Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών

Applications of Linear & Kernel Principal Component Analysis in CFD

Computational Project

8th semester course, offered in the School of Mechanical Engineering, National Technical University of Athens (NTUA)

Antonis Tzanetakis, 02116080

Supervisor: Kyriakos C. Giannakoglou, NTUA Professor

05 August 2022

Table of contents

Abstract

A powerful technique in gradient-based optimization in problems involving unsteady flows is the unsteady adjoint method. The adjoint equations are integrated backwards in time, which renders the availability of the primal fields necessary at each instant in time. This infers substantial storage requirements. The present project tackles the problem of compressing these primal fields through implementation of the methods of Linear Principal Component Analysis (LPCA) and Kernel Principal Component Analysis (KPCA). Their qualitative features and their performance are compared in two cases: a flow past one and a flow past two cylinders.

Keywords: unsteady adjoint, primal fields, data compression, linear, non-linear, SVD, PCA, LPCA, KPCA, reconstruction

1. Introduction

This project was realized in the context of the course “Computational Project”, offered in the eighth (8^{th}) semester of the school of Mechanical Engineering, NTUA. The data used were provided by the Parallel CFD & Optimization Unit (PCOpt).

In unsteady adjoint method the main idea is that the gradient of the objective function is computed with respect to certain design variables, is computed by solving the adjoint equations at a cost that is, more or less, equal to the cost of solving the primal flow problem.

However, the solution of the unsteady adjoint equations requires the availability of the solutions of the primal equations at every time step. This can be achieved either by solving the primal equations again from t = 0 (till the instant needed) or by storing the full primal fields. This, in substance, implies a choice between extremely high computational cost and intense storage requirements. Both decrease the affordability of the method.

A widely used way to deal with this problem is the technique of check-pointing (Griewank & Walther, 2000). In essence, this is a trade-off between computational cost and storage requirements: the primal fields are stored –in memory or occasionally, on the hard drive- at a certain number of points in time (“checkpoints”), and if an intermediate point is needed, the closest previous stored checkpoint is used to start the integration of the primal solver and both recomputation cost and memory needs can be now adjusted by the user.

There is, however, the alternative of computing the entire times series of the primal fields and then compressing it. The decompression/reconstruction produces primal solutions that are approximate which are then used in the integration of the adjoint equations. In the present project, ways to compress the data of the unsteady primal solution through Linear PCA and Kernel PCA are examined. The qualitative and quantitative results of the compression and reconstruction are the main aspects inspected.

Note here that it would be preferable that incremental versions of these techniques are implemented, as their standard variants need the complete time series to be available, which would partially beat the purpose. In this context, in a subsequent phase of the project, incremental SVD is also introduced to investigate this possibility.

2. Theoretical prerequisites

2.1. PCA

The main procedure behind Principal Component Analysis (PCA) is the following:

Assuming there are M data column vectors $\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_M} \in R^N$, the data matrix $\mathbf{X} = [\mathbf{x_1}\ \mathbf{x_2}\ \dots \mathbf{x_M}] \in \mathrm{R}^{\mathrm{NxM}}$ can be defined.

The mean of the data is given by $\mu = \frac{1}{\mathrm{N}}\mathbf{X1_N}$ and the mean matrix is then:

$$\bar{\mathbf{X}} = \mu \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix}^{\mathrm{T}} = \mu \mathbf{1}_{\mathrm{N}}^{\mathbf{T}}$$

Subtracting this from data matrix **X** results in the centered data matrix:

$$\mathbf{X_c} = \mathbf{X} - \bar{\mathbf{X}} = \mathbf{X}\left(\mathbf{I_M} - \frac{1}{\mathrm{N}}\mathbf{1_M}\mathbf{1}_{\mathrm{M}}^{\mathbf{T}}\right) = \mathbf{XH}, \quad \mathbf{H} \coloneqq \text{centering matrix}$$

Then, assuming **X** is mean centered or the procedure above has already been performed, the (sample) covariance matrix of its columns can be calculated:

$$\mathbf{C_X} = \frac{1}{\mathrm{M}}\mathbf{X}\mathbf{X}^{\mathbf{T}} = \begin{bmatrix} \sigma_{x_1}^2 & \cdots & \sigma_{x_1 x_N}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{x_N x_1}^2 & \cdots & \sigma_{x_N}^2 \end{bmatrix}$$

Remark: The diagonal of this matrix contains the variance of each vector $\boldsymbol{x_j}$.

If a suitable coordinate transform is applied, such that $\mathbf{C_X}$ becomes diagonal, in the new coordinate system, all of the covariances will be zero. This would mean that we end up with orthogonal vectors (since their inner products are zero) that are statistically independent.

This coordinate transform can be obtained by computing the eigendecomposition of $\mathbf{C_X}$:

$$\mathbf{C_X} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}$$

Then, the diagonal elements of $\boldsymbol{\Lambda}$ are sorted in a descending way, so that the first element corresponds to the direction in the data with the maximum variance, i.e. the maximum "energy". This way one can truncate the last $(\mathrm{M} - \mathrm{r})$, $\mathrm{r} << M$ eigenvalues and their corresponding eigenvectors, while losing the minimum "energy" possible. Finally, the data matrix is projected on the truncated eigenvectors:

$$\mathbf{Z} = \mathbf{U}_{\mathrm{r}}^{\mathrm{T}}\mathbf{X}$$

Next, applying PCA based on the Gram matrix can lead to lower computational cost (when M < N), plus it is a formulation needed for the kernelization of PCA (see section 2.3). This can be done as follows:

Starting from a -mean centered- data matrix $\mathbf{X} = [\mathbf{x_1}\ \mathbf{x_2}\ ...\ \mathbf{x_n}]$, an SVD is performed:

$$\mathbf{X} = \mathbf{U\Sigma V^T}$$

Then,

$$\mathbf{XX^T} = \mathbf{U\Sigma V^T V\Sigma U^T} = \mathbf{U\Lambda U^T}$$

$$\mathbf{X^T X} = \mathbf{V\Sigma U^T U\Sigma V^T} = \mathbf{V\Lambda V^T}$$

This means that one can simply interchange between Covariance and Gram matrices by going from right to left singular eigenvectors and vice versa. These can be easily computed from each other as follows:

$$\mathbf{U} = \mathbf{XV\Lambda^{-0.5}}$$

$$\mathbf{V} = \mathbf{X^T U\Lambda^{-0.5}}$$

The projection of the data matrix to the truncated eigenvectors is now replaced with:

$$\mathbf{Z} = \mathbf{U_r^T X} = \left(\mathbf{XV_r\Lambda_r^{-0.5}}\right)^{\mathbf{T}}\mathbf{X} = \mathbf{\Lambda_r^{-0.5} V_r^T X^T X}$$

In the end, the only matrices that are calculated are $\mathbf{X^T X}$, $\mathbf{V_r}$, $\mathbf{\Lambda_r}$ and **Z.**

Note: A flowchart of the possible derivations of LPCA can be found in Appendix A.

2.2. Non-linear PCA

The need for non-linear PCA arises naturally. When linear PCA fails to find low dimensional patterns, it is possible that our data exhibit non-linear behavior that could be represented in a more concise way using non-linear coordinate systems. It is the same notion as using a non-linear function to regress through some data points when a straight line is not a good enough model.

2.3. KPCA

Kernel PCA is one of the most widely used non-linear PCA-based algorithms. It is basically the kernelized version of LPCA.

During kernelization the design space is mapped to a higher dimensional one:

$$\varphi: R^N \to R^L, \quad L \gg$$

Specifically, it is preferable to map to an infinite-dimensional space. Essentially, this is a trade off in complexities: a non-linear finite-dimensional problem is replaced by a linear infinite-dimensional problem. Now, it would be preferable that the necessary

computations take place in the feature space, where things are linear, but this is not always possible, especially if our space is infinite dimensional.

The kernel trick solves this problem. It can be proved that: $\forall\varphi$: $\varphi(\mathbf{x})\varphi^{\mathrm{T}}(\mathbf{x}) = \mathrm{k}(\mathbf{x}, \mathbf{x}^{\mathbf{T}})$, namely that for any mapping function φ, the inner product (that is why the Gram based formulation of PCA was introduced in section 2.1) in the feature space can be calculated as a non-linear function ("kernel function") in the input space. Through the kernel trick, there is no need to calculate the explicit mapping on the feature space.

A widely used kernel function is the Radial Basis Function (RBF). The resulting Kernel matrix is defined as follows:

$$\mathbf{K}_{\mathrm{i,j}} = \mathrm{k}(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\frac{\|\mathbf{x_i} - \mathbf{x_j}\|_2^2}{2\sigma^2}\right)$$

where σ is the width constant.

This Kernel matrix is subsequently used as the new data matrix. Gram-based LPCA (see section 2.1) is used: Eigendecomposition is applied on a centered Kernel matrix $= \mathbf{H}\mathbf{K}_{\mu\neq 0}\mathbf{H}$:

$$\mathbf{K} = \mathbf{A}\boldsymbol{\Lambda}\mathbf{A}^{\mathrm{T}}$$

and the projection on the truncated eigenvectors $\mathbf{A_r}$ is computed:

$$\mathbf{B} = \boldsymbol{\Lambda}_{\mathbf{r}}^{-\mathbf{0.5}}\mathbf{A}_{\mathbf{r}}^{\mathbf{T}}\mathbf{K}$$

Finally, the reconstruction of the data that has been compressed through KPCA is achieved in an approximate fashion, through building an optimization problem, which is typically solved through the fixed point method. The objective function to minimize is the following (Scholkopf, 1998):

$$\rho(\mathbf{z}) = \mathrm{k}(\mathbf{z}, \mathbf{z}) - 2\sum_{\mathrm{k}=1}^{\mathrm{n}} \beta_{\mathrm{k}} \sum_{\mathrm{i}=1}^{\mathrm{l}} \mathrm{a}_{\mathrm{i}}^{\mathrm{k}} \mathrm{k}(\mathbf{p_i}, \mathbf{z}) + \mathrm{C}$$

where **z** is a snapshot of the primal field, β_k is the projection to the KPCA modes, a_i^k are the elements of the truncated KPCA modes and C is a constant.

Then, demanding its derivative with respect to **z** to be zero

$$\frac{\partial\rho(\mathbf{z})}{\partial\mathbf{z}} = 0$$

and after few calculations, the final expression to be used in the fixed point method is derived :

$$\mathbf{z} = \frac{\sum_{k=1}^{r} \beta_k \sum_{j=1}^{M} a_i^k k(\mathbf{x_i}, \mathbf{z}) \mathbf{x_i}}{\sum_{k=1}^{r} \beta_k \sum_{j=1}^{M} a_i^k k(\mathbf{x_i}, \mathbf{z})}$$

Note: A flowchart of the general algorithm of KPCA can be found in Appendix A.

