Model order reduction of time dependent PDEs by machine learning: application in CFD

Cipriano Innella
Simone Pescuma
Matteo Tomasetto

Contents

1 Introduction

Nowadays numerical simulation of time-dependent models holds a central and fundamental role in the field of applied mathematics. Indeed, several attempts and simulations have to be carried on in order to deepen the knowledge of models and to test their functioning varying its parameters and inputs. Moreover, already well tested numerical models are exploited to predict outputs of interest in a lot of real-life contexts such as biological, physical and financial ones.

Nevertheless, numerical analysis usually requires a huge effort in terms of computations, memory storage and time because of the high order of the models. Here it is the necessity of relying on a different numerical model which requires a sustainable effort while still being able to represent sufficiently well the original one which, from now on, will be called *high-fidelity model* (HF). This aim is achieved following a specific algorithm that involves the construction of a proper reduced-model that, because of its smaller order, is suitable for our goal.

The aim of this project is to study and exploit a model order reduction (MOR) method based on machine learning techniques in the case of a 2D fluid dynamic problem. In particular Prof. Francesco Regazzoni's *Model Learning* library and Prof. Luca Pegolotti's *Feamat* library have both been used to develop numerical results on MatLab and *Machine learning for fast and reliable solution of time-dependent differential equations* paper by Prof. Francesco Regazzoni, Prof. Luca Dedè and Prof. Alfio Quarteroni has been taken into account as a theoretical support.

2 Model order reduction

In this report we will limit ourselves to the important framework of time-invariant dynamical systems, so we can think to their high-fidelity model general form as:

$$\begin{cases} \dot{\mathbf{X}}(t) = \mathbf{F}(\mathbf{X}(t), \mathbf{u}(t)) & t \in (0, T] \\ \mathbf{X}(0) = \mathbf{X}_0 & \\ \mathbf{y}(t) = \mathbf{G}(\mathbf{X}(t)) & t \in (0, T] \end{cases} \tag{1}$$

with $\mathbf{F} : \mathbb{R}^N \times \mathbb{R}^{N_u} \to \mathbb{R}^N$ and $\mathbf{G} : \mathbb{R}^N \to \mathbb{R}^{N_y}$, where $\mathbf{X}(t) \in \mathbb{R}^N$ represents the high-fidelity state of the system, $\mathbf{u}(t) \in \mathbb{R}^{N_u}$ the input function and $\mathbf{y}(t) \in \mathbb{R}^{N_y}$ the output function. While the reduced model general form can be written as:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) & t \in (0, T] \\ \mathbf{x}(0) = \mathbf{x}_0 & \\ \tilde{\mathbf{y}}(t) = \mathbf{g}(\mathbf{x}(t)) & t \in (0, T] \end{cases} \tag{2}$$

with $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^{N_u} \to \mathbb{R}^n$ and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{N_y}$ that can both be evaluated with a smaller computational effort than $\mathbf{F}$ and $\mathbf{G}$ and where $\mathbf{x}(t)$ represents the reduced-order state which belongs to $\mathbb{R}^n$, typically with $n \ll N$. Finally $\tilde{\mathbf{y}}(t) \in \mathbb{R}^{N_y}$ represents the reduced-order output, which is in general different from the HF model output $\mathbf{y}(t)$.

Notice that the knowledge of the evolution of the full-order state $\mathbf{X}(t)$ is not essential since we are often interested only in certain synthetic output variables.

2.1 Model-based MOR

There are several techniques for model order reduction. Model-based MOR approach consists in relying on the high-fidelity model by constructing a reduced one through projection, i.e. approximating the full-order state space $\mathbb{R}^N$ with a lower-dimensional suitable subspace $\mathbf{V}$. This kind of approach let the reduced model inherit important properties, for instance stability, from the high-fidelity one and let possible the deduction of error estimates, but it requires the high-fidelity model to be fully known.

2.2 Data-driven MOR

Data-driven MOR approach consists in reconstructing the dynamic of the system just relying on pairs of inputs and outputs that, in a real-life context, are often gained by direct measurements of variables of interest. Because of its black-box nature, it is often exploited whenever the high-fidelity model is not accessible, hence it is very flexible and non intrusive, nevertheless any stability and convergence property can't be derived a-priori.

In a data-driven MOR approach, the high-fidelity model is not accessible, that is the functions $\mathbf{F}$ and $\mathbf{G}$ are not known, as well as the full-order state $\mathbf{X}(t)$, but only the input-output pairs $(\mathbf{u}(t), \mathbf{y}(t))$ are available. The problem consists in:

i) reconstructing the internal state of the system through its reduced description $\mathbf{x}(t)$ without the possibility of observing the true internal state of the system $\mathbf{X}(t)$;

ii) finding a model for the dynamics of $\mathbf{x}(t)$ itself.

Notice that goal ii) is just a way to reconstruct the map $\mathbf{u} \to \mathbf{y}$.

3 Model reduction strategy

The model order reduction problem can be reformulated as an optimization one in which we look for the best approximating model of the HF one into a class of simpler models, whose level of complexity is fixed a-priori.
It turns out to be extremely important to select a proper class of candidate models as well as their representation in order to define the optimization algorithm for solving the best-approximation problem.

In this framework, an Artificial Neural Network is convenient for the task of representing the right-hand side $\mathbf{f}$ and $\mathbf{g}$ in (2) through a training that lets it learn the physics from input-output pairs exploiting its ability to approximate any continuous function with a desired level of accuracy.

3.1 Dynamical model

We define a *model* as a general framework, which includes physical, mathematical or numerical model, and that associates a time-dependent output to a time-dependent input. We denote by $[0, T]$ a limited time interval, by $U \subset \mathbb{R}^{N_u}$ and by $Y \subset \mathbb{R}^{N_y}$ the sets where the input $\mathbf{u} : [0, T] \to U$ and the output $\mathbf{y} : [0, T] \to Y$ take values, respectively. We define an *experiment* as the action of inputing $\mathbf{u}(t)$ to the model and recording the corresponding output $\mathbf{y}(t)$ and we define a *sample* the couple $(\mathbf{u}, \mathbf{y})$.

We make some minimal and intuitive assumptions on the model:

i) *time invariance*: it is possible to consider, without loss of generality, each experiment starting from the initial time $t = t_0 = 0$;

ii) *existence of an initial state*: at time $t = 0$, for each experiment, the model is in the same initial state;

iii) *casuality principle*: the output $\mathbf{y}(t)$ can only depend on previous values of the input $\mathbf{u}(t)$ and not on future values;

iv) *no input-output direct dependence*: the output at time t depends only on the state of the system at the same time, but not directly on the input at time t.

For the sake of simplicity, we consider the case when both the input and output are continuous functions in time. Thus, the model can be seen as a map: $\varphi : \mathcal{U} \to \mathcal{Y}$ from the space of input signals $\mathcal{U} = \mathcal{C}^0([0, T]; U)$ to the space of output signals $\mathcal{Y} = \mathcal{C}^0([0, T]; Y)$.

3.2 Best-approximation problem

We perform N_s experiments with the HF model and we collect a set of N_s input-output pairs:

$$\{(\hat{\mathbf{u}}_j, \hat{\mathbf{y}}_j)\}_{j=1,\dots,N_s} \subset \mathcal{U} \times \mathcal{Y}$$

and select a subset of candidate models, which we denote by $\hat{\Phi} \subseteq \Phi$; we want to solve the best-approximation problem for the HF model in the least-squares sense in the subset $\hat{\Phi}$:

$$\varphi^* = \underset{\varphi \in \hat{\Phi}}{\operatorname{argmin}}\ J(\varphi)$$

where J is the following objective functional:

$$J(\varphi) = \frac{1}{2} \sum_{j=1}^{N_s} \int_0^T |\hat{\mathbf{y}}_j(t) - (\varphi(\hat{\mathbf{u}}_j))(t)|^2 \, dt \tag{3}$$

Thanks to the previous assumptions on the model, it is possible to restrict the choice of the input-output maps to those which are described by systems of ODEs. This reduces a lot the size of the problem we aim at solving.

