

TUTORIAL ModelFLOWS-app v0.1

This document presents a short tutorial on ModelFLOWS-app, an open source Software for data post-processing, patterns identification and development of reduced order models using modal decomposition and deep learning architectures. When using the software, please, reference us as:

A. Hetherington, A. Corrochano, R. Abadía-Heredia, E. Lazpita, E. Muñoz, P. Díaz, E. Maiora, M. López-Martín and S. Le Clainche, ModelFLOWS-app: data-driven post-processing and reduced order modelling tools, arxiv, 2023.

1. Installing and using ModelFLOWS-app

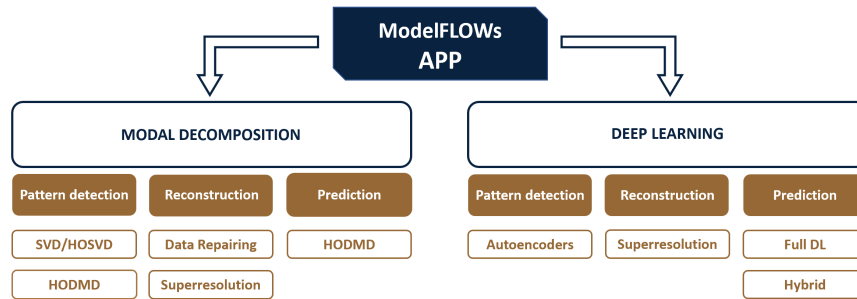


Figure 1: General distribution of ModelFLOWS-app software.

ModelFLOWS-app has been developed with Python 3.9 and is compatible with all posterior versions. This software does not require installation, but some Python libraries must be installed to the user's environment, Table 1:

Library	Version
tensorflow	2.10
protobuf	3.20
scikit-learn	1.2
keras-tuner	Latest
scipy	1.9.3
numba	0.56.4
hdf5storage	Latest
ffmpeg	Latest

Table 1: List of libraries required by ModelFLOWS-app

The web-browser version also requires the installation of the following libraries, Table 2:

Library	Version
streamlit	1.17
streamlit-option-menu	0.3.2

Table 2: List of additional libraries required by the ModelFLOWS-app web-browser version

These libraries are gathered in the *Requirements.txt* file found in each version’s folder and can be installed automatically running the *sudo pip install -r Requirements.txt* command in the console terminal inside the application path.

Once all libraries have been installed, ModelFLOWS-app can be initialised from inside the console terminal. This can be done by running *python ModelFLOWS_app.py* for the desktop version, while the web-browser version is started by running *streamlit run ModelFLOWS_webapp.py*.

The desktop version will open in the same command terminal [2](#), while the web-browser version will open up in Google Chrome [3](#). Once the application has loaded, the user will be welcomed with the main menu, from which they are able to navigate between the different modules.

```
ModelFLOWS-app

ModelFLOWS Application
-----

Authors: ModelFLOWS Research Group - E.T.S.I. Aeronautica y del Espacio - Universidad Politecnica de Madrid
Version: 0.1

This data-driven application consists of two modules:
- Modal Decomposition
- Deep Learning

Both blocks consist of algorithms capable of:
- detecting patterns,
- repairing and enhancing data,
- and predicting data from complex flow databases

The databases can be in the following formats:
- MATLAB ".mat"
- Numpy ".npy"
- Pickle ".pkl"
- Pandas ".csv"
- h5 ".h5"

This application takes in databases with the following shapes:
- 1D Arrays -> (1, n)
- 2D Matrices -> (x, y)
- 3D Tensors -> (y, x, t)
- 4D Tensors -> (m, y, x, t)
- 5D Tensors -> (m, y, x, z, t)

IMPORTANT: The data that defines the spatial mesh (x, y, z) must be located in the specified positions

For more information please visit: https://modelflows.github.io/modelflowsapp/
```

Figure 2: ModelFLOWS-app desktop version's main menu.

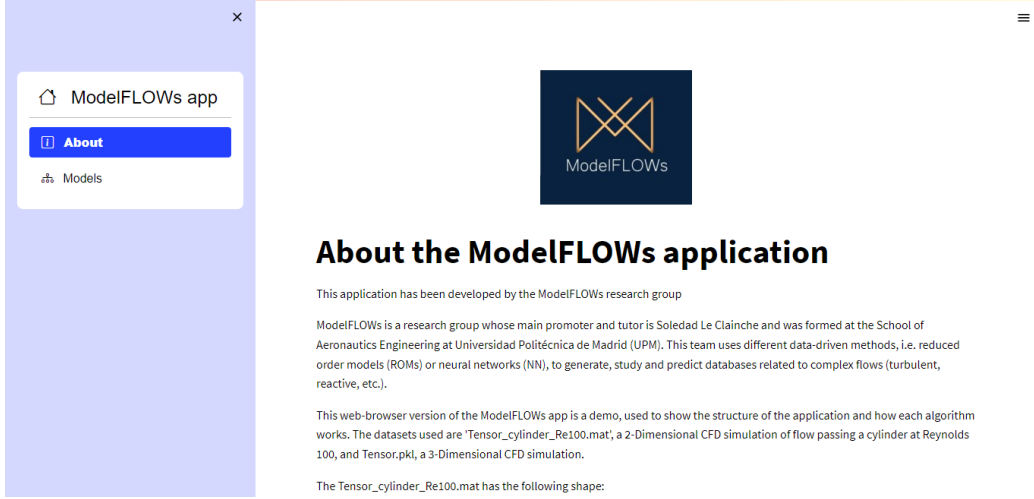


Figure 3: ModelFLOWS-app web-browser version's main menu.

The web-browser version of this application serves as a demonstration for inexperienced users to understand the terminology and also how each module works. Due to this matter, the focus will be set on this version, which is accompanied by a series of databases.

