Available online at www.sciencedirect.com
SCIENCE DIRECT®
Neural Networks 19 (2006) 864–876
Neural Networks
www.elsevier.com/locate/neunet

2006 Special Issue

Graph-based normalization and whitening for non-linear data analysis

Catherine Aaron

SAMOS-MATISSE, Paris, France

Abstract

In this paper we construct a graph-based normalization algorithm for non-linear data analysis. The principle of this algorithm is to get a spherical average neighborhood with unit radius. First we present a class of global dispersion measures used for "global normalization"; we then adapt these measures using a weighted graph to build a local normalization called "graph-based" normalization. Then we give details of the graph-based normalization algorithm and illustrate some results. In the second part we present a graph-based whitening algorithm built by analogy between the "global" and the "local" problem.

Keywords: Normalization; Geodesic distance; Graph; SOM

1. Introduction

Non-linear data analysis methods focus on a "local" indicator and require an adapted normalization beforehand. We can quote for instance:

- ISOMAP (Tenenbaum & de Silva, 2000) or Curvilinear Data Analysis (Lee, Lendasse, & Verleysen, 2000; Lee, Lendasse, & Verleysen, 2004) are based on the analysis of the curvilinear (or geodesic) distance. To compute this distance a local weighted graph is first calculated and the Dijkstra algorithm (Dijkstra (1951)) is computed on this graph. The preliminary normalization has to solve the following problem: **which is the best graph to represent the data topology?** As an illustration, in Fig. 1 we present the 2-nearest-neighbor graph for a spiral set with different scalings. It is obvious that only the last graph allows one to compute a reasonable geodesic distance.
- The Kohonen (or SOM) Algorithm (Kohonen (1995)) adapts the location of some weight vectors according to their neighborhood but **what is a "good" neighborhood for each weight vector?**

The usual normalization methods, such as the most used one – the division by standard deviation (that we will call "standard normalization" in the following) – are global methods and are not adapted to solve the previous questions, even if it is not theoretically clear that these methods fail to answer our problem (normalization of a datum drawn on a non-linear manifold). In this paper we are going to present results for "standard normalization" as an illustration of this failure.

In the first section we will present our normalization algorithm. First we present a class of global dispersion measures that can be easily adapted to local dispersion measures by using a weighted graph that links points.

The most general graph-based normalization method requires a local graph structure (G) and a weight function (f) as input. The precise choice of this parameter is left for future work; we will present some results for a trivial choice of graph and weighted functions. Results focus on the ability of our normalization to make sure that local graphs reflect data topology, and as consequence that geodesic distance and Kohonen maps are well computed.

In the second section we follow the same idea and build a graph-based whitening by introducing whitening at each step of the algorithm, which is helpful when data are rotated.

2. Graph-based normalization

2.1. Dispersion measures

We are going to use the following notations:

- Let $X = (x_{i,j})$ be a matrix in $\mathcal{M}_{n,p}$ that represents the n observations (individuals) of p variables.

E-mail address: catherine_aaron@hotmail.com.

0893-6080/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2006.05.022

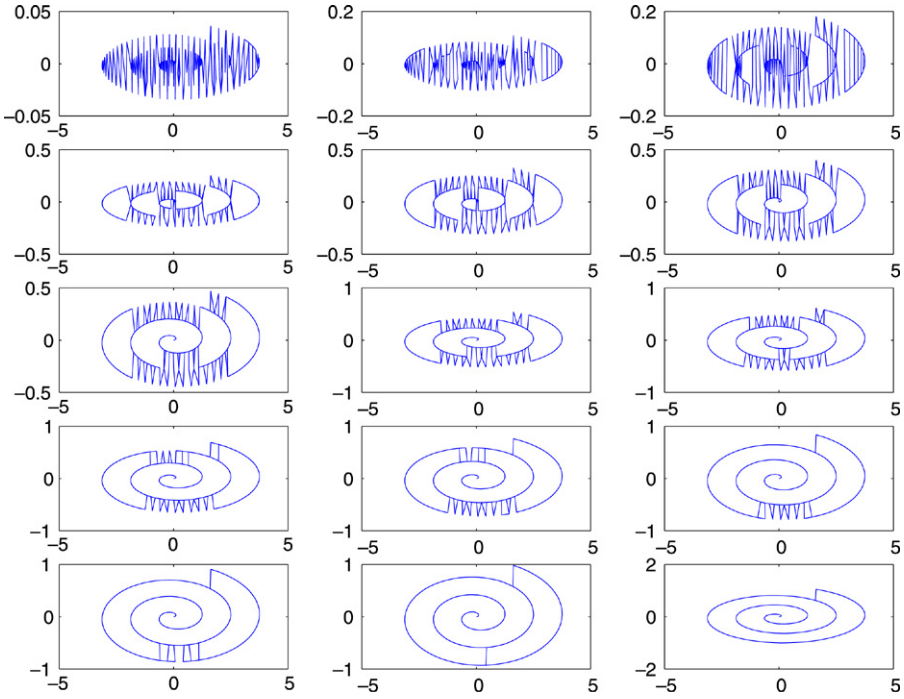

Fig. 1. 2-nearest-neighbor graph for a spiral set with different scaling.

- Let us denote $X_i = (x_{i,1}, \ldots, x_{i,p})$ and $X^k = (x_{1,k}, \ldots, x_{n,k})'$, respectively the ith line of X and the kth column of X.

The most popular dispersion measure for the kth variable is the observed variance:

$$\sigma_k^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i,k} - g_k)^2$$

with

$$g_k = \frac{1}{n}\sum_{i=1}^{n} x_{i,k}.$$

The usual normalization consists in multiplying X by S, a diagonal matrix in $\mathcal{M}_{p,p}$, such that:

$$S_{k,k} = \frac{1}{\sqrt{\sigma_k^2}}.$$

A larger class of dispersion measures is based on dispersion measurement around the barycenters, such as for instance:

$$\delta_k(p) = C\left(\sum_{i=1}^{n}|x_{i,k} - g_k|^p\right)^{1/p}.$$

The global normalization then consists in multiplying X by $S(p)$, a diagonal matrix in $\mathcal{M}_{p,p}$, such that:

$$S_{k,k}(p) = \frac{1}{\delta_k(p)}.$$

We will not be able to adapt these dispersion measures to our local problem because of the barycenter notion for instance. But we can remark that another expression of the variance is:

$$\sigma_k^2 = \frac{1}{n(n-1)}\sum_{i,j}(x_{i,k} - x_{j,k})^2.$$

In this paper we define and use the following dispersion measure class of the:

$$d_k(p) = \left(\sum_{i,j}|x_{i,k} - x_{j,k}|^p\right)^{1/p}.$$

We are not going to discuss the choice of p here. This kind of problem has already been discussed for "global" normalization and results depend on the kind of problem we have. We can first presume that when $\delta(p)$ is chosen for the global normalization, $d(p)$ should be chosen for the local one. In this paper we are only going to study $p \in \{1, 2\}$ (see Ding and Wilkins (2004) for some comparison of normalization methods, which are not exactly in our field).

2.2. Graph-based normalization algorithm

Let G be a weighted graph matrix on the X observations such that:

- $G(i, j) = 0$ if the ith and the jth individual are not connected
- $G(i, j) = f(d(X_i, X_j))$ with f a decreasing function of the distance between the ith and the jth individual.

Here we can define a "local" (graph-based) dispersion measure:

$$d_k(p, G) = \left(\sum_{i,j} G(i, j)|x_{i,k} - x_{j,k}|^p\right)^{1/p}.$$

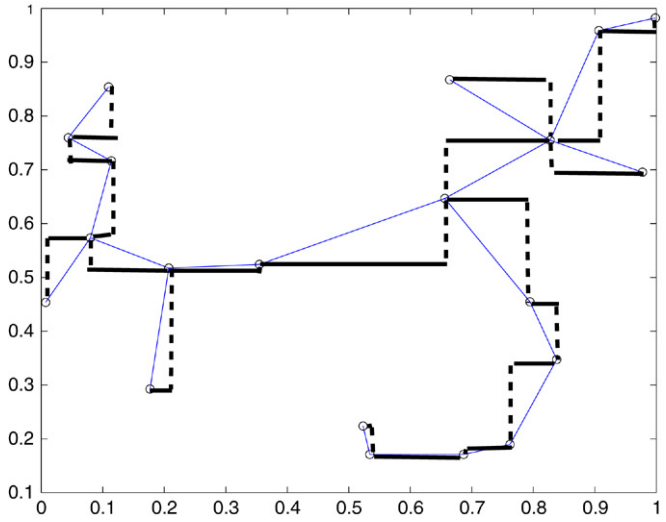

Fig. 2. Example of an MST graph and horizontal and vertical contributions to local dispersion (20 points uniformly drawn on $[0, 1]^2$).

