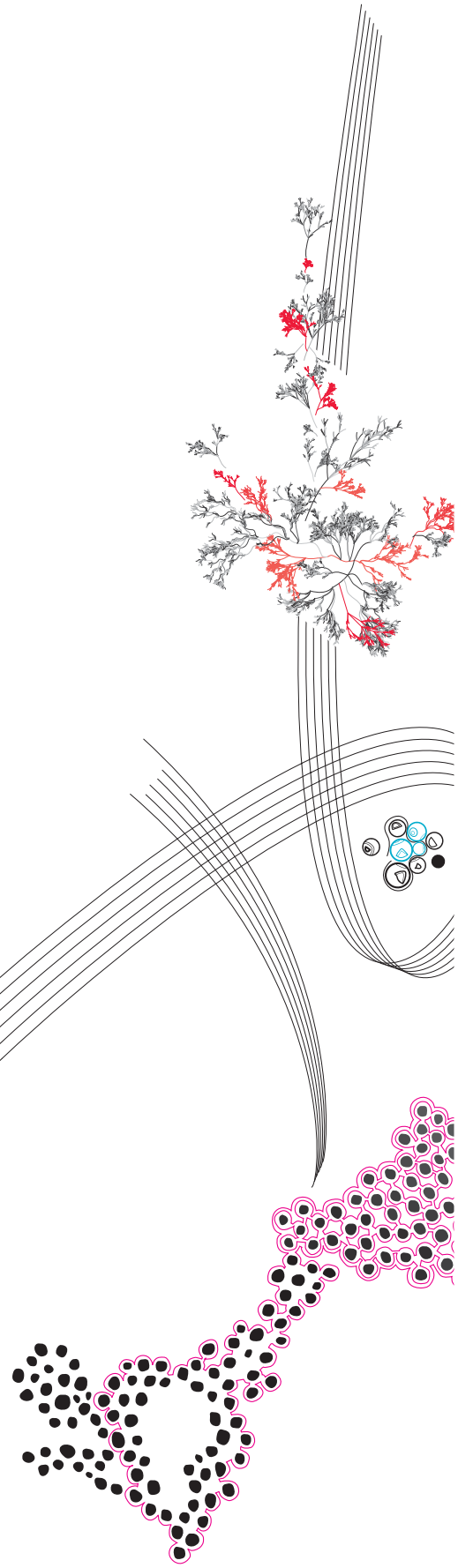


BSc Thesis Applied Mathematics
& Applied Physics

Model Reduction of Transport Phenomena with Kernel Principal Component Analysis

Erik Leering

Supervisor: S. Glas

July 19, 2022

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

I want to thank Silke Glas for her guidance and contributions to the research. Our conversations were long, interesting and fruitful and I am thankful for your help in the pursuit of an ambitious thesis. Furthermore I would like to thank Jeroen Verschuur for his flexibility and his help in the communication with Applied Physics and Bernard Geurts for joining my examination committee on such a short notice.

Model Reduction of Transport Phenomena with Kernel Principal Component Analysis

E. Leering*

July 19, 2022

Abstract

Plasma, to be understood as ionized gas, is part of many physical instruments and plays a major role in nuclear fusion [3]. Accurate simulations involving plasmas are computationally expensive. To perform analysis on the behaviour of plasma under varying parameters, Model Order Reduction (MOR) is desired. In this research, I aim to perform MOR with the use of kernel Principal Component Analysis (kPCA). To this end, I explain the workings behind kPCA and Proper Symplectic Decomposition (PSD), which is a necessary tool in the conservation of the symplectic structure generally found in plasma. Lastly, I apply a kPCA-derived Reduced Order Model (kPCA-ROM) on plasma with a linear Hamiltonian and a non-linear Hamiltonian and compare the results with a PSD-ROM.

Keywords: model,reduction,MOR,kPCA,PCA,kernel,Dynamical,System,Plasma,Vlasov, SDEIM,PSD,

1 Introduction

Plasmas occur in many applications and real-time simulations and predictions must be made to control plasma in certain scenarios. The applications vary from neon tubes and plasma displays to industrial applications such as amplifiers in telecommunication satellites and production of X-rays. The main motivational application for this research case is in controlled thermonuclear fusion [3], which only occurs on earth inside a hot plasma. The so-called magnetic confinement fusion method relies on confining the plasma with a magnetic field for a reasonable time. To that end, it is important that one can simulate and control the plasma. MOR helps in the computability of plasma simulations. The most popular Full Order Model (FOM) is the Particle In a Cell (PIC) method which simulates every particle. In a real plasma, such a simulation would take 10^{10} and more particles [3]. kPCA, which is an extension to the well-known method of Principal Component Analysis (PCA), is known to reduce the dimensions of non-linear data to only a fraction of the original dimensions [4]. With less components to simulate, the computational cost should decrease and viable real-time simulations can be performed. This thesis is outlined as follows: first, I explain the general workings of PCA. Then I extend the theory to kPCA. After that I describe the symplectic structure of certain Hamiltonian systems and how to perform PSD on linear as well as non-linear systems. then I attempt to find a unification of kPCA with PSD. I describe the Hamiltonian system of a simple plasma and use that and my previous findings to produce a ROM and compare the ROM produced using kPCA

*Email: e.leering@student.utwente.nl

with the ROM produced using PSD. Finally I reflect on my findings. In the appendix one can find two toy models that I used to learn these methods, one being a recreation of the paper written by García-González et al. [4], another being a simple pendulum example.

2 Principal Component Analysis (PCA)

The goal of PCA is to find a linear subspace of the data that explains as many of the features as possible. This is done by taking those dimensions that maximize the variance. Then one can possibly project the data unto this linear subspace.

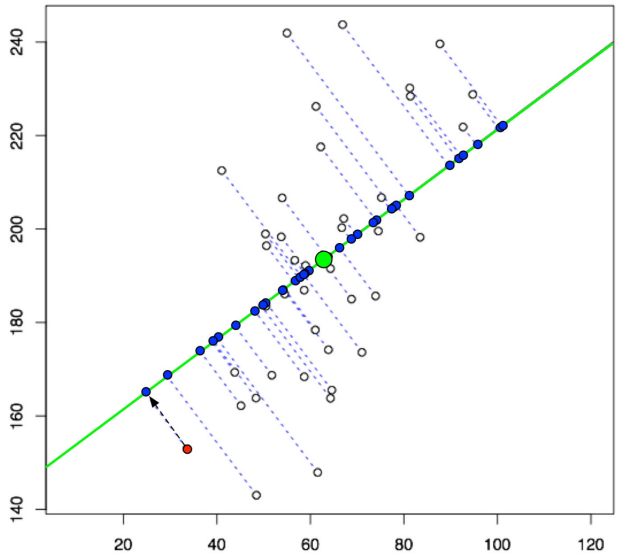


FIGURE 1: Example of a linear subspace found by PCA. [7]

Let the data considered be $X = \begin{bmatrix} x_1 & x_2 & \dots & x_{n_s} \end{bmatrix} \in \mathbb{R}^{d \times n_s}$, where $\{x_i\}$ represent n_s samples in $\mathbb{R}^d$, d being the dimension of one sample. As García-González et al. demonstrate perfectly in their work [4], there are two ways of performing PCA on the data.

2.1 Covariance Matrix method

The first method takes the covariance matrix $C \in \mathbb{R}^{d \times d}$ defined by $C = XX^T$. One then diagonalizes this matrix,

$$C = U' \Lambda U'^T, \tag{1}$$

where $\Lambda \in \mathbb{R}^{d \times d}$ is the diagonal matrix of eigenvalues and $U' \in \mathbb{R}^{d \times d}$ is the matrix of corresponding eigenvectors. The eigenvalues are sorted from large to small. MOR can be performed as follows: one takes the first k eigenvalues and the first k corresponding columns of eigenvectors of $U', U = U'[:, :k]$[1]. The columns of U' form an orthonormal basis in $\mathbb{R}^d$ [4], and the data can be projected to this basis as $Z = U^T X$, $Z \in \mathbb{R}^{k \times n_s}$. Equivalently, one element can be mapped as $z = U^T x$.

[1] Here I use Python or MATLAB notation for matrix elements

2.2 Gram Matrix method

Alternatively one can consider the $n_s \times n_s$ Gram matrix $G = X^T X$. Instead of straightforward diagonalization, one performs Singular Value Decomposition (SVD) on the data,

$$X = U' \Sigma V'^T, \tag{2}$$

with the same U' as before and V' being a $n_s \times n_s$ unit matrix. Σ is a $d \times n_s$ diagonal matrix of singular values of X. The singular values $\{\sigma_i\}$ relate to the eigenvalues $\{\lambda_i\}$ of C as $\sigma_i^2 = \lambda_i, i = 1, ..., d$. The SVD therefore also gives the diagonalization of C. More importantly, the SVD gives a diagonalization of the Gram matrix G [4]:

$$G = V' \tilde{\Lambda} V'^T. \tag{3}$$

Here $\tilde{\Lambda}$ is an $n_s \times n_s$ diagonal matrix of which the only d nonzero values are those of Λ. Once more the singular values can be sorted from large to small and dimensionality reduction can be achieved by taking only the first k singular values and the first k corresponding columns of matrix $V' : V = V'[:, :k]$. The data can now be projected as

$$Z = V^T G, \tag{4}$$

$$z = V^T g. \tag{5}$$

Here z represents one single element in reduced space, g is one column of G. Furthermore, the reduced data Z can be mapped back as

$$X \approx UZ. \tag{6}$$

The methods [2] can be summarized as follows:

Covariance method	Gram method
$C = XX^T$	$G = X^T X$
$C = U' \Lambda U'^T$	$X = U' \Sigma V'^T$
$U = U'[:, :k]$	$V = V'[:, :k]$
$Z = U^T X$	$Z = V^T G$

To introduce and map one new data sample x^* to reduced space using the Gram matrix method, one can construct a single column vector like those of G and map it the same way using V:

$$g^* = X^T x^*, \tag{7}$$

$$z^* = V^T g^*. \tag{8}$$

3 kernel-PCA

Although PCA described in the previous section is a powerful tool, it fails to find any dimensionality reduction when the data is non-linear. Therefore we extend the methods of PCA to kernel-PCA. The main difference between regular PCA and kPCA is that kPCA first tries to project the data to a high dimensional space, in the hopes that the nonlinear features of the data 'untangle'. Then, regular dimension reduction can be applied in order to reduce the data. This way, a significant order reduction is achieved even for nonlinear

[2] Interestingly, the covariance matrix method and the Gram matrix method for projecting the data are equivalent [4].

data. One other difference is that PCA works by either diagonalizing the covariance matrix C or the Gram matrix G, while kPCA requires the construction of G specifically.

The actual mapping of our data to a high dimensional space where it 'untangles' is usually unknown. Luckily, this mapping is not required directly. Since it is the Gram matrix G that is diagonalized to give us the main features (eq. 3), we need only construct G, without actually mapping the data. To this end, we implement a method of constructing G so that it has the same form as if it were constructed using data projected to a higher dimension. This method is called the kernel trick, and as it turns out, any bivariate symmetric form $\kappa(x, y)$ does just that, provided we take [4]

$$[G]_{ij} = \kappa(x^i, x^j). \tag{9}$$

This way of constructing G guarantees that G is symmetric and therefore constructing G remains computationally affordable. Furthermore this symmetric form ensures the diagonalization of G to yield only real eigenvalues and real corresponding eigenvectors. This is a useful property in symplectic model reduction, as I will illustrate later. It is considered good practise to center the Gram matrix G. This can be done per column g_j of G:

$$\tilde{g}_j = g_j - \frac{1}{n_s} G \mathbb{1}_{n_1} - \frac{1}{n_s} \mathbb{1}_{n_s \times n_s} g_j + (\frac{1}{n_s}^2 \mathbb{1}_{n_s}^T G \mathbb{1}_{n_s}) \mathbb{1}_{n_s}. \tag{10}$$

Matrix $\tilde{G}$ is then the centered version of matrix G. $\mathbb{1}_{n_s}$ represents an n_s column vector and $\mathbb{1}_{n_s \times n_s}$ represents an $n_s \times n_s$ matrix, both having all their entries equal to one. The data can now be mapped using equations 4 - 7 with the centered versions of G and g. Note that any newly introduced data point x^* still has to be transformed to g^* according to equation 7, and note also that g^* should be centered according to equation 10.

3.1 Backward-mapping

One complication in kPCA is the backward mapping. Where before we could take equation 6, the use of the kernel trick does not provide us with a matrix U that instantly returns x. One must construct a more elaborate backward-mapping of the reduced data $z^* \in \mathbb{R}^k$ to its pre-image $x^* \in \mathbb{R}^d$. To this end, I will consider the general functional

$$J(w) = \|z^* - z(w)\|^2. \tag{11}$$

The variable w in this functional is to be understood as a vector of weights. z^* is assumed to be the projection of some unknown sample x^* of the FOM. In order to find this x^*, we assume it to be a linear combination of samples that are readily available,

$$x^* \approx x(w) = \sum_i w_i x_i$$

with known projections, $z_i \leftarrow x_i$. To evaluate the functional (11), construct $g(w)$ and $z(w)$ using $x(w)$ in equation 7. If the Gram matrix G is centered, keep in mind that $g(w)$ should be centered according to equation 10 as well.

Since most of the original data samples are not close to the pre-image x^* that we try to approximate, they do not contribute much to the weights. In fact, having more points

makes it harder to find optimal weights in many cases[3]. In such cases, it is better to not consider these points in the first place. Most of the weights are therefore set to 0 and only those x_i from the original data set are considered that map to z_i in reduced space close to z^*.

Then, any optimization algorithm can be performed to find the optimal weights to minimize the functional from equation 11. One such algorithm is a stochastic gradient descent algorithm.

4 Symplectic model reduction for Hamiltonian systems

4.1 Hamiltonian systems and the canonical symplectic form

Consider a Hamiltonian system of N particles in one spatial dimension, neglecting particle interactions:

$$H(x,v) = \sum_{i=1}^{N} \frac{1}{2} v_i^2 - F(x_i), \tag{12}$$

$$\dot{x_i} = \frac{dH}{dv_i} = v_i, \qquad \dot{v_i} = -\frac{dH}{dx_i} = f(x_i). \tag{13}$$

Here $F(x)$ describes a potential and $f(x) = \frac{dF}{dx}(x)$ is a possibly non-linear function, $x, v \in \mathbb{R}^N, x = \begin{bmatrix} x_1 & ... & x_N \end{bmatrix}$ represent all particle positions, $v = \begin{bmatrix} v_1 & ... & v_N \end{bmatrix}$ represent all particle velocities. The time evolution of the FOM can be described using its state, $s \in \mathbb{R}^{2N} : s = \begin{bmatrix} x^T \\ v^T \end{bmatrix}$, and by rewriting the Hamiltonian system to

$$\dot{s} = \mathbb{J}_{2N} \nabla_s H(s). \tag{14}$$

Here, $\mathbb{J}_{2N} = \begin{bmatrix} 0 & \mathbb{I}_N \\ -\mathbb{I}_N & 0 \end{bmatrix} \in \mathbb{R}^{2N \times 2N}$ is the so-called canonical symplectic matrix and $\mathbb{I}_N \in \mathbb{R}^{N \times N}$ is the identity matrix. Matrix $\mathbb{J}_{2N}$ also serves to define symplectic matrices:

Definition 4.1 (Symplectic matrix)**.** Let $\mathbb{J}_{2N}$ denote the canonical symplectic matrix. A $2N \times 2k$ matrix A is called *symplectic* if

$$A^T \mathbb{J}_{2N} A = J_{2k}.$$

The flow $F_t : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ of a Hamiltonian system as described above preserves the total energy of the system and the canonical symplectic form [12], which in matrix notation reads

$$\left[DF_t\right]^T \mathbb{J}_{2N} \left[DF_t\right] = \mathbb{J}_{2N}. \tag{15}$$

Here, $\left[DF_t\right] \in \mathbb{R}^{2N \times 2N}$ describes the Jacobi matrix of the flow map F_t. According to definition 4.1, the Jacobi matrix of the flow of a Hamiltonian system as considered in this section is a symplectic matrix. The system (14) can be forwarded using simple simulation techniques, but the results improve when simulation techniques are used that preserve geometric properties such as the symplectic form [5].

[3]For example, let the data $X' \in \mathbb{R}^{2 \times N}$ describe N points on a circle centered around the origin, and assume all new points x^* to be considered are known to lie on this circle as well. Any non-zero weights assigned to points on the side opposite of x^* will pull the backward mapping into the interior of the circle, where we are certainly not expecting to find x^*. For inspiration, see the Appendix.