Introducing the following spaces $\mathcal{F}_n := \{\mathbf{f} \in \mathcal{C}^0(\mathbb{R}^n \times U; \mathbb{R}^n)$, which are Lipschitz continuous in $\mathbf{x}$ uniformly in $\mathbf{u}\}$ and $\mathcal{G}_n := \mathcal{C}^0(\mathbb{R}^n; Y)$, we

can finally reformulate the previous optimization least-squares problem in abstract form as follows:

$$\begin{cases} \min\limits_{\mathbf{f}\in\hat{\mathcal{F}},\mathbf{g}\in\hat{\mathcal{G}}} \frac{1}{2}\sum_{j=1}^{N_s}\int_0^T |\hat{\mathbf{y}}_j(t)-\mathbf{g}(\mathbf{x}_j(t))|^2\,dt \\ \text{s.t. } \dot{\mathbf{x}}_j(t)=\mathbf{f}(\mathbf{x}_j(t),\hat{\mathbf{u}}_j(t)), t\in(0,T], \qquad j=1,...,N_s \\ \quad \mathbf{x}_j(0)=\mathbf{x}_0, \qquad\qquad\qquad\qquad\qquad\quad j=1,...,N_s \end{cases} \tag{4}$$

where $\hat{\mathcal{F}}$ and $\hat{\mathcal{G}}$ are suitable approximation respectively of $\mathcal{F}_n$ and $\mathcal{G}_n$. The unknowns in this problem are the functions $\mathbf{f}$ and $\mathbf{g}$, thus we are performing optimization in function spaces. We want to parametrize both the functions $\mathbf{f}$ and $\mathbf{g}$ by a finite set of real parameters, which we denote respectively $\mu\in\mathbb{R}^{N_f}$ and $\nu\in\mathbb{R}^{N_g}$, and then optimize with respect to μ and ν. In this way the desired regularization is obtained by controlling the size of N_f and N_g.
The parametrization of $\mathbf{f}$ and $\mathbf{g}$ can be obtained in different ways, here we decided to rely on their representation in terms of ANNs.

An ANN consists of a number of simple processing units, called *neurons*, each one incorporating a linear and non-linear application, interconnected to form a complex network with n_L layers of neurons, where each neuron of a given layer has a connection towards each neuron of the next layer. Thanks to the *universal approximation theorem*, ANNs with a single hidden layer can approximate any continuous function on a compact set with an arbitrarily small error, provided a sufficient number of hidden neurons.

The numerical approximation of problem (4) is obtained discretizing both the state equation (2) and the objective functional (3). In particular, we split the time domain $[0,T]$ into a collection of time instants
$0=t_0<t_1<...<t_M=T$. For the sake of simplicity we consider the case of constant time step Δt, i.e. $t_k=k\Delta t$ for $k=0,...,M$. To generalize, we suppose that the j-th experiment takes place in the interval $[0,T_j]$, where $T_j=M_j\Delta t$ and we denote $\mathbf{u}_j^k=\hat{\mathbf{u}}_j(t_k)\in U$ and $\mathbf{y}_j^k=\hat{\mathbf{y}}_j(t_k)\in Y$ the input and output at discrete times. The discretized version of the objective functional J thus reads:

$$J=\frac{1}{2}\Delta t\sum_{j=1}^{N_s}\sum_{k=0}^{M_j-1}|\mathbf{y}_j^k-\mathbf{g}(\mathbf{x}_j^k;\nu)|^2$$

Finally, we decide to discretize the state equation by means of the forward Euler scheme.

Therefore, the discrete counterpart of problem (4) reads:

$$\begin{cases} \min\limits_{(\mu,\nu)\in\mathbb{R}^{N_f+N_g}} \frac{1}{2}\Delta t \sum_{j=1}^{N_s}\sum_{k=0}^{M_j-1} |\mathbf{y}_j^k - \mathbf{g}(\mathbf{x}_j^k;\nu)|^2 \\ \text{s.t. } \mathbf{x}_j^{k+1} = \mathbf{x}_j^k + \Delta t \mathbf{f}(\mathbf{x}_j^k, \mathbf{u}_j^k;\mu), \quad k = 0, ..., M_j - 1, \quad j = 1, ..., N_s \\ \quad \mathbf{x}_j^0 = \mathbf{x}_0, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad j = 1, ..., N_s \end{cases}$$

Once the optimization problem in the discrete setting has been solved, i.e. the ANN has been trained and the optimal weights $\mu \in \mathbb{R}^{N_f}$ and $\nu \in \mathbb{R}^{N_g}$ of $\mathbf{f}$ and $\mathbf{g}$ respectively have been found, the reduced model is available in the desired form.

4 Heat equation

Before dealing with the fluid dynamic problem, we decided to work with a less complex model in order to understand more easily the algorithm.
In particular, we have worked with the heat equation in a problem where the input was a collection of nine time-dependent functions which represented the thermal conductivity coefficient in nine subdomains and the output was the temperature in three points of the domain.

4.1 Mathematical formulation of the problem

Consider the domain $\Omega = (0, 1.5)^2$, whose boundary $\partial\Omega$ is partitioned into the top border Γ_t, in contact with a heat reservoir with zero temperature, the bottom border Γ_b, with a constant inward heat flux $\varphi = 1$, and the wall borders Γ_w, with no-flux boundary conditions. The time evolution of the temperature $\psi(\mathbf{x}, t)$ in the domain Ω is described by the heat equation:

$$\begin{cases} \frac{\partial}{\partial t}\psi(\mathbf{x},t) - \nabla\cdot(k(\mathbf{x},\mathbf{u}(t))\nabla\psi(\mathbf{x},t)) = 0 & \text{in } \Omega, \text{ for } t > 0 \\ k(\mathbf{x},\mathbf{u}(t))\nabla\psi(\mathbf{x},t)\cdot\mathbf{n} = 0 & \text{on } \Gamma_w, \text{ for } t > 0 \\ k(\mathbf{x},\mathbf{u}(t))\nabla\psi(\mathbf{x},t)\cdot\mathbf{n} = \varphi & \text{on } \Gamma_b, \text{ for } t > 0 \\ \psi(\mathbf{x},t) = 0 & \text{on } \Gamma_t, \text{ for } t > 0 \\ \psi(\mathbf{x},0) = 0 & \text{on } \Omega \end{cases}$$

The domain Ω is partitioned into 9 subdomains Ω_i of equal size, $i = 1, ..., 9$ as in Fig 1. Let us consider the piecewise thermal conductivity coefficient k, parametrized by the 9-dimensional input $\mathbf{u}(t) \in [10, 100]$, defined as:

$$k(\mathbf{x}, \mathbf{u}) = \sum_{i=1}^{9} u_i \mathbb{1}_{\Omega_i}(\mathbf{x}),$$

where $\mathbb{1}_{\Omega_i}$ is the indicator function of Ω_i. Finally, consider three probes, located at the center of the subdomains Ω_1, Ω_5 and Ω_9, measuring the time evolution of the temperature in such points. The output $\mathbf{y}(t) \in \mathbb{R}^3$ is the vector collecting the three temperature values.

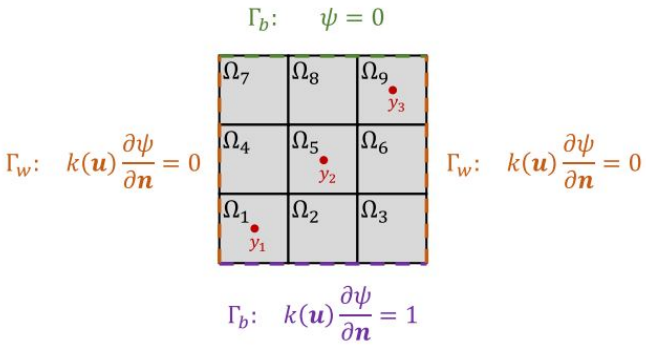


Figure 1: Domain and boundary condition. Image from [1].

