ESAIM: PROCEEDINGS AND SURVEYS, September 2023, Vol. 73, p. 68-88
Virginie Ehrlacher, Damiano Lombardi, Olga Mula, Fabio Nobile, Tommaso Taddei

PARAMETRIZED NON INTRUSIVE SPACE-TIME APPROXIMATION FOR EXPLICIT DYNAMIC FEM APPLICATIONS

Hamza BOUKRAICHI[1, 2], Nassim RAZAALY[3], Nissrine AKKARI[1], Fabien CASENAVE[1] and David RYCKELYNCK[2]

Abstract. In the following work, a benchmark of different non-intrusive model reduction approaches is performed on an explicit dynamic contact $3D-$ problem. The main purpose of this work is to evaluate the stability of the reduced model with respect to time along with the precision of these approaches with respect to the true solutions of interest. These solutions are the prediction of displacement and velocity fields. The precision of these approaches is also evaluated with respect to the evolution of some materials parameters. Six parameters vary in this study and we would like to predict the whole transient fast dynamic impact response with respect to each parameters. To this end, several models are trained : Proper Orthogonal Decomposition (POD) and Deep convolutional Neural Network (DcNN), in addition, a vectorized version of Interpolation in Grassman Manifolds is proposed. The benchmark performed illustrate that using DcNN's allows to achieve the best precision and stability in predicting physical fields.

Introduction

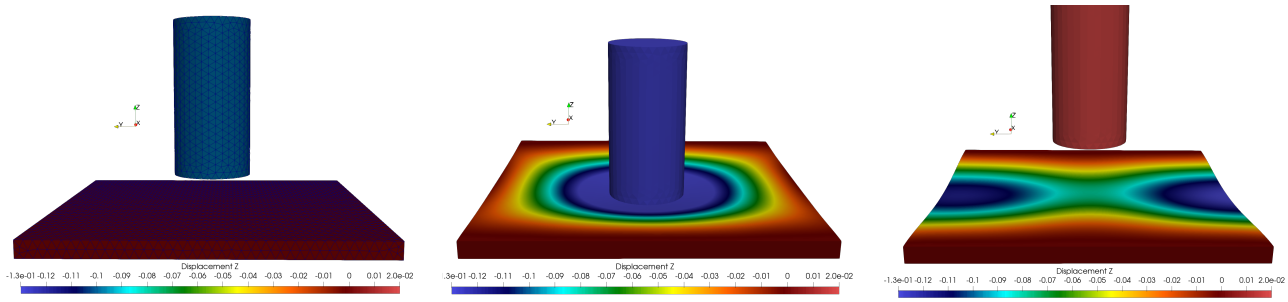
A large number of fast dynamic impact numerical simulations is crucial for optimization or uncertainty quantification and propagation problems, in order to study the influence of different scenarios and regimes in engineering problems. Dimension reduction approaches such as the Singular Values Decomposition [15] and model reduction techniques such as non-linear regressions based on Gaussian Process Regression [16] or Deep Learning [14], offer attractive alternatives to finite element impact problems in order to reduce the computational cost in these mutliple resolutions demanding tasks. Model Order Reduction based on the projection of the continuous High-Fidelity Partial Differential Equations (PDEs) on a reduced basis such as the one obtained by the Proper Orthognal Decomposition [6], is also a very important complementary option to finite element impact problems. These model order reduction approaches were considered in [2] where a Non-Negative Matrix Factorization is used to construct a reduced basis with the inequality constraints describing the contact forces. A nodeless superelement approach was proposed in [9] for the dual problem of the Lagrange multipliers, where the superlement dynamic is described by a set of orthogonal static modes. In [8], a multi-scale reduced order approach based on the LATIN method [12] and a mutli-grid solver is proposed for frictional contact problems. In [3] and [13], projection-based model order reduction is applied for parametrized variational inequalities in contact mechanics.

[1] Safran Tech, Etablissement Paris Saclay,Rue des Jeunes Bois-Chateaufort, 78114 Magny-Les-Hameaux, France
[2] MINES ParisTech, PSL University, MAT - Centre des matriaux, CNRS UMR 7633, BP 87 91003 Evry, France
[3] ISAE-ENSMA/Institut Pprime, Département FTC, 11 Boulevard Marie et Pierre Curie, 86073 Poitiers Cedex.

Article published online by EDP Sciences and available at https://www.esaim-proc.org or https://doi.org/10.1051/proc/202373068

In the following work, we are only interested in non-intrusive and non-projection-based model reduction approaches, i.e.without performing a projection of the continuous PDEs on a reduced basis. Non-intrusive approaches rely only on data driven method of modeling without using any physical information in the construction or the training of the reduced model, whereas intrusive methods use all the physical information available such as the exact PDE solved to construct their reduced model by projecting the equations on lower dimension spaces or constraining models with the equations in the training phase. The main motivation is the simplicity of the numerical framework in this case. We would like to compare different algorithms for constructing these reduction tasks, by resorting to non-linear regressions by Kriging, interpolation on Grassmann Manifolds, and Deep Learning. More precisely, the Gaussian Process Regression and the Grassmann interpolation are applied to predict the coefficients in the linear combination of Proper Orthognal Decomposition modes as an approximation of the fields of interest. As for the Deep Learning method for the non-linear regression, it is performed thanks to a deconvolutional neural network [17] as an approximation of the mapping between the parameters of interest and the fields of interest.

1. Problem Definition



(a) Before impact (b) Impact (c) After impact

Figure 1. Main steps of the simulation

In this paper we investigate the reducibility of a contact case in structural dynamics. We consider a cylinder of height $H_{cylinder}$ and diameter $D_{cylinder}$ which comes into contact with a plate, a rectangular cuboid of length, height and width $L_{plate}, W_{plate}, H_{plate}$ with an initial velocity v_z along the z-axis. Our objective is to study wave propagation and reflection phenomena on the plate following the shock over a discretized time grid denoted $T = (t_k)_{k\in[|1,n|]}$ with a time step Δt. This analysis requires the constructions of models to generate new parametrized simulations with a lower cost than solving the governing physical equations. Therefore we choose to apply data-driven methods such as metamodeling and deep learning models. Figure 1 shows the main steps of the finite element method simulation. Both solids are considered formed by uniform elastic and isotropic materials behaving linearly following the Hooke's law, which is written in the matrix form as

$$\boldsymbol{\sigma} = \frac{E}{1+\nu}\left(\boldsymbol{\varepsilon} + \frac{\nu}{1-2\nu}\mathrm{Tr}(\boldsymbol{\varepsilon})\right), \tag{1}$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ are respectively the second-order stress and strain tensors, E is the Young modulus, ν is the Poisson ratio, and Tr is the trace operator. Additionally, we denote the density of the materials by ρ, which plays a part in the response of the plate in such transient dynamical regimes. We introduce a unique parametrization of each simulation results :
let $\theta = (E_{cylinder}, \rho_{cylinder}, \nu_{cylinder}, E_{plate}, \rho_{plate}, \nu_{plate})$ a parameters vector that defines a unique finite element simulation denoted $S(\theta)$, S being the vector of the fields values on the grid of the plate. The objective here is to

construct regression models which function is to map the parameters vector θ to the simulation vector $S(\theta)$. As physical fields, we considered both displacement and velocity of the elements of the plate on the z-axis (u_z, v_z), both will be referred to as S in what follows.

Since both solids in contact have linear material and kinematic behavior, and we consider only the case of frictionless, adhesive-free normal contact, and considering a space discretization of both solids relying on the Finite Element discretization, the dynamic contact problem can be written as (see [1]) :

$$\begin{aligned} M\ddot{u} + Ku &= B^T\lambda \\ Bu - c &\geq 0 \\ \lambda^T(Bu - c) &= 0 \\ \lambda &\leq 0 \\ u(0) &= u_0 \\ \dot{u}(0) &= \dot{u}_0. \end{aligned} \tag{2}$$

Where a dot designates a time derivative, the first equation expresses the dynamic equilibrium of the two solids, the inequality constraint derives from the space discretization of the Hertz-Signorini-Moreau contact conditions which are enforced by introducing dual Lagrange multipliers. u is a space discretized displacement vector of the plaque and the cylinder degrees of liberty, M and K are respectively the block diagonal mass and stiffness matrices for the two solids. B is a signed boolean matrix which extracts from u the pair of dof governed by a contact condition, c is the vector of initial clearances, and λ is the vector of descretized Lagrange multipliers.

Then an explicit time discretization is performed in the dynamic case, finally solving the dynamic problem can be written as:

$$\forall n \geq 2, (u_n, \lambda_n) = \underset{v,\mu\leq 0}{\arg\min} \left[v^T(\frac{1}{\Delta t^2}M + K)v + \frac{1}{\Delta t^2}v^T M(-2u_{n-1} + u_{n-2}) - \mu^T(Bv - c) \right] \tag{3}$$

Where the superscript n designates the n-th time-step and Δt designates the difference between two consecutive time-steps.

Both solids are discretized in finite element meshes using 3D tetrahedric elements, 8546 nodes combined between the two solids, around 5000 nodes for the plaque mesh.

2. Methodology

The present Section describes the different methodologies used in this work.

2.1. Parametrized POD

In this Subsection, we recall the classic parametrized POD strategy for field prediction. A dataset of N samples is considered: $\{\theta_i, \mathbf{y}_i\}_{i\in[\![1,N]\!]}$, where $\theta_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in \mathbb{R}^n$ represent respectively a vectorial parameter of moderate dimensionality and a high-dimensional output. The objective is to build a predictive model $\theta \to \tilde{\mathbf{y}}(\theta)$. The methodology is decomposed in the following steps:

- Orthonormal basis generation. The data matrix $M = [\mathbf{y}_1, ..., \mathbf{y}_N] \in \mathbb{R}^{n\times N}$ is considered. In practice, note that the number of samples N is largely below the output dimensionality n. Its SVD decomposition yields
$$M = U\Sigma V^T, \tag{4}$$
with $U = [\boldsymbol{u}_1, ..., \boldsymbol{u}_N] \in \mathbb{R}^{n\times N}$ containing the spatial orthonormal eigenvectors satisfying $\boldsymbol{u}_i^T\boldsymbol{u}_j = \delta_{ij}$, the eigenvalues matrix $\Sigma = \mathrm{Diag}(\sigma_1, ...\sigma_N)$ and $V = [\boldsymbol{v}_1, ..., \boldsymbol{v}_N] \in \mathbb{R}^{N\times N}$, $\boldsymbol{v}_i^T\boldsymbol{v}_j = \delta_{ij}$.