Tensor_cylinder_Re100.mat: This tensor type database is a 2D simulation of a Reynolds 100 flow passing through a cylinder. The database is formed by two velocity components, the spatial resolution along the stream-wise and normal components is composed by 499×199 grid points and the temporal resolution is represented by 151 snapshots equi-distant in time with time interval $\Delta t = 0.2$. See Fig. 4.

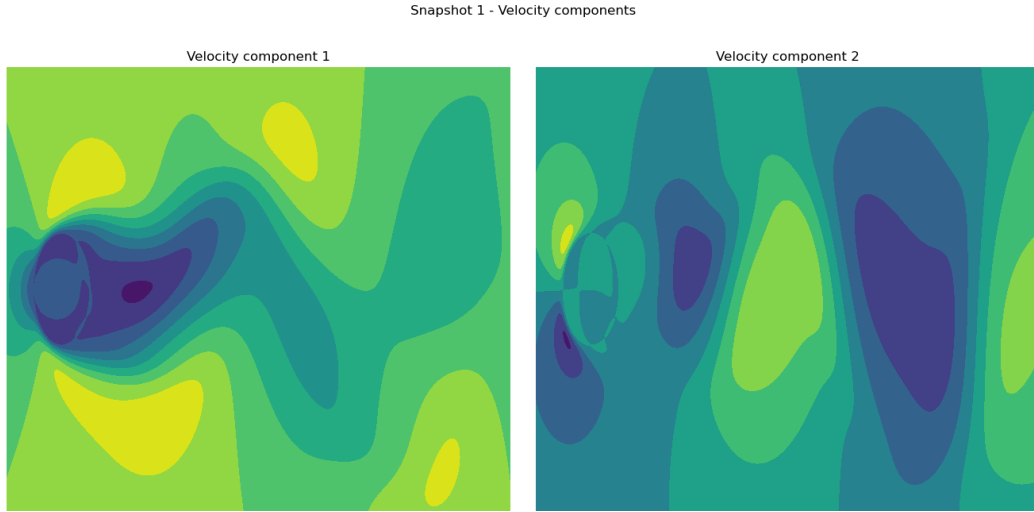


Figure 4: Streamwise and normal velocity components for the Tensor_cylinder_Re100.mat database in the first snapshot.

DS_30_Tensor_cylinder_Re100.mat: This is the first variant of the previous database, which consists of a downsampled version. This means that the spatial mesh has been reduced in size, in this case by a factor of 30, hence the final spatial resolution is 15×7 . See Fig. 5.

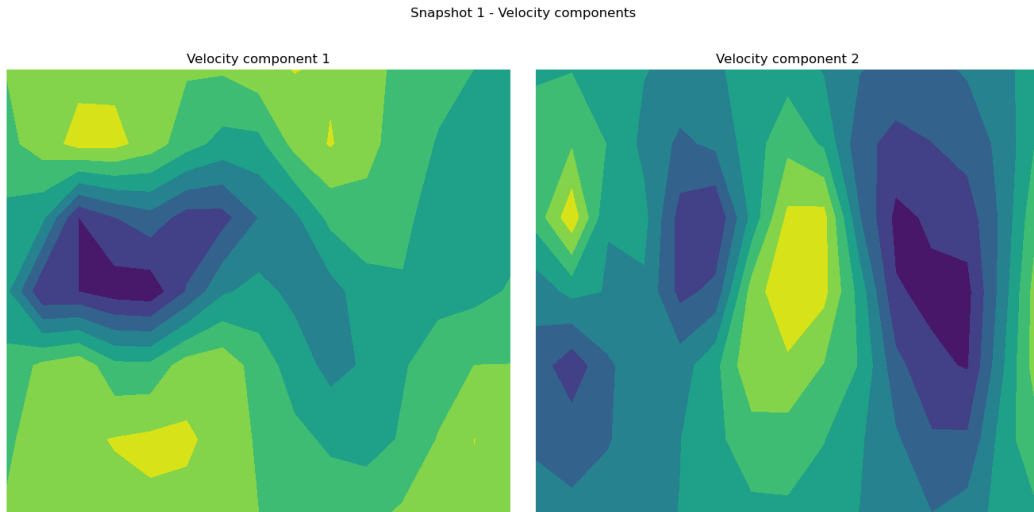


Figure 5: Streamwise and normal velocity components for the DS_30_Tensor_cylinder_Re100.mat database.

Gappy_Tensor_cylinder_Re100.mat: This variant is a “gappy” version of the database. This means that the database has missing velocity measurements, which have been substituted by NaN (Not A Number) values. It is important to note that the spatial mesh is not reduced in size. See Fig. 6.

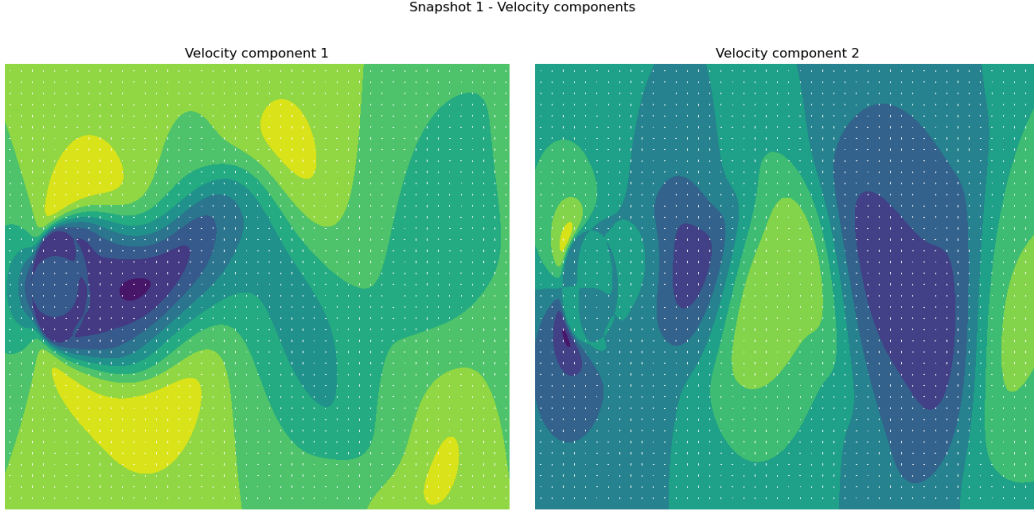


Figure 6: First snapshot velocity components for the *Gappy_Tensor_cylinder_Re100.mat* database.