2.4. PCA/SVD variant: incremental SVD

Standard SVD is performed on a full data matrix **X,** which requires storage of the full data. This is extremely counterproductive and it is one of the drawbacks iSVD is used to eliminate. In the original paper, Brand introduces the method for updating SVD when a single column vector is added to our data matrix (Brand, 2002). The algorithm's flowchart is as follows:

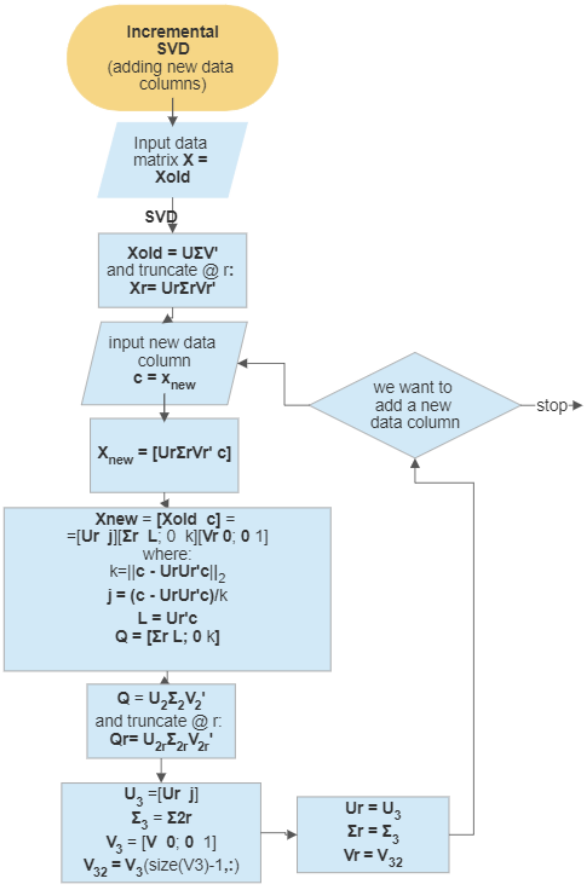


Figure 1: iSVD algorithm flow chart

2.5. Reduced Order Models (ROMs)

ROMs are data driven models of differential equations or even models of other models ("metamodels"), which are commonly used in engineering to create low cost computations (p.e. for simulation or optimization purposes). The main idea is that low

dimensional behavior in the equations is exploited to build high end approximations of the real phenomena.

In the current project, ROMs are only referred as an viable alternative to get approximations of the primal fields –with a reduced computational cost- without resorting to compression.

When discretizing PDEs, the dimension of the underlying computational scheme is artificially determined by the accuracy of the Finite Difference differentiation scheme (Brunton & Kutz, 2017). Higher order schemes produce higher dimensional systems. Essentially, complexities are been traded: from a continuous PDE to a discrete high dimensional system of ODEs. This system can be solved with various methods (p.e. RK-4). This can lead to a high dimensional system, which is computationally expensive to be solved.

Basic idea of ROMs: Our time dynamics are projected onto suitable basis functions (Galerkin projection) to a subspace where the majority of the dynamics takes place in much fewer dimensions and in that way reduce the order of the problem at hand.. A good choice of modal basis elements allows for a smaller set of modes to be chosen to achieve the desired accuracy.

3. Presentation of the cases: flow past one and two cylinders

In the present project the problems of 2D unsteady incompressible viscous flows [Re = 90] past one and two cylinders (cases 1 and 2 respectively) are investigated.

case	1	2
description	flow past a cylinder	flow past two cylinders
# cells [N]	12880	24574
# time steps [M]	2047	1200
time step [sec]	0.001	0.001
# periods used	2	1

Figure 2: Data specifications for the two cases.

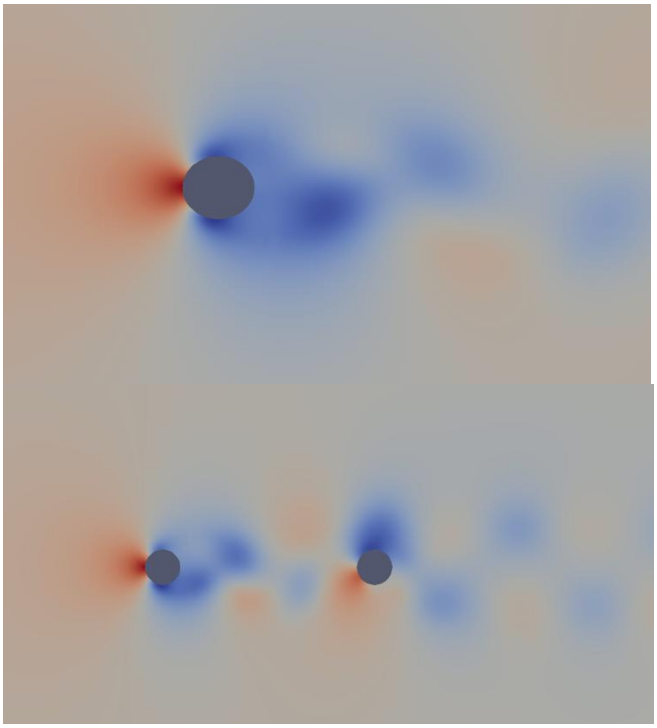
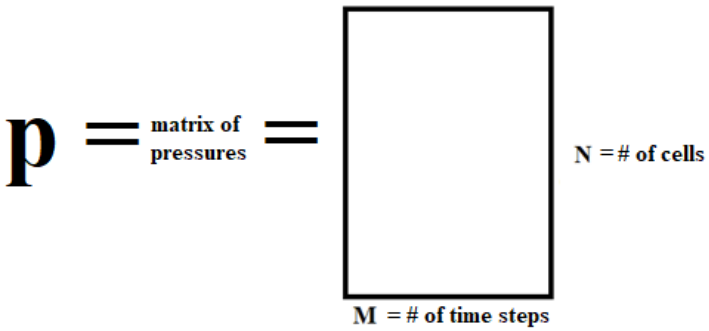


Figure 3: Pressure field of the two cases at t = 0.01 sec

The data are arranged in a pressure matrix as follows: every one of the 2D grid's cell's pressure time series corresponds to a row. This means that every column is actually a snapshot of the full pressure field. (p.e. Element (10,20) of the matrix corresponds to the pressure of the 10^{th} node of the grid after twenty (20) time steps.)



4. Results

4.1. General pre-processing:

The pressure matrix was immediately normalized before the compression, as follows:

$$\mathbf{p} = \left(\frac{1}{\max\{\mathbf{p}\}}\right) \cdot \mathbf{p}$$

Next, centering is necessary for PCA-based methods, as they are designed to find and retain coordinates with the maximum variance within the data. Variance in statistics is

mean-centered because it represents the “spread” of the data in a symmetric kind of way. A centering matrix was used to enforce this:

$$\mathbf{H} = \mathbf{I}_M - \frac{1}{M}\mathbf{1}_M\mathbf{1}_M^T$$

$$\mathbf{P} = \mathbf{p} \cdot \mathbf{H}$$

It is also noted that, in both cases, the compression was executed with respect to space (row-wise) because this is the largest dimension of the pressure matrix (N > M) which means that there is more room for compression.

4.2. Case 1: Flow past a cylinder

4.2.1. LPCA:

Firstly, the rank of the truncation needs to be determined. One commonly used semi-heuristic way to do so is visualized through a semi-logarithmic diagram:

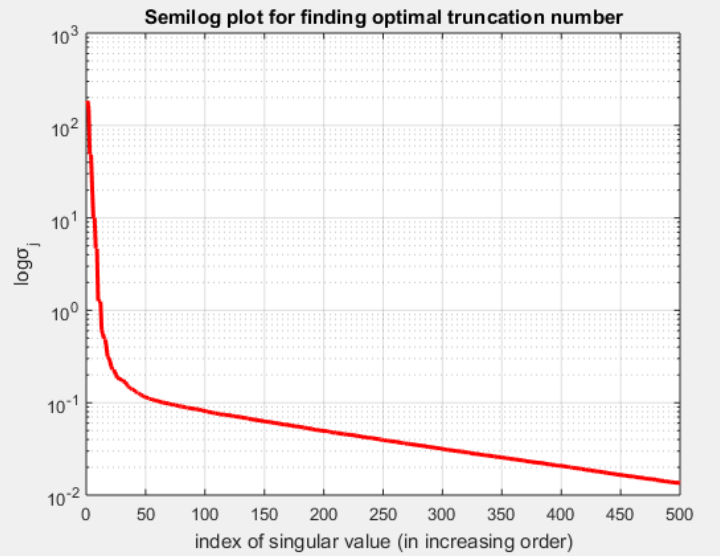


Figure 4: SemiLog plot of “energy" expressed by each singular value (for the first 500/2047 highest energy ones)

Truncation is usually chosen on the «knee» of the graph (which in the logarithmic plot becomes way more apparent than in the normally scaled one). The figure of the percentage of the cumulative “energy” of the dataset, that is expressed while adding an increasing number of singular values, is also helpful in choosing the “right” truncation:

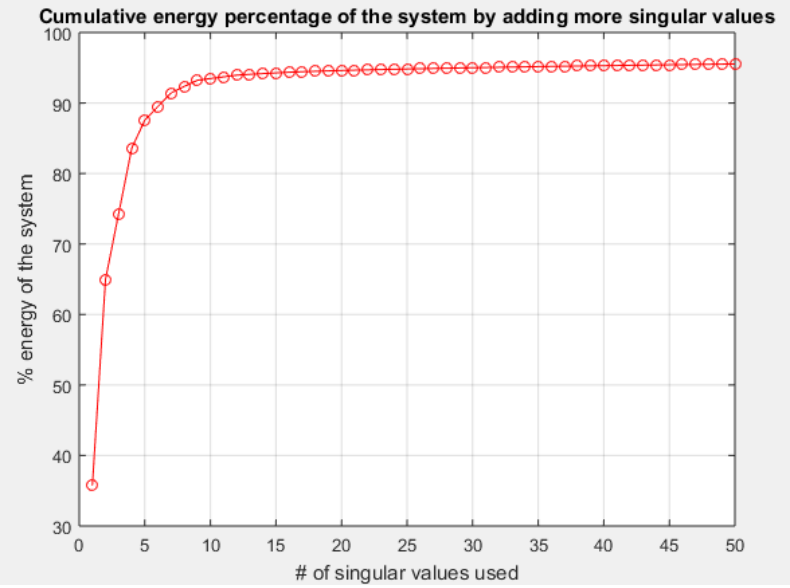
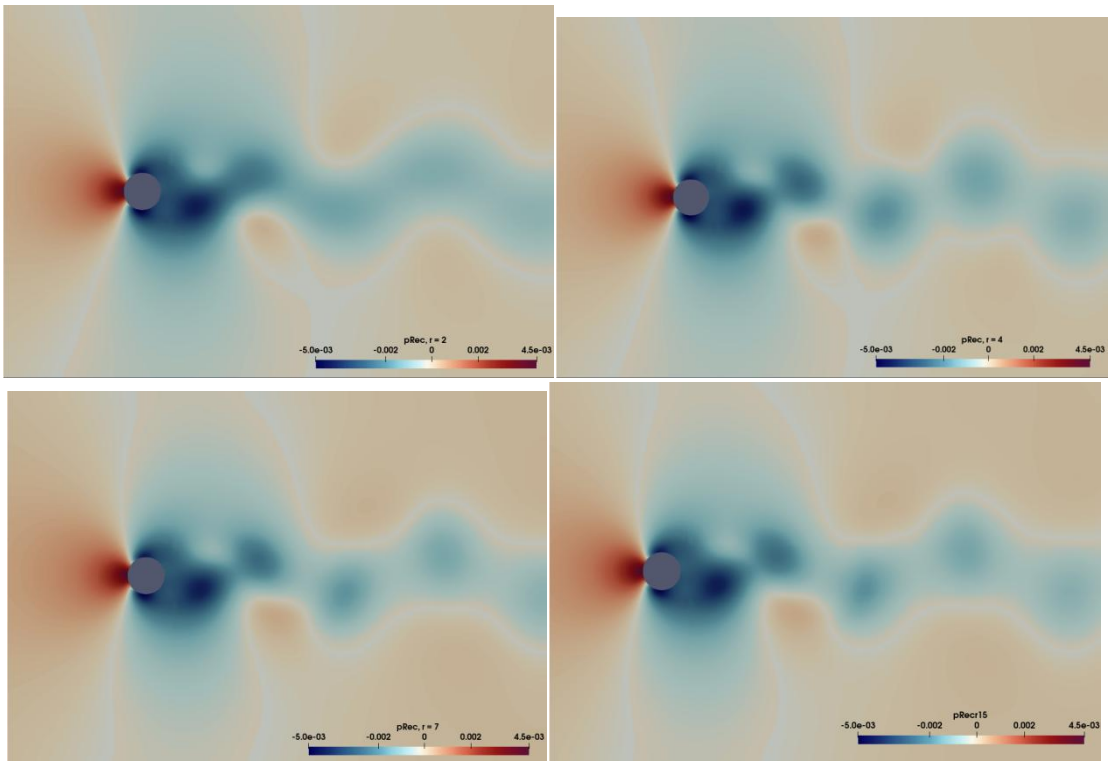


Figure 5: Percentage of cumulative "energy" of the dataset expressed as an increasing number of singular values is used.

