathbb{R}^k \rightarrow \left\{ z \in \mathbb{R}^k \mid z_i > 0, \sum_{i=1}^k z_i = 1 \right\}$$

$$\sigma(z) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, \text{ for } j = 1, \dots, K$$

SoftMax is again another logistic function that maps a vector of dimension k with nonzero entries, to a destination σ vector with the same characteristics as the first, but all the entries $\sigma_i \in [0,1]$ where the sum of all of them is always 1. This is used in supervised classification problems.

The plot of this function is to be considered like a sigmoid, but with softer asymptotes approach.

2.2.3 Hyperbolic Tangent

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Hyperbolic tangent is one of the most famous activation functions used in ANNs, specifically for Recurrent Neural Network (RNN) architectures in hidden layers, since it's faster near the origin, it helps mitigate *vanishing gradients*.

This function appears in hidden layers of the MATLAB default implementation of the LSTM neural network, which is a known evolution of RNNs [10].

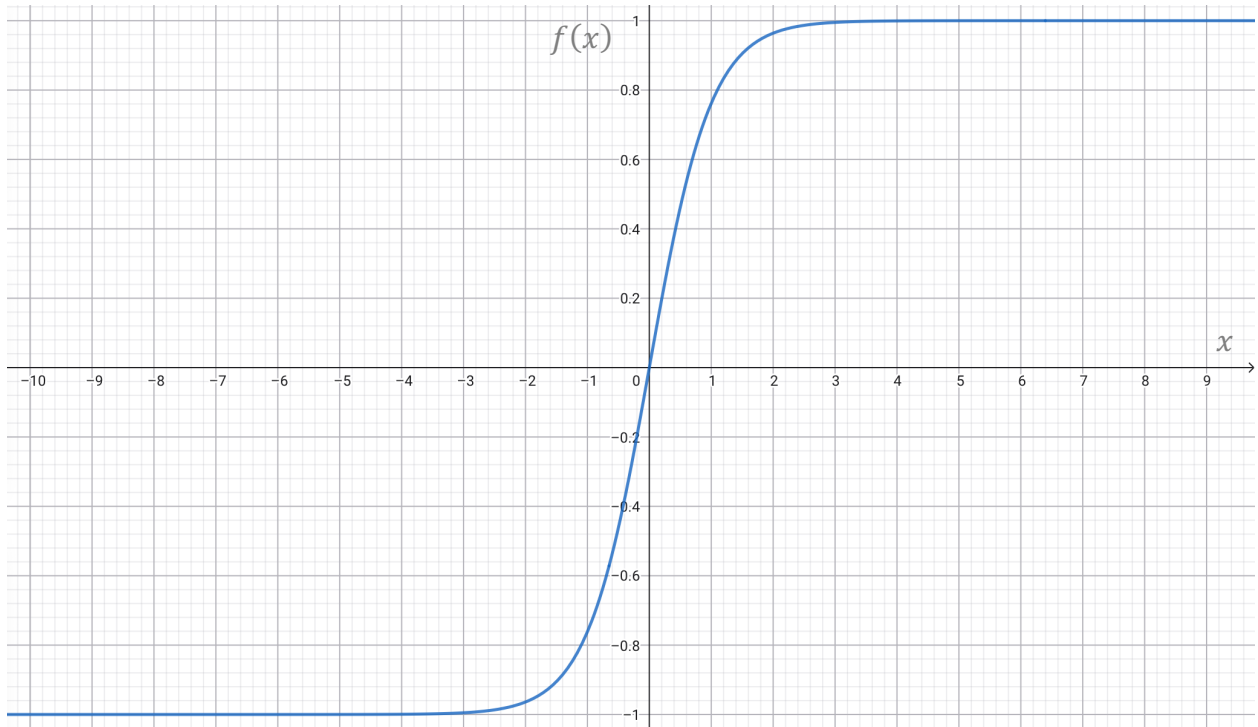


Figure 26: example plot of a hyperbolic tangent function $\tanh(x)$.

2.2.4 Rectified Linear Unit (ReLU)

$$\text{relu}(x) = x^+ = \max(0, x) = \frac{x + |x|}{2}$$

The rectified linear unit is an activation function that is zero for $x < 0$, and perfectly linear with derivative 1 for $x > 0$. It is used where an inhibition threshold is needed in regression problems, it's found on a ReLU layer in MATLAB implementation of FCNN architecture, and it's common in Convolutional Neural Networks (CNNs).

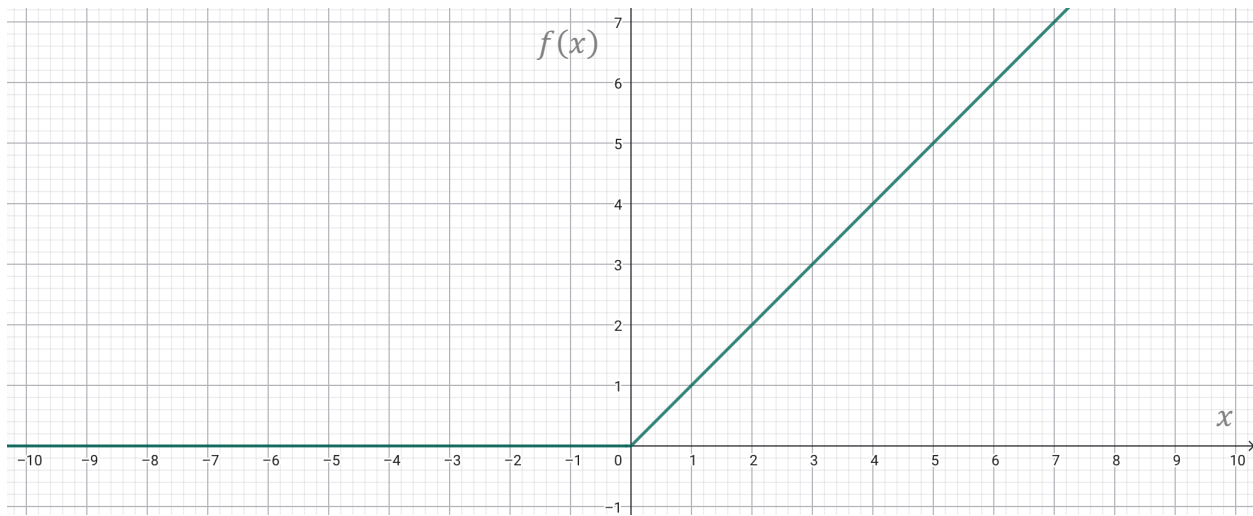


Figure 27: example plot of a ReLU function.

2.3 1st order backpropagation of a FCNN – time independent backward step

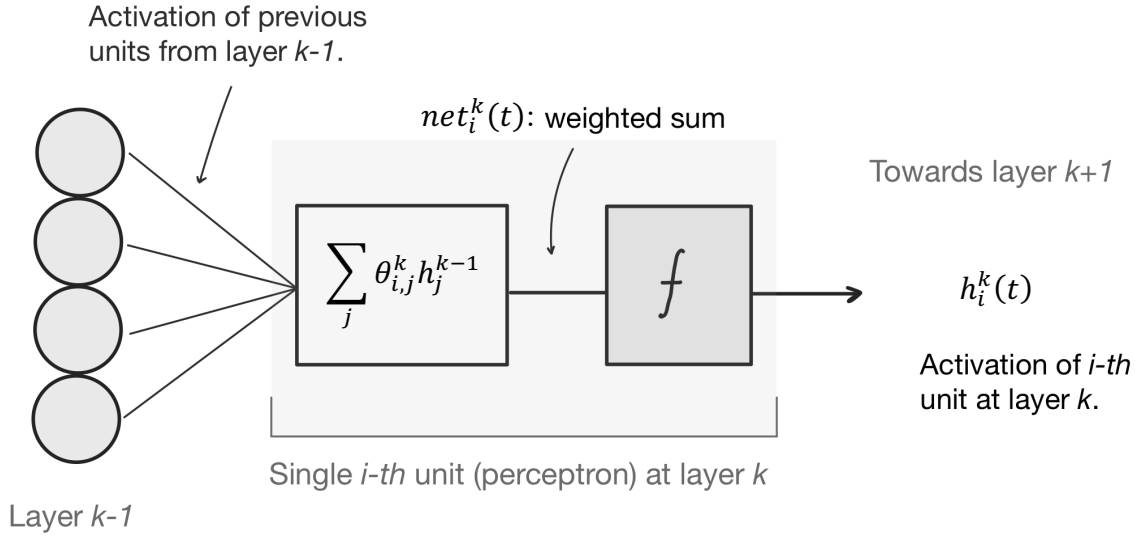


Figure 28: scheme of the forward step of a single unit in a feedforward FCNN. Citation to S. Hochreiter et. Al, 1997 [10].

In this diagram the working principle of a perceptron inside a feedforward fully connected neural network. f represents an activation function, used from the neural network to better represent non-linearities when performing regression, and to make sure the output of a unit (activation) remains constrained between 0 and 1.

To keep everything simple let's assume this is using a data driven approach with a mean squared error loss computed with the output of the NN obtained at output layer:

$$L_{mse} = \frac{1}{2N} \sum_t (y_t - T_t)^2 \quad 2.3.1$$

- y_t is the quantity reconstructed by the neural network, this is called output.
- T_t is the target real-world physical quantity, output of a sensor.
- t is the generic time index of a time series. L_{mse} is computed at the end of every epoch.

In traditional feedforward model architecture, the network parameters are usually updated with a process called **gradient descent**, a component of a broader mathematical framework, **backpropagation**.

Let's assume a *generic output layer*, the weights and biases of every perceptron of that layer are updated with the gradient descent rule:

$$\theta^{e+1} = \theta^e - \alpha \frac{\delta L_{mse}(\theta^e, X)}{\delta \theta} \quad 2.3.2$$

- θ^e is a general notation for network parameters at epoch e .
- α is the *learning rate*, a hyperparameter set in advance to optimize the learning process.

For simplicity, the notation of the output layer indexes has dropped.

The learning rate multiplies the error gradient, that for an *output layer* can be written as follows:

$$\frac{\delta E(X, \theta)}{\delta \theta_{ij}^k} = \frac{1}{N} \sum_{d=1}^N \frac{\delta}{\delta \theta_{ij}^k} \frac{(y_d - T_d)^2}{2} = \frac{1}{N} \sum_{d=1}^N \frac{\delta E_d}{\delta \theta_{ij}^k}$$

Obtained by naming E_d only the squared difference part of the error:

$$E_d = \frac{(y_d - T_d)^2}{2}$$

In this case we can apply the chain rule to the error derivative and write:

$$\frac{\delta E_d}{\delta \theta_{ij}^k} = \frac{\delta E_d}{\delta h_j^k} \frac{\delta h_j^k}{\delta \theta_{ij}^k} = \frac{\delta E_d}{\delta h_j^k} \frac{\delta h_j^k}{\delta net_j^k} \frac{\delta net_j^k}{\delta \theta_{ij}^k}$$

This can be obtained by simply following the diagram in Fig. 3.

We can rename terms and perform calculations to obtain:

$$\frac{\delta E_d}{\delta h_j^k} = \delta_j^k$$

$$\frac{\delta h_j^k}{\delta net_j^k} = \frac{\delta f(net_j^k)}{\delta net_j^k} = f'(net_j^k)$$

$$\frac{net_j^k}{\delta \theta_{ij}^k} = \frac{\delta}{\delta \theta_{ij}^k} \left(\sum_{d=1}^D \theta_{dj}^k \cdot h_d^{k-1} + b_j^k \right) = h_i^{k-1}$$

So, the error gradient for an *output unit* is:

$$\frac{\delta E_d}{\delta \theta_{ij}^k} = \delta_j^k f'(net_j^k) h_i^{k-1} \quad 2.3.3$$

If now we consider a *hidden layer*, the error must travel all the connections to a layer $k+1$ to the layer k , then the error will be backpropagated towards a layer $k-1$. The formulation is like the output layer, but the error term δ_j^k depends on the weighted sums of links to the units at layer $k+1$.

$$\delta_j^k = \frac{\delta E_d}{\delta h_j^k} = \sum_l^L \frac{\delta E}{\delta h_l^{k+1}} \frac{\delta h_l^{k+1}}{\delta h_j^k}$$

$$\frac{\delta E}{\delta h_l^{k+1}} = \delta_l^{k+1}$$

$$\frac{\delta h_l^{k+1}}{\delta h_j^k} = \frac{\delta h_l^{k+1}}{\delta net_j^k} \frac{\delta net_j^k}{\delta h_j^k} = g'(net_j^k) \frac{\delta}{\delta h_j^k} (w_{jl}^{k+1} h_j^k + b_l) = g'(net_j^k) w_{jl}^{k+1}$$

Putting all together we obtain:

$$\frac{\delta E}{\delta \theta_{ij}^k} = g'(net_j^k) h_i^{k-1} \sum_{l=1}^L (\delta_l^{k+1} w_{jl}^{k+1}) \quad 2.3.4$$

This is the backpropagation formula for a *hidden layer*. Notice that the error is propagated backwards through the hidden layers, and it will eventually reach the input layer. Now g is the activation function of a unit l , at layer $k+1$.

In this simple formulation of a first order backpropagation process, it is evident there isn't any temporal link between the data at time t with any other data point at a previous or future time. The network treats every point as independent.

This is not optimal when dealing with time series because at every time step, the output of the network should at least directly depend on what happened even several timesteps before.

The solution to this problem is to adopt Recurrent Neural Networks (RNNs) by briefly introducing the math behind the Back Propagation Through Time (BPTT).

2.4 Recurrent Neural Network and BPTT

Recursive Neural Networks (RNNs) excel in reproduction of continuous data, such as text, audio, and video, where learning a sequential input is key. The main characteristic of RNN architecture is recurrent connections, which allow the RNN to update its current *state* using historical and current data, to influence future outputs [10].

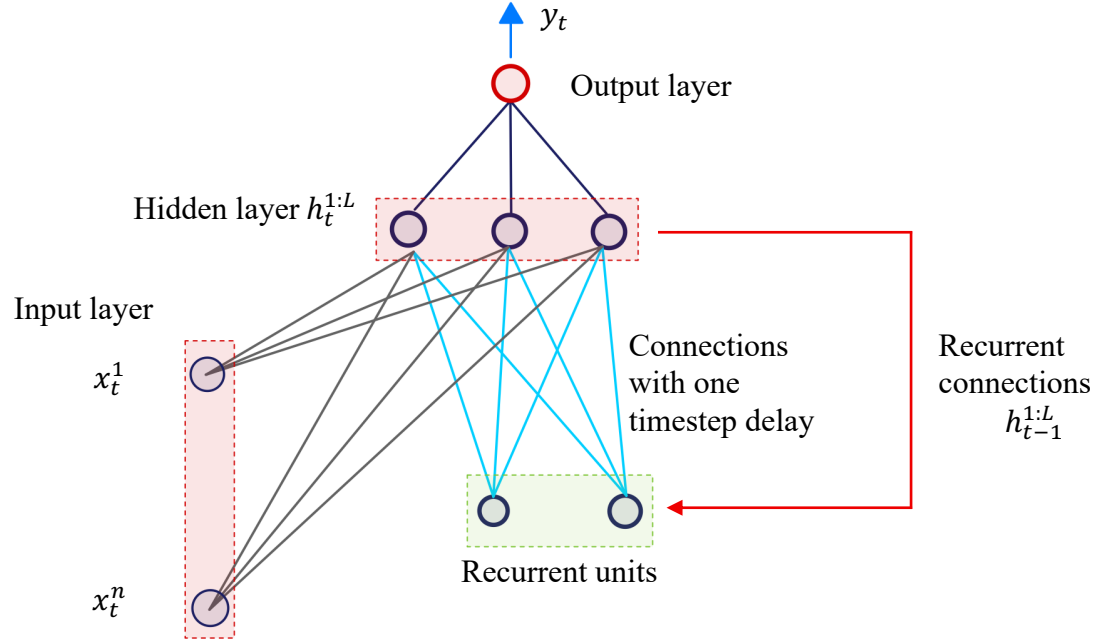


Figure 29: Scheme of an Elman type recurrent neural network. – Adapted from Coulibaly et. Al, 2001 [3].

This scheme represents the Elman type Recurrent Neural Network, first introduced by Elman J. L. in 1990 [7]. It is close to what is discussed in Hochreiter S. theory [10], when the notorious vanishing gradients problem is analyzed.

The mechanism behind this NN consists of introducing a layer of *context units* that store information about past output and have recurrent connections to other standard units of the same layer. In this case, *context units* are fully connected to share information with all a given hidden layer.

These special units haven't any weights and biases, they only act as a time delay data storage. The weighted averaging is performed in the hidden units:

$$h_t = f(W_x X_t + W_h h_{t-1} + b) \quad 2.4.1$$

- W_x is the fresh input data weight matrix.
- W_h is the recurrent weight matrix, the one used for the recurrent *context* connections.
- h_{t-1} is the previous hidden state, it now concurs to compute the next one: h_t .
- b is the bias.
- t is the time index, now time has a meaning and it's fundamental in the learning process.

and the learning process is completed using Back Propagation Through Time (BPTT).

In a standard RNN, an input sequence of length T produces a sequence of hidden states $(h_1, h_2, \dots, h_T)$. Since the RNN generates a prediction at every timestep, there will be a sequence of errors $(L_1, L_2, \dots, L_T)$. The final error at every epoch will be:

$$L_{mse} = \sum_{t=0}^T L_t \quad 2.4.2$$

This is always computed as the mean squared error, as before mentioned for Feedforward NNs. The gradient descent algorithm updates the weights as before, but for now we will only focus on the recursive weights.

$$W_h \leftarrow W_h - \alpha \frac{\delta L}{\delta W_h} \quad 2.4.3$$

Since the hidden state update is now recursive, a generic hidden state h_t will now depend on the previous h_{t-1} , which as well has a direct connection to h_{t-2} , and so on.

$$h_{t-1} = f(W_x X_{t-1} + W_h h_{t-2} + b)$$

To find the correct expression for the error gradient, we must consider using the multivariate chain rule:

$$\frac{df}{dz} = \frac{df}{dx} \frac{dx}{dz} + \frac{df}{dy} \frac{dy}{dz}$$

Since the matrix W_h has been applied at timestep $t, t - 1, t - 2$, etc. A change in this matrix will alter the loss at every timestep.

$$\frac{\delta L_t}{\delta W_h} = \sum_k^t \frac{\delta L_t}{\delta h_t} \frac{\delta h_t}{\delta h_k} \frac{\delta^+ h_k}{\delta W_h} \quad 2.4.4$$

- $\frac{\delta L_t}{\delta h_t}$: is the final error term.
- $\frac{\delta h_t}{\delta h_k}$: this expresses how much the historical state h_k affects the current at time t .
- $\frac{\delta^+ h_k}{\delta W_h}$: how much the recurrent matrix affects the state at timestep k (ignoring past dependences, this is an explicit derivative).

The state middle term is not an explicit derivative, but every state at a given time depends on the previous, so by applying chain rule again we can write:

$$\frac{\delta h_t}{\delta h_k} = \prod_{j=k+1}^t \frac{\delta h_j}{\delta h_{j-1}} \quad 2.4.5$$

The RNN, at every timestep, must retain information of every previous timestep since the beginning of time. A common issue is *Exponential Dilution*, when dealing with long time series, the contribution of state h_1 at timestep $t = 0 + dt$ has been multiplied by both the recurrent matrix W_h and the activation function so many times, it is now negligible.

Although, the main problem of BPTT strategy is, as mentioned before, *vanishing gradients*. If the influence of historical states on the current one is low, the product (13) will approach zero and nothing will be learned in a reasonable amount of time, therefore the error gradient will vanish.

S. Hochreiter in his research, solved this problem by introducing a different architecture that has the peculiarity to maintain error backflow from decaying across hundreds of timesteps. Refer to S. Hochreiter et. Al, 1997 for a complete demonstration [10].

2.5 Long-Short Term Memory architecture

LSTM Neural Networks have always been proven to be a solid choice when analyzing long time series. Their architecture greatly differs from a traditional RNN, while still maintaining recurrency, but with the ability to operate regression of longer time series with higher accuracy.

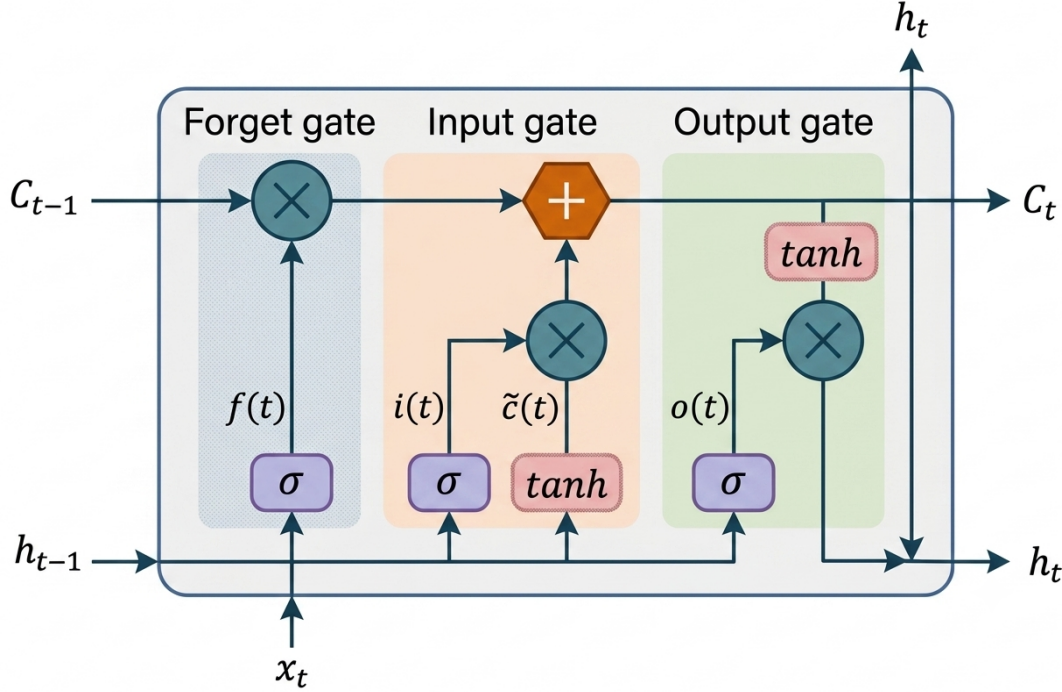


Figure 30: Diagram of a single LSTM unit.

This architecture uses *gates* to choose to retain or forget previous information from past timesteps, or to output a hidden state h_t using the past filtered context. In addition to recurrent connections if a more traditional RNN, which retains a memory for the past activation h_{t-1} , a LSTM uses *cell states* C_t and C_{t-1} .

An LSTM unit is always positioned in a layer with $1 \leq k < N$, hence it will always output a hidden state h_t and a *cell state* C_t at every time step t . Cell state is represented above as the flow line on the top portion of the diagram. It acts as a “conveyor belt” with minor interactions.

Here, the (X) box indicates a point wise multiplication, and the (+) is the sum.

Hidden and Cell states are then processed through a series of consecutive *gates*, these selectively add or remove information to the cell states.

Normally LSTM gates consist of sigmoid activation function and a point wise multiplication, the sigmoid ensures that data is *completely let through* ($\sigma = 1$), *completely discarded* ($\sigma = 0$), or *partially let through* ($0 < \sigma < 1$).

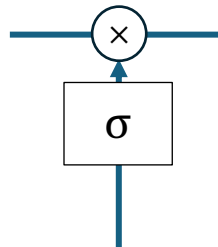


Figure 31: Diagram of a LSTM gate.

