

**Neuralnets for Multivariate And Time Series Analys
(NeuMATSA): A User Manual
(Version 5.0.1)**

MATLAB codes for:
Nonlinear principal component analysis
Nonlinear canonical correlation analysis
Nonlinear singular spectrum analysis

William W. Hsieh

Department of Earth and Ocean Sciences,
University of British Columbia,
6339 Stores Road,
Vancouver, B.C. V6T 1Z4, Canada

Copyright © 2008 William W. Hsieh

1 Introduction

If you have already used an older version of the codes, you can skip the Introduction section, and go directly to Section 2, “What’s new in this release?”.

For a set of variables $\{x_i\}$, principal component analysis (PCA) (also called Empirical Orthogonal Function analysis) extracts the eigenmodes of the data covariance matrix. It is used to (i) reduce the dimensionality of the dataset, and (ii) extract features from the dataset. When there are two sets of variables $\{x_i\}$ and $\{y_j\}$, canonical correlation analysis (CCA) finds the modes of maximum correlation between $\{x_i\}$ and $\{y_j\}$, rendering CCA a standard tool for discovering relations between two fields. The PCA method has also been extended to become a multivariate spectral method—the singular spectrum analysis (SSA). See von Storch and Zwiers (1999) for a description of these classical methods.

Neural network (NN) models have been widely used for nonlinear regression and classification (with codes available in many computer packages, e.g. in the MATLAB Neural Network toolbox). Recent developments in neural network modelling have further led to the nonlinear generalization of PCA, CCA and SSA.

This manual for NeuMATSA (Neuralnets for Multivariate And Time Series Analysis) describes the neural network codes developed at UBC for nonlinear PCA (NLPCA), nonlinear CCA (NLCCA), and nonlinear SSA (NLSSA). There are several approaches to nonlinear PCA; here, we follow the Kramer (1991) approach, and the Kirby and Miranda (1996) generalization to closed curves. Options of regularization with weight penalty have been added to these models. The NLCCA model is based on that proposed by Hsieh (2000).

To keep this manual brief, the user is referred to the review paper Hsieh (2004) for details. This and other relevant papers are downloadable from the web site:
<http://www.ocgy.ubc.ca/projects/clim.pred/download.html>.

This manual assumes the reader is already familiar with Hsieh (2004), and is aware of the relatively unstable nature of neural network models because of local minima in the cost function, and the danger of overfitting (i.e. using the great flexibility of NN models to fit to the noise in the data). The codes are written in MATLAB using its Optimization Toolbox, and should be compatible with MATLAB 7, 6.5, and 6.

This manual is organized as follows: Section 3 describes the NLPCA code of Hsieh (2001a), which is based on Kramer (1991). Section 4 describes the ‘NLPCA.cir’ code, i.e. the NLPCA generalized to closed curves by having a circular neuron at the bottleneck, as in Hsieh (2001a), which follows Kirby and Miranda (1996). This code is also used for NLSSA (Hsieh and Wu, 2002). Section 5 describes the NLCCA code of Hsieh (2001b), which is an improved version of Hsieh (2000). A demo problem is set up in each section. The **sans serif** font is used to denote the names of computer commands, files and variables. *Section 6 is on hints and trouble-shooting.*

Whether the nonlinear approach has a significant advantage over the linear approach is highly dependent on the dataset—the nonlinear approach is generally ineffective if the data record is short and noisy, or the underlying relation is essentially linear. Also time-averaging (e.g. daily data to seasonal data) can mask underlying nonlinear relations due to the Central Limit Theorem (Yuval and Hsieh, 2002). Because of local minima in the cost function, an ensemble of optimization runs from random initial weight parameters is needed, with the best run selected as the solution. There is no guarantee that the best run is close to the global minimum.

The computer programs are free software under the terms of the GNU General Public License as published by the Free Software Foundation. Please read the file ‘LICENSE’ in the program directory before using the software.

2 What's new in this release?

Since this is a major upgrade, users should not mix codes from this release with codes from earlier releases. The major improvements are: (1) The appropriate weight penalty parameter(s) is now objectively determined by the codes, in contrast to previous releases. (2) Robust options have been introduced in the codes to deal with noisy datasets containing outliers. Early stopping during model training is no longer implemented. Technically, the nonlinear optimization is now performed by `fminunc.m` (instead of `fminu.m`) in the Matlab Optimization toolbox. [Unfortunately, for a slight improvement in accuracy, `fminunc.m` in Matlab 7 runs at a half to a quarter the speed of that in Matlab 6. Hence for Matlab 7 users, a notable speedup can be achieved if the old M-files `fminunc.m` and `fminsub.m` from Matlab 6 are used.]

In the NLPCA and NLPCA.cir models, to determine the best weight penalty parameter P to use, the codes now check if the solution for a particular weight penalty value undesirably projects neighbouring points to distant parts of the NLPCA curve (i.e. if neighbouring points are assigned very different values of the nonlinear principal component (NLPC)). For each data point \mathbf{x} and its nearest neighbour $\tilde{\mathbf{x}}$, the model assigns u and \tilde{u} as their NLPC. With C denoting the (Pearson) correlation coefficient, Hsieh (2007) defined an index for measuring inconsistency between nearest neighbours in a dataset:

$$I = 1 - C(u, \tilde{u}). \quad (1)$$

When some neighbouring points are projected to distant parts of the NLPCA curve, the difference between u and \tilde{u} becomes large for such pairs, causing C to drop, hence a rise in the inconsistency index I .

Given enough model flexibility (by using enough hidden neurons), the model chooses the most appropriate P by solving the problem multiple times with P ranging from large to small values, and chooses the optimal P yielding the smallest value for the information criterion $H = \text{MSE} \times I$ (or $\text{MAE}^2 \times I$ if the new robust version is used), with MSE denoting the mean square error, and MAE, the mean absolute error. In other words, we do not just pick the solution with the lowest MSE or MAE, but also ensure the solution is reasonably consistent in assigning NLPC to nearest neighbours, thereby eliminating unreasonable zigzag curves as the solution.

NLPCA with more than one bottleneck neuron is now implemented with the addition of a term in the cost function trying to force the bottleneck neurons to be uncorrelated with each other.

In the new NLCCA model, for each ensemble run starting from random initial weights, weight penalty is also randomly chosen from a range of given values. With test data eliminating overfitted solutions, the optimal solution is chosen without the need for the user to choose the weight penalty as in earlier releases.

For increased robustness against outliers in noisy datasets, an option to use MAE instead of MSE is now available in NLPCA and NLPCA.cir, and to use biweight midcorrelation (and MAE) instead of Pearson correlation (and MSE) is available in NLCCA. The advantage of the biweight midcorrelation over the Pearson correlation is shown in Cannon and Hsieh (2008).