In the end, LPCA achieves incredible compression. Retaining only 15 out of 2047 singular values, 95% of the system's "energy" is expressed.

After that, a parametric study of resulting reconstructed fields is performed while increasing the number of modes used [at time t = 0.01 sec]:



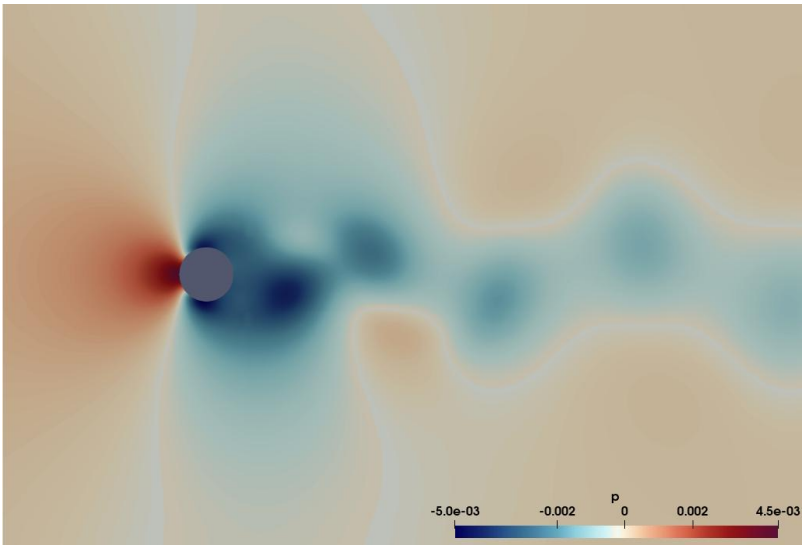


Figure 6: Parametric study: pressure field at t = 0.01 sec. Four reconstruction cases (r=2,4,7,15) and the exact one.

As more modes are retained, the reconstruction gets closer to the actual field. This can be viewed in local structures near the cylinder in the upper part of the figure and in how well defined, in each case, are the ups and downs in the wake of the cylinder.

A more illuminating way to compare the difference made by each chosen truncation is to plot the L2-error for the reconstruction of every snapshot.

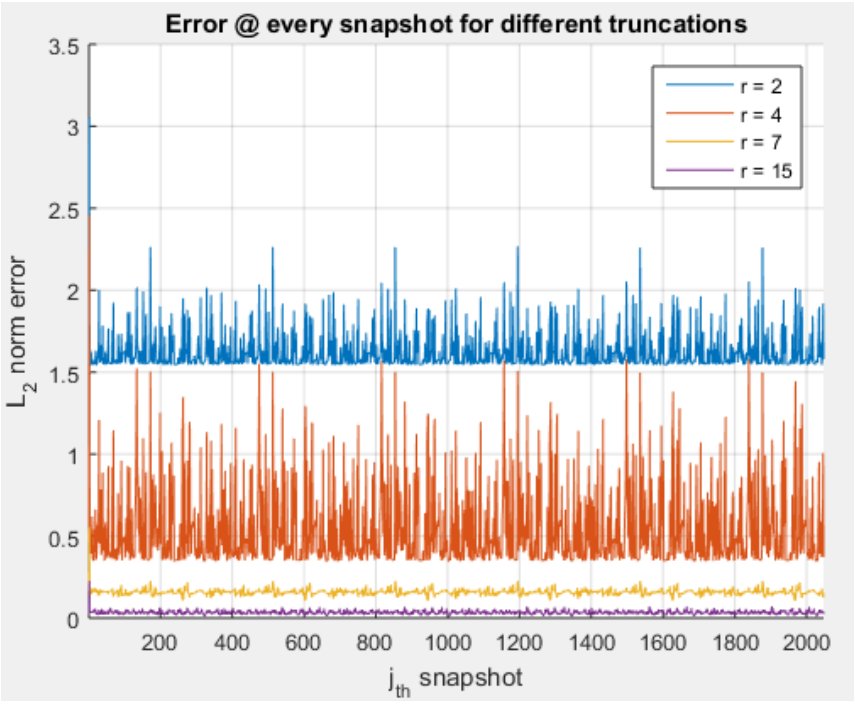


Figure 7: L2-error for every snapshot in time for LPCA, for four different truncations r ={2,4,7,15}

As the number of modes maintained rises, less information is lost, i.e. the error decreases. In the PCA case, higher truncations infer lower errors for every snapshot.

Remark: There is an intrinsic periodicity in the error signals. This is not caused by the obvious periodicity of our phenomenon. After all, only two periods in our data have been kept, but here the emergence of a higher frequency is apparent. This periodicity is caused by the spatial symmetry of the flow. It is revealing to observe that this symmetry-caused periodicity is already captured even with truncation r = 2 (see fig. 8) confirming that as one of the most dominant patterns of the flow, it is represented in the lower modes.

Next, the absolute error at every node at t = 0.01 sec is showcased. The imaging is done in absolute scales, meaning only patterns are examined here; colors are irrelevant. It is reminded that the formula of the absolute error is as follows:

$$E_{abs} = |p_{real} - p_{reconstructed}|$$

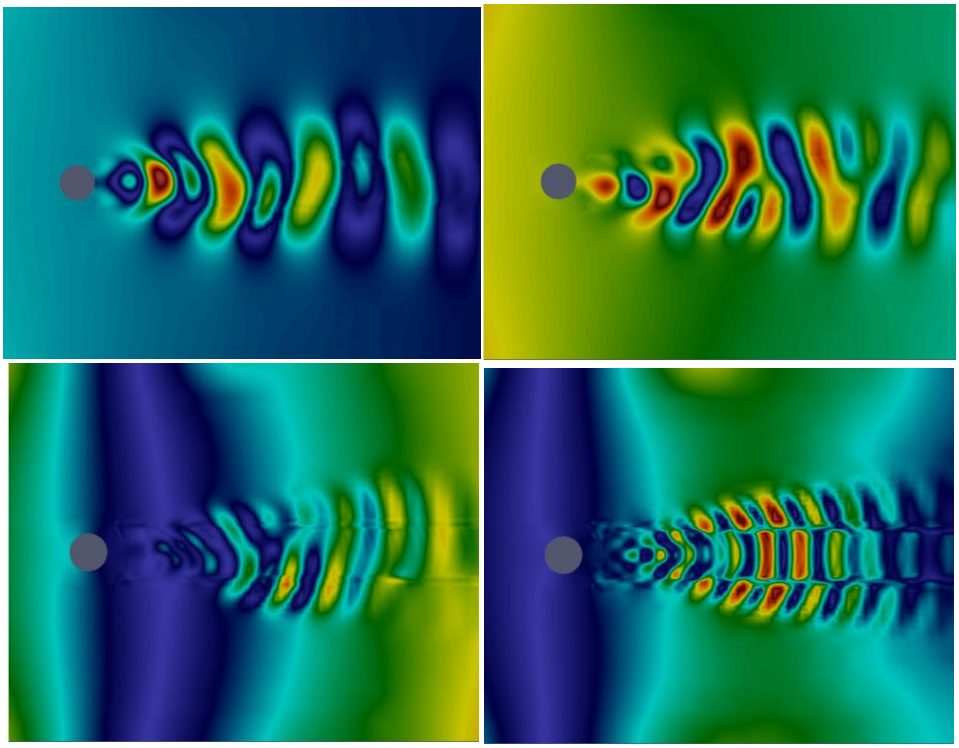


Figure 8: Qualitative study of the absolute error in LPCA, for four different truncations r = {2,4,7,15}.

It is here made clear that as the number of truncated modes r decreases, the absolute error patterns have lower spatial frequencies. This is due to the fact that in PCA the absolute error is the weighted sum of the eigenmodes which were excluded through truncation and higher r-eigenvalues correspond to higher frequencies in LPCA (see following figure). Except for that, there are obviously not enough modes to capture all of the inherent qualitative structure of the phenomenon.

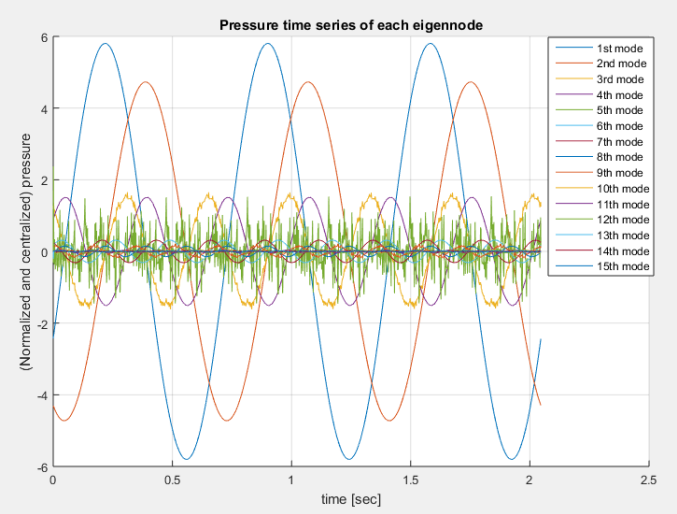


Figure 9: Time series of pressure of each retained projected node.