There are three of them in a standard LSTM unit:

- **Forget gate** decides which fraction of a previous cell state C_t and fresh input x_t is to be retained in the next unit's processing.
- **Input gate** determines the new information to process.
- **Output gate** produces the current hidden state h_t and cell state C_t .

1st step – forget layer

The first step of a LSTM unit process is filtering out a fraction of the cell state content. A first *forget gate* outputs $0 \leq f_t \leq 1$, a multiplicative factor for the past C_{t-1} cell state. If $f_t = 0$, the past context is completely discarded and overwritten with a new one.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Every layer may have its own bias and weight matrix.

2nd step – input layer

The next passage is deciding which will be the new content to update in the cell state C_t . This is done in two steps, the first computes the input i_t through a sigmoid and the second outputs the new cell state candidate $\tilde{C}_t$.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] - b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Then, the cell state is updated with a sum operation, considering also the previous *forget gate* contribution.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

Again, $\times$ is a pointwise multiplication.

3rd step – output layer

Now that the cell state has been updated, the LSTM unit will output a current hidden state h_t based on current input data and previous hidden state combined $[h_{t-1}, x_t]$.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \times \tanh(C_t)$$

NOTE: This is a traditional base implementation of the LSTM unit, other specific variations to this architecture may be implemented in dedicated software, like MATLAB.

The LSTM architecture has been the final choice to model physical systems in this thesis work, due to its superior performance in the reconstruction of long time series. The main advantages of this architecture are the following.

- **Effective Management of Long-Term Dependencies:** unlike standard architecture, Long Short-Term Memory (LSTM) networks are built to track historical dependencies across extended temporal sequences. By decoupling the internal cell state C_t from the hidden state h_t , the network establishes an isolated memory track. The flow of information into and out of this state is regulated via specialized gating mechanisms (forget, input, and output gates).
- **Robustness Against Vanishing Gradients:** during Backpropagation Through Time (BPTT), traditional Recurrent Neural Networks (RNNs) often suffer from vanishing gradients, severely limiting their capacity to learn long-range temporal connections. LSTMs inherently resolve this mathematical bottleneck. The additive formulation of the memory C_t acts as a constant error carousel, ensuring that gradients propagate stably backward through hundreds of time steps. This mechanism guarantees the successful learning of long-term patterns without exponential decay, as originally demonstrated by Hochreiter and Schmidhuber [10].
- **Native Adaptability to Dynamic Sequential Data:** LSTMs are intrinsically designed to process variable-length sequential data, making them a highly effective framework for time-series forecasting and dynamic signal processing. For the development of a digital twin representing a hydraulic system, the network can natively ingest and map complex continuous inputs—such as pressure differentials, flow rates, and potentially noisy or contradictory sensor readings like piston velocity and position. This allows the model to accurately capture highly non-linear dynamic variations, leveraging both the immediate sensor data and the historical behavior of the system.

3 Physics Informed Neural Networks (PINNs)

3.1 The flaw of a purely data driven approach

One of the key aspects of constructing a Digital Twin for mechanical system modeling is the level of knowledge of the system itself. As mentioned in the first chapters, if our information about a certain system is approximative and incomplete, traditional modeling methods such as FEM, CFD or 1D simulation is impossible to use.

While LSTM networks are an optimal tool to approach this time dependent problem, they lack a very important aspect, they have zero knowledge of the real physics that governs a specific mechanical system.

As we later show, if we train a LSTM to directly solve a regression problem to reconstruct several sensor data streams of pressure, position, velocity and force set or feedback, the neural network has no clue of any physical link between those quantities, and it will act as a simple *curve fitting tool*.

To demonstrate this, we will first describe the mechanical system we are going to analyze, then we will swap a single sensor of one experiment with another measurement from a different experiment. If our LSTM is purely data driven (with just a L_{mse} loss using the mean squared error), it will perfectly reconstruct every sensor channel, even the purposely wrong one. This will completely invalidate its purpose as a digital twin modeling method for PHM, since it would be insufficient to spot, for example, a certain faulty component that deviates from its correct behavior.

3.2 Physical description of the real system

Overview

The physical system the final Digital Twin will reproduce is a single component of the ASTIB test bench, available in the laboratories inside Aerospace and Mechanics Dimeas department at Polytechnic of Turin. This is a test bench designed for the ASTIB project, founded by the Clean Sky 2 Joint Undertaking, under the European Union H 2020 research program. Clean Sky 2 is finalized to introduce the next generation of lower emission aircraft by 2050. One of this research is aimed at developing fully electric servo actuators for aircraft landing gear and control surfaces, in contrast to more traditional hydraulic ones. The ASTIB test bench is in fact purposed to test an electric servo actuator system (EMA), which is position controlled and disturbed using a force controlled hydraulic servo actuator (HLA) [4].

The following pictures will better represent the test bench.

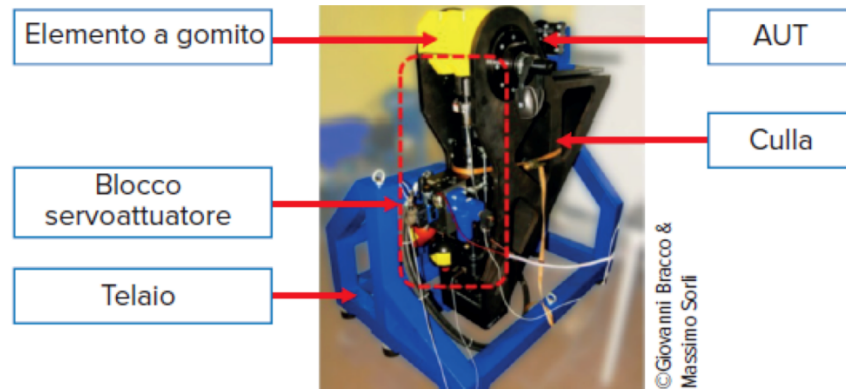


Figure 32: overview of the ASTIB testbench.

The coupling revolute joint is visible, here HLA and EMA relate to a mechanical transmission ratio of approximately 0,45.

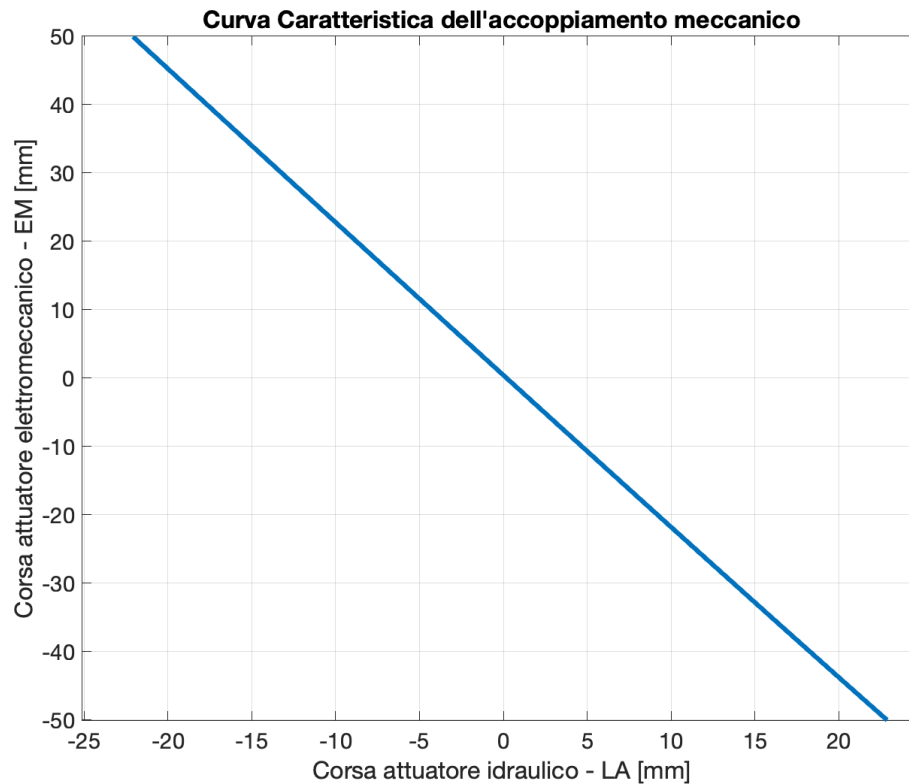


Figure 33: Relation between HLA and EMA strokes.

To safely operate this equipment, the HLA is connected to a hydraulic framework featuring an emergency Shut Off Valve (SOV), accumulators and other hardware, as shown in Fig. 12, and in the configuration scheme (Fig. 14).



Figure 34: safety hydraulic equipment.

In the following Fig. 13 is visible the hydraulic supply group controls power regulation, supply pressure adjustment and gradual startup. A filtering system is also present to prevent introducing dirty oil inside the equipment.

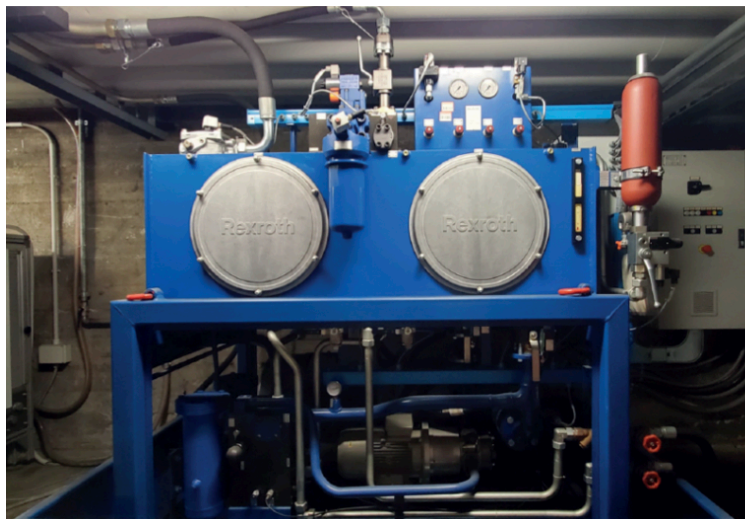


Figure 35: supply and power regulation group.

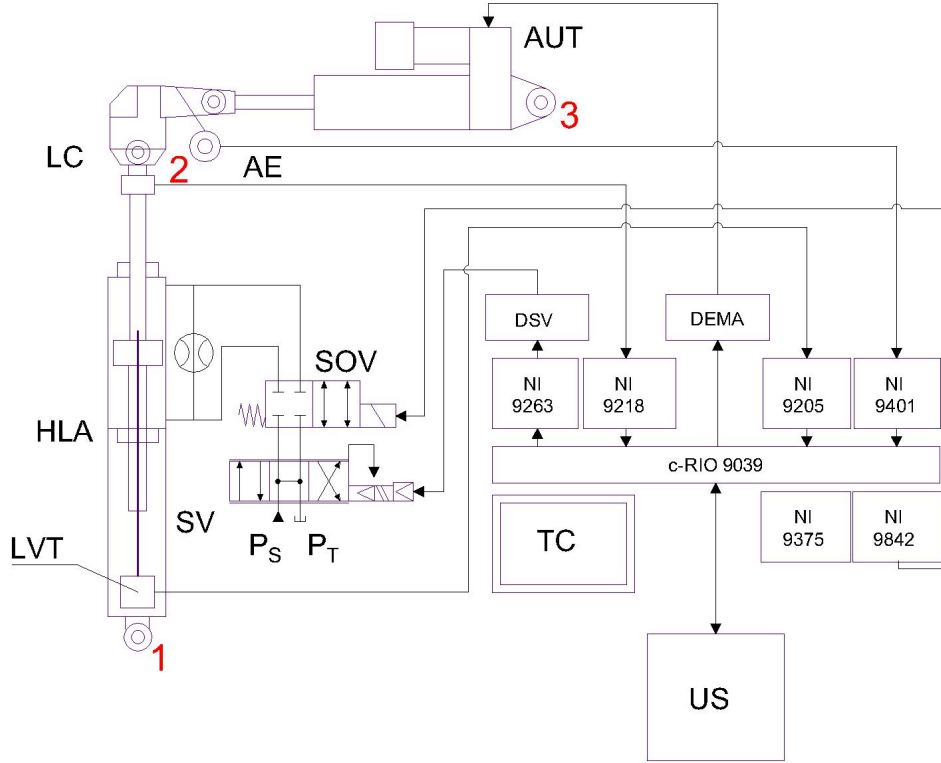


Figure 36: configuration scheme of the ASTIB test bench.

In this scheme, adapted from [4], a complete configuration of the ASTIB testbench is visible.

The HLA is powered by a high bandwidth proportional servo valve (SV) which is normally opened, but since it is mounted before a shut off safety valve, the HLA can be disconnected.

Inside the HLA, fixed to the cylinder, the LVT magnetic speed sensor is mounted. This is a high bandwidth sensor capable of measuring the relative velocity of the piston with respect to the cylinder, with a maximum value of 100 mm/s .

A load cell is located between the piston rod end, and the revolute joint. This is used to obtain force feedback; it has a maximum load capacity of 50 kN and an accuracy of $0,1\%$ full scale.

Strain gauge sensors are used to measure HLA chamber pressures and supply pressure p_S . In static condition, all the experiments will be conducted with $p_S = 140 \text{ bar}$.

An angular encoder is mounted fixed to the revolute joint, measuring both the position of HLA and EMA, the Electromechanical Actuator, which is position controlled.

The HLA is composed of a low friction cylinder with a maximum stroke $h_{HLA} = 100 \text{ mm}$ and a thrust area $A_{HLA} = 13,5 \cdot 10^{-6} \text{ m}^2$.

HLA dynamic model

In this configuration schematics, the architecture of ASTIB test bench. This thesis will only focus on modeling the servo valve SV, but for academical purposes, in Fig. 15 the full force control block diagram will be shown.

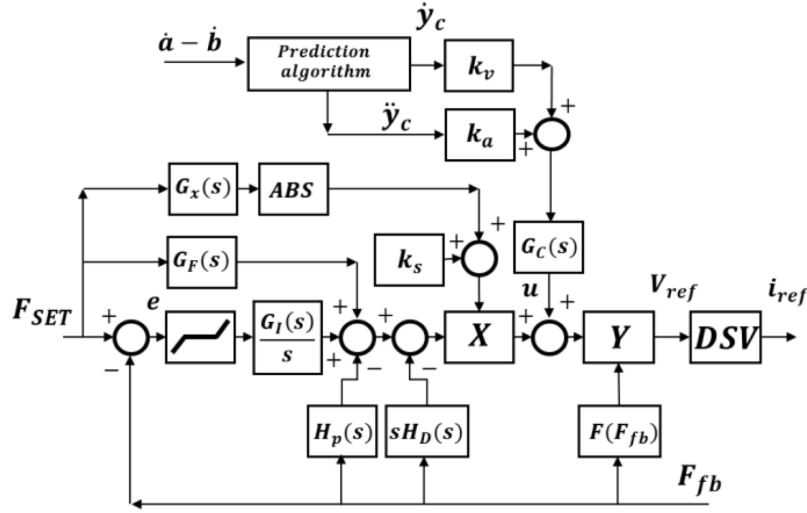


Figure 37: force control loop of the HLA.

In this diagram F_{SET} and F_{fb} represent the force set imposed by the user (US block Fig. 14) and feedback measure by the load cell (LC block in Fig. 14).

$\dot{a} - \dot{b}$ is a position signal better explained in the next diagram, i_{ref} is the error compensation current signal activating the SV solenoid. DSV is a conversion block which strictly belongs to the SV control and interface, in our experiments the signal picked up by sensors will be V_{ref} . This is a standard procedure for most servosystems.

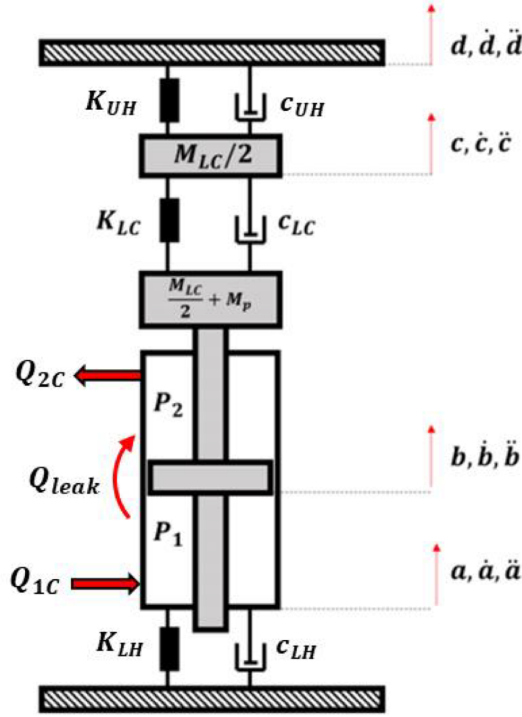


Figure 38: free body diagram of the hydraulic linear actuator (HLA).

This free body diagram shows the core dynamics of HLA, EMA, and fixed frame. In this case we suppose the lower frame to be fixed relative to our reference system, and we pose a , b and c as the coordinates of any relative motion:

- a = relative motion of HLA cylinder.
- b = rel. piston displacement.
- c = EMA stroke.
- d = frame displacement relative to itself caused by a deformation.

In this case, $\dot{a} - \dot{b}$ is the velocity of the piston relative to the cylinder.

Since we will only focus on the SV, a comprehensive description of the force loop control, or governing equation about the HLA dynamics won't be introduced.

The pressures inside each actuator chamber can be computed using flow continuity law:

$$\sum Q = \frac{\delta V(t)}{\delta t} + \frac{V(t)}{\beta_{eq}} \frac{\delta p(t)}{\delta t} \quad 3.2.1$$

- β_{eq} acts somehow like a spring stiffness, its subscript *equivalent* means that tubes rigidity, water contamination (% of water volume) and other factors have been considered. This will make the oil appear as *more compressible* than it is if a compressibility test would've been carried out in controlled environment. Thus, $\beta_{eq} < \beta_{oil}$.



45

$$\frac{Q_A - (\dot{a} - \dot{b})A - Q_{bp} - Q_{leak}}{V_A + (a - b)A} \cdot \beta_{eq} = \frac{dP_A}{dt} \quad 3.2.2$$

$$\frac{-Q_B + (\dot{a} - \dot{b})A + Q_{bp} + Q_{leak}}{V_B - (a - b)A} \cdot \beta_{eq} = \frac{dP_B}{dt} \quad 3.2.3$$

Q_{leak} and Q_{bp} go from chamber A to B because in this configuration, $p_A > p_B$. Q_{leak} represents the oil leaking through clearance between piston and cylinder inner surface, caused by imperfect sealing. Q_{bp} through SV bypass orifice will be better explained when analyzing the SV.

As shown on Fig. 14, the LVT sensor is placed inside the cylinder and fixed to it, measuring the piston velocity. This is a high bandwidth magnetic sensor that can be modeled as a zero-order system [4].

$$V_{LVT} = k_{LVT}(\dot{a} - \dot{b}) \quad 3.2.4$$

Servo valve

The component to reproduce with a Digital Twin is the 4 – 3, 5 ports, hydraulic servo valve (SV), specifically, a Bosh Rexroth 4WSE2EM-6-2X-20-8-ET-315-K17-EV. The naming 4 – 3 refer to its configuration, it has 4 ways (P1, P2, T and R) and 3 positions. From Fig. 17 we can see its scheme, with hydraulic piloting to an amplifier and mechanical internal feedback.

This is a proportional servo valve, meaning that its output can be controlled by proper input signal modulation.

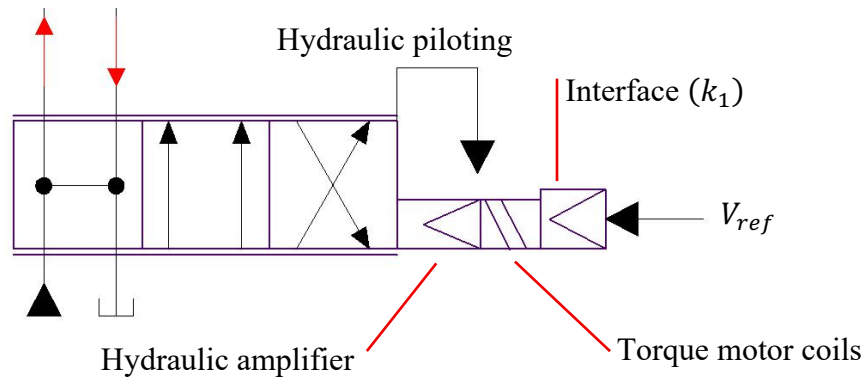


Figure 40: zoom on the SV.

In the next scheme we will explain the core mechanism of SV output modulation and its proportionality with V_{ref} .



Hydraulic amplifier consists of a double hydraulic piloting stage starting from supply ports, thus transmitting P_s . If the *flapper* obstructs one of its two nozzles, blocking oil flow from piloting orifice to R discharge port, the pressure will build up in the piloting conduit. This pressure will push against the spool end area A_s resulting in the spool itself changing position.

47

The final proportional output x_v is the result of the static equilibrium of torque generated by the *torque motor* and the resisting torque from *feedback spring*. This is the result of an electric command opposed to mechanical feedback. If V_{ref} is released, the spool returns to the center and ports C1 and C2 are entirely covered by spool lobes again. This SV is normally opened but coupling it with the shut off valve (SOV) we obtain a normally closed configuration.

In this scheme, the internal geometry of the SV is detailed. Flow through each port is computed using an electric equivalent Wheatstone circuit [4].

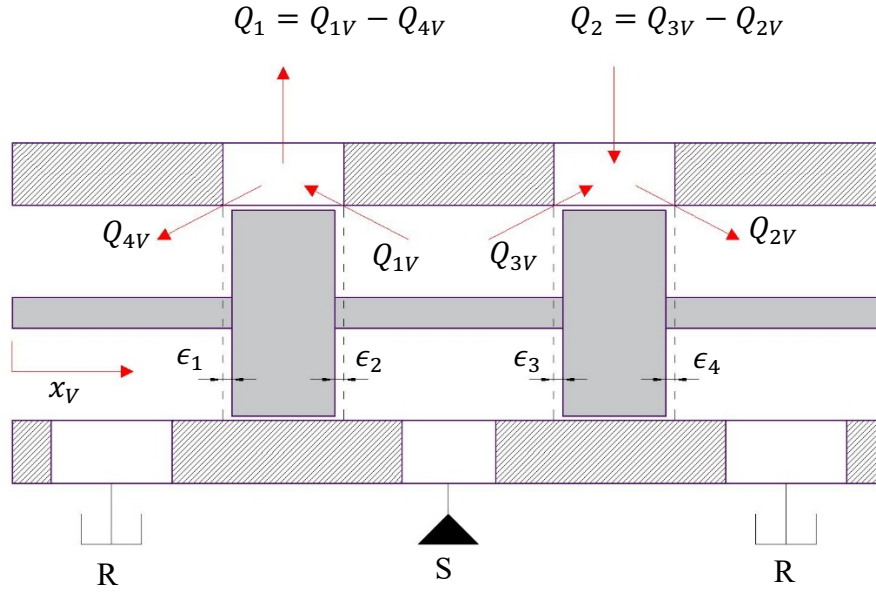


Figure 42: SV functional scheme.

Using a general approach, we can distinguish 2 contributions to the final oil flow through a port:

- (R_a) **turbulent flow**, corresponding to the oil flow with high Reynolds number.
- (R_c) **laminar flow**, with low Reynolds number, through a small aperture.