3 Nonlinear Principal Component Analysis (NLPCA)

3.1 Files in the directory for NLPCA

The downloaded tar file `NLPCA5.0.tar` can be uncompressed with the Unix `tar` command:
`tar -xf NLPCA5.0.tar`

and a new directory NLPCA5.0 will be created containing the following files:

`setdat.m` sets up the demo problem, produces a demo dataset `xdata` which is stored in the file `data1.mat`. This demo problem is instructive on how NLPCA works. To see the demo dataset, the reader should get onto MATLAB, and type in: `setdat` which also creates an (encapsuled) `postscript` plot file `data1.eps`. The plot displays the theoretical mode, and the data `xdata` (generated from the theoretical mode plus considerable Gaussian noise) (shown as dots). The objective is to use NLPCA to retrieve the underlying theoretical mode from `xdata`.

`nlpca1.m` is the main program for calculating the NLPCA mode 1. (Here `xdata` is loaded). To run NLPCA on the demo dataset on unix, use:

```
matlab < nlpca1.m > out1new &
```

(The ‘&’ allows unix to run the job on the background, as this run could take some time). Compare the new output file `out1new` with the downloaded output file `out1` in this directory; they should be similar. *The user should then use `nlpca1.m` as a template for his/her own main program.*

`plmodel1.m` plots the output mode 1, also saving the plot as an (encapsuled) `postscript` file `plmodel1.eps`. Alternatively, `pltmodel1.m` gives a more compact plot.

`nlpca2.m` is the main program for calculating the NLPCA mode 2, after removing the NLPCA mode 1 from the data. On unix, use: `matlab < nlpca2.m > out2new &`

As the dataset here contains only 1 theoretical mode + noise, there is no meaningful mode 2 for `nlpca2.m` to extract. In the directory `Eg2`, there is a sample problem containing 2 theoretical modes, so you could try `nlpca2.m` there (see Sec. 3.4). The problem of this mode-by-mode extraction process is that the first NLPCA mode could end up extracting a combination of the 2 underlying theoretical modes. An alternative (better?) approach is to use 2 bottleneck neurons in the NN as done in the directory `Nb2`.

`plmode2.m` plots the output mode 2, and writes out the plot on a `postscript` file. Alternatively, `pltmode2.m` gives a more compact plot.

`param.m` contains the default parameters for the problem (some of the parameters are reset in `nlpca1.m`). It contains the parameters for the demo problem. After the demo, *the user will need to put the parameters relevant to his/her problem into this file.*

`out1` is the downloaded output file from the UBC run of `nlpca1.m` for the demo problem.

`mode1.mat` contains the extracted nonlinear mode 1 (and the linear mode 1).

`manual.m` contains the online manual. To see manual while in MATLAB, type in: `help manual`

The remaining files do not in general require modifications by the user:

`chooseTestSet.m` randomly divides the dataset into a training set and a test set (also called a validation set). Different runs in the ensemble will in general have different training datasets and different test sets.

`calcmode.m` computes the NLPCA mode (or PCA mode).

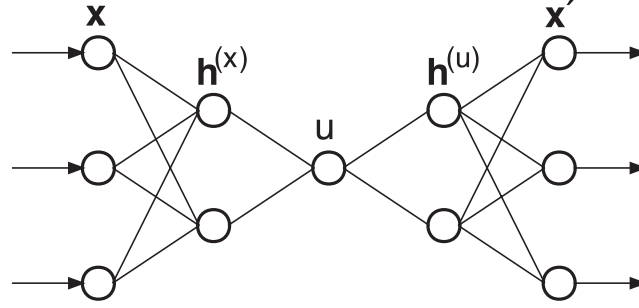


Figure 1: The NN model for calculating nonlinear PCA (NLPCA). There are 3 ‘hidden’ layers of variables or ‘neurons’ (denoted by circles) sandwiched between the input layer \mathbf{x} (with l variables or neurons) on the left and the output layer \mathbf{x}' on the right. (In the code, `xdata` is the input, while `xi` is the output). Next to the input layer is the *encoding* layer (with m neurons), followed by the *bottleneck* layer (shown here with a single neuron u), which is then followed by the *decoding* layer (also with m neurons). A nonlinear function maps from the higher dimension input space to the lower dimension bottleneck space, followed by an inverse transform mapping from the bottleneck space back to the original space represented by the outputs, which are to be as close to the inputs as possible by minimizing the cost function J (i.e. minimizing the MSE or MAE of \mathbf{x}' relative to \mathbf{x}). Data compression is achieved by the bottleneck, with the bottleneck neuron giving u , the nonlinear principal component (NLPC). In the current version, more than 1 neuron is allowed in the bottleneck layer.

`costfn.m` computes the cost function.

`mapu.m` maps from `xdata` to u , the nonlinear PC (i.e. the value of the bottleneck neuron) (Fig. 1).

`invmapx.m` computes the inverse map from u to \mathbf{x} (where \mathbf{x} is the NLPCA approximation to the original `xdata`), then the cost function J .

`nondimen.m` and `dimen.m` are used for scaling and descaling the data. The input are nondimensionalized according to the scaling option selected, NLPCA is then performed, and the final results are again dimensionalized.

`extracalc.m` can be used for projecting new data onto an NLPCA mode, i.e. calculate \mathbf{x}' and u from given \mathbf{x} , or calculate \mathbf{x}' from given u . Also needed by the plotting programs (e.g. `plmode1.m`).

3.2 Parameters stored in `param.m` for NLPCA

Parameters are classified into 3 categories, ‘*’ means the user normally has to change the value to fit the problem. ‘+’ means the user may want to change the value, and ‘−’ means the user rarely has to change the value.

+ `iprint` = 0, 1, 2 or 3 increases the amount of printout.

Hint: A printout summary from an output file (e.g. `out1`) can be extracted by using the unix `grep` command, e.g.

```
grep # out1
grep ## out1
```

`grep ### out1`

where more brevity is attained by using more ‘#’ signs.