And we would like to get that $\forall k\, d_k(p, G) = 1$.

So we have to apply $Y_{(1)} = X.S_{(1)}(p, G)$ with $S_{(1)}(p, G)$ a diagonal matrix in $\mathcal{M}_{p,p}$ with:

$$S_{k,k}(p) = \frac{1}{d_k(p, G)}.$$

After this first step we may not have obtained yet that $\forall k d_k(p, G) = 1$, and, as we may also have changed the organization of distances ($d(Y_{(1)i}, Y_{(1)j})$ may not be organized as $d(X_i, X_j)$). We therefore have to iterate the following algorithm:

- initialization: $Y_{(0)} = X$, $G_{(0)} = graph(X)$
- step $t + 1$
- $G_{(t+1)} = graph(Y_{(t)})$
- $d_k(p)(t+1) = (\sum_{i,j} G_{(t+1)}(i, j)|Y_{(t)i,k} - Y_{(t)j,k}|^p)^{1/p}$ (and compute the associated matrix $S(p, G)(t + 1)$)
- $Y_{(t+1)} = Y_{(t)}S(p, G)(t + 1)$.

2.3. Choice of a graph

2.3.1. Graph structure

Classical graph structures are the following:

- (1) k-nearest neighbors (this kind of graph is not a problem for our algorithm)
- (2) Minimal Spanning Tree (MST) (Fig. 2)
- (3) ε-neighborhood X_i and X_j are connected if and only if $d(X_i, X_j) < \varepsilon$. This kind of graph should not be used "directly". Indeed, after each re-scaling, the distances between points are going to change and a fixed ε is not adapted to this. So, if we want to work with such graphs we have to use an adaptive ε parameter.

In the following we only focus on k-nearest-neighbor and MST graphs, but the construction of efficient $\varepsilon(X)$ for ε-neighborhood graphs should be an interesting problem to develop in the future.

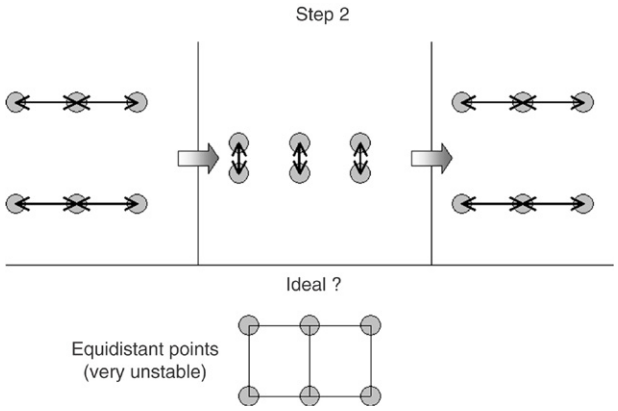

Fig. 3. Extreme case to illustrate instability for 1-nearest neighbors.

2.3.2. Number of neighbors for k-nearest-neighbor graphs

In the "dispersion measure" Section 2.1 we have seen that, for $p = 2$ and the complete graph (i.e. the $(n - 1)$-nearest-neighbor graph), the graph-based normalization and the usual division by standard deviation were exactly the same. We can generalize to the obvious fact that, if k is too large, we lose information on local problems and it will not be adapted to strongly non-linear problems.

If we select too small a value for k we lose stability. This can be illustrated with the extreme case (Fig. 3) of instability with a 1-nearest-neighbor graph:

So the choice of k comes from the compromise between locality and stability.

Nevertheless, empirical results show that instability cases do not appear and that the smallest choice for k, $k = 1$, is "the best" choice.

2.3.3. MST versus k-nearest neighbors

Even if we have not observed instability cases, we have to keep this problem in mind. An alternative to an arbitrary choice of k for k-nearest neighbors can be the choice of the Minimal Spanning Tree (MST). The main inconvenience of such a choice is the running time: a k-nearest-neighbor graph needs an running time of $O(kn^2)$ whereas an MST graph needs $O(n^2 \log(n))$ which may be quite long for a huge data set (our developed MathLab program is long, more than 1 hour for $n > 1000$).

The same running-time problem exists for any MST-based choices of graphs such as, for instance:

- k-nearest neighbors with k the maximum number of neighbors reached in the MST
- connecting i to j if $d(X_i, X_j)$ is shorter than the maximum edge length of the MST starting from X_i or X_j.

This running-time problem has motivated the stochastic version of the algorithm that is briefly described here.

The algorithm is mostly the same. We now need a new input (n_1) that represents the size of a sampling of X. At each iteration we select n_1 points on which the local dispersion is computed and we apply re-scaling on the total set.

In the following algorithm we denote $alea(X, n_1)$ a sampling of n_1 elements of X.

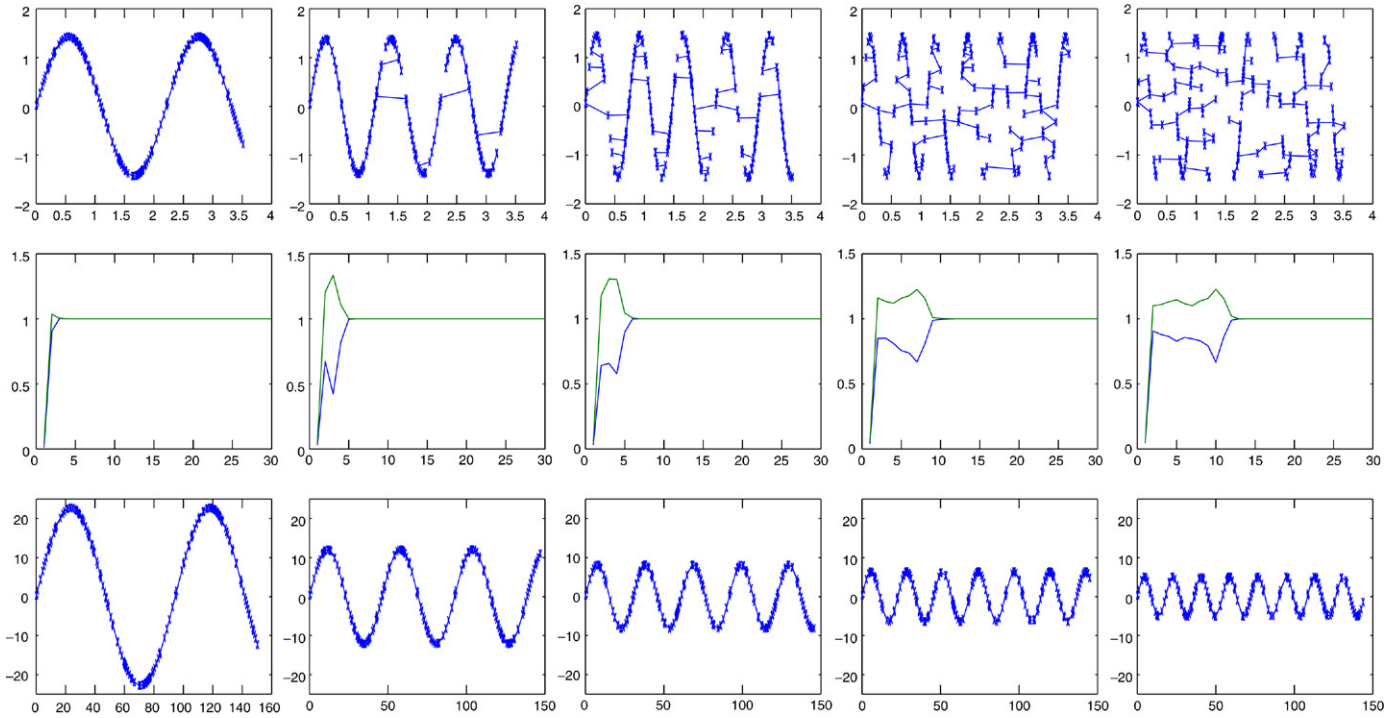

Fig. 4. Results for sinusoidal data ($\omega \in \{10, 20, 30, 40, 50\}$).