Another strategy implies the covariance matrix eigen-decomposition: $C = M^T M = V\Sigma^2 V^T$, with $\boldsymbol{u}_i = \dfrac{M\boldsymbol{v}_i}{\sigma_i}$.

- Number of modes selection. Given an accuracy threshold ϵ, the number of modes r selected is the smallest integer satisfying:

$$\frac{\sum_{i=1}^{r}\sigma_i^2}{\sum_{i=1}^{N}\sigma_i^2} > 1-\epsilon. \tag{5}$$

The prediction is then sought in the span of the reduced orthonormal basis $\mathcal{U}_r = [\boldsymbol{u}_1, ...\boldsymbol{u}_r]$.

- Scalar coefficients in the reduced basis. The prediction is expressed as

$$\tilde{\mathbf{y}}(\theta) = \sum_{i=1}^{r} h_i(\theta)\boldsymbol{u}_i, \tag{6}$$

with the scalar functions $\theta \to h_i(\theta)$ satisfying

$$h_i(\theta_k) = < \mathbf{y}_k, \boldsymbol{u}_i >, \tag{7}$$

$<,>$ being the canonical scalar product in $\mathbb{R}^n$.
Each of the coefficients h_i is trained using a regression method (Kriging, RBF, ANN, random forest,...) based on the DoE $\{\theta_k, < \mathbf{y}_k, \boldsymbol{u}_i >\}_k$, yielding a predictive function $\theta \to \tilde{h}_i(\theta)$.

- Prediction. The sought predictive model finally reads

$$\tilde{\mathbf{y}}(\theta) = \sum_{i=1}^{r} \tilde{h}_i(\theta)\boldsymbol{u}_i. \tag{8}$$

2.2. Parametrized Space-Time POD

In the context of space-time POD, the dataset is given as follows: $\{\theta_i, S_i\}_{i\in[\![1,N]\!]}$, where $\theta_i \in \mathbb{R}^d$ and $S_i = [y(\theta_i, t_1), ..., y(\theta_i, t_m)] \in \mathbb{R}^{n\times m}$. Each matrix S_i contains the snapshots for all time steps. For this case, we propose to consider the time as an independent parameter, and recast the dataset to fit the framework of classic parametrized POD (Subsection 2.1: $\{\hat{\theta}_k, \hat{\mathbf{y}}_k\}_{k\in[\![1,mN]\!]}$ where for each $k \in [\![1, mN]\!]$ corresponds to a couple of integers $(i, l) \in [[\![1, N]\!]] \times [[\![1, m]\!]]$, with $\hat{\theta}_k = (\theta_i, t_l)$ and $\hat{\mathbf{y}}_k = y(\theta_i, t_l)$. The previous framework is then used for training and prediction.

2.3. Parametrized Space-Time POD: Interpolation on Grassmann Manifolds

Friderikos et al. proposed a framework for parametrized space-time POD based on interpolation on Grassmann manifolds [7], suitable for scalar parameters. The underlying idea is to interpolate POD basis associated with a limited number of training points on Grassmann manifolds, so as to obtain a robust non-intrusive ROM corresponding to a target parameter, separating both reduced spatial and temporal basis. A methodology to extend this algorithm for vectorial input parameters is proposed. The main steps of the algorithm are summarized in Subsubsection 2.3.1. The proposed modifications for the extension to the vectorial case is given in Subsubsection 2.3.2.

2.3.1. *Scalar version (from [7])*

- *Dataset* : Similarly to Subsection 2.2, the dataset is given as follows: $\{\theta_i, S^{(i)}\}_{i\in[\![1,N]\!]}$, where $\theta_i \in \mathbb{R}$ and $S_i = [y(\theta_i, t_1), ..., y(\theta_i, t_m)] \in \mathbb{R}^{n\times m}$. This version is coined 'scalar' because it is restricted parameters θ taken as real numbers only.

- *Set a number of modes r* : We propose the following strategy to set the number of modes r. We fix a given threshold accuracy $\epsilon = 10^{-12}$, then perform a SVD decomposition on the mean output matrix. The number of modes r is then chosen according to the eigenvalues decay, similarly to POD (Subsection 2.1)

$$\bar{S} = \frac{1}{N}\sum_{i=1}^{N} S^{(i)} = \bar{U}\bar{\Sigma}\bar{V}^T, \tag{9}$$

with $\bar{\Sigma} = \text{Diag}(\sigma_1, ..., \sigma_m)$. r is the smallest integer satisfying:

$$\frac{\sum_{i=1}^{r}\bar{\sigma}_i^2}{\sum_{i=1}^{m}\bar{\sigma}_i^2} > 1 - \epsilon. \tag{10}$$

- *Oriented SVD for each output matrix, with mode extraction of order p* : For each output matrix $S^{(i)}$, an oriented SVD [7] decomposition is performed:

$$S^{(k)} = \Phi^{(k)}\Sigma^{(k)}{\Psi^{(k)}}^T. \tag{11}$$

The r first modes are extracted, leading to matrices $\Phi_r^{(k)} \in \mathbb{R}^{n\times r}, \Sigma_r^{(k)} \in \mathbb{R}^{r\times r}, \Psi_r^{(k)} \in \mathbb{R}^{m\times r}$. The following reconstructed matrix output is also evaluated:

$$S_r^{(k)} = \Phi_r^{(k)}\Sigma_r^{(k)}{\Psi_r^{(k)}}^T, \tag{12}$$

with $k \in [\![1, N]\!]$.

- *Interpolation on compact Stiefel manifolds* : Using the datasets $\{\theta_k, \Phi_r^{(k)}\}_{k\in[\![1,N]\!]}$ and $\{\theta_k, \Psi_r^{(k)}\}_{k\in[\![1,N]\!]}$, define a map $\theta \to \tilde{\Phi}(\theta) \in \mathbb{R}^{n\times r}$ and (resp.) $\theta \to \tilde{\Psi}(\theta) \in \mathbb{R}^{m\times r}$. Let us denote by $\{\theta_k, Y_k\}_{k\in[\![1,N]\!]}$ such a generic dataset of orthonormal basis, assuming $Y_k \in \mathbb{R}^{n\times r}$. A set of matrices $\{Z_k\}_k$ is first evaluated:

$$\begin{cases} Z_1 = 0_{\mathbb{R}^{n\times r}}, \\ Z_k = U_k \tan^{-1}(\Sigma_k) V_k^T, \quad k > 1, \end{cases} \tag{13}$$

where the orthonormal matrices $U_k \in \mathbb{R}^{n\times r}$, $V_k \in \mathbb{R}^{r\times r}$ and the diagonal matrix $\Sigma_k \in \mathbb{R}^{r\times r}$ are computed by performing the SVD on the following matrix:

$$Y_k(Y_1^T Y_k)^{-1} - Y_1 = U_k \Sigma_k V_k^T.$$

An interpolator $\theta \to Z(\theta)$ using linear Lagrange polynomial basis functions $(L_i)_{i\in[|1,N|]}$, suitable for scalar parameters, is then defined, as well as $\theta \to U(\theta), \Sigma(\theta), V(\theta)$ obtained by applying a SVD decomposition on $Z(\theta)$:

$$Z(\theta) = \sum_{i=1}^{N} L_i(\theta) Z_i = U(\theta)\Sigma(\theta)V^T(\theta), \tag{14}$$

The final interpolator $\theta \to \tilde{Y}(\theta)$ finally reads:

$$\tilde{Y}(\theta) = \left[Y_1 V(\theta)\cos(\Sigma(\theta)) + U(\theta)\sin(\Sigma(\theta))\right] V(\theta)^T. \tag{15}$$

- *Square matrix of the mixed part: Interpolation* : The following matrices are evaluated:

$$M_i = \tilde{\Phi}(\theta_i)^T S_r^{(k)} \tilde{\Psi}(\theta_i) \in \mathbb{R}^{r\times r}, i \in [\![1, N]\!]. \tag{16}$$

An interpolation $\theta \to \tilde{M}(\theta)$ suitable for scalar parameters similar to Eq. (14) is built:

$$\tilde{M}(\theta) = \sum_{i=1}^{N} L_i(\theta) M_i. \tag{17}$$

- *Final Predictor* : The predictor of the entire solution reads:

$$\tilde{S}(\theta) = \tilde{\Phi}(\theta)\tilde{M}(\theta)\tilde{\Psi}(\theta)^T \in \mathbb{R}^{n\times m}. \tag{18}$$

Algorithm 1 Interpolation on Grassman manifolds algorithm.

Require: Dataset $\{\theta_i, S^{(i)}\}_{i\in[\![1,N]\!]}$, precision threshold ϵ

Perform SVD on Dataset $\bar{S} = \frac{1}{N}\sum_{i=1}^{N} S^{(i)} = \bar{U}\bar{\Sigma}\bar{V}^T$, $\bar{\Sigma} = \text{Diag}(\sigma_1, ..., \sigma_m)$.