4.2 Proper Symplectic Decomposition (PSD)

In order to reduce the order of the model, one takes n_s samples of the (possibly simulated) system at different times. The data is represented as $S = \begin{bmatrix} s_1 & \dots & s_{n_s} \end{bmatrix} = \begin{bmatrix} X \\ V \end{bmatrix} \in \mathbb{R}^{2N \times n_s}$, where $X, V \in \mathbb{R}^{N \times n_s}$ represent all positions and all velocities of the samples, respectively. Tyranowski and Kraus [12] describe the method of PSD. One takes a matrix $A \in \mathbb{R}^{2N \times 2k}, k < N$, satisfying definition 4.1. The symplectic inverse $A^+ \in \mathbb{R}^{2k \times 2N}$ of A is defined as

$$A^+ = \mathbb{J}_{2k}^T A^T \mathbb{J}_{2N}, \tag{16}$$

which is an inverse in the sense that $A^+ A = \mathbb{I}_{2k}$ and $AA^+ = \mathbb{I}_{2N}$. A sample of the data $s \in S$ can be projected to $z = A^+ s, z \in \mathbb{R}^{2k}$, and can be projected back as $s = Az$. Tyranowski and Kraus [12] name two methods of constructing a symplectic matrix A using the available data: the Cotangent lift algorithm and the Complex SVD algorithm.

For the cotangent lift algorithm, the data is ordered as $\Delta = \begin{bmatrix} X_1 & \dots & X_{n_s} & V_1 & \dots & V_{n_s} \end{bmatrix} \in \mathbb{R}^{N \times 2n_s}$ and one performs SVD on Δ as in equation 2. Let $\Phi \in \mathbb{R}^{N \times k}$ be the matrix with the first K columns of $U' : \Phi = U'[:, :k]$. The symplectic matrix A can then be constructed as

$$A = \begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix}. \tag{17}$$

The Complex SVD algorithm is an extension on the Cotangent lift algorithm. First, SVD is performed in the same manner as in the Cotangent lift algorithm. Then one considers a possibly complex matrix U' resulting from the SVD and splits the real values from the imaginary values in the first K columns: $\Phi = \Re\mathrm{e}(U'[:, :k]), \Psi = \Im\mathrm{m}(U'[:, :k])$. Then

$$A = \begin{bmatrix} \Phi & -\Psi \\ \Psi & \Phi \end{bmatrix}. \tag{18}$$

Notice that this expression simplifies to equation 17 if U' only has real values. Furthermore, if one takes A from the Cotangent lift algorithm, its symplectic inverse is simply its transpose, $A^+ = A^T$ (see the appendix, A.1).

Replacing $s = Az$ in the FOM (14) and using $A^+ A = \mathbb{I}_{2k}$ gives the ROM

$$\dot{z} = A^+ \mathbb{J}_{2N} \nabla_s H(Az) = \mathbb{J}_{2k} \nabla_z H(Az), \tag{19}$$

which can be seen as a lower dimensional Hamiltonian system with the Hamiltonian $\tilde{H}(z) = H(Az)$ that preserves the symplectic form [12]. This can put in the form of a theorem:

Definition 4.2 (Symplectic ROM for Hamiltonian systems)**.** The symplectic ROM for Hamiltonian systems as described by equation 14 is given by equation $\dot{z} = \mathbb{J}_{2k} \nabla_z H(Az)$.

Corollary 4.0.1 (Symplectic MOR for linear Hamiltonian systems)**.** *In case the Hamiltonian system (12) is linear, that is, $f(x) = \alpha x$, it is possible to explicitly define the ROM Hamiltonian $\tilde{H}(z) = H(Az)$. In fact, $\tilde{H}(z) = H(z)$.*

Proof. First consider the FOM (14), which can be rewritten in matrix form to

$$\dot{s} = \mathbb{J}_{2N} \nabla_s H(s) = \begin{bmatrix} 0 & \mathbb{I}_N \\ -\mathbb{I}_N & 0 \end{bmatrix} \begin{bmatrix} -\alpha \mathbb{I}_N \\ \mathbb{I}_N \end{bmatrix} s = \begin{bmatrix} \mathbb{I}_{\mathbb{N}} \\ \alpha \mathbb{I}_N \end{bmatrix} s =: \begin{bmatrix} dH \end{bmatrix} s. \tag{20}$$

This defines an equivalent Hamiltonian matrix $\left[dH\right] = \begin{bmatrix} \mathbb{I}_{\mathbb{N}} \\ \alpha\mathbb{I}_N \end{bmatrix}$. Defining A using equation (17), for which $A^+ = A^T$ (A.1), it follows that the ROM in matrix form is

$$\dot{z} = A^+\mathbb{J}_{2N}\nabla_s H(Az) = A^T\left[dH\right]Az = \begin{bmatrix} \Phi^T & 0 \\ 0 & \Phi^T \end{bmatrix}\begin{bmatrix} \mathbb{I}_{\mathbb{N}} \\ \alpha\mathbb{I}_N \end{bmatrix}\begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix} z = \tag{21}$$

$$= \begin{bmatrix} \Phi^T & 0 \\ 0 & \alpha\Phi^T \end{bmatrix}\begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix} z = \begin{bmatrix} \Phi^T\Phi & 0 \\ 0 & \alpha\Phi^T\Phi \end{bmatrix} z = \begin{bmatrix} \mathbb{I}_k \\ \alpha\mathbb{I}_K \end{bmatrix} z = \mathbb{J}_{2k}\nabla_z H(z). \tag{22}$$

□

This implies that any FOM with $s = \begin{bmatrix} x \\ v \end{bmatrix}$ results in a ROM with $z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ where z_1 relates to z_2 in exactly the same way x relates to v. If the simulation technique does not dependent explicitly on the dimension of the state s that one inserts, then the exact same simulation can be used for the transformed state z.

5 Symplectic MOR for non-linear Hamiltonians using SDEIM

It is not difficult to see that Corollary 4.0.1 does not hold for non-linear Hamiltonians. Chaturantabut & Sorensen (2010) [2], Sorensen & Embree (2015) [11], Sargsyan, Brunton & Kutz (2015) [10] and Peng & Mohseni (2016) [9] address this issue extensively, all with an implementation of the Discrete Empirical Interpolation Method (DEIM). In this context it is appropriate to call the following method Symplectic DEIM (SDEIM), as Peng and Mohseni do. Consider once more the general FOM (14) and ROM (19), but now split them in a linear part L and a non-linear part $f_N(s)$:

$$\dot{s} = Lx + f_N(s), \tag{23}$$

$$\dot{z} = \tilde{L}z + A^T f_N(Az). \tag{24}$$

The idea is by splitting the system, we can use Corollary 4.0.1 on the linear part. The non-linear part will be approximated using other methods. Notice how the expression now states that we must map our low-dimensional state z to the higher dimensional s before we can evaluate the non-linear function $f_N : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$. Then we map the result back to a lower dimension. This is not saving any computation time.

Instead, we will commit ourselves to another PSD: let $S_N \in \mathbb{R}^{2N\times n_s}$ be the matrix representing the non-linear function $f_N(s)$, evaluated at all samples in matrix S: $S_N = \left[f_N(s_1), ..., f_N(s_{n_s})\right]$. We assume $f_N(s)$ to lie approximately on the range of a symplectic matrix $A_N \in \mathbb{R}^{2N\times 2k}$, which is constructed using the methods of section 4, only now we use S_N as our data instead of S. We aim to evaluate $f_N(Az)$ only at $2k$ spatial indices. Therefore, we project Az on a $2k$-dimensional subset of Az, which I will call $\tilde{s} \subset Az$. Then we can evaluate $f_N(\tilde{s})$ in the hopes that this approximates $f_N(s)$ well.

If $f_N(s)$ indeed does lie approximately on the range of A_N, we can find a coefficient vector $\tau(s) \in \mathbb{R}^{2k}$ such that $f_N(s) \approx A_N\tau(s)$, and we can design a projection matrix $P \in \mathbb{R}^{2N\times 2k}$ such that $P^T f_N(s) = P^T A_N\tau(s)$ holds exactly for all n_s elements in S_N. In other words, we would like to have a projection matrix P for which we can find a corresponding coefficient matrix $\Gamma = \left[\tau(s_1), ..., \tau(s_T)\right]$ such that $P^T S_N = P^T A_N\Gamma$.

The means of obtaining P are perfectly illustrated by Sorensen & Embree [11]. The DEIM processes the columns of A_N one at a time. The projection matrix P is found by a greedy algorithm. Each step aims to find a new pivot-point that is to be added to matrix P. The idea is that the pivot with the largest error in our projection compared to the real value is where we will place the next pivot in P. For this algorithm, let $p_j = \begin{bmatrix} p_1 & ... & p_j \end{bmatrix}^T$ be a vector of indices, and $P_j^{(2N\times j)} = \mathbb{I}_{2N\times 2N}(:, p_j)$ be the identity matrix with the columns shuffled to correspond with the indices of p_j. This way, P_j is a projection. Finally, let $A_{Nj} = \begin{bmatrix} a_1, ..., a_j \end{bmatrix}$ be the first j columns of A_N. The algorithm for obtaining the projection matrix P is then described by Algorithm 1.

Algorithm 1 SDEIM for obtaining projection matrix P

Require: Symplectic matrix $A_N \in \mathbb{R}^{2N\times 2k}$
Ensure: Projection matrix $P \in \mathbb{R}^{2N\times 2k}$
  $\mathbf{a}$=$A_N(:,1)$
  $p_1 = \arg\max(|\mathbf{a}|)$
  $\mathbf{p} = [p_1]$
  for $j = 2, 3, ..., 2k$ **do**
    $\mathbf{a} = A_N(:, j)$
    $c = A_N(\mathbf{p}, 1 : j-1)^{-1}\mathbf{a}(\mathbf{p})$
    $\mathbf{r} = \mathbf{a} - A_N(:, 1 : j-1)\mathbf{c}$
    $p_j = \arg\max(|\mathbf{r}|)$
    $\mathbf{p} = [\mathbf{p}; p_j]$
  end for
  $P = \mathbb{I}(:, \mathbf{p})$

Now that we have obtained a projection matrix P, we want to use it to evaluate f_N. To do so, we construct a linear projection term $P_A = P^T A_N$ and a non-linear projection term $P_{NL} = A_N^T P$. The linear projection term maps z to $\tilde{x}$ so that f_N can be evaluated at 2k spatial indices and the non-linear projection term projects the result to z again. The ROM becomes:

$$\dot{z} = \tilde{L}z + P_{NL} f_N(P_A z) \tag{25}$$

Note that if one changes the problem so that the function f_N is linear, the simple replacement of P by $I_{2N\times 2N}$ makes this expression equivalent to the last equation Corollary 4.0.1.

6 Symplectic kPCA

In order to construct a ROM that preserves the Hamiltonian and the symplectic form, we apply the methods of section 4. Only, matrix A is constructed using kPCA as in section 3.

Let $X, V \in \mathbb{R}^{N\times n_s}$ be measurements of the positions and velocities of N particles at n_s distinct times. Order the data as $S = \begin{bmatrix} X \\ V \end{bmatrix} = \begin{bmatrix} s_1 & ... & s_{n_s} \end{bmatrix} \in \mathbb{R}^{2N\times n_s}$ and $\Delta = \begin{bmatrix} X & V \end{bmatrix}$. $s_i \in \mathbb{R}^{2N}$ is the i'th column of S and represents the state of the system at one time. The $n_s \times n_s$ Gram matrix G is constructed using the kernel trick, $[G]_{ij} = \kappa(s_i, s_j)$. The kernel trick once more ensures us that the Gram matrix is symmetric and therefore, the eigenvalues and eigenvectors from the diagonalization $G = V'\Lambda V'^T$ are real. Let $\Phi \in \mathbb{R}^{n_s\times k}$ take

the first k columns of $V' : \Phi = V'[:, :k]$. Since Φ is real, we can use the Cotangent lift method to construct $A \in \mathbb{R}^{2n_s \times 2k}$ using equation 17.

Here we encounter a clash between model reduction using kPCA and symplectic model reduction as described in section 4. PSD requires $z = A^+ s$, which is key to retaining the symplectic flow. This will not work as the dimensions do not match. Furthermore, any projection using S and not G results in a linear projection: there would be little point in applying kPCA if we use it to find a projection of the data onto a linear subspace of the data. kPCA requires $Z = \Phi^T G$. This offers us a possible mapping to a lower dimension, but takes away all means of simulating the system as the symplectic form is not conserved and none of the MOR methods developed in sections 4 and 5 apply. For now, it does seem we have arrived at a dead-end. This research unfortunately offers no practical solution to this problem. However, theoretical means of adjusting $Z = \Phi^T G$ to preserve the symplectic form are discussed in section 9.

6.1 Linear symplectic projections using kPCA

Let $\Delta = \begin{bmatrix} X & V \end{bmatrix} = \begin{bmatrix} \xi_1 \\ ... \\ \xi_N \end{bmatrix} \in \mathbb{R}^{N \times 2n_s}$. ξ_i can be regarded as a single trajectory. Since particle interactions are neglected, it is possible to consider the particle trajectories as the samples (as opposed to taking the states). In terms of section 2, this means changing the dimension $d \to n_s$ and taking N samples of independent trajectories instead of n_s samples of the state of the system. Applying the techniques of section 3 now yields an $N \times N$ Gram matrix constructed using $[G]_{ij} = \kappa(\xi_i, \xi_j)$.

Suppose the SVD $G = V' \Lambda V'^T$ is available. As discussed in section 3, the use of the kernel trick ensures us that G has real eigenvalues and eigenvectors. Take the first k columns of $V \in \mathbb{R}^{N \times N}$, $\Phi = V[:, :k] \in \mathbb{R}^{N \times k}$. The symplectic matrix $A \in \mathbb{R}^{2N \times 2k}$ is now constructed using equation 17 and the data is projected to $z = A^T s$. The ROM follows from sections 4 and 5.

7 Plasma model

Plasma, previously described as ionized gas, is modelled in many different ways [3]. One of the simpler models is derived from the Vlasov equation:

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} - E(x)\frac{\partial f}{\partial v} = 0. \tag{26}$$

Where $f(t, x, v)$ is the particle density function in a plasma and $E(x) = -\frac{\partial \phi(x)}{\partial x}$ is an external electric field of the potential $\phi(x)$. As described by Tyranowski & Kraus [12], the Particle In a Cell (PIC) method for the Vlasov equation has the form of a Hamiltonian system of equations. PIC uses the Ansatz $f(t, x, v) = \sum_{i=1}^{N} w_i \delta(x - x_i(t))\delta(v - v_i(t))$, where x_i, v_i, w_i represent one particle's position, velocity and weight, respectively. Inserting this in the Vlasov equation (26), one obtains a system of ordinary differential equations:

$$\dot{x_i} = v_i, \tag{27}$$

$$\dot{v_i} = \frac{\partial \phi}{\partial x}(x_i), \tag{28}$$

From here we can describe the system as a Hamiltonian system as in 12:

$$H(x,v) = \sum_{i=1}^{n}[\frac{1}{2}v_i^2 - \phi(x_i)], \tag{29}$$

$$\dot{x_i} = \frac{\partial H}{\partial v_i} = v_i, \tag{30}$$

$$\dot{v_i} = -\frac{\partial H}{\partial x_i} = -E(x_i). \tag{31}$$

Here $x = (x_1, ..., x_N)$ and $v = (v_1, ..., v_N)$ are all of the particle positions and velocities at one time.

8 Numerical experiments

Having developed strategies of applying kPCA on dynamical models (albeit linearly), it is time to use our tools on plasma simulations. I refer to Section 7 for the PIC modelling of plasma. In the coming models, I consider 1000 particles which are simulated for a total of 20 seconds with an interval of $dt = 0.01$ for a total of 2000 time steps. Since simulation techniques that conserve the symplectic form are preferred over standard simulation techniques [5], I make use of the Störmer-Verlet method:

$$v_{n+\frac{1}{2}} = v_n + \frac{h}{2}\nabla_x H(x_n, v_{n+\frac{1}{2}}), \tag{32}$$

$$x_{n+1} = x_n + \frac{h}{2}(\nabla_v H(x_n, v_{n+\frac{1}{2}}) + \nabla_v H(x_{n+1}, v_{n+\frac{1}{2}})), \tag{33}$$

$$v_{n+1} = v_{n+\frac{1}{2}} + \frac{h}{2}\nabla_x H(x_{n+1}, v_{n+\frac{1}{2}}). \tag{34}$$

This method will be used for the simulation of every model. Another famous example is the implicit-midpoint method and there are many more [13].

It is possible to vary the character of plasma simulations by varying the electric field, as has been demonstrated in section 7. In what follows, I will compare linear kPCA model reduction techniques and PCA model reduction techniques w.r.t. the FOM simulations of plasma. The kernel applied in kPCA for the construction of G is the Gaussian kernel,

$$\kappa(\xi_i, \xi_j) = exp(\frac{-||\xi_i - \xi_j||^2}{||\xi_0||^2}). \tag{35}$$

This kernel, also known as the Radial Basis Function, computes the similarity of two points, here the similarity between two trajectories. This kernel has been chosen because it is relatively straightforward and works well in many kPCA applications [4]. For the backward-mapping, I use gradient-descend, with a one-size-fits all adjustment described in Appendix E, from here to be referred to as optimization back-mapping. I investigate the plasma under two conditions, depending on the electric field. The first situation gives a linear Hamiltonian, while the latter gives a non-linear Hamiltonian.

