SoftwareX 12 (2020) 100630

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

PCAfold: Python software to generate, analyze and improve PCA-derived low-dimensional manifolds

Kamila Zdybał [a,b,*], Elizabeth Armstrong [c], Alessandro Parente [a,b], James C. Sutherland [c]

[a] Université Libre de Bruxelles, École polytechnique de Bruxelles, Aero-Thermo-Mechanics Laboratory, Brussels, Belgium
[b] Université Libre de Bruxelles and Vrije Universiteit Brussel, Combustion and Robust Optimization Group (BURN), Brussels, Belgium
[c] Department of Chemical Engineering, University of Utah, Salt Lake City, UT, USA

ARTICLE INFO

Article history:
Received 14 September 2020
Received in revised form 14 November 2020
Accepted 16 November 2020

Keywords:
Low-dimensional manifolds
Dimensionality reduction
Sampling
Principal component analysis

ABSTRACT

Many scientific disciplines rely on dimensionality reduction techniques for computationally less expensive handling of multivariate data sets. In particular, Principal Component Analysis (PCA) is a popular method that can be used to discover the underlying low-dimensional manifolds in high-dimensional data sets. PCA-derived manifolds are formed by projecting the original data set onto a new basis spanned by the first few Principal Components (PCs). In many cases, it is crucial that the manifold maintains certain topological characteristics for its subsequent analysis and parameterization. Avoiding overlap or steep gradients of a dependent variable could be two desired examples. In this paper, we present PCAfold, a Python software package that can be used to generate, improve and analyze low-dimensional manifolds. Our software incorporates data preprocessing, clustering and sampling techniques, uses PCA as a data reduction technique and utilizes a novel approach to assess the quality of the obtained low-dimensional manifolds.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00048
Code Ocean compute capsule	NA
Legal Code License	MIT license
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments	Python 3.7 with Cython, matplotlib, NumPy, SciPy
If available Link to developer documentation/manual	https://pcafold.readthedocs.io
Support email for questions	kamilazdybal@gmail.com, Elizabeth.Armstrong@chemeng.utah.edu, Alessandro.Parente@ulb.be, James.Sutherland@utah.edu

1. Motivation and significance

Many scientific disciplines use data science techniques to reduce dimensionality of numerical or experimental data sets. Often, high-dimensional data sets exhibit low-rank structures that allow for effective description of a problem using fewer variables. Such data sets can be projected onto fewer dimensions to form a low-dimensional manifold. In that new space, the analysis and handling of data is potentially easier and computationally less demanding. The quality of the obtained manifold is often pertinent to the success of the analysis performed in the reduced space. One question that can be addressed in this regard is: does the manifold possess certain desired topological characteristics? For instance, if a dependent variable is to be regressed on the identified manifold, it is crucial to minimize behaviors such as overlaps or steep gradients [1]. Among the methods used to identify manifolds empirically, Principal Component Analysis (PCA) is

* Corresponding author at: Université Libre de Bruxelles, École polytechnique de Bruxelles, Aero-Thermo-Mechanics Laboratory, Brussels, Belgium.
E-mail addresses: Kamila.Zdybal@ulb.ac.be, kamilazdybal@gmail.com (K. Zdybał), Elizabeth.Armstrong@chemeng.utah.edu (E. Armstrong), Alessandro.Parente@ulb.be (A. Parente), James.Sutherland@utah.edu (J.C. Sutherland).

https://doi.org/10.1016/j.softx.2020.100630

widely recognized and used, mainly due to its linearity and ease of implementation. Numerous areas of research such as image processing and classification [2], biology [3], fluid dynamics [4] or combustion [5,6] exhibit growing interest in low-dimensional data representations. In this paper, we describe `PCAfold` - a Python software package for generating, analyzing and improving PCA-derived manifolds.

`PCAfold` introduces a variety of data preprocessing tools that allow the user to alter the manifold through initial data manipulation, prior to performing PCA. For instance, it is recognized that scaling can have a significant effect on the shape of the manifold [7]. A less explored problem is the understanding of data sampling to tackle imbalanced data sets in creating PCA representation. Several authors report an advantage of changing sampling densities of train and test data when using data analysis or machine learning techniques [8,9]. This suggests that PCA-derived manifolds can be altered and possibly improved through adequate sampling. The software that we propose in this paper can contribute to research on the effect of data manipulation on the quality of low-dimensional data representation. As an example, if the manifold is used as an input (regressor) for techniques such as Artificial Neural Networks (ANN) or Gaussian Process Regression (GPR), the learning process can be improved if the manifold satisfies necessary criteria. This brings the attention to the second important functionality introduced in the presented software which is manifold assessment. The novel metric provided in the `PCAfold` software has been proposed in [10]. It can be used for quantitative assessment of manifold dimensionality and for comparing manifold parameterizations according to scales of variation and uniqueness of dependent variable values.

1.1. Related work and comparison with existing tools

In addition to including novel functionalities which will be described later, `PCAfold` software also combines several tools and algorithms from the existing literature into one toolbox. Such a framework broadens the type and quality of analysis that can be performed by the user. Within the data preprocessing tools, scaling of multivariate data sets can be performed with several popular options such as Auto, VAST [11], Pareto [12], Range or Poisson [13]. Two outlier detection algorithms based on Mahalanobis distance and Principal Component classifier (PCC) proposed in [14] are implemented in this software. The effect of both scaling and outlier detection on low-dimensional manifolds was also studied in [7]. In addition, kernel density weighting [15] is available as another data preprocessing option for unbalanced data sets. Among a few simple clustering techniques, the option to cluster combustion data sets by partitioning mixture fraction variable is available [6]. `PCAfold` includes functions for splitting data sets into train and test data for use in conjunction with data reduction or machine learning algorithms. In addition to random splitting, extended sampling methods are implemented here that allow the user to draw samples from local clusters [8,16] defined by the user.