- initialization: $Y_{(0)} = alea(X, n_1)$, $G_{(0)} = graph(Y_{(0)})$, $Z_{(0)} = X$,
- step $t + 1$
- $Y_{(t+1)} = alea(Z_{(t)}, n_1)$
- $G_{(t+1)} = graph(Y_{(t+1)})$
- $d_k(p)(t+1) = (\sum_{i,j} G_{(t+1)}(i, j)|Y_{(t+1)i,k} - Y_{(t+1)j,k}|^p)^{1/p}$ (and compute the associated matrix $S(p, G)(t + 1)$)
- $Z_{(t+1)} = Z_{(t)}S(p, G)(t + 1)$.

2.3.4. Choice of the weight function f

As with the graph structure with a ε-neighborhood, the choice of the weight function has to be adaptive. In the following we decided to only choose a constant function $f = 1$. The following question: "is a 'good' adaptive function f better (and quicker) than an adaptive graph?" remains to be answered in the future.

2.4. Some results

In this section we present some results of graph-based normalization. Here, we mainly present the effect of normalization on sinusoidal examples. We chose these kinds of data because the standard normalization is not adapted to them. Indeed data which are not adapted to standard normalization must be neither linear (i.e. "very folded") nor symmetrical in the different directions. Sinusoidal drawing satisfies all these conditions when the dimension is 2. For higher dimension we can find other satisfying examples. As our results are mostly empirical and graphical we only study examples for dimensions 2 and 3.

Results will be presented for:

- Graph organization
- Geodesic distance computation
- SOM organization.

2.4.1. Results for graph organization

In Figs. 4 and 5 we present the results of our normalization algorithm for 200 points drawn on a sinusoid: X^1 corresponds to a uniform drawing of 200 points on [0, 1] and $X^2 = \sin(\omega X^1)$ (with $\omega \in \{10, 20, 30, 40, 50\}$ in Fig. 4 and $\omega \in \{60, 70, 80, 90\}$ in Fig. 5).

For each frequency we present (vertically):

- The usually normalized (division by standard deviation) data and corresponding MST (the horizontal goes from 0 to 4 and the vertical scale from -2 to 2).
- The evolution of the dispersion for vertical and horizontal direction through the algorithm (30 iterations); the two dispersions are expected to be 1.
- Finally, the last picture for each example represents the graph-based normalized set (and its MST graph): horizontal scale from 0 to 150 and vertical scale from -25 to 25—except for $\omega \neq 90$.

The parameters of the algorithm were $p = 1$ and a 4-nearest-neighbors graph structure.

As the frequency increases we can observe that:

- the usual normalization quickly fails to recompute MST (starting with $\omega = 20$)
- the algorithm needs more iterations to converge to a solution (i.e. a scaling that gives local dispersions equal to 1 in each direction)

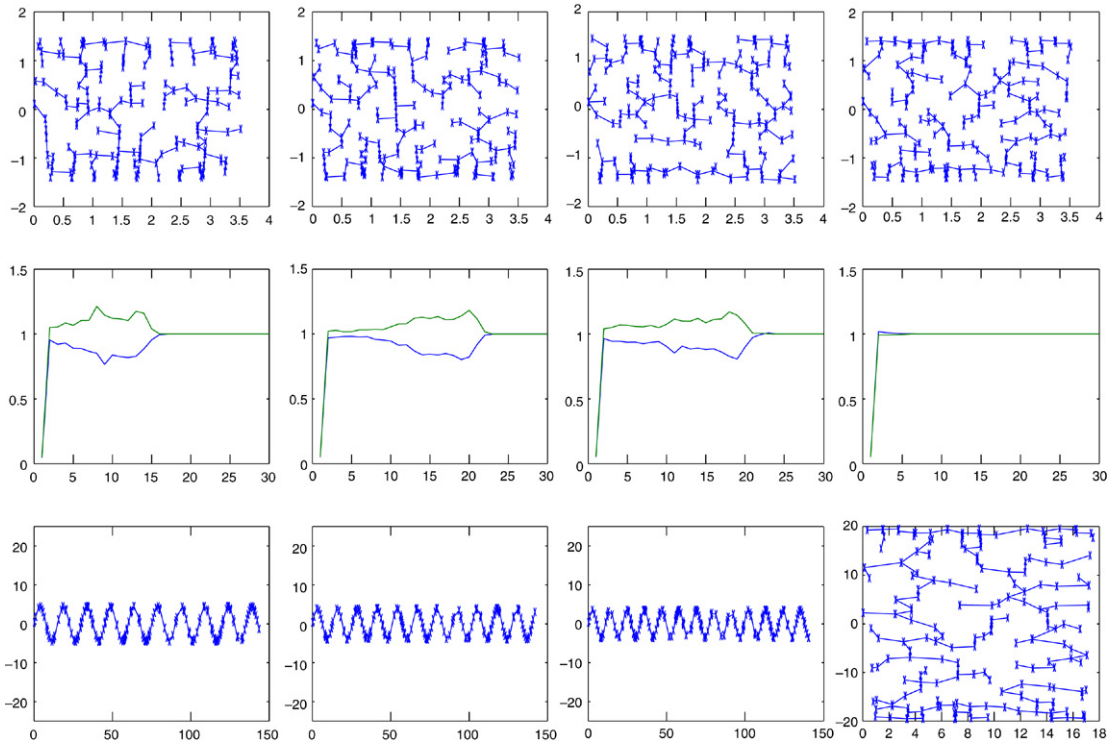

Fig. 5. Results for sinusoidal data ($\omega \in \{60, 70, 80, 90\}$).

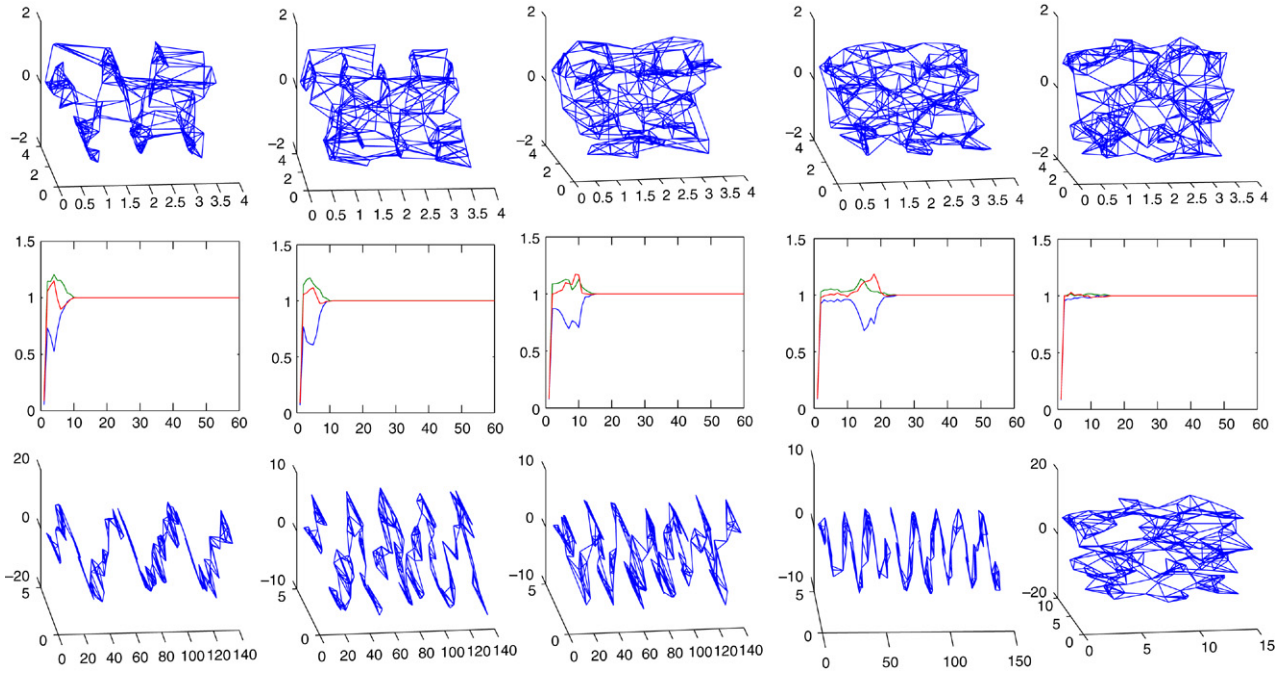

Fig. 6. 3-D sinusoidal data $\omega \in \{20, 30, 40, 50, 60\}$.