After that, the quantitative aspect of the results is examined through imaging of the absolute errors in relative scales (color is the important aspect here):

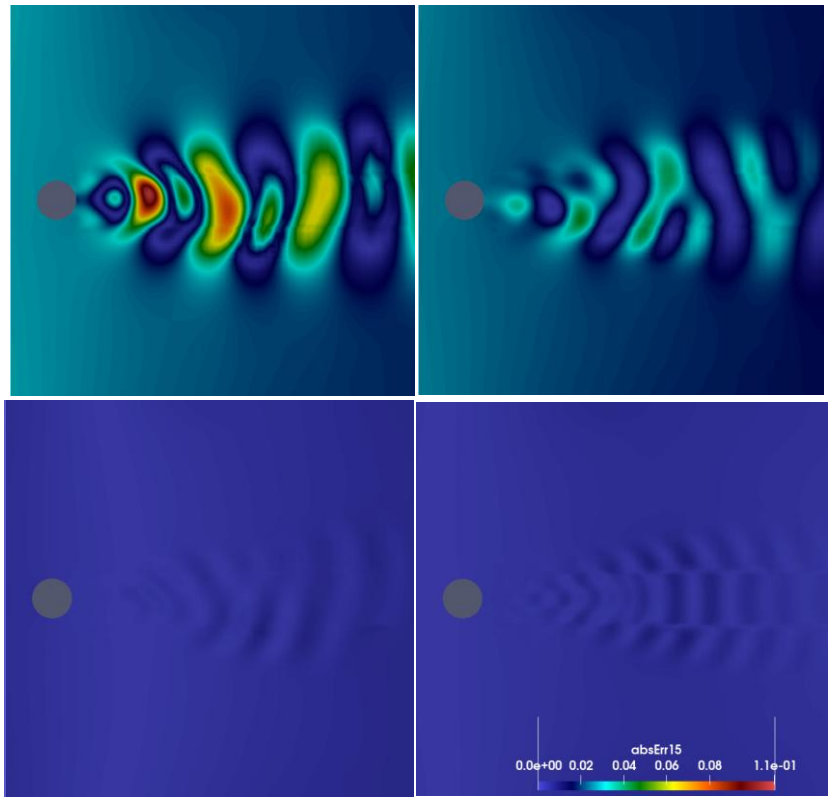


Figure 10: Qualitative study of absolute error in LPCA, for four different truncations: r ={2,4,7,15} .

As expected, as less modes are truncated, the error in quantitative terms is much lower and more diffused.

4.2.2. KPCA

Firstly, the kernel function chosen is the Radial Basis Function (RBF), and that is used to create the kernel matrix:

$$k(\mathbf{p_i}, \mathbf{p_j}) = \exp\left(-\frac{\|\mathbf{p_i}-\mathbf{p_j}\|_2^2}{2\sigma^2}\right)$$

The sigma σ that was used in the present paper was heuristically found balancing two main goals (while accounting for the fact that normalization has been enforced): (1) good reconstruction convergence, (2) most information possible to pass through the EVD of the Kernel matrix.

During the process of the reconstruction it is well established that problems may arise (Bakir et al, 2004), as the fixed point method's convergence may depend on good initialization. This is extremely important in data that are dealt with KPCA because they commonly exhibit non-linearities (that's why we use a non-linear PCA-based method). Indeed, in case 1 the said problem seems to occur. The convergence of the reconstruction algorithm requires an initialization absurdly close to the solution: our initial guess for each snapshot of the field must be no more than 20-25% off with respect to the snapshot of the original data.

$$\mathbf{p_{j_0}} = 0.8\,\mathbf{p_{j_{real}}}, \qquad j = 1{:}\,M$$

Of course, the real data have been thrown away. Else, there is no point in executing compression.

In literature, solutions to this problem include transforming reconstruction to a more complex optimization problem which typically has multiple initializations for increased chance of convergence (Bakir et al, 2004). However, there are two main drawbacks in these methods:

- They further slow down the –already sluggish- reconstruction algorithm
- They must be quite closely adjusted to the specific problem.

The two solutions presented below are quite more easily generalized and do not burden speed as much.

A. Hybrid LPCA-KPCA:

In Hybrid L-KPCA, firstly, LPCA is performed and truncated to a very small r, such that by itself LPCA would create non acceptable reconstructions. Then, KPCA is performed and the LPCA-reconstructed results are used as an initialization of the fixed point method in KPCA reconstruction.

The algorithm is presented in the following flow chart:

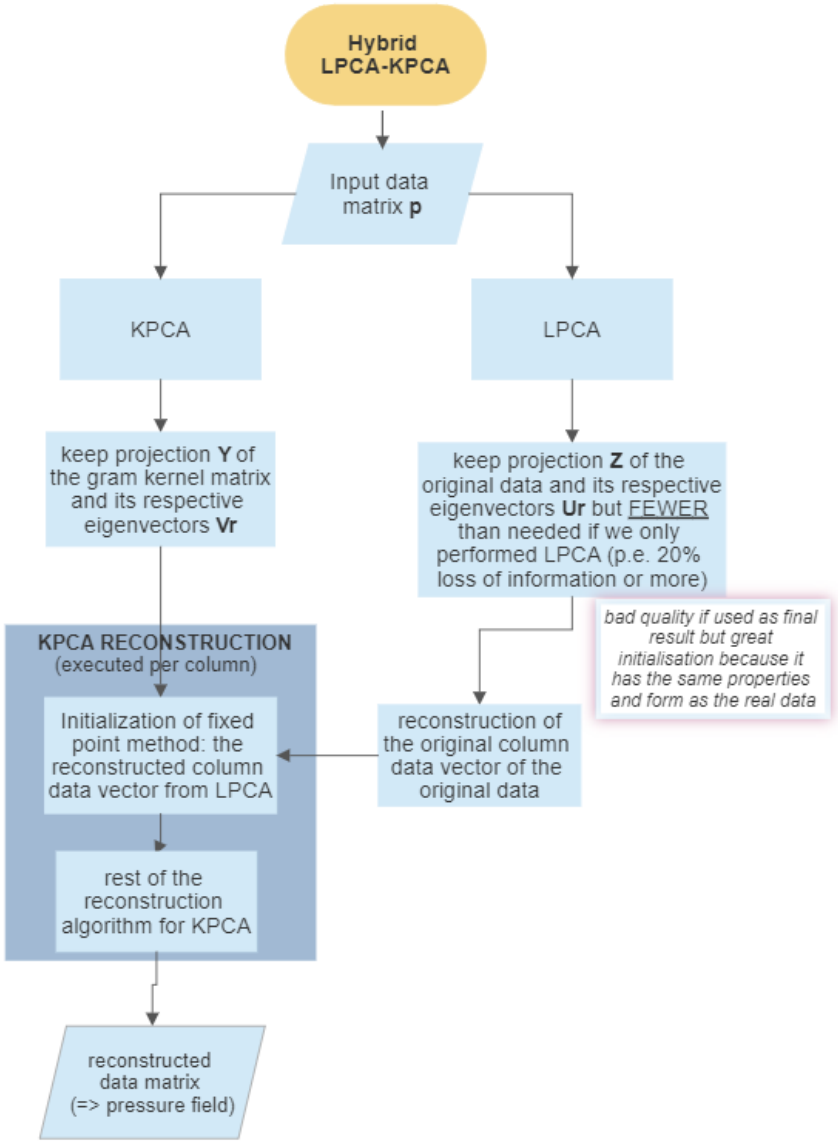


Figure 11: Hybrid LPCA-KPCA algorithm flowchart

B. Smoothing out steep changes of consecutive (in time) values:

This is enforced on our data by the following loop:

for i=1:Niter

for j =1:M

$$\hat{\boldsymbol{p}}_{\boldsymbol{j}} = \omega \cdot \boldsymbol{p}_{\boldsymbol{j}} + \frac{(1-\omega)\left(\boldsymbol{p}_{\boldsymbol{j-1}} + \boldsymbol{p}_{\boldsymbol{j+1}}\right)}{2}$$

end_for

end_for

where $\boldsymbol{p_j}$ are column vectors (i.e. snapshots of the pressure field) of the original data matrix **p**. Both ω which is a relaxation parameter and Niter (the number of iterations used) must be adjusted for our specific problem so that the transformed data is smoothed out enough while ensuring it is not heavily distorted.

An example to the effect of this method for the pressure evolution on a cell for a small time window:

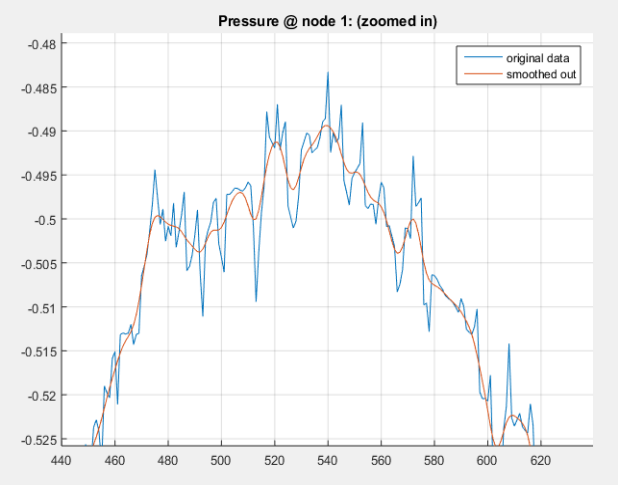


Figure 12: Smoothing out of pressure time series (for a specific time window) @ 1st original node.

Advantages and disadvantages of the two proposed solutions:

*Hybrid**:***

+ Use of the exact original data
+ More powerful compression than LPCA
+ Can be parallelized
- Increased computational cost in doing both LPCA and KPCA
- The quality of our initialization depends on the quality of the results of LPCA

Smoothing out:

+ Lower computational cost
+ More powerful compression than LPCA
- Our data are slightly tampered with
- More hyperparameters to tweak to tailor to each application

Remark: Hybrid method requires less KPCA modes (which means smaller sum loops) resulting to faster reconstruction! This, along with the parallelization

possibility means that both the wall clock time and the CPU time of the two methods are comparable.

After that, a parametric study of KPCA reconstruction for different truncations is performed (r ={1,2,3,15}):

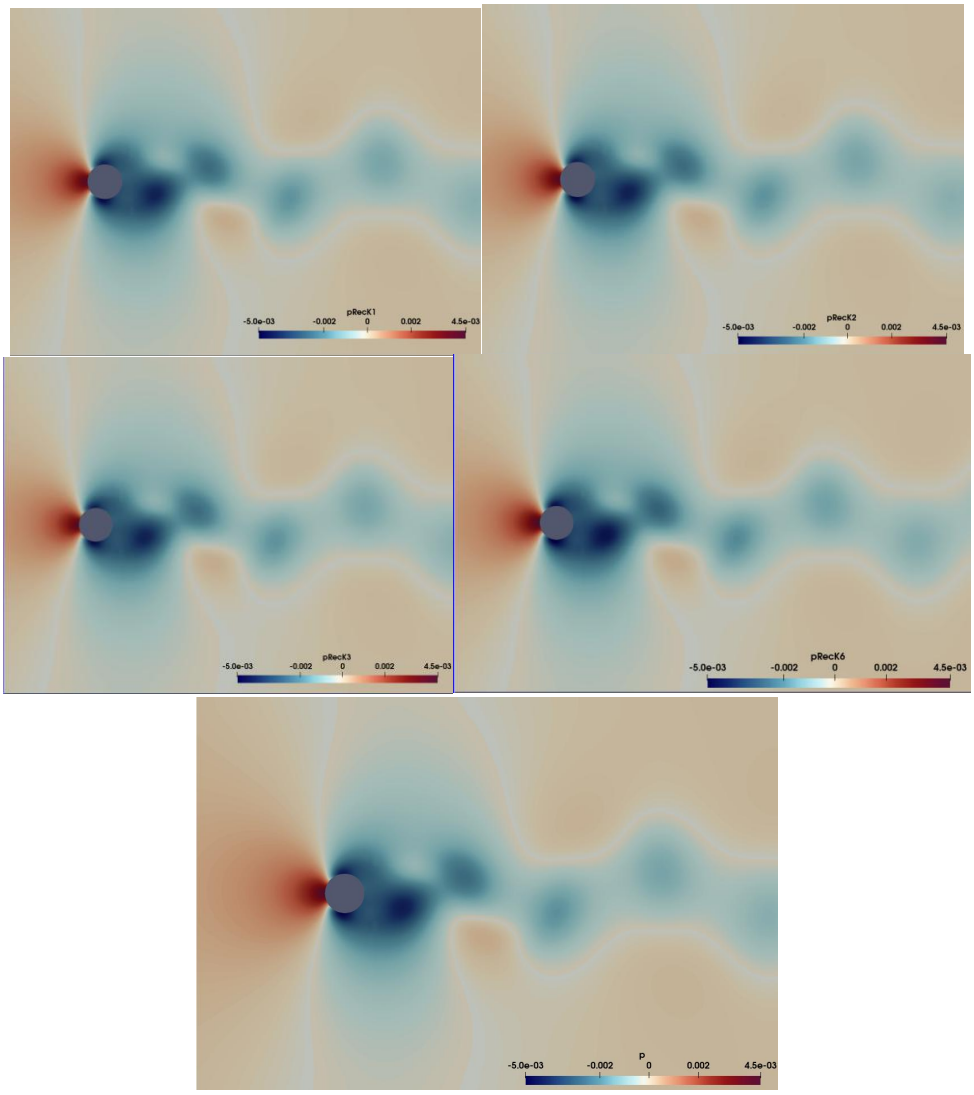


Figure 13: Parametric study, KPCA: pressure field at t = 0.01 sec. Four reconstruction cases (r=1, 2, 3, 15) and the exact one.