Fluid resistance can be written as:

$$R_c = \frac{12\mu_{oil} (x_v - \epsilon_1)}{2.5 w_s h_{rad}^3} \quad 3.2.5$$

$$R_a = \frac{\rho_{oil}}{2 C_d^2 A_1^2} \quad 3.2.6$$

The small aperture causing the laminar flow would be an underlap, but in this SV, there is none, hence we will only consider a turbulent term R_a .

Let's now consider the case when port C1 is connected to supply port S. The electric equivalent circuit will be as follows:

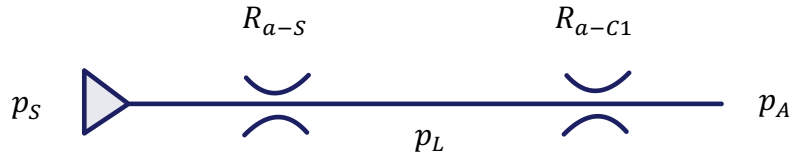


Figure 43: electric equivalent circuit of S – C1 path.

In this circuit p_L represents an internal pressure building up in the valve chamber. It won't match the supply pressure because we are introducing a resistance in the supply port. Furthermore, the pressure at the end of this circuit would be p_A , since port C1 directly communicates with chamber A of the HLA.

$$p_S - p_L = R_{a-s} Q_1^2$$

$$p_L - p_A = R_{a-c1} Q_1^2$$

If the port S is sufficiently large compared to C1, we can assume $R_{a-s} \approx 0$, solving for Q_1 :

$$p_L = p_S$$

$$Q_A^2 = \frac{p_S - p_A}{2R_A} \rightarrow Q_A = \sqrt{\frac{p_S - p_A}{2R_A}}$$

Sign convention is the same as Fig. 17. Analogous considerations can be applied to other ports.

We can also compute the flow through a bypass orifice using the same theory, rewritten to use the bypass flow coefficient [4]:

$$Q_{bp} = Q_{bp} A_o \frac{2(p_A - p_B)}{\rho_{oil}} \quad 3.2.7$$

This flow rate is considered positive while flowing from p_A to p_B .

As mentioned above, a fundamental output parameter of the SV is the spool position x_v . To improve the accuracy of this system under medium to large variation in input parameters, it is necessary to implement the *flapper – nozzle* system as a 3rd order model [4].

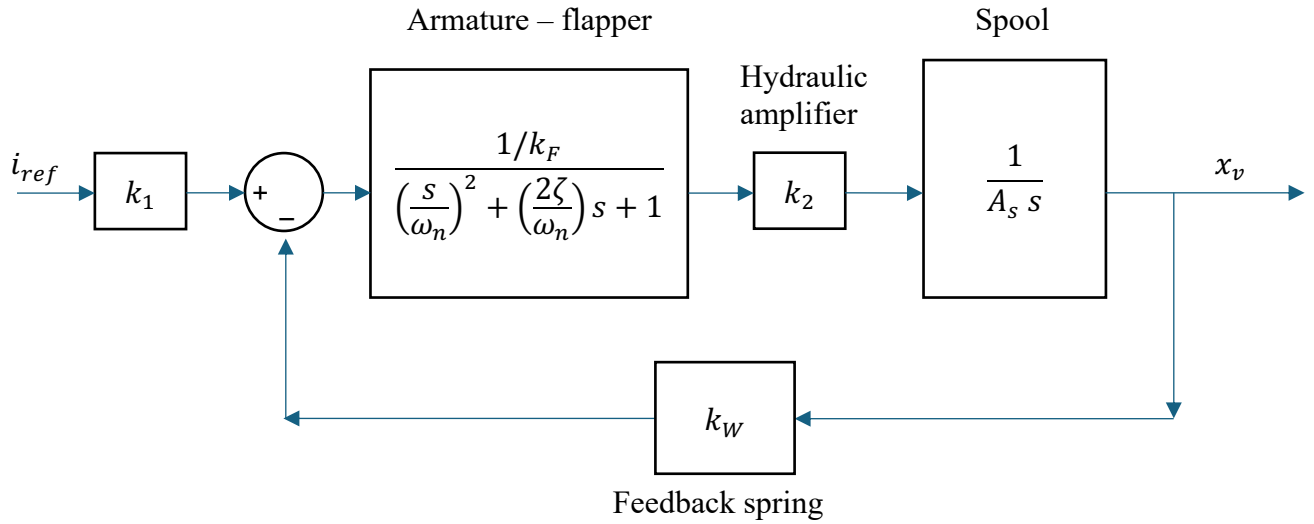


Figure 44: 3rd order model for the Flapper – Nozzle system.

Physical quantities will be better detailed in the next table, let's now consider this 3rd order model. Since there is a feedback line, it can be rewritten as a closed loop transfer function.

The forward path TF is a product of the 3 main blocks:

$$H(s) = \frac{1/k_F}{\left(\frac{s}{\omega_n}\right)^2 + \left(\frac{2\zeta}{\omega_n}\right)s + 1} k_2 \frac{1}{A_s s} = \frac{k_2/(k_F A_s)}{\frac{s^3}{\omega_n^2} + \frac{2\zeta}{\omega_n}s^2 + s}$$

As it can be seen, this is in fact a 3rd order model. The error TF entering the forward path can be written as:

$$E(s) = i_{ref} k_1 - x_v k_W$$

The spool position is obtained by multiplying the error by the forward TF.

$$x_v = E(s) \cdot H(s)$$

Substituting the two TF inside this equation and solving for x_v we obtain:

$$x_v \left(\frac{s^3}{\omega_n^2} + \frac{2\zeta}{\omega_n}s^2 + s + \frac{k_W k_2}{k_F A_s} \right) = i_{ref} \frac{k_1 k_2}{k_F A_s}$$

$$x_v = i_{ref} \cdot \frac{\frac{k_1 k_2}{k_F A_s}}{\frac{s^3}{\omega_n^2} + \frac{2\zeta}{\omega_n}s^2 + s + \frac{k_W k_2}{k_F A_s}} = i_{ref} \cdot TF_{CL}$$

TF_{CL} represents the final closed loop transfer function we are looking for. Now the entire diagram of Fig. 22 can be redrawn as a single block with i_{ref} as an input, TF_{CL} as the only TF, and x_v as output.

In MATLAB we will apply this TF with the *filter*() function to the gained error compensation current i_{ref} .

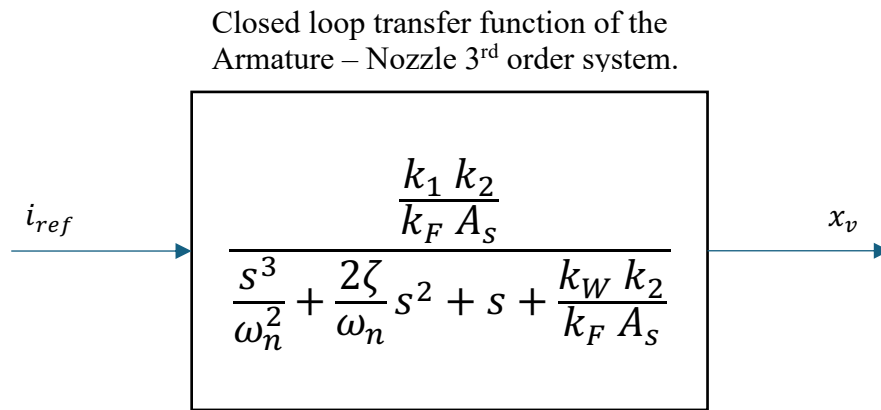


Figure 45: 3rd order model redrawn as a single closed loop TF.

Symbols Legend

Full technical data sheet of this SV is available for consultation on the Bosh Rexroth website; we will only report a portion of this data. We will limit to what is required to implement a simplified, yet sufficient representation of the Digital Twin of this SV. In this thesis modeling this component through a physically informed neural network should be attempted with an incomplete knowledge of the real mechanical system.

Spool – Flapper		
K_{cond}	0,001 V/A	Static conditioning gain: $V_{ref} \rightarrow i_{ref}$
X_{max}	$0,50 \cdot 10^{-3} \text{ m}$	Spool maximum stroke, from position 1 to 3
$GM = X_{max}/i_{max}$	0,05 m/A	SV spool mechanical gain
A_s	$1,68 \cdot 10^{-5} \text{ m}^2$	Spool end cross section (piloting stage)

Flow characteristics		
Q_n	0 m	Nominal SV flow
ρ_{oil}	$857,20 \text{ kg/m}^3$	Oil density (depends on oil used)
ΔP_n	$70 \cdot 10^5 \text{ Pa}$	Nominal differential pressure from S to R: $p_S - p_1 + p_1 - p_2 + p_2 - p_R = p_S - p_R$. Nominal flow rate is assumed.
C_v	0,68	SV Flow coefficient

Port geometry		
$\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4$	0 m	Spool – port overlap/underlap
w_s	0,108 m	Port width
h_{rad}	$3,125 \cdot 10^{-6} \text{ m}$	Port radial clearance

$A_{max} = \frac{Q_n}{C_v \cdot \sqrt{\Delta P_n / \rho_{oil}}}$	$5,42 \cdot 10^{-6} m^2$	Port maximum area
A_o	$7,85 \cdot 10^{-5} m^2$	Bypass orifice maximum flow area
C_{bp}	1,0262	Bypass orifice discharge coefficient

SV dynamics		
ω_n	$2,51 \cdot 10^3 rad/s$	SV first stage 1 st resonant frequency
ζ_n	0,50	SV first stage damping factor
ζ	0,70	SV damping factor
k_F	$511,5417 Nm/m$	Armature – Flapper TF proportional gain
k_1	$2,8246 Nm/A$	Torque motor gain
k_2	$0,0968 m^3/ms$	Hydraulic amplifier gain
k_w	$74,2848 Nm/m$	Feedback wire proportional gain

3.2 Data acquisition

Data logging is performed by the control hardware c-RIO 9039 that enables data acquisition through multiple channels using its dedicated modules. This controller supports up to 8-bit acquisition with a maximum baud rate of 115,200 bps.

The c-series transmitters NI 9218, NI 9263, NI 9205 and NI 9401 allow digital and analog acquisition of data. Refer to Fig. 4 for further details.

0.000000	-0.005076	-0.066959	-0.004578	2.760614	2.782587	3.933037	3.087303	0.000000	-0.007615	0.000000
0.000000	0.005746	-0.064301	-0.020935	2.761598	2.783243	3.932381	3.087303	0.000000	-0.006303	0.000000
0.000000	-0.007044	-0.061183	0.005707	2.757335	2.787834	3.921886	3.087303	0.000000	-0.010894	0.000000
0.000000	-0.003109	-0.063185	-0.006683	2.764222	2.776028	3.934021	3.087303	0.000000	-0.008271	0.000000
0.000000	-0.003764	-0.071390	-0.000916	2.756679	2.783898	3.931725	3.087303	0.000000	-0.008599	0.000000
0.000000	0.004107	-0.071949	-0.023193	2.761926	2.779963	3.930413	3.087303	0.000000	-0.010566	0.000000
0.000000	0.009354	-0.063251	-0.021515	2.758318	2.779307	3.935660	3.087303	0.000000	-0.013846	0.000000
0.000000	-0.003109	-0.066894	-0.003479	2.756351	2.785866	3.926805	3.087303	0.000000	-0.008271	0.000000
0.000000	-0.003437	-0.061117	-0.000336	2.757990	2.775700	3.931069	3.087303	0.000000	-0.004335	0.000000
0.000000	-0.003109	-0.072047	-0.008362	2.754383	2.776356	3.928773	3.087303	0.000000	-0.011878	0.000000
0.000000	-0.001141	-0.067813	-0.008331	2.754055	2.777011	3.931725	3.087303	0.000000	-0.004663	0.000000

Figure 46: sample picture of the sensors logs file structure.

The sensor log file has a plain text ‘.txt’ format and it’s structured in columns corresponding to input and output data streams, as follows:

- **Channel 1:** HLA force set command V_{SET} (input)
- **Channel 2:** HLA force feedback from load cell LC V_{fb} (output)
- **Channel 3:** LVT sensor (output)
- **Channel 4:** servo valve error compensation signal V_{ref} (output)
- **Channel 5:** pressure of HLA chamber A V_{pA} (output)
- **Channel 6:** pressure of HLA chamber B V_{pB} (output)
- **Channel 7:** supply pressure from power regulator group V_{pS} (input)
- **Channel 8:** digital encoder measurements (input)
- **Channel 9:** EMA position set command EMA_{SET} (input)
- **Channel 10:** EMA position feedback derived from digital encoder EMA_{fb} (output)
- **Channel 11:** empty data stream, unused.

Natively all the data is acquired with a sample frequency of $f_s = 1000 \text{ Hz}$ to minimize aliasing phenomena. Considering results of the Nyquist’s Theorem and antialiasing filters applied during acquisition, the maximum frequency of an event to be sampled in this condition should be below $f_{max} = 1000/2,56 \cong 390 \text{ Hz}$, limit near the cutoff frequency of a standard Antialiasing filter.

In the table below, data streams are divided in input and output variables to be processed by the LSTM PINN.

Table 3: detail of input and output converted variables.

Input Variables – $f_s = 1000 \text{ Hz}$	
$F_{SET} = V_{set} \cdot K_F$	Force set [N]
$b_{SET}/i_{transm} = EMA_{SET} \cdot K_{EMA}$	EMA set [m]
V_{ref}	Error compensation voltage [V]
$p_S = V_{pS} \cdot K_p$	Supply pressure [Pa]
t	Time vector [s] (virtual variable computed from sample frequency)
x_v	Spool position (virtual variable computed in pre-processing) [m]
Output variables – $f_s = 1000 \text{ Hz}$	
$p_A = V_{pA} \cdot K_p$	Pressure of HLA chamber A [Pa]
$p_B = V_{pB} \cdot K_p$	Pressure of HLA chamber B [Pa]
$F_{fb} = V_{fb} \cdot K_F$	LC force feedback [N]
$b_{fb}/i_{transm} = EMA_{fb} \cdot K_{EMA}$	EMA feedback from digital encoder [m]
$LVT = (V_{LVT} + o_f) \cdot K_{LVT}$	LVT velocity signal [m/s]
Sensor gains and offsets	
$K_F = 5000$	Load cell (or force command) gain [N/V]
$K_{EMA} = 5 \cdot 10^{-3}$	EMA position gain [m/V]
$K_p = 35 \cdot 10^5$	Pressure transducer gain [Pa/V]
$K_{LVT} = 0,1$	LVT proportional gain [m/Vs]
$o_f = 0,055$	LVT static offset [V]
$i_{trans} = \frac{y_{HLA}}{y_{EMA}} = 0,45$	Revolute joint transmission ratio

3.3 Proof of concept

Now that the entire physics of this system has been introduced, we will focus on the real practical reason we are using a physics informed approach rather than a purely data driven one. Although the complete implementation of this algorithm, model, and the yet to be introduced optimization process, will be later detailed in dedicated chapters, two results will be now analyzed. For a moment, let's now take a step beyond skipping all theory introductions and we evaluate the fully trained, with optimized hyperparameters, LSTM PINN model in two variations.

The first will have the physical laws disabled, so that no physical link between reconstructed data is present. The second is the final full implementation with everything enabled.

As mentioned before, we will demonstrate the main flaw of a pure data driven approach by considering an experiment whose output and input variables are listed in Tab. 2.

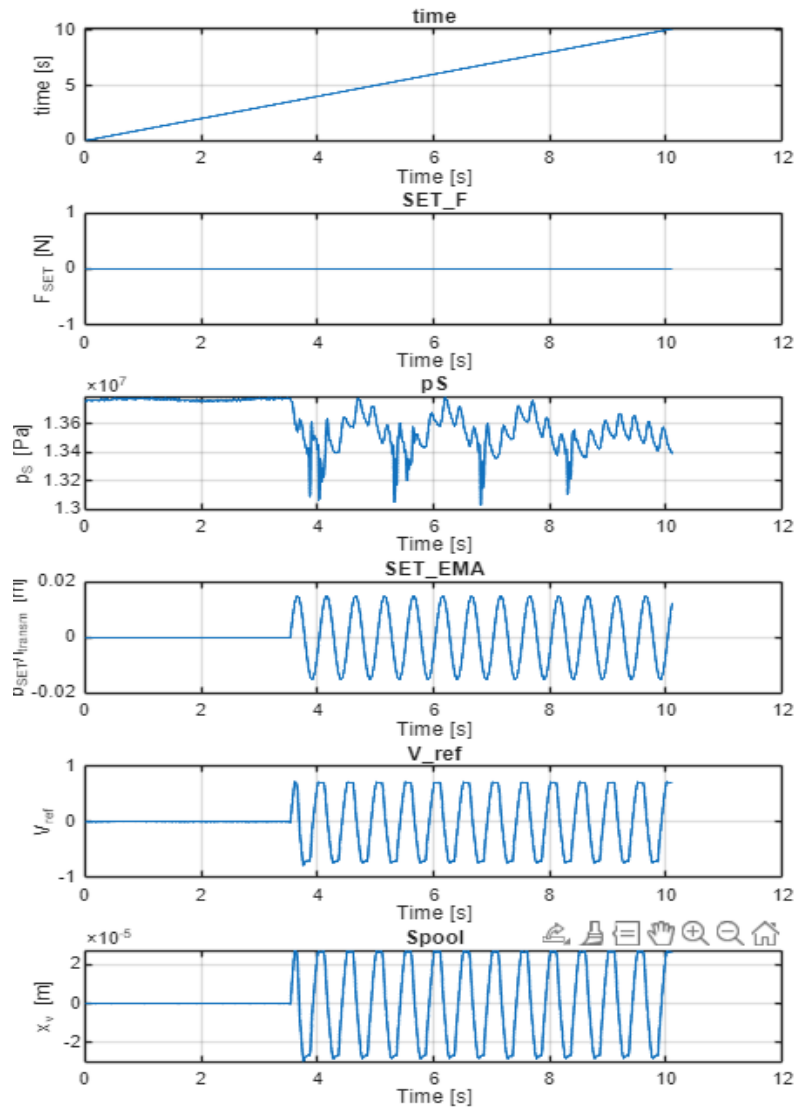


Figure 47: Input variables of sample experiment.

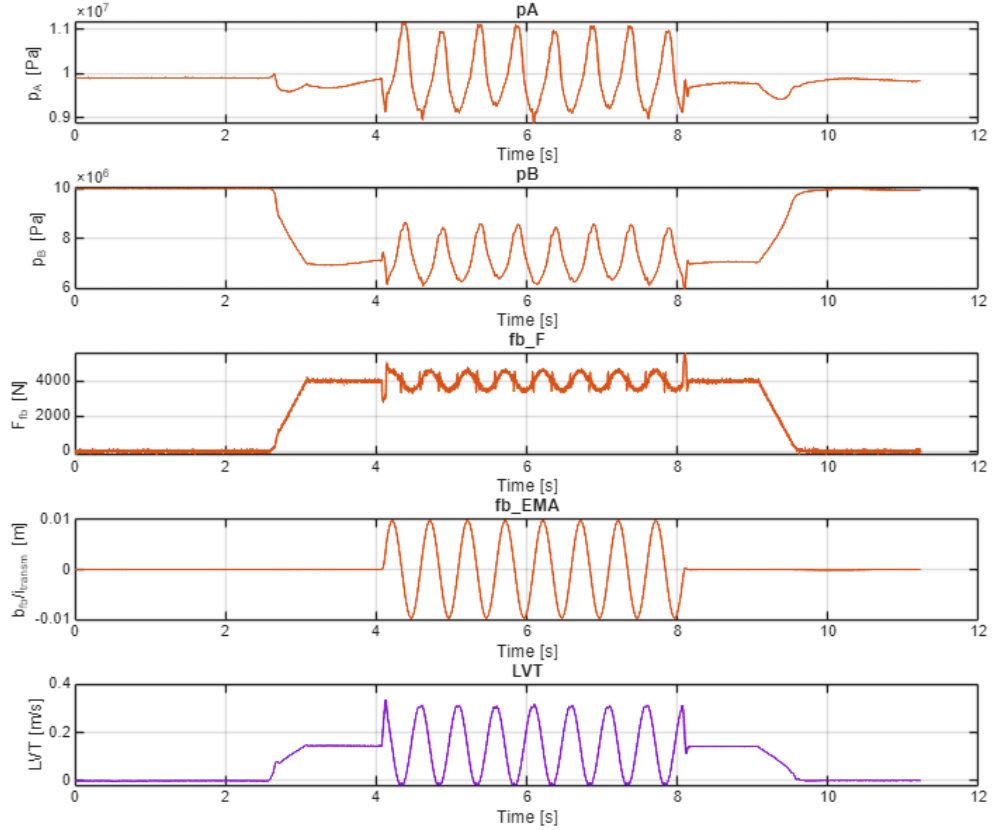


Figure 48: Output variables of sample experiment.

In this experiment, the EMA is set to perform sinusoidal oscillations around the resting position, HLA is disturbing this motion with a force set as on Fig. 25: a ramp command until 4000 N, which will be held constant for approximately 7 seconds, and a symmetric ramp returning to zero.

Output variables are measured with ASTIB test bench acquisition system as on Fig. 26, but the LVT (shown in purple) has been purposely modified to mismatch the real working conditions. A double ramp signal, similar in structure to the HLA force set, has been added to it introducing a velocity signal that wasn't in the experiment before, thus making no physical sense.

In the next diagram we will analyze the data driven LSTM prediction of output variables. Notice that in this case, only a numerical loss has been introduced:

$$L_{mse} = \frac{1}{N} \sum_{i=1}^N (y(t_i) - T(t_i))^2$$

The noisy red curve is a LSTM prediction $y(t)$, while the blue curve is a sensor output $T(t)$.

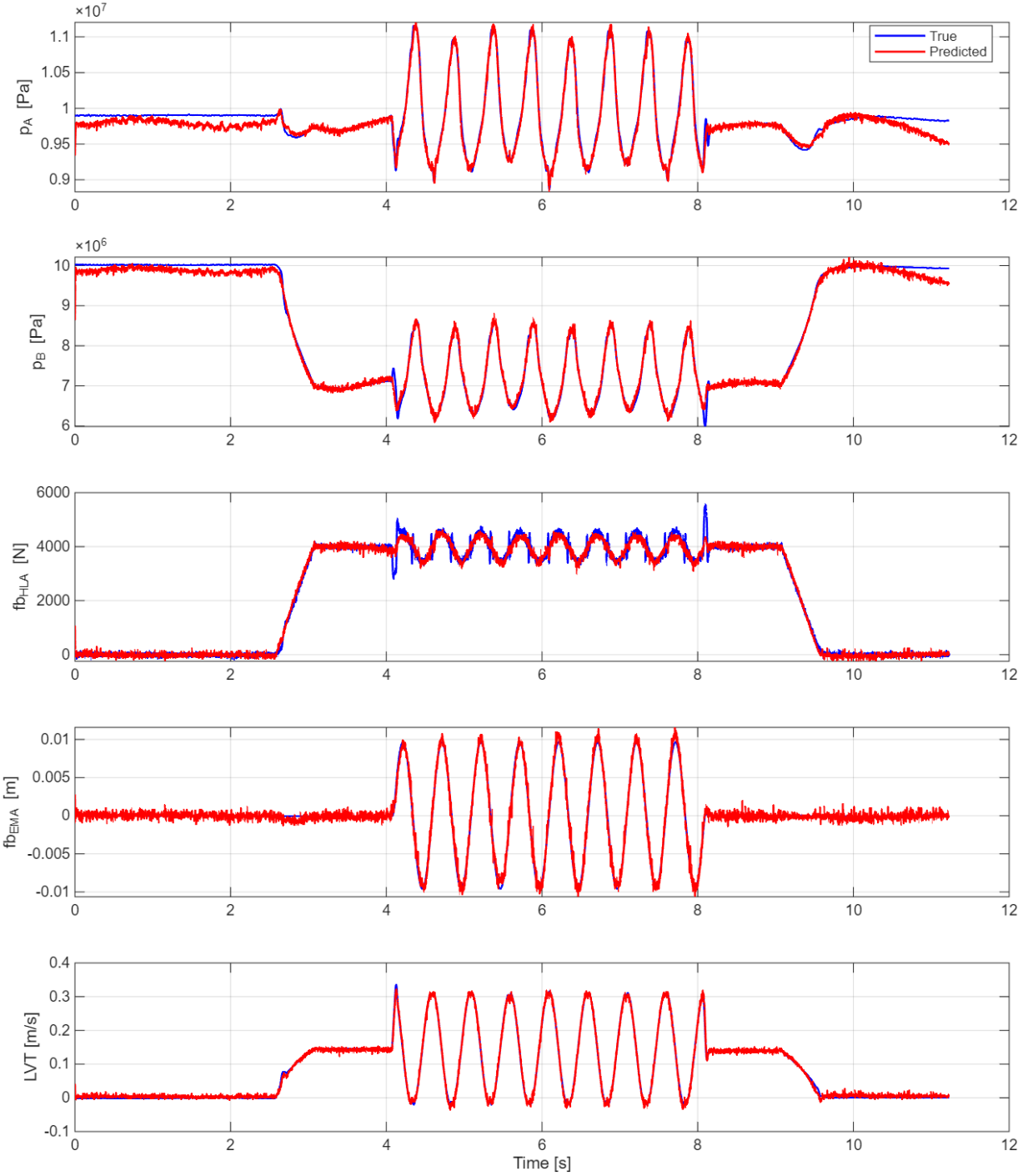


Figure 49: Time series comparison plot with data driven approach.

