Clustering protein conformations using SOM

Guillaume Bouvier
Bioinformatique Structurale Team
Institut Pasteur, Paris

November 29, 2013

Plan

Introduction and theory

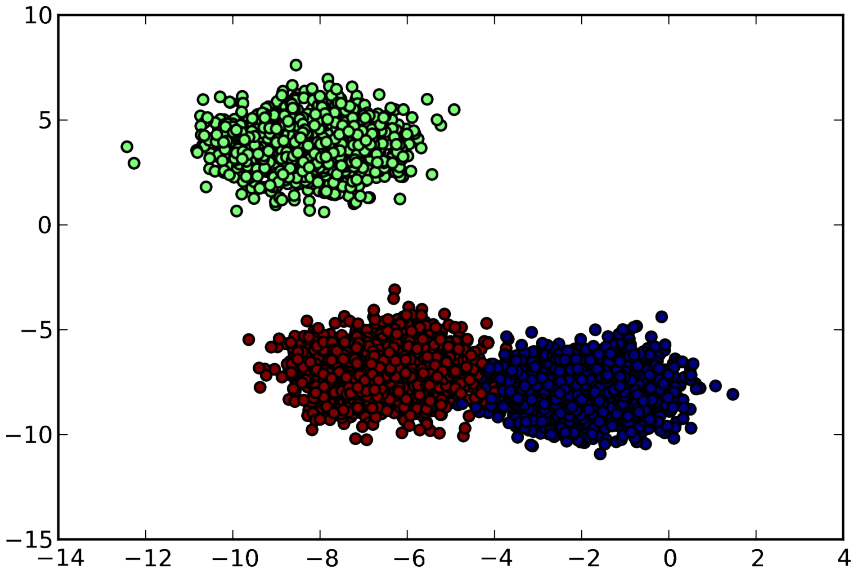
Application to protein conformation clustering

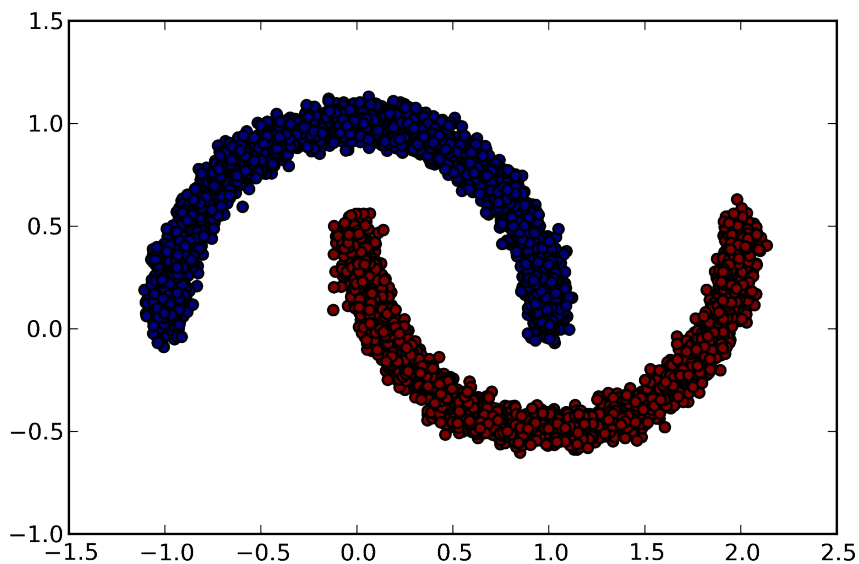
How to do in practice?

What is a cluster?

[...]many authors [...]attempt to define just what a cluster is in terms of internal cohesion – *homogeneity* – and external isolation – *separation*.