It is easily seen that much of the information is already captured by r = 1. Thus, the main patterns are adequately expressed in every case. However, if one looks closely enough, there are local qualitative differences in the four truncation cases. This means that in KPCA, while r increases the changes are mostly quantitative; further reduction of the error in respect to the real field is attained.

Remark: A problem in KPCA is that the "right" truncation is not always apparent, which means that –slow- heuristic methods may be required.

Next, the absolute error at every node at t = 0.04 sec is showcased. The imaging is done in absolute scales, meaning only patterns are examined here; colors are irrelevant.

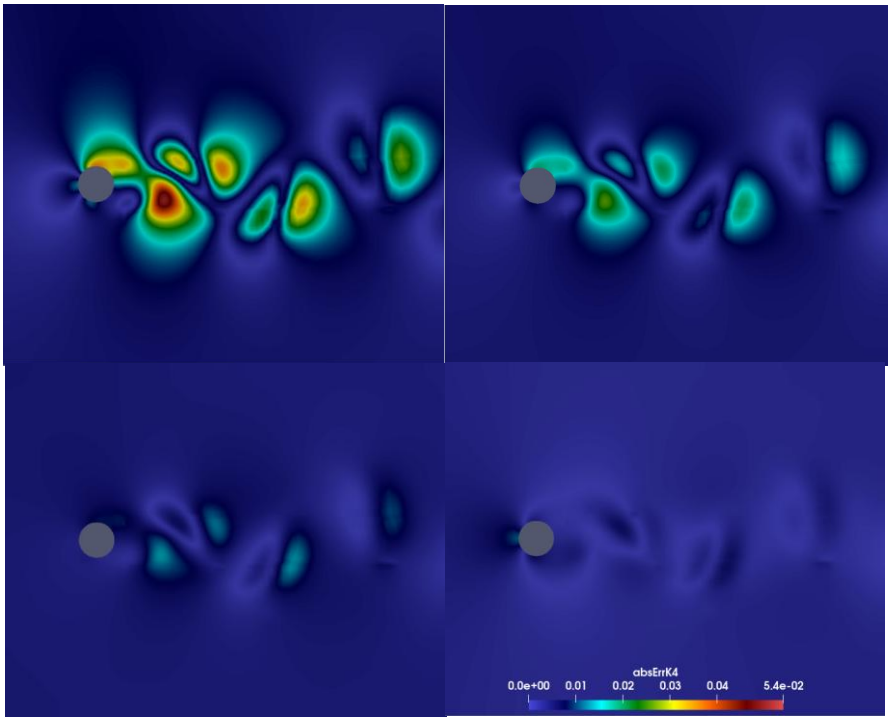


Figure 14: Both qualitative and quantitative imaging of the absolute error in all fields @ t= 0.04 for four different truncation cases (r=1, 2, 3, 14).

Again, it is observed that the qualitative results are excellent even in very low truncation (r=1), as the error is distributed on the patterns of the wake in a similar way -mainly in the "steep" spots- as the patterns produced by the phenomenon itself. It also seems that the absolute error is more diffused before the cylinder and more compact in the wake of the cylinder.

4.2.3. Comparisons between LPCA and KPCA in case 1

In order to compare LPCA and KPCA some parameters are fixed and set as equal and others, which are the ones compared are free to change.

Firstly, truncation is chosen to the same low number of modes r = 2. This results in:

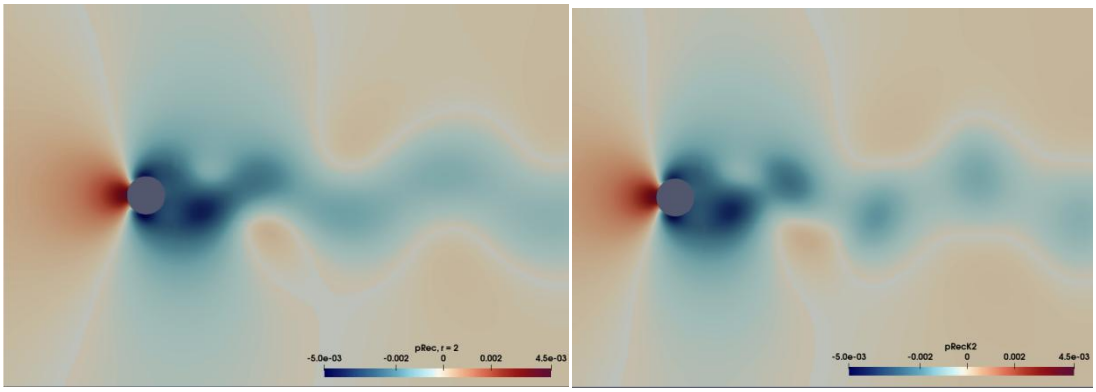


Figure 15: LPCA and KPCA reconstruction for same truncation: r = 2.

The local steep ups and downs of the wake of the pressure field in LPCA are captured in a smoothed out way (as if linearized), whereas already by r = 2, KPCA manages to capture the non-linearity of these local phenomena! In KPCA there is also smaller L2-error and an overall closer similarity to the actual field.

Then, same percentage of "energy" of the system is retained for both LPCA and KPCA. Examining the difference in the absolute errors (between the exact and approximate solutions) of the pressure fields, we can conclude that the qualitative features of the information lost are quite different (fig. 19). This may result to better or worse out of sample predictions, thus in better or worse generalizability which can make a huge difference in the quality of our final model. Validated by the figure below, the same L2-error occurs approximately in seven retained modes of LPCA and four retained modes in KPCA (which means that about 6-8% of the information is lost):

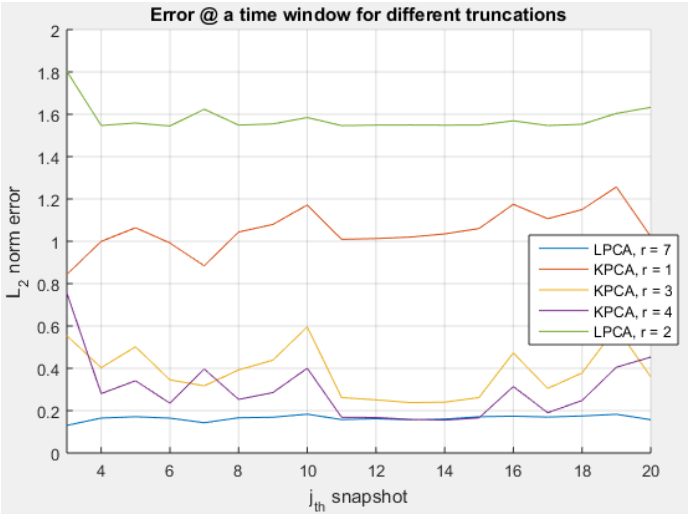


Figure 16: Comparative diagram of L2-error in both LPCA and KPCA truncations.

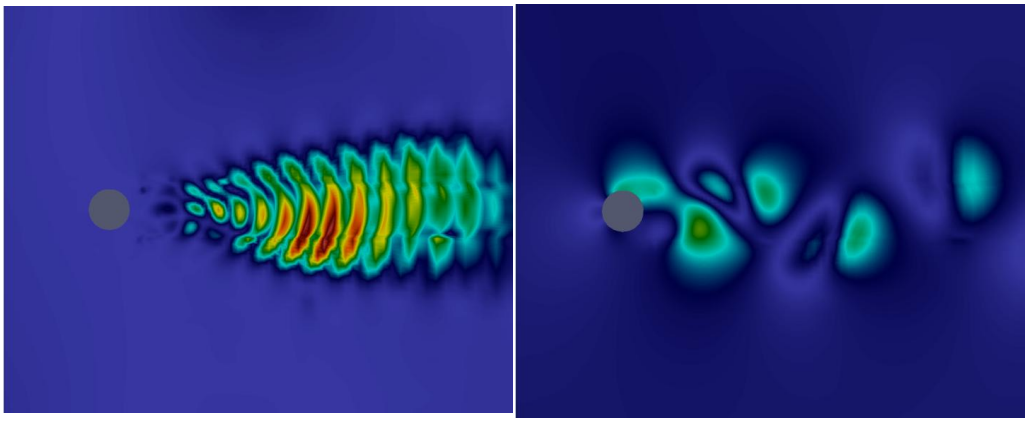


Figure 17: Qualitative distribution of absolute error of pressure fields in LPCA and KPCA reconstruction respectively, for same percentage of lost information (~7%).

As expected, it is easy to see that KPCA is superior in this aspect. In LPCA the error is of both qualitative and quantitative character, whereas in KPCA the error is almost exclusively quantitative.

4.3. Case 2: Flow past two cylinders

This case is briefly examined, mainly as a confirmation of the results and conclusions drawn by case 1.

4.3.1. LPCA

Firstly, the right truncation must be chosen in a similar way to case 1:

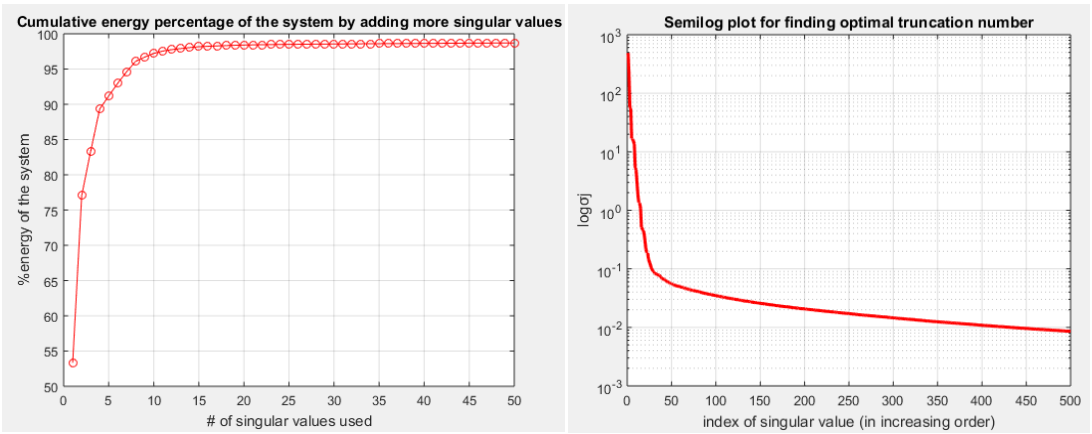


Figure 18: Cumulative "energy" captured as more singular values are used and semiLog plot of "energy" expressed by each singular value (for the first 50/2047 and 500/2047 highest energy ones respectively)

Even though the problem may seem "less" linear than case 1, this is not accurate: the intrinsic structure is captured with even less modes. It seems that 7 out of 24880 modes hold approximately 95% of the "energy" of the system.