8.1 Linear-Hamiltonian plasma simulations

The first electrical field that we consider is $E(x) = \beta^2 x$, which coincides with the field considered by Tyranowski and Kraus [12]. As it turns out, there exists an analytical

solution for the time-evolution of plasma under such a field:

$$X_i(t) = \frac{1}{\beta} V_i sin(\beta t) + X_i(0) cos(\beta t), \tag{36}$$

$$V_i(t) = V_i(0) cos(\beta t) - \beta X_i sin(\beta t). \tag{37}$$

Furthermore, we also take the same initial distribution for the state of our particles as Tyranowski and Kraus:

$$f(0, x, v) = \frac{1}{\eta\sqrt{2\pi}} e^{-\frac{1}{2\eta^2}x^2} \left(\frac{1}{1+a}\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}v^2} + \frac{a}{1+a}\frac{1}{\eta\sqrt{2\pi}} e^{-\frac{1}{\sigma^2}(v-v_0)^2}\right),$$

$$\eta = 10, \qquad a = 0.3, \qquad v_0 = 4, \qquad \sigma = 0.5.$$

To be precise, we perform rejection sampling in 2D for our initial particle conditions on this function. Using $x, v \in [-0.8, 0.8]$, we only accept randomly selected values for x, v that are plausible under $f(0, x, v)$.

The goal is to perform model reduction that is precise enough to predict the behaviour of the plasma, even for arbitrary β. Let us first examine the behaviour of plasma under this electric field:

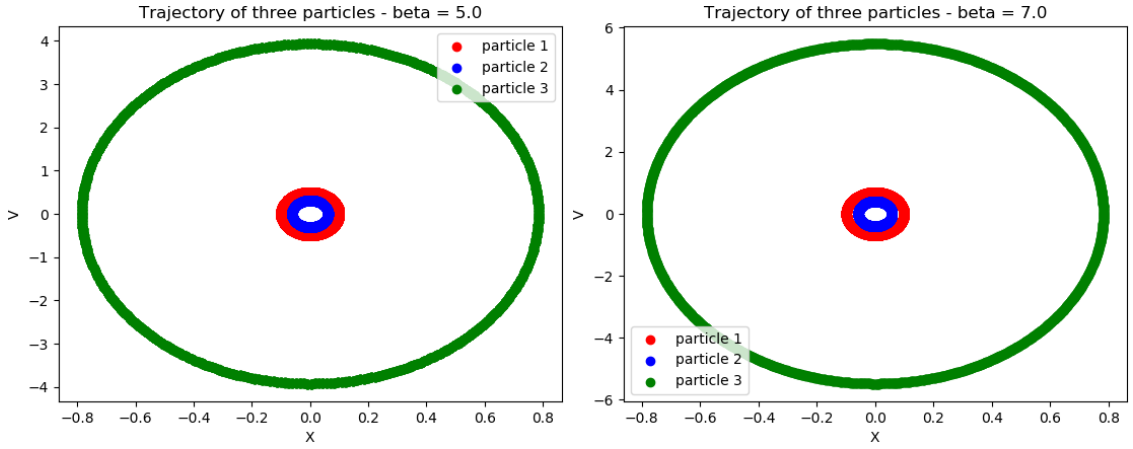


FIGURE 2: Trajectory of several particles for 2 values of β

As we can see, the trajectory of our particles is elliptic. The initial distribution is the same in both of the above simulations. Furthermore, the same particles are tracked. It can be seen that β mostly changes the maximum velocity reached. We will also have a glimpse at the final state of our simulation and exact solution. Since an analytical solution is available, it is worthwhile to investigate how well our FOM simulation performs against this analytical solution, for every β we plan to use in the construction of our ROM. This is presented in the following graphs:

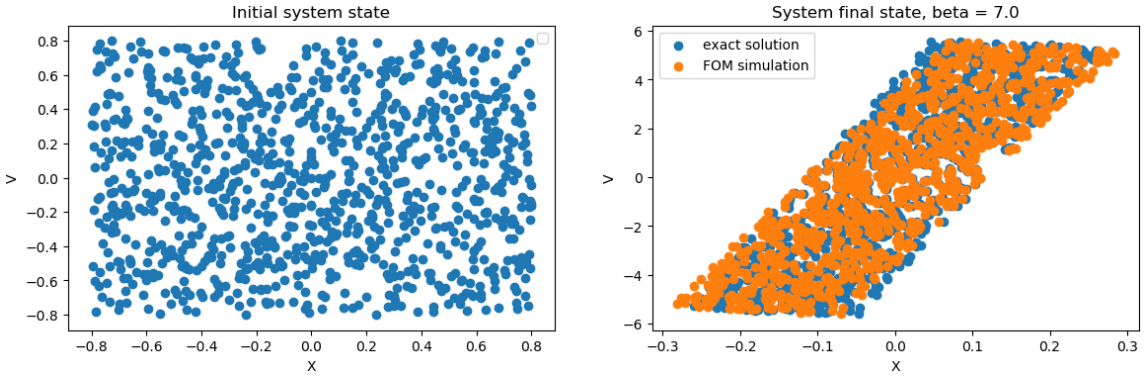


FIGURE 3: Phase space from t=0 to t=2000.

The electric field rotates the particles and stretches the ensemble in the V-axis of phase space. The exact solution and the Störmer-Verlet simulation ensembles overlap almost entirely, but the individual particles differ by non-negligible amounts. This may be a sign of the time steps $dt = 0.01$ being somewhat large. For the other value of β, the result is slightly better. The errors are examined as follows:

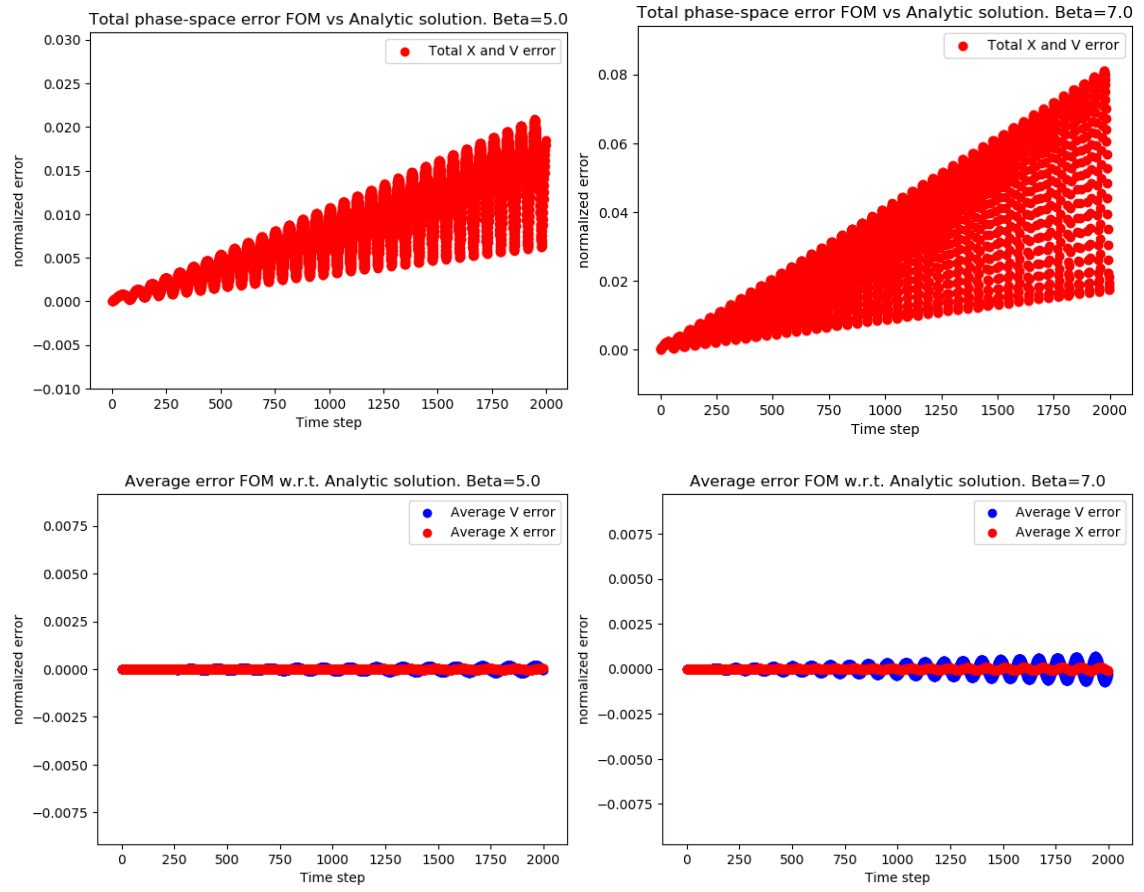


FIGURE 4: Error of all particles for 2 values of β, FOM vs exact solution.

The total error of the phase space at any time (up) is determined as

$\frac{1}{N}\sum_{i=1}^{N}\sqrt{\Delta x_i^2+\Delta v_i^2}$ and the average X error is found as $\frac{1}{N}\sum_{i=1}^{N}\Delta x_i$, the average V error idem (down). The error increases linearly as we progress further in the simulation, as expected [5]. Most notably, the error is due to the velocity of the particles, which is the direction in which we stretch the ensemble.

The Hamiltonian for this electric field is linear: inserting this electric field in the equations of section 7 leads us to $\dot{x_i} = v_i, \dot{v_i} = E(x_i) = \beta^2 x_i$. This implies we can use Corollary 4.0.1 to construct a ROM. We will base our ROM on the FOM simulation of the plasma, not the exact solution. Furthermore, we take FOM simulations using 2 different β's: $\beta = \begin{bmatrix}5, 7\end{bmatrix}$ and structure our data as $S = \begin{bmatrix}X_{\beta_1}, X_{\beta_2} & V_{\beta_1}, V_{\beta_2}\end{bmatrix}$. It is possible to derive a ROM with only one β, but the results are severely worse. Naturally more β's improve the performance.

To analyze how well I can reduce the dimensions of this model, I plot the decay of the singular values:

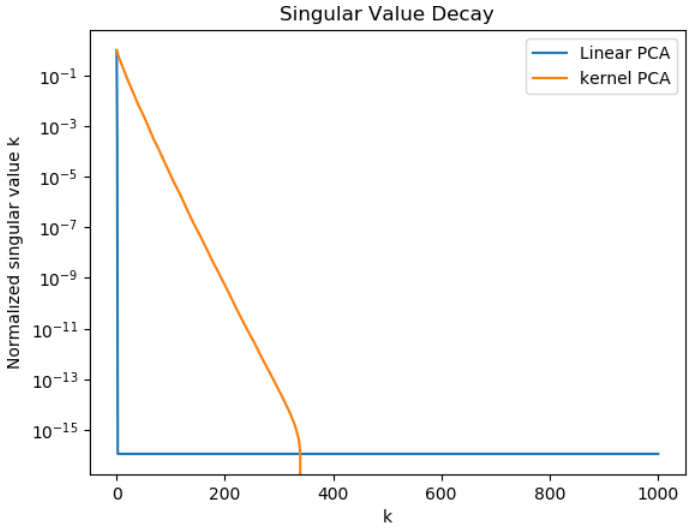
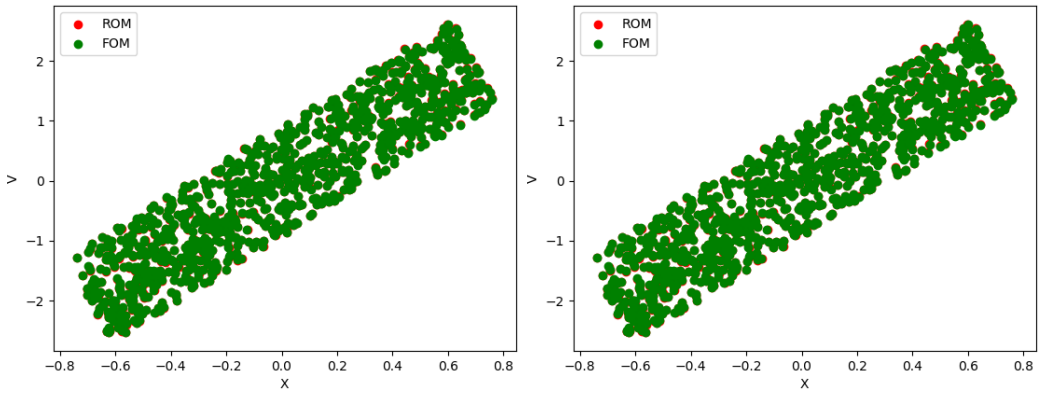


FIGURE 5: Decay of Singular Values.

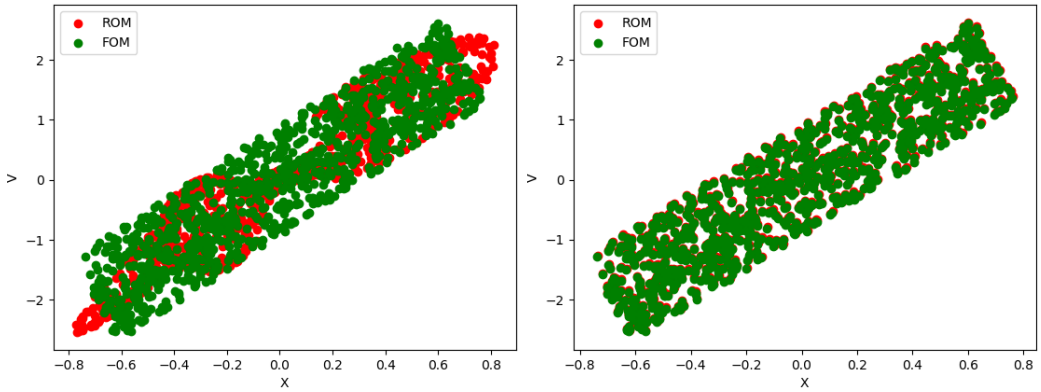
This graph shows a significantly faster drop in singular values for regular PCA than for kernel-PCA. It would seem there is no benefit in using kPCA for model reduction to this problem. Still, I will construct a ROM using kPCA (a kPCA-ROM) for a linear projection as in section 6.1 which only takes $k = 10$ features and compare it with the FOM. Furthermore, I will do the same for a ROM derived using regular PCA (a PCA-ROM) which takes only two features. The choice of only including 2 features in the PCA-ROM is justified by the instant decay of singular values using regular PCA.

The ROM's are simulated in time for every β. The results are compared at 100 time steps with the FOM. Since these ROM's are produced as linear projections of the data, it is also possible to map them back using the symplectic matrix A according to section 4. I compare both methods:



(A) kPCA-ROM with optimization back-mapping.

(B) PCA-ROM with optimization back-mapping.



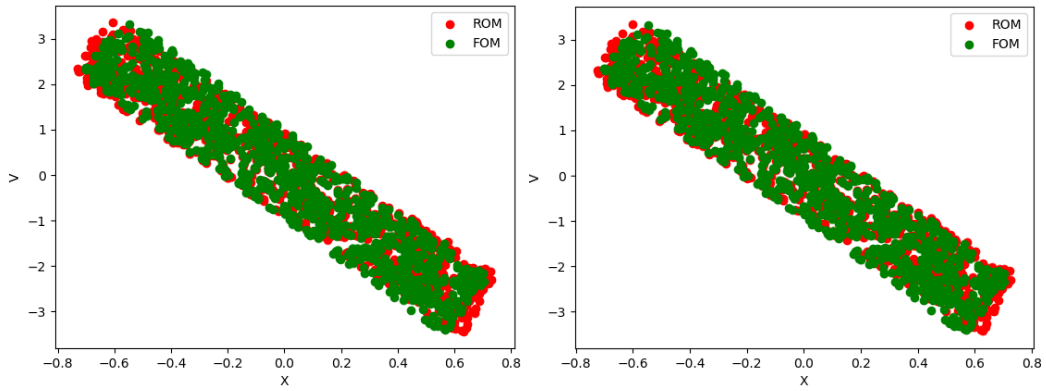
(C) kPCA-ROM with linear back-mapping.

(D) PCA-ROM with linear back-mapping.

FIGURE 6: Comparison of reduced models for $\beta = 0.5$.