- the graph-based normalization allows MST recognition when the usual normalization fails but a "saturation effect" occurs when frequencies are too high (here for $\omega = 90$).

We also present some results for dimension 3. In Fig. 6 the sample corresponds to: X^1 uniform on [0, 1], X^2 uniform on [0, 1] and $X^3 = \sin(\omega X^1)$. Each time we draw 200 points and test a 4-nearest-neighbor normalization (so it is possible to measure the effect of a new "noise" dimension by comparing the previous examples: the saturation effect happens from $\omega = 60$, i.e. sooner than for the previous 2-dimensional example). Results are presented for $\omega \in \{20, 30, 40, 50, 60\}$.

In Fig. 7 we tested the normalization on spirals.

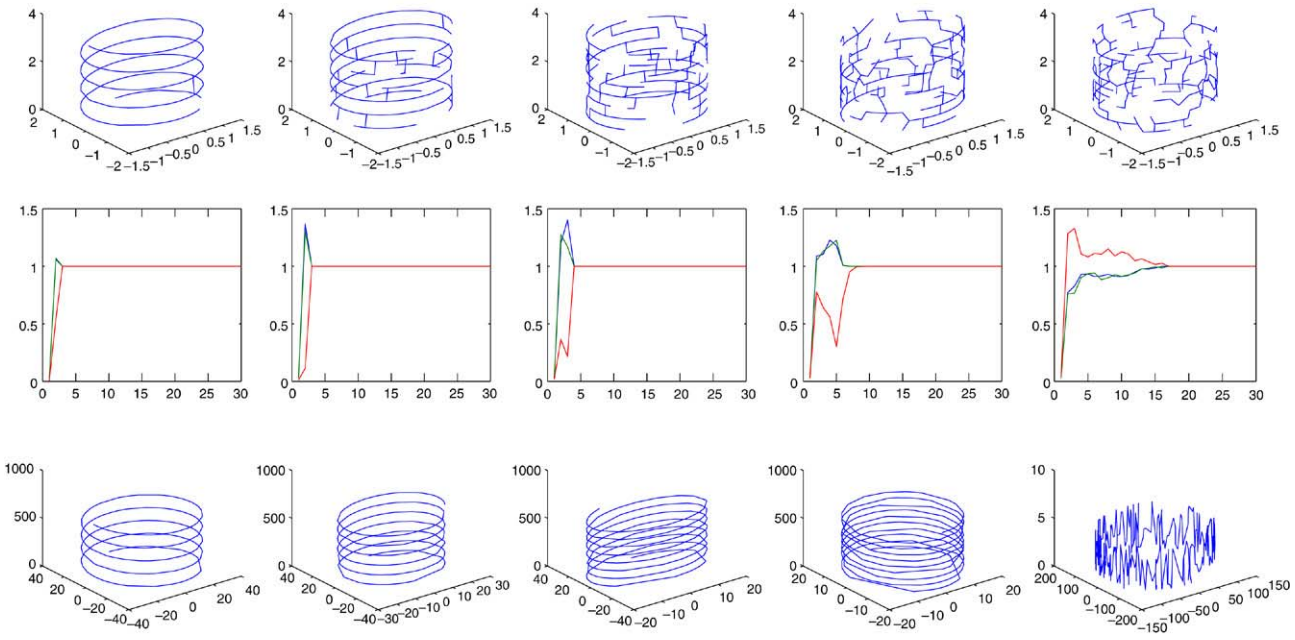

Fig. 7. Spiral.

The parameters of the algorithm are the same as in the first test ($p = 1$ and a 4-nearest-neighbor graph).

2.4.2. Results for geodesic distance computation

In Fig. 8 we present the results of our normalization algorithm for geodesic distance computation on sinusoidal samples. Each set of points has been drawn as follows: X^1 corresponds to a uniform drawing of 100 points on [0, 1] and $X^2 = \sin(\omega X^1)$; $\omega \in \{10, 15, 20, 25, 30\}$.

As the "true" organization is known here we can observe the evolution of the quality of the geodesic distance computation through the normalization algorithm. For that, at each step of the graph-based normalization algorithm we estimate the geodesic distance via the Dijkstra algorithm (run on a graph based on the MST: the MST is computed, then we connect a point X_i to a point X_j if $d(X_i, X_j)$ is inferior to twice the maximum edge length of the MST that has X_i or X_j as extremity).

For each example we present all the scatter plots of the "true" geodesic distance (on the horizontal axis) versus the computed one (on the vertical axis) until the convergence and, to conclude in each example, we present the evolution of the axis weight (thin lines) and the evolution of the correlation coefficient between the "true" and "computed" geodesic distance (plain and dashed line).

As the frequency increases we need more and more steps of the algorithm to converge, and finally the saturation effect happens for $\omega = 30$.

2.4.3. Results for Kohonen maps

We observed that Kohonen maps (Kohonen (1995)) are very sensitive to normalization, even more than the graph-recognition and the geodesic-distance computation. To understand this sensitivity a little let us quickly sum up the Kohonen algorithm:

At each iteration we:

- Draw a point in the observation set
- Search the nearest weight vector of the drawn point
- Move the nearest weight vector and its neighbors toward the drawn point.

The algorithm usually ends with the only nearest weight vector of the drawn point moving (this is the 0-neighbors step). A weight vector moves if the drawn point is inside the Voronoï cell of the weight vector (Voronoï cells computed with the weight-vector set). And these Voronoï cells are very sensitive to normalization, as illustrated in Fig. 9.

If the scaling is not "good", the Voronoï cells of the weight vectors will not give a good representation of the topology of the set which will imply instability and wrong movements of the weight vectors. We computed a Kohonen String on the two previous examples (i.e. the same data as in Fig. 9). We choose to parameterize the Kohonen string with 50 units and we observe that the results strongly depend on the scaling (Fig. 10):

These examples show how the SOM algorithm is sensitive to scaling. The example has been "hand made" and we are now going to see whether the graph-based normalization algorithm gives results that are compatible with the Kohonen Algorithm.

Now, to present results we analyze the same example as in Figs. 4 and 5 but, instead of drawing the MST on initial and final data, we present the result of a Kohonen string with a number of units that depends on the ω frequency (the number of units is equal to $3\omega/2$). We can observe (Fig. 11) that Kohonen maps are rather more sensitive to normalization than graph computation but our algorithm manages to re-scale data so that we observe a good topological organization (of course there is also a saturation effect).