Tensor.pkl: This database corresponds to the 3D wake behind a circular cylinder at Reynolds number 220 and with spanwise wavenumber 4. The database is given in tensor form and is formed by three velocity components (streamwise, normal spanwise), the spatial resolution along the streamwise, normal and spanwise components is composed by $100 \times 40 \times 40$ grid points and the temporal resolution is represented by 150 snapshots equi-distant in time with time interval $\Delta t = 1$. See Fig. 7.

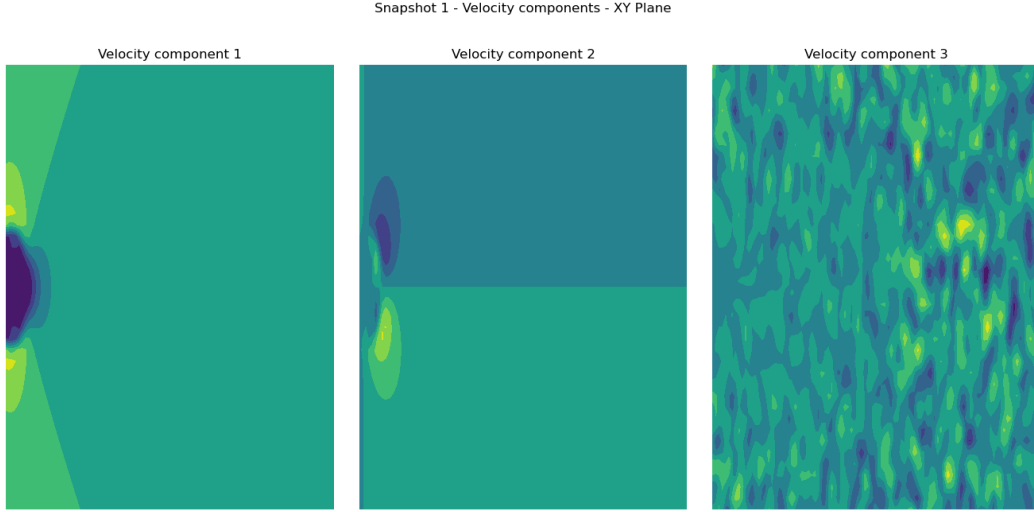


Figure 7: Streamwise, normal and spanwise velocity components in the first snapshot of the *Tensor.pkl* database.

DS_30_Tensor.pkl: This is the only variant of the *Tensor.pkl* database, where the spatial components are downsampled by a factor of 12 and 2, in the X and Y axis respectively, resulting in a database formed by $8 \times 20 \times 64$ grid points. See Fig. 8.

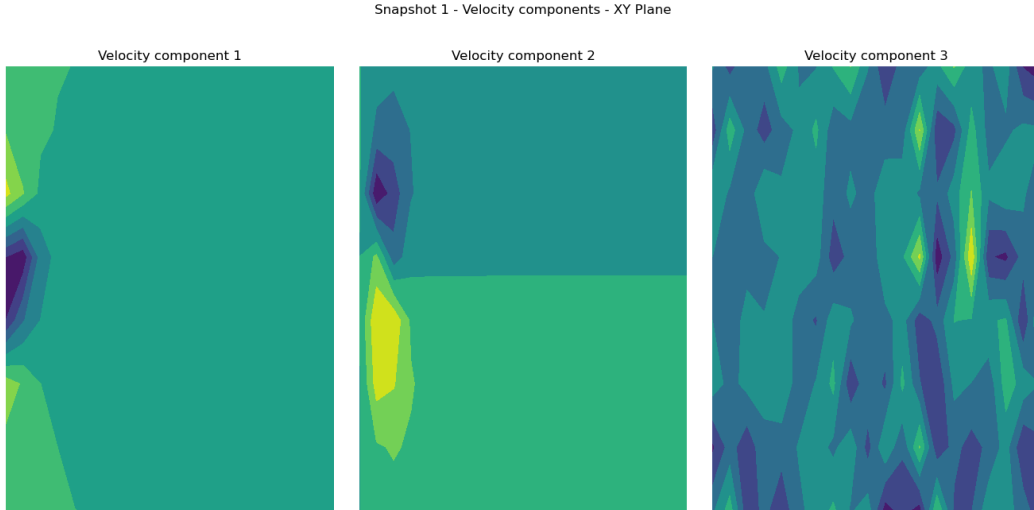
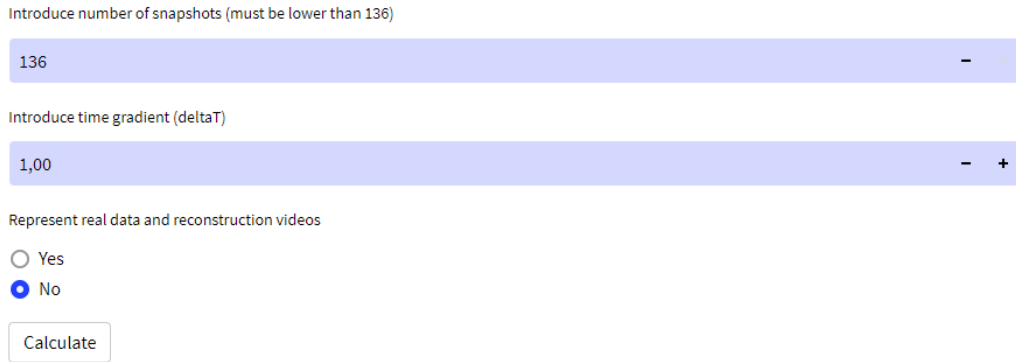


Figure 8: First snapshot velocity components for the *DS_30_Tensor.pkl* database.