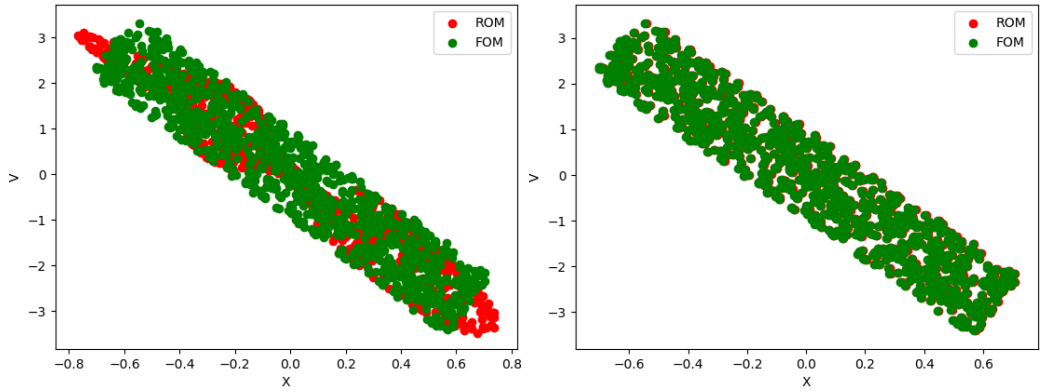
It seems as if both our ROM do a fine job in mimicking the FOM, provided we use the optimization method for backward-mapping. Similar results are obtained for $\beta = 7.0$ and the corresponding graphs can be found in the appendix B.1. If we consider the linear backward-mapping however, the kPCA-ROM seems to perform much worse. This is to be expected: usual means of PCA model reductions capture all the features in only 2 singular values, while the kPCA approach in this setting needs about 350 (see figure 14). The errors between the ROM's and the FOM can be found in the appendix.

Of interest in this research is the performance of the ROM's w.r.t. novel values for the parameters. The ROM performances for novel $\beta = 0.6$ is as follows:



(A) kPCA-ROM with optimization back-mapping.

(B) PCA-ROM with optimization back-mapping.



(C) kPCA-ROM with linear back-mapping.

(D) PCA-ROM with linear back-mapping.

FIGURE 7: Comparison of reduced models for $\beta = 0.6$.

Here we can draw two conclusions: PCA outperforms kPCA and the linear backward-mapping approach outperforms the optimization method for backward-mapping. Even new values for β can be mimicked perfectly with only $k = 2$ features using symplectic PCA.

8.2 Non-linear-Hamiltonian plasma simulations

Alternatively, we consider the non-linear case described by Hesthaven, Pagliantini and Ripamontiand [6]. They make use of an electric field $E(x) = -x^3$ and a modified expression for the Vlasov equation that includes a spatial scaling parameter ν, $\frac{\partial f}{\partial t} + \frac{1}{\nu} v \frac{\partial f}{\partial x} - E(x)\frac{\partial f}{\partial v} = 0$. This changes the Hamiltonian slightly, with a resulting set of differential equations:

$$\dot{x} = \frac{1}{4\nu} v,$$
$$\dot{v} = -x^3.$$

Otherwise the system remains unchanged. No exact solution exists,[4] so I will only compare the ROM's with the FOM's. The initial distribution is altered as well:

$$f(0, x, v) = \left(\frac{1}{\sqrt{2\pi}\alpha} e^{-0.5 v^2 \alpha^{-2}}\right)\left(1 + \beta cos\left(4\pi \frac{x + 0.8}{1.6}\right)\right),$$

$$\alpha = 0.07 \qquad \beta = 0.02.$$

[4] or is known to me, at least,

Once more I perform rejection sampling in 2D in the same manner as in the previous section. In this case one can vary the parameter ν instead of β. The particle trajectories are as follows:

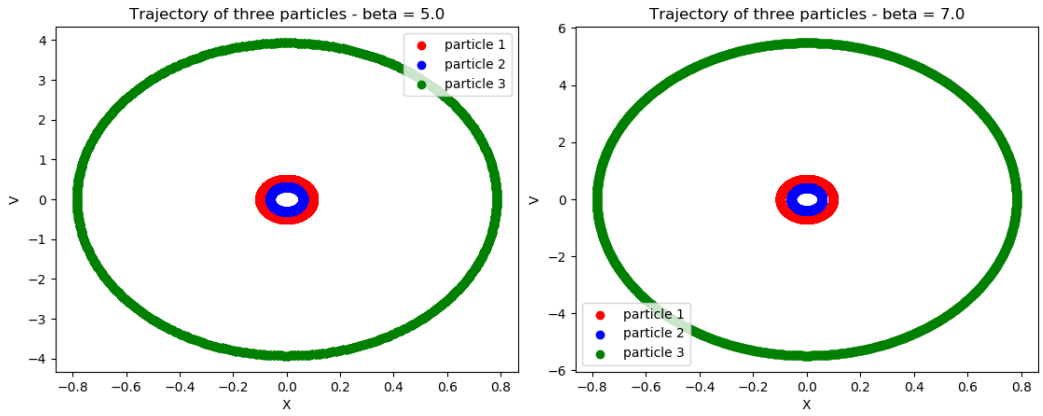


FIGURE 8: Trajectory of several particles for 2 values of ν.

As a first attempt at model reduction, the ROM's are constructed in the exact same manner as in the previous section, giving a kPCA-ROM and a PCA-ROM. Here I assume Corollary 4.0.1 is approximately true. This is a grave assumption, as I will soon demonstrate. To analyze how well I can reduce the dimensions of this model, I plot the decay of the singular values:

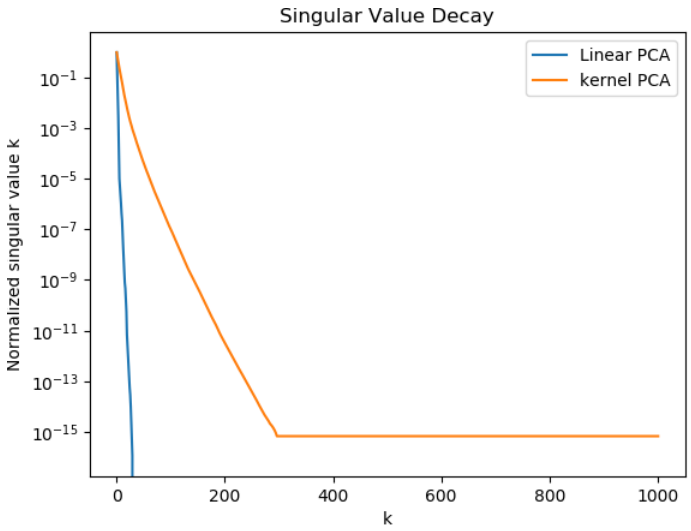
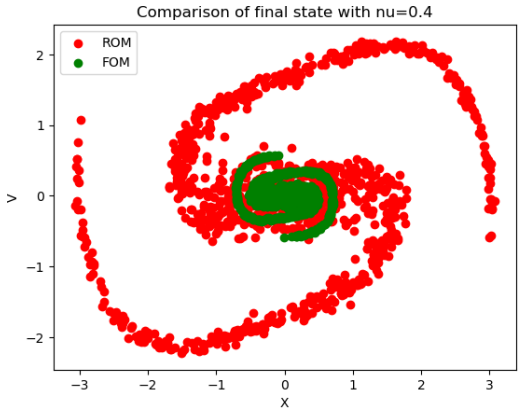
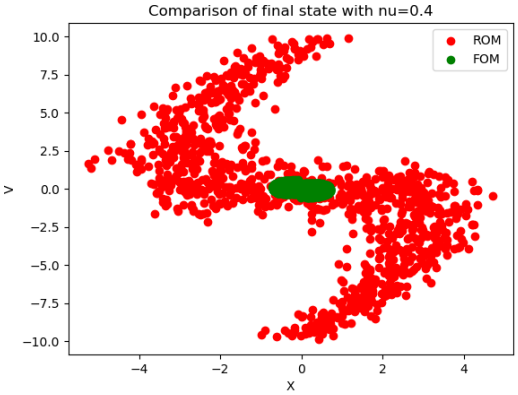


FIGURE 9: Decay of Singular Values.

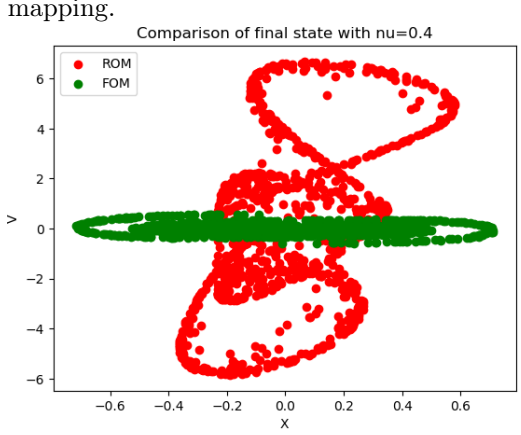
This graph is similar to the figure 14. Once more PCA seems to do better than kPCA by no small margin. Even though the PCA singular values decay slower than before, Figure 9 is sufficient justification for me to use once more the first $k = 2$ features for the PCA-ROM and the first $k = 10$ features for the kPCA-ROM. The ROM's are simulated in time for every ν. The results are compared at 100 time steps with the FOM.



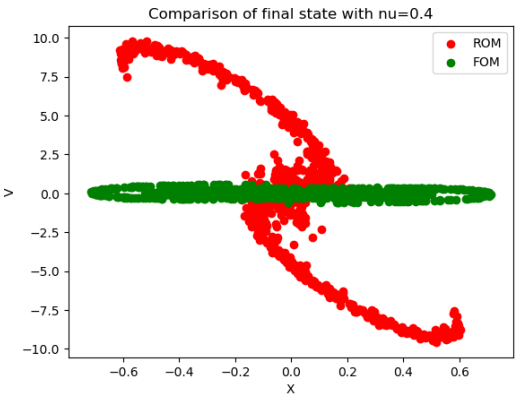
(A) kPCA-ROM with optimization back-mapping.



(B) PCA-ROM with optimization back-mapping.



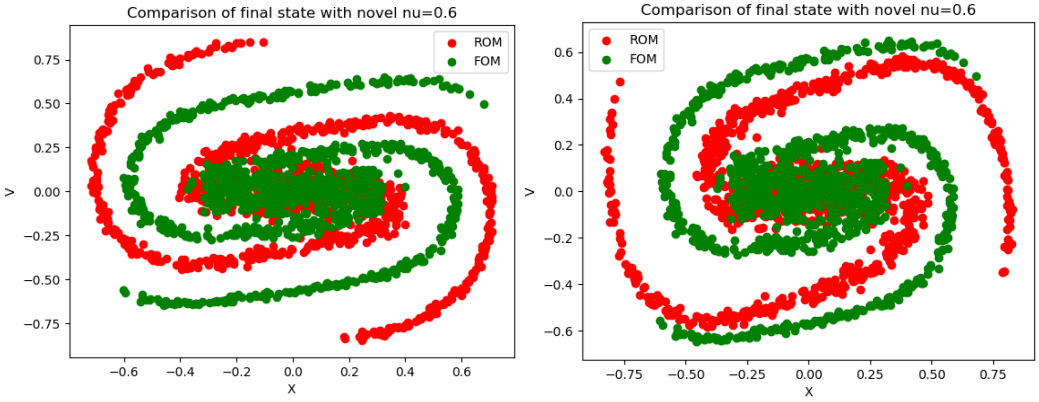
(C) kPCA-ROM with linear back-mapping.



(D) PCA-ROM with linear back-mapping.

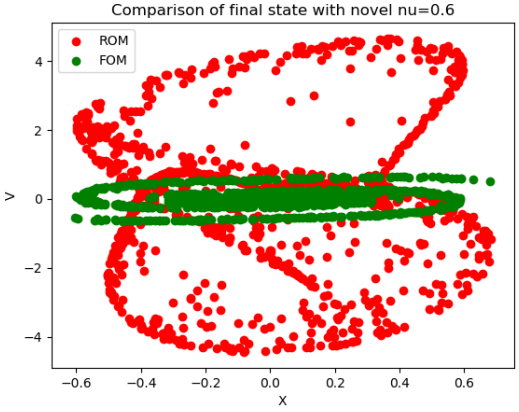
FIGURE 10: Comparison of reduced models for $\nu = 0.4$.

The above are all unimpressive imitations of the FOM. The kPCA-ROM with optimization back-mapping does seem to get the shape right and all ROM's have some amount of rotation. All ROM's are however insufficiently accurate to represent the FOM. The same holds for $\nu = 0.8$, see Appendix B.2. Nevertheless the ROM performances for novel $\nu = 0.6$ are still presented:

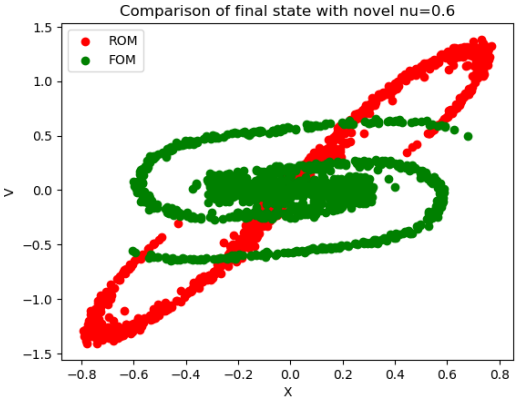


(A) kPCA-ROM with optimization back-mapping.

(B) PCA-ROM with optimization back-mapping.



(C) kPCA-ROM with linear back-mapping.



(D) PCA-ROM with linear back-mapping.

FIGURE 11: Comparison of reduced models for $\nu = 0.6$.

Interestingly (and most unexpectedly) the linear approximations do resemble the FOM if we apply optimization back-mapping. Both ROM's however generally perform awful and no conclusions can be made about how kPCA compares to PCA apart from the significant difference in the decrease of singular values. Graphs of the errors can be found in the Appendix.

8.2.1 SDEIM MOR on non-linear plasma simulations

To improve the ROM's, the Symplectic Discrete Empirical Interpolation Method as described in section 5 is applied. The ROM's are now described by equation 25 and are simulated using the Störmer-Verlet. Note that the non-linear part requires A_N to be obtained using regular PCA. For this simulation, the PCA-ROM, the kPCA-ROM and the projection of the non-linear S_N all take $k = 10$ features. The trajectories remain and the final state remain unchanged w.r.t. the previous section. The singular value decomposition is as follows:

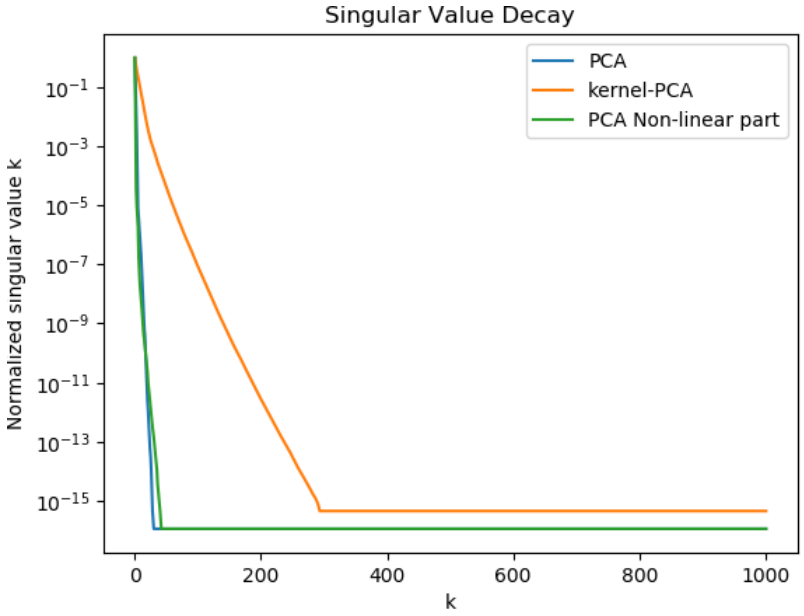
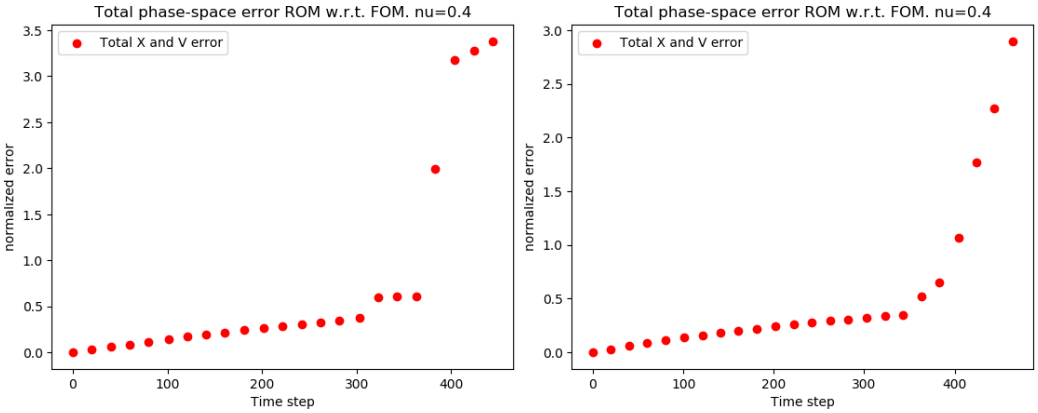


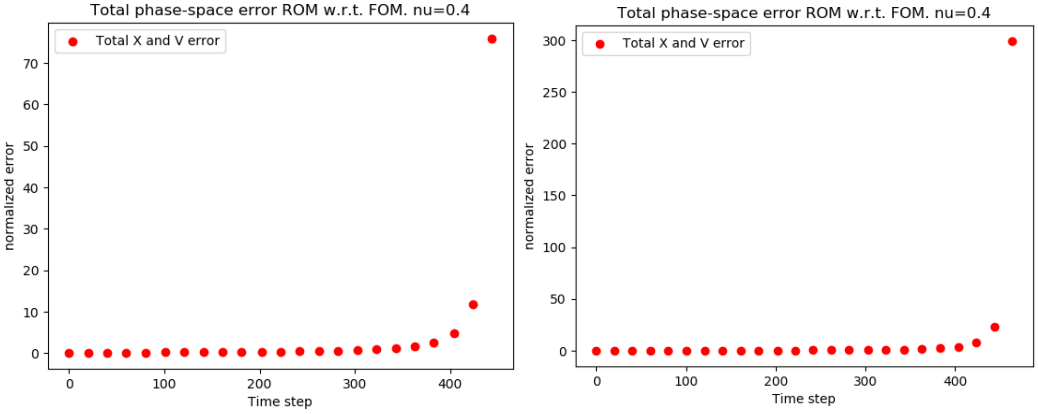
FIGURE 12: Decay of Singular Values.