4.2 Numerical results

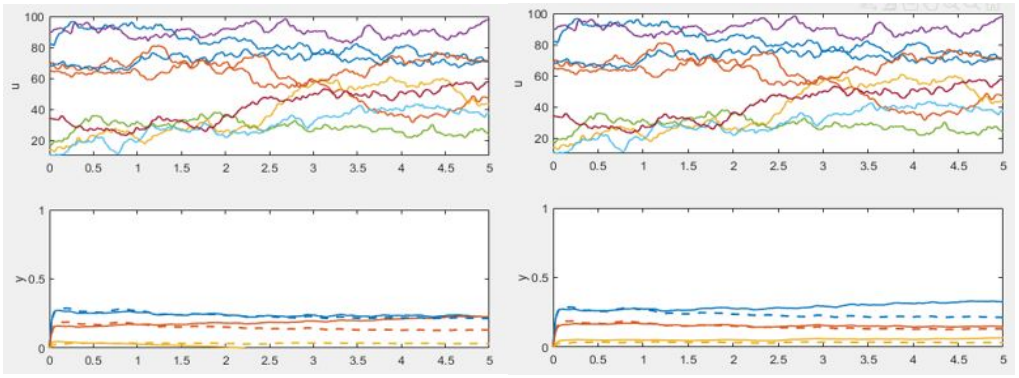
We have checked the functioning of the *model learning* library in the case of the described problem by using 9 input variables and 3 output ones consistently to the model formulation above.

First of all we have built the dataset - training, validation and test - required for our purpose solving the HF model with several different inputs. For the HF resolution, we consider the $P2$ Finite Element approximation on a 30 by 30 uniform square elements grid and we employ the backward Euler scheme, with $\Delta t = 10^{-2}$, for the time discretization, implemented in the Matlab finite element library *Feamat*.

We have trained different ANNs by exploiting different kinds of input functions - such as random, linear combination of trigonometric functions, and linear composition of polynomials - and checked the output on a time horizon of $T = 1$. After the training phase with $T = 1$, we have generated a test set with time horizon $T = 5$ for testing the ability of prediction in time. Depending on the kind of input, different scenarios arose.

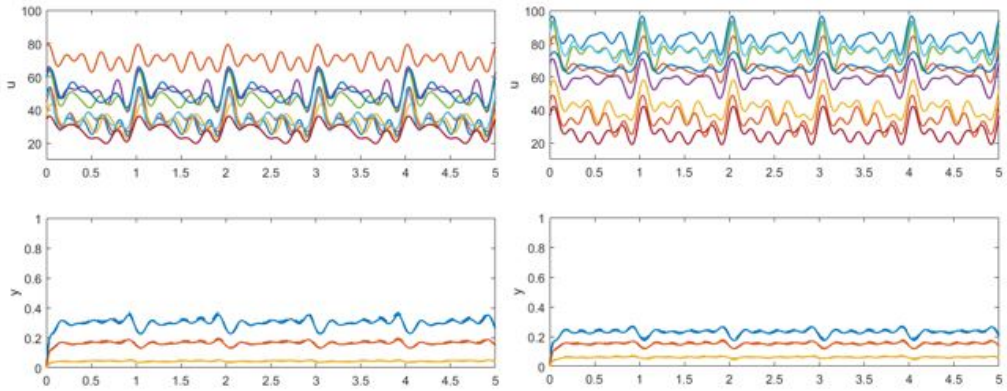
Considering as input functions random generated ones, we have noticed that the learning process works fine, while the prediction is not good. We tried to increase the number n of the order of the reduced internal state and we noticed that prediction improved. This fact is related to the randomic

nature of the input itself which makes prediction hard to deal with.
The following pictures show some significant plots of the above critical situation where the solid lines represent the ANN output, while the dashed lines are the corresponding sample of the test set.

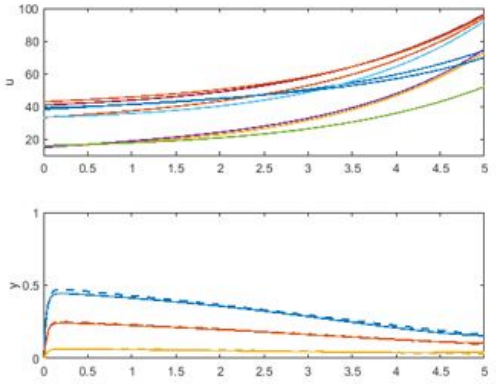


(a) n=3, layF=7, T=5 (b) n=5, layF=7, T=5

Considering as input functions linear combination of either trigonometric functions and polynomials, we have noticed that both learning and prediction work really well. Differently from the previous case, the goodness of prediction holds and this is related to the higher regularity of the input functions. The following pictures show some plots of these two cases.



(a) n=3, layF=7, T=5 (b) n=5, layF=7, T=5



(a) n=3, layF=7, T=5

5 Navier-Stokes equation

At last we are now ready to face our goal problem, which is the one governed by the unsteady formulation of Navier-Stokes equations. In particular we consider a fluid flowing in a rectangular channel with a rigid cylindrical obstacle placed in the middle of it. We selected as inputs the velocity of the fluid on the inflow boundary, i.e. the left side of the rectangle, and the Reynolds number of the fluid, while as output we chose the resistance exerted by the cylinder on the flowing fluid itself and the lift coefficient.

5.1 Mathematical formulation of the problem

Since we are considering a 2 dimensional problem: the domain is $\Omega \subset \mathbb{R}^2$, the unknown velocity $\mathbf{u}(\mathbf{x}, t)$ is a vector function such that $\mathbf{u}(x, y, t) = (u_1(x, y, t), u_2(x, y, t))$, the unknown pressure p is a scalar function and μ is the kinematic viscosity of the fluid. Apart from the conservation of linear momentum equation, we considered the conservation of mass equation which is, in general, satisfied by incompressible fluids and, in particular, is satisfied by fluids whose density ρ is constant. Moreover, we chose boundary and initial conditions coherently to the formulation of our problem, in particular we selected a no-slip condition for the velocity on the surface of the cylinder and on the parallel banks of the channel.

Hence the unsteady formulation of Navier-Stokes equations read:

$$\begin{cases} \rho\frac{\partial}{\partial t}\mathbf{u} - \mu\Delta\mathbf{u} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f} & \text{in } \Omega\times(0,T) \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega\times(0,T) \\ \mathbf{u} = \mathbf{0} & \text{in } \Gamma_w\times(0,T) \\ \mathbf{u} = \boldsymbol{\varphi}_{in} & \text{in } \Gamma_{in}\times(0,T) \\ \mu\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n} = 0 & \text{in } \Gamma_{out}\times(0,T) \\ \mathbf{u} = \mathbf{u}_0 & \text{in } \Omega\times\{0\} \end{cases}$$

where $\Omega\subset\mathbb{R}^2$ is a rectangle drilled in correspondence of the cylinder, Γ_{in} and Γ_{out} are the inflow and outflow side of the boundary $\partial\Omega$, Γ_w is the part of the boundary $\partial\Omega$ composed of the two horizontal sides of the rectangle and the outer surface of the cylinder itself. Moreover, $\mathbf{f}$ is a forcing term per unit mass, $\boldsymbol{\varphi}_{in}$ is the input function which represents the velocity of the fluid on the inflow side of the boundary, and $\mathbf{u}_0$ is the velocity of the fluid in the whole domain Ω at the initial time $t=0$. Notice that for our simulations we fixed $\mathbf{f}=\mathbf{0}$ and $\mathbf{u}_0=\mathbf{0}$, the latter represents the state of rest of the fluid. Finally, we recall that the *Reynolds number* of the fluid is defined as:

$$Re = \frac{|\mathbf{U}|L\rho}{\mu}$$

where L is a characteristic length of the domain Ω, in our context it is the length of the channel, and $\mathbf{U}$ is a characteristic velocity of the fluid.
If $Re \ll 1$ the convective term $(\mathbf{u}\cdot\nabla)\mathbf{u}$ can be neglected and Navier-Stokes equations reduce to Stokes equation whose existence and uniqueness of the solution has been proved, while if $Re \gg 1$ the convective term can't be neglected and problems concerning the existence and uniqueness of the solution arise.