Next, the quantitative aspect of the results is examined through imaging of the absolute errors in relative scales (color is the important aspect here) for truncations r = {2, 7}:

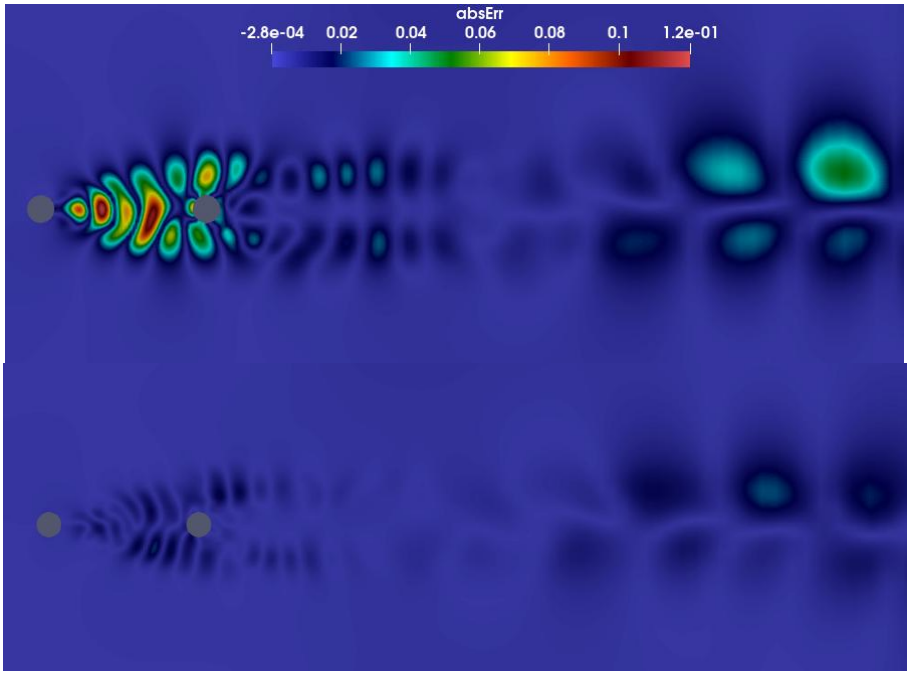


Figure 19: Quantitative study of absolute error in LPCA, for twp different truncations: r = {2, 7}.

4.3.2. KPCA

The kernel function used and the algorithm implemented are exactly the same as case 1.

In case 1, it has been established that KPCA reconstruction may turn out problematic. However, in case 2, convergence of the fixed point method is easily attained without resorting to methods A and B previously proposed, by just implementing (under)relaxation. Going from one snapshot to the next runs without any issues.

4.3.3. Comparisons between LPCA and KPCA in case 2

The qualitative differences of LPCA and KPCA are again showcased by demonstrating the L2-error in absolute scale. This means that only patterns are examined here –and colors are irrelevant. The same L2-error occurs in 7 retained modes of LPCA and 2 retained modes in KPCA (when 5-7% of the information is lost), as shown by the figure below:

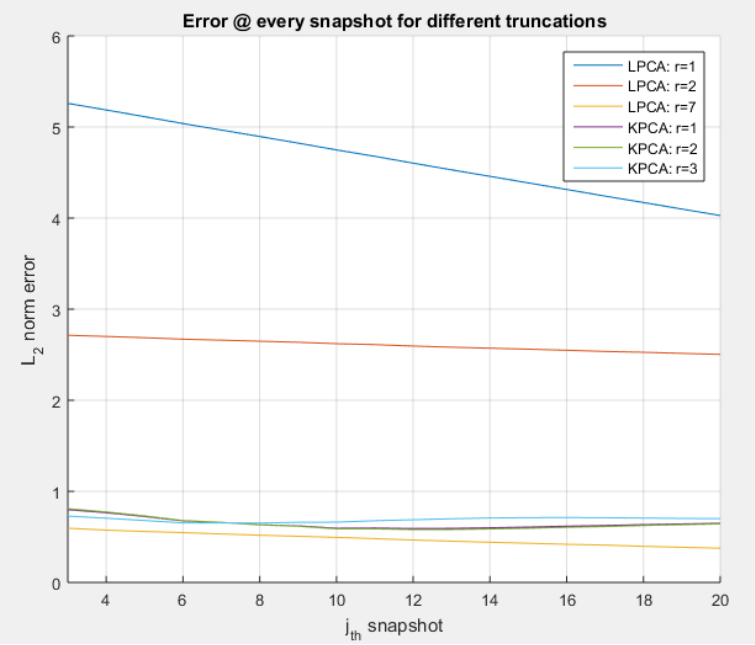


Figure 20: Comparative diagram of L2-error in LPCA and KPCA truncations.

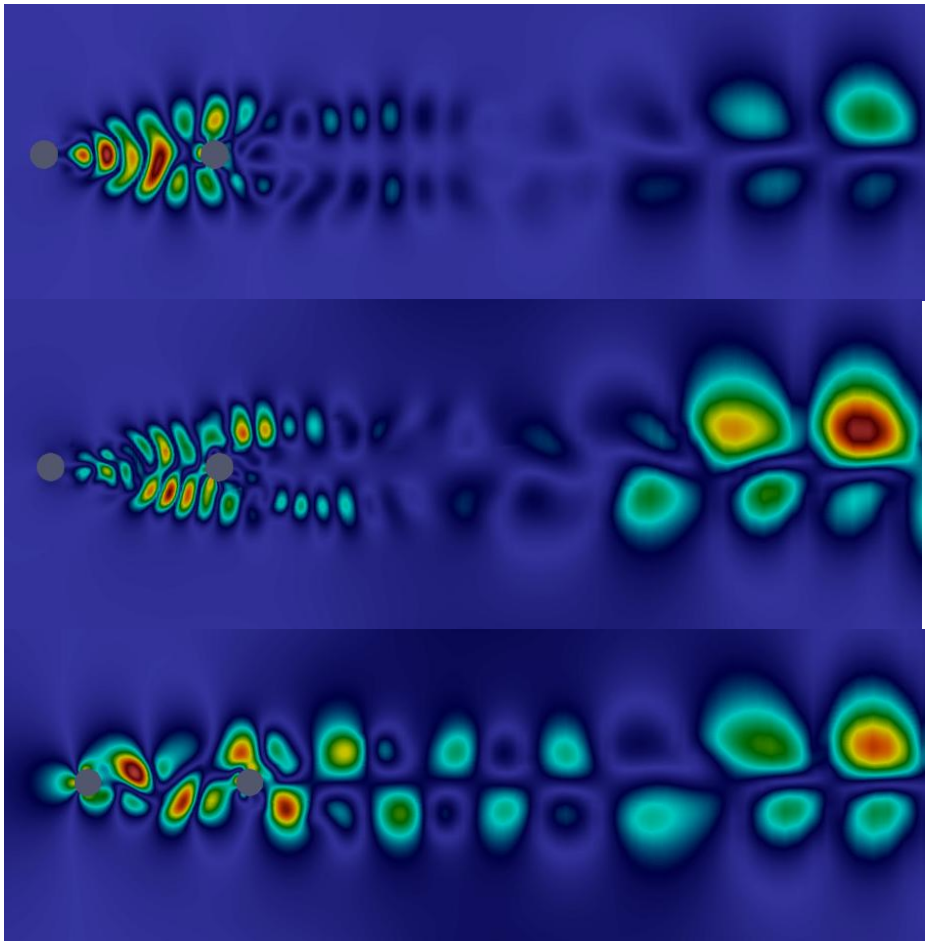


Figure 21: Qualitative differences between LPCA r = {2, 7} and KPCA r = {2}.

The conclusions drawn from case 1 are validated. However, in the present case, LPCA is behaving relatively better than before, as the system is sufficiently approximated with fewer modes. It is also observed (fig. 22) that the L2-error in both LPCA and KPCA has far less variance than in case 1. This must be a result of the fact that the data has in general lower dimensional behavior.

Finally, the advantages and disadvantages of LPCA and KPCA can be summed up (for both case 1 & 2):

LPCA:

+ Extremely fast and has small computational cost
+ Deals best with data lying on linear subspaces
+ Exact pre-imaging
+ Easily implemented
- Does not deal well with non-linear phenomena

KPCA:

+ Deals with non-linearity
+ Can achieve bigger CR or better quality results for same truncation
+ Qualitatively better results
- Approximate pre-imaging
- Possibly problematic reconstruction
- Way bigger computational cost
- More hyperparameters to tweak, tailored to each application
- Truncation may need to be done heuristically

4.4. Incremental SVD

Trying out iSVD in case 1, the final singular values discovered with full SVD and iSVD are incredibly close (starting with 100 initial snapshots and updating up until M = 2047 for iSVD):

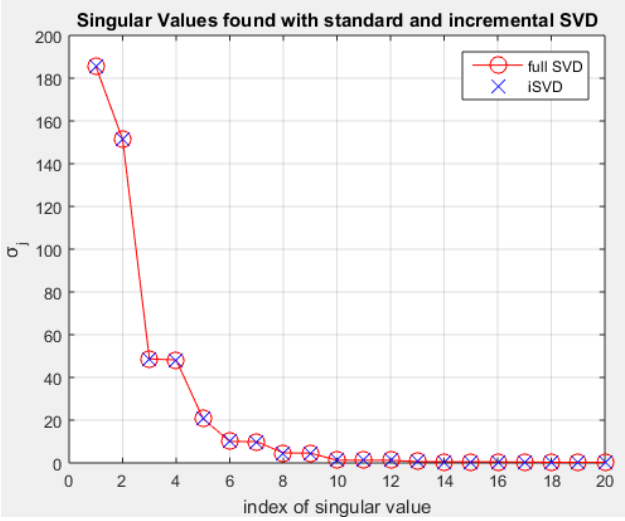


Figure 22: Singular values found through full and incremental SVD

This means that when implementing iSVD, the results would be extremely close to these presented in section 4. It is noted that a small code for iSVD has been developed in MATLAB and added to Appendix B.

In theory, an incremental version of KPCA (Chin et al, 2007 and Wang et al, 2021) could also be implemented to approximate these time instants, facilitating the expansion of this method's great performance to inherently non-linear data.

5. Another possible application: projection of PDEs to POD modes

Another way to utilize PCA is to directly apply it to the PDE at hand, to create a Reduced Order Model (ROM). As discussed in section 2, ROMs are all about choosing a basis: PCA (SVD) or KPCA could be used to find an optimal basis in a data driven way.