These databases are also introduced to the user in the *About* section of

the ModelFLOWS-app web-browser version. It is to be noted that the default database used in all modules is *Tensor_cylinder_Re100.mat* and its variants, while the only module where the *Tensor.pkl* and its variant are selectable is the *Deep Learning Data Reconstruction* module.

In the following sections, each module will be shown using the above-mentioned databases. All modules follow the same structure: the user sets the algorithm or model parameters, being able to leave the default values, which in some cases are the optimal values for such model. In some cases, the user can select the model outputs, which can be plots or videos. Once the model inputs and outputs are selected, the user can initialize by pressing the *Calculate* button, see Fig. 9.



Introduce number of snapshots (must be lower than 136)

136 - +

Introduce time gradient (deltaT)

1,00 - +

Represent real data and reconstruction videos

☐ Yes

☒ No

Calculate

Figure 9: ModelFLOWS-app web-version Calculate button.

Once the run has completed, the most relevant plots will appear on screen while the rest will be saved into a folder in the same directory as the application, and whose name appears at the bottom of the application screen. A new case can be run by pressing the *Refresh* button, Fig. 10.

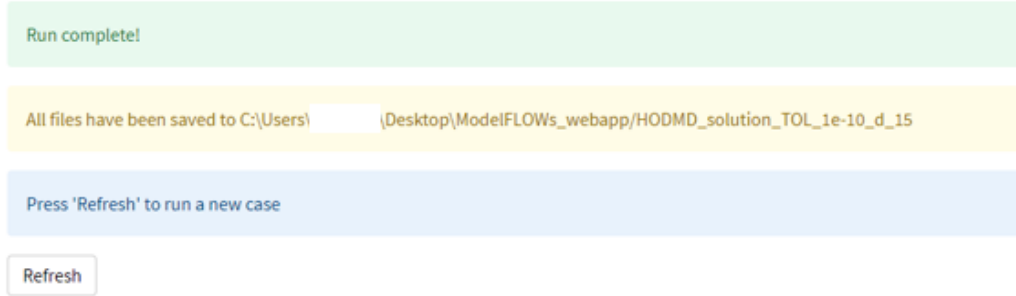


Figure 10: ModelFLOWS-app web-version end of run saved file directory and Refresh button.

ModelFLOWS-app is comprised by two modules, these being modal decomposition and deep learning. At the same time, both of these modules are divided into three types of actions: pattern detection, data reconstruction, and data forecasting or prediction.

1.1. Modal decomposition

1.1.1. Pattern detection

There are 3 algorithms used for pattern detection, which are Singular Value Decomposition (SVD), Higher Order Dynamic Mode Decomposition (HODMD) and Multi-dimensional HODMD (mdHODMD), which has two variants (classic and iterative), and High Order Singular Value Decomposition (HOSVD).

- **Singular Value Decomposition, SVD:** This algorithm allows the user to configure the following parameter, Tab. 3:

Parameter	Default value
Number of SVD modes to retain	18

Table 3: List of configurable parameters for SVD

Once the run has completed, a folder is created where the following files are saved:

- Temporal (SVD coefficients) modes tensor in Numpy array format named *time_modes.npy*

- Reconstructed tensor in Numpy array format named *Reconst.npy*
- Mode plots, Fig. 11

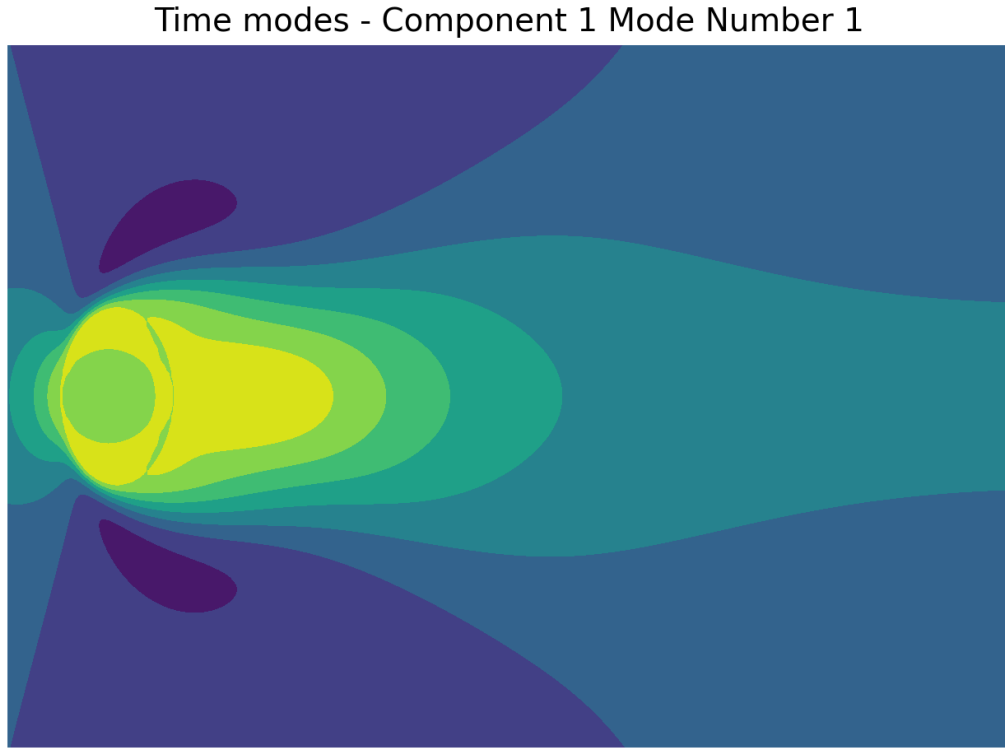


Figure 11: Real value plots for the first velocity component of the first SVD mode.