Determine r the smallest integer satisfying $\frac{\sum_{i=1}^{r}\bar{\sigma}_i^2}{\sum_{i=1}^{m}\bar{\sigma}_i^2} > 1-\epsilon$.

for $k \in [|1, n|]$ **do**

Perform SVD on $S^{(k)} = \Phi^{(k)}\Sigma^{(k)}\Psi^{(k)^T}$

Perform SVD on $Y_k(Y_1^TY_k)^{-1} - Y_1 = U_k\Sigma_kV_k^T$, Y_k being respectively $\Psi_r^{(k)}$ or $\Phi_r^{(k)}$

end for

Construct the matrices $Z_1 = 0_{\mathbb{R}^{n\times r}}$, $Z_k = U_k\tan^{-1}(\Sigma_k)V_k^T, k > 1$ for respectively $\Psi_r^{(k)}$ or $\Phi_r^{(k)}$

for $k \in [|1, n|]$ **do**

Use Lagrange polynomial basis functions to obtain a mapping for $(\theta_i, Z(\theta_i))_{i\in[\![1,N]\!]}$

$\tilde{Y}(\theta) = [Y_1V(\theta)\cos(\Sigma(\theta)) + U(\theta)\sin(\Sigma(\theta))]\, V(\theta)^T$ Y_k being respectively $\Psi_r^{(k)}$ or $\Phi_r^{(k)}$

$M_i = \tilde{\Phi}(\theta_i)^T S_r^{(k)}\tilde{\Psi}(\theta_i) \in \mathbb{R}^{r\times r}, i \in [\![1, N]\!]$

$\tilde{M}(\theta) = \sum_{i=1}^{N} L_i(\theta)M_i$

Prediction for θ $\tilde{S}(\theta) = \tilde{\Phi}(\theta)\tilde{M}(\theta)\tilde{\Psi}(\theta)^T$

end for

2.3.2. *Vectorial version (contribution)*

In this case, the dataset reads: $\{\theta_i, S_i\}_{i\in[\![1,N]\!]}$, where $\theta_i \in \mathbb{R}^d$ and $S_i = [y(\theta_i, t_1), ..., y(\theta_i, t_m)] \in \mathbb{R}^{n\times m}$. This version is coined 'vectorial' because the parameters θ are now taken as real vectors.

The modifications proposed are performed on Equations (14) and (17): they permit to obtain mappings $\theta \to \tilde{Y}$ and $\theta \to \tilde{M}$ based on datasets $\{\theta_i, Y_i\}_{i\in[\![1,N]\!]}$ and $\{\theta_i, M_i\}_{i\in[\![1,N]\!]}$ respectively. As $M_i \in \mathbb{R}^{r\times r}$ with r relatively small, a regression based on random forests (using the Python package scikit learn) is directly used on the components of the matrix.

Instead, as $Y_i \in \mathbb{R}^{q\times r}$ (with $q = n$ or $q = m$) represents matrices of spatial or temporal eigenvectors, possibly very large, a different strategy is adopted. As $Y_i = [Y_i^{(1)}, ..., Y_i^{(r)}]$ with $Y_i^{(l)} \in \mathbb{R}^q, l \in [\![1, r]\!]$, parametrized POD regressors as introduced in Subsection 2.1 are used to obtain independent regressors for each column of Y_i, denoted as $\tilde{Y}^{(l)}(\theta)$ built using the DoE $\{\theta_i, Y_i^{(l)}\}_{i\in[\![1,N]\!]}$. The regressor on the full eigenvector matrix reads:

$$\tilde{Y}(\theta) = [\tilde{Y}^{(1)}(\theta), ..., \tilde{Y}^{(r)}(\theta)]. \tag{19}$$

2.4. **Deep convolutional neural regressor**

A Deep convolutional Neural Regressor (*DcNR*) consists in learning to generate the physical data S over a grid with the parameters vector θ as an input. As indicated by its name, the internal structure of this network is formed by a succession of transposed convolutional layers of adequate dimensions in order to obtain a regression model of the physical field in the adequate size. The objective function in this case being:

$$\min_{\gamma} \sqrt{\mathop{\mathbb{E}}_{(t,\theta)} ||S(\theta, t) - N(\gamma, \theta, t)||_2^2}, \tag{20}$$

In practice, the empirical mean is used to approximate the expectation:

$$\min_{\gamma} R(\gamma, \Theta, T) = \sqrt{\sum_{(t,\theta)\in(T\times\Theta)} \frac{1}{|T||\Theta|} ||S(\theta,t) - N(\gamma,\theta,t)||_2^2}, \tag{21}$$

where N denotes the neural network, γ its trainable weights, Θ the training set of parameters vectors and $|\Theta|$ its cardinal, T the the set of time steps and $|T|$ its cardinal. The use of optimal Latin Hyper Cube sampling with a significantly large data set ensures that the introduced bias due to the use of the average to approximate the expectation is kept reasonably low. We trained two types of DcNR, the first type generating the whole time series of the simulation at each call and the second generating only one time step at each call, thus having the time step t as an additional parameter. Both networks have the same architecture except for the number of filters at each feature map. Both architectures start with a linear upscaling of the parameter vector then a succession of 3D transposed convolution used for physical values in [4] for the first time, batch normalization [10] and hyperbolic tangent activation. We used Adam ([11]) as the optimization algorithm for the descent step.

TABLE 1. Network architecture - Transposed convolutions

Layer	Kernel Size	Stride	Padding	n-filters Full	n-filters time parametrized
1	(3,3,3)	(1,1,1)	(0,0,0)	1024	128
2	(3,3,3)	(1,1,1)	(0,0,0)	512	64
3	(3,3,4)	(1,1,1)	(0,0,0)	256	32
4	(5,5,5)	(1,1,1)	(0,0,0)	196	16
5	(4,4,5)	(2,2,1)	(0,0,0)	128	8

Algorithm 2 Training of DcNR. All experiments in this paper used the default values $\lambda = 0.0001$, $m = 2000$ simulations, $\epsilon_0 = 10^{16}$, $\epsilon_{obj} = 10^{-3}$, Epochs=10000.

Require: λ learning rate, m batch size, ϵ_0 initial error of model, ϵ_{obj} error objective, Epochs number of maximum epochs authorized, γ DcNr weights, M number of training simulations.

```
s ← M/m
i ← 1
while (i ≤ Epochs) or (ε_obj ≤ ε_i) do
  ε_i ← 0
  for j ∈ [|1, s|] do
    Sample (t_k, θ_k)_{k=1}^m ∈ (T_batch × Θ_batch)
    γ ← γ − λAdam(R(γ, Θ_batch, T_batch))
    ε_i ← ε_i + R(γ, Θ, T)
  end for
  i ← i + 1
  ε_i ← ε_i / s
end while
```

For reproducibility purposes, both Table 1 and Algorithm 2 show the architectures and the training algorithm used in all experiments.

3. Numerical Examples

3.1. Data Sampling

In this section we present the data range used for sampling and generating data for the training and testing phase. Values were chosen as: $L_{plate} = 8m$, $W_{plate} = 4m$, $H_{plate} = 0.5m$, $H_{cylinder} = 4m$, $D_{cylinder} = 1m$, $N_T = 100$, $\Delta t = 4 \times 10^{-2}s$, and $v_z = 100m/s$.

The variable parameters identified in Section 1 are sampled following Table 2 values with a latin hypercube sampling framework. We sampled $N_{Training} = 100$ values for the training data set and $N_{Testing} = 25$ values for the testing data set.

Table 2. Parameters sampling

P	Mean Value	Variation (%)	Min Value	Max Value
Young modulus$E$	2.2×10^{11} MPa	20%	1.7×10^{11} MPa	2.64×10^{11} MPa
Densityν	$7800kg/m^3$	20%	$6240kg/m^3$	$9360kg/m^3$
Poisson's ratio ρ	0.3	20%	0.24	0.35

We define Ω as the discretized space of the plate and Ω_{z^*} the 2D surface resulting from the intersection of the plate grid and the plan of equation $z = z^*$. As a quantity of interest, we choose to train our models to predict displacement and velocity on the z-axis for the plate, and to compare our models, we fix two time steps $t_1 = 0.01s$ and $t_2 = 0.03s$, two parameters vectors $\theta_1 = \left(2.12 \times 10^{11}, 7.87 \times 10^3, 3.13 \times 10^{-1}, 2.20 \times 10^{11}, 6.47 \times 10^3, 2.72 \times 10^{-1}\right)$ and $\theta_2 = \left(1.71 \times 10^{11}, 6.68 \times 10^3, 3.27 \times 10^{-1}, 2.31 \times 10^{11}, 6.86 \times 10^3, 3.41 \times 10^{-1}\right)$, two points $p_1 = (1.66, 0.88, 0.2875)$ and $p_2 = (5, 3.33, 0.2875)$ and $\Omega_{z=0.2875}$ as parametrized points to asses the precision of our models to predict values of interest.

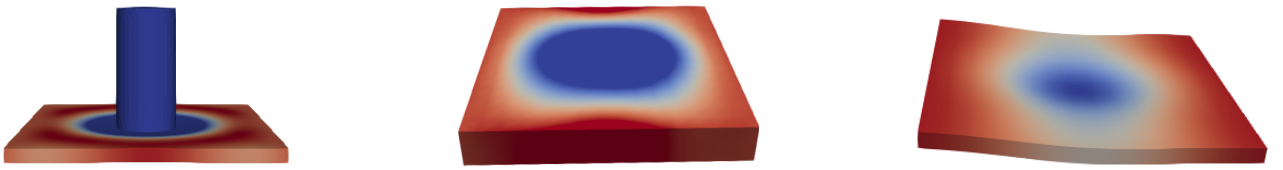
We define a relative error indicator in order to quantify the precision of our metamodels, noted η. For a metamodel M, a parameter vector θ, a space point $p = (x, y, z)$, and a time value t:

$$\eta_p(M, \theta, p, t) = \frac{|M(\theta, p, t) - S(\theta, p, t)|}{\sqrt{\mathop{\mathbb{E}}_{p \in \Omega} |S(\theta, p, t)|^2}}. \tag{22}$$

Then we define the global error indicator to compare models:

$$\eta(M, \theta, t) = \mathop{\mathbb{E}}_{p \in \Omega}[\eta_p(M, \theta, p, t)]. \tag{23}$$

3.2. Data preprocessing



(a) Data generation on full mesh

(b) Projection on containing box

(c) Inverse projection on initial mesh

Figure 2. Data processing steps

To ensure that our simulation data can be processed by convolutional neural networks, a projection phase of the data on an Euclidean 3D grid is necessary, we choose the smallest box that contains the whole range of displacement of the original grid, using the Finite Element basis functions we construct the projection operator from the initial mesh onto the containing box in addition to the inverse projection operator. Then after training our models and computation of test values, an inverse projection operation is done on the original grid. Figure 2 shows a visualization of this framework. Besides, since neural networks are more suited to learn from continuous fields, we used extrapolation, using an extension of the finite element basis functions, over the area were no motion of the grid is detected for each time step, see Figure 3. It prevents the neural network from having to learn the sharp discontinuities of the boundaries of the plate. After the projection operation is performed we scale the physical data using a standard affine scaler to standardize data by removing the mean and scaling to the unit variance.