Once again PCA seems to outperform kPCA in the decay of singular values. When the ROM's are simulated however, the final state cannot be plotted. To investigate why, consider the errors, which propagate as follows:



(A) kPCA-ROM error with optimization back-mapping.

(B) PCA-ROM error with optimization back-mapping.



(C) kPCA-ROM error with linear back-mapping.

(D) PCA-ROM error with linear back-mapping.

FIGURE 13: Comparison of reduced models for $\nu = 0.4$.

As we can see, the errors grow exponentially for every ROM and just after 400 time steps the ROM's completely seize to function. I will elaborate on why in the discussion. For now it suffices to say that the SDEIM extension could not save the ROM's for non-linear Hamiltonian plasma simulations.

9 Discussion

9.1 Combining kPCA with symplectic MOR

Unfortunately no combination of kPCA with symplectic model reduction was found in this research. In general, kPCA model reduction does not conserve the symplectic form of the system. This does not mean that the two methods are completely incompatible. In fact, it would be a most interesting topic to further explore how kPCA might be combined with symplectic model reduction. In this work I consider the trajectories to find a reduced model instead of the states. When the Gaussian kernel is applied to a single state (and normalized to a single state, not a trajectory), it projects the data as follows:

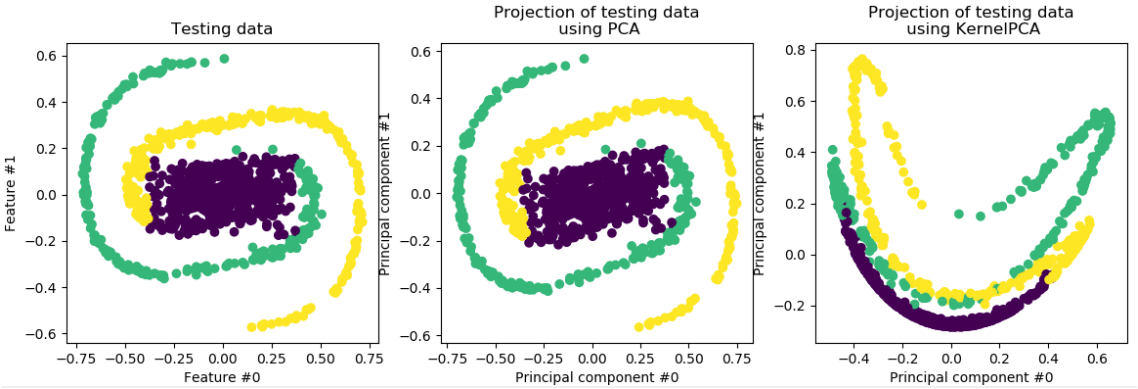


FIGURE 14: kPCA and PCA data projection of a single state.

As we can see, the data does seem to untangle somewhat in the first two principal components. Naturally this is not sufficient to perform proper model reduction, but it shows promise. It might be worthwhile to search for a kPCA technique on the states instead of on the trajectories and construct a non-linear yet symplectic reduced model out of that. A start to that search might look as follows. As shown in the paper of Buchfink, Glas and Haasdonk [1], once can generalize the mapping of the physical data in terms of an 'encoder' $e(s) : \mathbb{R}^{2N} \to \mathbb{R}^{2k}$ and a 'decoder' $d(z) : \mathbb{R}^{2k} \to \mathbb{R}^{2N}$. Symplectic model reduction can be performed as long as the encoder is a symplectic map:

$$\left[D_s e(s)\right]^T \mathbb{J}_{2k} \left[D_s e(s)\right] = \mathbb{J}_{2N}. \tag{38}$$

Here $\left[D_s e(s)\right]$ represents the Jacobi matrix of the encoder. Let $X, V \in \mathbb{R}^{N \times n_s}$ be positions and velocities as before, and let

$$S = \left[XV\right] \in \mathbb{R}^{N \times 2n_s}, \qquad G : [G]_{ij} = \kappa(s_i, s_j), \tag{39}$$

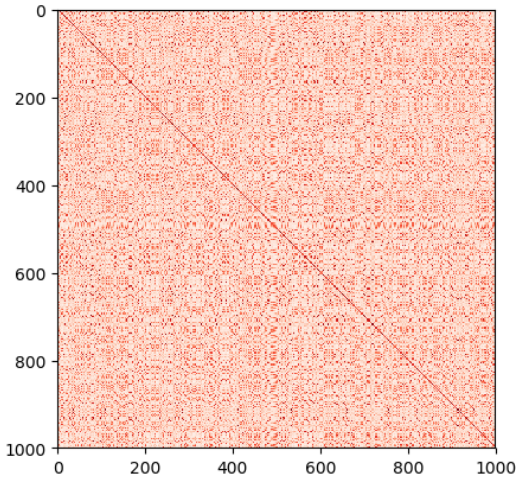
$$G_x : [G_x]_{ij} = \kappa(x_i, x_j), \qquad G_v : [G_v]_{ij} = \kappa(v_i, v_j). \tag{40}$$

One suggestion I would bring forward is to pursue a more general way of mapping the data to reduced space. In section 4 I consider the cotangent lift algorithm and the complex SVD algorithm as a means of constructing a symplectic matrix A. It might be worth having a look at the complex SVD method, under the right circumstances:

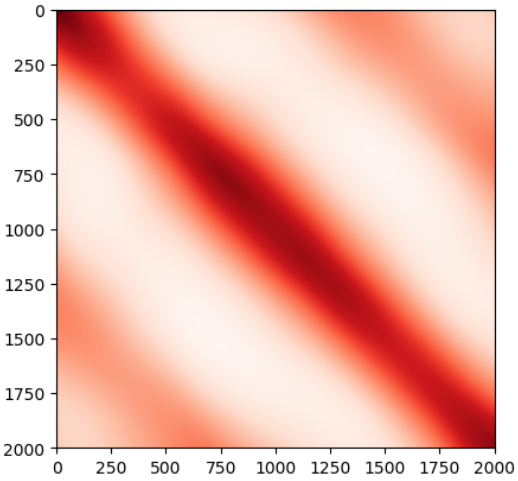
$$Z = A^+ \tilde{G}, \qquad A = \begin{bmatrix} \Phi & -\Psi \\ \Psi & \Phi \end{bmatrix}, \tag{41}$$

$$\Phi^T \Psi = \Psi^T \Phi, \qquad \Phi^T \Phi + \Psi^T \Psi = 0. \tag{42}$$

Here one might use $\tilde{G} = \begin{bmatrix} G_x & G_v \end{bmatrix}^T$ and relate Φ, Ψ to G_x, G_v. A final thing to consider are the structures of G, G_x, G_v: the Gaussian kernel $\kappa(x, y)$ gives the similarity between the points x, y. All trajectories considered here are orbits with some amount of similarity, so I expect a Gram matrix on the trajectories to have a clear line in the trace (every point is identical to itself), but otherwise with a lot of noise. The states however diffuse over time, particles close to the center of phase-space move slower than particles on the outside. I expect only states that are close in time to add any structure to the Gram matrix. Let us find out:



(A) kPCA Gram matrix impression on trajectories.



(B) kPCA Gram matrix impression on states.

As expected, kPCA Gram matrix on the states resembles a stair matrix [8] or a diagonal matrix. Some promising results may be found by further investigating these structures. Without too many restrictions, I believe it to be possible to find the right building blocks to perform real symplectic kPCA.

9.2 PCA versus kPCA for linear model reduction

In this work it turns out that PCA overall performs better in MOR for dynamical systems when independent trajectories are considered and the ROM is a linear projection. This can be seen in the decay of singular values (Figures 14 and 9). Note that the decay of singular values does not depend on how one transforms the data later: even if there was a method that combines kPCA with symplectic model reduction, kPCA is unlikely to outperform PCA if performed on the particle trajectories. One could experiment with many different kernels to solidify this statement, but the Decay of Singular Values as presented here suggests than finding a kernel for which kPCA outperforms PCA in this context is highly unlikely.

Furthermore, regardless on whether or not kPCA outperforms PCA in terms of the singular values, when the method is applied to find a linear projection of the data onto a lower dimensional subspace, it will almost certainly be outperformed by PCA in how well it models the FOM. This is simply by design: kPCA is not designed to find a linear subspace for the data, but instead finds a linear subspace for the non-linearly transformed data, see equation 4 with g in terms of equation 9. PCA is designed to find a linear subspace that maximizes the variance (therefore explaining most of the data). Since any linear subspace on which we project using kPCA does not maximize the variance per se, it is certain to be outperformed by PCA.

9.3 Failure of the SDEIM in this work

As we could see (or, not see) in the results, the SDEIM failed to give a useful projection with which we can approximate the non-linear part of the Hamiltonian. As it turns out however, it is not necessarily the implementation of the SDEIM that is at fault. I tried to run simulations using equation 23, but to no avail. The results are exactly the same

as when the SDEIM was used. Most likely, the transformation from reduced space to physical space is to blame. This might explain why the linear back-mapping of the results performs so poorly. To check this, I inspected A, A^+. Their products should be identity, $AA^+ = \mathbb{I}_{2N}, A^+A = \mathbb{I}_{2k}$. An impression of how well this is the case is given as follows:

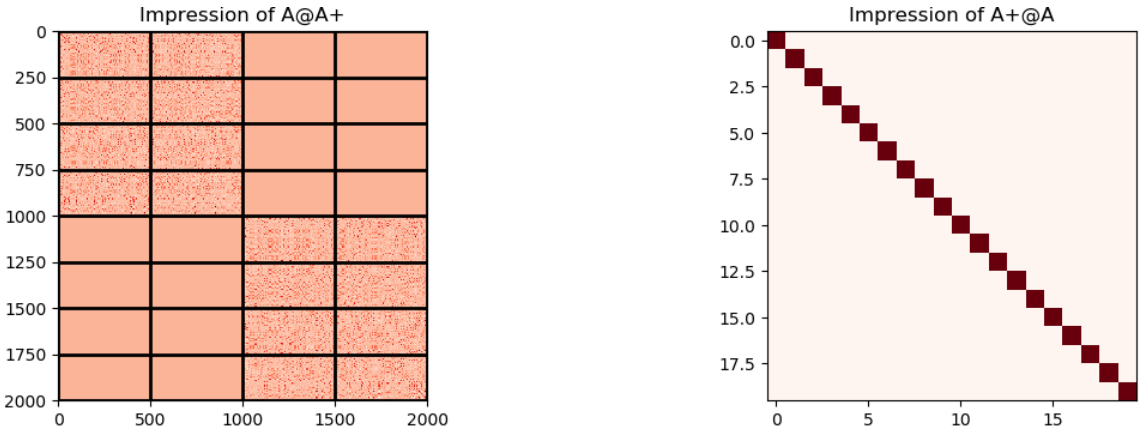


FIGURE 16: kPCA-ROM $A, A+$ impression.

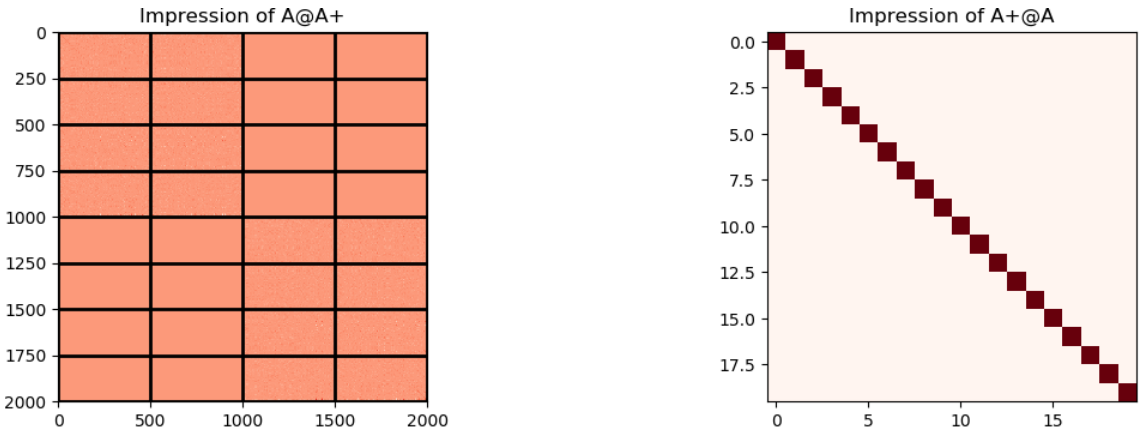


FIGURE 17: PCA-ROM $A, A+$ impression.

As we can see, A^+A is approximately identity, as it is supposed to be. Upon microscopical inspection a similar line is present under AA^+, but it approximates $\mathbb{I}_{2N}$ quite poorly. this implies Az also approximates s poorly. This is disastrous for every ROM considered in this work since section 5 relies heavily on this backward-mapping to work, explicitly in equation 23 and implicitly in equation 25.

It might be the case that the Cotangent lift algorithm was not applicable and I had to use the complex SVD algorithm. In the linear kPCA-ROM case this does surprise me however, as the construction of G should ensure us only real eigenvectors V and therefore $\Phi = V[:, :k]$ should contain all the information about the eigenvalues. In other words, I expect $\Psi = 0$ in equation 18 and the complex SVD algorithm should reduce to the Cotangent lift algorithm for the linear kPCA-ROM. Further investigation as to why this backward-mapping fails is most needed.

9.4 Incompatibility of SDEIM and kPCA

If the problems discussed in the previous discussions were to be solved, there remains yet one more obstacle in the application of kPCA MOR on non-linear Hamiltonian plasma simulations. The reason is simple: equation 25 relies on A to map the data back to physical dimensions so that the non-linear function can be approximated, while for kPCA non-linear projection of the data makes this impossible. In fact, one would need to apply optimization backward-mapping as described in section 3 at every time step of the simulation before one can approximate the non-linear function. This will almost certainly be more expensive in terms of computational power than a straightforward FOM simulation. As of now, I am in no position to advise future researchers on a strategy to solve this problem - I can only advise them to seek other means of approximating the non-linear behaviour of the system.

10 Conclusions

10.1 kPCA for linear Hamiltonian systems

In case the Hamiltonian is linear, MOR can be performed highly accurate, even for new parameter values. A linear ROM that conserves the symplectic form as in section 4 suffices. It is possible to build a linear ROM based on kPCA. This is however not advised: by the design, a symplectic PCA-ROM outperforms a linear symplectic kPCA-ROM. Furthermore, the backward-mapping used for kPCA (optimization, section 3) is outperformed by a linear backward-mapping (symplectic matrix A, section 4). For the best results, one uses the work of Tyranowski and Kraus [12].

10.2 kPCA for non-linear Hamiltonian systems

In the case of non-linear Hamiltonians, the desire to preserve the canonical symplectic form conflicts with the desire to apply kPCA on the data. As has been shown in section 5, we have to evaluate the non-linear part of the Hamiltonian $f_N(s)$. To this end, we applied strategies to approximate $f_N(s)$ using only $2k$ spatial indices of s in the hopes that these spatial indices of s approximate the overall behaviour well. With the help of the SDEIM, this evaluation of $f_N(s)$ is projected to our ROM. The effectiveness of this method could unfortunately not be demonstrated in this work because of a mistake in the construction of the symplectic matrix A and its symplectic inverse A^+ as has been shown in the discussion.

In case one uses a PCA-ROM or a linear kPCA-ROM, the data can be transformed and projected in a straightforward way, $z = As, s = A^+z$. In theory this property makes it possible to use the SDEIM to approximate the non-linear behaviour of the system. From this point, one can follow the instructions of Peng and Mohseni [9] to construct a ROM.