- **HODMD and Multi-dimensional HODMD:** There are two HODMD algorithms implemented in this application: HODMD and Multi-dimensional HODMD (mdHODMD). The difference between these algorithms is that HODMD performs SVD on the input data, while the mdHODMD algorithm performs HOSVD.

At the same time, the mdHODMD algorithm has two versions: iterative (mdHODMD-it) and non-iterative. The main difference between the classic and iterative versions is that the iterative version starts off by performing HOSVD on the input tensor, reconstructs the tensor, and then starts a new iteration by re-applying HOSVD, this time on

the reconstruction, comparing afterwards if any new HOSVD modes have been deleted regarding the previous iteration. This iteration process continues until no improvement is found.

Both algorithms and variants, which can be easily selected before the parameters menu, have the following input parameters, Tab. 4:

Parameter	Default value
SVD tolerance	$1e^{-10}$
HODMD tolerance	$1e^{-3}$
Number of windows in HODMD (d)	15
Number of snapshots	151
Time interval	1

Table 4: List of configurable parameters for HODMD, mdHODMD and mdHODMD-it algorithms.

The menu presents an option that allows to create videos of a specific component or both velocity components of the original and reconstructed data.

After running this algorithm, the following files are saved to the solution folder:

- DMD modes tensor in Numpy array format named *DMDmode.npy*
- Reconstructed tensor in Numpy array format named *TensorReconst.npy*
- Growth rate, frequency and amplitude of all DMD modes in Numpy array format named *GrowthRateFrequencyAmplitude.npy*
- DMD mode plots, Fig. 12
- Growth rate vs. Frequency plot
- Amplitude vs. Frequency plot, Fig. 13
- Two plots, one per velocity component, comparing the original and reconstructed velocity in a given point of the spatial mesh

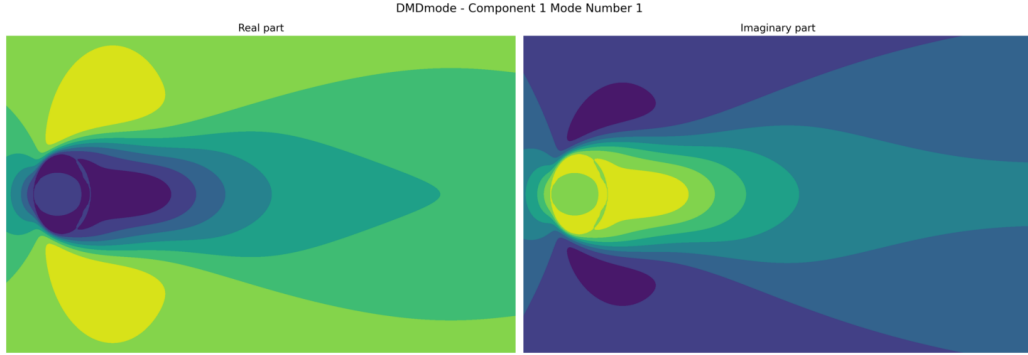


Figure 12: Real and imaginary value plots for the first velocity component of the first DMD mode.

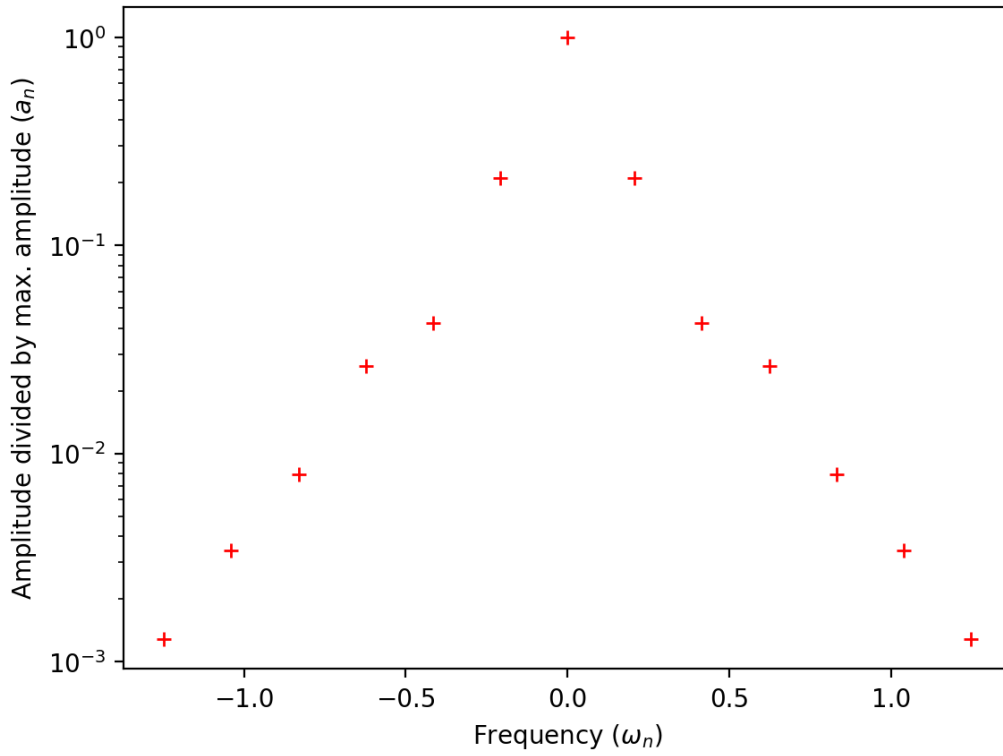


Figure 13: DMD modes Amplitude vs. Frequency.

- **High Order Singular Value Decomposition, HOSVD:** This algorithm has the following configurable parameters, Tab. 5:

Parameter	Default value
SVD tolerance	$1e^{-4}$
Number of snapshots	120

Table 5: List of configurable parameters for HOSVD

The HOSVD algorithm also allows the user to plot videos for one or both velocity component of the original and reconstructed data.