(a) Extrapolation (b) Zero-Fill

FIGURE 3. Extrapolation vs zero-fill projection

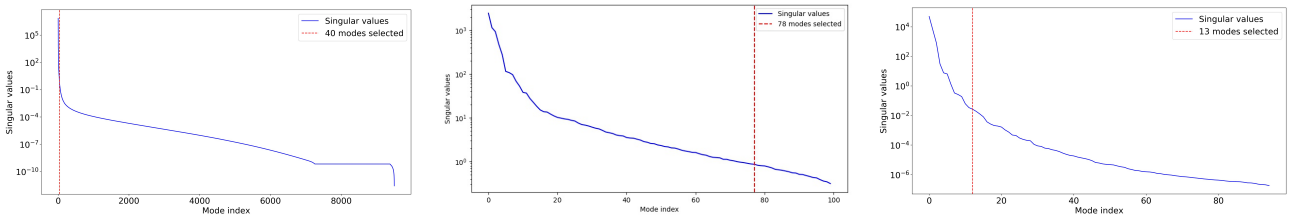
3.3. **Trained metamodels**

To provide extensive comparisons, we train multiple versions of each model described in Section 2:

- **POD**: We train different POD models with multiple metamodels over the orthogonal projection coefficients (random forest, Gaussian process and linear). We choose to keep POD models with random forest considering it held the best trade-off between precision and computational cost for our problem:
  - *POD_p*: it takes as input the parameter vector p and outputs the value of S over all the area of interest and for all time steps.
  - *POD_pt*: Time-Space POD, it takes as input the parameter vector p and the time value t and outputs the value of S over all the area of interest at the instant t.
- **Stief** : We trained a generalized model for interpolation on Grassman manifolds using also a random forest model for inference over parameters.
- **DcNR**: We train multiple DcNR :
  - *NN*: it takes as input the parameter vector p and outputs the value of S over all the area of interest and for all time steps.
  - *NN_t*: it takes as input the parameter vector p and the time value t and outputs the value of S over all the area of interest at the instant t.

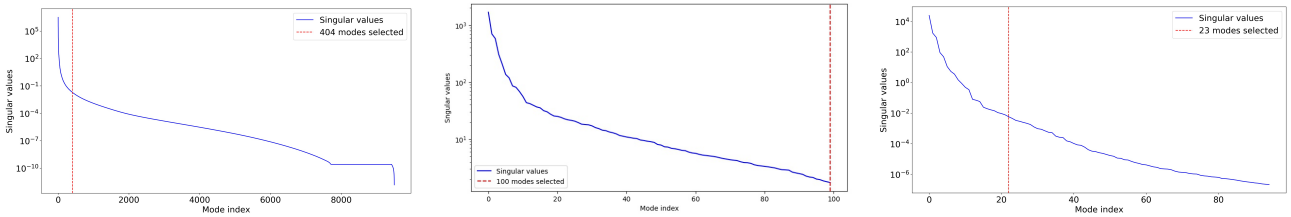
Since the error threshold for the root mean squared error was set as 10^{-3} when training the DcNR, we train our linear models (POD metamodels) with an error threshold of 10^{-6} and with the same projected and scaled data to ensure fair comparison.

3.4. **Results**



(a) POD_pt displacement (b) POD_p displacement (c) Stief displacement

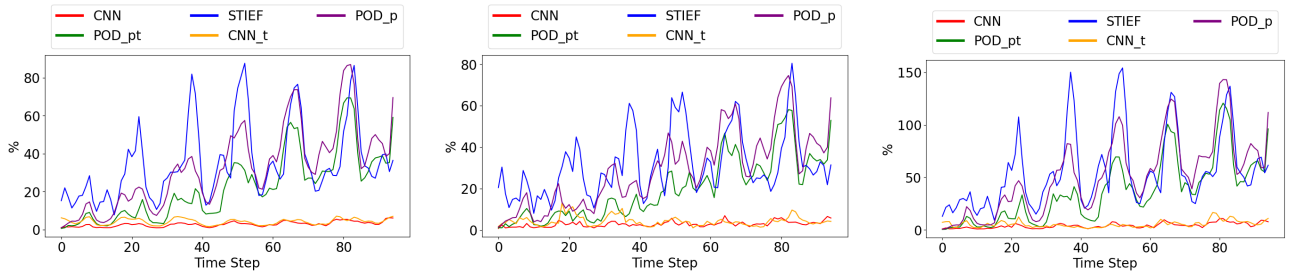
FIGURE 4. Singular values decay and modes selection for linear models - Displacement



(a) POD_pt velocity (b) POD_p velocity (c) Stief velocity

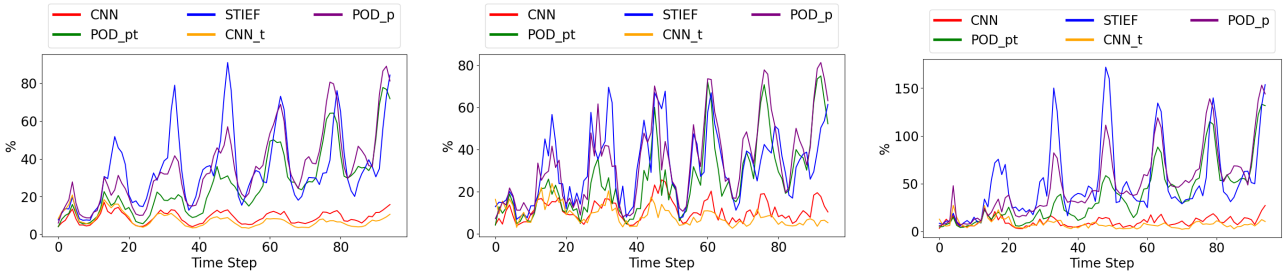
FIGURE 5. Singular values decay and modes selection for linear models - Velocity

Figures 4 and 5 show the different singular values decay for each linear model, all models require more modes to fit the velocity which can be explained by the fact that velocity data are less linearly reducible than displacement data. The POD_{pt} model selects an inferior ratio of modes than the POD_p model, considering both parametric and time-space information in the construction of the basis gives better approximation properties to the linear model. Nonetheless, the cost of singular value decomposition operation done for the construction of the POD_{pt} model is more important and, the POD_{pt} corresponding metamodel must learn coefficients in a high dimension ($m \times r$, where m is the number of the time steps and r are the retained POD modes) thus being more difficult to train.



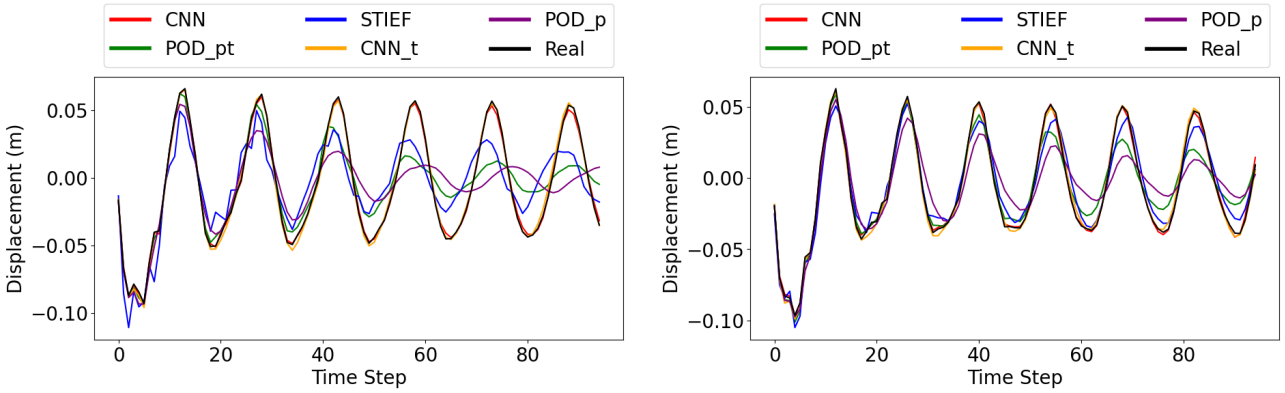
(a) Global error indicator (b) Pointwise error indicator on p_1 (c) Pointwise error indicator on p_2