Unfortunately in the case of kPCA, there is no easy way to map z to s, and the process of finding $2k$ spatial indices of s to approximate $f_N(s)$ is computationally expensive, making the kPCA method less desirable. In theory, one would have to perform optimization at every time step of the simulation to find an approximation of s with which one can approximate $f_N(s)$. To make up for that one would have to find a very fast method of backward-mapping or use some hybrid ROM - or better yet, develop an approximation of $f_N(s)$ that is compatible with kPCA.

References

[1] Patrick Buchfink, Silke Glas, and Bernard Haasdonk. Symplectic model reduction of hamiltonian systems on nonlinear manifolds. *arXiv preprint arXiv:2112.10815 [math.NA]*, 2021.

[2] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM J. Sci. Comput.*, 32:2737–2764, 2010.

[3] Francis Filbet and Eric Sonnendrücker. Numerical methods for the Vlasov equation. In *Numerical mathematics and advanced applications*, pages 459–468. Springer, 2003.

[4] Alberto García-González, Antonio Huerta, Sergio Zlotnik, and Pedro Díez. A kernel principal component analysis (kPCA) digest with a new backward mapping (pre-image reconstruction) strategy. *arXiv preprint arXiv:2001.01958*, 2020.

[5] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the störmer–verlet method. *Acta Numerica*, 12:399–450, 2003.

[6] Jan S. Hesthaven, Cecilia Pagliantini, and Nicolò Ripamonti. Rank-adaptive structure-preserving model order reduction of hamiltonian systems. *ESAIM : Mathematical Modelling and Numerical Analysis*, 56(2):617–650, March 2022.

[7] Sebastian Kirsch. Principal components analysis explained for dummies. Progammathically.com, `https://programmathically.com/principal-components-analysis-explained-for-dummies/`, 2022.

[8] Hao Lu. Stair matrices and their generalizations with applications to iterative methods i: A generalization of the successive overrelaxation method. *SIAM Journal on Numerical Analysis*, 37(1):1–17, 1999.

[9] Liqian Peng and Kamran Mohseni. Symplectic model reduction of hamiltonian systems. *SIAM Journal on Scientific Computing*, 38:A1–A27, 02 2016.

[10] Syuzanna Sargsyan, Steven L. Brunton, and J. Nathan Kutz. Nonlinear model reduction for dynamical systems using sparse sensor locations from learned libraries. *Phys. Rev. E*, 92:033304, Sep 2015.

[11] D. C. Sorensen and M. Embree. A deim induced cur factorization. *arXiv preprint arXiv:1407.5516 [math.NA]*, 2014.

[12] Tomasz M Tyranowski and Michael Kraus. Symplectic model reduction methods for the Vlasov equation. *arXiv preprint arXiv:1910.06026*, 2019.

[13] Meiqing Zhang and Robert D. Skeel. Cheap implicit symplectic integrators. *Applied Numerical Mathematics*, 25(2):297–302, 1997. Special Issue on Time Integration.

A Proof of $A^T = A^+$

Theorem A.1. *Let $\Delta \in \mathbb{R}^{N\times M}$ and its Singular Value Decomposition $\Delta = U'\Sigma V^T$ be given. Suppose $U' \in \mathbb{R}^{N\times N}$. It follows that $\Phi = U'[:, :K]$ is also real. Then, the symplectic matrix A constructed using equation 17 transposed is equal to the symplectic inverse of A, $A^T = A^+$.*

Proof. The proof is straightforward: plug matrix A in equation 16. It will yield A^T.

$$\begin{aligned} A^+ = \mathbb{J}_{2k}^T A^T \mathbb{J}_{2N} &= \begin{bmatrix} 0 & -\mathbb{I}_N \\ \mathbb{I}_N & 0 \end{bmatrix} \begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix}^T \begin{bmatrix} 0 & \mathbb{I}_N \\ -\mathbb{I}_N & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\mathbb{I}_N \\ \mathbb{I}_N & 0 \end{bmatrix} \begin{bmatrix} \Phi^T & 0 \\ 0 & \Phi^T \end{bmatrix} \begin{bmatrix} 0 & \mathbb{I}_N \\ -\mathbb{I}_N & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -\Phi^T \\ \Phi^T & 0 \end{bmatrix} \begin{bmatrix} 0 & \mathbb{I}_N \\ -\mathbb{I}_N & 0 \end{bmatrix} = \begin{bmatrix} \Phi^T & 0 \\ 0 & \Phi^T \end{bmatrix} = A^T. \end{aligned}$$

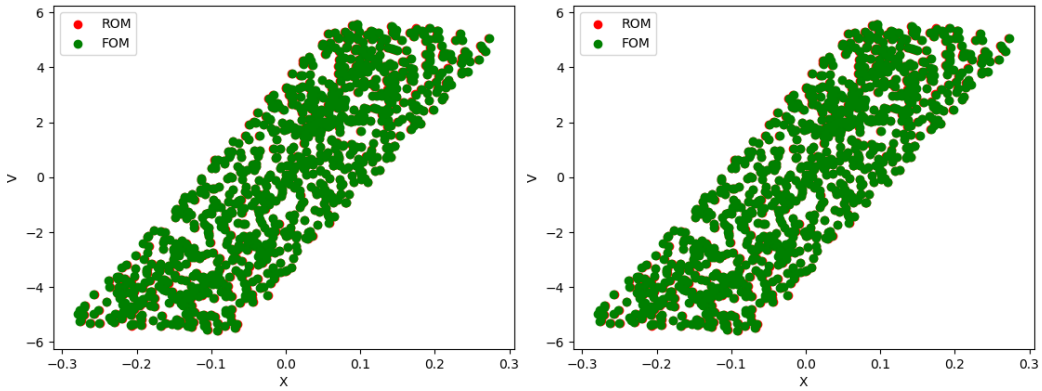
Quite easily one can also check that A^T is indeed the symplectic inverse of A by checking $A^T A = \mathbb{I}_{2k}$ and $AA^T = \mathbb{I}_{2N}$:

$$\begin{aligned} A^T A &= \begin{bmatrix} \Phi^T & 0 \\ 0 & \Phi^T \end{bmatrix} \begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix} = \begin{bmatrix} \Phi^T\Phi & 0 \\ 0 & \Phi^T\Phi \end{bmatrix} = \begin{bmatrix} \mathbb{I}_K & 0 \\ 0 & \mathbb{I}_K \end{bmatrix} = \mathbb{I}_{2k}, \\ AA^T &= \begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix} \begin{bmatrix} \Phi^T & 0 \\ 0 & \Phi^T \end{bmatrix} = \begin{bmatrix} \Phi\Phi^T & 0 \\ 0 & \Phi\Phi^T \end{bmatrix} = \begin{bmatrix} \mathbb{I}_N & 0 \\ 0 & \mathbb{I}_N \end{bmatrix} = \mathbb{I}_{2N}. \end{aligned}$$

Here I used the fact that the columns of U' form an orthonormal basis in $\mathbb{R}^{N\times N}$ [4]. □

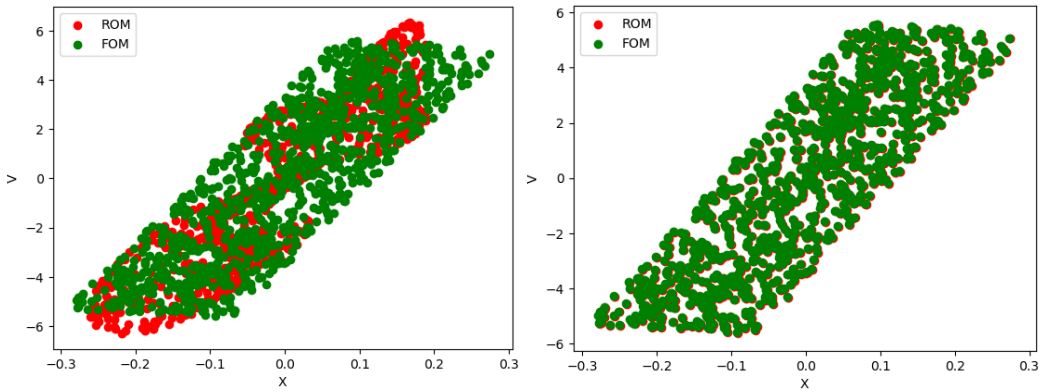
B Results - extra graphs

B.1 Linear Hamiltonian ROM-FOM comparison $\beta = 0.7$



(A) kPCA-MOR with optimization back-mapping.

(B) PCA-MOR with optimization back-mapping.



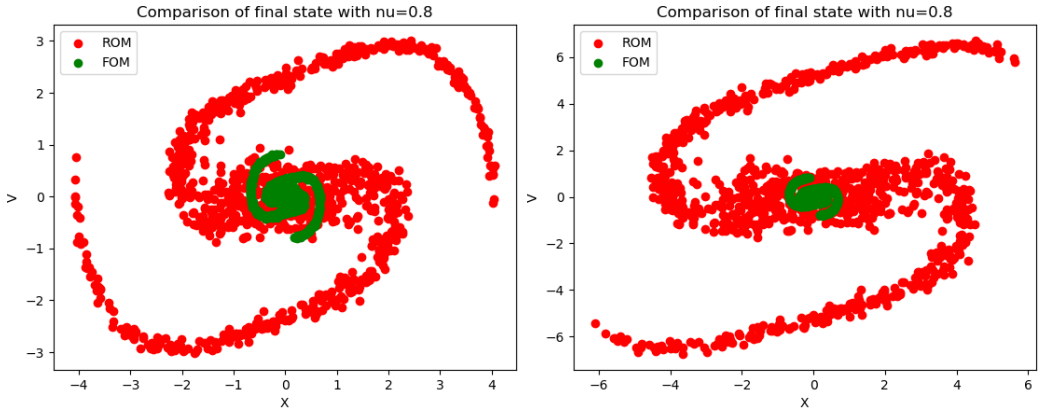
(C) kPCA-MOR with linear back-mapping.

(D) PCA-MOR with linear back-mapping.

FIGURE 18: Comparison of reduced models for $\beta = 0.7$.

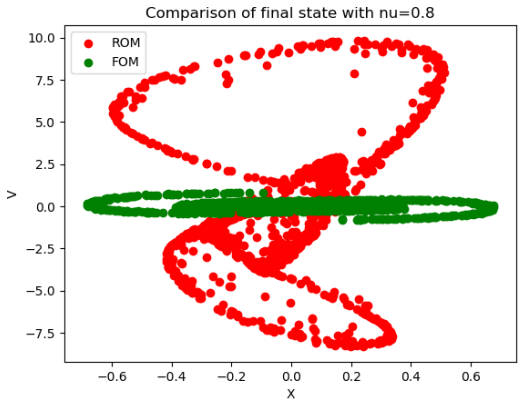
As promised, the results for $\beta = 0.7$ are identical to the results for $\beta = 0.5$.

B.2 Non-linear Hamiltonian ROM-FOM comparison $\nu = 0.8$

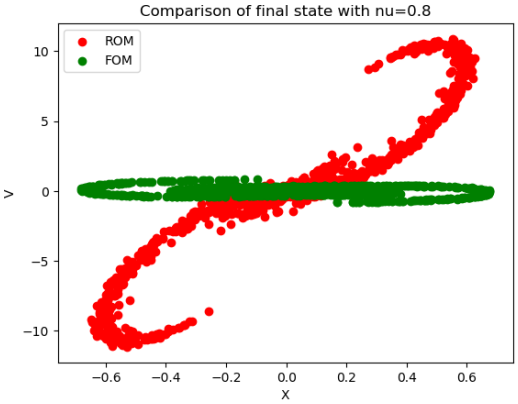


(A) kPCA-ROM with optimization back-mapping.

(B) PCA-ROM with optimization back-mapping.



(C) kPCA-ROM with linear back-mapping.

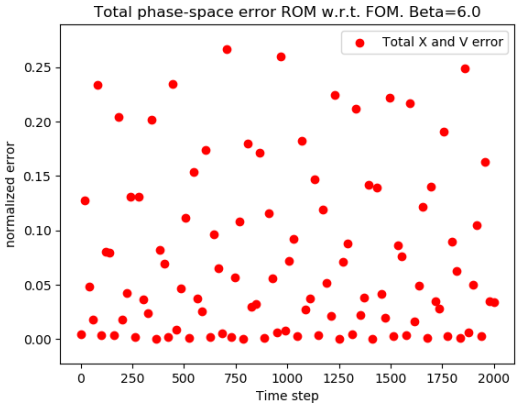


(D) PCA-ROM with linear back-mapping.

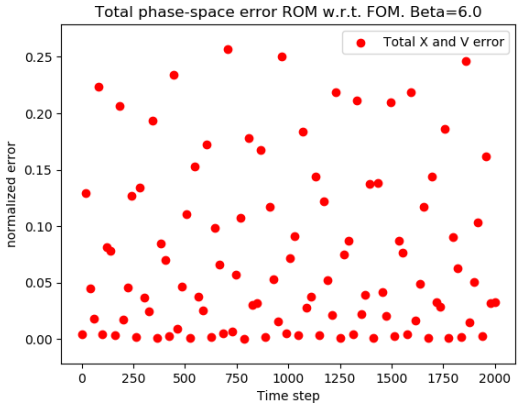
FIGURE 19: Comparison of reduced models for $\nu = 0.8$.

Indeed the ROM's do not improve for $\nu = 0.8$. It also no longer looks like kPCA outperforms PCA, for as far as that may have been the case.

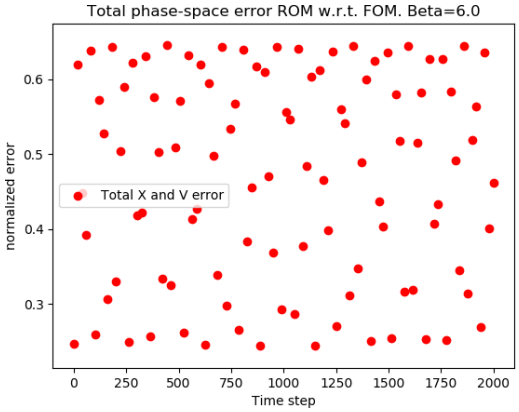
B.3 Error graphs of novel parameters



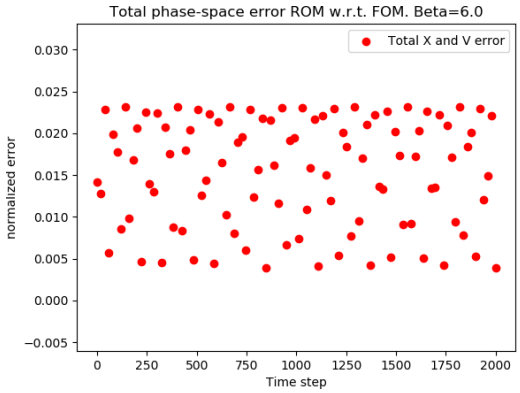
(A) kPCA-ROM with optimization back-mapping.



(B) PCA-ROM with optimization back-mapping.

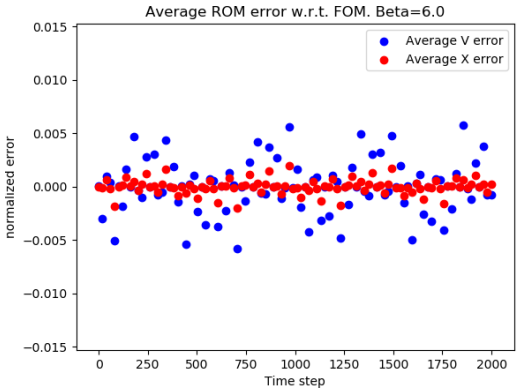


(C) kPCA-ROM with linear back-mapping.

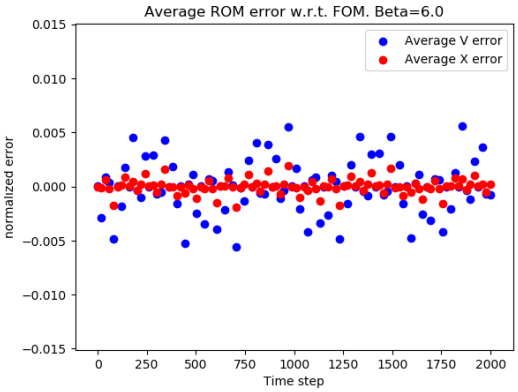


(D) PCA-ROM with linear back-mapping.

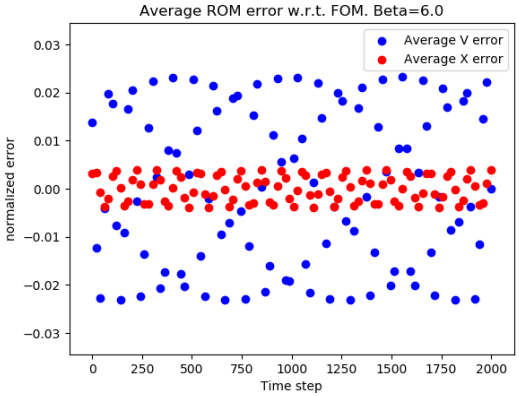
FIGURE 20: Linear-Hamiltonian total error comparison of reduced models for $\beta = 0.6$.