The solution folder contains the following files:

- Temporal modes tensor in Numpy array format named *time_modes.npy*
- Reconstructed tensor in Numpy array format named *TensorReconst.npy*
- A plot showing all singular values vs. the SVD retained singular values, Fig. 14
- Mode plots

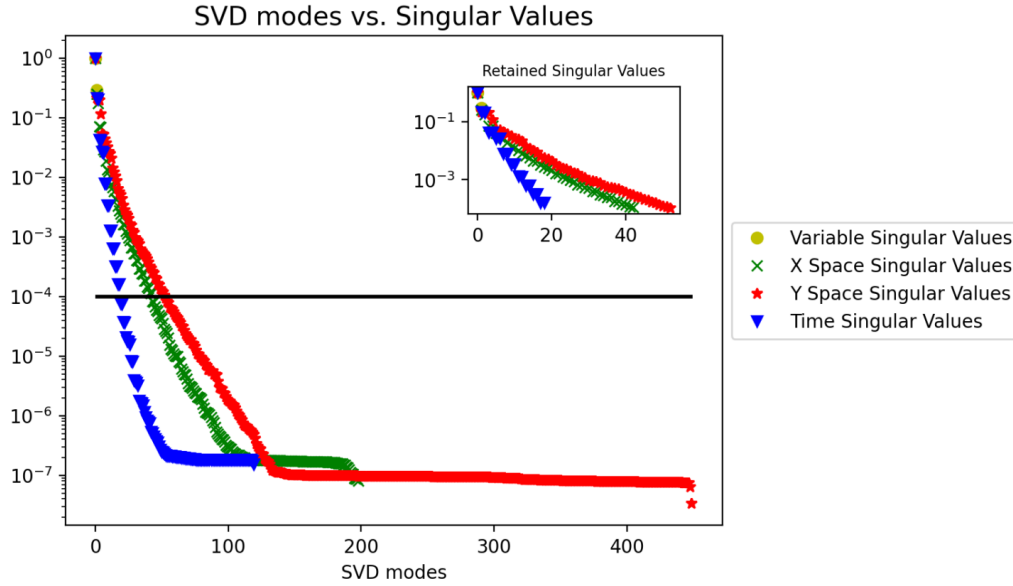


Figure 14: Number of SVD mode vs. singular values in HOSVD.

1.1.2. Data reconstruction

The data reconstruction module allows the user to repair datasets with missing data, and enhance a downsampled dataset to a higher resolution (superresolution) using modal decomposition techniques.

- **Data repairing:** This module uses the *Gappy-Tensor-cylinder-Re100.mat* database, which has missing data, fills in the missing data with zeros, the mean value of the velocity, or by interpolation, which can be linear or to the nearest value. Once this initial reconstruction is complete, HOSVD is applied, retaining as many modes as the user desires. Then the reconstructed tensor is built and compared to the ground truth which, in this case, is *Tensor-cylinder-Re100.mat*.

This module takes in the following parameters, Tab. 6:

Parameter	Default value
Number of SVD modes to retain	18
Data completion	nearest value interpolation

Table 6: List of configurable parameters for HOSVD

The user can also select to plot the singular values decay before and after the reconstruction, as well as a data comparison plot where the initial gappy data is visually compared to the reconstruction and ground truth data, Fig. 15.

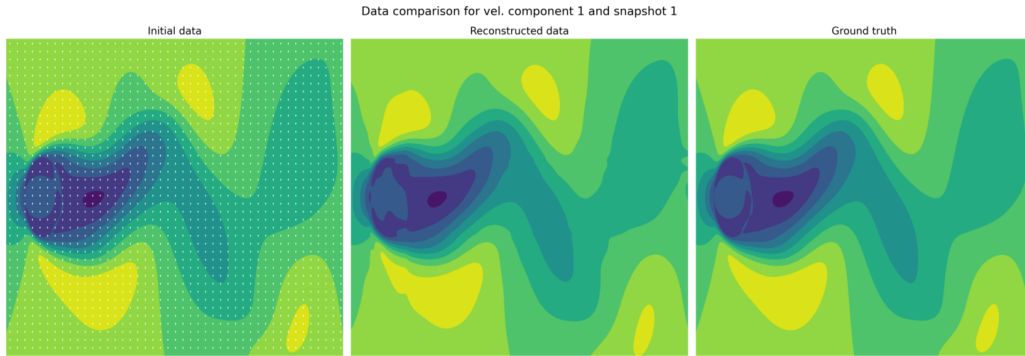


Figure 15: Comparison between the initial data, the reconstructed data, and the ground truth.

The run completes creating a solutions folder where the following files are saved:

- Repaired tensor in Numpy array format named *Repaired_data.npy*
- If the option is selected, an initial reconstruction vs. final reconstruction singular values decay plot.
- **Superresolution:** The data superresolution module enhances the *DS_30_Tensor_cylinder_Re100.mat* resolution to the resolution selected by the user, which is a factor power of 2, Table 7:

Parameter	Default value
Enhancement factor (2^{factor})	$factor = 4$

Table 7: List of configurable parameters for HOSVD

The output of this module is:

- Enhanced resolution tensor in Numpy format named *Enhanced_data.npy*

Figure 16 shows a comparison between the original and enhanced resolution data for the first velocity component of the first snapshot.