This is a time series plot, where the neural network is performing the forward step to predict the 5 output variables listed in Table 2. In this experiment the EMA is still from $t = 6$ s, to $t = 8$ s, but the LSTM network perfectly reconstructs it, validating the hypothesis. This network is incapable of learning physical systems behavior because it only acts as a *curve fitter*, with no physical link between its outputs.

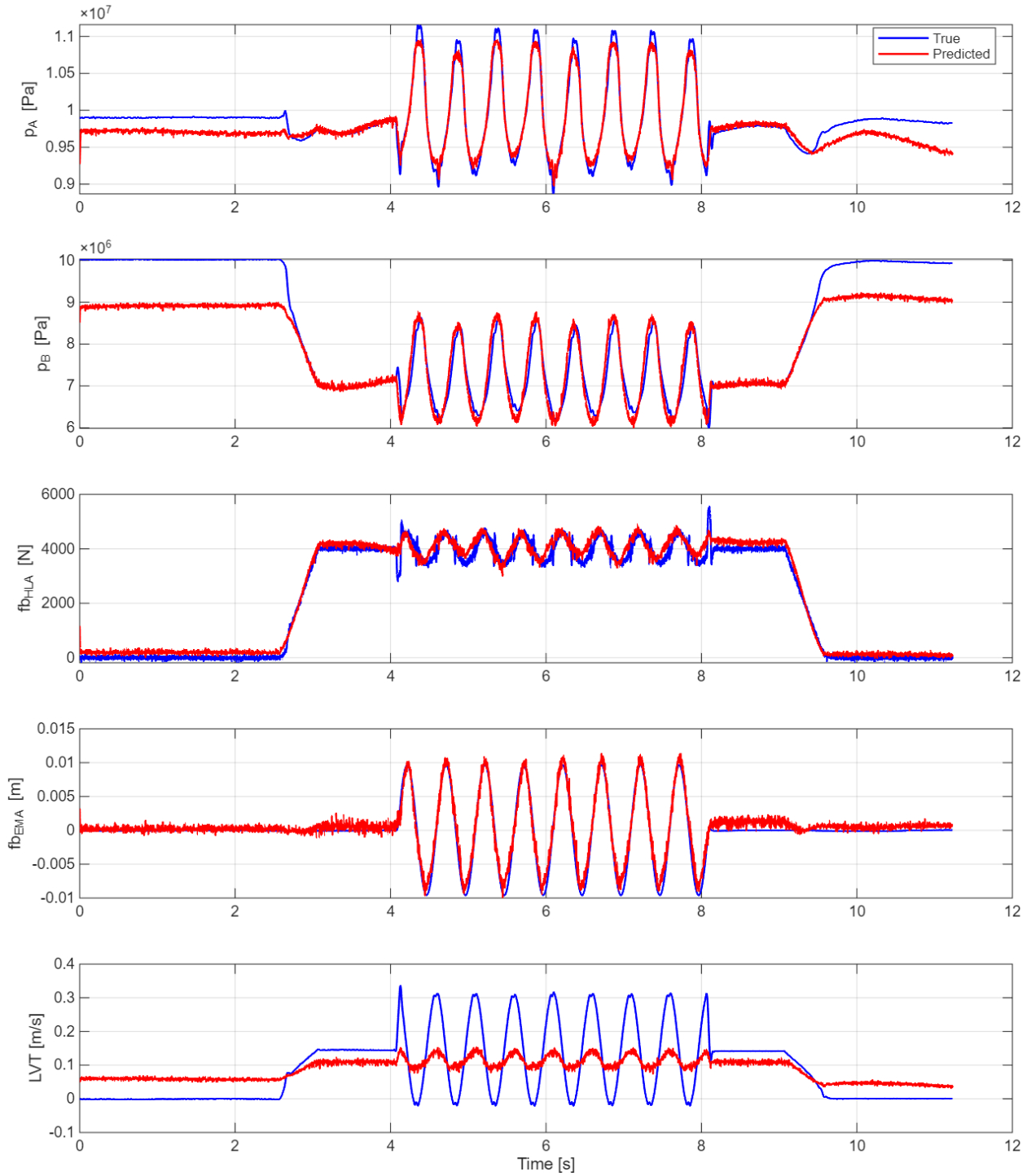


Figure 50: Time series comparison plot with physics informed approach.

The physical informed approach is expected to fail in reconstructing outputs of this experiment; we will demonstrate in the next chapters how a physical law is integrated inside the custom loss function and how it penalizes the model during backpropagation through time.

For now, we limit to show the final loss structure, used by the gradient descent optimization algorithm. At every epoch, gradient descent considers this loss term:

$$L_{tot} = L_{mse} + \lambda_c L_c$$

- L_{mse} represents the numerical mean squared error loss mentioned before.
- λ_c is a weight hyperparameter used to balance loss contribution, constant during training.
- L_c is a loss tied to any physical law that if not respected, will penalize the model.

If the weight hyperparameter is constant as hypothesis, the physical loss term is impossible to satisfy in this case, the BPTT will find a way to minimize the L_{tot} by accepting a penalty from both L_c and L_{mse} , coming up with a tradeoff configuration between physics and numerical losses.

Notice that, since λ_c is a hyperparameter fixed during the training process, the neural network is prevented to simply ignore L_c by forcing the weight term to zero.

In the final physical loss implementation the LVT signal is tied to other output variables and the model tries to minimize the total loss by predicting a nearly constant term that minimizes the total loss in a single epoch. This is a known behavior of a neural network training process, these models are inherently ‘lazy’ and try to minimize the error the most effective way possible, which is, most of the times, not the correct solution. This is also noticeable in the next scatter plot.

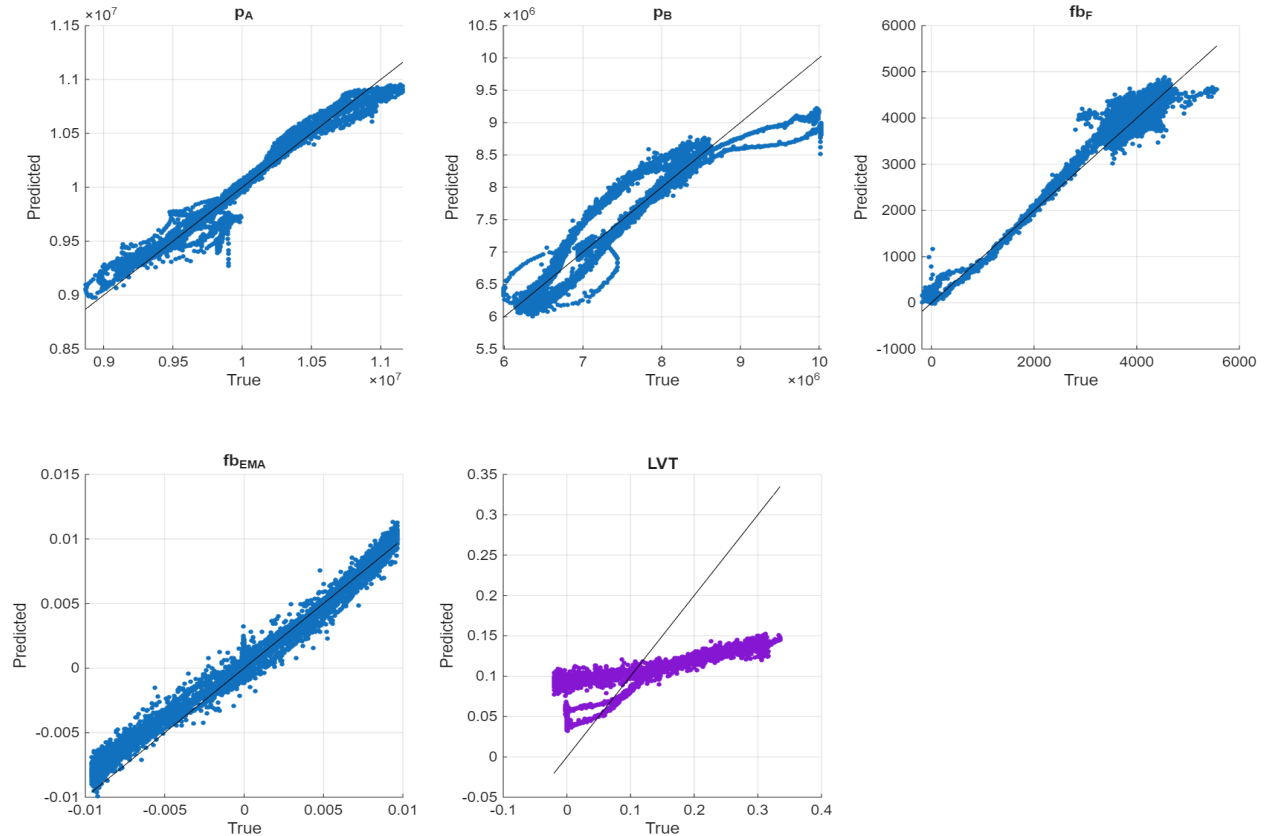


Figure 51: scatter plot of the physics informed approach.

3.4 Physical law implementation

This implementation of the LSTM PINN uses a customized training loop with user defined loss function architecture, a detailed explanation on the integration of this algorithm in MATLAB will be addressed in dedicated chapters.

Here, the physical framework of this loss term will be analyzed, and it will be suggested an approach that makes use of the continuity laws to link the change in oil pressure inside HLA chambers to the flow rate through the servo valve (SV) and to the piston position and velocity signals.

Law of compressibility

Equation (3.1) is reported here, this is obtained applying eq. (3.2.1) to HLA chamber A which is loading (Q_A term is positive).

$$\frac{Q_A - (\dot{a} - \dot{b})A - Q_{bp} - Q_{leak}}{V_A + (a - b)A} \cdot \beta_{eq} = \frac{dP_A}{dt}$$

The first term contains position and velocity signal from EMA position sensors and LVT inside the HLA actuator. Let's focus on LVT, by refactoring equation (3.2.4) we obtain:

$$\dot{a} - \dot{b} = \frac{V_{LVT} + o_f}{k_{LVT}} = LVT$$

If we consider $a \cong 0$, meaning the cylinder could be considered still for slow variation of the inputs (slow dynamics hypothesis), we can say that the entire relative position signal $a - b$ is generated by a motion of the piston.

This position signal is measured on the EMA side. As in Table 2, we can write:

$$b_{fb}/i_{transm} = EMA_{fb} \cdot K_{EMA}$$

Where $i_{transm} = 0,45$ as mentioned in Chapter 3.2, on Fig. 11.

The second term of this equation is a pressure time derivative, that could be evaluated using *automatic differentiation* functions in MATLAB, like `dlgradient()`. Even if this is a computationally efficient and stable algorithm, performing derivatives of a neural network prediction can quickly lead to numerical instability.

As seen on Figg. 27-28, the red curve representing neural network's output is noisy, and a pressure signal measured in Pa, since this test bench is supplied by hydraulic oil from the power regulator group that could reach up to $140 \text{ bar} = 14 \cdot 10^6 \text{ Pa}$. Noise oscillations showed in that diagram can easily reach orders of magnitude of 100 or even 1000 Pa, even though an explicit derivative implemented in the model could be sometimes beneficial [23], we decide not to implement this derivative approach in this physical loss. Oscillations would be enough to completely break the physics, and it would be impossible for the model not to incur in modal collapse and predict completely flattened quantity that minimize this derivative, main term of the physical loss (Fig. 29).

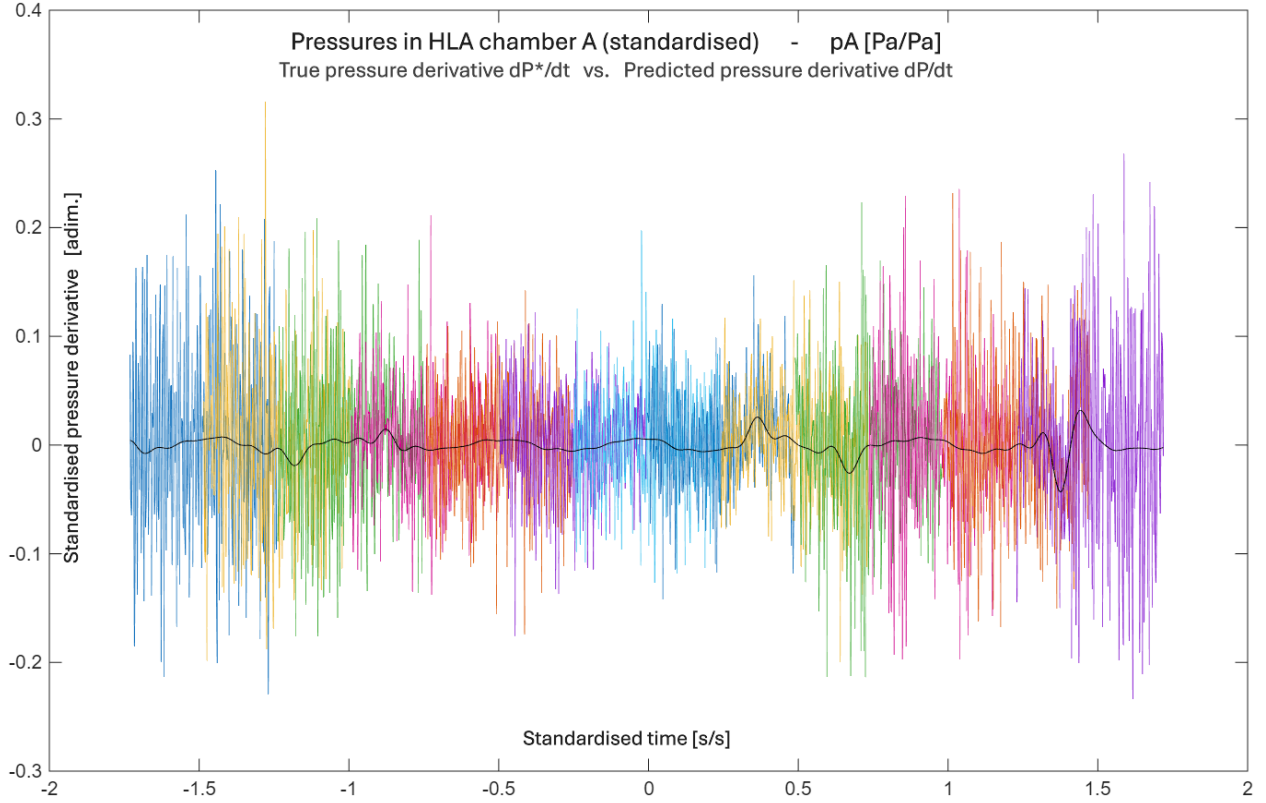


Figure 52: standardized pressure derivative vs standardized time.

In this diagram is clearly noticeable the effect of differentiation of a noisy variable. The center black curve is the same standardized derivative of p_A in time using the true value of pressure in HLA chamber A measured by pressure transducer and filtered in data pre-processing phase.

Colored noisy lines represent the derivative of p_A as a network prediction in early training phases (after ~ 100 epochs). This behavior is destined to persist until the network comes up with a ‘lazy’ solution to minimize the derivative, which will be integral part of the physical continuity loss L_C .

We can compute the average peak value of the colored pressure derivative lines by reversing standardization pass:

$$\frac{d(f - \mu_f)/\sigma_f}{dt} = \frac{1}{\sigma_f} \frac{df}{dt}$$

Then, by ignoring the mean value μ_f , which is time independent, we obtain the non-standardized function derivative by multiplying it by σ_f , its standard deviation. Variables are standardized to avoid the effect of different orders of magnitude as mentioned in Chapter 1.2 (Data Quality and Preprocessing step). This is an optimal procedure during evaluation of the numerical loss of the data driven approach.

$$\sigma_{Pa} = 5,82 \cdot 10^{+5}$$

We can assume a derivative peak value, as the $\sim 1\%$ high percentile, of approximately:

$$\left. \frac{d\tilde{p}_A}{dt} \right|_{0,95} = 0,25$$

This means:

$$\left. \frac{dp_A}{dt} \right|_{0,95} = 1,45 \cdot 10^{+5} \text{ Pa/s}$$

This is unacceptable when comparing this number to other terms, which order of magnitude is significantly smaller. This means the loss must be implemented in integral form.

Integrating the law of compressibility

In this section we propose a naïve approach to implement the compressibility law of a fluid and refactor it into a loss term that can be processed by a BPTT optimization algorithm.

NOTE: Initially, the physical quantities should be de-standardized, the compressibility law does not function properly when operating with normalized variables since every one of them should have a different order of magnitude.

We report again equation (3.1) for chamber A:

$$\frac{Q_A - (LVT)A - Q_{bp} - Q_{leak}}{V_A + (-b_{fb}/i_{transm})A} \cdot \beta_{eq} = \frac{dP_A}{dt}$$

If we take the integral from $0 < t < t$, (up to the variable time t), we obtain:

$$\int_0^t \left(\frac{Q_A - (LVT)A - Q_{bp} - Q_{leak}}{V_A + (-b_{fb}/i_{transm})A} \cdot \beta_{eq} \right) \cdot dt = p_A(t) - p_{A0}$$

The second term is just an application of the fundamental theorem of calculus, while the first term is integrated by parts.

$$\int_0^t R(t)dt = p_A(t) - p_{A0}$$

In ideal conditions, this law will be perfectly respected, however, since the PINN will reconstruct the output learning this physical law each epoch, the right-hand side integral will never equal the second term. Thus, we can write the difference of the 2 as an error term:

$$\int_0^t R(t)dt - p_A(t) + p_{A0} = E_C \cdot p_{SCALE}$$

To account the order of magnitude of the HLA chamber pressure we could either standardize the error term again or just scale it down with a term with the same order of magnitude of a general pressure term.

$$p_{SCALE} = 30 \text{ MPa}$$

$$r_A = \frac{E_{CA}}{p_{SCALE}}$$

Analogous considerations can be applied to chamber B, to obtain:

$$r_B = \frac{E_{CB}}{p_{SCALE}}$$

The final compressibility loss term is computed every epoch as follows:

$$L_C = \frac{1}{N} \sum_{i=0}^N (r_A^2 + r_B^2) \quad 3.4.1$$

written in form of a mean squared error, summing through all N timesteps.

The core idea behind a variable extreme integration is to link a physical law to every iteration of the learning process of a neural network. As of Eq. (2.4.2), every timestep an error is computed, then all the contribution are summed together to form the final loss term. Since each loss component is greater than zero by design, only by singularly reducing every term, the final loss can be minimized.

During BPTT, each t -th error gradient with respect to the weight matrix is referred to timestep t . If we had performed the integration of compressibility law from $t = 0$ to $t = T$, we would have obtained the same error gradient each timestep. In this situation the BPTT would have been under constrained, since the physical loss term would have been constant in time. No information about prediction errors would have been retained during timestep between the first and the last timestep, hence the gradient would always point to the wrong direction.

With a variable extreme integration, at a generic timestep t , the residual term is a cumulative summation of all the previous terms as well, so the generic r_j written for the j -th chamber is a vector where each entry is a step of this discrete time cumulative summation (discrete integration). By hypothesis, the only solution to reduce this residual to the minimum, is to minimize $r_j(t)$ at every timestep.

4 Hyperparameters search

4.1 Introduction to hyperparameters

The training process of a neural network does not only depend on weight matrices and backpropagation. As showed in (1.3.2) with a conventional static backpropagation, but this could be extended to BPTT as well, the parameters update depends on the *learning rate* α .

$$\theta^{e+1} = \theta^e - \alpha \frac{\delta L_{mse}(\theta^e, X)}{\delta \theta}$$

A neural network has no direct control on this number, which in a conventional model is kept constant during the training process.

In (3.3.2) we introduced a formulation to compute the total loss, including the physics contribution.

$$L_{tot} = L_D + \lambda_C L_C$$

Again, a *loss weight* λ is another parameter not controlled by the model itself.

These values are called *hyperparameters*, chosen by the user to optimize training. Every parameter not directly controlled by a model falls in this category, this includes the following:

- *Learning rate*: gradient descent scalar factor.
- *Epochs*: the number of times a neural network processes the entire dataset.
- *Batch size*: a batch is a subset of the dataset; it represents the number of examples the model processes before updating its parameters. This can improve parallelization since minibatches are processed simultaneously on a machine.
- *Loss weights*: hyperparameter used for balancing loss terms.

Most important hyperparameters are detailed below.

LEARNING RATE

The learning rate α governs the weights update process. It defines which fraction of the loss gradient, computed in backpropagation, is responsible for the change in network parameters, as in Chapter 2.3. Weights and biases are updated to converge the error function to a global minimum, if the learning rate is too small, this process won't end in a reasonable amount of time, and the training would be inefficient. If α is too high, corrections are heavy and the algorithm could skip past a minimum point.