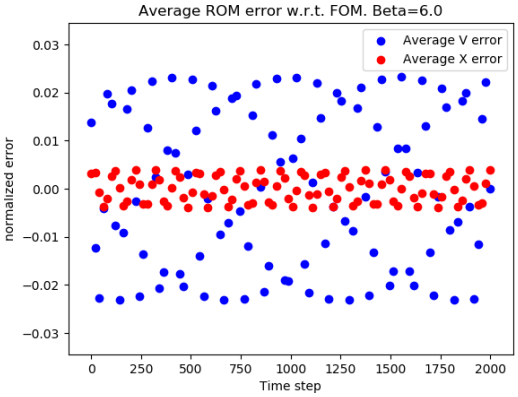
(A) kPCA-ROM with optimization back-mapping.



(B) PCA-ROM with optimization back-mapping.

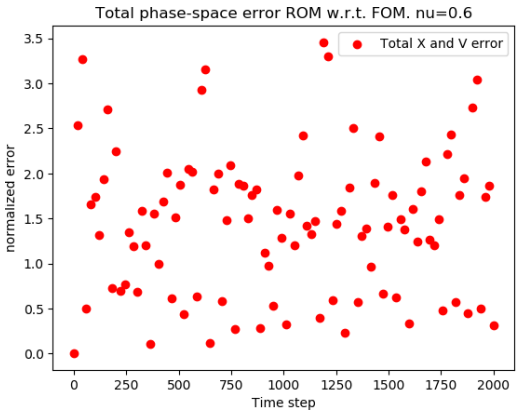


(C) kPCA-ROM with linear back-mapping.

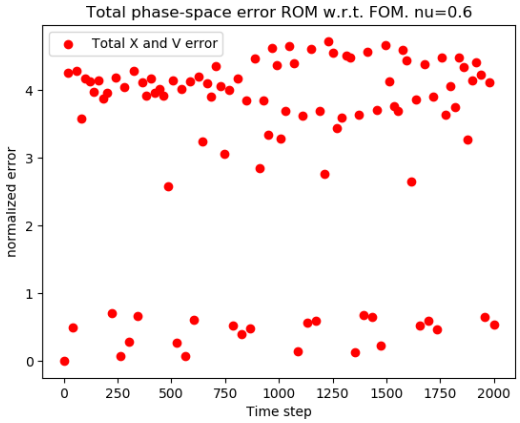


(D) PCA-ROM with linear back-mapping.

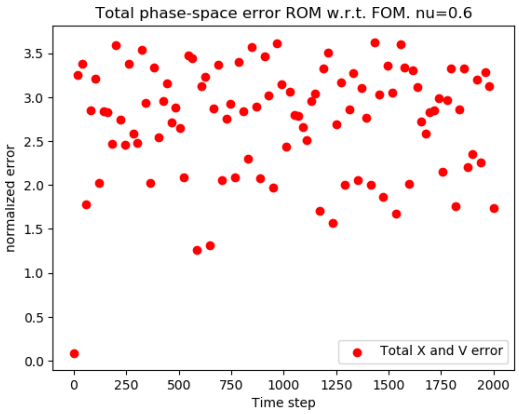
FIGURE 21: Linear-Hamiltonian X,V error comparison of reduced models for $\beta = 0.6$.



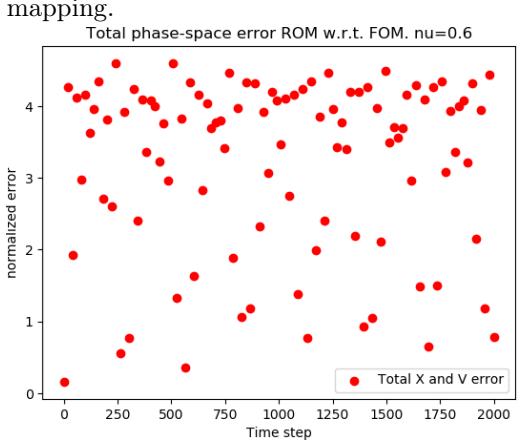
(A) kPCA-ROM with optimization back-mapping.



(B) PCA-ROM with optimization back-mapping.

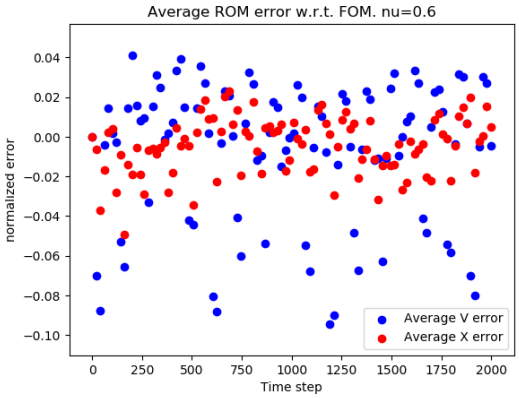


(C) kPCA-ROM with linear back-mapping.

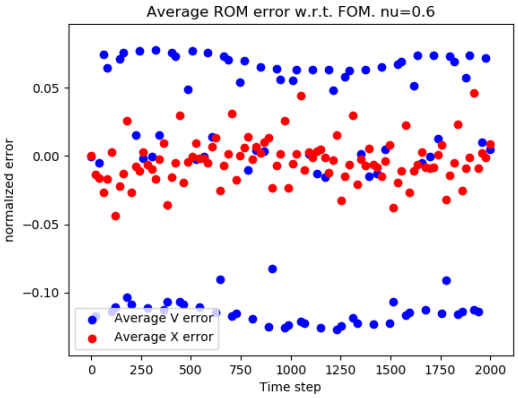


(D) PCA-ROM with linear back-mapping.

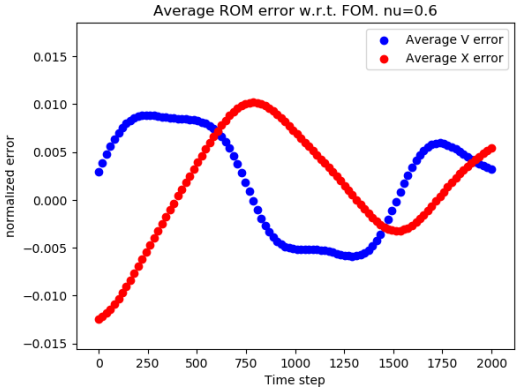
FIGURE 22: Non-linear Hamiltonian total error comparison of reduced models for $\nu = 0.6$.



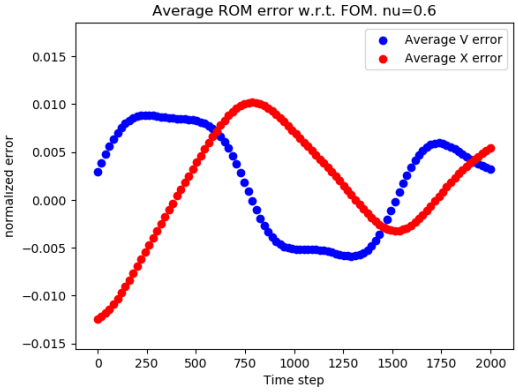
(A) kPCA-ROM with optimization back-mapping.



(B) PCA-ROM with optimization back-mapping.



(C) kPCA-ROM with linear back-mapping.



(D) PCA-ROM with linear back-mapping.

FIGURE 23: Non-linear-Hamiltonian X,V error comparison of reduced models for $\nu = 0.6$.

C kernel-PCA applied on images

In a first attempt to apply kPCA, I reconstruct the problem of García-González [4]. I made a video of my hand opening and closing several times and edited it to get 475 black and white pictures, with dimensions (523,352) for a total of 184096 pixels per picture. The goals is catch all features in as few dimensions possible, reducing the dimensions from 184096 to less than 10. In order to do this, I took 100 random pictures out of the first 175 as my data. The data is ordered as a 184096×100 matrix S. I compare both the usual PCA and the kPCA method.

Since the matrix of data S (184096×100) is too large to perform straightforward SVD on, PCA was performed by diagonalizing the Gram matrix instead. The Gram matrix is constructed as $G_{100\times100} = S^T S$. G is then diagonalized and the dimension is reduced to $k = 3$. G The data can easily be mapped to reduced space, $Z = \Phi^T G$ as in section 3. Since the matrix U from the SVD of X is not available, the data projected using PCA must undergo the same optimization process as the data projected using kernel-PCA.

To compare, we apply kPCA on the data as well. G is constructed using the Gaussian kernel, normalized by the first picture. Once more, G diagonalized and the dimension is reduced to $k = 3$. The images are mapped in the same way as in regular PCA. New images s^* are mapped to $g^* : [g^*]_i = \kappa(s^*, s_i)$. The Decay of the Singular Values of both matrices

is given in the following graph:

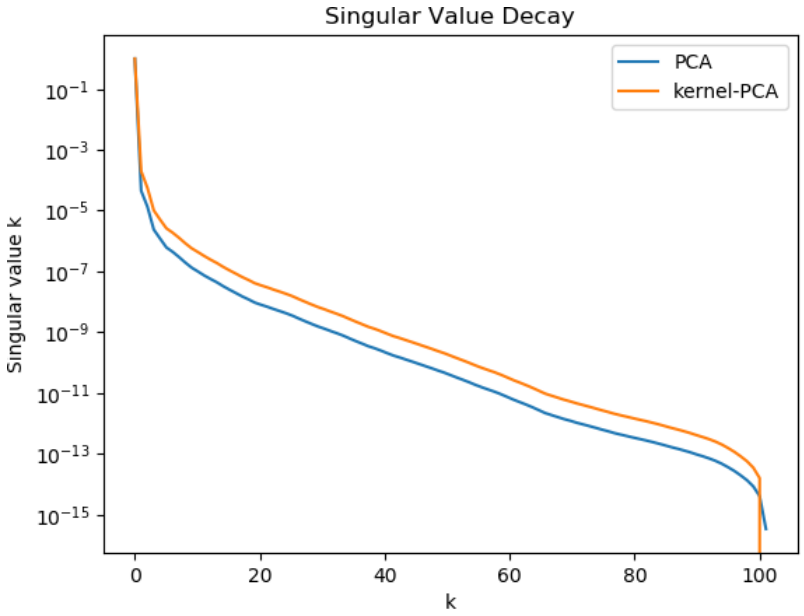
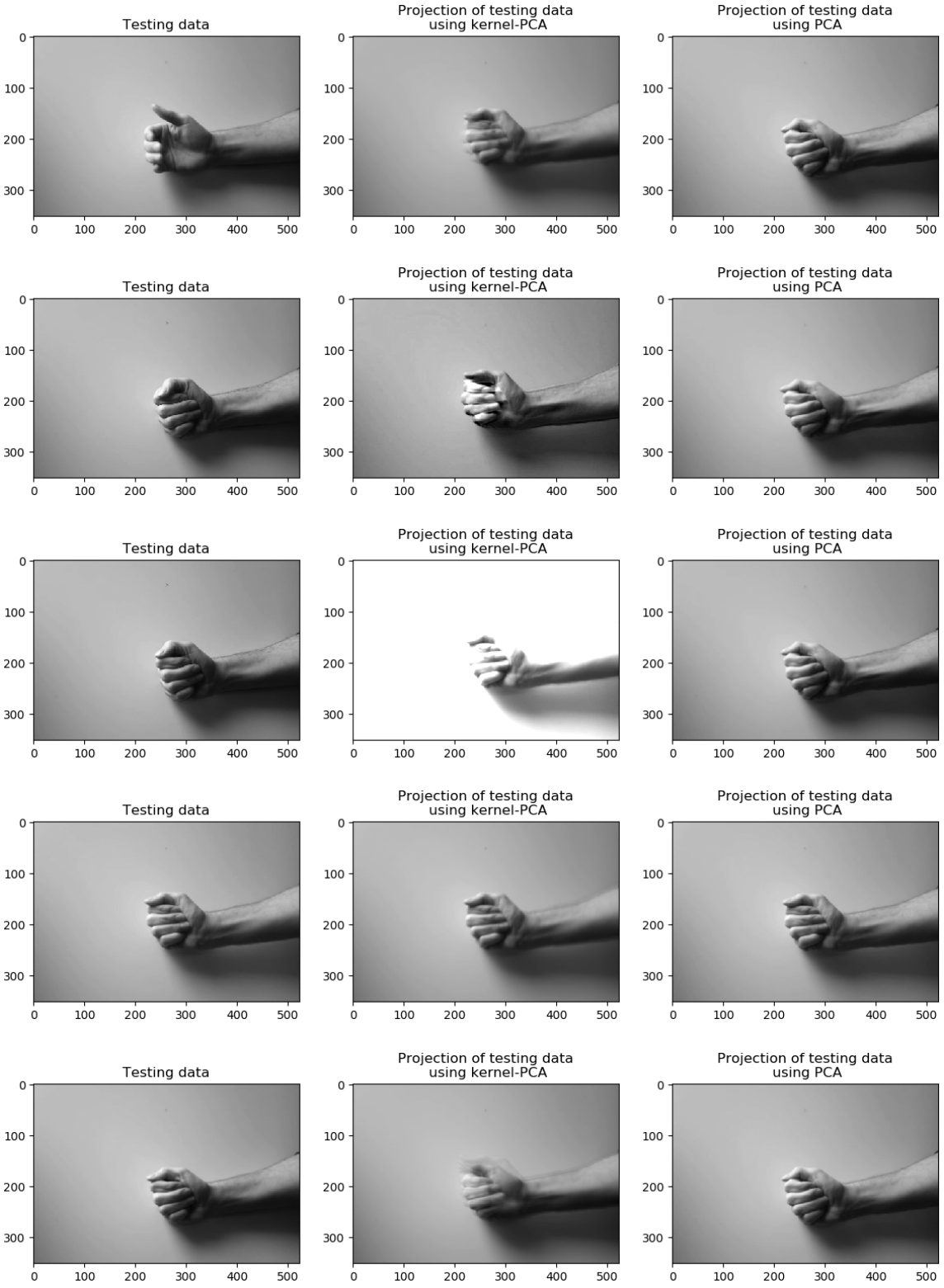


FIGURE 24: Decay of Singular Values

The decays are actually almost identical. It seems like there is no great benefit of using kPCA over PCA so far. In fact, PCA once more performs slightly better. A few examples of reconstructed images can be found below. To prevent the back-mapping from being stuck in local minimum, most of the PCA reconstructions were mapped to the image of the closest point in reduced space. This turns out to be a better starting point for back-mapping than the initial point I have chosen for all back-mappings to start, which explains why the PCA reconstructions look slightly better than the kPCA reconstructions.

As a side-note, I did not center the data for these images, nor did I center the Gram matrix G. This is good practise, but not necessary. What is necessary is that one is consistent in either centering their data or not in comparing ROM's.



Unfortunately there is no example where my hand is open, but the reconstructions would compare in the same way.

D Model reduction of N-pendulums

With slight abuse of notation, in this section I will at times refer to the angle θ of a pendulum as X and its momentum p as V.

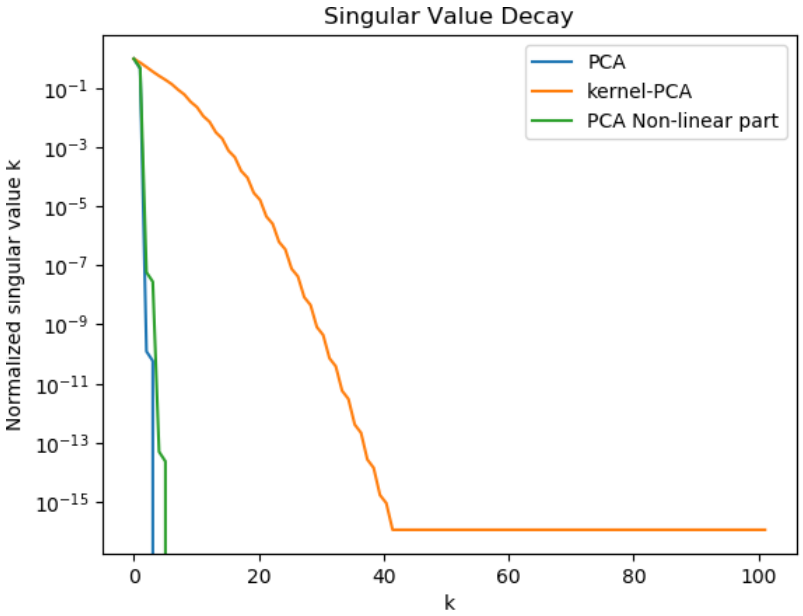
To test out how kPCA may be applied to the dimensional reduction of a dynamical system, I constructed a toy model of 100 pendulums. The system of equations derived from the Hamiltonian for one pendulum is [5]:

$$\dot{\theta} = p \tag{43}$$

$$\dot{p} = -sin(\theta) \tag{44}$$

One pendulum is simulated for 2000 time steps using the implicit midpoint method, which is a symplectic integrator. From its path, 100 values for the angle and momentum are taken with a constant phase shift. These values serve as initial conditions for our 100 pendulums, all of which follow the same path. Then these are simulated for 1000 time steps to construct the data on which we apply kPCA. This way, the paths of all pendulums are in fact dependent on the path of only one pendulum, and model reduction is ensured.