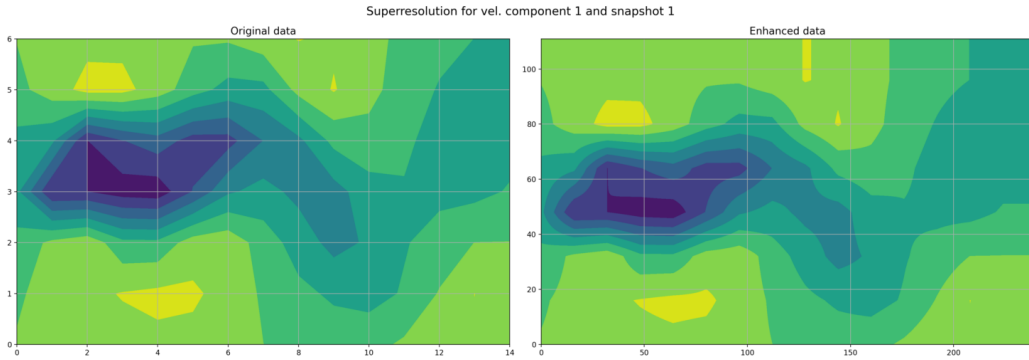


Figure 16: Comparison between the original data and the enhanced resolution data.

1.1.3. Prediction

Data prediction is done by adapting the HODMD and mdHODMD algorithms (and both of their variants) for such purpose. This is achieved

by reconstructing the tensor with a higher number of snapshots, which are selected by the user.

Predictive HODMD and Predictive mdHODMD: Similarly to the pattern detection module, the user can select between both types of HODMD and, at the same time, between both variants of mdHODMD. Table 8 shows the configurable parameters for the predictive HODMD algorithms.

Parameter	Default value
SVD tolerance	$1e^{-10}$
HODMD tolerance	$1e^{-3}$
Number of windows in HODMD (d)	15
Number of snapshots	151
Time interval	1
Final snapshot in the prediction interval	300

Table 8: List of configurable parameters for the HODMD algorithms.

The following options are also selectable:

- Retain DMD modes by setting a growth rate tolerance
- If the previous option is selected, a tolerance value can be set. By default 0.5.
- Set the growth rate of the retained modes to 0.
- Plot videos for one or both velocity components of the original and predicted data.

Once the run has completed, the following files are saved into the solution folder:

- DMD modes tensor in Numpy array format named *DMDmode.npy*.
- The reconstructed tensor with all future predicted snapshots selected by the user, in Numpy array format, named *TensorPred.npy*.
- Growth rate, frequency and amplitude of all DMD modes in Numpy array format named *GrowthRateFrequencyAmplitude.npy*.
- DMD mode plots.

- Growth rate vs. Frequency plot.
- Amplitude vs. Frequency plot.
- Two plots, one per velocity component, comparing the original and predicted velocity in a given point of the spatial mesh, Fig. 17.

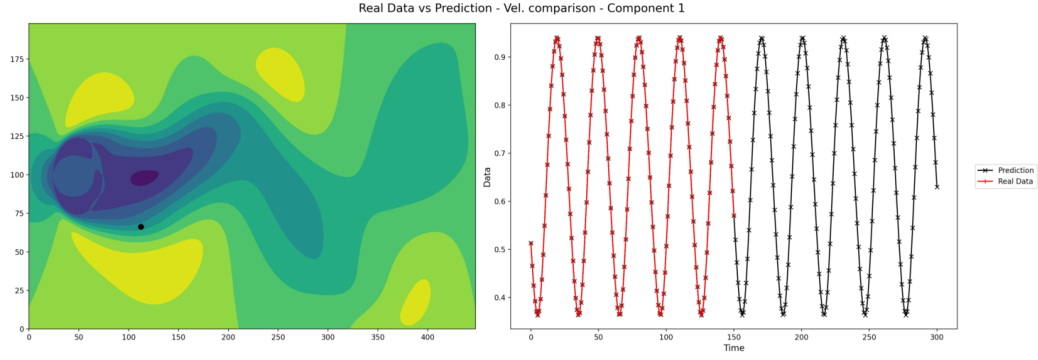


Figure 17: First velocity component comparison between the original data and prediction.

1.2. Deep learning

This module gathers all the deep learning models designed to detect patterns, reconstruct data, and also make predictions.

1.2.1. Pattern detection

This module uses an autoencoders model, which is trained to detect patterns in databases. The model parameters, Tab. 9, are:

Parameter	Default value
Batch size	64
Training epochs	200
Number of autoencoder dimensions	10

Table 9: List of configurable parameters for the autoencoders model.

The following output options can be selected:

- Plot the autoencoders.

- Plot videos for both velocity components of the original and reconstructed data.

When the run completes, the following files are saved to the solution folder:

- Autoencoder components tensor in Numpy array format named *RecAE.npy*.
- Plots of both autoencoder components, Fig. 18.
- RRMSE plot.
- Relative RRMSE plot.

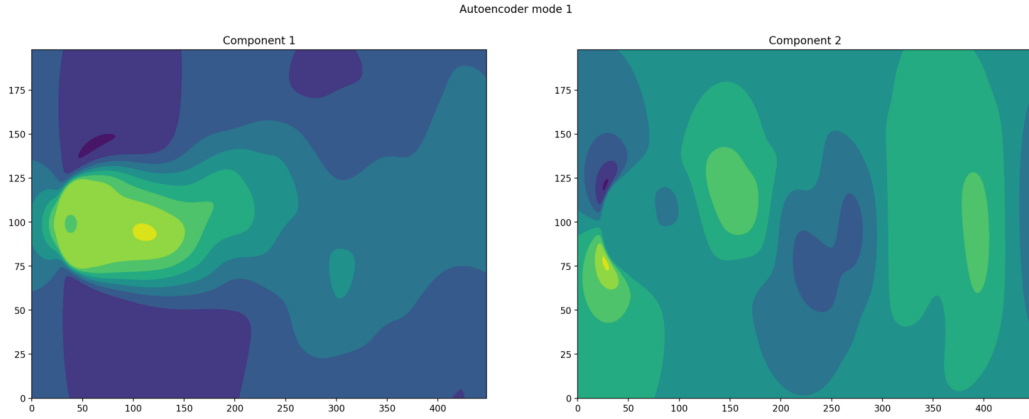


Figure 18: Autoencoder's first component.