5.2 Algebraic formulation

In order to derive the finite element linear system we deal with space and time discretization of Navier-Stokes equations where - for the sake of semplicity - we consider $\rho=1$; we apply the Galerkin approximation using $P2$ element for the velocity (V_h) and $P1$ element for the pressure (Q_h):

$$\forall t>0 \text{ find } \mathbf{u}_h\in V_h\subset[H^1(\Omega)]^2 \text{ and } p_h\in Q_h\subset L^2(\Omega) \text{ such that:}$$

$$\int_\Omega \frac{\partial\mathbf{u}_h}{\partial t}\mathbf{v}_h + a(\mathbf{u}_h,\mathbf{v}_h) + b(p_h,\mathbf{v}_h) + c(\mathbf{u}_h,\mathbf{u}_h,\mathbf{v}_h) = 0 \quad \forall\mathbf{v}_h\in V_h\cap[H^1_{\Gamma_D}(\Omega)]^2$$

$$b(q_h,\mathbf{v}_h) = 0 \quad \forall q_h\in Q_h$$

with $\mathbf{u}_h$ that satisfies the boundary conditions set in the problem above and where:

$$a(\mathbf{u}_h, \mathbf{v}_h) = \int_\Omega \mu(\nabla \mathbf{u}_h + \nabla^T \mathbf{u}_h) \cdot \nabla \mathbf{v}_h \, d\Omega$$

$$b(p_h, \mathbf{v}_h) = -\int_\Omega p_h \nabla \cdot \mathbf{v}_h \, d\Omega$$

$$c(\mathbf{u}_h, \mathbf{u}_h, \mathbf{v}_h) = \int_\Omega (\mathbf{u}_h \cdot \nabla)\mathbf{u}_h \cdot \mathbf{v}_h \, d\Omega$$

If we denote with $\mathbf{u}$ and $\mathbf{p}$ the vectors of the nodal values of $\mathbf{u}_h$ and $\mathbf{p}_h$ we obtain the following semi-discrete approximation:

$$\begin{cases} M\frac{\partial}{\partial t}\mathbf{u}(t) + A\mathbf{u}(t) + B^T\mathbf{p}(t) + C(\mathbf{u}(t))\mathbf{u}(t) = \mathbf{0} & \text{in } \Omega \times (0,T) \\ B\mathbf{u}(t) = \mathbf{0} & \text{in } \Omega \times (0,T) \\ \mathbf{u} = \mathbf{u}_0 & \text{in } \Omega \times \{0\} \end{cases}$$

where:

$$C(\mathbf{u}(t)) = [C_{m,i}(t)]_{m,i} \qquad C_{m,i}(t) = \int_\Omega \mathbf{u}(t) \cdot \nabla \varphi_i \varphi_m \, d\Omega$$

with $\{\varphi_i\}_{i=1}^{N_h}$ basis for V_h is the convective matrix; similarly we can define the mass matrix M, the stiffness matrix A and the divergence matrix B. Moreover we use the backward Euler method to discretize the problem in time: it means that we set $t^k = k\Delta t$ for $k = 0, ..., M$ and $\mathbf{u}^{(k)} \simeq \mathbf{u}(t^k)$, $\mathbf{p}^{(k)} \simeq \mathbf{p}(t^k)$ getting the following formulation:

$$\begin{cases} M\dfrac{\mathbf{u}^{(k+1)} - \mathbf{u}^{(k)}}{\Delta t} + A\mathbf{u}^{(k+1)} + B^T\mathbf{p}^{(k+1)} + C(\mathbf{u}^{(k+1)})\mathbf{u}^{(k+1)} = \mathbf{0} \\ B\mathbf{u}^{(k+1)} = 0 \end{cases}$$

where the equations hold in $\Omega \times (0,T)$. Now for each k we collect in the vector $\mathbf{x}^{(k)}$ all the nodal values of the velocity in the x, y direction and the pressure, hence it has length equal to the number of vertices of the mesh. Finally the solution at each time step is given by the linear system:

$$\tilde{A}\mathbf{x}^{(k+1)} = \tilde{B}\mathbf{x}^{(k)} + \tilde{\mathbf{b}} \qquad \forall k \geq 0$$

where $\mathbf{x}^{(0)}$ is given by the initial condition, while the matrices $\tilde{A}$ and $\tilde{B}$ and the vector $\tilde{\mathbf{b}}$ can be computed by the previous system taking into account the definition of the vector $\mathbf{x}^{(k)}$ and the boundary conditions.

5.3 Mesh and outputs

Exploiting GMSH software, we have created an unstructured triangular mesh with non overlapping elements in order to deal with the numerical analysis of our problem. In particular, the vertices of the rectangle are $(-2,-2)$, $(5,-2)$, $(5,2)$ and $(-2,2)$, while the cylinder is represented in terms of a circular hole centered in $(1,0)$ with radius $R = 0.5$. Globally there are 2474 vertices and 4948 elements.

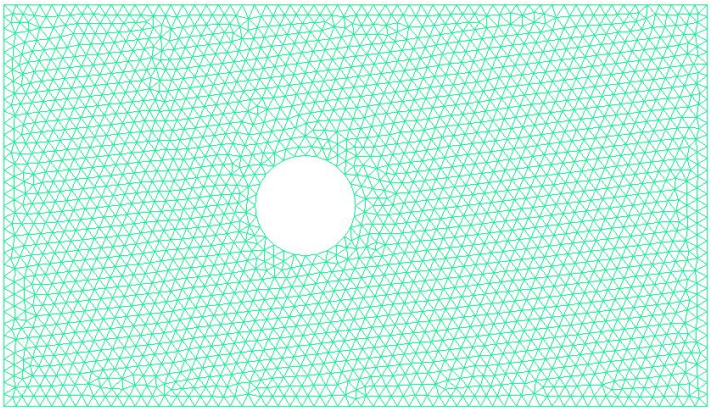


Figure 5: Mesh grid

The outputs considered are the *drag* and the *lift* coefficient of the obstacle in the channel: for their evaluation, we exploit the variational form of the Navier–Stokes equations instead of using their definitions, the latter being directly related to the integral of the stress acting on the cylinder boundaries. The two approaches, that are equivalent at the 'continuous level', are in fact different at the 'discrete level', but it yields accurate computation of the outputs.

In order to introduce the method to approximate the drag and lift coefficient, we recall the weak formulation of the Navier–Stokes equations. This can be done by introducing suitable functional spaces for the velocity $\mathbf{u}$ and the pressure p:

$$\text{find } \mathbf{u} \in [H^1(\Omega)]^2 \text{ and } p \in L^2(\Omega) \text{ such that:}$$

$$a(\mathbf{u},\mathbf{v}) + b(p,\mathbf{v}) + c(\mathbf{u},\mathbf{u},\mathbf{v}) = 0 \ \forall \mathbf{v} \in [H^1(\Omega)]^2$$

$$b(q,\mathbf{u}) = 0 \ \forall q \in L^2(\Omega)$$

where

$$a(\mathbf{u},\mathbf{v}) = \int_\Omega \mu(\nabla\mathbf{u} + \nabla^T\mathbf{u}) \cdot \nabla\mathbf{v} \, d\Omega$$

$$b(p, \mathbf{v}) = -\int_\Omega p\nabla \cdot \mathbf{v}\ d\Omega$$

$$c(\mathbf{u}, \mathbf{u}, \mathbf{v}) = \int_\Omega \rho(\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{v}\ d\Omega$$

and where ρ is the density and μ is the viscosity.
We will focus on the drag coefficient which, by definition, is:

$$c_D(\mathbf{u}, p) = -\frac{2}{L\rho V^2} \oint_{\Gamma_{\text{cylinder}}} \mu(\nabla\mathbf{u} + \nabla^T\mathbf{u} - p\mathbb{I})\hat{\mathbf{n}} \cdot \hat{\mathbf{v}}_\infty\ d\Gamma$$

where Γ_{cylinder} is the boundary of the cylinder, V is the norm of the velocity of the incoming fluid, L is the length of the channel, $\mathbb{I}$ is the identity tensor, $\hat{\mathbf{n}}$ is the outer unit normal with respect to the cylinder and $\hat{\mathbf{v}}_\infty$ is the unit vector directed as the horizontal axis.