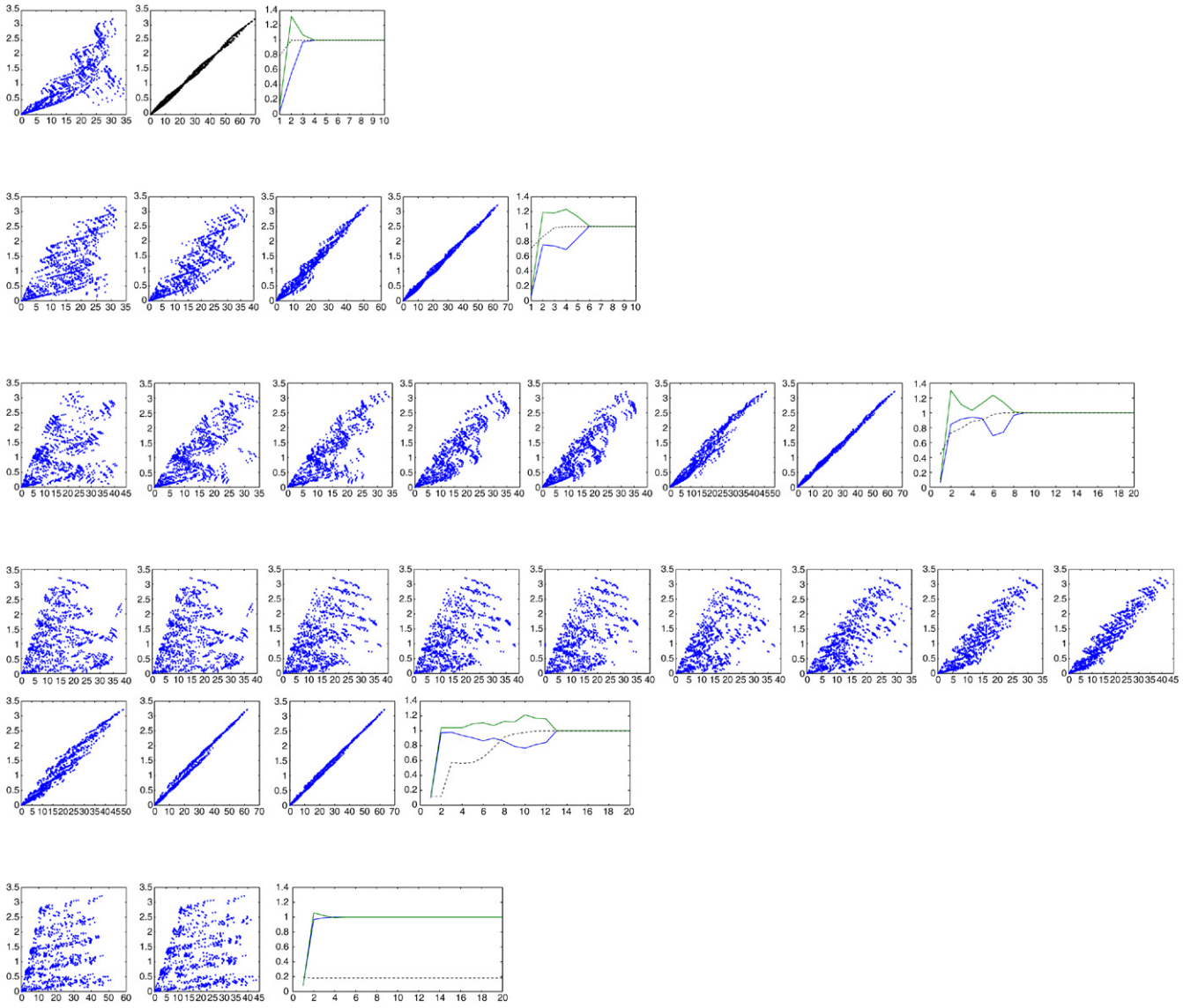

Fig. 8. Normalization and geodesic distance computation for a 2D sinusoidal drawing of 100 points and a frequency $\omega \in \{10, 15, 20, 25, 30\}$.

2.4.4. Choice of the graph

In this section we tested the effect of the number of nearest neighbors for the graph on sinusoidal samples of 100 points and for some examples of frequencies. To sum up the results:

- $\omega = 20$: from 1 to 10 (and maybe more) nearest-neighbor graphs and MST manage to normalize the data
- $\omega = 30$: from 1 to 7 nearest-neighbor graphs and MST manage to normalize the data
- $\omega = 40$: from 1 to 5 nearest-neighbor graphs and MST manage to normalize the data (see Fig. 12)
- $\omega = 50$: from 1 and 2 nearest-neighbor graphs and MST manage to normalize the data
- $\omega = 60$: from 1 and 2 nearest-neighbor graphs and MST manage to normalize the data
- $\omega = 70$: all normalization fail because of the saturation.

We can observe the "instability" effect of a small number of nearest neighbors in term of variations of axis weights: for a small number of nearest neighbors the algorithm quickly converges but there are high variations of axis weights, as the number of neighbors increases, the convergence time is longer but the variation is smaller. As the effect of the size of the variation is not so important we may prefer, in the future, 1-nearest neighbor or MST graphs. As the time for the algorithm is really quicker for 1-nearest-neighbor we may prefer this kind of normalization, but we have to keep in mind the possible instability and have a careful look at the evolution of axis weights to validate the results.

3. Graph-based whitening

3.1. Motivation

When the different variables are not, as in the previous examples, made of a subset of independent and a subset of these variable functions ($X = (X^1, \ldots, X^p)$ with $X^1, \ldots, X^k$ independent variables and $X^{p+i} = f_i(X_j)$) but is a

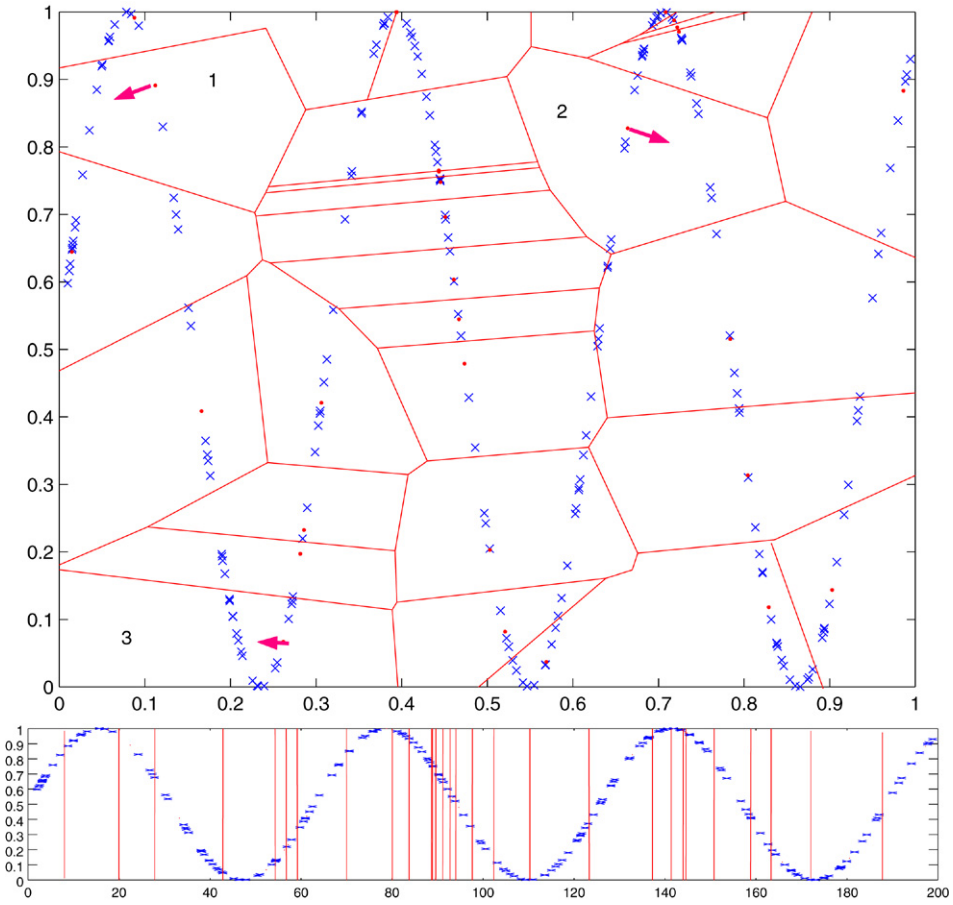

Fig. 9. Voronoï cells of random points (simulation of weight vectors) on a sinusoid, and an example of "wrong" movement in the first case. Data points are represented by crosses and weight vector by points.

transformation of such a set (for instance a rotation), the previous algorithm may fail.

The problem is now how to adapt the graph-based normalization to such data?

3.2. Whitening

In this section, for convenience's sake, we will suppose that the data observation X is centered. For "linear" data analysis, the solution to the previous problem is given by the whitening or PCA method (Joliffe (2002)): a set of observations X gets a covariance matrix Ω (symmetrical, definite and positive) and there exists a linear transformation (represented by a matrix A) such that $\mathrm{cov}(X.A) = Id$.

Indeed $\mathrm{cov}(X) = X'X = P'\Delta P$ by diagonalization, with Δ a diagonal matrix. Let us denote D the diagonal matrix such that $D_{i,i} = \Delta_{i,i}^{-1/2}$.

Then $A = PD$ is a solution.

We are going to adapt this here to our "local" problem for a non-linear data set.

3.3. Graph-based whitening

For the whitening we can only work on a "binary" weighted graph: $G(i, j) = 1$ if X_i is linked to X_j and $G(i, j) = 0$ otherwise. We also need to work on a "symmetrical" graph

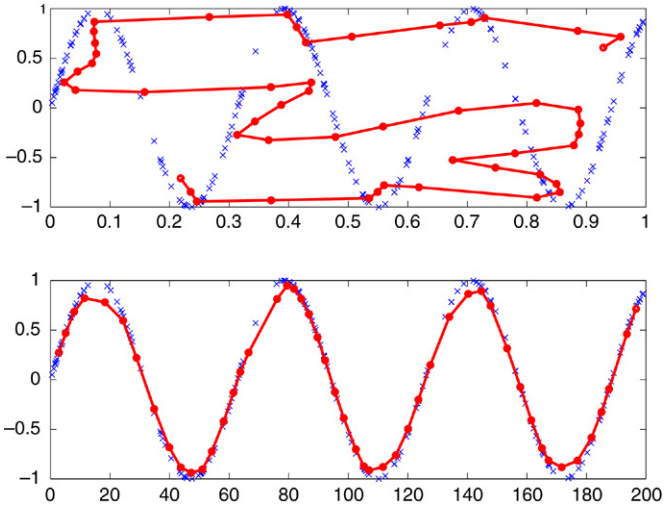

Fig. 10. Results for a Kohonen String with two different scalings.