- + `Err_norm = 1` or `2` uses either the L1 or the L2 norm to measure the error between the data and the model output, i.e. uses the mean absolute error (MAE) or mean square error (MSE) for the variable `Err`. L1 is a less sensitive norm than L2, hence more robust to outliers in noisy datasets.
- `linear = 0` means NLPCA; `linear = 1` uses only linear transfer functions, thereby reducing to linear PCA. [When `linear = 1`, `m`, the number of hidden neurons in the encoding layer, should be set to equal `nbottle` (the number of hidden neurons in the bottleneck layer), and `testfrac` (see below) be set to 0].
- * `nensemble` is the number of runs in the ensemble. Many runs may converge to shallow local minima and hence wasted. Increase `nensemble` if results appear unstable/irreproducible from serious local minima problem.
- `testfrac = 0` does not reserve a portion of the data for testing or validation, i.e. all data are used for training. Use only when data is not noisy, or a linear solution is wanted.
`testfrac > 0` (usually between 0.1 to 0.2), then a fraction of the data record will be randomly selected as test data.
Note: with `testfrac > 0`, different ensemble runs will work with different training datasets, so there will be more scatter in the solutions of individual ensemble runs than when `testfrac = 0` (where all runs have the same training dataset, as all data are used for training).
- + `segmentlength = 1` if autocorrelation of the data (in time) is not of concern; otherwise the user might want to set `segmentlength` to about the autocorrelation time scale. Data for training and testing are chosen in segments with length equal to `segmentlength`. For instance, if monthly data are available, and the autocorrelation time scale is 7 months, then set `segmentlength = 7`.
- `overfit_tol` is the tolerance for overfitting, should be 0 (recommended) or a small positive number ($\ll 0.1$). Only ensemble runs having `Err` over the test set no worse than (`Err` over the training set) $\times(1+\text{overfit_tol})$ are accepted, and the selected solution is the run with the lowest `Err` over all the data.
 When `overfit_tol = 0`, program automatically chooses an overfit tolerance level to decide which ensemble runs to accept. [The program calculates `overfit_tol2`, the average value of `Errtest - Errtrain` for ensemble runs where `Errtest` is less than `Errtrain`, i.e. error for the test data is less than that for the training data. The program then also accepts ensemble runs where `Errtest` exceeds `Errtrain` by no more than `overfit_tol2`].
- `info_crit = 1` means the final model solution is selected among the accepted runs based on the minimum information criterion `H`. If 0, the run with the minimum `Err` is selected.
- * `xscaling` controls scaling of the x variables before performing NLPCA.
 - = -1, no scaling (not recommended unless variables are of order 1).
 - = 0, scales all the x_i variables by removing the mean and dividing by the standard deviation of each variable.
 - = 1, scales all x_i variables by removing the mean and dividing by the standard deviation of the 1st x variable (i.e. x_1). (Similarly for `xscaling = 2, 3, ...`)

Note: When the x variables are the *principal components* of a dataset and the 1st variable is from the leading mode, `xscaling = 1` is preferred. [If `xscaling = 0`, the importance of the higher PC modes (with weak signal-to-noise ratio) can be greatly exaggerated by the scaling].

For instance, if the x_1 variable ranges from 0 to 3, and x_2 from 950 to 1050, then poor convergence can be expected from setting `xscaling = -1`, while `xscaling = 0` should do much better. However, if the x_i variables are the leading principal components, with x_1 ranging from -9 to 10, x_2 from -3 to 2, and x_3 from -1 to 1, then `xscaling = 0` will greatly exaggerate the importance of the higher modes by causing the standard deviations of the scaled x_1 , x_2 and x_3 variables to be all equal to one, whereas `xscaling = 1` will not cause this problem.

- + `penalty_store` stores a variety of values for the weight penalty parameter used in the cost functions. During ensemble runs with random initial weights, the `penalty` parameter value is also randomly chosen among the values given in `penalty_store`. Increasing `penalty` leads to less nonlinearity (i.e. less wiggles and zigzags when fitting to noisy data). Too large a value for `penalty` leads to linear solutions and trivial solutions. For instance, `penalty_store = [1 0.1 0.01 0.001 0.0001 0]` covers a good range of weight penalty values. It is important to proceed from large to small weight penalty, since for small enough penalty, the program may not find any acceptable solution (due to overfitting) and will (correctly) terminate at that point.
- `maxiter` scales the maximum number of function iterations allowed during optimization. Start with 1, increase if needed.
- + `initRand` = a positive integer, initializes the random number generator, used for generating random initial weights. Because of multiple minima in the cost function, it is a good idea to rerun with a different value of `initRand`, and check that one gets a similar solution.
- `initwt_radius` controls the initial random weight radius for the ensemble. Adjust this parameter if having convergence problem. (Suggest starting with `initwt_radius = 1`). The weight radius is smallest for the first ensemble run, increasing by a factor of 4 when the final ensemble run is reached (so the earliest ensemble runs may tend to be more linear than the later runs).

Input dimensions of the network (Fig. 1):

- * `l` is the number of input variables x_i .
- * `m` is the number of hidden neurons in the encoding layer (i.e. the first hidden layer). For nonlinear solutions, $m \geq 2$ is needed. In the current version, `m` is no longer set in `param.m`, but the main program (`nlpca1.m`) assigns `m` from the values specified by the vector `m_store`. Increase the `m` values in `m_store` if the user expects an NLPCA solution of considerable complexity.
- `nbottle` is the number of hidden neurons in the bottleneck layer. Usually `nbottle = 1`, but `nbottle > 1` is allowed, with the directory `Nb2` containing an example where `nbottle = 2` is used.

Note: The total number of (weight and bias) parameters is $2lm + 4m + l + 1$ (for `nbottle = 1`). In general this number should be less than the number of observations in the dataset (though this is not a strict requirement since increasing weight penalty will automatically reduce the effective number of free parameters). If the number of observations is much less

than the number of parameters, principal component analysis may be needed to pre-filter the dataset, so only the leading principal components are used as input to NLPCA.

3.3 Variables in the codes for NLPCA

The reader can skip this subsection if he/she does not intend to modify the basic code.

Global variables are given by:

```
global iprint Err_norm linear nensemble testfrac segmentlength ...
       overfit_tol info_crit xscaling penalty maxiter initRand ...
       initwt_radius options n l m nbottle iter Jscale xmean xstd ntrain ...
       xtrain utrain xitrain ntest xtest utable xitest Err inearrest I H
```

Be careful not to introduce user variables with the same name as some of the global variables (e.g. n , l , m , I , H).

If you add or remove variables from this global list, it is **CRITICAL** that the same changes are made on **ALL** of the following files:

```
nlpca1.m, nlpca2.m, param.m, calcmode.m, costfn.m, mapu.m, invmapx.m, extracalc.m,
plmode1.m, plmode2.m, pltmode1.m, pltmode2.m.
```

`options` are the options used for the m-file `fminunc.m` in the MATLAB optimization toolbox.

`n` is the number of observations [automatically determined from `xdata`].

`xdata` is an $l \times n$ data matrix, as there are l ' x_i ' variables ($i = 1, \dots, l$), and n observations.