SVD:

Starting from a data matrix $\mathbf{X} = [\mathbf{x_1}\ \mathbf{x_2}\ \dots \mathbf{x_n}]$,

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\boldsymbol{V^T}$$

And as already discussed an optimal rank = r approximation (where r << max(N,M)) can then be made:

$$\mathbf{X_r} = \mathbf{U_r}\boldsymbol{\Sigma_r}\boldsymbol{V_r^T} \cong \mathbf{X}$$

Then, choosing **Ur** as a basis:

$$\boldsymbol{\Psi} = \mathbf{U_r}$$

The columns of matrix Ur contain the orthogonal modes necessary to form the ideal basis.

Remark: A big advantage of this method is that it is equation-free; meaning that our dynamics might as well be a black box and the choice of modes will still work.

Galerkin projection onto POD modes:

PDEs can also be expressed as a sum of a linear and a non-linear part:

$$\mathbf{u}_t = \mathbf{L} \cdot \mathbf{u} + \varepsilon \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \dots, x, t, \boldsymbol{\beta}) \quad \text{(II)}$$

In the continuous space, modal expansion can be expressed as (through a Galerkin expansion):

$$\boldsymbol{u}(t) \cong \boldsymbol{\Psi} \cdot \boldsymbol{\alpha}(t)$$

Plugging this into (II) and multiplying by $\boldsymbol{\Psi}^{\mathbf{T}}$:

$$\frac{\mathrm{d}\mathbf{a}(t)}{\mathrm{d}t} = \boldsymbol{\Psi}^{\mathrm{T}}\mathbf{L}\boldsymbol{\Psi}\boldsymbol{\alpha}(t) + \boldsymbol{\Psi}^{\mathrm{T}}\mathrm{N}(\boldsymbol{\Psi}\boldsymbol{\alpha}(t), \boldsymbol{\beta})$$

The linear part is of dimension rxr (which is our choice of truncation) and is computed not on every time step but only once in the beginning. The non-linear part at first glance needs to be computed at every instant but there are methods, like Gappy POD, principled sparse sampling or greedy selection techniques that solve this problem.

6. Discussions and Conclusions

The compression rate is only part of the story. KPCA achieves a much bigger compression but -even disregarding the computational cost- as remarked throughout the span of this project there are many important aspects as to why one may choose to implement LPCA or KPCA.

LPCA is an incredibly fast and easily implemented algorithm. It is a great tool in problems that produce data that lie on a linear subspace (which are many more than strictly “linear problems”). Even if they are not found in their entirety on such a subspace, it can find linear substructures that best explain the data within these limitations. If certain conditions are met it may also serve as a good tool for initialization of reconstruction attempts of other non-linear dimensionality reduction/compression techniques.

KPCA is a much slower algorithm, with more fractious implementation, requiring fine tuning, experience and sometimes even a general good sense by the user. However if properly applied and possible occurring problems are dealt with, it can achieve great results in non-linear problems: bigger compression rates than LPCA for same truncation or far better capture of the inherent patterns for the same final error, that is also distributed in a less distorting way resulting to more generalizable models.

In general, when one faces a problem, LPCA should be first run, its results be checked and if the cumulative error plot does not produce a defined "knee" –which probably means that the problem has inherent non-linear structure- or if the compression rate or the magnitude of the error are not deemed sufficient, KPCA may be the only alternative to getting great compression, a good model and a relatively straightforward method for finding pre-images without resorting to Neural Networks.

References

• Scholkopf, B., Smola A. et al (1998). *Nonlinear Component Analysis as a Kernel Eigenvalue Problem*, Neural Computation 10, 1299–1319 (1998)

• Scholkopf, B., Mika, S. et al (1998) *Kernel PCA Pattern Reconstruction via Approximate Pre-images*, Available at: https://www.researchgate.net/publication/2949760_Kernel_PCA_Pattern_Reconstruction [accessed 25 July 2022]

• Kapsoulis, D. (2019). *Low-cost metamodel-assisted evolutionary algorithms with applications in shape optimization in fluid dynamics*, NTUA, Athens, GR, Retrieved from https://www.didaktorika.gr/eadd/handle/10442/46420?locale=en

• Griewank A., Walther A (2000). *Algorithm 799: Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation*. ACM Transactions on Mathematical Software (TOMS).2000;26(1):19-45.

• Brunton, S., Kutz, N. (2017). *Data Driven Science & Engineering*. 1st ed. Cambridge Publication Press

• Ham, J., Lee, D.D, Mika, S. & Schölkopf, B.. (2004). *A kernel view of the dimensionality reduction of manifolds.* Max Planck Technical Report No. TR-110. 10.1145/1015330.1015417.

• Bakir, G.H, Weston, J. & Schölkopf, B. (2004). *Learning to find pre-images*. Available at: http://www.thespermwhale.com/jaseweston/preimage.pdf [accessed 27 July 2022]

• Brand, M. (2002). Incremental *Singular Value Decomposition of Uncertain Data with Missing Values*. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds) Computer Vision — ECCV 2002. ECCV 2002. Lecture Notes in Computer Science, vol 2350. Springer, Berlin, Heidelberg.

• Vezyris, C., Papoutsis-Kiachagias, E., Giannakoglou, K. (2019). *On the Incremental Singular Value Decomposition Method to Support Unsteady Adjoint-Based Optimization.* Available at: http://velos0.ltt.mech.ntua.gr/research/docs/Vezyris_IncrementalSVD_2019.pdf [accessed 29 July 2022]

• Chin, Tat-Jun & Suter, David. (2007). *Incremental Kernel Principal Component Analysis.* IEEE transactions on Figure processing: a publication of the IEEE Signal Processing Society. 16. 1662-74. 10.1109/TIP.2007.896668.

• Wang, YongGuang & Yao, ShuZhen & Xu, Tian. (2021). *Incremental Kernel Principal Components Subspace Inference With Nyström Approximation for Bayesian Deep Learning*, IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3062747.