$(G(i, j) = 1 \longrightarrow G(j, i) = 1)$. If the graph is not symmetrical we can easily make it symmetrical by adding edges.

With such a graph, the previous "graph-based normalization" consists in the (classical) normalization of the "new" data E formed by all the edge vectors:

$\forall i_0 \exists (i, j)$ such that $E_{i0} = X_i - X_j$ with $G(i, j) = 1$

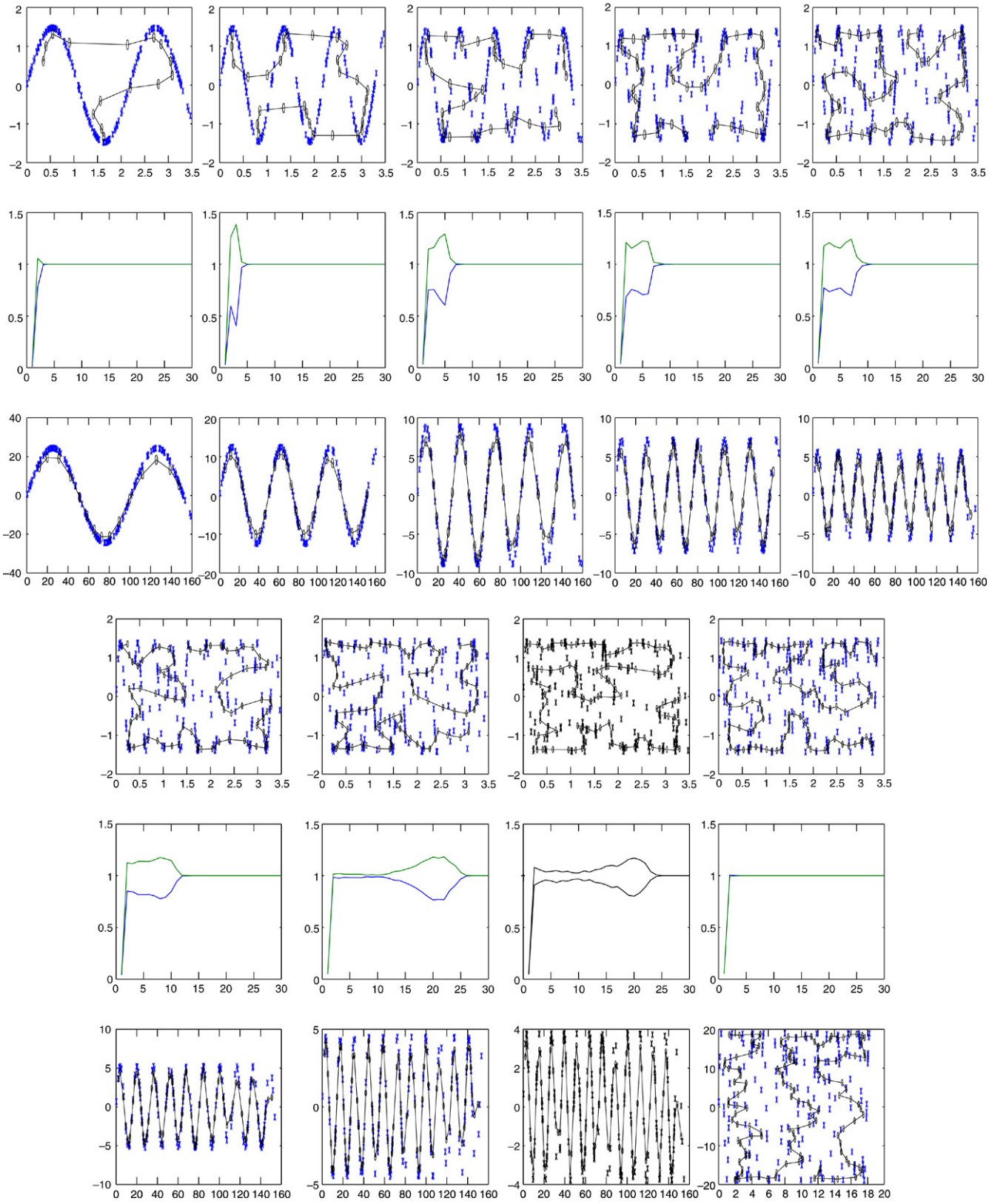

Fig. 11. Normalization and Kohonen maps.

and conversely:

$\forall(i, j)$ with $G(i, j) = 1, \exists i_0$ such that: $E_{i0} = X_i - X_j$.

As we choose a symmetrical graph, it is obvious that the mean of E is null.

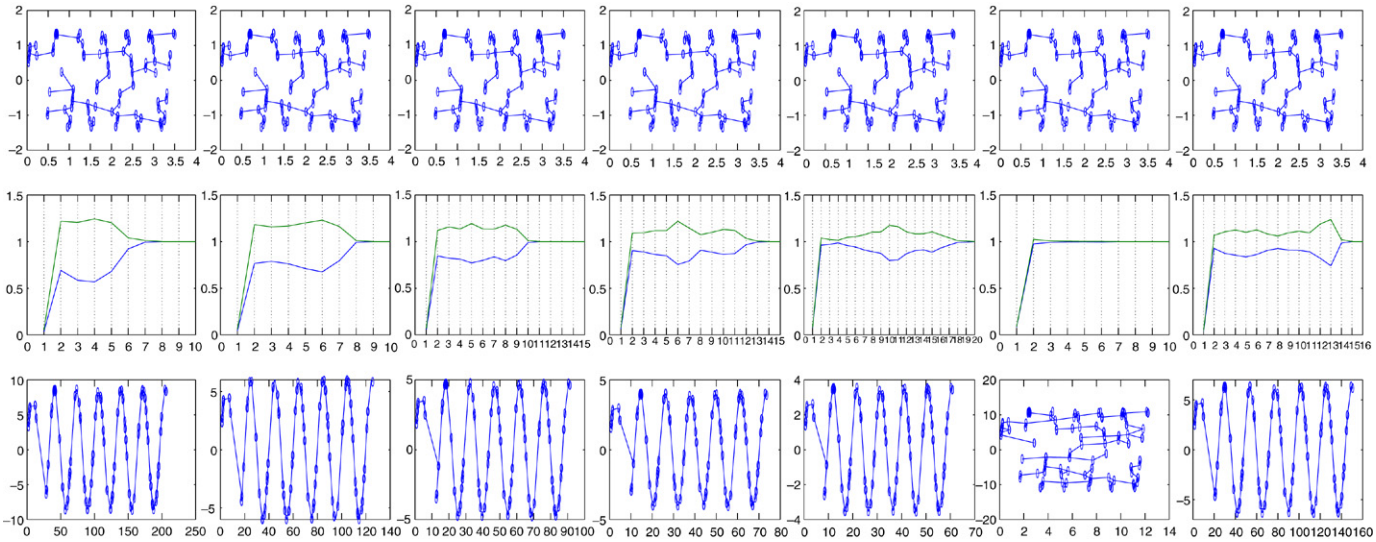

Fig. 12. Example for $\omega = 40$ and 1-nearest-neighbor to 6 nearest-neighbor normalization (with saturation for 6-nearest-neighbor) and MST-normalization.

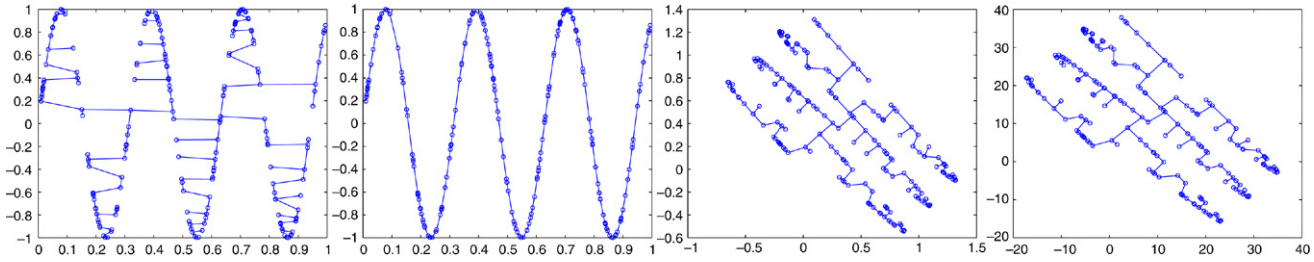

Fig. 13. Graph-based normalization for a subset and the same subset that has been rotated; in the second case the normalization fails.