(A common error is to input an $n \times l$ data matrix, i.e. the transpose of the correct matrix, which leads MATLAB to complain about incorrect matrix dimensions).

`iter` is the number of function iterations.

`Jscale` is a scale factor for the cost function J .

`xmean` and `xstd` are the means and standard deviations of the `xdata` variables.

`ntrain`, `ntest` are the number of observations for training and for testing (i.e. validating).

`xtrain`, `xtest` are the portions of `xdata` used for training and testing.

`utrain`, `utable` are the bottleneck neuron values (the NLPC) during training and testing.

[$\text{mean}(\text{utrain}) = 0$ and $\text{utrain} * \text{utrain}' / \text{ntrain} = 1$ are forced on the cost function J .]

`xitrain`, `xitest` are the model predicted values for x during training and testing.

`Err` is the error computed over all data (training+testing), `Err` can be the MAE or the MSE (as specified by `Err_norm`).

`inearest` indicates the location of the nearest neighbour for each data point.

`I` is the inconsistency index I in Eq.(1), measuring the inconsistency between the NLPC of nearest neighbours (see Sec. 2).

H is the holistic information criterion H for selecting the best model (see Sec. 2) among various models built with different values of `penalty` and `m`. ($H = \text{MSE} \times I$ or $\text{MAE}^2 \times I$ depending on whether `Err_norm` = 2 or 1).

All the weights of the NN are stored in the vector `W`. In the main program `nlpca.1`, NLPCA is repeatedly run for a range of `penalty` values as specified by the vector `penalty_store`, and for a range of `m` values as specified by `m_store`. `W` for each pair of `penalty` and `m` values is stored in the cell array `W_store`. Plot programs like `plmodel1.m` retrieves `W` by `W = W_store{ip, im}`, where `ip` and `im` are chosen based on the information criterion H . User can look at other NLPCA solutions by specifying `ip` and `im` in the plot program. For instance, if `penalty_store` = [1 0.1 0.01 0.001 0.0001 0] and `m_store` = [3 4 5], then choosing `ip` = 2 and `im` = 3 would select the NLPCA solution for `penalty` = 0.1 and `m` = 5.

3.4 Extracting more than one NLPCA mode

In the directory `Eg2`, `setdat2.m` produces a dataset containing 2 nonlinear theoretical modes plus a little Gaussian noise. Two bottleneck neurons are used (`nbottle` = 2) to extract the 2 modes simultaneously. The main program `nlpca.m` produced `modes.mat` and `out` (the printout file). Use `plmodes.m` to plot the modes.

The disadvantage of the mode-by-mode extraction approach with a single bottleneck neuron (i.e. extract mode 2 from the residual after the mode 1 solution has been removed) is that the NLPCA mode 1 may be a combination of the 2 theoretical modes.

With `nbottle` = 2, the NLPCA solution is no longer a curve but a curved surface. The 2 bottleneck neurons may be correlated, so one cannot generate a meaningful 1-D curve from the 2-D surface by holding one bottleneck neuron at zero and vary the other. Hence, the cost function J has an extra term added, namely $(\mathbf{u}_1 \cdot \mathbf{u}_2)^2$, where \mathbf{u}_1 and \mathbf{u}_2 are the n -dimensional vectors containing the values of the first and second bottleneck neurons, respectively, thereby forcing the bottleneck neurons to be uncorrelated with each other. This will allow meaningful extraction of two 1-D curves from the 2-D surface by the plotting program `plmodes.m`. When `nbottle` > 1, one needs to use larger `m` (and `nensemble`) than when `nbottle` = 1.

`plmodes.m` generates first the plot file `plmodes.eps` showing the 2-D surface from the NLPCA solution, then the first 1-D curve in `plmodel1.eps` and the second 1-D curve in `plmodel2.eps`. Since these are nonlinear curves, the percent variance explained by each 1-D curve does not in general add up to that explained by the 2-D surface. The corresponding linear modes are then plotted in `pllinmodes.eps`, `pllinmodel1.eps` and `pllinmodel2.eps`.

4 Nonlinear Principal Component Analysis with a circular bottleneck neuron (NLPCA.cir)

The NLPCA model of the previous section (when used with one hidden neuron) extracts an open curve solution, whereas the NLPCA.cir model in this section is capable of extracting a closed curve solution, and is also used for nonlinear singular spectrum analysis (NLSSA).

4.1 Files in the directory for NLPCA.cir

The downloaded tar file `NLPCA.cir.tar` can be uncompressed with the Unix `tar` command:

```
tar -xf NLPCA.cir5.0.tar
```

and a new directory `NLPCA.cir5.0` will be created containing the following files:

`setdat.m` sets up the demo problem, produces a demo dataset `xdata` which is stored in the file `data1.mat`, and creates a `postscript` plot file `data1.eps`. The plot shows the two theoretical modes, and the input data as dots. The user should go through the demo as in Sec.3.

`nlpca1.m` is the main program for calculating the NLPCA.cir mode 1. The user should later use `nlpca1.m` as a template for his/her own main program.

`nlpca2.m` is the main program for calculating the NLPCA.cir mode 2 after removing mode 1 from the data.

`param.m` contains the default parameters for the problem (some of the parameters are reset in `nlpca1.m`). The user will need to put the parameters relevant to his/her problem into this file.

`out1` and `out2` are the output files from running `nlpca1.m` and `nlpca2.m` respectively.

`mode1.mat` contains the extracted nonlinear mode 1 (and the linear mode 1). Similarly for `mode2.mat`.

`plmode1.m` plots the output mode 1, also saving the plot on a `postscript` file `plmode1.eps`. Alternatively, `pltmode1.m` gives a more compact plot.

`plmode2.m` plots the output mode 2. Alternatively, `pltmode2.m` gives a more compact plot.

`manual.m` contains the online manual.

The following files do not in general require modifications by the user:

`chooseTestSet.m` divides the dataset into a training set and a test set.

`calcmode.m` computes the NLPCA.cir mode.

`costfn.m` computes the cost function.

`mappq.m` maps from the input neurons `xdata` to the bottleneck (p, q) (Fig. 2).

`invmapx.m` computes the inverse map from (p, q) to the output neurons `xi` (where `xi` is the model approximation to the original `xdata`), and the cost function J .

`nondimen.m` and `dimen.m` are used for scaling and descaling the data.

`extracalc.m` can be used for projecting new data onto an NLPCA.cir mode, i.e. calculate \mathbf{x}' and p, q from given \mathbf{x} , or calculate \mathbf{x}' from given p and q . Also needed by the plotting programs (e.g. `plmode1.m`).