Appendix A: Supplementary figures

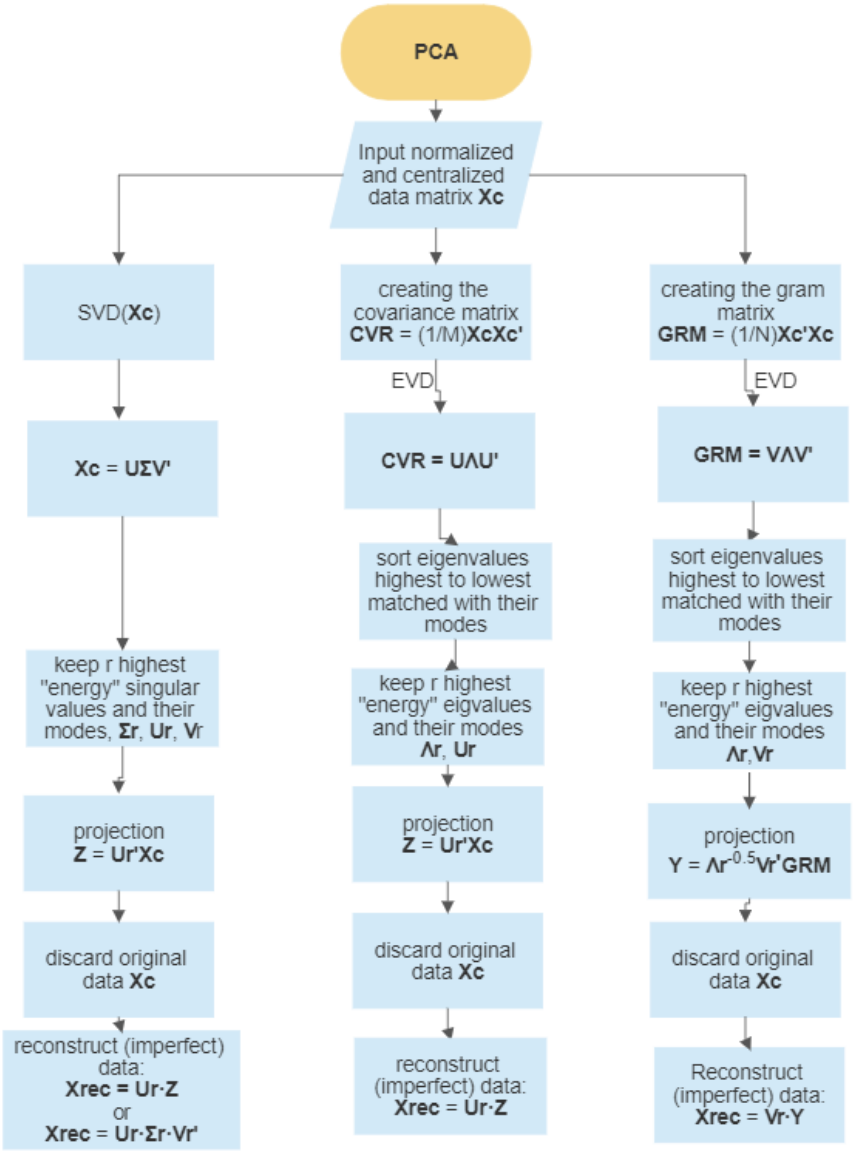


Figure A1: Flow chart of three useful PCA derivations

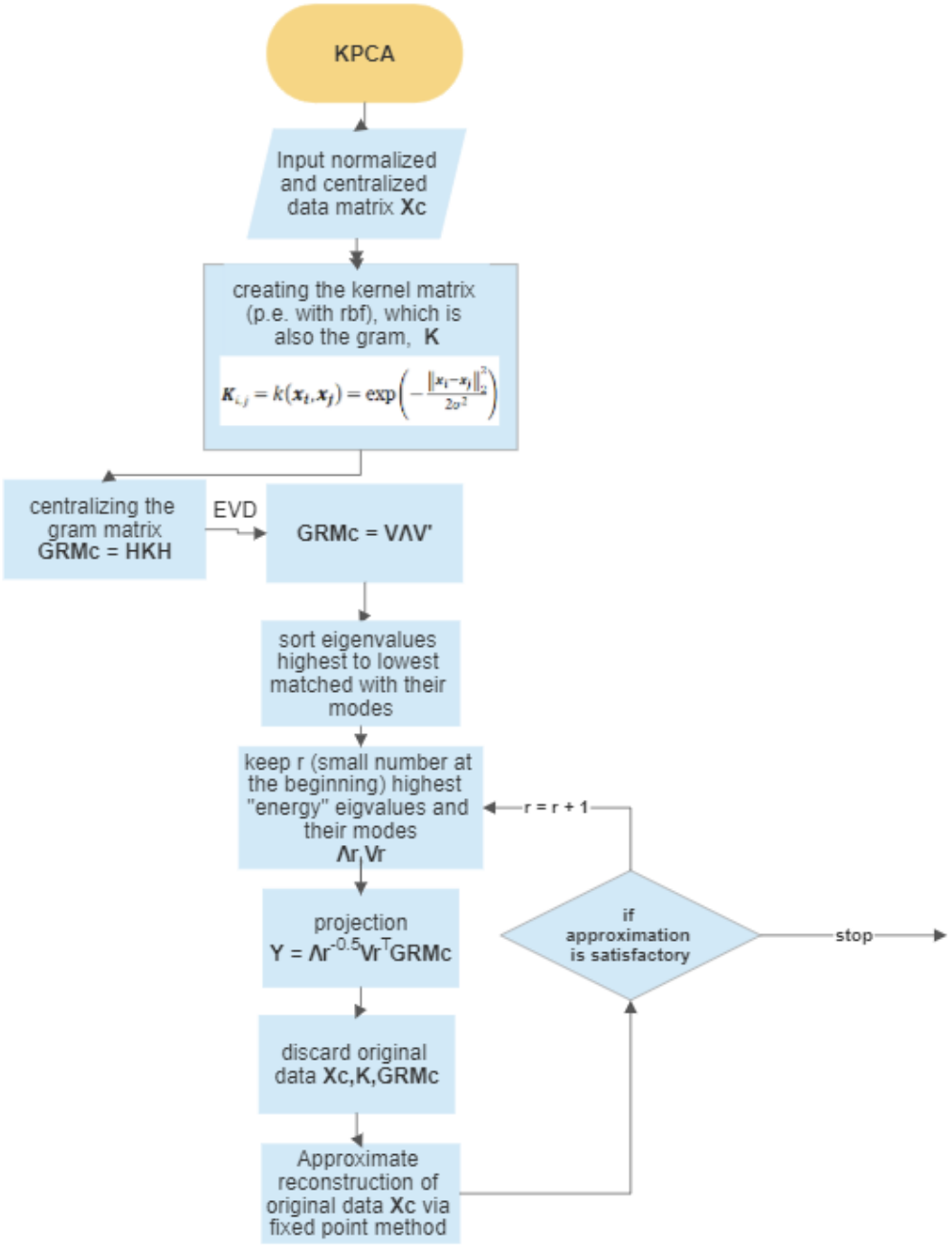


Figure A2: KPCA algorithm flow chart

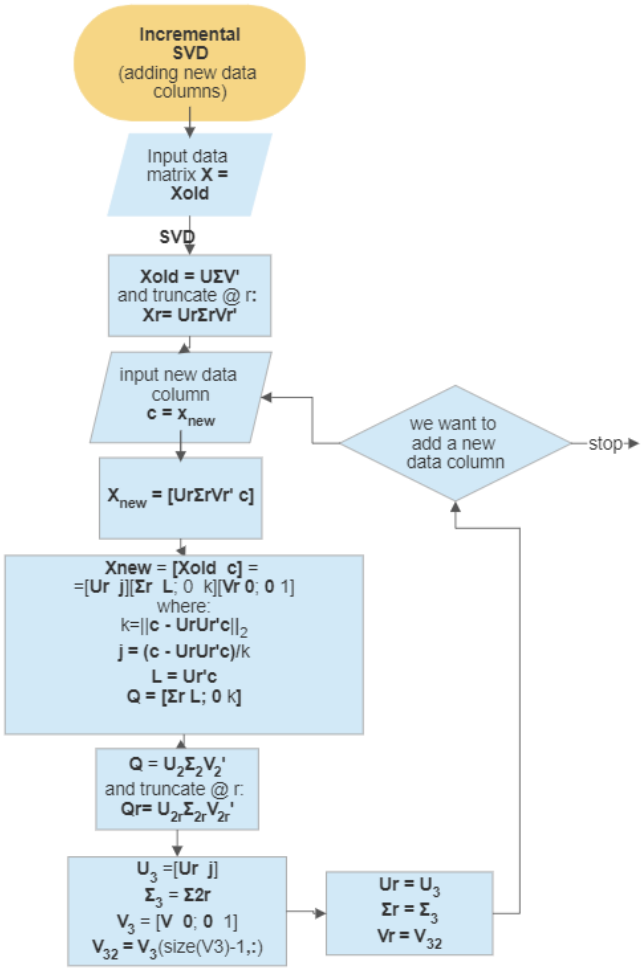


Figure A3: iSVD algorithm flow chart

Appendix B: Code in MATLAB

```
%MAIN CODE PCA AND KPCA
%% DIMENSIONALITY REDUCTION WITH LPCA and KPCA

%% - INPUTS

load('p.mat'); %shortcut for my data only

%% initialisations

N = size(p,1);
M = size(p,2);
X = p;
X = p/max(max(abs(p)));

%----------------------

K = zeros(M,M);
dt = 0.001; % from CFD Simulation
time_vec = 0:dt:(M-1)*dt;
L2_norm = 0;
```

```
sigma =0.99; % for kernel
dx = 0.01;
x_vec = 0:dx:(N-1)*dx;


%% SOLVING KPCA RECONSTRUCTION PROBLEM: smoothing out

rlx = 0.5;
X_temp2(:,1) = X(:,1);
X_temp2(:,M) = X(:,M);
X_temp2 = X;
for iter =1:10
   for j=2:(M-1)

       X_temp2(:,j) = rlx*X_temp2(:,j) + (1-rlx)*((X_temp2(:,j-1) +
X_temp2(:,j+1))/2);
   end

end

figure(100)
hold on
plot(X(1,:))
plot(X_temp2(1,:))

figure(101)
hold on
plot(X(9609,:))
plot(X_temp2(9609,:))

col = X(:,1);
col2 = X(:,2);
differ = col2./col;
max(differ)

col = X_temp2(:,1);
col2 = X_temp2(:,2);
differ = col2./col;
max(differ)

X = X_temp2;

%% General pre-processing
H = eye(M) - (1/(M))*ones(M); % centering matrix
X_c = X*H; %centered data matrix -column wise-;

%%     -LINEAR PCA
% ---------- THROUGH COVARIANCE ------------
cvr = (1/M)*(X_c*X_c'); %covariance matrix

[U,L] = eig(cvr); %EVD of covariance matrix

[~,idx]=sort(diag(L),'descend'); %sorting
 L2=L(idx,idx);
U2 = U(:,idx);

sr = 15; % truncation point
```

```
Z = U2(:,1:sr)'*X_c;  % projection
X_rec = U2(:,1:sr)*Z; %reconstructed

% -------- THROUGH SVD (SPEEDY) ---------

[Us,Ss,Vs]= svd(X_c,'econ');

figure(2)
plot(diag(Ss)/sum(diag(Ss))*100, 'ro')
title('singular values for linear pca (via svd)')
figure(3)
plot(cumsum(diag(Ss))/sum(diag(Ss))*100, 'ro')
figure(25)
semilogy(diag(Ss(1:500,1:500)), 'r-', 'LineWidth', 2)
grid on
title('Semilog plot for finding optimal truncation number')
xlabel('index of singular value (in increasing order)')
ylabel('logσj')

sr2 = 15; %truncation point
Ur = Us(:,1:sr2);
Sr= Ss(1:sr2,1:sr2);
Vr = Vs(:,1:sr2);

Xr = Ur*Sr*Vr'; % reconstructed


%% KPCA main
sigma =0.3;
% ---------  Kernel Matrix: RBF kernel function
for k = 1:M % spatial nodes

    for m= 1:M % time steps


        L2_norm = sum((X_c(:,k)-X_c(:,m)).^2);

        K(k,m) = exp(-L2_norm/(2*sigma^2));

    end
end


gram_k_c = H*K*H; %gram matrix, kernelized, mean centered


[V,LAMDA] = eig(gram_k_c); % EVD of gram matrix

 [~,idx]=sort(diag(LAMDA),'descend'); %sorting
 LAMDA2=LAMDA(idx,idx);
V2 = V(:,idx);

% -- singular values, energy, truncation point exploration
SING2 = sqrt(abs(LAMDA2(:,:)));
figure(3)
plot(cumsum(diag(SING2))/sum(diag(SING2))*100, 'ro')
title('singular values for kpca')
```

```
% truncating to rank=r
r = 15;
V2_r = V2(:,1:r); % truncated eigenvectors
LAMDA2_r = LAMDA2(1:r,1:r);

%% PROJECTION ON TRUNCATED EIGENVECTORS

Y = (LAMDA2_r)^(-0.5)*V2_r'*gram_k_c; % projection on feature space

reconstrK = (V2_r)*Y; %gram matrix reconstruction

%% KPCA RECONSTRUCTION

r =6;
BETA = Y;
Z_new = zeros(size(X_c));
omega = 0.01;
sr3 = 8; % LPCA truncation for initialization
resid2final = zeros(M,1);
mn  = mean(X')'*ones(1,M);

for j = 1:M
%     z_old = 1.2*X_c(:,j);
    z_old = 0.9*X(:,j);
%     z_old = 0.99*(Ur(:,1:sr3)*Zr(1:sr3,j) + mn(:,j)); %     z_old =
0.99*Ur(:,1:sr3)*Zr(1:sr3,j)*(10^4);
%     if j == 1
%         z_old = 0.95*X(:,1);
%
% %     elseif j == 2
% %         z_old = 0.95*X(:,2);
%     else
%         z_old = Z_new(:,j-1);
%     end
    epsilon = 1;
    resid2new = 40;
    resid2old = 10;
    while abs(resid2new-resid2old) > 0.0001

        resid2old = resid2new;
        sum_out1 = zeros(size(X_c(:,j)));
        sum_out2 = 0;
        for k = 1:r
                sum_in1 = zeros(size(X_c(:,j)));
                sum_in2 = 0;
            for i = 1:M
                L2 = sum((X(:,i)-z_old).^2);
                sum_in1 = sum_in1 + V2_r(i,k)*exp(-
L2/(2*sigma^2))*X(:,i);
                sum_in2 = sum_in2 + V2_r(i,k)*exp(-L2/(2*sigma^2));

            end
            sum_out1 = sum_out1 + BETA(k,j)*sum_in1;
            sum_out2 = sum_out2 + BETA(k,j)*sum_in2;
        end

        z_new_star = sum_out1/sum_out2;
```

```
        z_new = omega*z_new_star + (1-omega)*z_old;

        epsilon = max(abs(z_new - z_old)); % conv. criterion
        resid2new = sum((z_new - X(:,j)).^2) % convergence criterion

        z_old = z_new;

        resid2final(j) = resid2new;
    end
    Z_new(:,j) = z_new;
end
```

% iSVD code

```
%% INCREMENTAL SVD

r12 = 20;  % pre-selected truncation rank
X_cOLD = X_c(:,1:100); % initially available snapshots
[U1s,S1s,V1s] = svd(X_cOLD, 'econ'); % SVD performed in initial
snapshots
% truncating to pre-selected rank
U1s = U1s(:,1:r12);
S1s= S1s(1:r12,1:r12);
V1s = V1s(:,1:r12);

% Updating our SVD
for j = 101:1:M

    c = X_c(:,j); % new data column

    L = U1s'*c; %projection of new data to old spanned space
    k = sqrt(sum(((c-U1s*U1s'*c)).^2));
    Q1 = [S1s L];
    Q2 = zeros(1,size(Q1,2)-1);
    Q = [Q1; Q2 k];

    [U12,S12,V12] = svd(Q,'econ'); % svd of the incremented singular
values matrix

    U12= U12(:,1:r12);
    S12= S12(1:r12,1:r12);
    V12 = V12(:,1:r12);

    J  = (c-U1s*U1s'*c)/k;

    U2s = [U1s J]*U12;
    V2s = [V1s zeros(size(V1s,1),1);zeros(1,size(V1s,2)) 1]*V12;
    S2s = S12;
    V2s2 = V2s(1:size(V2s,1)-1,:); % throw out the eigenvector
matching the lowest energy singular value


    U1s = U2s;
    V1s = V2s2;
    S1s = S2s;
end
```