FIGURE 6. Error indicator η for displacement averaged over test



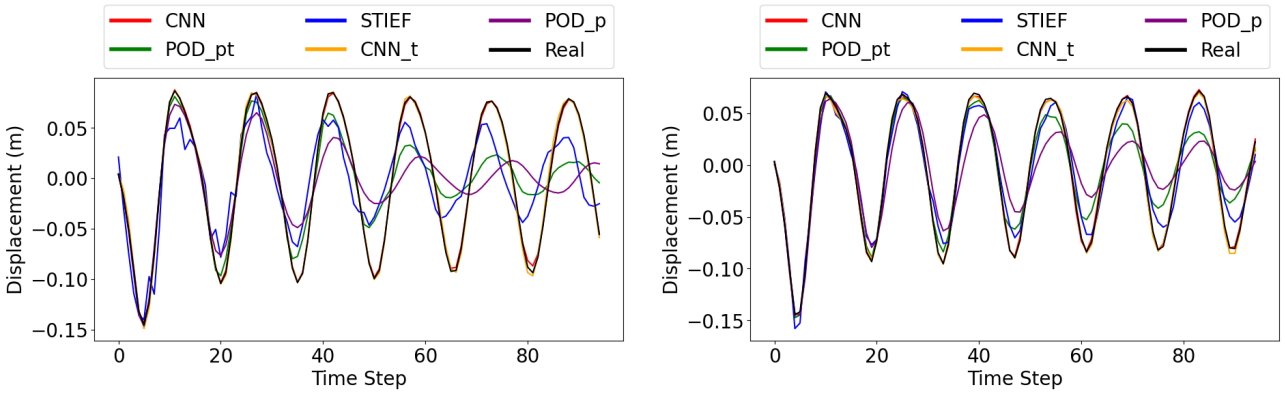
(a) Global error indicator (b) Pointwise error indicator on p_1 (c) Pointwise error indicator on p_2

FIGURE 7. Error indicator η for velocity averaged over test



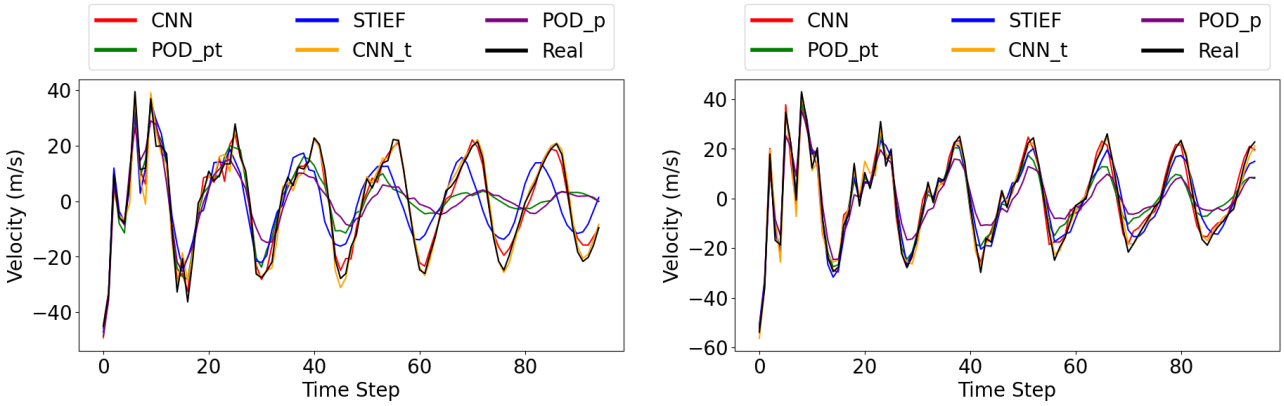
(a) (p_1, θ_1) (b) (p_1, θ_2)

FIGURE 8. Models prediction on p_1 - Displacement



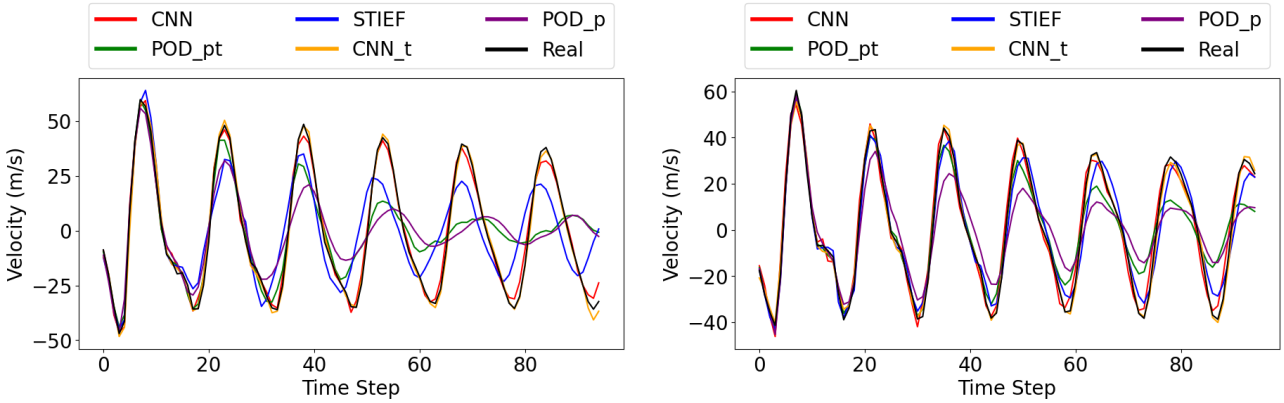
(a) (p_2, θ_1) (b) (p_2, θ_2)

FIGURE 9. Models prediction on p_2 - Displacement



(a) (p_1, θ_1) (b) (p_1, θ_2)

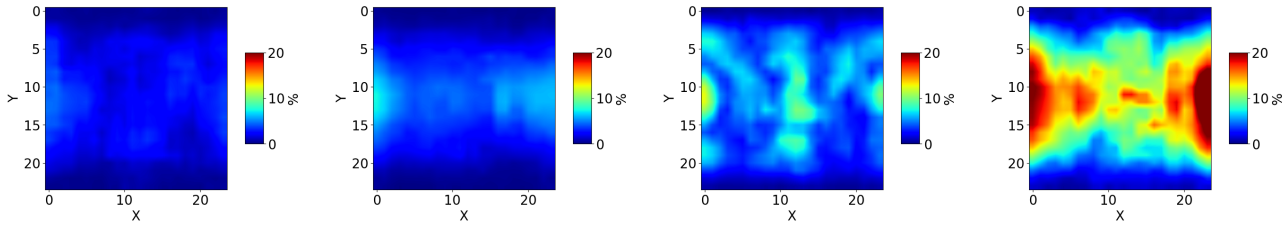
FIGURE 10. Models prediction on p_1 - Velocity



(a) (p_2, θ_1) (b) (p_2, θ_2)

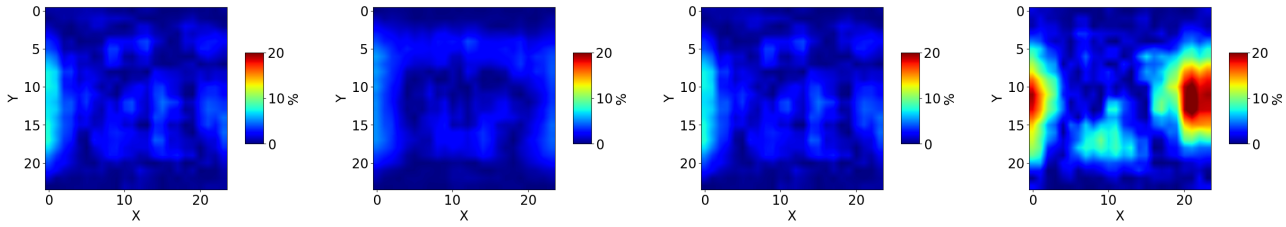
FIGURE 11. Models prediction on p_2 - Velocity

Figures 6 and 7 show that best precision is achieved by the neural networks, while within the linear models, the POD_{pt} holds the best behavior, which is also shown by the figures 8, 9, 10 and 11 where the velocity is, as expected, harder to fit by all models. The intuition of comparing between a model whose objective is to learn the whole physical simulation CNN and a model whose objective is to learn a regression scheme time step per time step CNN_t is to test the hypothesis that a model who has access to a full time interval of the simulation captures better the background physics. Nonetheless the CNN_t results shows slighlty better results since CNN model is also more difficult to train as the number of trainable parameters increases with the size of the time interval considered. Within those figures, a time dependency of the accuracy can be seen over most models. This time dependency is reduced when using the neural networks, and especially in the model CNN_t.



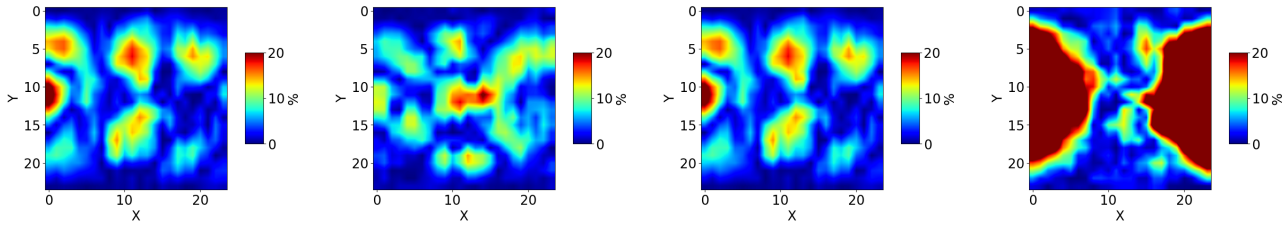
(a) Displacement t_1 (b) Displacement t_2 (c) Velocity t_1 (d) Velocity t_2

FIGURE 12. CNN error on S averaged on test



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

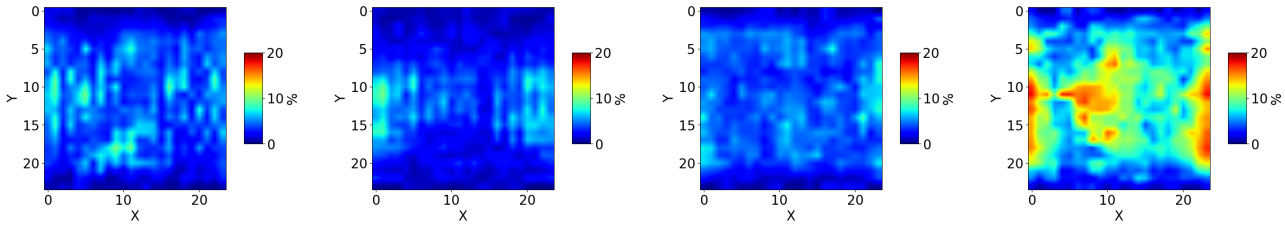
FIGURE 13. CNN error on S for displacement