4.2 Parameters stored in `param.m` for NLPCA.cir

Parameters are classified into 3 categories, ‘*’ means user normally has to change the value to fit the problem. ‘+’ means user may want to change the value, and ‘-’ means user rarely has to change the value.

+ `iprint` = 0, 1, 2 or 3 increases the amount of printout.

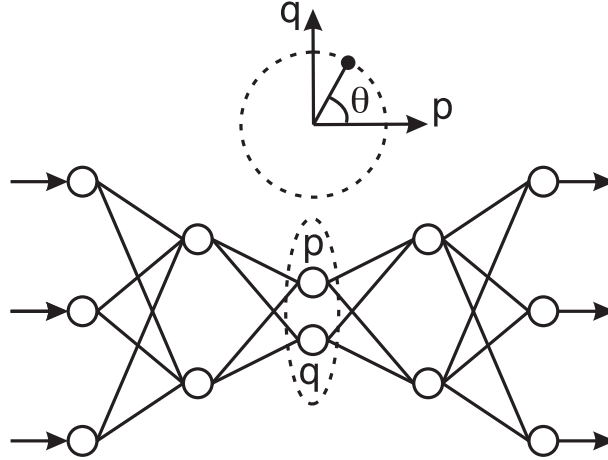


Figure 2: The NLPCA with a ‘circular neuron’ at the bottleneck. Instead of having one bottleneck neuron u as in Fig.1, there are now two neurons p and q constrained to lie on a unit circle in the p - q plane, so there is only one free angular parameter (θ). [This is also the network used for performing NLSSA (Hsieh and Wu, 2002), where the inputs to this network are the leading principal components from the singular spectrum analysis of a dataset].

- + `Err_norm = 1` or `2` uses either the L1 or the L2 norm to measure the error between the data and the model output, i.e. uses the mean absolute error (MAE) or mean square error (MSE) for the variable `Err`. L1 is a less sensitive norm than L2, hence more robust to outliers in noisy datasets.
- `linear = 0` means NLPCA.cir; `linear = 1` reduces to the straight line PCA solution.
- * `nensemble` is the number of runs in the ensemble. (Many runs may converge to shallow local minima and hence wasted).
- `testfrac = 0` does not reserve a portion of the data for testing or validation, i.e. all data are used for training. Use only when data is not noisy, or a linear solution is wanted.
`testfrac > 0` (usually between 0.1 to 0.2), then a fraction of the data record will be randomly selected as test data.
- + `segmentlength = 1` if autocorrelation of the data is not of concern; otherwise the user may want to set `segmentlength` to about the autocorrelation time scale.
- `overfit_tol` is the tolerance for overfitting, should be 0 (recommended) or a small positive number ($\ll 0.1$). Only ensemble runs having `Err` over the test set no worse than (`Err` over the training set) $\times(1+\text{overfit_tol})$ are accepted, and the selected solution is the run with the lowest `Err` over all the data. When `overfit_tol = 0`, program automatically chooses an overfit tolerance level to decide which ensemble runs to accept.
- `info_crit = 1` means the final model solution is selected among the accepted runs based on the minimum information criterion H . If 0, the run with the minimum `Err` is selected (which can overfit).
- * `xscaling` controls scaling of the x variables before performing NLPCA.cir.
`xscaling = -1`, no scaling (not recommended unless variables are of order 1).

`xscaling = 0`, scales all the x_i variables by removing the mean and dividing by the standard deviation of each variable.

`xscaling = 1`, scales all x_i variables by removing the mean and dividing by the standard deviation of the 1st x variable (i.e. x_1). (Similarly for 2, 3, ...)

Note: When the x variables are the *principal components* of a dataset and the 1st variable is from the leading mode, `xscaling = 1` is preferred. [If `xscaling = 0`, the importance of the higher PC modes (with weak signal-to-noise ratio) can be greatly exaggerated by the scaling].

- + `penalty_store` stores a variety of values for the weight penalty parameter used in the cost functions. During ensemble runs with random initial weights, the `penalty` parameter value is also randomly chosen among the values given in `penalty_store`. Increasing `penalty` leads to less nonlinearity (i.e. less wiggles and zigzags when fitting to noisy data). Too large a value for `penalty` leads to linear solutions and trivial solutions. For instance, `penalty_store = [1 0.1 0.01 0.001 0.0001 0]` covers a good range of weight penalty values. It is important to proceed from large to small weight penalty, since for small enough penalty, the program may not find any acceptable solution (due to overfitting) and will (correctly) terminate at that point.
- `maxiter` scales the maximum number of function iterations allowed during optimization. Start with 1, increase if needed.
- + `initRand` = a positive integer, initializes the random number generator, used for generating random initial weights.
- `initwt_radius` controls the initial random weight radius for the ensemble. Adjust this parameter if having convergence problem. (Suggest starting with `initwt_radius = 1`).

In older releases, there was an option (`zeromeanpq = 1`) for forcing the bottleneck neurons p and q to have zero mean. This option is no longer available as it is not entirely consistent with the current use of an information criterion to select the right model.

Input dimensions of the networks:

* l is the number of input variables x_i .

* m is the number of hidden neurons in the encoding layer (i.e. the first hidden layer). For nonlinear solutions, $m \geq 2$ is needed. In the current version, m is no longer set in `param.m`, but the main program (`nlpca1.m`) assigns m from the values specified by the vector `m_store`. Increase the m values in `m_store` if the user expects an NLPCA solution of considerable complexity.

Note: The total number of free (weight and bias) parameters is $2lm + 6m + l + 2$. In general this number should be less than the number of observations in the dataset (though this is not a strict requirement since increasing weight penalty will automatically reduce the effective number of free parameters). If this is not the case, principal component analysis may be needed to pre-filter the dataset, so only the leading principal components are used as input to NLPCA.cir.

4.3 Variables in the codes for NLPCA.cir

The reader can skip this subsection if he/she does not intend to modify the basic code.

Global variables are given by:

```
global iprint Err_norm linear nensemble testfrac segmentlength ...
    overfit_tol info_crit xscaling penalty maxiter initRand ...
    initwt_radius options n l m iter Jscale xmean xstd ntrain xtrain ...
    ptrain qtrain xitrain ntest xtest ptest qtest xitest Err inearrest I H
```

Be careful not to introduce user variables with the same name as some of the global variables (e.g. n , l , m , I , H).