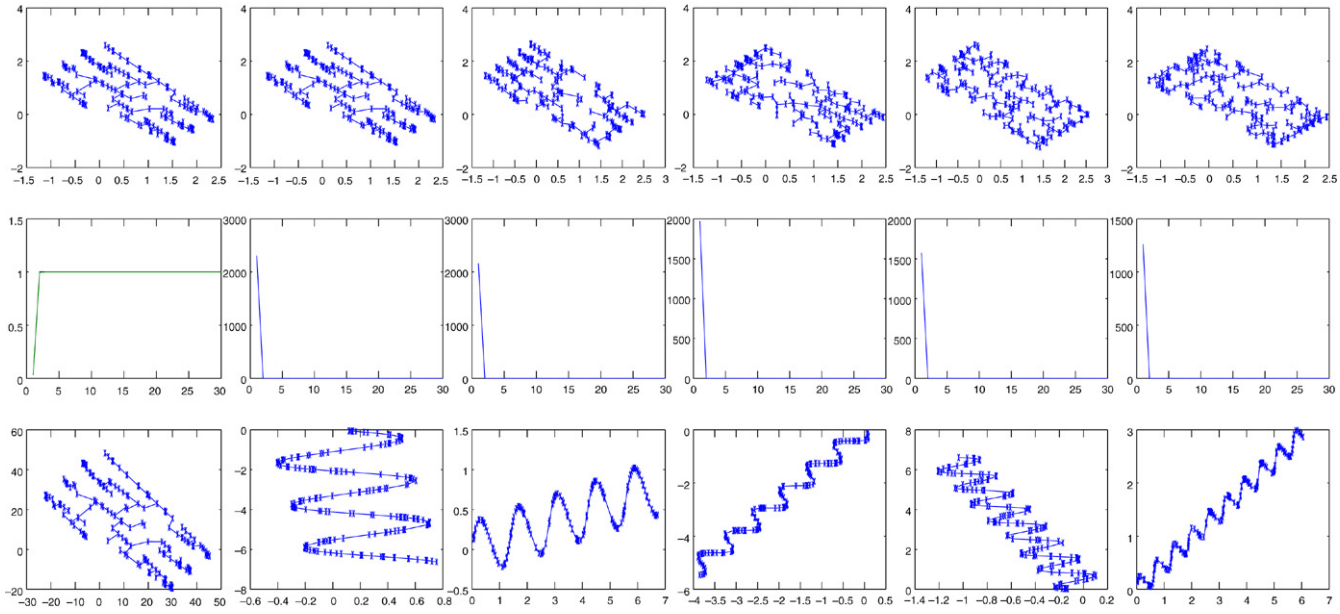

Fig. 14. Results for sinusoidal data ($\omega \in \{20, 30, 40, 50, 60\}$); the first graph corresponds to the graph-based normalization ($\omega = 20$) and the following are the graph-based whitening for $\omega \in \{20, 30, 40, 50, 60\}$.

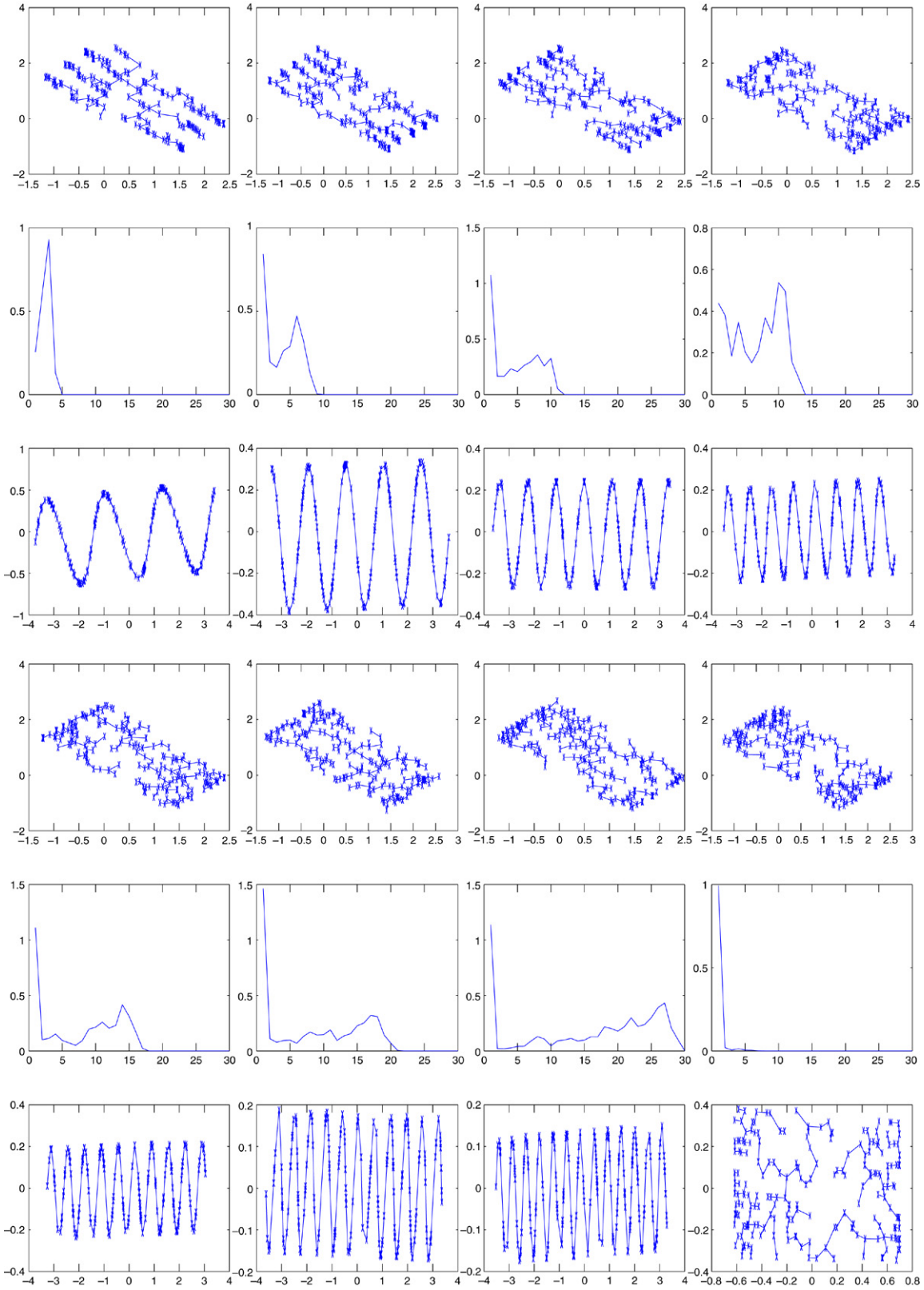

Fig. 15. Results for sinusoidal data and 2-nearest neighbors whitening ($\omega \in \{20, 30, 40, 50, 60, 80\}$).

Thus standard normalization on E (measuring dispersion with $(\mathbb{E}(|E|^p))^{1/p}$) is equivalent to graph-based normalization (we choose the same parameter p) (Fig. 13) .

With this analogy we can build graph-based whitening by computation of the "classical" (i.e. linear) whitening of the edges data set E.