In order to apply kPCA on the system, I once more consider the pendulum trajectories as my samples. The data is reduced to a dimension of $k = 3$ for the PCA-ROM and for the kPCA-ROM. From here, the data is mapped to reduced space using the methods of section 3. As a first attempt, the ROM constructed to simulate the reduced data is as if the Hamiltonian were linear (naturally, for small angles, this is approximately true). In other words, I use Corollary 4.0.1. I simulate the ROM's using Störmer-Verlet. The Decay of the Singular Values are as follows:



Once more, PCA proves superior when considering trajectories. The SVD of the non-linear part is drawn as well. Some of the states of the system as approximated as follows:

[5] This derivation can be found in any book on classical mechanics or on any online resource.

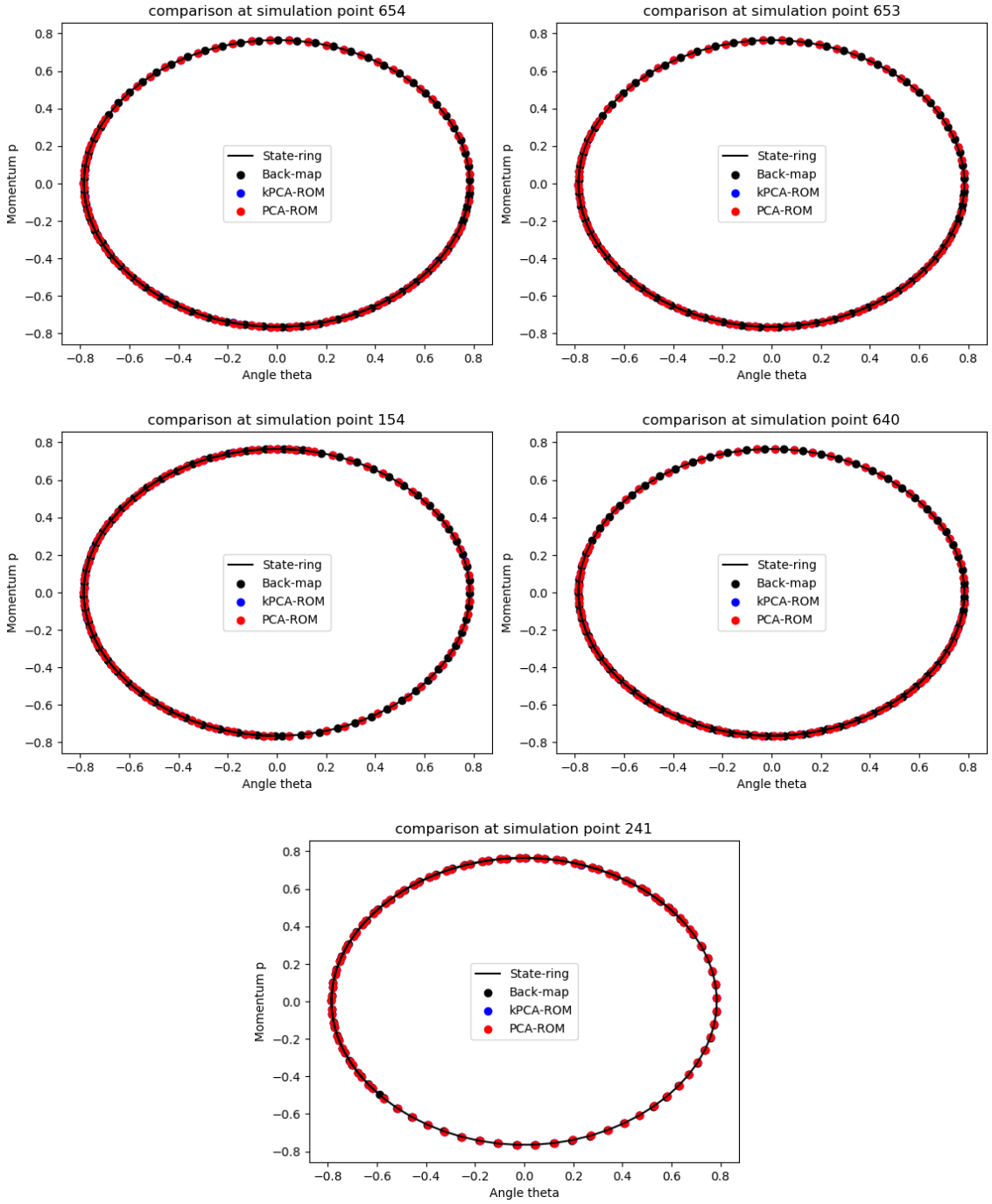
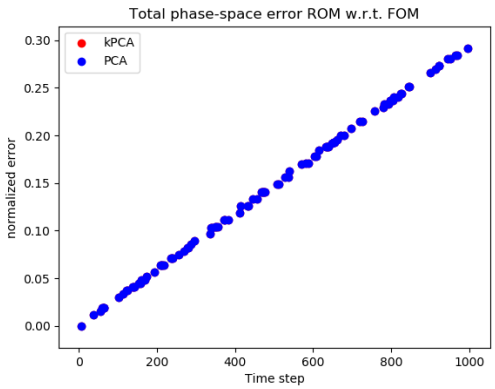


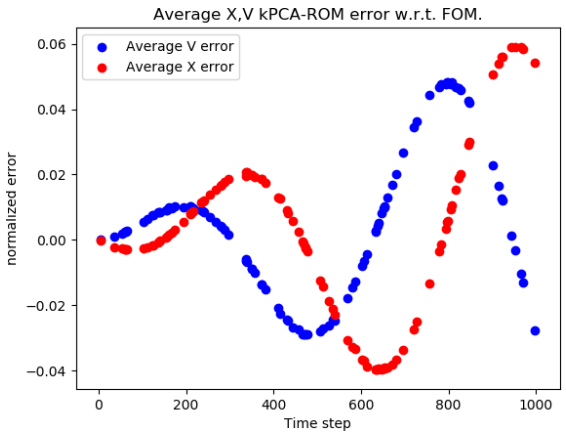
FIGURE 26: Phase diagrams of the pendulums. The ROM's have a dimension of k=3.

In the phase diagrams, I added in black a line on which the original samples lie at each point in time. Since there is no dissipation, energy is conserved, and the samples should stay on that line at all times. We can see that the kPCA-ROM and the PCA-ROM overlap completely. They do not match the back-map (with which I mean the real sample, no back-mapping involved) exactly, but the ROM's do manage to stay on the ring, which is an excellent quality, since energy should be preserved.

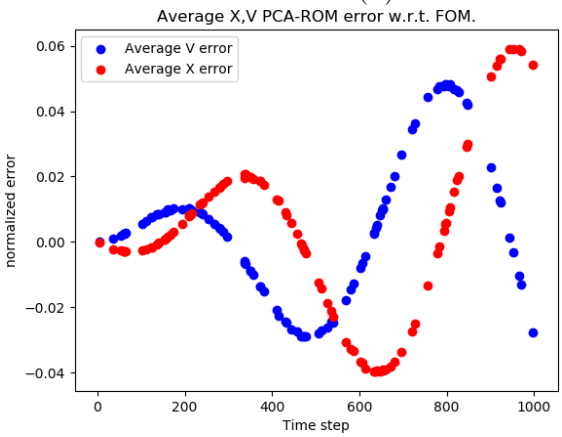
I will also zoom in on the errors:



(A) Total error



(B) kPCA-MOR average X,V error.



(C) PCA-MOR average X,V error.

We can see that the PCA-ROM and the kPCA-ROM indeed overlap completely, error and all. This seems like the only scenario where a kPCA based linear ROM can equal PSD.

One more time, I apply the techniques of section 5 and apply SDEIM to the problem at hand. It is very much possible that the problems arising in the plasma results are not solved here.

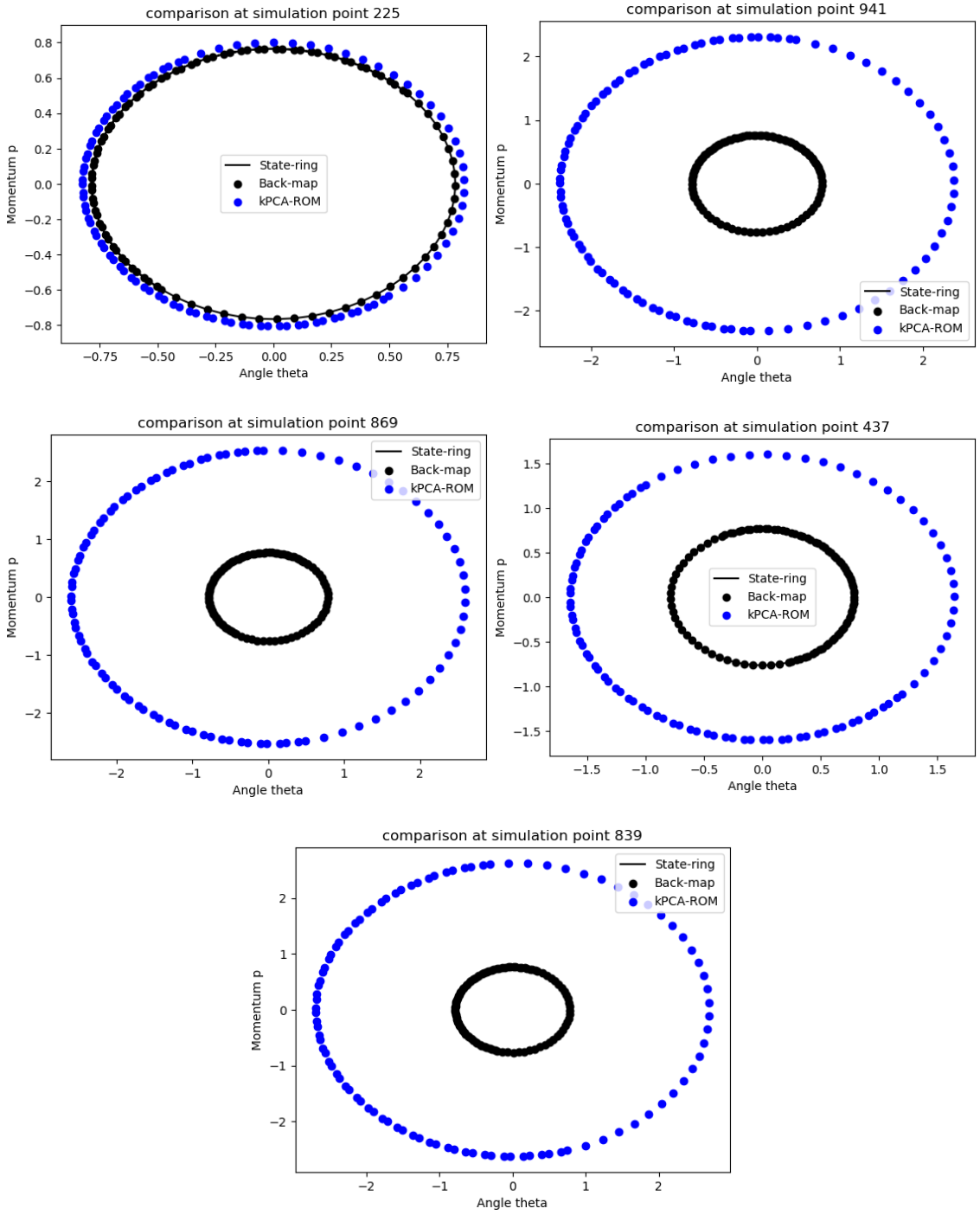
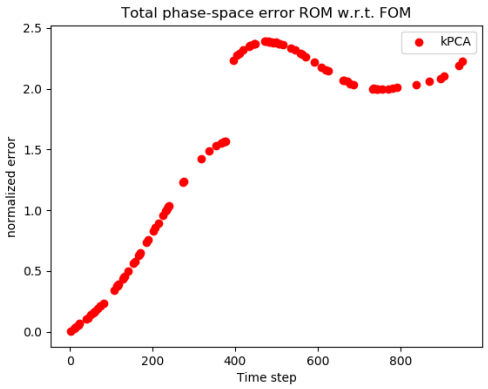


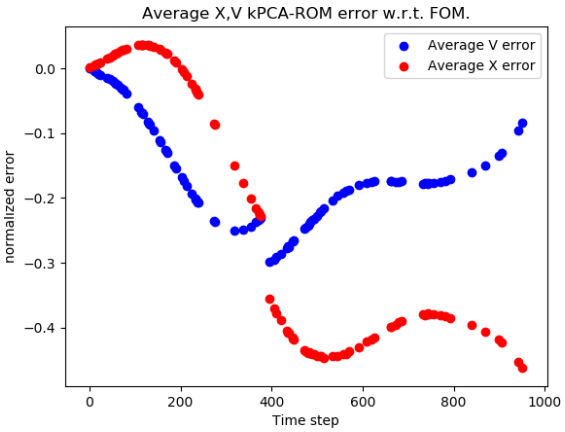
FIGURE 28: Phase diagrams of the pendulums. The ROM's have a dimension of k=3.

Clearly, the projections of Section 5 do not improve the model. What can be observed immediately is that the new model seems to have no energy conservation. One suggestion of my supervisor Jeroen Verschuur was to project the angle and momentum to those angles and momenta that retain the energy that they should have. Though this was more in the context of the linear ROM's, it may very well be applied here. As the system consists of independent pendulums, all of which should conserve their own energy, such a projection

should be a feasible solution to our problem. At the very least, the main issue of energy not being conserved would be solved in doing do. Unfortunately, there was no time remaining for me to further explore this suggestion. It is however a very interesting extra topic: what projection onto energy-preserving states models our pendulums best? An orthogonal projection might do the job. I will also zoom in on the errors again:



(A) Total error



(B) kPCA-MOR average X,V error.

E One-size-fits-all gradient descend using a PID control

In this section, I try to elaborate on the gradient descend method that I use for the backward-mapping of my ROM's. I refer to section 3 for context. The goal is to find optimal weights.The initial weights are determined by $d(i)$: $[w_0]_i = (1 - d(i))^\alpha$. Although this initially assumes that s_j lies in the interior of all s_i, which need not hold necessarily, it seems like a reasonable initial set of weights.The parameter α depends on how close you want to make the original guess to the one closest point. The stochastic gradient descent algorithm approximates the gradient in terms of the weights and aims to move in the direction of steepest descend. In that spirit, algorithm 2 is implemented.

Note: the choice for stochastic gradient descent is based on its robustness and simple implementation. It almost assuredly finds a local minimum. To possibly improve, one can alter this method to the mini-batch gradient descend method, where one takes several randomly selected weights instead of only one. The choice of δ depends on the problem. Generally one can take $\epsilon \times 10^{-3}$ or so.

E.1 PID control

One improvement I made to the Gradient Descend algorithm is to use a PID control on η. Since convergence was often slow for several points in reduced space, while it was oscillating for others, it seemed like a good idea to make the rate of change towards the direction of steepest descend depend on the error that we still have to bridge. If the algorithm oscillates around a certain (local) minimum, the parameters for the proportional and integral part of the PID are decreased to a fraction of their size. Initial values $\mathbb{P}, \mathbb{I}, \mathbb{D}$ are chosen and at each step the current error $p = J(w)$, the integral error $I = \int J(w)$ and h times the derivative $d = J(w_i) - J(w_{i-1})$ are taken to compute $\eta = |\mathbb{P}p + \mathbb{I}I + \mathbb{D}d|$. At each step of the optimization, the algorithm checks that it is not oscillating around an equilibrium. If

Algorithm 2 Gradient descend

Require: Data X, transformed data Z, Functional J
Require: Vector with indices of close points c
Ensure: weights w that minimize $J(w)$
  ini **d**: $[d]_i$ from (eq. 18) for points c
  ini $w_0 : [w_0]_i = (1 - [d]_i)^\alpha$
  $\mathbf{w} = \frac{w_0}{||w_0||}$
  $l = 0$
  while $J(\mathbf{w}) > \epsilon$ **do**
    for $w_i \in \mathbf{w}$ **do**
      calculate $\nabla_{wi} J$
      $w_i \to w_i - \eta * \nabla_{wi} J$
    end for
    if $||\nabla J|| < \delta$ **then**
      Reset to new weights: $\mathbf{w} = \mathbf{0}$
      $[w]_l = 1$
      $l \to l + 1$ ▷ This means we are stuck in a local minimum
    end if
    $\eta \to 0.999\eta$
  end while

that is the case, the parameters $\mathbb{P}, \mathbb{I}$ are decreased to 99% of their value. If the algorithm resets the weights because it is stuck in a (too high) local minimum, the parameters $\mathbb{P}, \mathbb{I}$ are reset to their original value.