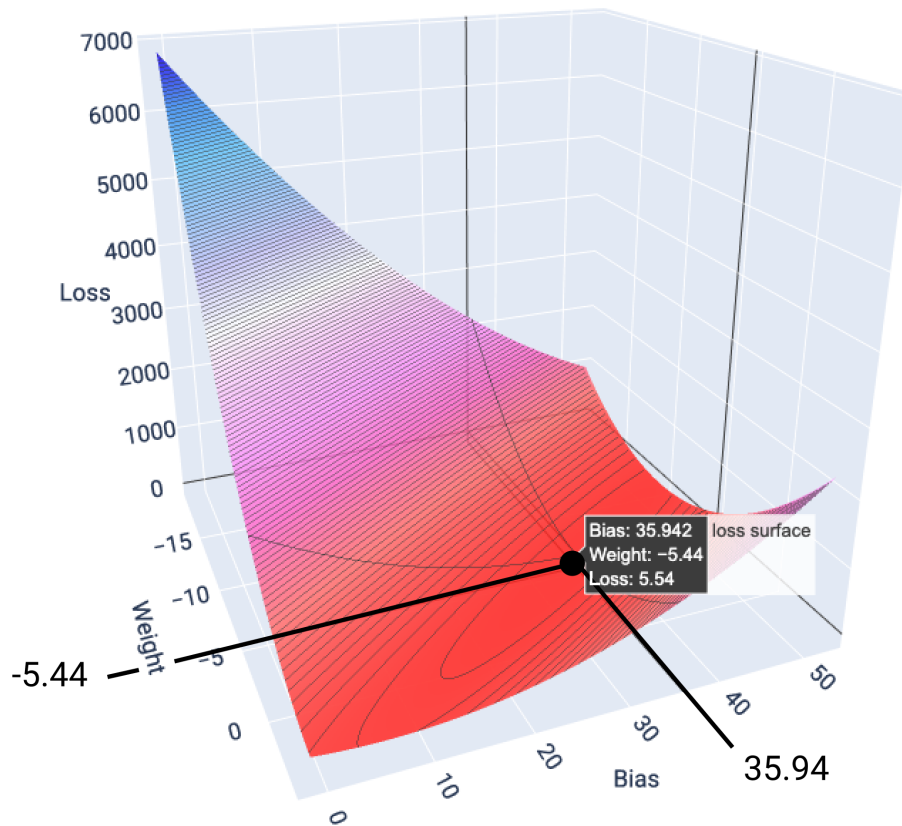


Figure 53: loss function vs network parameters – Google, CC BY 4.0

This diagram shows a loss function, computed for every combination of weights and biases, of a generic FCNN model performing a linear regression problem [10]. Powerful implementations may have a more complex *loss surface*, but the backpropagation algorithm must find the global minimum to minimize the error.

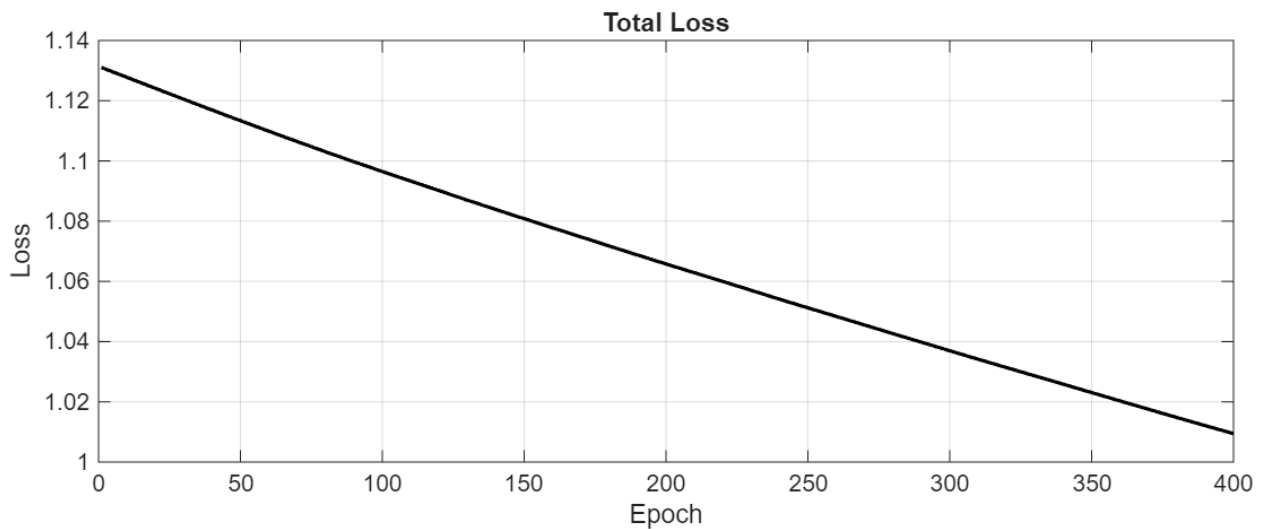


Figure 54: LSTM PINN with an extremely small learning rate.

In this case, the LSTM PINN model has been trained with an insufficient learning rate over 400 epochs. Every iteration of the gradient descent, this model makes negligible improvements and nothing significant can be learned in a reasonable amount of time.

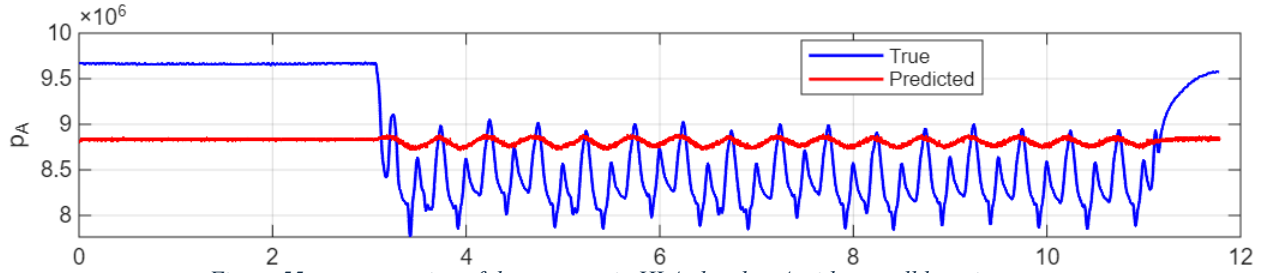


Figure 55: reconstruction of the pressure in HLA chamber A with a small learning rate.

In contrast, if a big learning rate is fixed, the evaluated loss curve violently oscillates, suggesting a heavy update of network parameters. In this case, the model might never converge.

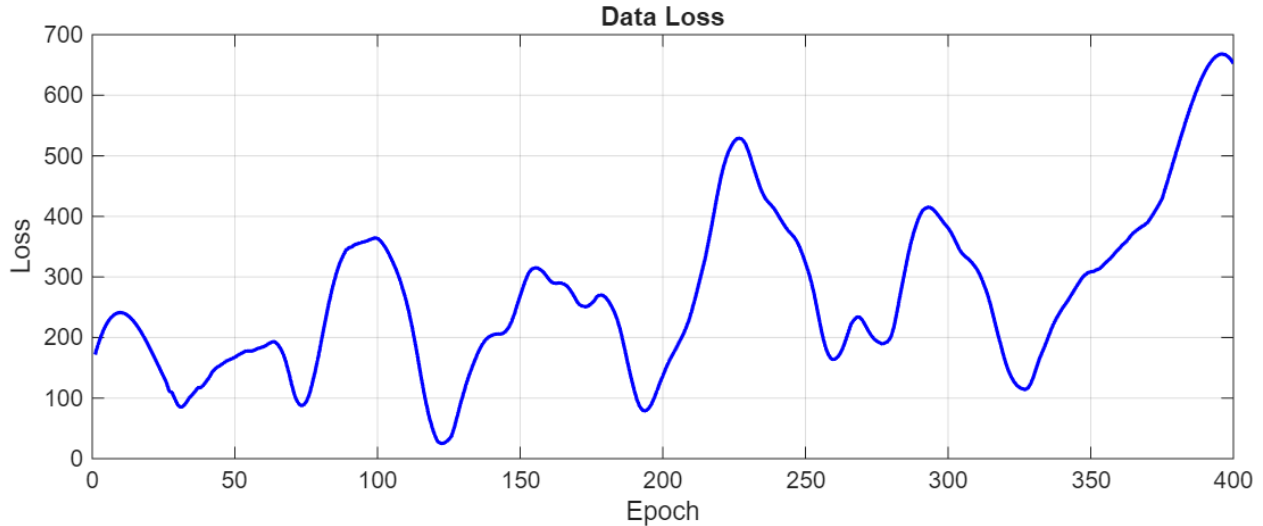


Figure 56: LSTM PINN with a big learning rate.

In this diagram the loss is slowly increasing with epochs, this could happen with extremely big learning rates, making convergence impossible.

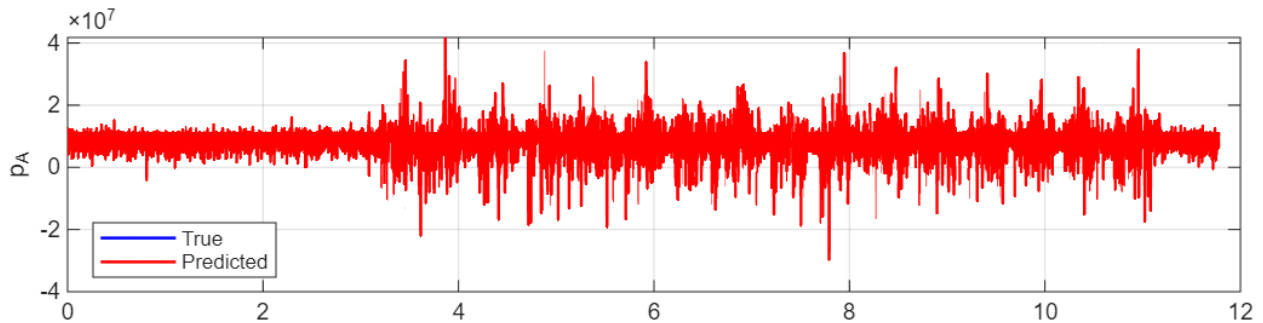


Figure 57: reconstruction of the pressure in HLA chamber A with a big learning rate.

BATCH SIZE

This number governs the quantity of samples the network processes before updating its parameters. In a LSTM neural network, weight update is performed each epoch using backpropagation through time (BPTT). If we consider a single time series, this means processing a vector of time samples with dimensions:

$$x \in \mathbb{R}^T$$

Where R is a set of real number, expressed in computer floating point math. The model performs the forward step, every timestep, with a single sample x_t and evaluates the instant loss L_t . At the end of an epoch, the total loss is computed with a summation over t as in (2.4.2).

This is the process of a BPTT with batch size $B = 1$, every sample is processed in series.

If we fix the number of batches B , where B is an integer, we could process a set of B sample points *in parallel*. This happens if we have multiple time series – experiments with different set and feedback signals in a servo system – and we need to train a neural network on all of them. Let's consider a generic experiment with C sensors, this current batch size and several time samples. The dataset can be represented by the tensor:

$$X \in \mathbb{R}^{C \times T \times B}$$

Where:

- C is the *channels* dimension, equals the number of sensors. If in one experiment, pressure in both HLA chambers, piston LVT velocity and position are processed, channel dimensions will be $C = 4$.
- B is the number of mini batches.
- T is the time dimension as before.

The n batches are computed in parallel and independently one each other. In a LSTM unit, no *cell status* C_t links data point across the B dimension, since there is no temporal dependence amongst the data. At the end of an epoch, a loss vector – B long – is computed.

$$L^B = \sum_t L_t^B$$

$$L^B = [L_1, \dots, L_B] \in \mathbb{R}^B$$

This is a set of independent losses that will be aggregated to form a single scalar value. Typically, this is obtained with an average:

$$L = \frac{1}{B} \sum_{b=1}^B L_b$$

At this point, the neural network parameters update is performed with BPTT.

Another application of this strategy is to slice a single long time series into smaller subset, again called *mini batches*, with batch size $N_B = n < T$. This can greatly increase the training speed compared to series processing ($B = 1$) due to a better parallelization of the algorithm.

This technique would sever the time dependence across the B dimension of the tensor, but this issue is partially solved with the loss aggregation process.

Since every batch is processed independently, in a LSTM neural network, the cell state C_t – and consequently the previous hidden state h_{t-1} – is reset at the beginning of every mini batch, this means all the context information is initialized at zero. In the training process of a recurrent neural network like LSTM, the model first reacts to a big error term when the first cell and hidden states are created. This leads to *warm up phase* noticeable in a diagram if an experiment from the training set is plotted.

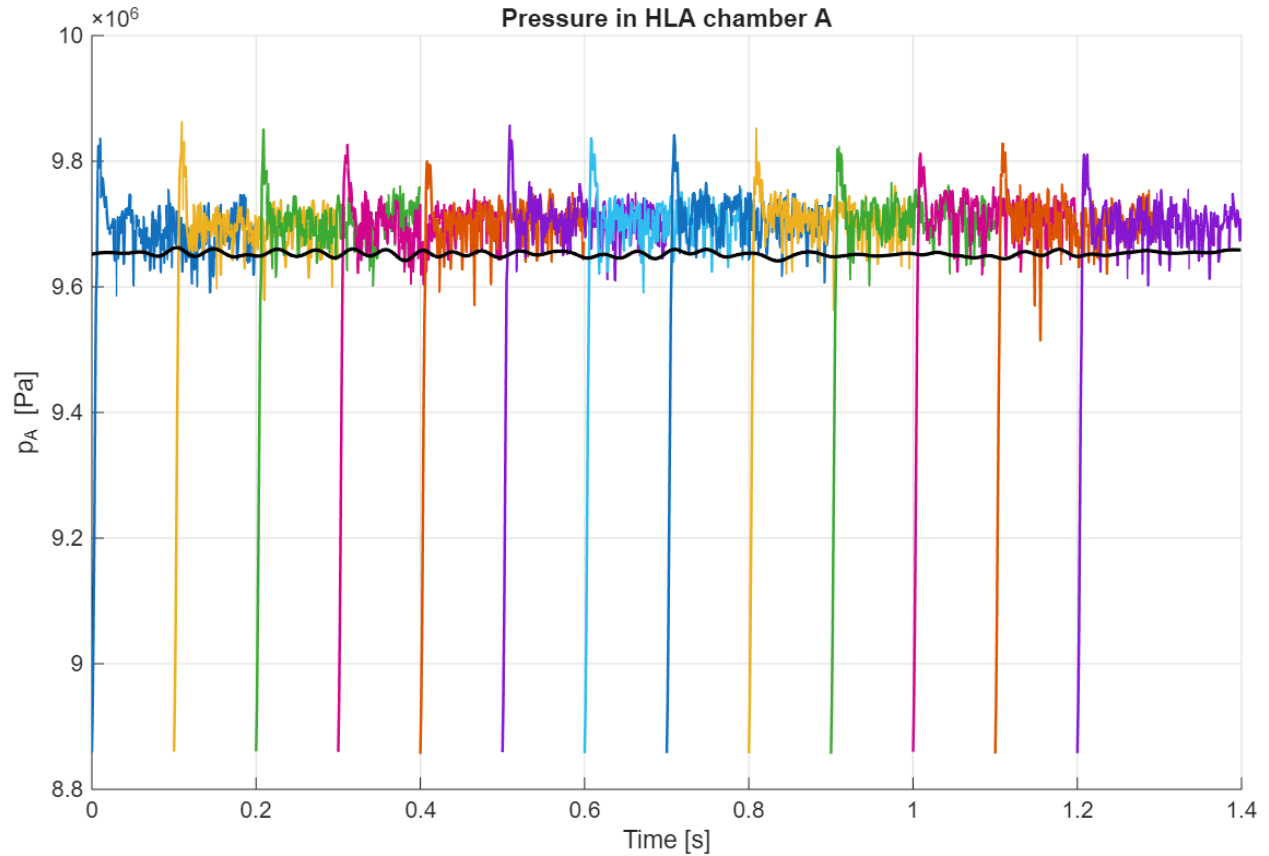


Figure 58: comparison between sensor p_A and reconstructed p_A using minibatches.

This graph shows a portion of the pressure in HLA chamber A. The black line is the real pressure p_A measured by sensors on the ASTIB testbench. Colored noisy lines represent a LSTM reconstruction of this pressure, after ~ 300 pochs, using this minibatch settings:

- Batch size $N_B = 200$ samples.
- Overlap $S_B = 100$ samples, to better represent time dependence batches are overlapping.

Mini batches implementation can significantly speed up the training of a model without a noticeable degradation of prediction accuracy. In the next table, training time is analyzed comparing a full series processing strategy, to the current mini batch implementation.

The model is trained using 1 experiment (training set) with a total of 10110 samples in time, for each channel. Input data matrix has a total of 7 channels considering time vector and valve spool position. Output matrix has 5 channels. The final total loss is evaluated on 1 different experiment (validation set) with similar characteristics. See Chapter 3.2 for the complete physical description.

Full series processing – training set:

- *Input matrix X* , size: $7 \times 10110 \times 1 - C \times T \times B$.
- *Output matrix T* , size: $5 \times 10110 \times 1 - C \times T \times B$.

Mini batches of 200 samples with 50% overlap – training set:

- *Input matrix X* , size: $7 \times 200 \times 100 - C \times T \times B$.
- *Output matrix T* , size: $5 \times 200 \times 100 - C \times T \times B$.

Training through 50 epochs	Full series processing $B = 0$	Mini batches $N_B = 200 = T, S_B = 100$
Time in seconds	64 s	6 s
Final loss (MSE, data & phys.), averaged across all channels	1,588	$1,507 \cdot 10^{-1}$

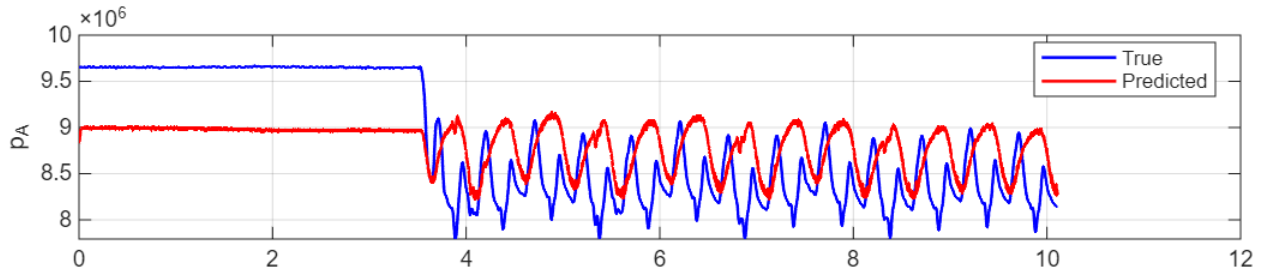


Figure 59: pressure channel reconstruction with series processing.

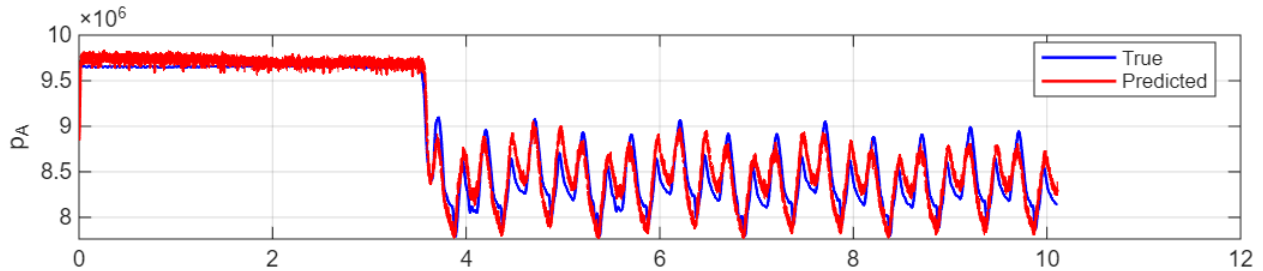


Figure 60: pressure channel reconstruction with mini batches.

When a LSTM unit is processing an extremely long time series, vanishing gradients may still cause problems. The gradient on a LSTM unit is directly multiplied by the cell states C_t . If we backpropagate it to initial timesteps, it will depend on the forget gate retain rate f_t .

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \in [0, 1]$$

Even if we consider the absurd case where each timestep the forget gate output is $f_t = 0,99$, which means 99% of the past information retained at current timestep, the gradient backpropagated through time reaching $t = 0$ would be neglectable.

$$f_t^{10110} \cong 6,5 \cdot 10^{-45}$$

TRUNCATED BPTT

The first experiment with full series processing incurred in memory constraints and the computation was significantly impacted by the data series length (time dimension T). To reduce memory footprint, some software like MATLAB, implement a common optimization for very long time series called *truncated back propagation through time* (T-BPTT).

This consists of performing the backward step of parameters update using gradient descent-based algorithms, after a fixed amount of timesteps $k < T$. After the update process, current cell and hidden states are stored and passed to the network again, to prevent any time relation from being severed.

4.2 Variable learning rate and Adam optimizer

The LSTM implemented for this thesis, uses two different optimization strategies related to the update of the network parameters:

- *Variable learning rate* reduces the hyperparameter α when the loss is approaching small numbers. In case of excessive learning rate reduction (Loss Stagnation), the algorithm intervenes to reinvigorate the learning process to prevent the gradient contribution to parameters update to vanish in the gradient descent process.
- *Adam Optimizer* is a proven effective gradient descent-based algorithm that combines two popular methods to optimize the learning process, demonstrated in research by Diederik P. Kingma and Jimmy Lei Ba, ICLR 2015 [5].

4.2.1 Variable Learning Rate

In this implementation, the raw learning rate decays with epochs with the following formulation.

$$\alpha_j = \alpha_0 \cdot \gamma^{\lfloor (ep-1)/step \rfloor}$$

The exponent is floored. Here ep represents the current epoch, $step = 200 \text{ epochs}$ is a hardcoded interval and γ is the variable learning rate decay factor.

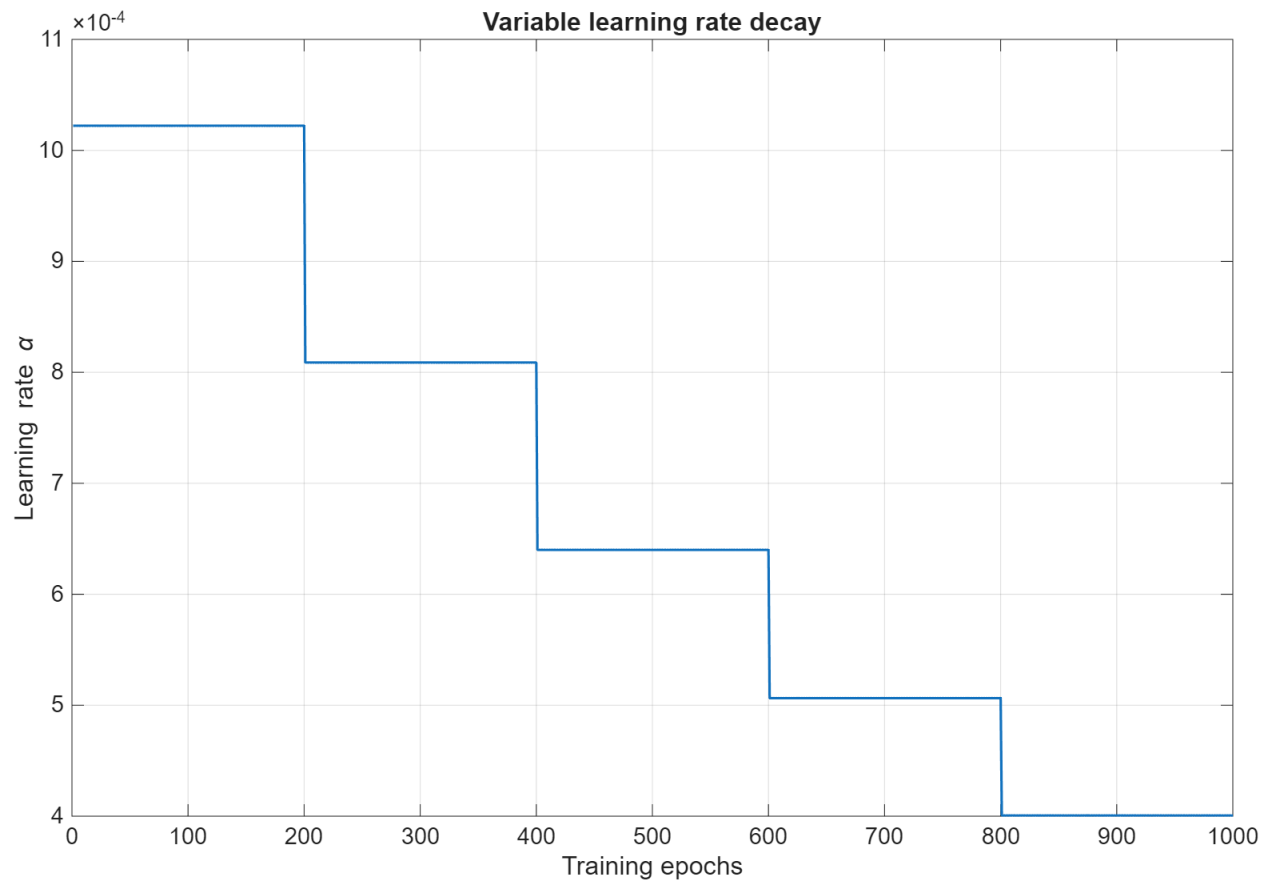


Figure 61: learning rate decay over 1000 epochs.