Principal Component Analysis [17] is used in the `PCAfold` software as a dimensionality reduction technique and several PCA-related tools such as selection of principal variables (PVs) using B4, B2 and M2 methods [18,19] and techniques for finding the number of retained eigenvalues [18,20,21] are included. The influence of principal variable selection on the PCA reconstruction of data sets was studied in [22]. PCA functionalities also accommodate treatment of sources of Principal Components which can be valuable for implementing PC-transport approaches such as proposed in [23]. Several novel functionalities for performing PCA on sampled data sets and analyzing the results are introduced. The novel normalized variance metric implemented in `PCAfold` has been proposed in [10] and utilizes Nadaraya–Watson kernel regression [24] for quantitative assessments of low-dimensional manifolds.

Several existing Python libraries provide some of the functionalities implemented in `PCAfold`. The scikit-learn Python library [25] contains a commonly used implementation of Principal Component Analysis as well as functions for standardizing data sets, clustering and sampling. Software that introduces multivariate preprocessing techniques but for time series data sets, was proposed in [26]. The OpenMORe package [27] can be used for clustering reacting flow data sets and it additionally introduces utilities for evaluating clustering solutions. Python software for under-sampling or over-sampling of class-imbalanced data sets is the Imbalanced-learn toolbox [28]. It is also worth mentioning an outlier detection toolbox PyOD [29] that is much broader in the scope of available detection algorithms. While the aforementioned packages accomplish specific data analysis tasks, we note that `PCAfold`'s aim is to create a complete workflow for improving and analyzing PCA-derived low-dimensional manifolds. It thus brings together some of the popular techniques for completeness and also introduces novel tools that extend the functionalities of the aforementioned packages.

2. Software description

2.1. Software overview and architecture

`PCAfold` is an open-source library written in Python 3.7. The software can be downloaded from the GitLab repository provided and installed following the standard installation from the `setup.py` file. `PCAfold` has a simple architecture composed of three modules: `preprocess`, `reduction` and `analysis`. Specifically:

- The `preprocess` module incorporates a variety of data preprocessing tools including centering and scaling, kernel density weighting, outlier detection, clustering and sampling.
- The `reduction` module contains an implementation of Principal Component Analysis (PCA) along with functionalities to perform PCA on sampled data sets.
- The `analysis` module includes an implementation of novel metrics built atop kernel regression for assessing quality of low-dimensional manifolds.

`PCAfold` is designed to handle multivariate data set(s) supplied by the user. The user can follow a complete workflow using all three modules in series as presented in Fig. 1. However, each module can be used as a standalone toolbox as well. Modules can also be used with other software packages. One of the strengths of `PCAfold` is that it is designed as a toolbox that provides many functionalities that give the user broad flexibility to compose workflows tailored to a particular problem. Many of the functionalities require calling a single function and few larger functionalities such as data sampling and performing PCA introduce a class for easier handling of the available methods. Plotting functions relevant to each module aid in the analysis of data sets and speed up results generation.

2.2. Software functionalities and sample code snippets

In this section we present the functionalities of each module. The format for the user-supplied input data matrix $\mathbf{X} \in \mathbb{R}^{N \times Q}$ common to all modules is that N observations are stored in rows and Q variables are stored in columns. The initial dimensionality of the data set is determined by the number of variables Q. Typically, $N \gg Q$.

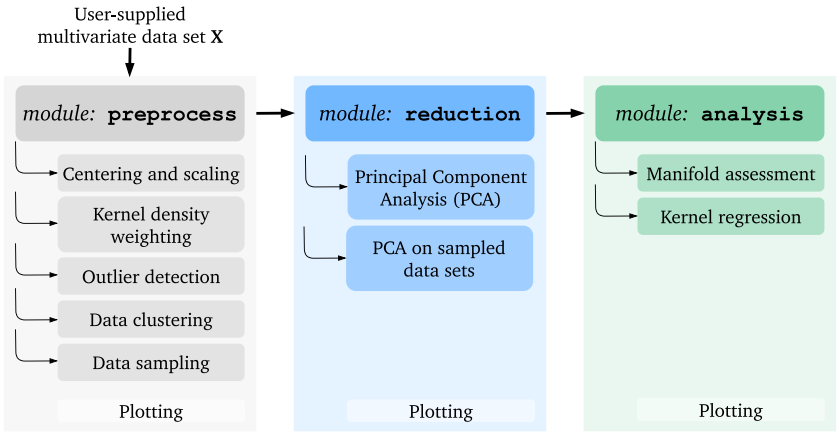


Fig. 1. Software architecture and overview for combining `PCAfold` modules into a complete workflow.