The drag coefficient c_D can be evaluated once the Navier–Stokes equations are resolved and the velocity field $\mathbf{u}$ and the pressure p are known. However, the computation of c_D by means of the definition can lead to inaccurate results even if the computational grid is quite fine. Better results can be achieved by using an alternative expression for c_D, which we indicate with $\hat{c}_D$, dependent on the variational form used for the Navier–Stokes equations. In order to provide the expression of $\hat{c}_D$, we first express c_D in the following form:

$$c_D(\mathbf{u}, p) = \frac{2}{L\rho V^2} \oint_{\partial\Omega} \mu(\nabla\mathbf{u} + \nabla^T\mathbf{u} - p\mathbb{I})\hat{\mathbf{n}} \cdot \mathbf{\Phi}_\infty\ d\Gamma$$

where $\mathbf{\Phi}_\infty \in [H^1(\Omega)]^2$ is such that:

$$\begin{cases} \mathbf{\Phi}_\infty|_{\Gamma_{\text{cylinder}}} = -\hat{\mathbf{v}}_\infty \\ \mathbf{\Phi}_\infty|_{\partial\Omega\setminus\Gamma_{\text{cylinder}}} = 0 \end{cases}$$

By means of the Gauss theorem and Green's identity, drag coefficient expression becomes:

$$c_D(\mathbf{u}, p) = \frac{2}{L\rho V^2} \int_\Omega \nabla \cdot (\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u} - p\mathbb{I})^T\mathbf{\Phi}_\infty)\ d\Omega =$$

$$= \frac{2}{L\rho V^2} \int_\Omega (\nabla \cdot (\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u} - p\mathbb{I})) \cdot \mathbf{\Phi}_\infty + \mu(\nabla\mathbf{u} + \nabla^T\mathbf{u} - p\mathbb{I}) \cdot \nabla\mathbf{\Phi}_\infty)\ d\Omega$$

Owing to the Navier-Stokes equations and their weak formulation, we have:

$$\int_\Omega \nabla \cdot (\mu(\nabla\mathbf{u} + \nabla^T\mathbf{u} - p\mathbb{I})) \cdot \mathbf{\Phi}_\infty\ d\Omega = \int_\Omega (\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{\Phi}_\infty\ d\Omega = c(\mathbf{u}, \mathbf{u}, \mathbf{\Phi}_\infty)$$

$$\int_\Omega \mu(\nabla \mathbf{u} + \nabla^T \mathbf{u} - p\mathbb{I}) \cdot \nabla \mathbf{\Phi}_\infty \, d\Omega = a(\mathbf{u}, \mathbf{\Phi}_\infty) + b(p, \mathbf{\Phi}_\infty)$$

In conclusion, since $b(q, \mathbf{u}) = 0 \ \forall q \in L^2(\Omega)$, $\hat{c}_D$ reads:

$$\hat{c}_D(\mathbf{u}, p) = \frac{2}{L\rho V^2} \cdot (a(\mathbf{u}, \mathbf{\Phi}_\infty) + b(p, \mathbf{\Phi}_\infty) + b(q, \mathbf{u}) + c(\mathbf{u}, \mathbf{u}, \mathbf{\Phi}_\infty))$$

$$\forall q \in L^2(\Omega)$$

where - for the sake of simplicity - ρ will be fixed as $\rho = 1$.
The provided form of the drag coefficient allows accurate computations and will be used in this work. In practice, going from the variational formulation to the algebraic formulation, we compute the drag coefficient at each time k in the following manner:

$$\hat{c}_D^{(k+1)} = (\tilde{A}\mathbf{x}^{(k+1)} - \tilde{B}\mathbf{x}^{(k)} - \tilde{\mathbf{b}}) \cdot \mathbf{D} \qquad \forall k \geq 0$$

where $\hat{c}_D^{(0)} = 0$ and $\mathbf{x}^{(k+1)}$ is the solution of the linear system $\tilde{A}\mathbf{x}^{(k+1)} = \tilde{B}\mathbf{x}^{(k)} + \tilde{\mathbf{b}}$ obtained previously; notice that now, since the boundary conditions are not considered in the weak formulation introduced above, we do not apply them on the matrices $\tilde{A}$,$\tilde{B}$ and on $\tilde{\mathbf{b}}$. Moreover $\mathbf{D}$ is a vector such that $\mathbf{D}(i) = 1$ if the value $\mathbf{x}(i)$ is the x-component of the velocity at a cylinder node of the mesh, $\mathbf{D}(i) = 0$ otherwise.

Regarding the lift coefficient, the same strategy can be applied with the only difference that now $\mathbf{D}$ is related with the y-component of the velocity at the cylinder nodes.

5.4 Numerical results

In this section we provide the main results we have collected in response to our numerical simulations. As we have previously done for the heat equation problem, we have exploited several ANNs trained with different types of input, e.g. discontinuous, trigonometric, polynomial and random, and we have changed other important parameters, e.g. the dimension of the state space of the reduced model n, the number of layers related to the functions $\mathbf{f}$ and $\mathbf{g}$, in order to state significant conclusions about the correlation between the registered outputs and those parameters.

Hereafter some plots will be shown, note that - for every pair of plots - the first one represents the input (the blue line is the constant Reynolds number, the red one is the x-component of the velocity, the yellow one in the y-component of the velocity) while the second one represent the output (the blue line is the drag coefficient, the red one is the lift coefficient; dashed line is the real output, solid one is the prediction). Sometimes, because of scale reasons, the Reynolds number is not shown, even though it is always fixed and considered as constant.

We have chosen reasonable ranges of the inputs for physical reasons, in particular the Reynolds number is such that $Re \in (10, 1000)$, the x-velocity of the fluid on the inflow boundary is always positive, while the y-velocity can be either positive or negative.

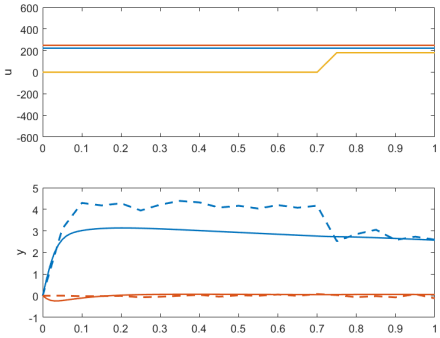
5.4.1 Discontinuous input

First of all we consider the results concerning ANNs trained with discontinuous inputs. In particular we have created such inputs by considering a constant x-velocity and a constant y-velocity with a single jump discontinuity at a random time. The drag is affected by the discontinuity of the input and shows up with a fast variation.

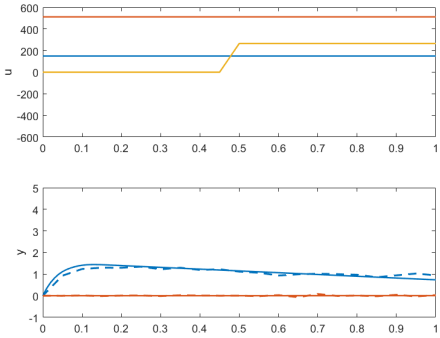
As far as the training phase of the ANNs with $T = 1$, we have increased the dimension of the state space n of the reduced model and, in the last simulation, we have increased the layers related to the function $\mathbf{f}$. By increasing the dimension n, we see that the overall L^2-error decreases. For $n = 4$ and $n = 5$ the predicted output badly recognize the discontinuity of the input, for $n = 7$ it shows up with a slow variation, while for $n = 6$ it shows up with a fast variation in accordance with the real output. Lastly fixing $n = 6$ and increasing the number of layers related to the function $\mathbf{f}$ from 7 to 10, we see that the overall L^2-error significantly decreases.