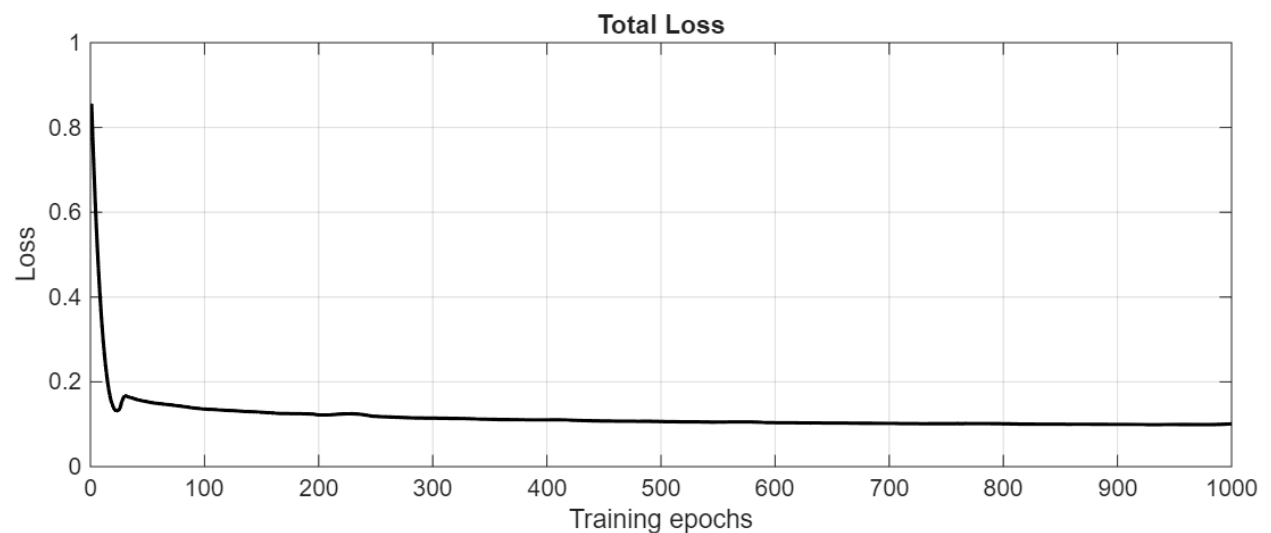


Figure 62: total loss evaluation over 1000 epochs.

In Fig. 59, the learning rate α is only decaying, highlighting a well-conditioned problem with no loss stagnation, mainly thanks to a correct mini batches implementation and optimized hyperparameters. The total loss in Fig. 60 appears stable through all the epochs.

4.2.2 Adam Optimizer – a brief introduction

This gradient descent-based algorithm used for parameters update in a backpropagation process, was first introduced in research by Diederik P. Kingma and Jimmy Lei Ba, ICLR 2015 [5]. It combines two different strategies to formulate a well optimized method that outperforms the most traditional gradient descent shown in Chapter 2.3, in (2.3.2), and most other methods.

In this paragraph we will briefly introduce the algorithm. A more complete and detailed demonstration is available for consultation on the original research article [5].

Adam births as an optimized stochastic gradient descent method that is memory efficient and derives from the adaptive moment estimation. It combines *Ada-Grad* that computes variable learning rate using squared error gradients [6], and *RMS-Prop* that uses moving averages of past error gradients [20].

The algorithm is structured as follows:

- α : step size (raw learning rate), typically set to $\alpha = 0,001$ for most experiments.
- $\beta_1, \beta_2 \in [0,1)$: exponential decay rates for moving averages.
- Compute the first order gradient and its moving average at timestep t .

$$g_t = \nabla_{\theta} f_t(\theta_{t-1})$$

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1)g_t$$

- All moving averages are initialized to 0 at $t = 0$. This will make m_t biased towards zero for the entire process. To address this issue, bias corrected terms should be computed:

$$\tilde{m}_t = \frac{m_t}{1 - \beta_1^t}$$

- Compute the second order – squared – bias corrected gradient term:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2)g_t^2$$

$$\tilde{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Update the parameter with the following rule:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\tilde{m}_t}{\sqrt{\tilde{v}_t} + \epsilon}$$

Where ϵ is a small constant (e.g. 10^{-8}) to prevent division by zero.

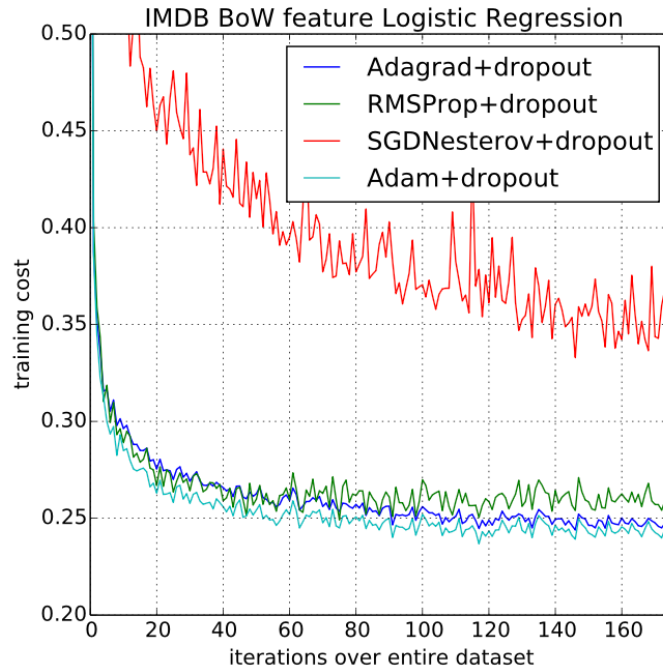


Figure 63: experiment of a NN performing logistic regression with Adam, compared to other SGD-based methods.

In this picture, a neural network is trained with a movie review dataset from IMDB website (Maas et. Al, 2011 [13]), the *Adam* optimizer resulted in faster convergence and lower training cost compared to other stochastic gradient based methods. This model is performing logistic regression to reconstruct the 10.000 most frequent words used in these movie reviews.

4.2 Bayesian optimization for hyperparameters search

A state-of-the-art approach when searching the correct combinations of hyperparameters to optimize a neural network is based on Bayesian Optimization. This method has been first introduced in research by Rasmussen and Williams, in 2006 [16], but is well demonstrated in research by Peter I. Frazier, 2018 [15], and it's based on the Bayes Rule for posterior probability.

When optimizing a model determining its hyperparameters, we need to solve the following problem:

$$\max_{(x \in A)} f(x)$$

- Where $x \in \mathbb{R}^D$ | $D \ll \infty$, meaning that x is a sufficiently small set of parameters, belonging to the network hyperparameters space and possibly a limited interval of that space.
- The function $f(x)$ is continuous and expensive to evaluate, like some function of the model loss after a sufficiently large number of epochs. Furthermore, determining a *global* maximum is difficult.

NOTE: $f(x)$ could be some function of the model loss which values near zero (of the argument, hence the loss) approach a maximum. Function $f(x)$ could depend on hyperparameters as well, this will lighten the notation, and we will refer to it as only $f(x)$. This leads to find $f(x)$ global maximum.

If we again set the $f(x)$ to be a stochastic noisy function of normal Gaussian probability distribution. This function will represent a probability density.

First, we sample the $f(x)$ at random finite collection of points $x_1, \dots, x_k \in \mathbb{R}^D$, as a result we can write the vector:

$$[f(x_1), \dots, f(x_k)]$$

This function will have a Gaussian prior distribution which is multivariate normal with a mean and a covariance matrix.

$$\mu_0(x_{1:k}) = [\mu_0(x_1), \dots, \mu_0(x_k)]$$

$$\Sigma_0(x_{1:k}, x_{1:k}) = [\Sigma_0(x_1, x_1), \dots, \Sigma_0(x_1, x_k), \dots, \Sigma_0(x_k, x_1), \dots, \Sigma_0(x_k, x_k)]$$

This means evaluating the loss, after some epochs, for several initial random sets of hyperparameters (even 1 is sufficient), hence several $k \leq n$ realizations.

Now we need to infer the value of $f(x)$ for a new point in the space x . If we have n previous realizations, the prior distribution of $f(x)$ is again normal, but now we need to compute a new possible distribution given past information – i.e. loss realization with previous hyperparameter choice.

This is done by Bayes' Rule:

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}$$

Where:

- $P(A|B)$ is the posterior probability distribution of event A, given that B has occurred.
- $P(B|A)$ is the likelihood, probability of event B, given that A has occurred.
- $P(x)$ is the prior probability of a generic event x .

This new *posterior probability distribution* will have mean and variance matrix:

$$f(x)|f(x_{1:n}) = \text{Normal}(\mu_n(x), \sigma_n^2(x))$$

Mean can be assumed as a constant value, except from some specific applications, while the variance, can be chosen amongst common versatile matrix functions, called kernels.

$$\Sigma_0(x, x') = \alpha_0 \exp(-|x - x'|^2)$$

This is the *power exponential* or *Gaussian* kernel, where:

$$\|x - x'\|^2 = \sum_{i=1}^d \alpha_i (x_i - x'_i)^2$$

Another common choice is the *Matern* kernel.

ACQUISITION FUNCTION

Before we computed the posterior distribution, a criterion should be defined on how to sample the next point in hyperparameters space x . This is not done randomly, but since Bayesian Optimization is defined as an optimized state-of-the-art search approach, greatly outperforming other search methods, the next sample point is chosen to grant the best statistic improvement in minimizing the loss function $f(x)$.

MATLAB default implementation of Bayesian Optimization Hyperparameter Search – function `bayesopt(...)` – adopts the expected improvement statistic function.

Suppose we evaluated the $f(x_n)$ at iteration n , and the largest observed value so far is:

$$f_n^* = \max_{m \leq n} f(x_m)$$

At the next evaluation we will either have f_n^* again, or $f(x)$, depending on whether the new point is able to further increase the function. We can call *improvement* the expression:

$$[f(x) - f_n^*]^+$$

Where:

$$a^+ = \max(a, 0)$$

Indicates the positive part.

The *expected improvement* can be of course defined as the expected value:

$$EI_n(x) = E_n[[f(x) - f_n^+]^+]$$

The expected improvement, for this posterior distribution, can be computed in closed form with integration by part as in research of Clark in 1961 [2]:

$$EI_n(x) = [\Delta_n(x)]^+ \sigma_n(x) \phi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right) - |\Delta_n(x)| \Phi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right)$$

This solves the problem:

$$x_{n+1} = \operatorname{argmax} EI_n(x)$$

A complete demonstration, applications and further evolutions of this method are available in Frazier et. Al 2018 research paper [15].

OPTIMIZED HYPERPARAMETERS

Applying Bayesian Optimization to this LSTM PINN implementation led to determine the optimized hyperparameters suitable for best performance and accuracy of the model.

%% Variable space for Bayesian Optimization

```
optimVars = [ ...  
    optimizableVariable('NLSTM'      ,single([1 4]),'Type','integer')  
    optimizableVariable('HiddenUnits', {'64','128','256'},'Type','categorical')  
    optimizableVariable('Bidirectional', {'true','false'})  
    optimizableVariable('Norm'       , {'none','layer'})  
    optimizableVariable('Dropout'    , single([0 0.5]))  
    optimizableVariable('LearnRate'  , single([5e-4 5e-3]), 'Transform','log')  
    optimizableVariable('Gamma'     , single([0.70 0.99]))  
    optimizableVariable('LamC0'     , single([1e-2 1e2]), 'Transform','log')];
```

Snippet 1: bayesopt hyperparameters and their search space on MATLAB.

The MATLAB implementation of Bayesian Optimization has been adopted to include more hyperparameters than what listed in Chapter 4.1, the full list is detailed below.

- *nLSTM*: is the number of LSTM layers in series, unless complex applications, the default result should be 1. To reconstruct derivatives and other complex links between variables, it is often necessary a unit number greater than 1.
- *Hiddenunits*: is the number of LSTM units, this governs the size of cell state vector, selecting multiple of 32 can improve parallelization and scaling on some modern GPU architectures equipped with dedicated Tensor Cores to accelerate matrix computations.
- *Bidirectional*: is whether the LSTM unit should be processed with reversed time vector to gain future context in supervised learning, rather than past context only, see Chapter 2.5 for LSTM introduction. Suitable for long time series or batches with several time samples.
- *Norm*: if set to *layer*, inserts a normalization layer after each LSTM timestep, standardizing the hidden state to help center its values around zero. Sometimes this cleans up the signal improving accuracy.
- *Dropout*: the dropout noise disables some hidden units to increase learning performance by reducing overfitting and improving generalization. Overfitting occurs when the model is learning noise patterns rather than actual main data structure. 0,5 means that 50% of hidden units will be disabled. This is why multiple of 64 has been chosen rather than 32.
- *LearnRate*: this is the raw learning rate α of Adam optimizer.
- *Gamma*: this is the learning rate exponential decay factor.
- *LambdaC0*: physical loss weight factor.

In this snippet, *type – categorical* means that the model choses the listed hyperparameter alternatives without interpolation.

Transform – log is used to slice the search space if it spans across several orders of magnitude. To ensure the model doesn't choose a fixed sample stepping for a hyperparameter search interval.

If an interval is defined as follows:

$$\psi \in [1 \cdot 10^{-4}, 1 \cdot 10^4]$$

And the model choses a fixed sample stepping of ~ 10 , it will sample 0,001; 10,001; ...; 90,001 ignoring orders of magnitude of $O(10^{-3})$, $O(10^{-2})$, ..., $O(10^0)$. This is called linear sampling, while in logarithmic sampling, points will be equally spaced in logarithmic domain with variable steps in linear domain.

Bayesian optimization has been run with 100 iterations, meaning that if we consider the case that the first point x in hyperparameters space is only one, $n = 100$ including the first iteration.

To evaluate a reliable loss term, with every point in hyperparameters space, this LSTM PINN model has been trained through at least 400 epochs. This number has been considered sufficient for this application as a good tradeoff between speed and final accuracy.

After the optimization process the model performs a final train of ≤ 1000 epochs.

Table 4: optimized hyperparameters via bayesopt.

Hyperparameter	Final value
NLSTM	2
Hidden Units	128
Bidirectional	false
Normalization layer	none
Dropout	14,61% (15%)
Learn rate α	$5,18 \cdot 10^{-4}$
Learn rate decay factor γ	0,9319
Physical loss weight factor λ_c	0,0102

It is worth analyzing the final architecture of this LSTM PINN.

6×1 [Layer](#) array with layers:

1	'in'	Sequence Input	Sequence input with 7 dimensions
2	'lstm1'	LSTM	LSTM with 128 hidden units
3	'drop1'	Dropout	15% dropout
4	'lstm2'	LSTM	LSTM with 128 hidden units
5	'drop2'	Dropout	15% dropout
6	'fcOut'	Fully Connected	5 fully connected layer

Snippet 2: LSTM PINN layer sequence.

- 1 – *Sequence input* layer that slices the input vector every timestep, it accepts a single point per channel, hence a vector of size 7.
- 2 – *LSTM 1* is the recurrent unit with nLSTM = 128.
- 3 – *Dropout* is a layer that inhibits some inputs using dropout noise, 15% of units are deactivated, data in this MATLAB snippet are rounded up to an integer.
- 4 – *LSTM 2* is the same architecture as 3.
- 5 – *Dropout* is another dropout layer at 15%.
- 6 – *FC Out* is a fully connected layer that transforms a multi-dimensional hidden state into the actual output.

This model is executed in parallel for every minibatch.

5 Model integration in MATLAB environment

5.1 *dlarrays* – MATLAB smart vectors

dlarrays are a common data format used in custom training loop of a Deep Neural Network.

It serves many purposes; amongst them it is an efficient way for deep learning algorithms to keep track of the Computational Graph, which is a virtual diagram that reports every operation a model performs on the vectors. It tracks their transformation process from the beginning – when they correspond to input vectors – to the network’s output.

This is the fundamental tool used by MATLAB or any other deep learning software implemented on a machine, used to perform the *back propagation*.

If a computational graph is broken at some point, meaning that a vector is being converted out from a *dlarray* format and back again – this is sufficient to interrupt the graph – the model loses track of the previous computational history, and it cannot reconstruct a valid expression for this specific vector. This will result in Automatic Differentiation (AD) being unable to compute derivatives, and the backpropagation could not be performed.

In MATLAB, *dlarrays* label each dimension:

```
for k=1:nVal
    Xv_dl{k}=dlarray(gpuArray(permute(xv{k},[2 1 3])), 'CTB');
    Tv_dl{k}=dlarray(gpuArray(permute(tv{k},[2 1 3])), 'CTB');
end
```

Snippet 3: dlarray conversion in MATLAB.

- *S*: is the spatial dimension.
- *C*: is channels dimension, a channel is a single sensor data stream.
- *B*: batch dimension. This is used when slicing an input tensor in mini batches.
- *T*: time dimension used for temporal evolution (time series).
- *U*: *unspecified dimension*, contains other dimensions with no correlation.

MATLAB will always permute a *dlarray* internally after a conversion, to match the S-C-B-T-U sequence (current release: MATLAB r2026b). When operating on these vectors, this must be considered, and a specific regard should be taken for sizes order.

GPU ARRAYS

MATLAB, and particularly the Parallel Computing Toolbox, can accelerate computations using a graphics processor that supports the CUDA language. Compatibility with other languages is not guaranteed.

In this thesis, the LSTM PINN model uses two conversions in sequence to both optimize performance and computational efficiency.

Beyond the mini batch implementation as in Chapter 4, the following has been adopted:

- FP32 floating point data format, to increase performance on some GPU architecture that accelerate matrix computation with dedicated cores.
- *gpuArray* conversion that enables vectors to be stored and computed using a graphic processor to greatly increase speed.

This implementation is not compulsory, the presence of a graphics processor is automatically detected before running a script, and if missing it enables CPU computation.

```
try
    if gpuDeviceCount == 0, error('No CUDA device detected.');
```

```
end

if ~exist('gpuIndex', 'var') || isempty(gpuIndex)
    devID = 1; % default
else
    devID = gpuIndex;
end

g = gpuDevice(devID);
reset(g);

fprintf('[GPU] %s (%.1f GB, CC %s)\n', ...
        g.Name, g.TotalMemory/1e9, g.ComputeCapability);
useGPU = true;

catch ME
    warning(sprintf('%s:GPUerr', computer('arch')), '%s - %s Using CPU.', computer, ME.message)
    useGPU = false;
end
```

Snippet 4: try - catch - me loop for GPU initialization.

5.3 Sensor data pre-processing

Data streams gathered from the transmitter-conditioner section of a servo system are inherently noisy, mainly due to electromagnetic interference. As shown in Fig. 56, if we'd zoom the reconstructed signal of a neural network, it's evident the low-quality signal to noise ratio. Feeding noisy time series to a model would result in network learning noise patterns or reconstructing wrong frequency contents.

Noise from the data is partially removed with a low-pass Butterworth filter.

5.3.1 Butterworth Filter

This transfer function is based on a specific type of polynomials first introduced by the English physician Stephen Butterworth in 1930 [17]. It is known for having a flat amplitude spectrum up to its cutoff frequency

```

[b_butter,a_butter]=butter(8,30/500,'low');           % filtro di Butterworth
for k=1:length(data)
    data{k}.SET_F=data{k}.SET_F;
    data{k}.fb_F=data{k}.fb_F*LC.K;
    data{k}.LVT=(data{k}.LVT+LVT.offset)*LVT.K;
    data{k}.pA=data{k}.pA*PT.K;
    data{k}.pB=data{k}.pB*PT.K;
    data{k}.pS=data{k}.pS*PT.K;
    data{k}.SET_EMA=data{k}.SET_EMA * 1e-3;
    data{k}.fb_EMA=data{k}.fb_EMA*EMA.K;

    % filtering
    data{k}.LVT=filtfilt(b_butter,a_butter,data{k}.LVT); % compensazione ritardo di fase di Butterworth
    data{k}.V_ref=filtfilt(b_butter,a_butter,data{k}.V_ref);
    data{k}.pA=filtfilt(b_butter,a_butter,data{k}.pA);
    data{k}.pB=filtfilt(b_butter,a_butter,data{k}.pB);
    data{k}.pS=filtfilt(b_butter,a_butter,data{k}.pS);
    data{k}.fb_EMA=filtfilt(b_butter,a_butter,data{k}.fb_EMA);

```

Snippet 5: data conversion as in Chapter 3, and Butterworth filtering.

The Butterworth polynomials are first computed in form of arrays of coefficient related to the numerator and denominator of a transfer function, later to be applied with filters.

The filter has these characteristics:

- Cutoff frequency $f_c = 30 \text{ Hz}$, in digital filters this number must be specified as a ratio between the cutoff frequency and the Nyquist factor $f_N = f_c/2 = 1000/2 = 500 \text{ Hz}$. This value is the *normalized frequency*:

$$\frac{f_c}{f_N/2} = 0,6\pi \frac{\text{rad}}{\text{sample}}$$

- 8th degree Butterworth polynomials at numerator and denominator.
- The parameter 'low' is used to build a lowpass filter configuration.

Data have been filtered with the *filtfilt* function in MATLAB to recover the phase delay of this transfer function, increasing denoising effectiveness.