2.2.1. Data preprocessing

The `preprocess` module contains data preprocessing functionalities. Data centering and scaling and inversion of that operation can be performed with `center_scale` and `invert_center_scale` functions respectively. Multivariate outliers can be detected using the `outlier_detection` function which returns indices of the outlying observations. Kernel density weighting of a data set can be performed by instantiating a `KernelDensity` object. Weights can be computed based on a single or multiple conditioning variables. Several clustering functions (all of which return a vector of cluster classifications `idx`) that rely on binning a single variable vector are implemented. This includes equal bins, user pre-defined bins or separating close-to-zero observations. With the `DataSampler` class, the user can sample observations from clusters. Sampling options such as: (1) fixed number of samples from clusters, (2) percentage of samples from clusters, (3) manual sampling from clusters based on a sampling dictionary supplied and (4) random sampling are implemented. As long as a vector of cluster classifications `idx` is known, sampling can be performed in the following way:

```
from PCAfold import DataSampler
import numpy as np

# Generate a dummy data set:
X = np.random.rand(1000,3)

# Generate a dummy vector of cluster classifications:
idx = np.zeros((1000,))
idx[500:800] = 1

# Instantiate DataSampler class object:
selection = DataSampler(idx, idx_test=[], random_seed=100, verbose=True)

# (1) Select equal number of samples from each cluster:
(idx_train, idx_test) = selection.number(20, test_selection_option=1)

# (2) Select equal percentage of samples from each cluster:
(idx_train, idx_test) = selection.percentage(20, test_selection_option=1)

# (3) Select samples manually from each cluster:
(idx_train, idx_test) = selection.manual({0:200, 1:100},
sampling_type='number', test_selection_option=1)

# (4) Select samples at random from each cluster:
(idx_train, idx_test) = selection.random(20, test_selection_option=1)
```

Sampling functions return indices of train and test data samples. Depending on the option selected, the user has a choice for how the test samples accompanying the training data are selected — either as all remaining samples or as a meaningful subset of those. Keeping user pre-defined test samples is also allowed, whenever such specific test set exists within the data set. In that case, the user can specify the `idx_test` parameter *a priori* when instantiating a `DataSampler` object. Train samples will then be selected ignoring the indices from the pre-defined `idx_test`. The data set can be partitioned into train and test data:

```
# Partition the original observations into train and test data:
X_train = X[idx_train,:]
X_test = X[idx_test,:]
```

2.2.2. Data reduction

The `reduction` module introduces Principal Component Analysis as a dimensionality reduction technique. Given a data set $\mathbf{X}$, PCA can be performed by instantiating a `PCA` object. Two techniques for performing PCA are included: (1) eigendecomposition of the covariance matrix and (2) Singular Value Decomposition (SVD) of the preprocessed data set. The user can specify the scaling method to preprocess the data and the number of Principal Components to return. Below is an example of PCA performed on a dummy data set:

```
from PCAfold import PCA
import numpy as np

# Generate a dummy data set:
X = np.random.rand(100,10)

# Instantiate PCA class object:
pca_X = PCA(X, scaling='auto', n_components=2)

# Eigenvectors:
eigenvectors = pca_X.A

# Eigenvalues:
eigenvalues = pca_X.L

# Principal Components:
principal_components = pca_X.transform(X)

# Reconstruct the data set from the first two Principal Components:
X_rec = pca_X.reconstruct(principal_components)
```

Basic plotting of PCA results can be easily achieved with functions available in this module:

- `plot_2d_manifold` plots a two-dimensional manifold
- `plot_parity` plots a parity plot for a reconstructed variable
- `plot_eigenvectors` plots weights on eigenvectors and `plot_eigenvectors_comparison` compares weights on eigenvectors from several `PCA` class objects

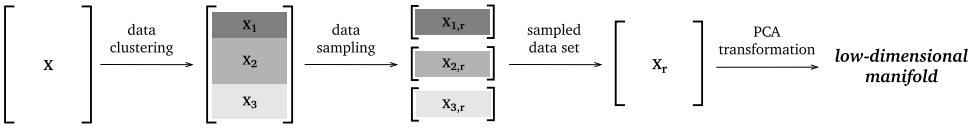


Fig. 2. Overview for performing PCA on a sampled data set $\mathbf{X_r}$.

- `plot_eigenvalue_distribution` plots an eigenvalue distribution and `plot_eigenvalue_distribution_comparison` compares eigenvalue distributions from several `PCA` class objects
- `plot_cumulative_variance` plots eigenvalues and their cumulative sum for visualizing the percent of variance explained by each Principal Component individually and cumulatively

The `reduction` module can be combined with clustering and sampling functionalities from the `preprocess` module to perform PCA transformation on adequately sampled data sets $\mathbf{X_r}$ instead of the full original data set $\mathbf{X}$. The roadmap for such analysis is schematically presented in Fig. 2.

The steps for standard PCA performed on the original data set $\mathbf{X}$ begin with centering and scaling the data:

$$\mathbf{X_{cs}} = (\mathbf{X} - \mathbf{C}) \cdot \mathbf{D}^{-1} \tag{1}$$

where $\mathbf{C}$ and $\mathbf{D}$ are centers and scales computed on the data set $\mathbf{X}$. The next step is the eigendecomposition of the covariance matrix $\mathbf{S}$[1]:

$$\mathbf{S} = \frac{1}{N-1}\mathbf{X_{cs}}^{\mathrm{T}}\mathbf{X_{cs}} \xrightarrow{\text{eigendecomposition}} \mathbf{A}, \mathbf{L} \tag{2}$$

where N is the number of observations in $\mathbf{X}$, $\mathbf{A}$ are the eigenvectors and $\mathbf{L}$ are the eigenvalues obtained. The last step is performing basis transformation:

$$\mathbf{Z} = \mathbf{X_{cs}}\mathbf{A} \tag{3}$$

where $\mathbf{Z}$ is a matrix of Principal Components (PCs). A truncation to q first PCs defines the q-dimensional manifold. The number of retained PCs (q) can be controlled by the `n_components` parameter wherever needed. In contrast, there are several ways in which PCA can be informed (*biased*) by the sampled data set $\mathbf{X_r}$. Four such *biasing* options are implemented in the `PCAfold` software and the reader is referred to the software documentation for more detailed information.[2] Options can be selected using the `biasing_option` parameter wherever needed. For a sampled data set $\mathbf{X_r}$ we obtain new centers $\mathbf{C_r}$ and scales $\mathbf{D_r}$. The new matrices of eigenvectors $\mathbf{A_r}$ and of PCs $\mathbf{Z_r}$ are obtained with one of the four options. As can be expected, the new PCs defined by $\mathbf{Z_r}$ will be different from $\mathbf{Z}$. The q-dimensional *biased* manifold can now be defined using the q first PCs in $\mathbf{Z_r}$. With the choice of the clustering and sampling technique the user can control the level and type of *bias* imposed on the manifold. For instance, selecting equal number of samples from each cluster will balance the identified features, whenever such balance is desired. The user can for instance perform under-representation of clusters with large number of observations with respect to their initial sample density in $\mathbf{X}$. Several functions are then implemented to enable analysis of PCA performed on sampled data sets, many of which result in generating plots. Those include:

- `analyze_centers_change`: compares normalized $\mathbf{C_r}$ with $\mathbf{C}$
- `analyze_eigenvector_weights_change`: compares $\mathbf{A_r}$ with $\mathbf{A}$
- `analyze_eigenvalue_distribution`: compares the eigenvalue distribution for PCA performed on $\mathbf{X}$ versus with one of the four *biasing* options implemented

Finally, it is worth noting that local PCA (i.e., PCA in local clusters of data) can be easily performed by the user by combining clustering functionalities with the `PCA` class. Local eigenvectors and PCs can be obtained by passing an extracted cluster instead of a full data set as an input $\mathbf{X}$ for `PCA` class.[3]

2.2.3. Manifold analysis

The `analysis` module includes functions for quantitative assessments of manifold dimensionality and for comparing manifold parameterizations according to scales of variation and uniqueness of dependent variable values. In particular, a metric of normalized variance $\mathcal{N}(\sigma)$, described in [10], that can characterize the topology of the manifold is expressed as:

$$\mathcal{N}(\sigma) = \frac{\sum_{i=1}^{N}(y_i - \mathcal{K}(\hat{x}_i; \sigma))^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{4}$$

for any dependent variable y_i, where $\bar{y}$ is the average quantity over the whole manifold and $\mathcal{K}(\hat{x}_i; \sigma)$ is the weighted average quantity calculated using Nadaraya–Watson kernel regression [24] with a Gaussian kernel of bandwidth σ centered around the ith observation in normalized coordinates $\hat{x}_i$. Independent variables x_i are centered and scaled by default to reside inside a unit box (resulting in $\hat{x}_i$) so that the bandwidths apply equally to each dimension. $\mathcal{N}(\sigma)$ is computed for each bandwidth using the `compute_normalized_variance` function in an array of bandwidth values σ that are either computed according to interpoint distances or specified directly through the `bandwidth_values` parameter. This function returns a `VarianceData` object from which computed quantities can be accessed as attributes of the class.

The normalized variance for a dependent variable over a manifold without directly overlapping observations will approach zero as the bandwidth approaches zero and will smoothly approach one as the bandwidth increases to encompass the manifold. Larger values of bandwidth at which this rise occurs indicate larger scales of variation in the dependent variable which should better facilitate regression over the manifold. Any sharp, discontinuous derivatives in the dependent variable, which can indicate the manifold is folded over itself, show up as additional humps in the rise of normalized variance over bandwidth as smaller scales of overlap are found among the larger scales of non-overlapping variation. These features can therefore be analyzed for various PCA-derived manifolds in order to indicate which methods give rise to improved representations of dependent variables.

[1] This is the default option for performing PCA implemented in the `PCA` class. Optionally, the user can set `use_eigendec=False` and PCA will be performed through Singular Value Decomposition (SVD) of the preprocessed data matrix.

[2] It can be found under **User Guide → Data reduction → Biasing options**.

[3] An example can be found in the software documentation under **Tutorials & Demos → Global and local PCA**.

3. Illustrative example

To present how the major functionalities from all three modules can be incorporated into one workflow we analyze a data set describing combustion of syngas in air. The data set **X** has 50,000 observations (rows) and 11 variables (columns) and was generated from a steady laminar flamelet model using Spitfire software [30]. The original variables are temperature and 10 chemical species mass fractions from the chemical mechanism by Hawkes et al. [31]. There is also a complementary data set $\mathbf{S_X}$ representing source terms of the original variables, whose columns correspond to heat release rate and mass production rate of chemical species. The data set in the `.csv` format can be imported from the `docs/tutorials` directory in the software repository:

```
import numpy as np

# Original variables:
X = np.genfromtxt('docs/tutorials/data-state-space.csv', delimiter=',')

# List of names of the original variables:
variables_names = ['$T$', '$H_2$', '$O_2$', '$O$', '$OH$', '$H_2O$', '$H$', '$HO_2$', '$CO$', '$CO_2$', '$HCO$']

# Corresponding source terms of the original variables:
S_X = np.genfromtxt('docs/tutorials/data-state-space-sources.csv', delimiter=',')
```

It is worth noting that both **X** and $\mathbf{S_X}$ can be transformed to a new basis identified by PCA. We will refer to **X** in the new space as Principal Components (PCs) and to $\mathbf{S_X}$ in the new space as sources of PCs [23]. In this example, we will perform PCA on the original data set and on its subset, sampled at equal amounts from local clusters (as schematically presented in Fig. 2). We will thus obtain two different manifolds — an initial one and a *biased* one. We will then assess the topology of both manifolds quantitatively using metrics from the `analysis` module.