If you add or remove variables from this **global** list, it is **CRITICAL** that the same changes are made on **ALL** of the following files:

```
nlpca1.m, nlpca2.m, param.m, calcmode.m, costfn.m, mapu.m, invmapx.m, extracalc.m,
plmode1.m, plmode2.m, pltmode1.m, pltmode2.m.
```

options are the options used for the m-file `fminunc.m` in the MATLAB optimization toolbox.

n is the number of observations [automatically determined from **xdata**].

xdata is an $l \times n$ data matrix, as there are l ' x_i ' variables, and n observations.

iter is the number of function iterations.

Jscale is a scale factor for the cost function **J**.

xmean and **xstd** are the means and standard deviations of the **xdata** variables.

ntrain, **ntest** are the number of observations for training and for testing.

xtrain, **xtest** are the portions of **xdata** used for training and testing.

ptrain, **ptest** are the bottleneck neuron values (the NLPC) during training and testing. Similarly for **qtrain**, **qtest**. [Note that in this version there is no longer an option to impose $\text{mean}(p) = \text{mean}(q) = 0$ during training].

xitrain, **xitest** are the model predicted values for x during training and testing.

Err is the error computed over all data (training+testing), **Err** can be the MAE or the MSE (as specified by **Err_norm**).

inearrest indicates the location of the nearest neighbour for each data point.

I is the inconsistency index I in Eq.(1), measuring the inconsistency between the NLPC of nearest neighbours (see Sec. 2).

H is the holistic information criterion H for selecting the best model (see Sec. 2) among various models built with different values of **penalty** and **m**. ($H = \text{MSE} \times I$ or $\text{MAE}^2 \times I$ depending on whether **Err_norm** = 2 or 1).

In the main program `nlpca.1`, `NLPCA.cir` is repeatedly run for a range of `penalty` values, as specified by `penalty_store`, and for a range of `m` values, as specified by `m_store`. `W`, containing all the weights for the NN computed for each pair of `penalty` and `m` values, is stored in the cell array `W_store`. Plot programs like `plmodel.m` retrieves `W` by `W = W_store{ip, im}`, where `ip` and `im` are chosen based on the information criterion `H`. User can look at other `NLPCA.cir` solutions by specifying `ip` and `im` in the plot program.

5 Nonlinear Canonical Correlation Analysis (NLCCA)

The downloaded tar file `NLCCA.tar` can be uncompressed with the Unix `tar` command:

```
tar -xf NLCCA5.0.tar
```

and a new directory `NLCCA5.0` will be created containing the following files:

5.1 Files in the directory for NLCCA

`setdat.m` sets up the demo problem, produces two datasets `xdata` and `ydata` which are stored in the file `data1.mat`. It also produces two (encapsuled) postscript plot files `data1x.eps` and `data1y.eps`, with `data1x.eps` containing the `xdata` (shown as dots), and its two underlying theoretical modes, and `data1y.eps` containing the `ydata` and its two theoretical modes. The objective is to use NLCCA to retrieve the two underlying theoretical modes from `xdata` and `ydata`.

`nlcca1.m` is the main program for calculating the NLCCA mode 1. (Here the datasets `xdata` and `ydata` are loaded). The user should use this file as a template for his/her own main program.

`nlcca2.m` is the main program for calculating the NLCCA mode 2 after the NLCCA mode 1 solution has been subtracted from the data. User needs to give the name of the file containing the NLCCA mode 1 solution.

`param.m` contains the default parameters for the problem (some of the parameters are reset in `nlcca1.m`). The user will need to put the parameters relevant to his/her problem into this file.

`out1` and `out2` are the outputs from running `nlcca1` and `nlcca2` respectively.

`model1.mat` contains the linear and nonlinear mode 1, the nonlinear mode having multiple solutions with no. of hidden neurons $l_2 = m_2 = 2, 3, 4$. After subtracting mode 1 from the data, the residual was fed into the NLCCA model to extract the second NLCCA mode—again with the number of hidden neurons $l_2 = m_2 = 2, 3, 4$ yielding the file `model2.mat`.

`plmodel1.m` plots the output mode 1, producing two postscript plot files (`plmodel1x.eps` and `plmodel1y.eps`), whereas `pltmodel1.m` produces more compact plots onto one postscript file. User can specify which solution to plot out. The default is to plot the best solution based on the overall score (see below).

`plmodel2.m` plots the output mode 2 (producing plot files `plmodel2x.eps` and `plmodel2y.eps`), and so does `pltmodel2.m` in a more compact format.

`manual.m` contains the online manual.

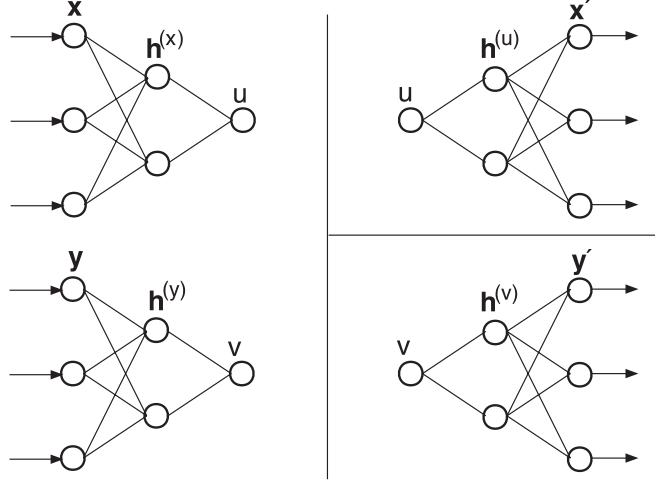


Figure 3: The three NNs used to perform NLCCA. The double-barreled NN on the left maps from the inputs \mathbf{x} and \mathbf{y} to the canonical variates u and v . Starting from the left, there are l_1 input \mathbf{x} variables, denoted by circles. The information is then mapped to the next layer (to the right)— a ‘hidden’ layer $\mathbf{h}^{(x)}$ (with l_2 neurons). For input \mathbf{y} , there are m_1 neurons, followed by a hidden layer $\mathbf{h}^{(y)}$ (with m_2 neurons). The mappings continued onto u and v . The cost function J forces the correlation between u and v to be maximized. On the right side, the top NN maps from u to a hidden layer $\mathbf{h}^{(u)}$ (with l_2 neurons), followed by the output layer \mathbf{x}' (with l_1 neurons). The cost function J_1 minimizes the MSE or MAE of \mathbf{x}' relative to \mathbf{x} . The third NN maps from v to a hidden layer $\mathbf{h}^{(v)}$ (with m_2 neurons), followed by the output layer \mathbf{y}' (with m_1 neurons). The cost function J_2 minimizes the MSE or MAE of \mathbf{y}' relative to \mathbf{y} .

The following files do not in general require modifications by the user:

`bicor.m` computes the biweight midcorrelation.