As far as the predictive ability of the ANNs with $T = 3$, we have fixed $n = 6$ and 10 layers for the function $\mathbf{f}$. The predicted output does not replicate perfectly the real one, but it is largely acceptable with a good shape.

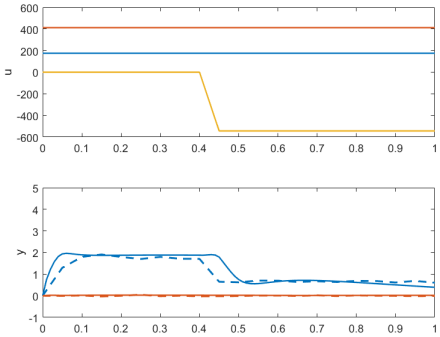
	n=4 layF=7 T=1	n=5 layF=7 T=1	n=6 layF=7 T=1	n=7 layF=7 T=1	n=6 layF=10 T=1	n=6 layF=10 T=3
L^2-error	0,402	0,398	0,340	0,380	0,212	0,504



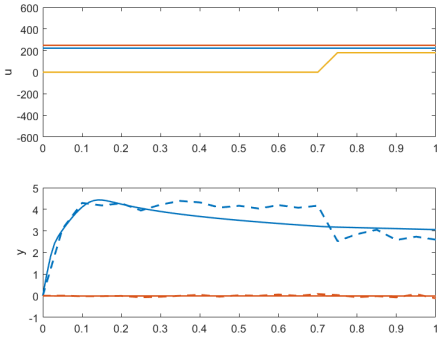
(a) n=4, layF=7, T=1



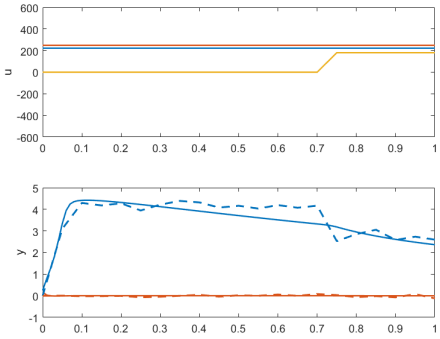
(b) n=5, layF=7, T=1



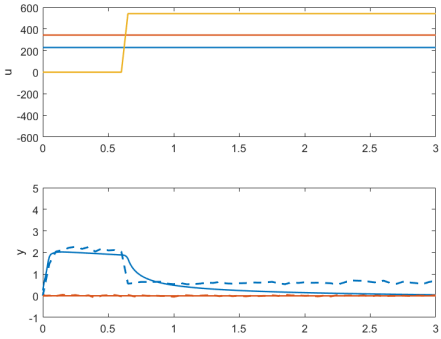
(c) n=6, layF=7, T=1



(d) n=7, layF=7, T=1



(e) n=6, layF=10, T=1



(f) n=6, layF=10, T=3

5.4.2 Polynomial input

Now we deal with the results regarding ANNs trained with polynomial inputs for the incoming velocity. In particular we have created such inputs by constructing a linear combination of $t, t^2, ..., t^5$ exploiting random-generated coefficients $a_i, i = 1, ..., 5$, i.e. every input function has the form:

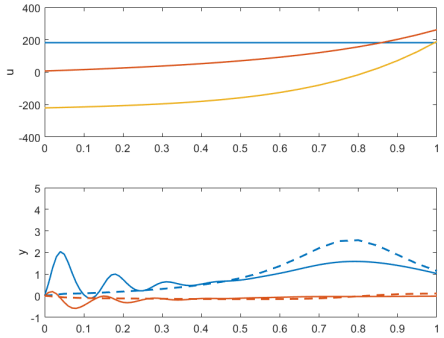
$$u(t) = \sum_{i=1}^{5} a_i t^i$$

As far as the training phase of the ANNs with $T = 1$, we have increased the dimension of the state space n of the reduced model and, in the last simulations, we have increased the layers related to the function $\mathbf{f}$. By increasing the dimension n, we see that the overall L^2-error generally decreases. Indeed, for $n = 3$ and $n = 4$ the output is bad and the error is high, for $n = 5$ and $n = 6$ it decreases as expected, while for $n = 7$ it blows up, this fact may be related to an overfitting phenomenon which makes the ANNs too specific and unable to adapt to slightly different scenarios from the training one. Fixing the best ANN, which is the one with $n = 6$, and increasing the layers of $\mathbf{f}$ to 10, the overall L^2-error is acceptably low, even if it is higher to the case with a lower number of layers.

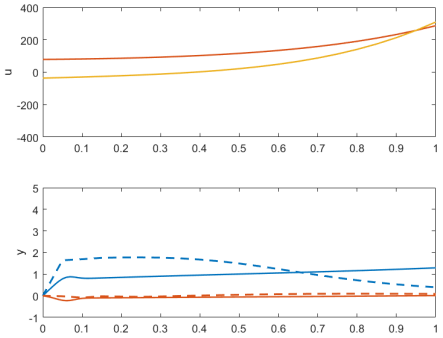
As far as the predictive ability of the ANNs with $T = 3$, either in the case of $n = 6$ and 10 layers for the function $\mathbf{f}$ and in the case of 7 layers, the predicted output is not good and the overall L^2-error is unacceptably high: the prediction is fine until the training time $T = 1$ but, for $1 < t < 3$, it does not faithfully replicate the true output.

	n=3 layF=7 T=1	n=4 layF=7 T=1	n=5 layF=7 T=1	n=6 layF=7 T=1	n=7 layF=7 T=1
L^2-error	2,170	2,400	0,670	0,346	1,220

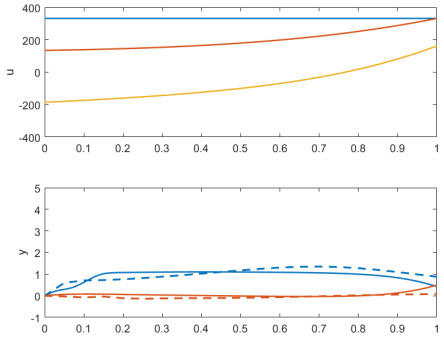
	n=6 layF=10 T=1	n=6 layF=7 T=3	n=6 layF=10 T=3
L^2-error	0,619	3,330	2,600