3.1. PCA on a sampled data set

We begin by performing a PCA transformation of the original data set **X** and the corresponding $\mathbf{S_X}$. Next, we cluster the data set into four clusters based on binning the first PC source vector using the `preprocess.zero_neighborhood_bins` function. This function bins observations into clusters by enforcing that at least one or two clusters are created for observations within a specified offset from zero, which corresponds to a steady state for sources. The number of samples available within each cluster can be obtained using `preprocess.get_populations`.

```
from PCAfold import preprocess
from PCAfold import reduction
from PCAfold import analysis

# Perform PCA on the data set:
pca_X = reduction.PCA(X, scaling='auto', n_components=2)

# Transform original variables to the PC space:
Z = pca_X.transform(X, nocenter=False)

# Transform sources of the original variables to the PC space:
S_Z = pca_X.transform(S_X, nocenter=True)

# Cluster the data set:
idx = preprocess.zero_neighborhood_bins(S_Z[:,0], k=4, zero_offset_percentage=2, split_at_zero=True, verbose=True)

# Compute populations of each cluster:
populations = preprocess.get_populations(idx)
```

In this example, the smallest cluster has 2416 observations and we will select 2400 samples from each cluster manually using `DataSampler.manual`:

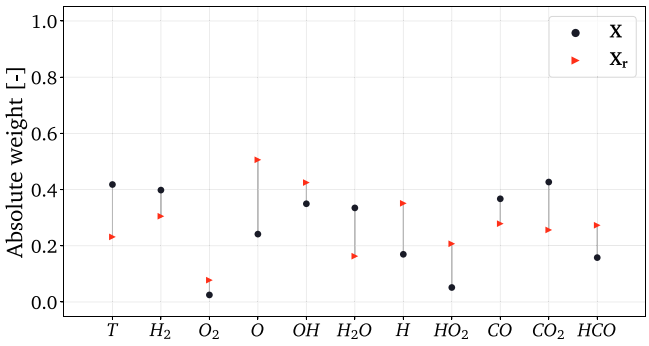


Fig. 3. Weights change on the first eigenvector when performing PCA on the original data set **X** (black dots) and on the sampled data set $\mathbf{X_r}$ (red triangles). Each weight corresponds to one of the original variables in **X**. Plotted using the `reduction.analyze_eigenvector_weights_change` function.

```
# Instantiate DataSampler class object:
sample = preprocess.DataSampler(idx, idx_test=[], random_seed=100, verbose=True)

# Select 2400 samples from each cluster:
(idx_manual, _) = sample.manual({0:2400, 1:2400, 2:2400, 3:2400}, sampling_type='number', test_selection_option=1)
```

When `verbose=True`, additional information on sampling is printed. This feature can be useful for an initial inspection of the current sampling density. PCA can now be performed on a reduced data set sampled from **X** using `reduction.pca_on_sampled_data_set`. This function will sample the data set according to indices in `idx_manual` and will perform PCA with one of the available *biasing* options. In this example, we select `biasing_option=2`:

```
# Perform PCA on a sampled data set according to biasing option 2:
(eigenvalues, eigenvectors, Z_r, S_Z_r, C, D, C_r, D_r) = reduction.pca_on_sampled_data_set(X, idx_manual, scaling='auto', n_components=2, biasing_option=2, X_source=S_X)
```

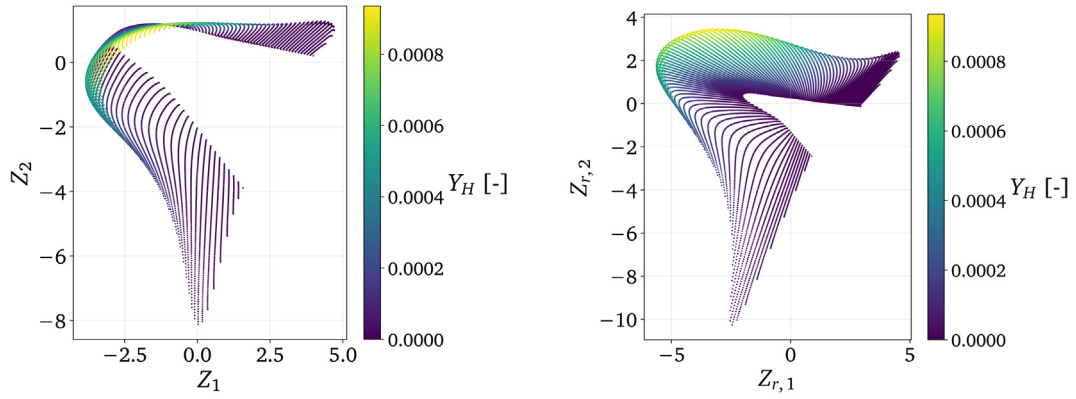
This function primarily returns the new eigenvalues, eigenvectors, PCs and (optionally) sources of PCs. Several plotting functions are then implemented to help the user analyze the effect of changing sampling densities on the low-dimensional representation of the original data set:

```
# Plot the weights change on the first eigenvector:
plt = reduction.analyze_eigenvector_weights_change(np.vstack((pca_X.A[:,0], eigenvectors[:,0])).T, variables_names, legend_label=['$\mathbf{X}$', '$\mathbf{X_r}$'], save_filename='eigenvector-1.pdf')

# Plot the initial two-dimensional manifold:
plt = reduction.plot_2d_manifold(Z[:,0], Z[:,1], color_variable=X[:,6], x_label='$Z_{1}$', y_label='$Z_{2}$', colorbar_label='$Y_H$ [-]', save_filename='Z.pdf')

# Plot the biased two-dimensional manifold:
plt = reduction.plot_2d_manifold(Z_r[:,0], Z_r[:,1], color_variable=X[:,6], x_label='$Z_{r, 1}$', y_label='$Z_{r, 2}$', colorbar_label='$Y_H$ [-]', save_filename='Z_r.pdf')
```

Specifically, the change in eigenvectors can be compared as presented in Fig. 3. Such comparison can be useful since weights on the eigenvector represent the contribution of each original variable in the corresponding PC. The obtained two-dimensional manifolds can be viewed as well and Fig. 4 presents a comparison between the two-dimensional manifold defined by **Z** (the original PCs) and by $\mathbf{Z_r}$ (the *biased* PCs). It is worth noting that after



(a) Initial manifold defined by **Z**. (b) *Biased* manifold defined by $\mathbf{Z_r}$.