1.2.2. Data reconstruction

The deep learning superresolution network allows the user to enhance the resolution of *DS_30_Tensor_cylinder_Re100.mat* or *DS_30_Tensor.pkl* to the ground truth resolution. This model presents various data preprocessing options, such as adding noise and/or scaling the data, but mainly applies SVD to the input data and trains the model with the SVD output matrices.

This model has the following parameters, Tab. 10:

Parameter	Default value
Batch size	16
Neurons per layer	50
Hidden layers activation function	elu
Output layers activation function	tanh
Learning rate	$5e^{-3}$
Training data size	121
Training epochs	150

Table 10: List of configurable parameters for the deep learning superresolution model.

The training data can also be shuffled. Once the model parameters have been configured, the output options can be selected:

- Save all data files, such as matrices and tensors.
- Create a plot comparing the reconstruction, the input data and the ground truth.
- Plot videos comparing the reconstruction, the input data, and ground truth.

If the user selects all output options, the following files are saved to the solution folder:

- Enhanced tensor in Numpy array format named *EnhancedData.npy*.
- Tables containing the model training results.
- Model training weights
- Plots comparing the ground truth, initial data and reconstruction for all velocity components during the first 10 snapshots, Fig. 19.
- Videos comparing the ground truth and reconstructed data.

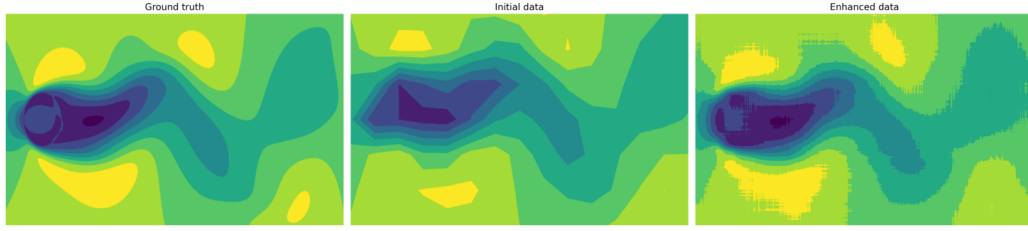


Figure 19: Plot comparing the ground truth, initial data and prediction of the first velocity component for the first snapshot.

1.2.3. Prediction

Two model types are available: full deep learning, and hybrid deep learning. This last one applies SVD on the training data during the data pre-processing stage. The user can also select between a Convolutional Neural Network (CNN) architecture or a Recurrent Neural Network (RNN) architecture.

- **Full deep learning model:** Both architectures will be presented together in this section. The configurable parameters, Tab. 11, are:

Parameter	CNN Default value	RNN Default value
Batch size	4	4
Training epochs	3	20
Hidden layers activation function	relu	linear
Output layers activation function	linear	linear
Neurons per layer	N/A	50
Shared dims	30	50
Learning rate	$1e^{-2}$	$1e^{-2}$

Table 11: List of configurable parameters for the full deep learning prediction model.

This model outputs the following files to the solution folder:

- Model training summary in various formats.
- Tables containing the prediction error statistics for each prediction.
- Loss function evolution during training plot.

- Plots comparing the ground truth and predicted data.
- **hybrid deep learning model:** This model performs SVD on the input data and trains the model with the SVD core tensor. Other data preprocessing options that this model offers are:
 - Input variables scaling.
 - Temporal coefficients scaling.

The configurable parameters, Tab. 12, are:

Parameter	CNN Default value	RNN Default value
Number of SVD modes to retain	15	15
Batch size	8	8
Training epochs	200	200
Hidden layers activation function	relu	relu
Output layers activation function	tanh	tanh
Neurons per layer	30	30
Shared dims	20	20
Learning rate	$2e^{-3}$	$2e^{-3}$

Table 12: List of configurable parameters for the full deep learning prediction model.

This model outputs the following files to the solution folder:

- Various Numpy array format files containing the RRMSE error measurements.
- Model training summary in various formats.
- Tables containing the model training error.
- Loss function evolution during training plot.
- Plots comparing the ground truth and predicted data.
- Training loss and RRMSE plots.
- Other error measurement plots.
- Plots comparing the original vs. predicted data of both velocity components, Fig. 20.

Prediction vs.Original Data - Comp. 1 Snapshot 2

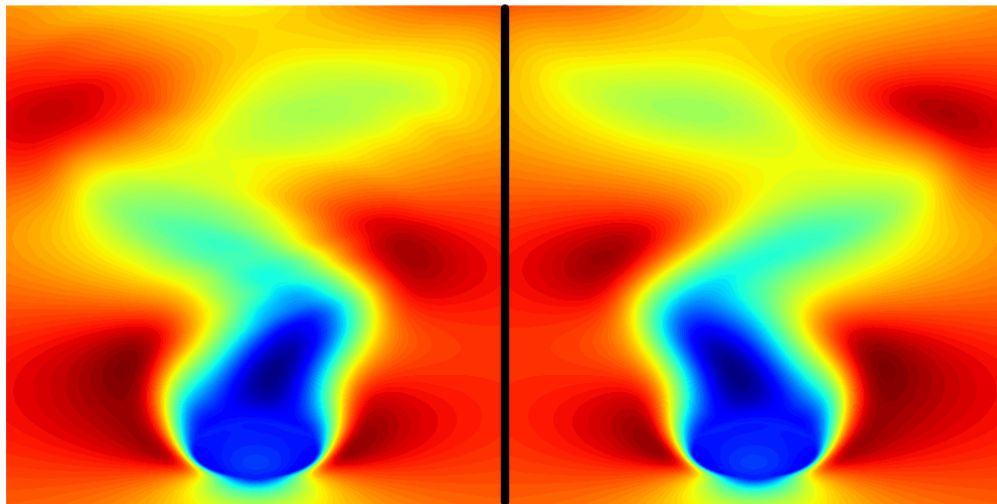


Figure 20: Plot comparing the original and predicted data of the first velocity component for snapshot 2.