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

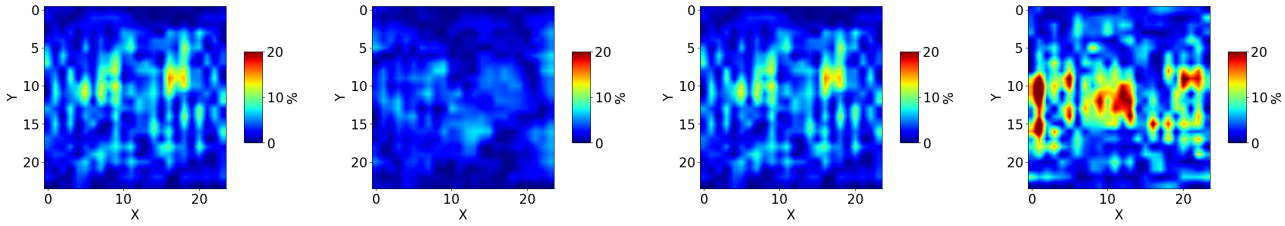
FIGURE 14. CNN error on S for velocity

Figures 12, 13 and 14 shows an introduced noise and error by the neural networks following the structure of the multiple convolution. A way of correcting this noise was developed in [5] by using a physical submodel over an area of interest that takes as an input boundary conditions learned by a neural network; this method was not used in this paper. We can also see a time dependency of the error, but globally the model achieves good accuracy.



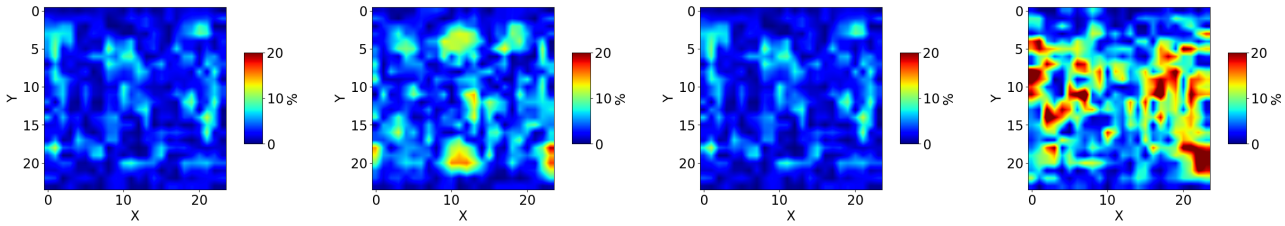
(a) Displacement t_1 (b) Displacement t_2 (c) Velocity t_1 (d) Velocity t_2

FIGURE 15. CNN_t error on S averaged on test



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

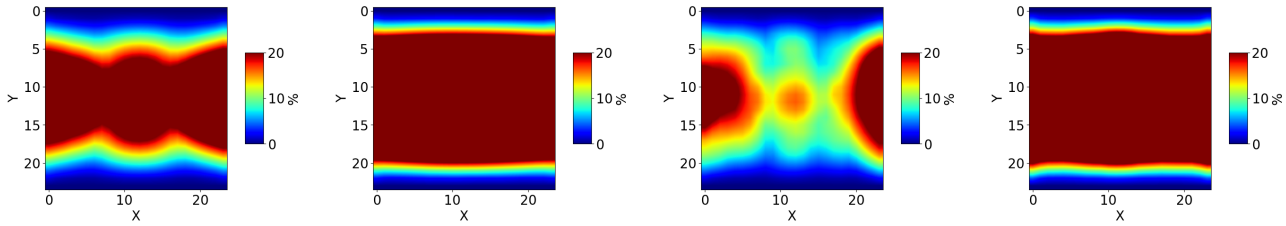
FIGURE 16. CNN_t error on S for displacement



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

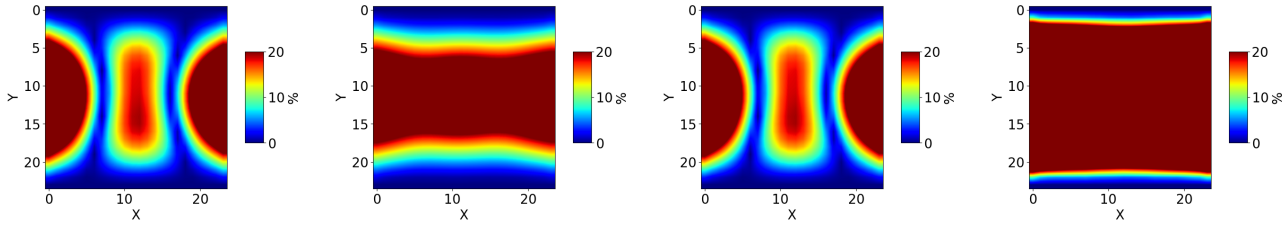
FIGURE 17. CNN_t error on S for velocity

Figures 15, 16 and 17 shows that the introduced noise and error by the neural network architecture is higher in the CNN_t model but the time dependency on the error was completely eliminated, globally the model achieves better accuracy results than the CNN model (5% of error at t_2 for CNN_t vs around 8% for CNN).



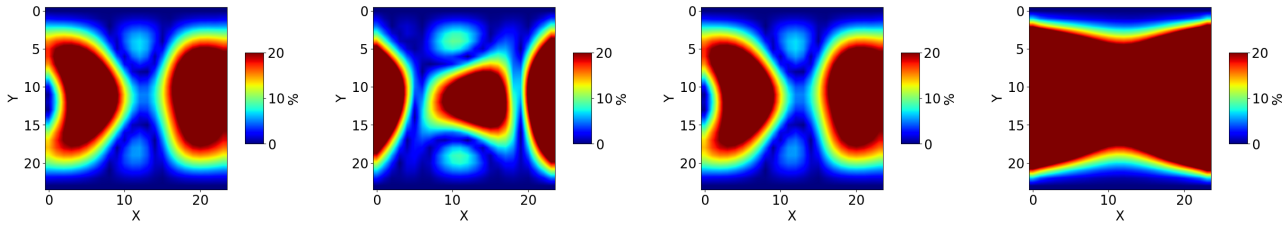
(a) Displacement t_1 (b) Displacement t_2 (c) Velocity t_1 (d) Velocity t_2

FIGURE 18. POD_p error on S averaged on test



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

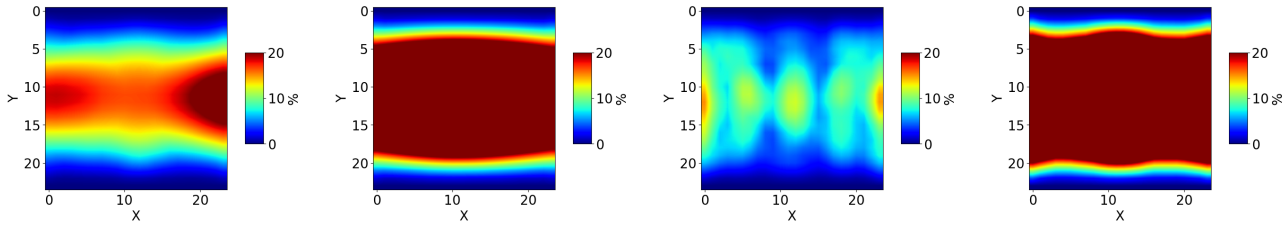
FIGURE 19. POD_p error on S for displacement



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

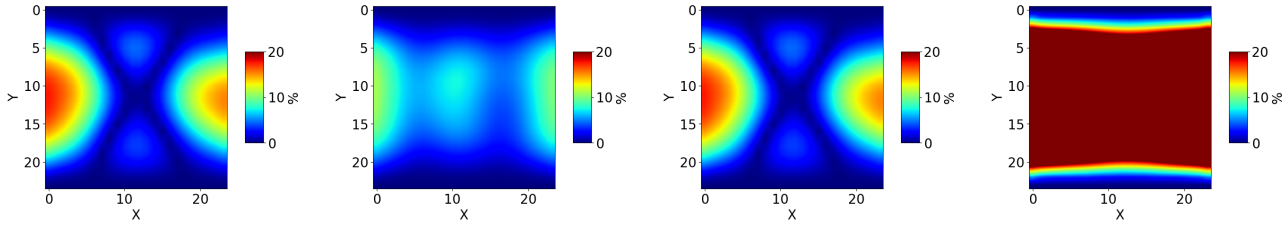
FIGURE 20. POD_p error on S for velocity

Figures 18, 19 and 20 shows that the error of the POD_p model has a physical and structured shape and a poor precision. Time dependency of the error is highly present for this model even though it cannot be seen over these figures but it is present on figures 6 and 7.