Fig. 4. Comparison of the two-dimensional PCA-derived manifolds obtained by transforming (a) the original data set **X** and (b) the sampled data set $\mathbf{X_r}$. Plotted using the `reduction.plot_2d_manifold` function. Manifolds are additionally colored by one of the original variables (mass fraction of species H) which can be specified as an additional input parameter `color_variable`. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

changing sample densities and performing PCA on a subset of the original data set, the shape of the low-dimensional manifold changed substantially. Notably, the region where the manifold was severely folded over itself (top left corner of Fig. 4a) has unfolded in Fig. 4b. This can have useful implications for modeling using PCs. For instance, a non-linear regression technique might struggle to regress a dependent variable in the overlapped region of Fig. 4a compared to regressing over the manifold presented in Fig. 4b.

3.2. Manifold quality assessment

In this section we continue the example by quantitatively assessing the two manifolds presented in Fig. 4 using the functionalities of the `analysis` module. We use the `analysis.compute_normalized_variance` function and compute $\mathcal{N}(\sigma)$ quantities as per Eq. (4). We define an array of 25 bandwidth values σ ranging from $10^{-3.5}$ to $10^{0.5}$. The independent variables are two PCs that define the manifold: (Z_1, Z_2) for the initial manifold and $(Z_{r,1}, Z_{r,2})$ for the *biased* manifold. The dependent variables of interest we select for assessing the manifolds include the mass fraction of species H and the first two sources of PCs tied to each manifold: (S_{Z_1}, S_{Z_2}) for the initial manifold and $(S_{Z_{r,1}}, S_{Z_{r,2}})$ for the *biased* manifold.

```
# Bandwidth values:
bandwidth_values = np.logspace(-3.5, 0.5, 25)

# Create dependent variables matrices:
depvars_initial = np.hstack((S_Z, X[:,[6]]))
depvars_biased = np.hstack((S_Z_r, X[:,[6]]))

# Create names for dependent variables:
depvar_names_initial = ['$S_{Z_1}$', '$S_{Z_2}$', '$Y_H$']
depvar_names_biased = ['$S_{Z_{r, 1}}$', '$S_{Z_{r, 2}}$', '$Y_H$']

# Compute normalized variance quantities on the initial manifold:
variance_data_initial = analysis.compute_normalized_variance(Z, depvars_initial, depvar_names=depvar_names_initial, bandwidth_values=bandwidth_values)

# Compute normalized variance quantities on the biased manifold:
variance_data_biased = analysis.compute_normalized_variance(Z_r, depvars_biased, depvar_names=depvar_names_biased, bandwidth_values=bandwidth_values)

# Plot the comparison of normalized variance quantities:
plt = analysis.plot_normalized_variance_comparison((variance_data_initial, variance_data_biased), ([], []), ('Greys', 'Reds'), save_filename='N.pdf')
```

Fig. 5 presents the computed quantities $\mathcal{N}(\sigma)$ for the selected dependent variables. Quantities in gray-scale correspond to the initial manifold defined by **Z** and presented in Fig. 4a. Quantities in red-scale correspond to the *biased* manifold defined by $\mathbf{Z_r}$ and presented in Fig. 4b. Firstly, we note that $\mathcal{N}(\sigma)$ rises at much smaller bandwidths for the **Z**-defined manifold compared to the *biased* manifold, which conveys the much smaller scales of variation present in the variables of interest, especially for S_{Z_1} and Y_H. In addition, the multiple humps occurring during the rise of normalized variance for S_{Z_1} and Y_H on the **Z**-defined manifold indicate multiple scales of variation: scales of overlap that are orders of magnitude smaller than the non-overlapping scales of variation. From the analysis of the data presented in Fig. 5, we would conclude that the *biased* manifold defined by $\mathbf{Z_r}$ offers a superior parameterization of the variables of interest compared to the **Z**-defined manifold and should be easier to model. This is consistent with our visual comparison of manifolds in Fig. 4. Looking at Fig. 4b, we see that the region with the highest values of Y_H is spread over a much larger portion of the manifold and does not overlap with states close to zero as happens in Fig. 4a. Therefore, the normalized variance for Y_H has a single rise occurring at a much larger bandwidth for the *biased* manifold than the initial manifold, which has variation in Y_H at smaller scales in its region of overlap.

For more illustrative examples the reader is referred to **Tutorials & Demos** section of the software documentation.

4. Impact

Although `PCAfold` was initially developed for application on data sets describing reactive flows, it can also be used in other domains since manifold methods are of broad interdisciplinary interest. The current version of the code can be used for data sets coming from any research domain, as long as the data set complies with the structure mentioned in Section 2.2. This motivates the decision to make the software available to the research community.