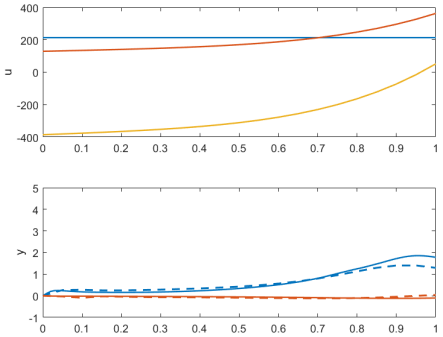
(a) n=3, layF=7, T=1



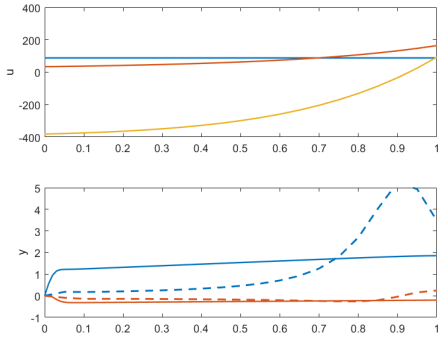
(b) n=4, layF=7, T=1



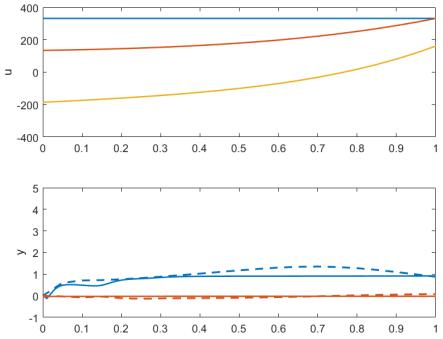
(c) n=5, layF=7, T=1



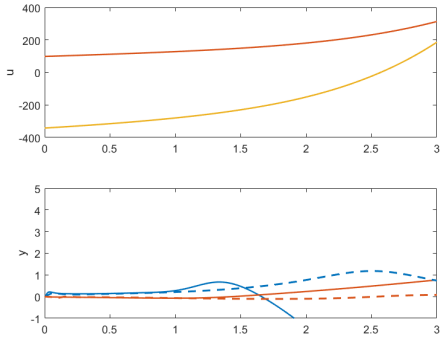
(d) n=6, layF=7, T=1



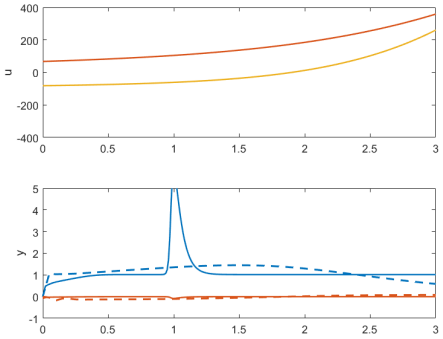
(e) n=7, layF=7, T=1



(f) n=6, layF=10, T=1



(g) n=6, layF=7, T=3



(h) n=6, layF=10, T=3

5.4.3 Trigonometric input

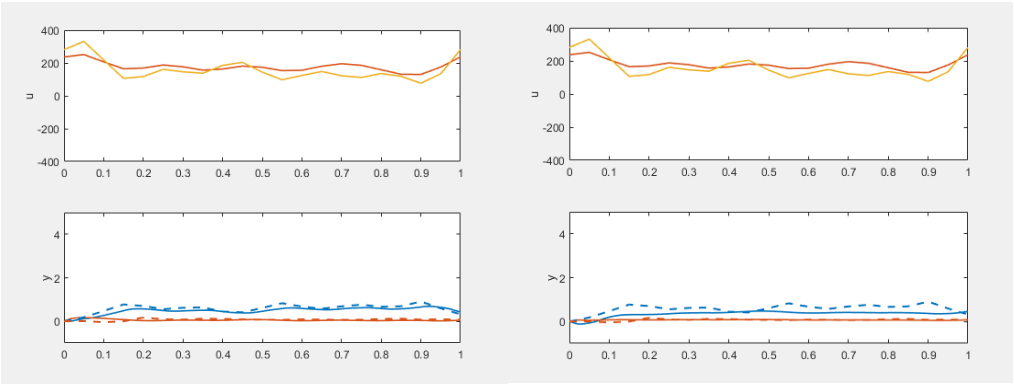
Here we analyze the results as far as ANNs trained with trigonometric inputs for the incoming velocity. In particular we have created such inputs by constructing a linear combination of $\sin(t)$, $\cos(t)$, $\sin(2t)$, $\cos(2t)$,..., $\cos(5t)$ exploiting random-generated coefficients $a_i, b_i, i = 1, ..., 5$, i.e. every input function has the form

$$u(t) = \sum_{i=1}^{5} a_i \sin(it) + b_i \cos(it)$$

Concerning the training phase of the ANNs with $T = 1$, also here we have increased the dimension of the state space n of the reduced model and lastly we have increased the layer related to the function **f**. Differently from the polynomial case, the overall L^2-error seems to be good from the starting dimension $n = 3$ and, by increasing it, the error decreases up to $n = 4$, while for $n = 5$ and $n = 6$ it increases from one order and then it returns to decrease for $n = 7$. Now taking the best ANNs, i.e. with $n = 4$, and increasing the layers of **f** to 10, the overall L^2-error is still low but a little higher of the previous one, this because we may need more iteration to get a better accuracy. Again with the best ANNs ($n = 4$) we tested the predictive ability with $T = 3$, either in the case of 10 layers for the function **f** and in the case of 7 layers, and, as we can see, the overall L^2-error is better in the case of 10 layers. Therefore the results obtained in the trigonometric case are very good: this may be due to the constant trend that we can notice in the outputs.

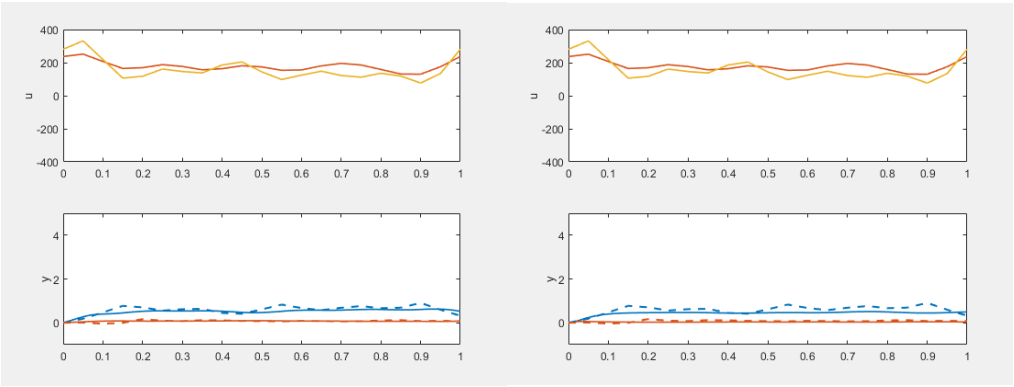
	n=3 layF=7 T=1	n=4 layF=7 T=1	n=5 layF=7 T=1	n=6 layF=7 T=1	n=7 layF=7 T=1
L^2-error	0,558	0,399	1,270	1,290	0,974