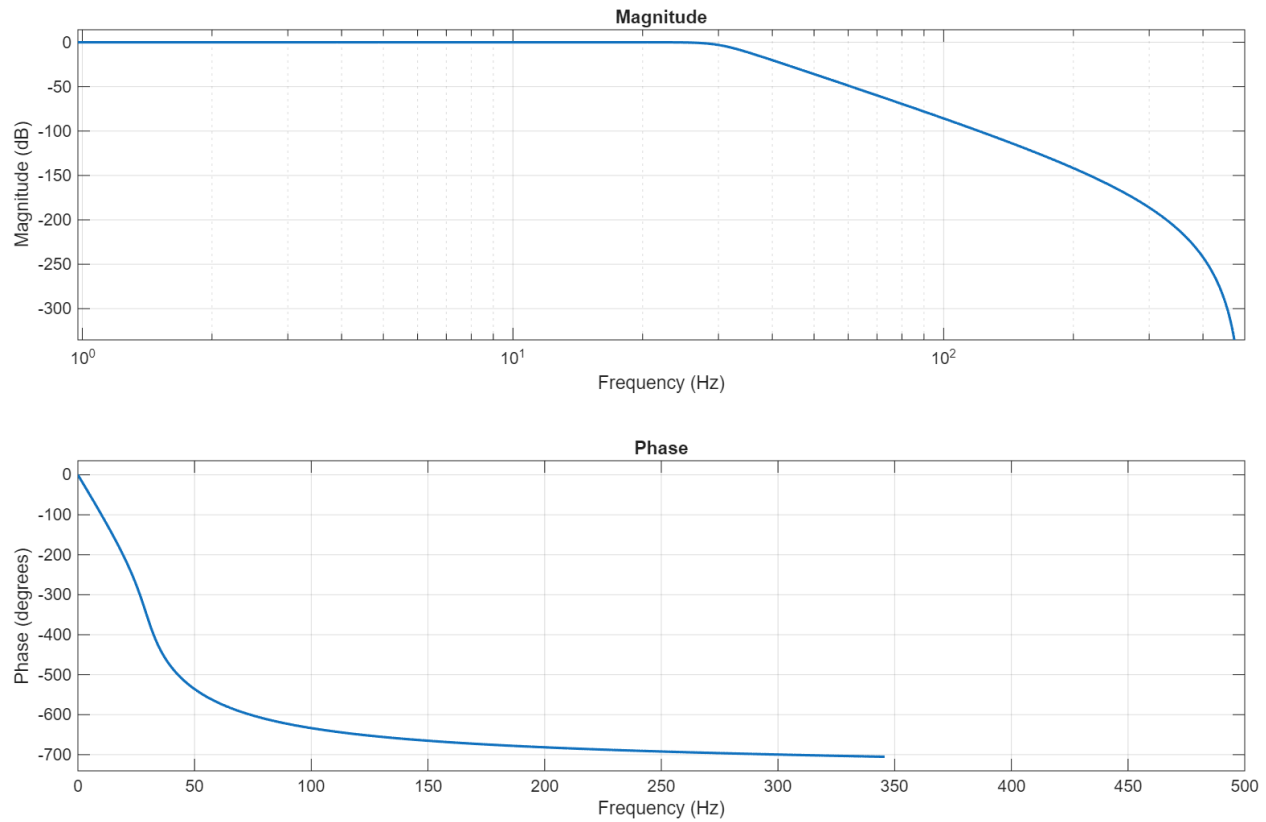


Figure 64: Bode plot of this Butterworth filter.

5.3.2 Data sorting and normalization

After the filtering pass, data is standardized.

```
% Normalisation
xt_flat = vertcat(xt{:});
xv_flat = vertcat(xv{:});
tt_flat = vertcat(tt{:});
tv_flat = vertcat(tv{:});

phys.mu_in = mean([xt_flat; xv_flat], 1);
phys.mu_out = mean([tt_flat; tv_flat], 1);

phys.sig_in = max(std([xt_flat; xv_flat],0,1), 1e-9);
phys.sig_out = max(std([tt_flat; tv_flat],0,1), 1e-9);

for k=1:nTrain
    xt{k} = (xt{k} - phys.mu_in) ./ phys.sig_in;
    tt{k} = (tt{k} - phys.mu_out) ./ phys.sig_out;
end

for k=1:nVal
    xv{k} = (xv{k} - phys.mu_in) ./ phys.sig_in;
    tv{k} = (tv{k} - phys.mu_out) ./ phys.sig_out;
end
```

Snippet 6: data standardization.

This means that a time series is transformed to center all its points around zero by subtracting the mean and dividing them by the standard deviation.

$$\tilde{x}(t) = \frac{x(t) - \mu_t}{\sigma_t}$$

The advantage, as mentioned in Chapter 1, is being able to compare different physical units and values bringing all of them to the same order of magnitude. This will solve imbalances in the numerical data driven loss. To compute a physical law, data must be un-normalized as shown in Chapter 3.4.

The training dataset is obtained from 60% of total experiments, the rest will compose the validation dataset. All experiments are shuffled every time the preprocessing script is executed.

Normalized vectors are then converted to dlarrays as shown in Chapter 5.1 and exported to an external variable, to be processed by both bayesopt (Chapter 4.2) and training scripts.

5.3.3 Training loop

```
load('HLA_PINN_best_20260604_155512.mat','bestH'); % |||| Change accordingly to the last filename ||||

h = table2struct(bestH);

numLayers = h.NLSTM;
hiddenUnits = str2double(string(h.HiddenUnits));
dropoutRate = h.Dropout;
bidirectional = string(h.Bidirectional) == "true";

layers = [sequenceInputLayer(N.nIn,'Name','input')];
for i = 1:numLayers
    if bidirectional
        layers = [layers; bilstmLayer(hiddenUnits,'OutputMode','sequence','Name',sprintf('bilstm%d',i))];
    else
        layers = [layers; lstmLayer(hiddenUnits,'OutputMode','sequence','Name',sprintf('lstm%d',i))];
    end
    layers = [layers; dropoutLayer(dropoutRate,'Name',sprintf('drop%d',i))];
end
layers = [layers; fullyConnectedLayer(N.nOut,'Name','fc')];

net = dlnetwork(layerGraph(layers));
```

Snippet 7: hyperparameters loading and model building.

Optimized hyperparameters (Table 4, chapter 4.2) are loaded in the training loop, the LSTM network architecture is built, and the model is ready for training. The MATLAB variable *net* contains the entire architecture, and it's used as an argument for functions performing forward and backward steps in the training process.

```

lr      = h.LearnRate;
gamma   = h.Gamma;
logLamC = dlarray(log(h.LamC0));
avg_l=0; avgSq_l=0;
avg = []; avgSq = [];

decayPeriod = 200; % def = 200
epochs = 1000;      % def = 2500
printEvery = 50;

lossLog = zeros(epochs,3); gradNorm = zeros(epochs,1);

```

Snippet 8: training parameters.

The model is trained for 1000 epochs to account for the slow convergence speed of some outputs, due to small orders of magnitude in the physical law. The LVT sensor returns velocity readings that are $O(10^{-1})$ m/s, very different from a pressure signal having $O(10^{+9})$ Pa. The backpropagation gradients derived from the physical loss term will be small and so the convergence speed.

```

mon = trainingProgressMonitor("XLabel","Epoch","Metrics",["Total","Data","Cont"],"Visible","on");
fprintf('[TRAINING] Starting...\n');
lr_report=zeros(1,epochs);
for ep = 1:epochs
    lr_step = lr * gamma^floor((ep-1)/decayPeriod);
    [Lt,Ln,Lc,gr_net,lamC] = dlfeval(@modelLoss_PINN_LSTM, net, Xt_dl,Tt_dl,phys,logLamC, names);

    % Gradient Clipping
    gr_net = dlupdate(@(g) max(min(g,1),-1), gr_net);

    [net,avg,avgSq] = adamupdate(net,gr_net,avg,avgSq,ep,lr);

    lossLog(ep,:) = gather([Lt Ln Lc]);
    gradNorm(ep) = sqrt(sum(arrayfun(@(i) sum(extractdata(gr_net.Value{i}).^2, 'all'), 1:height(gr_net))));

    recordMetrics(mon, ep, "Total",lossLog(ep,1), "Data",lossLog(ep,2), "Cont", lossLog(ep,3));

    if mod(ep,printEvery)==0 || ep==epochs
        fprintf('ep %4d | Ltot %.3e | Ln %.3e | Lc %.3e | λC %.2e\n', ...
            ep,lossLog(ep,1),lossLog(ep,2),lossLog(ep,3),lamC);
    end
    if ~isValid(mon), break; end
end

```

Snippet 9: the main custom training loop.

This is a fully customized training loop using an external loss function, later detailed, and *Adam* as the gradient descent optimizer.

The function *dlfeval()* performs the custom loss function evaluation, represented with the handle *@modelLoss_PINN*, using the network *net*, *Xt dl* and *Tt dl* that are training input and target dlarrays, a struct containing all physical constants *phys* and the physical loss weight hyperparameter *logLamC*.

This last hyperparameter is transformed to enable logarithmic sampling for future implementations that may require Adam to have direct control on that hyperparameter during the backward step. In this case the weight is fixed after bayesopt.

GRADIENT CLIPPING

This passage is performed by the function:

$$\nabla_{net} = dlupdate(@g) \max(\min(g, 1), -1), \nabla_{net})$$

That clips the gradients of data driven and physical loss, all contained in ∇_{net} , between -1 and +1.

Exploding gradients are a common phenomenon in the training process of a physics informed neural network (PINN). During the backpropagation, the model goes through transient phases where not all the losses could be satisfied this is mostly caused by the physical loss that may be exploding to infinity. Clipping is a simple, yet effective strategy that slows the learning and parameters update but stabilizes it in the long term.

Another strategy, adopted in this thesis, to avoid exploding gradients, is the subdivision of the training dataset in mini batches. This significantly simplifies the learning since the time series is shortened, increasing convergence speed and accuracy while reducing the probability that a gradient could explode. In Fig. 62 of Chapter 4.2.1 the total loss term is stable and constantly reducing, if we'd report normalized gradients in log scale, we will notice they never explode, nor trigger clipping.

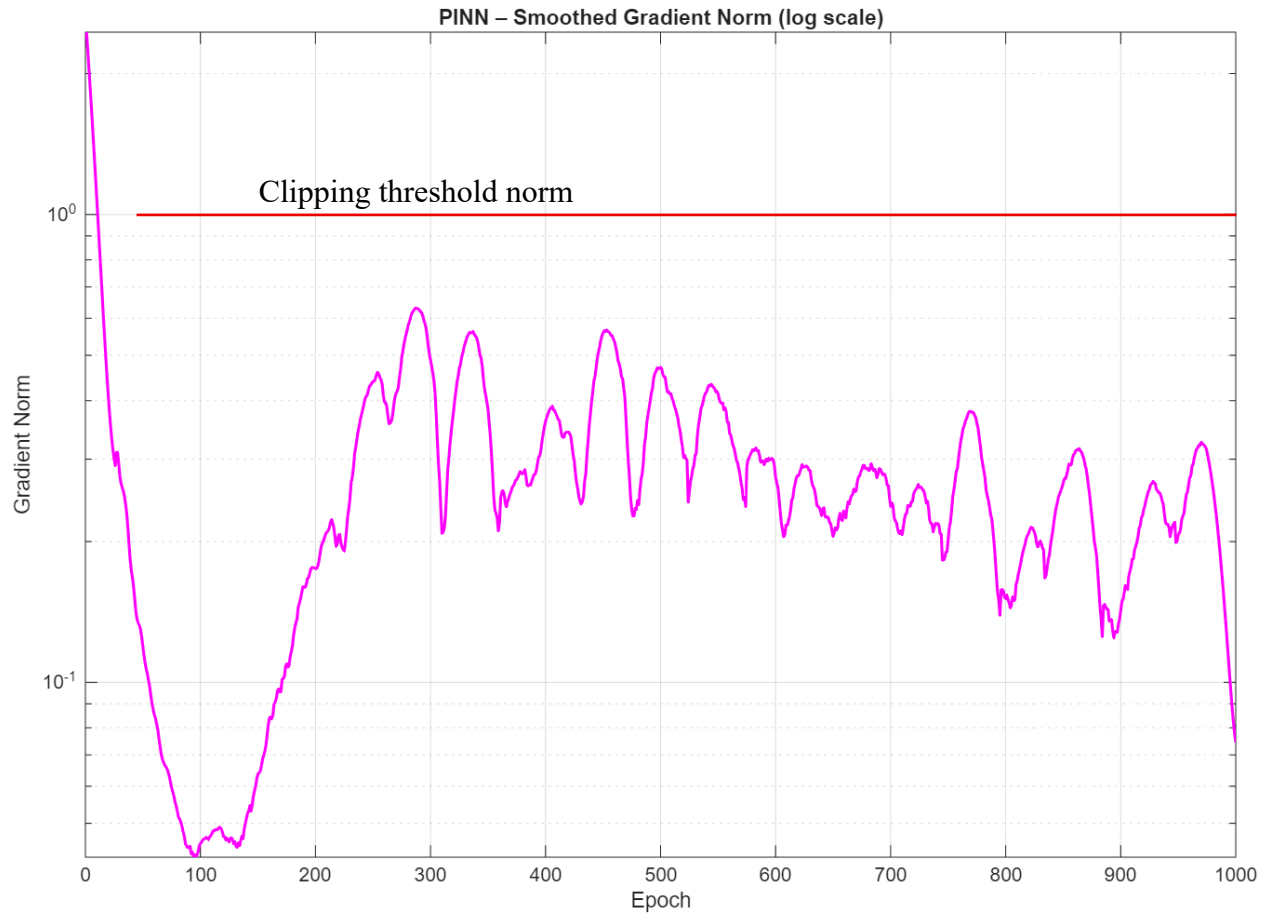


Figure 65: absolute value of gradients in log scale.

5.3.3 Loss Function

The loss function is divided in two parts, *modelLoss_PINN* is a local function computing the final contribution:

$$L = L_n + \lambda_c L_c$$

And the loss gradients with respect to network parameters *net.Learnables*:

$$\nabla_{net} = \frac{dL}{d\theta}$$

The single loss terms are computed by *PINN_losses_V4*, an external function determining numeric and physical law mean squared errors.

```
% Model Loss Function
function [Lt,Ln,Lc,gr_net,lamC] = modelLoss_PINN_LSTM(net,X,T,phys,logLamC,names)

    lamC = exp(logLamC);

    Yp = forward(net,X);
    [Ln,Lc] = PINN_Losses_V4(Yp,X,T,phys,names);
    Lt = Ln + lamC*Lc; % loss : data + phys
    [gr_net] = dlgradient(Lt, net.Learnables);
end
```

Snippet 10: internal loss function.

In the external loss function, all the necessary calculation for the numerical data drive loss and the physical law have been implemented. As mentioned in Chapter 3.3, a physics informed approach consists of linking the network predicted variables – outputs $y(t)$ – together with a physical law that must be satisfied. Not respecting this relation will result in a loss term.

Hence, in this physical law, the target variables of the supervised learning must not enter computation.

```

% Extraction -----
Pscale = single(3e7);
Cd_bp = phys.Cd_bp; Ao_bp = phys.Ao_bp;
beta = phys.beta*0.75; A = phys.A;
% sig_t = phys.sig_in(1);

b_d = Yp(idx_out('LVT'),:,:);
b = Yp(idx_out('fb_EMA'),:,:);
pA = Yp(idx_out('pA'),:,:);
pB = Yp(idx_out('pB'),:,:);
pS = Xdl(idx_in('pS'),:,:);
true_pA = TARG(idx_out('pA'),:,:);
true_pB = TARG(idx_out('pB'),:,:);

% Continuity Loss - Integral form
% Un-normalisation -----
b_d = unN_out(b_d,idx_out('LVT'));
b = unN_out(b,idx_out('fb_EMA')).*phys.i_trans;
pA = unN_out(pA,idx_out('pA'));
pB = unN_out(pB,idx_out('pB'));
pS = unN_in(pS,idx_in('pS'));
true_pA = unN_out(true_pA,idx_out('pA'));
true_pB = unN_out(true_pB,idx_out('pB'));
pT = single(0.1e+6); % Valve return pressure (T port)
epsilon = single(1e-9); % Small constant

```

Snippet 11: data extraction and un-normalization.

Data is first extracted to single dlarrays lighten the notation, then normalized and un-normalized vectors are computed.

In the first row, the normalizing pressure $p_{scale} = 3 \cdot 10^7 \text{ Pa}$ is fixed as in Chapter 3.4, along with the returning pressure at port T in the servo valve and the small constant ϵ to avoid divisions by zero when computing the bypass flow rate.

```

% Spool position from virtual sensor
Xv = Xdl(idx_in('Spool'),:,:);
Xv = unN_in(Xv,idx_in('Spool'))*(-1); % flip the sign

h_sv = phys.hsv; w_sv = phys.wsv; eps1 = phys.eps1;
rho = phys.rho; Cdv = phys.Cdv;
Av = sqrt((w_sv*(abs(Xv)-eps1)).^2+(w_sv*h_sv).^2); % SV cross section
Rav = rho./(2*(Cdv*Av).^2); % Turbulent resistance;

% Flow rates - assembling vectors to avoid if{} cycle:
Qload = @(x) sqrt(abs(pS-x)./(2*Rav));
Qdisc = @(y) sqrt(abs(y-pT)./(2*Rav));

mark = max(0, sign(Xv)); % Flow rate depends on spool position - [+] = LOAD; [-] = UNLOAD
Q1 = Qload(pA).*mark;
Q1 = Q1 - abs(mark-1).*Qdisc(pA);

Q2 = Qload(pB).*abs(mark-1);
Q2 = Q2 - Qdisc(pB).*mark;
Qbp = Cd_bp*Ao_bp .* sqrt((2*abs(pA-pB)/rho) + epsilon) .* sign(pA-pB);
Q1 = single(0);
% Q1 = 0.2*Qbp;
V1 = phys.V1; V2=phys.V2;

```

Snippet 12: flow rates through the servo valve.

In the first group of rows, the spool position is extracted from training dataset and un-normalized. The second group extract valve geometry physics constants and the Wheatstone circuit fluid turbulent resistance as in Chapter 3.

After flow rate computation in load and unload configurations, the actual signed flow rate vectors are built based on spool position. The code follows this order:

- $sign(x_v)$ is computed, the spool position sign is flipped to match load configuration in Chapter 3.
- The filtering vector $mark = \max(sign(x_v), 0) = sign(x_v)^+$ is used to filter out the unloading configuration of a chamber.
- The loading flow rate in a chamber is computed:

$$Q_A = Q_1 = \langle \sqrt{\frac{|p_S - p_A|}{2R_a}}, mark \rangle$$

Where $\langle a, b \rangle$ represent the inner product of vectors **a** and **b**.

- Filter is then reversed to fill the **Q** vector with unloading flow rates.

This algorithm avoids using *if* cycles, that could slow down the code.

In the final group of rows, the bypass flow rate is also computed.

The leaking oil across chambers have been initialized to zero; as stated amongst the hypothesis of this work, is stated that we only have approximate knowledge of this system, a leaking flow rate across chambers of the HLA should be considered as a finer detail, complex to determine or measure with decent accuracy. Since we are interested in rougher approximation of this servo valve, and the physical loss to be a general, yet accurate, representation of the working principle of this component, with a black box approach.

```
% Only flow rates have implicit sign
rhsA = beta./(V1 + A*b).*(Q1 + A*b_d - Q1 - Qbp);
rhsB = beta./(V2 - A*b).*(-A*b_d + Q2 + Q1 + Qbp);

pA_unfmt = stripdims(pA);
pB_unfmt = stripdims(pB);
true_pA_unfmt = stripdims(true_pA);
true_pB_unfmt = stripdims(true_pB);

rhsA_unfmt = stripdims(rhsA);
rhsB_unfmt = stripdims(rhsB);

W = single(size(rhsA_unfmt,3));
U_d1 = dldarray(triu(ones(W, W, 'single')));

% Permute RHS to [Channel, Time, Batch] for pagemtimes
rhsA_CTB = permute(rhsA_unfmt, [1, 3, 2]);
rhsB_CTB = permute(rhsB_unfmt, [1, 3, 2]);
```

Snippet 13: compressibility law terms are prepared for integration.

In this code section the right-hand side of the compressibility law is computed. For example, in chamber j the equation is:

$$\frac{dp}{dt} = \frac{Q_j + A_C \cdot LVT - Q_{leak} - Q_{bp}}{V_{j0} - A_C \cdot b}$$

Mass flow rates sign convention is:

- $Q > 0$ if a chamber is loading.
- $Q < 0$ if it's unloading.

Signs remain coherent in all physical terms implemented in this function.

The integration of this equation leads to different terms:

$$\int_{t=0}^t \frac{dp}{dt} = p(t) - p(0)$$

Which is the pressure time vector $p(t)$ minus the initial conditions $p(0)$ for the considered mini batch.

The right-hand side must be integrated in time from $t = 0$ to a variable extreme, to constrain the problem to every timestep (Chapter 3.4). Since timestep are a discrete quantity, this term must be written inside a discrete integral.

MATLAB dlarrays does not currently support any direct function that performs a cumulative sum over their components (current release: MATLAB r2026b).

To do so, a manual process must be implemented. In Snippet 13, the summation is performed by multiplying the array with an upper triangular matrix of 1.

Let $x_t \in \mathbb{R}^T$ be a generic time series array T long, and $U_T \in \mathbb{R}^{T \times T}$ an upper triangular matrix. The discrete integration function could be written as a single array mapping each component as follows:

$$x_t = \begin{bmatrix} x(t_1) \\ \dots \\ x(T) \end{bmatrix}$$

$$U_T = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & 1 & \dots & 1 \\ 0 & 0 & 0 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

By multiplication and times dt , we obtain:

$$F_t = dt \cdot U_T \cdot x_t = \begin{bmatrix} x(t_1) \\ x(t_1) + x(t_2) \\ \dots \\ x(t_1) + \dots + x(T) \end{bmatrix} dt$$

By scrolling through the entries of F_t , we are evaluating the discrete integral function at every timestep.

This procedure is performed every batch, since an input tensor is $x_t \in \mathbb{R}^{C \times B \times T}$, integration should be performed along the C and T dimension, iterating every $b < B$. In other words, we are slicing the tensor in *pages* and integrating inside this portion.

MATLAB has implemented a built-in function to perform *page-wise matrix multiplication* called `pagetimes()`. Inputs are the right-hand side integrand term times dt , and the upper triangular integration matrix.

```

% Perform the discrete integral
% pagetimes multiplies the [1 x 200] Time vectors by the [200 x 200] matrix U
% for every batch simultaneously
dt = 1 / phys.Fc;
integral_A_CTB = pagetimes(rhsA_CTB * dt, U_dl);
integral_B_CTB = pagetimes(rhsB_CTB * dt, U_dl);

% Back to CBT
integral_A = permute(integral_A_CTB, [1, 3, 2]);
integral_B = permute(integral_B_CTB, [1, 3, 2]);

% Initial condition of the integral (every batch)
pA_zero = true_pA_unfmt(1,:,1);
pB_zero = true_pB_unfmt(1,:,1);

% Physical residuals
int_rhsA = (pA_unfmt - pA_zero - integral_A) / (Pscale);
int_rhsB = (pB_unfmt - pB_zero - integral_B) / (Pscale);

% Losses
Ln = mean((Yp - TARG).^2, "all");
Lc = mean(int_rhsA.^2 + int_rhsB.^2, 'all');

```

Snippet 14: integral computation and Lmse losses.

Then, every residual r_j for the j -th chamber is computed according to what shown in Chapter 3.4, and the physical mean squared error – normalized by p_{scale} – is obtained as shown in Snippet 14.

At this point, the data driven loss is also computed with normalized arrays and the two are fed back into the training loop, and to *Adam* optimizer.

5.4 Training data analysis

The complete filtered training dataset is plotted in the following diagram, for a single training experiment.