Many research areas exploit the fact that a multi-dimensional problem can be effectively re-formulated in lower-dimensional spaces. `PCAfold` software can be particularly useful for analyzing the effect of data preprocessing, prior to applying a reduction

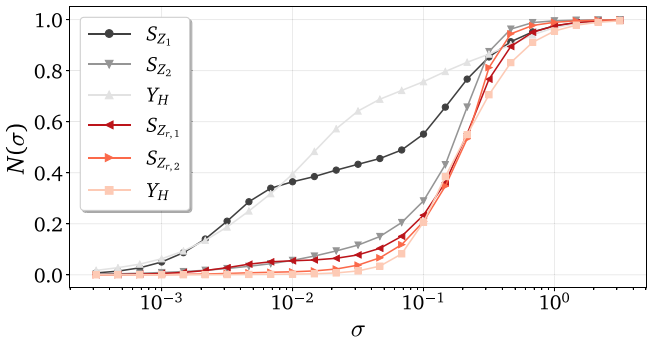


Fig. 5. Normalized variance $\mathcal{N}(\sigma)$ computed on the initial manifold from Fig. 4a (gray-scale) and on the biased manifold from Fig. 4b (red-scale). S_{Z_1} and S_{Z_2} are the first two sources of the initial PCs and $S_{Z_{r,1}}$ and $S_{Z_{r,2}}$ are the first two sources of the *biased* PCs. Y_H is the mass fraction of species H. Plotted using the `analysis.plot_normalized_variance` function. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

technique, on the quality of the low-dimensional representation obtained. It is worth noting that the manifold assessment metrics available can be equally applied to manifolds derived by means of techniques other than PCA. This can be appealing to researchers using techniques such as Locally Linear Embedding (LLE) or Autoencoders, which have also exhibited growing popularity in recent years. `PCAfold` can easily bring tools from outside of its scope into the workflow whenever needed. As an example, several basic data clustering techniques were introduced here, but the user can incorporate other clustering algorithms (such as K-Means algorithm [32]) and, for instance, combine those with the sampling functionalities available. In addition to that, the interest in data sampling in the machine learning community makes the code appealing for use in conjunction with Artificial Neural Networks (ANN) or for general statistical data analysis. The authors have developed a thorough documentation of the `PCAfold` software which includes relevant equations and references for implemented techniques and sample code snippets listed under every functionality documented. Numerous illustrative tutorials are included in the software documentation and the corresponding Jupyter notebooks can be found in the `docs/tutorials` directory. This also makes the current version of the software appealing for students or researchers new to the field of data analysis.

Parts of the `PCAfold` library have been translated and collected from other scripts developed over the years by our research groups at the University of Utah and Université Libre de Bruxelles. Previous scripts have been widely used in combustion research. As an example, this included PCA-based modeling of multi-component reactive flows [33,34] or analysis of PCA-derived combustion manifolds sensitivity to data preprocessing [7]. Recently, `PCAfold`'s workflow is used in our groups to improve the topology of low-dimensional manifolds that exist in turbulent reacting flows to aid in combustion simulations. `PCAfold` has also been used in [10] to demonstrate the application of the normalized variance metric on combustion data sets coming from simulation and experiment. Since PCA is used extensively in our research groups, the software will continue to be maintained and updated with new techniques.

5. Conclusions

In this paper we argue that PCA-derived low-dimensional manifolds are often not unique and can be altered by modifying the original data set on which PCA is performed. Tuning Principal Components can have certain advantages such as minimizing the amount of overlap present in the manifold. We thus propose `PCAfold`: a software package that brings together tools to generate, assess and improve representations of multivariate data sets in lower-dimensional spaces. The software introduces two main functionalities: (1) it implements techniques for performing PCA on sampled data sets which allow for tuning the obtained manifold and (2) it provides novel measures for assessing topology of low-dimensional manifolds. Apart from that, `PCAfold` contains a variety of standard data preprocessing tools and plotting functions which allow for a broad range of data analysis that can be performed. The proposed software is multi-disciplinary and will be a valuable tool for researchers using low-dimensional manifold methods.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research of the first author is supported by the F.R.S.-FNRS Aspirant Research Fellow grant.

The authors gratefully acknowledge the support of Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement no. 714605.

References

[1] Pope SB. Small scales, many species and the manifold challenges of turbulent combustion. Proc Combust Inst 2013;34(1):1–31. http://dx.doi.org/10.1016/j.proci.2012.09.009.

[2] Han Y, Xu Z, Ma Z, Huang Z. Image classification with manifold learning for out-of-sample data. Signal Process. 2013;93(8):2169–77. http://dx.doi.org/10.1016/j.sigpro.2012.05.036.

[3] Fieseler C, Zimmer M, Kutz JN. Unsupervised learning of control signals and their encodings in *C. elegans* whole-brain recordings. 2020, arXiv preprint URL https://arxiv.org/abs/2001.08346.

[4] Ehlert A, Nayeri CN, Morzyński M, Noack BR. Locally linear embedding for transient cylinder wakes. 2019, arXiv preprint URL https://arxiv.org/abs/1906.07822.

[5] Yang Y, Pope SB, Chen JH. Empirical low-dimensional manifolds in composition space. Combust Flame 2013;160(10):1967–80. http://dx.doi.org/10.1016/j.combustflame.2013.04.006.

[6] Parente A, Sutherland J, Tognotti L, Smith P. Identification of low-dimensional manifolds in turbulent flames. Proc Combust Inst 2009;32(1):1579–86. http://dx.doi.org/10.1016/j.proci.2008.06.177.

[7] Parente A, Sutherland JC. Principal component analysis of turbulent combustion data: Data pre-processing and manifold sensitivity. Combust Flame 2013;160(2):340–50. http://dx.doi.org/10.1016/j.combustflame.2012.09.016.

[8] May R, Maier H, Dandy G. Data splitting for artificial neural networks using SOM-based stratified sampling. Neural Netw 2010;23(2):283–94. http://dx.doi.org/10.1016/j.neunet.2009.11.009.