`chooseTestSet.m` divides the dataset (`xdata` and `ydata`) into a training set and a test set.

`chooseTestuv.m` divides the nonlinear canonical variates u and v (Fig. 3) into a training set and a test set.

`calcmod.m` computes the NLCCA mode (or CCA mode). NLCCA uses 3 neural networks (NN1, NN2 and NN3) (Fig. 3) with 3 cost functions J , J_1 and J_2 , and weight vectors \mathbf{W} , \mathbf{W}_1 and \mathbf{W}_2 , respectively. The best ensemble run is selected for the forward map $(\mathbf{x}, \mathbf{y}) \rightarrow (u, v)$, and its (u, v) become the input to the two inverse maps.

`costfn.m` computes the cost function J for the forward map $(\mathbf{x}, \mathbf{y}) \rightarrow (u, v)$.

`mapuv.m` maps $(\mathbf{x}, \mathbf{y}) \rightarrow (u, v)$.

`costfn1.m` computes the costfn J_1 for the inverse map: $u \rightarrow \mathbf{x}'$ (i.e. from u to \mathbf{x}_i).

`invmapx.m` computes the inverse map: $u \rightarrow \mathbf{x}'$.

`costfn2.m` computes the costfn J_2 for the inverse map: $v \rightarrow \mathbf{y}'$ (i.e. from v to \mathbf{y}_i).

`invmapy.m` computes the inverse map: $v \rightarrow \mathbf{y}'$.

`nondimen.m` and `dimen.m` are used for scaling and descaling the data. The input are nondimensionalized according to the scaling option selected, NLCCA is then performed, and the final results are again dimensionalized.

`extracalc.m` can be used for projecting new data onto an NLCCA mode, and is used by the plotting programs.

5.2 Parameters stored in `param.m` for NLCCA

Parameters are classified into 3 categories, ‘*’ means user normally has to change the value to fit the problem. ‘+’ means user may want to change the value, and ‘–’ means user rarely has to change the value.

+ `iprint` = 0, 1, 2 or 3 increases the amount of printout.

Note: Printout summary from an output file (e.g. `out1`) can be extracted by the unix `grep` command: `grep # out1`

+ `irobust` = 0 uses the Pearson correlation in NN1 and the MSE (mean square error) in NN2 and NN3.

`irobust` = 1 uses the biweight midcorrelation in NN1 and the MAE (mean absolute error) in NN2 and NN3. This is the new robust version for handling noisy datasets with outliers.

Actually, various combinations are possible by specifying a 3-element vector for `irobust`: e.g. `irobust` = [1 0 0] uses biweight midcorrelation in NN1 and MSE in NN2 and NN3 (this is the default). `irobust` = [1 0 1] uses biweight midcorrelation in NN1, MSE in NN2, and MAE in NN3, etc.

– `linear` = 0 means NLCCA; `linear` = 1 extracts the linear mode instead. [When `linear` = 1, it is recommended that the number of hidden neurons in the encoding layers be set to 1, and `testfrac` be set to 0].

* `nensemble` is the number of runs in the ensemble. (Many runs may converge to shallow local minima and hence wasted). E.g. [60 30 30] uses 60 runs in the 1st NN, and 30 runs in the 2nd NN and in the 3rd NN. It is advisable to use more runs in the 1st NN.

– `testfrac` = 0 does not reserve a portion of the data for testing or validation, i.e. all data are used for training. Use only when data is not noisy, or a linear solution is wanted. `testfrac` > 0 (usually between 0.15 to 0.25), then a fraction of the data record will be randomly selected as test data.

+ `segmentlength` = 1 if autocorrelation of the data is not of concern; otherwise the user may want to set `segmentlength` to about the autocorrelation time scale.

– `overfit_tol` is the tolerance for overfitting, should be a small positive number ($\ll 0.1$). Only runs having correlation of u, v over the test set no worse than the (correlation over the training set) $\times (1 - \text{overfit_tol})$, or the error (MSE or MAE) over the test set no worse than (error over the training set) $\times (1 + \text{overfit_tol})$ are accepted, and the selected solution is the run with the highest correlation and lowest error over all the data.

* `scaling` controls scaling of the x and y variables before performing NLCCA. `scaling` can be specified as a 2-element vector [`scaling`(1), `scaling`(2)], or simply as a scalar (then both

elements of the vector will take on the same scalar value).

`scaling(1) = -1`, no scaling (not recommended unless variables are of order 1).

`scaling(1) = 0`, scales all x_i variables by removing the mean and dividing by the standard deviation of each variable.

`scaling(1) = 1`, scales all x_i variables by removing the mean and dividing by the standard deviation of the 1st x variable (i.e. x_1). (Similarly for 2, 3, ...)

Note: When the x variables are the *principal components* of a dataset and the 1st variable is from the leading mode, `scaling(1) = 1` is preferred. [If `scaling = 0`, the importance of the higher PC modes (with weak signal-to-noise ratio) can be greatly exaggerated by the scaling].

Similarly for `scaling(2)` which controls the scaling of the y variables.

- + `penalty_store` stores a variety of values for the weight penalty parameter used in the cost functions. During ensemble runs with random initial weights, the `penalty` parameter value is also randomly chosen among the values given in `penalty_store`. Increasing `penalty` leads to less nonlinearity (i.e. less wiggles and zigzags when fitting to noisy data). Too large a value for `penalty` leads to linear solutions and trivial solutions. E.g. if data record is relatively noisy and short, one may use `penalty_store = [3e-1 1e-1 3e-2 1e-2 3e-3 1e-3]`; but if data record is of good quality, one may use `penalty_store = [1e-2 3e-3 1e-3 3e-4 1e-4 3e-5 1e-5]`.
- + `maxiter` scales the maximum number of function iterations allowed during optimization. Start with 1, increase if needed.
- + `initRand` = a positive integer, initializes the random number generator, used for generating random initial weights.
- `initwt_radius` controls the initial random weight radius for the ensemble runs. Adjust this parameter if having convergence problem. (Suggest starting with `initwt_radius = 1`). The weight radius is smallest for the first ensemble run, increasing by a factor of 4 when the final ensemble run is reached.

Note: `maxiter` and `initwt_radius` can each be specified as a 3-element row vector, giving the values to be used for the forward mapping network and the two inverse mapping networks separately. (e.g. `maxiter = [2 0.5 1.5]`). (If given as a scalar, then all 3 networks use the same value.)