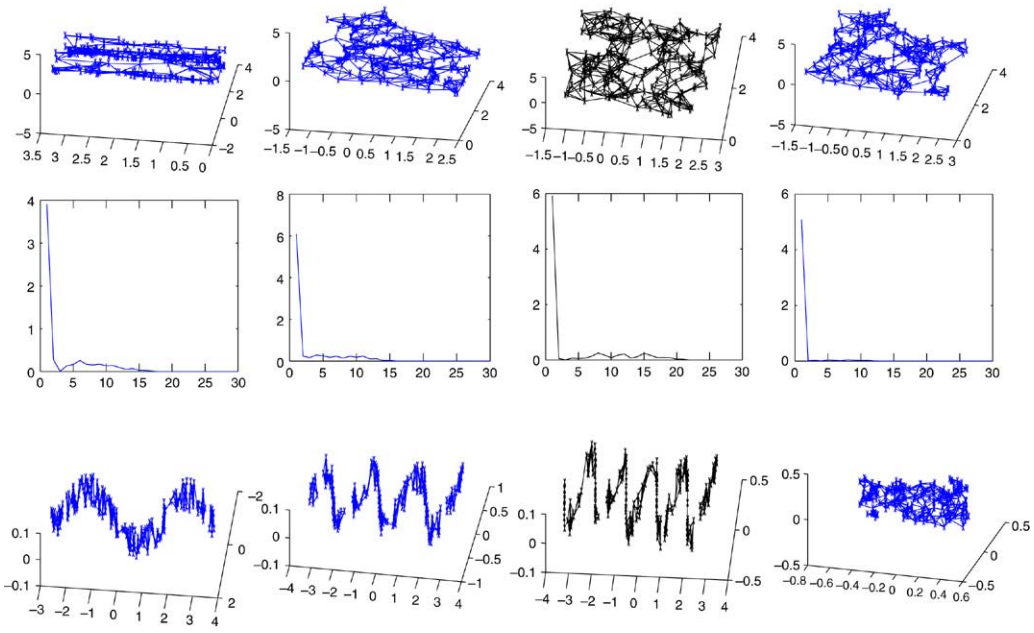

Fig. 16. Results for 3D-sinusoidal data and 2-nearest neighbors whitening ($\omega \in \{10, 20, 30, 40\}$).

The algorithm is the following:

- For it $= 1$ to ItMax
- Compute the edge data E of the data set X (according to the chosen graph structure)
- Find the "whitening" matrix A such that $\text{cov}(EA) = Id$.
- Apply $X := XA$.

3.4. Preliminary results and algorithm amelioration

In this section we are going to present some results for graph-based whitening on some data, similar to the data presented in Figs. 4–7 but that we have rotated (each time by a $\pi/4$ rotation). First we present an example of graph-based normalization to show the failure of this algorithm when the data have been rotated, then the result of the graph-based whitening.

For graph-based normalization the convergence indicators were all the axis weight. For graph-based whitening, at each iteration we compute a whitening matrix A that is expected to converge to the identity matrix. We therefore choose $\|A - Id\|$ as a convergence indicator and we expect it to be null at the end of the algorithm.

In Fig. 14 we can observe that the result of graph-based whitening is quite good but data are still rotated. It is easy to understand why: once a local variance is equal to an identity matrix, all data rotations of the graph-based whitened data are solutions (i.e. have a local variance equal to an identity matrix). So to finally get homogeneous results we propose to apply a Principal Component Analysis (PCA) to the graph-based whitened data.

3.5. Final results

Finally we present the result for the 2-nearest-neighbor whitening method for the same data that we used in Section 2.4.1 (Figs. 4–7). Contrary to what we did in the Section 2.4.1 we have not presented results for 4-nearest neighbors because the whitening is much more sensitive than the normalization to the number of neighbors and the saturation effect comes sooner for whitening (Figs. 15–17).

4. Conclusion and further work

Even if theoretical aspects of the algorithm are not yet clear, is the convergence assured, is the solution unique and why does the solution allow us to recognize the "good" topology? The empirical results are quite encouraging. We may thing that if, instead of working on a k-nearest-neighbor graph, we work on a complete graph with a "good" weight function f, theoretical aspects will be easier to understand, but this has not yet been done. As said before a more detailed discussion of the choice of the graph and of the weighted function also has to be considered.

There are many possible further developments.

For example we yet uses the ability of the normalization to get valid results in computation of the geodesic distance and the Kohonen maps to build an indicator of the "good parametrization" of Kohonen maps. We understand by "good parametrization" knowledge of the dimension and a "good" number of weight vectors in each direction. This leads us to a more detailed idea of the intrinsic dimension of a datum.

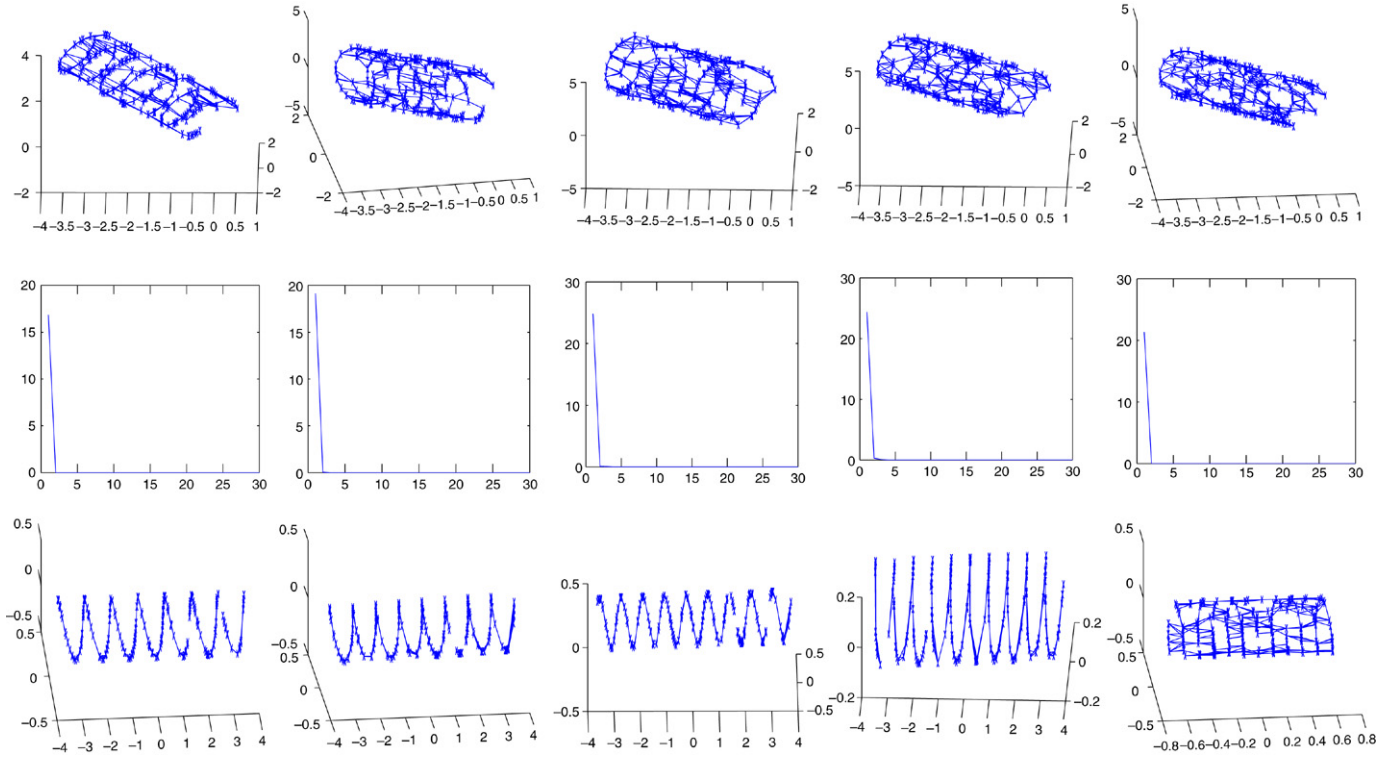

Fig. 17. Results for spiral data and 2-nearest neighbors whitening.

References

Dijkstra, E. W. (1951). A note on two problems in connection with graphs. *Numerische Mathematik*, *1*, 269–271.

Ding, Y., & Wilkins, D. (2004). The effect of normalization on microarray data analysis. *DNA and Cell Biology*, *23*(10), 635–642.

Joliffe, I. T. (2002). *Springer series in statistics. Principal component analysis*. Berlin: Springer.

Kohonen, T. (1995). *Self-organizing maps*. Berlin: Springer.

Lee, J. A., Lendasse, A., & Verleysen, M. (2000). A global geometric framework for non-linear dimensionality reduction. In *Proceedings of the 8th European symposium on artificial neural networks*: *Vol. 1* (pp. 13–20).

Lee, J. A., Lendasse, A., & Verleysen, M. (2004). Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis. *Neurocomputing*, *57*, 49–76.

Tenenbaum, J. B., & de Silva, V. (2000). A global geometric framework for non-linear dimensionality reduction. *Science*, *290*, 2319–2323.