[9] Buda M, Maki A, Mazurowski MA. A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw 2018;106:249–59. http://dx.doi.org/10.1016/j.neunet.2018.07.011.

[10] Armstrong E, Sutherland JC. A technique for characterizing feature size and quality of manifolds. Combust Theory Model 2020. Manuscript submitted for publication.

[11] Keun HC, Ebbels TM, Antti H, Bollard ME, Beckonert O, Holmes E, et al. Improved analysis of multivariate data by variable stability scaling: application to NMR-based metabolic profiling. Anal Chim Acta 2003;490(1):265–76. http://dx.doi.org/10.1016/S0003-2670(03)00094-1.

[12] Noda I. Scaling techniques to enhance two-dimensional correlation spectra. J Mol Struct 2008;883–884:216–27. http://dx.doi.org/10.1016/j.molstruc.2007.12.026.
[13] Keenan MR, Kotula PG. Accounting for Poisson noise in the multivariate analysis of tof-SIMS spectrum images. Surf Interface Anal 2004;36(3):203–12. http://dx.doi.org/10.1002/sia.1657, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sia.1657.
[14] Shyu M-L, Chen S-C, Sarinnapakorn K, Chang L. A novel anomaly detection scheme based on principal component classifier. Tech. rep., Coral Gables, FL, USA: Department of Electrical and Computer Engineering, University of Miami; 2003.
[15] Coussement A, Gicquel O, Parente A. Kernel density weighted principal component analysis of combustion processes. Combust Flame 2012;159(9):2844–55. http://dx.doi.org/10.1016/j.combustflame.2012.04.004.
[16] Gill AA, Smith GD, Bagnall AJ. Improving decision tree performance through induction- and cluster-based stratified sampling. In: Yang ZR, Yin H, Everson RM, editors. Intelligent Data Engineering and Automated Learning – IDEAL 2004. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004, p. 339–44. http://dx.doi.org/10.1007/978-3-540-28651-6_50.
[17] Jolliffe I. Principal component analysis. New York: Springer Verlag; 2002, http://dx.doi.org/10.1007/b98835.
[18] Jolliffe IT. Discarding variables in a principal component analysis. I: Artificial data. J R Stat Soc Ser C Appl Stat 1972;21(2):160–73. http://dx.doi.org/10.2307/2346488.
[19] Krzanowski WJ. Selection of variables to preserve multivariate data structure, using principal components. J R Stat Soc Ser C Appl Stat 1987;36(1):22–33. http://dx.doi.org/10.2307/2347842.
[20] Kaiser HF. The application of electronic computers to factor analysis. Educ Psychol Meas 1960;20(1):141–51. http://dx.doi.org/10.1177/001316446002000116.
[21] Frontier S. Étude de la décroissance des valeurs propres dans une analyse en composantes principales: Comparaison avec le modèle du bâton brisé. J Exp Mar Biol Ecol 1976;25(1):67–75. http://dx.doi.org/10.1016/0022-0981(76)90076-9.
[22] Isaac BJ, Coussement A, Gicquel O, Smith PJ, Parente A. Reduced-order PCA models for chemical reacting flows. Combust Flame 2014;161(11):2785–800. http://dx.doi.org/10.1016/j.combustflame.2014.05.011.
[23] Sutherland JC, Parente A. Combustion modeling using principal component analysis. Proc Combust Inst 2009;32(1):1563–70. http://dx.doi.org/10.1016/j.proci.2008.06.147.
[24] Härdle W. Applied nonparametric regression. In: Econometric Society Monographs, Cambridge University Press; 1990, http://dx.doi.org/10.1017/CCOL0521382483.
[25] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in python. J Mach Learn Res 2011;12(85):2825–30, URL http://jmlr.org/papers/v12/pedregosa11a.html.
[26] Ahmadzadeh A, Sinha K, Aydin B, Angryk RA. MVTS-data toolkit: A python package for preprocessing multivariate time series data. SoftwareX 2020;12:100518. http://dx.doi.org/10.1016/j.softx.2020.100518.
[27] D'Alessio G, Cuoci A, Parente A. OpenMORe: A python framework for reduction, clustering and analysis of reacting flow data. SoftwareX 2020. Manuscript submitted for publication. URL https://github.com/burn-research/OpenMORe.
[28] Lemaître G, Nogueira F, Aridas CK. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. J Mach Learn Res 2017;18(17):1–5, URL http://jmlr.org/papers/v18/16-365.html.
[29] Zhao Y, Nasrullah Z, Li Z. PyOD: A python toolbox for scalable outlier detection. J Mach Learn Res 2019;20(96):1–7, URL https://jmlr.org/papers/v20/19-011.html.
[30] Hansen MA. Spitfire. 2020, URL https://github.com/sandialabs/Spitfire.
[31] Hawkes ER, Sankaran R, Sutherland JC, Chen JH. Scalar mixing in direct numerical simulations of temporally evolving plane jet flames with skeletal CO/H2 kinetics. Proc Combust Inst 2007;31(1):1633–40. http://dx.doi.org/10.1016/j.proci.2006.08.079.
[32] MacQueen J. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. 1, University of California Press; 1967, p. 281–97, URL https://projecteuclid.org/euclid.bsmsp/1200512992.
[33] Biglari A, Sutherland JC. A filter-independent model identification technique for turbulent combustion modeling. Combust Flame 2012;159(5):1960–70. http://dx.doi.org/10.1016/j.combustflame.2011.12.024.
[34] Biglari A, Sutherland JC. An a-posteriori evaluation of principal component analysis-based models for turbulent combustion simulations. Combust Flame 2015;162(10):4025–35. http://dx.doi.org/10.1016/j.combustflame.2015.07.042.