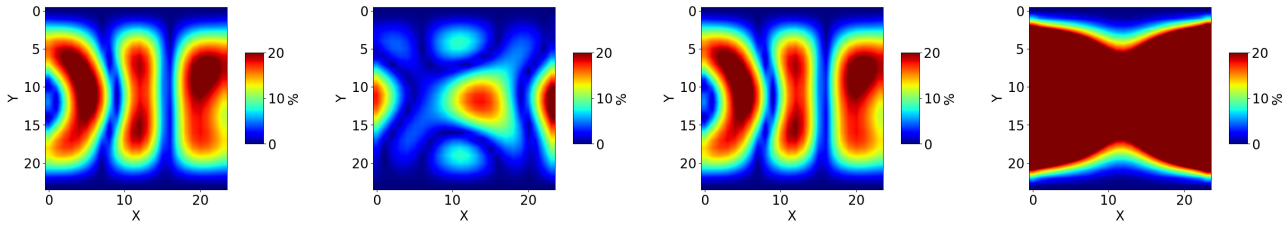
(a) Displacement t_1 (b) Displacement t_2 (c) Velocity t_1 (d) Velocity t_2

FIGURE 21. POD_pt error on S averaged on test



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

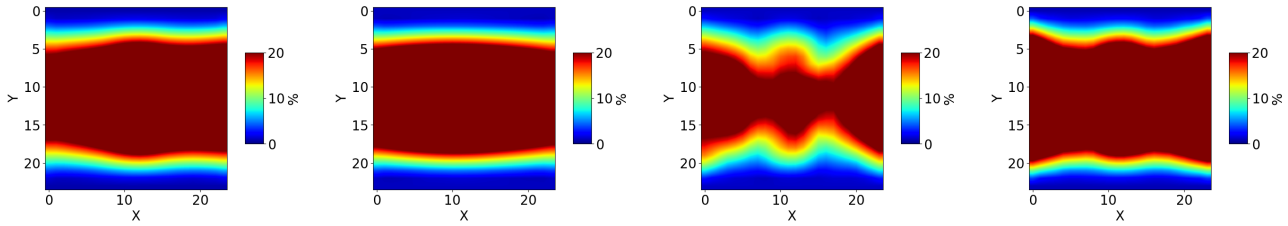
FIGURE 22. POD_pt error on S for displacement



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

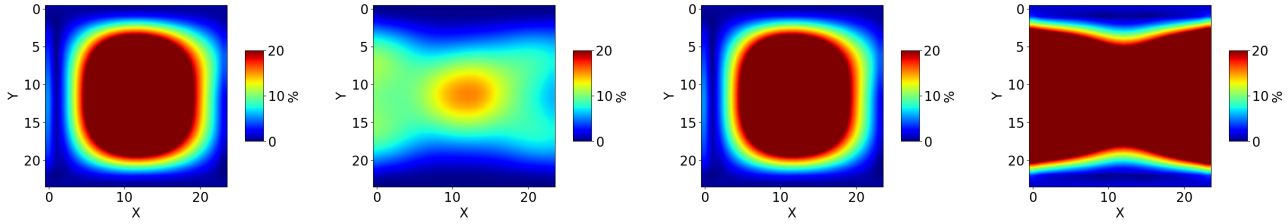
FIGURE 23. POD_pt error on S for velocity

Figures 21, 22 and 23 shows that the error of the POD_{pt} model has also a physical and structured shape, but a better precision over the test set than the POD_p model (10% of error at t_1 for POD_{pt} vs around 16% for POD_p). Time dependency of the error is highly present for this model.



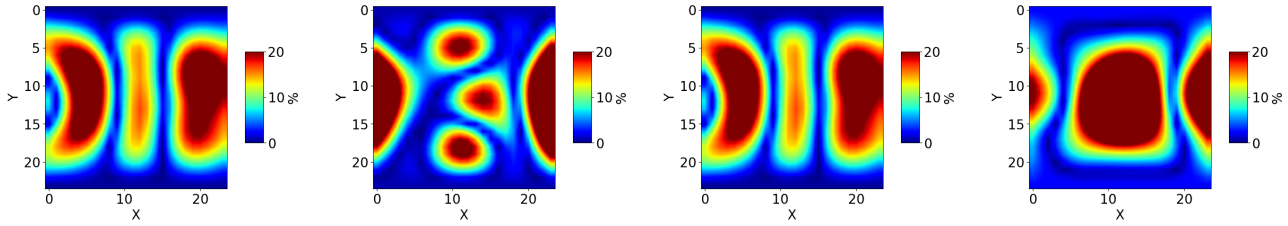
(a) Displacement t_1 (b) Displacement t_2 (c) Velocity t_1 (d) Velocity t_2

FIGURE 24. Stief error on S averaged on test



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 25. Stief error on S for displacement



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 26. Stief error on S for velocity

Figures 24, 25 and 26 shows that the error of the Stieff model is the highest within all models and has a structured shape on the error. Nonetheless our generalization approach for Grassman manifolds interploation shows some good prediction properties for some parametric values and some region of the grid. So, we can conclude that our approach is still not optimal and requires more investigation to generalize the scalar version.

4. CONCLUSION

In this paper we investigate the capacity of linear models relying on modal analysis to construct a regression model for a non-reducible problem, we also showed the benefits and the costs of considering both parametric and time-space information in the modal analysis step. We also propose an approach to generalize Grassman manifold interpolation from scalar output to vector ones, but it still requires further investigations. We compare all methods to deep convolutional neural regressor and we illustrate that for contact cases such as the one

investigated, linear methods behave very poorly and it is recommended to use non-linear data driven methods such as DcNR. We also illustrate that having the time step as a parameter of the DcNR helps reducing the dependency of the error with respect to the time.

References

[1] Maciej Balajewicz, David Amsallem, and Charbel Farhat. Projection-based model reduction for contact problems, 2015.

[2] Maciej Balajewicz, David Amsallem, and Charbel Farhat. Projection-based model reduction for contact problems. *International Journal for Numerical Methods in Engineering*, 106(8):644–663, 2016.

[3] Amina Benaceur, Alexandre Ern, and Virginie Ehrlacher. A reduced basis method for parametrized variational inequalities applied to contact mechanics. *International Journal for Numerical Methods in Engineering*, 121(6):1170–1197, 2020.

[4] Mathis Bode, Michael Gauding, Zeyu Lian, Dominik Denker, Marco Davidovic, Konstantin Kleinheinz, Jenia Jitsev, and Heinz Pitsch. Using physics-informed super-resolution generative adversarial networks for subgrid modeling in turbulent reactive flows. *CoRR*, abs/1911.11380, 2019.

[5] Hamza Boukraichi, Nissrine Akkari, Fabien Casenave, and David Ryckelynck. Uncertainty quantification in a mechanical submodel driven by a wasserstein-gan. *IFAC-PapersOnLine*, 55(20):469–474, 2022.

[6] Anindya Chatterjee. An introduction to the proper orthogonal decomposition. *Current science*, pages 808–817, 2000.

[7] Orestis Friderikos, Marc Olive, Emmanuel Baranger, Dimitrios Sagris, and Constantine David. A non-intrusive space-time interpolation from compact stiefel manifolds of parametrized rigid-viscoplastic fem problems. *arXiv preprint arXiv:2102.09216*, 2021.

[8] A. Giacoma, D. Dureisseix, A. Gravouil, and M. Rochette. A multiscale large time increment/fas algorithm with time-space model reduction for frictional contact problems. *International Journal for Numerical Methods in Engineering*, 97(3):207–230, 2014.

[9] Michel Géradin and Daniel J. Rixen. A 'nodeless' dual superelement formulation for structural and multibody dynamics application to reduction of contact problems. *International Journal for Numerical Methods in Engineering*, 106(10):773–798, 2016.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[12] Pierre Ladevèze, J-C Passieux, and David Néron. The latin multiscale computational method and the proper generalized decomposition. *Computer Methods in Applied Mechanics and Engineering*, 199(21-22):1287–1296, 2010.

[13] Simon Le Berre, Isabelle Ramière, Jules Fauque, and David Ryckelynck. Condition number and clustering-based efficiency improvement of reduced-order solvers for contact problems using lagrange multipliers. *Mathematics*, 10(9), 2022.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

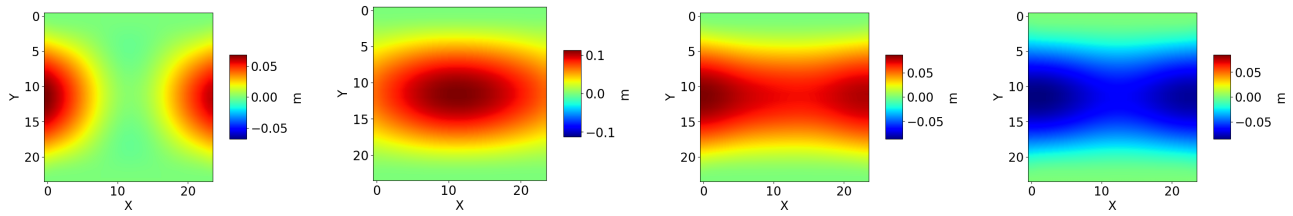
[15] Neil Muller, Lourenço Magaia, and B. M. Herbst. Singular value decomposition, eigenfaces, and 3d reconstructions. *SIAM Review*, 46(3):518–545, 2004.

[16] Sascha Ranftl, Wolfgang von der Linden, and MaxEnt 2021 Scientific Committee. Bayesian surrogate analysis and uncertainty propagation. In *Physical Sciences Forum*, volume 3, page 6. MDPI, 2021.

[17] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pages 2528–2535. IEEE, 2010.

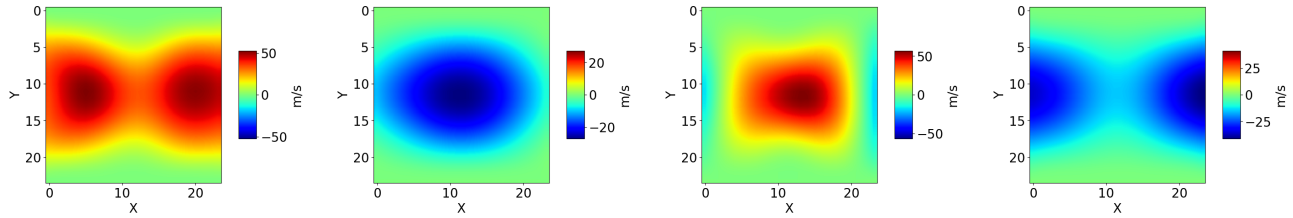
A. Supplementary results

In this section we present to the reader additionnal 2D visualization for further comparison between models.