Input dimensions of the networks:

- * `l1` = l_1 , the number of input x variables.
- * `m1` = m_1 , the number of input y variables.
- * `l2` = l_2 , the number of hidden neurons in the first hidden layer after the input x variables.
- * `m2` = m_2 , the number of hidden neurons in the first hidden layer after the input y variables. For nonlinear solutions, both l_2 and m_2 need to be at least 2. In the current version, `l2` and `m2` are no longer specified in `param.m`. Instead they are now specified in the main program `nlcca1.m`.

Note: The total number of free (weight and bias) parameters is $2(l_1l_2 + m_1m_2) + 4(l_2 + m_2) + l_1 + m_1 + 2$. In general this number should be less than the number of observations in the dataset. If this is not the case, principal component analysis may be needed to pre-filter the datasets, so only the leading principal components are used as input to NLCCA.

5.3 Variables in the codes for NLCCA

The reader can skip this subsection if he/she does not intend to modify the basic code.

Global variables are given by:

```
global iprint irobust linear nensemble testfrac segmentlength ...
    overfit_tol scaling penalty penalty_store maxiter initRand ...
    initwt_radius options n l1 m1 l2 m2 iter xmean xstd ymean ystd ...
    ntrain xtrain ytrain utrain vtrain xitrain yitrain ntest xtest ...
    ytest utest vtest xitest yitest corruv Errx Erry
```

Be careful not to introduce user variables with the same name as some of the global variables (e.g. `n`, `l1`, `m1`, `l2`, `m2`). If you add or remove variables from this global list, it is **CRITICAL** that the same changes are made on **ALL** of the following files: `nlcca1.m`, `nlcca2.m`, `param.m`, `calcmod.m`, `costfn.m`, `mapuv.m`, `costfn1.m`, `invmapx.m`, `costfn2.m`, `invmapy.m`, `extracalc.m`.

`options` are the options used for the m-file `fminunc.m` in the MATLAB optimization toolbox.

`n` is the number of observations [automatically determined from `xdata` and `ydata` (assumed to have equal number of observations) by the function `chooseTestSet`].

`xdata` is an $l_1 \times n$ data matrix, while `ydata` is an $m_1 \times n$ data matrix.

(A common error is to input an $n \times l_1$ data matrix for `xdata`, or an $n \times m_1$ matrix for `ydata`, which leads MATLAB to complain about inconsistent matrix dimensions).

If `xdata` and `ydata` contain time series data, it is useful to record the measurement time in the variables `tx` and `ty` respectively (both $1 \times n$ vectors). The variables `tx` and `ty` are not used by NLCCA in the calculations.

`iter` is the number of function iterations.

`xmean` and `xstd` are the means and standard deviations of the x_i variables (Similarly, `ymean` and `ystd` are for the y_j variables).

`ntrain`, `ntest` are the number of observations for training and for testing.

`xtrain`, `xtest` are the portions of `xdata` used for training and testing, respectively.

`ytrain`, `ytest` are the portions of `ydata` used for training and testing.

`utrain`, `utest` are the canonical variate u values during training and testing. Similarly for `vtrain` and `vtest`.

`xitrain`, `xitest` are the model predicted values for x during training and testing. (Similarly for `yitrain` and `yitest`).

`corruv` is the (Pearson or biweight mid) correlation over all data (training+testing).

`Errx` is the error (MSE or MAE) of x over all data (training+testing). (Similarly for `Erry`).

`score` can be used to select the best model among several models (with different number of hidden neurons), i.e. choose the highest overall `score` (normalized by that from the linear model), where

`score` = `corruv/sqrt(Errx*Erry)`, (if MSE used for the error);

`score` = `corruv/(Errx*Erry)`, (if MAE used).

6 Hints and trouble-shooting

A common error message from MATLAB is: ‘Inner matrix dimensions must agree’. Check the dimensions of your input data matrices `xdata` and `ydata`. For instance, it is easy to erroneously put in an $n \times l$ data matrix for `xdata` in `NLPCA`, when it should be an $l \times n$ matrix (i.e. the transpose).

Increasing the number of hidden neurons allows more complicated nonlinear solutions to be found, but also more chance of overfitting. One should generally follow the principle of parsimony in that the smallest network which can do the job is the one to be favoured.

Sometimes, the program gives the error that no ensemble run has been accepted as the solution. One can rerun with a bigger ensemble size. One can also avoid using zero or very small weight penalty and/or too many hidden neurons to reduce overfitting, which is usually the cause for having no ensemble runs accepted.

Sometimes, one gets the warning message about $\text{mean}(u^2)$ or $\text{mean}(v^2)$ not close to unity. This means the normalization terms in the cost function were not effective in keeping $\text{mean}(u^2)$ or $\text{mean}(v^2)$ around unity. This usually occurs when large weight penalty were used, so the penalty terms dwarfed the normalization term. One can either ignore the warning or scale up the normalization term in the cost function (in `invmapx.m` for `NLPCA`, and in `mapuv.m` for `NLCCA`).

References

- Cannon, A.J. and Hsieh, W.W., 2008. Robust nonlinear canonical correlation analysis: application to seasonal climate forecasting. *Nonlinear Processes in Geophysics*, **12**, 221-232.
- Hsieh, W.W. 2000. Nonlinear canonical correlation analysis by neural networks. *Neural Networks*, **13**, 1095-1105.
- Hsieh, W.W., 2001a. Nonlinear principal component analysis by neural networks. *Tellus*, **53A**, 599-615.
- Hsieh, W.W. 2001b. Nonlinear canonical correlation analysis of the tropical Pacific climate variability using a neural network approach. *Journal of Climate*, **14**, 2528-2539.
- Hsieh, W.W. 2004. Nonlinear multivariate and time series analysis by neural network methods. *Reviews of Geophysics*, **42**, RG1003, doi:10.1029/2002RG000112.
- Hsieh, W.W. 2007. Nonlinear principal component analysis of noisy data. *Neural Networks*, **20**, 434-443. DOI 10.1016/j.neunet.2007.04.018.
- Hsieh, W.W. and Wu, A. 2002. Nonlinear multichannel singular spectrum analysis of the tropical Pacific climate variability using a neural network approach. *Journal of Geophysical Research* **107(C7)**, 3076, DOI: 10.1029/2001JC000957.
- Kirby, M.J. and Miranda, R. 1996. Circular nodes in neural networks. *Neural Computation*, **8**, 390-402.
- Kramer, M.A. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, **37**, 233-243.
- von Storch, H. and Zwiers, F.W. 1999. *Statistical Analysis in Climate Research*. Cambridge Univ. Pr., 484 pp.
- Yuval, and Hsieh, W.W. 2002. The impact of time-averaging on the detectability of nonlinear empirical relations. *Quarterly Journal of the Royal Meteorological Society*, **128**, 1609-1622.