	n=4 layF=10 T=1	n=4 layF=7 T=3	n=4 layF=10 T=3
L^2-error	0,873	1,300	0,580



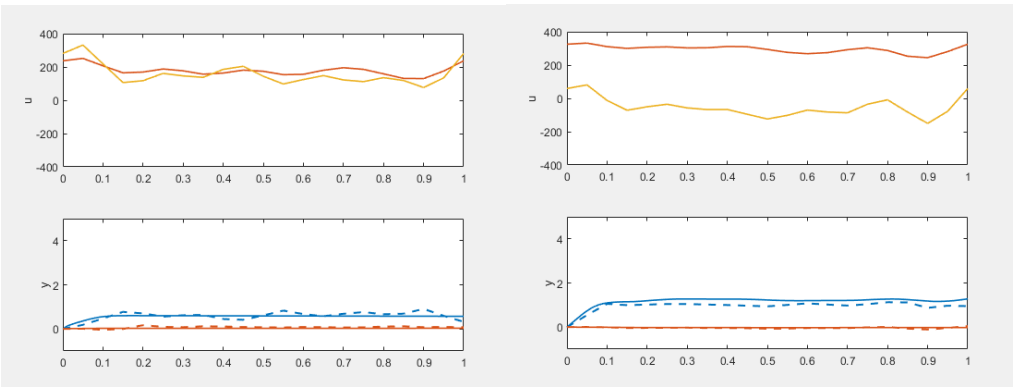
(a) n=3, layF=7, T=1

(b) n=4, layF=7, T=1



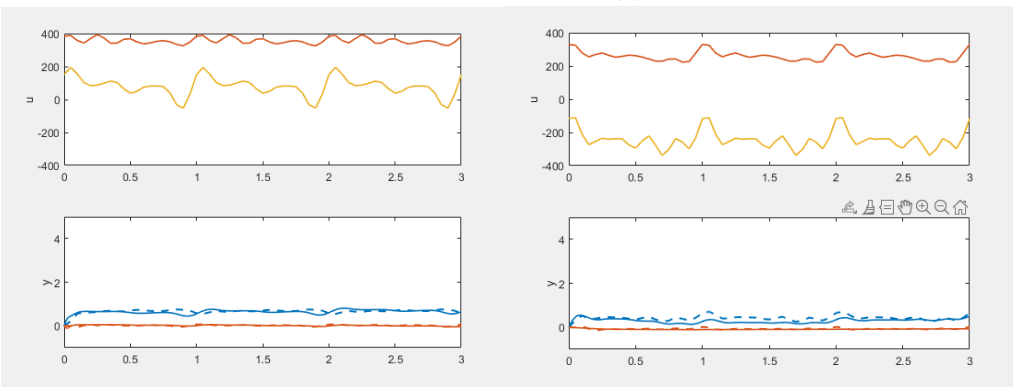
(c) n=5, layF=7, T=1

(d) n=6, layF=7, T=1



(e) n=7, layF=7, T=1

(f) n=4, layF=10, T=1



(g) n=4, layF=7, T=3

(h) n=4, layF=10, T=3

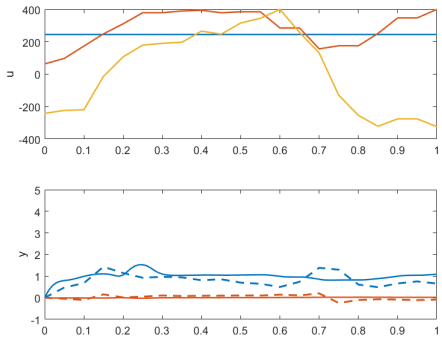
5.4.4 Random input

Lastly, we consider the results related to ANNs trained with random inputs. In particular we have created such inputs by generating random value to assign to each time step Δt.

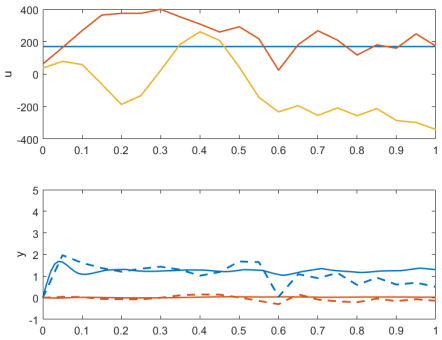
As far as the training phase of the ANNs with $T = 1$, we have increased the dimension of the state space n of the reduced model and, in the last simulation, we have increased the layers related to the function $\mathbf{f}$. By increasing the dimension n, we see that the overall L^2-error decreases, nevertheless if the real output slightly oscillates,the ANN generates an almost-constant output, which is not accurate.

As far as the predictive ability of the ANNs with $T = 3$, in the case of $n = 8$ and 10 layers for the function $\mathbf{f}$, even though the overall L^2-error is not too high, graphically the predicted output is not good and it shows up with the same tendency of representing constants. The latter may be related to the fact that random inputs are very general and so they are able to represent extremely various input whose nature is very different.

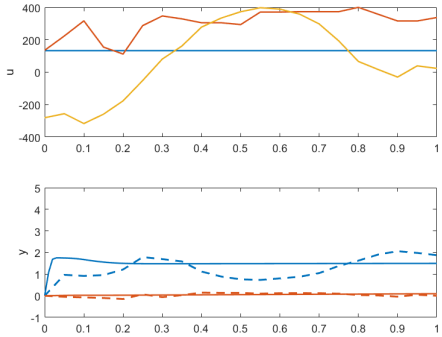
	n=5 layF=7 T=1	n=6 layF=7 T=1	n=7 layF=7 T=1	n=8 layF=7 T=1	n=8 layF=10 T=1	n=8 layF=7 T=3
L^2-error	0,825	0,708	0,670	0,640	0,680	0,774



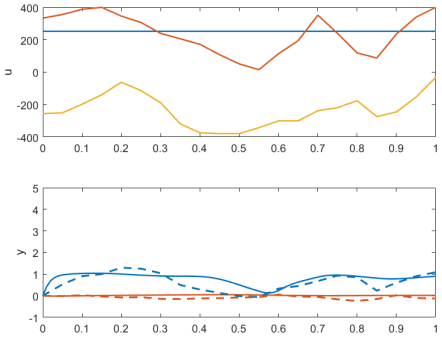
(a) n=5, layF=7, T=1



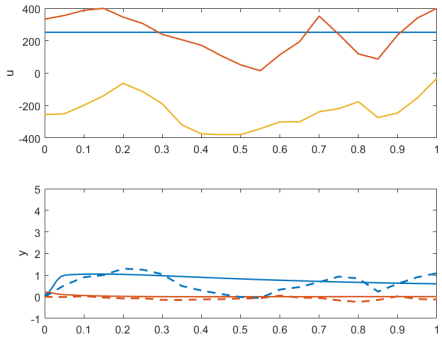
(b) n=6, layF=7, T=1



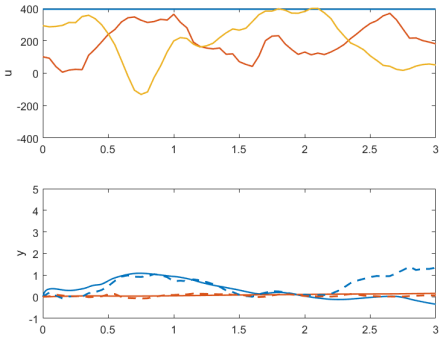
(c) n=7, layF=7, T=1



(d) n=8, layF=7, T=1



(e) n=8, layF=10, T=1



(f) n=8, layF=10, T=3

6 Conclusion

We have exploited a data-driven nonlinear MOR approach to predict output of interest of selected problems. In order to state significant conclusions and observations about the simulations we have carried on, we rely on the results we have previously exhibited.

The general result we gained with every type of input we used to train the ANNs is that a large dimension of the state space of the reduced model is better than a lower one: this is related to the fact that the larger the number of state variables, the better. Indeed, it is easier to represent and simplify a model whose dimension is very high with a consistent number of variables, rather than with a very small number of variables. That is related to the fact that reducing the order of the model is equivalent to forcing a lower order one to represent it exploiting less variables each of which has to contain the dynamic of much more variables.

In our simulations, we have always selected the best ANNs in terms of dimension of the reduced model and we have increased the number of layers of neurons of the ANNs. This increasing has generally led to an improvement with respect to the predictive ability: the latter is related to the fact that the larger the number of layers , the better. Indeed, a more complex ANN in terms of layers is capable to learn and predict better than a less complex one.

A peculiar remark is that it is not possible to increase either the dimension of the reduced model and the layers of the ANN to get arbitrary small errors. Indeed, in both cases there exist a threshold beyond which overfitting phenomenon occurs. In such cases, the ANN loses its capacity to deal with input functions different from those it was trained with and it becomes too specific. In other words, the ANN is very good at learning and predicting when facing inputs similar to those it was trained with, but at the same time it is very bad when dealing with input slightly different from those exploited in the training phase.

Another important aspect concerns the complexity of the input functions. Results show a better learning and predictive phase for ANNs trained with a low level of complexity of input functions. For example the best results in terms of magnitude of the overall L^2-error have been obtained with discontinuous data which represent the simplest class of input functions we considered, indeed the latter is extremely simple being those functions constant and very regular apart from a single discontinuity in time.

References

[1] F. Regazzoni, L. Dedè, A. Quarteroni, Machine learning for fast and reliable solution of time-dependent differential equations, J. Comput. Phys, 2019

[2] L. Dedè, Optimal flow control for Navier–Stokes equations: drag minimization, J. Numer. Methods Fluids, 2007

[3] A. Quarteroni, Numerical models for differential problems, Springer, 2017

[4] A. Quarteroni, G. Rozza, Reduced order methods for modeling and computational reduction, Springer, 2014

[5] M. Guo, J.S. Hesthaven, Data-driven reduced order modeling for time-dependent problems, Comput. Methods Appl. Mech. Eng, 2019

[6] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys, 2018

[7] L. Pegolotti, Feamat, https://github.com/lucapegolotti/feamat

[8] F. Regazzoni, Model Learning, https://github.com/FrancescoRegazzoni/model-learning