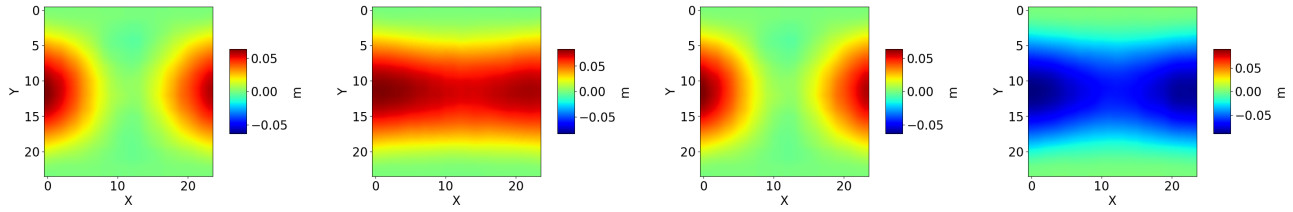
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 27. Real values of displacement



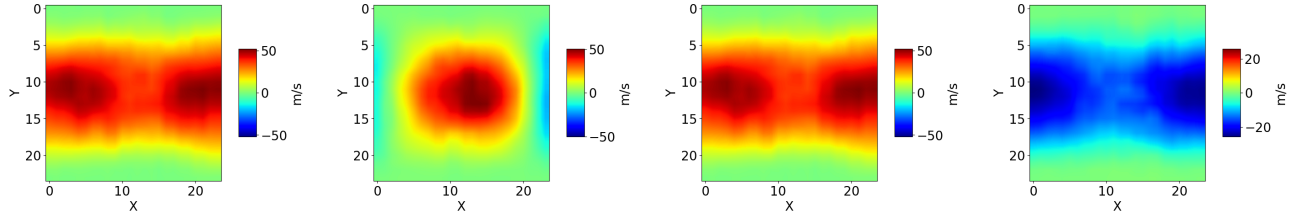
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 28. Real values of velocity



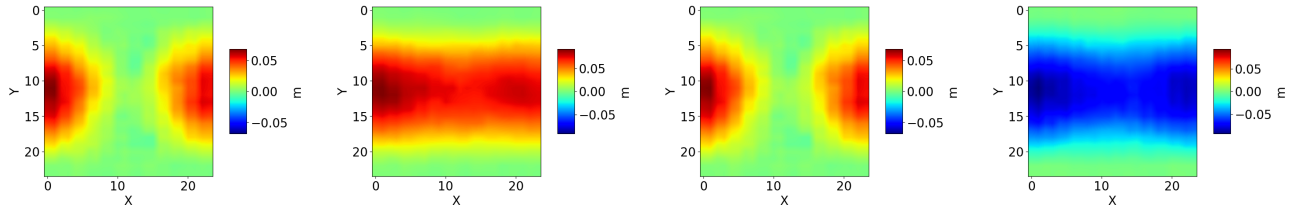
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 29. CNN prediction on S for displacement



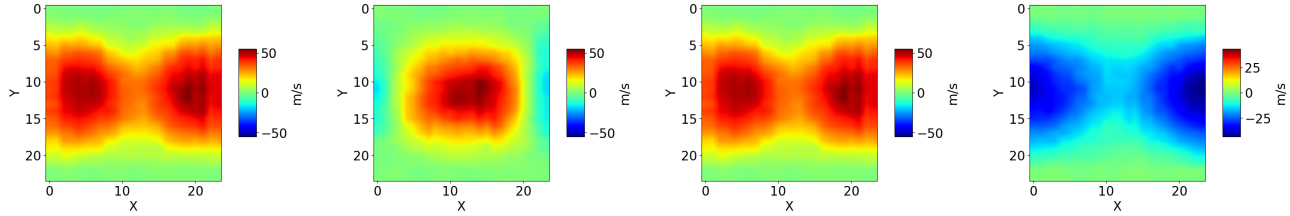
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 30. CNN prediction on S for velocity



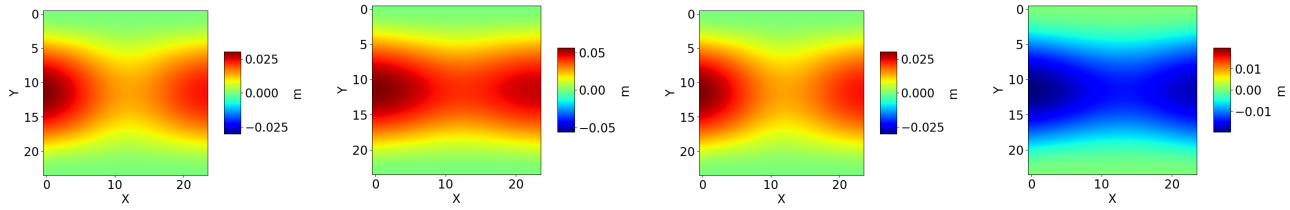
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 31. CNN_t prediction on S for displacement



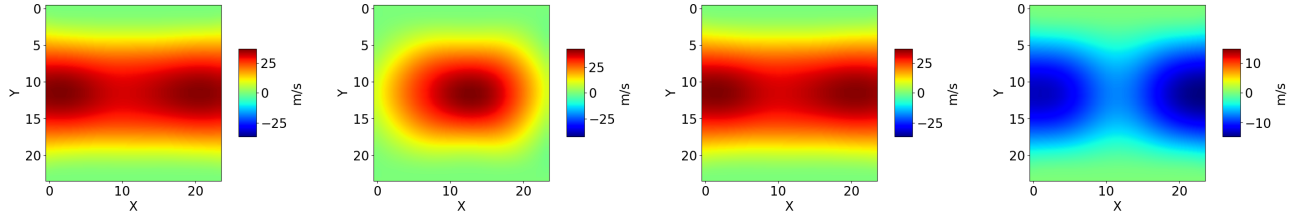
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 32. CNN_t prediction on S for velocity



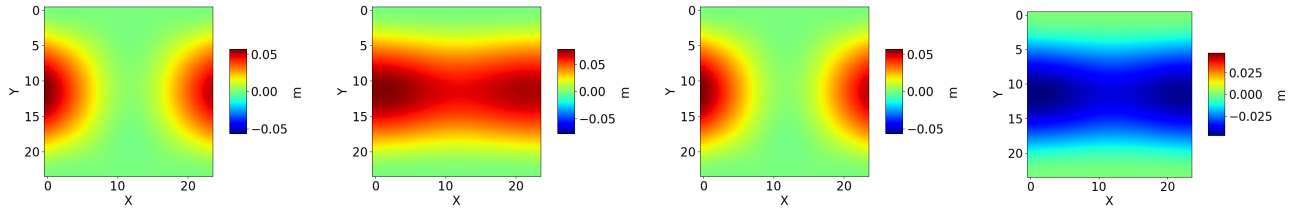
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 33. POD_p prediction on S for displacement



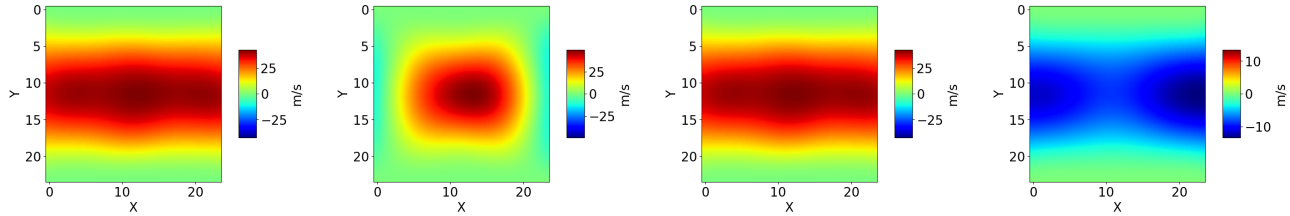
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 34. POD_p prediction on S for velocity



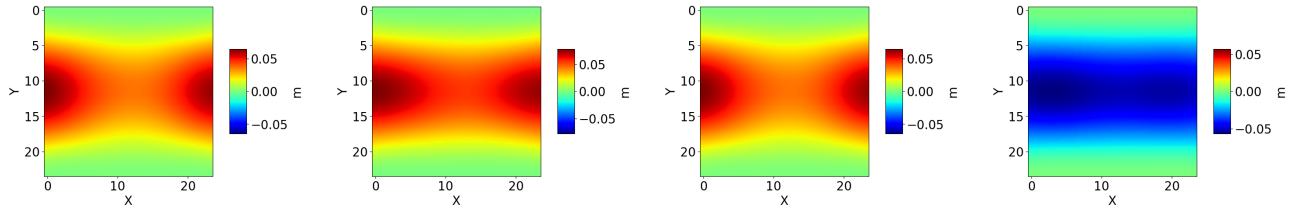
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 35. POD_pt prediction on S for displacement



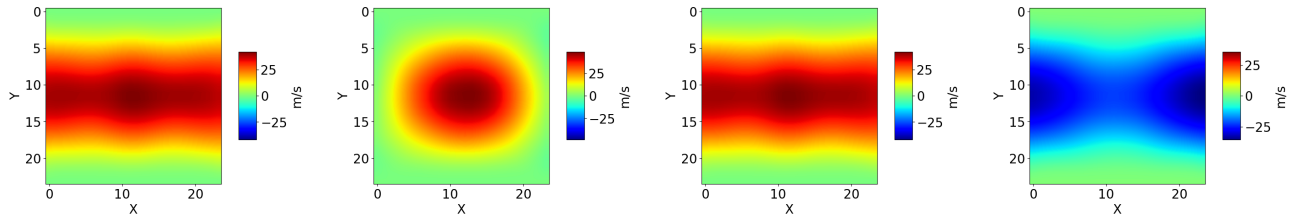
(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 36. POD_pt prediction on S for velocity



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 37. Stief prediction on S for displacement



(a) (t_1, θ_1) (b) (t_1, θ_2) (c) (t_2, θ_1) (d) (t_2, θ_2)

FIGURE 38. Stief prediction on S for velocity