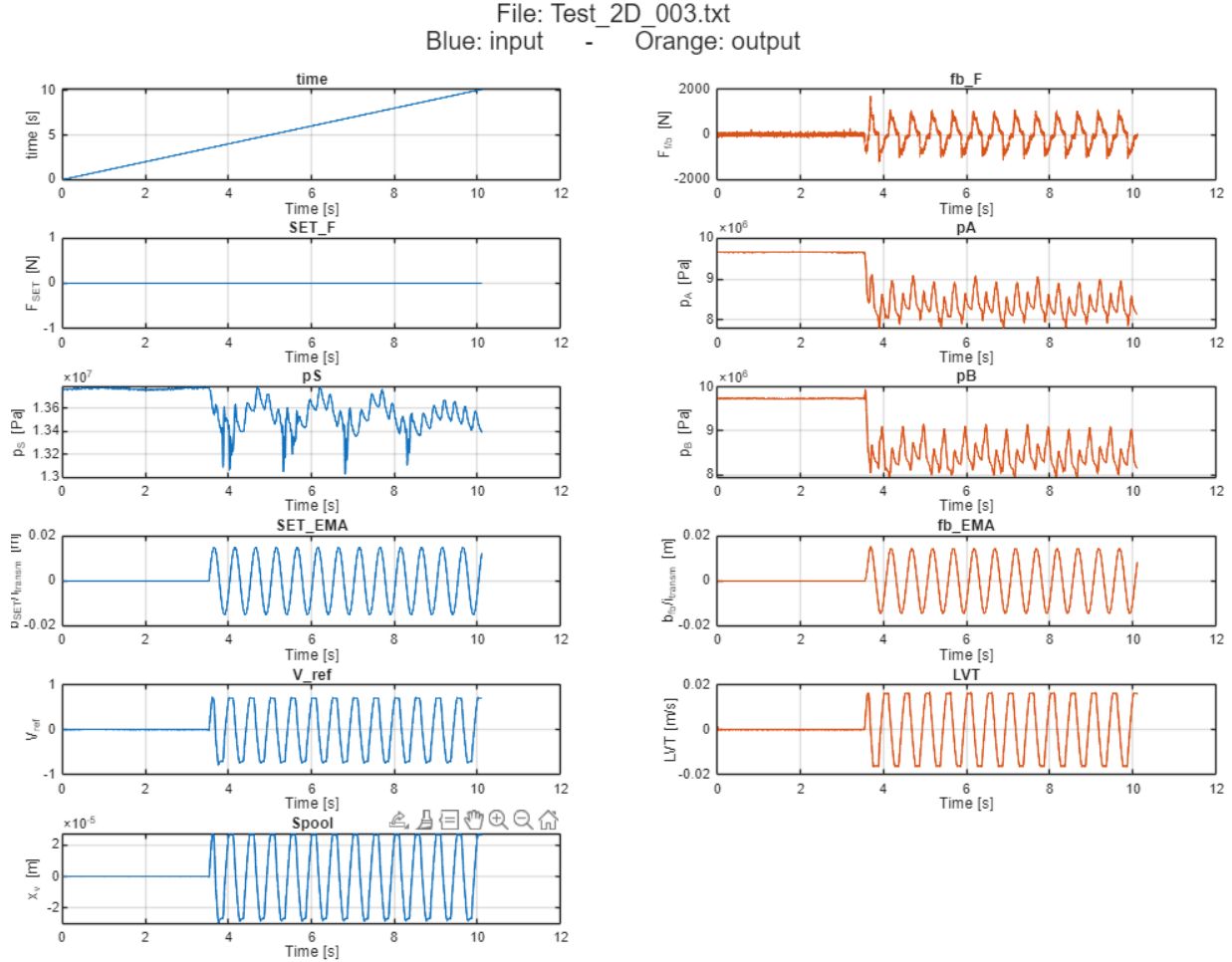


Figure 66: full plot of the training dataset, divided in inputs and outputs.

On the left, blue plots indicate the input dataset of the LSTM PINN model, while on the right, the 5 output channels are plotted.

It's immediately evident that the $F_{f/b}$ greatly differs from F_{set} . In this force controlled hydraulic servo system, since velocity is a disturbance, the EMA actuator – position controlled – is actively disturbing the HLA. The entire servo system is responding to a sudden change in feedback force, generated by the EMA change in position, with a slight phase delay. This is enough to allow $F_{f/b}$ to rise well above 1000 N. Furthermore, spikes in pressure readings in chambers A and B confirm this phenomenon.

5.5 Result analysis and limitations

When training this model on a more specific dataset containing experiments with sinusoidal EMA position and $F_{set} = 0\text{ N}$, results are very promising. The following three diagrams will report 2 significant models of the training set, then a validation experiment.

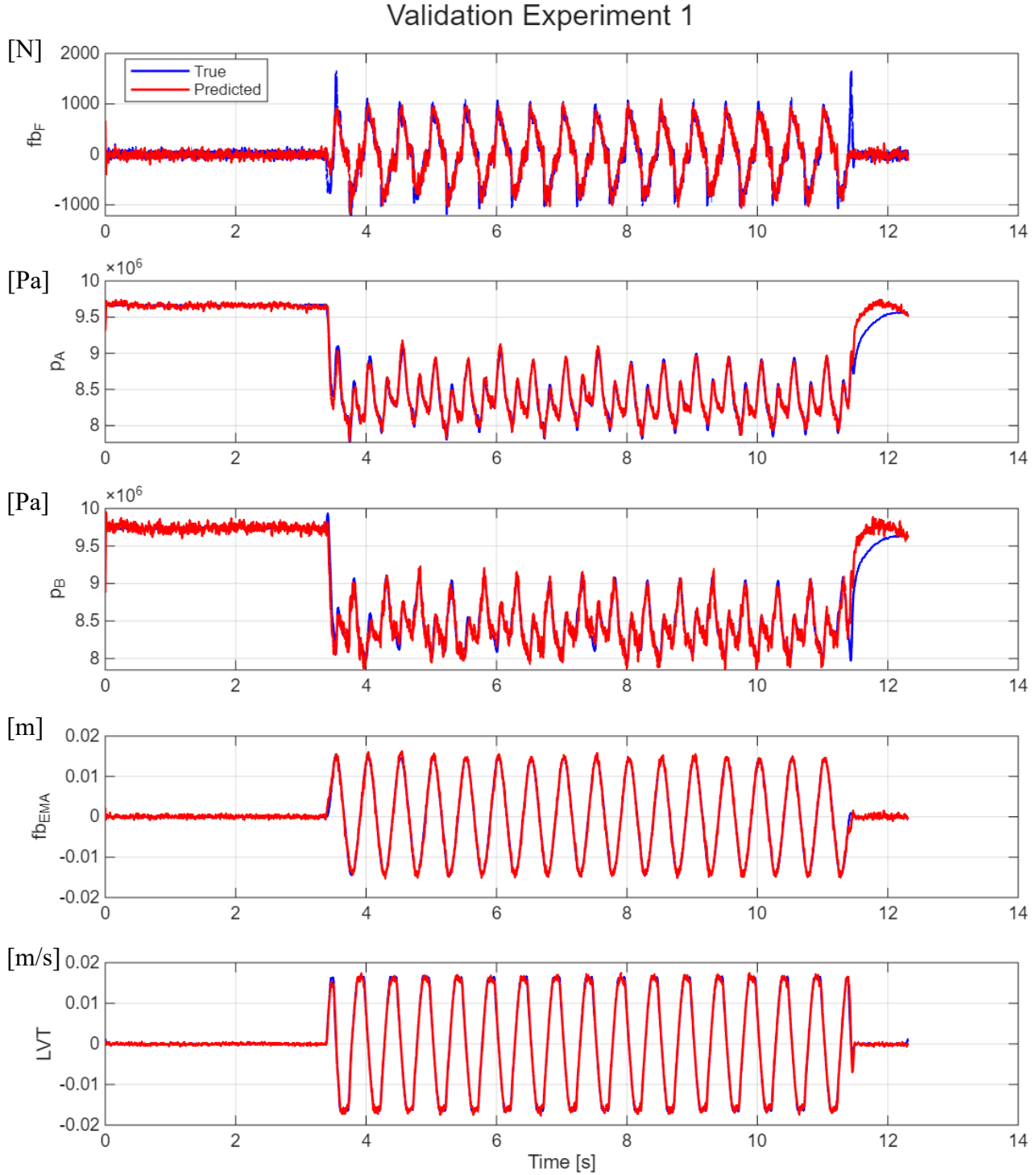


Figure 67: validation experiment.

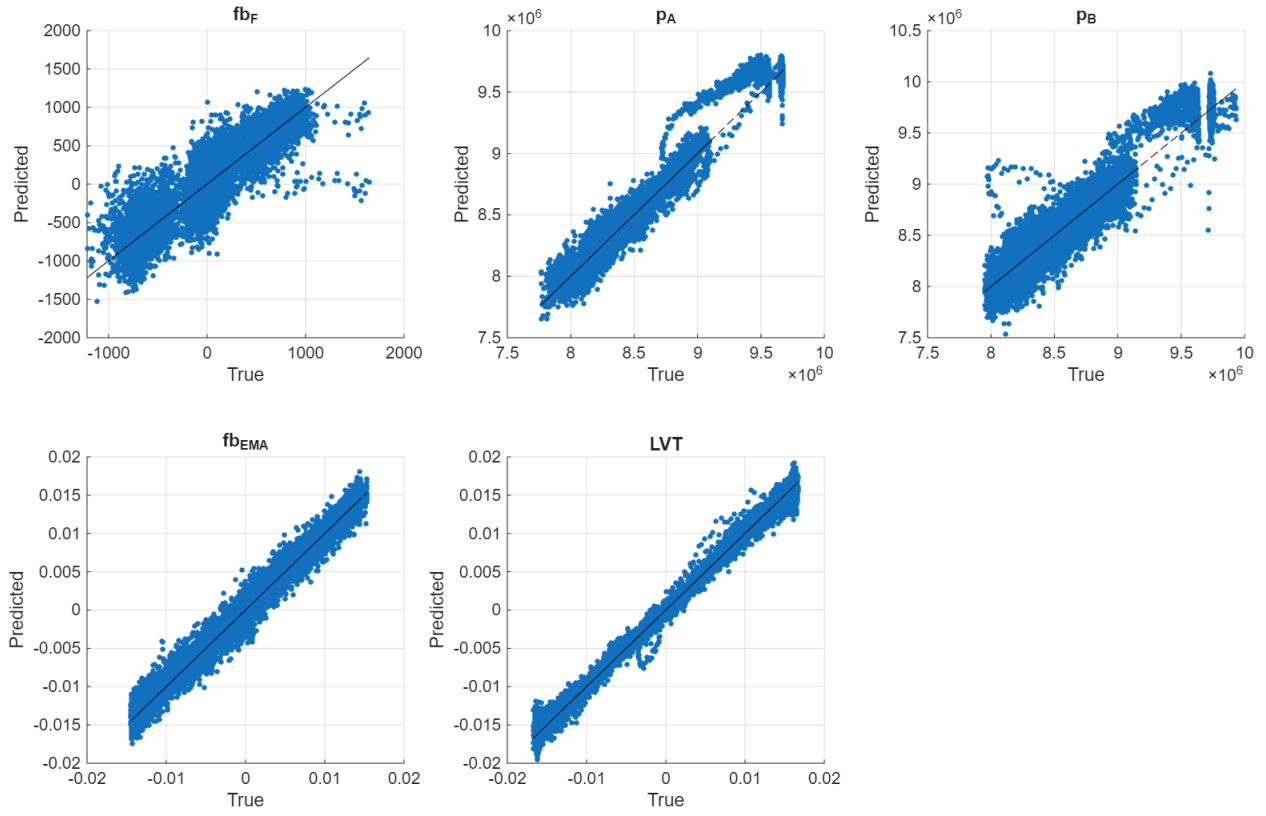


Figure 68: scatter plot on validation experiment.

We can conclude that if we select an adequate training set, that could be accurately described with the compressibility law, this LSTM PINN returns satisfying results.

This validation set has some similarities with others in the training set.

When training this model on a more generalized dataset composed of 10 experiments of the ASTIB test bench with many different set and feedback signals of bot force and position (HLA and EMA commands), results are inaccurate specifically for chamber pressure p_A and p_B .

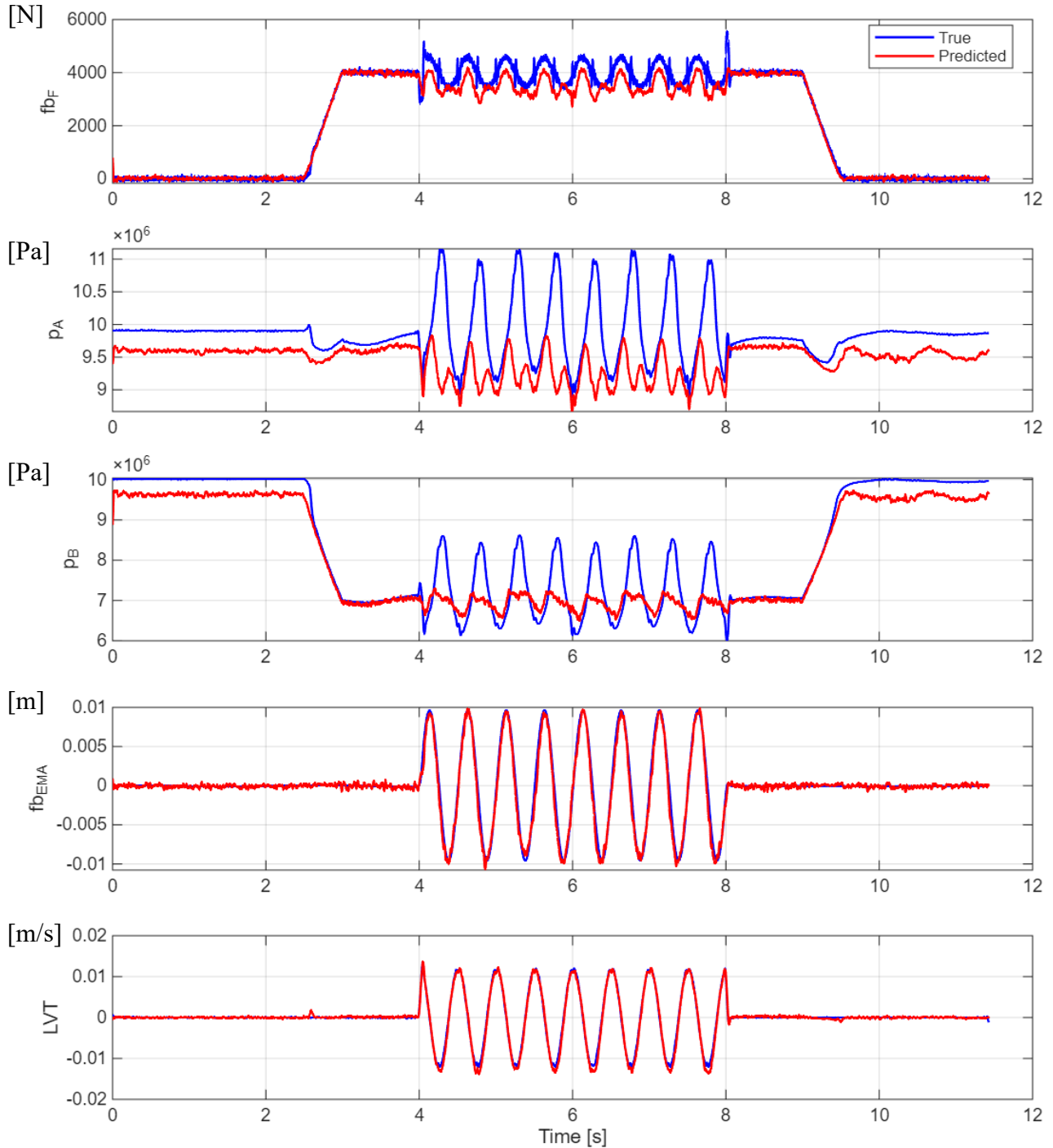


Figure 69: validation experiment with a general dataset.

This partial fitting is due to the physical law being insufficient to describe the characteristics of this servo system. A more specific physical description is necessary to enhance the model accuracy when inferencing the pressure in each chamber.

This model has implemented a link between chamber pressures p_A and p_B , the *LVT* sensor measuring piston velocity, the feedback from the EMA actuator b , and indirect link to the spool position through mass flow rates. There is no information on mechanical quantities governing the force balance on the piston.

Fitting on the force feedback F_f/b is accurate, but this quantity does not appear in any implemented physical law, so the model acts as a *curve fitter* for this quantity.

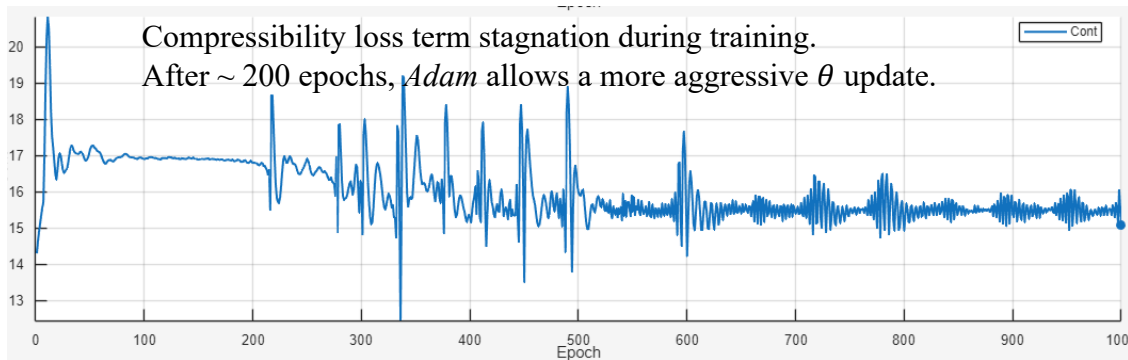


Figure 70: physical loss stabilizing to a small value.

The physical loss retains a residual stagnation, even after 1000 epochs, suggesting a slight inaccuracy in the physical description of this servo system. After a while, *Adam* optimizer intervenes to reinvigorate learning by raising α , but this does not completely solve the long-term loss stagnation. This could be due to the physical law being either slightly inaccurate or insufficient to completely describe the servo system.

6 Conclusions

6.1 Final considerations

This thesis demonstrates that it is possible to model a Digital Twin of a component in a complex servo system, under certain conditions. When the compressibility law starts to show its limits, regression will be inaccurate because this servo system needs additional information about the core physics.

Additionally, it has been discovered that when a validation set is composed of several experiments that have nothing in common with the training set, even the data driven approach – LSTM with pure numerical L_{mse} loss – is prone to failure.

Considering the remaining successful cases, when an accurate regression is performed by the LSTM PINN, this approach offers numerous advantages. First of them, a forward step of a neural network could be executed almost in real time, meaning that the time of computation required by such a model, can be considered neglectable when compared to a more standard simulation strategy that use e.g. ODEs and a ODE45 solving algorithm. The next table compares times of execution of a digital twin of the ASTIB test bench modelled in Simulink, with this LSTM PINN.

LSTM PINN simulation time – forward step	Simulink computation time – ODEs solution
0,45 s	11,4 s
(parallel computation – GPU accelerated)	(series computation)

This is an unfair comparison, but it helps to visualize the main advantage of a trained neural network in servo system simulation.

6.2 Future work

The first hypothesis of this work was to consider an approximate knowledge of a servo system and try to construct a digital twin of one single component using a general physical law that could govern its working principle. This was the main obstacle that made a more traditional simulation mean ineffective.

Considering the results obtained, we can conclude that this cannot be entirely demonstrated. A physics informed neural network purpose, is to output a system response that complies to strict requirements of a physical law. The core physics that govern a complex system like the ASTIB test bench, should not be partially implemented, nor should be taken a 1st order approximation, leaving to data driven part of a neural network the role of emulating the remaining physics, that hasn't been implemented.

If we consider a case where a complete and detailed physical description has been implemented, this model could be integrated in many environments. In research by Taimun et. Al (2026) [19], such a model could easily solve a *State synchronization problem*, where a response should be computed in real time and compared to the current outputs from sensors on the real machine.

References

- 1 A. Jardine, D. Lin, and D. Banjevic, “A review on machinery diagnostics and prognostics implementing condition-based maintenance,” *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, 2006 doi: 10.1016/j.ymssp.2005.09.012
- 2 Clark, C. E. (1961). The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162. <http://links.jstor.org/sici?sici=0030-364X%28196103%2F04%299%3A2%3C145%3ATGOAFS%3E2.0.CO%3B2-U>
- 3 Coulibaly, Paulin & Anctil, François & Aravena, Ramón & Bobée, Bernard. (2001). Artificial Neural Network Modeling of Water Table Depth Fluctuations. *Water Resources Research - WATER RESOUR RES.* 37. 885-896. <https://doi.org/10.1029/2000WR900368>
- 4 De Martin, Andrea & Chiavaroli, Piergiorgio & Evangelista, Giuseppe & Jacazio, Giovanni & Sorli, Massimo. (2018). Real Time Loading Test Rig for Flight Control Actuators Under PHM Experimentation. 10.1115/IMECE2018-86967
- 5 Diederik P. Kingma, & Jimmy Ba. (2017). Adam: A Method for Stochastic Optimization. <https://doi.org/10.48550/arXiv.1412.6980>
- 6 Duchi, John & Hazan, Elad & Singer, Yoram. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research.* 12. 2121-2159. https://www.researchgate.net/publication/220320677_Adaptive_Subgradient_Methods_for_Online_Learning_and_Stochastic_Optimization
- 7 Elman, J. L., Finding structure in time, April – June 1990, *Cognitive Science*, 14, 179-211. [https://doi.org/10.1016/0364-0213\(90\)90002-E](https://doi.org/10.1016/0364-0213(90)90002-E)
- 8 Google, "Linear regression: Gradient descent," *Machine Learning Crash Course*. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/linear-regression/gradient-descent> [Accessed: June 4, 2026]
- 9 Greff, Klaus et al. *Transactions on Neural Networks and Learning Systems*. IEEE, 10 Oct 2017. “LSTM: A Search Space Odyssey”. <https://arxiv.org/pdf/1503.04069>
- 10 Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-Term Memory. *Neural Computation.* 9. 1735-1780. 10.1162/neco.1997.9.8.1735

- 11 Jean-Charles MARÉ, Jian FU, "Review on signal-by-wire and power-by-wire actuation for more electric aircraft", Chinese Journal of Aeronautics, Volume 30, Issue 3, 2017, Pages 857-870, ISSN 1000-9361, <https://doi.org/10.1016/j.cja.2017.03.013>
- 12 Liu, Jian-wei & Xu, Bing-rong & Song, Zhi-yan. (2025). A Survey of Recursive and Recurrent Neural Networks. 10.48550/arXiv.2510.17867
- 13 Maas, Andrew & Daly, Raymond & Pham, Peter & Huang, Dan & Ng, Andrew & Potts, Christopher. (2011). Learning Word Vectors for Sentiment Analysis. 142-150.
- 14 Olah, Christopher. "Understanding LSTM Networks". Colah's Blog, 27 Aug 2015. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- 15 Peter I. Frazier, "A Tutorial on Bayesian Optimization," 2018. <https://arxiv.org/abs/1807.02811>
- 16 Rasmussen, Carl & Williams, Christopher. (2005). Gaussian Processes for Machine Learning. 10.7551/mitpress/3206.001.0001
- 17 S. Butterworth. "On the theory of filter amplifiers". October 1930.
- 18 Stiles LR, Freisner AL, Mayo J, Landis KH, Kothmann BD. Impossible to resist: the development of rotorcraft Fly-by-Wire technology. American helicopter society international 60th annual forum; 2004 June 7–10; Baltimore, USA; 2004.
- 19 Taimun, Md & Alam, Md & Fareed, Sheikh Muhammad. (2026). Digital Twin-Enabled Predictive Maintenance for Textile and Mechanical Systems. World Journal of Advanced Engineering Technology and Sciences. 18. 187-203. 10.30574/wjaets.2026.18.1.0001
- 20 Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012
- 21 Tim. Parallel For All. NVIDIA. "Deep Learning in a Nutshell: Sequence Learning" Dettmers. 7 Mar 2016.
- 22 United States Patent 1565097, Mummert 1925. <https://www.freepatentsonline.com/1565097.pdf>
- 23 Yaohua Zang, Gang Bao, Xiaojing Ye, Haomin Zhou, "Weak adversarial networks for high-dimensional partial differential equations", Journal of Computational Physics, Volume 411, 2020, 109409, ISSN 0021-9991, <https://doi.org/10.1016/j.jcp.2020.109409>