Everitt, 2011



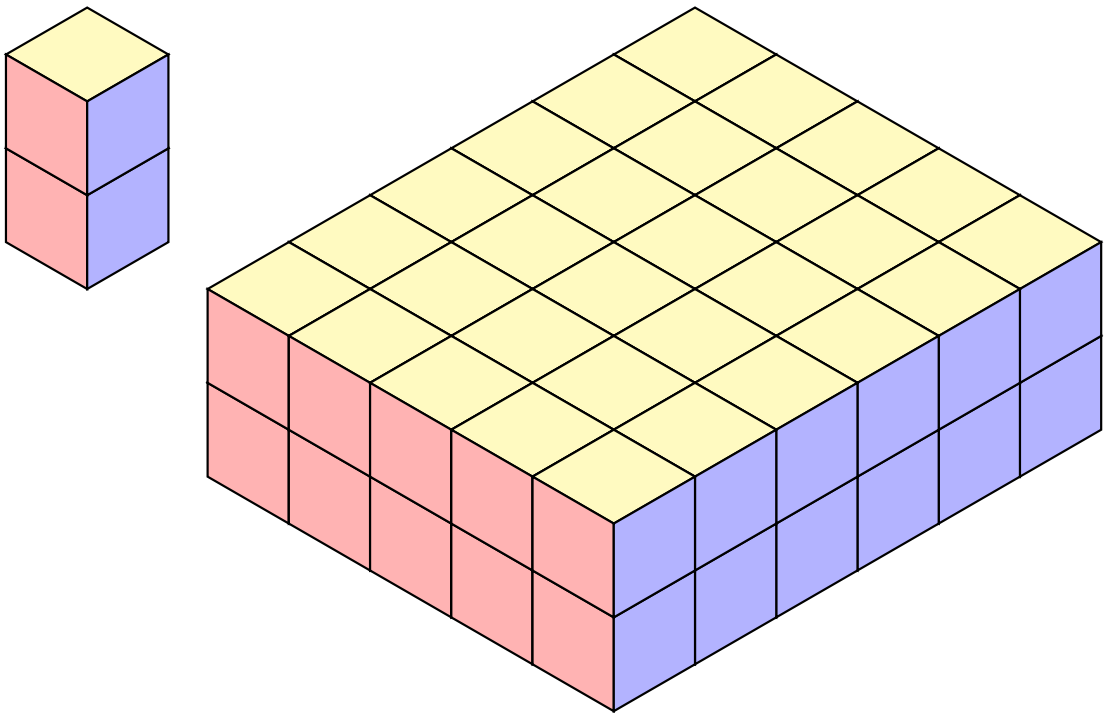


Clusters with **internal cohesion** and/or **external isolation**.

What is a SOM?

- SOM stand for **Self-Organizing Map** and was first described by the Finnish professor Teuvo Kohonen.
- A SOM is an **artificial neural network** that is trained using an **unsupervised learning** process.
- The **dimension of the SOM** $(X \times Y)$ is chosen by the user (only 2D SOM will be aborded).
- Mathematically, a 2D-SOM is a **3D-matrix** of dimension $(X \times Y \times n)$ with n the dimension of the input space.
- A **neuron** is a cell characterized by its position (i, j) in the $(X \times Y)$ plane of the SOM. Its dimension is n.

SOM architecture

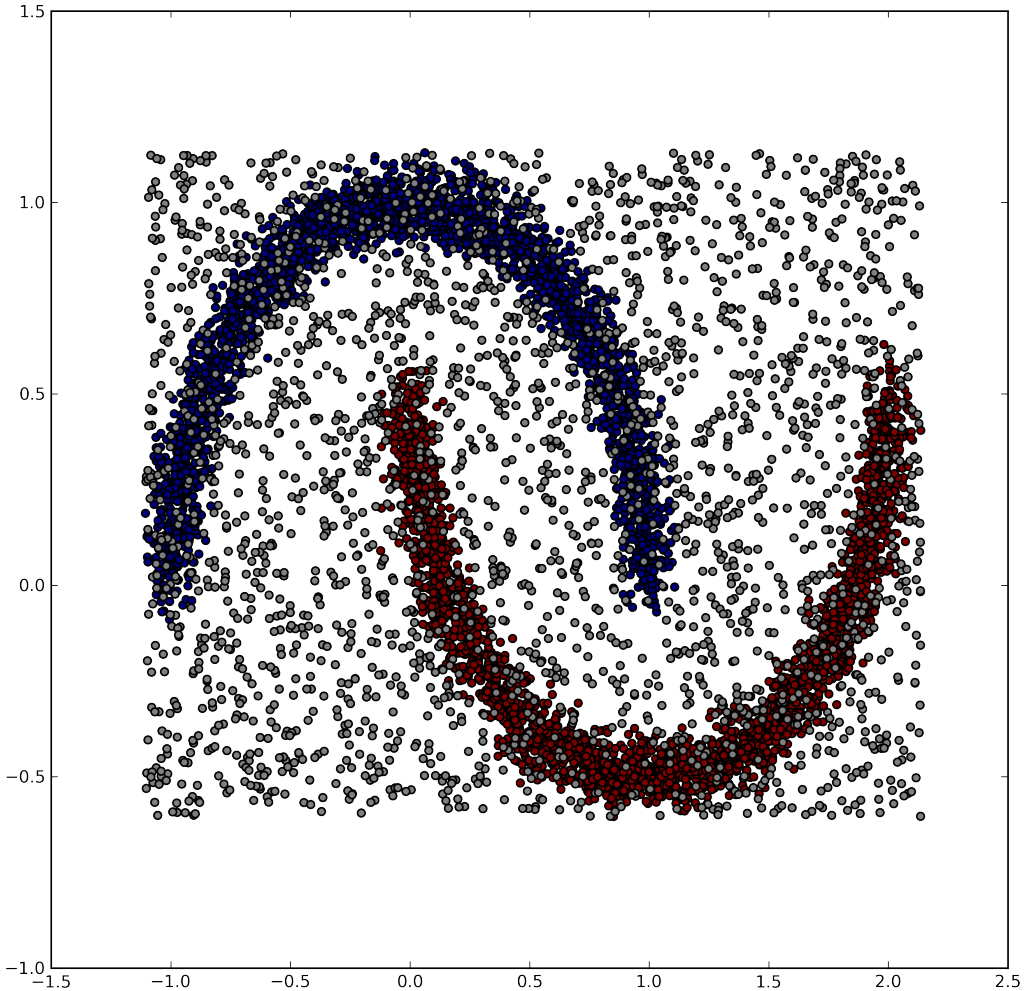


A torus!

Example of an input vector of dimension $n = 2$ and a SOM of dimension 5×5. The SOM is periodic: a torus.

Initialization of the SOM

The SOM is initialized randomly with an uniform distribution within the range of the input data.



Example of SOM initialization on the moon dataset. Take care to differentiate the **SOM space** which is a **2D lattice** of 2D vectors for this example and the **input space**, represented on the left.

The algorithm

For each **iteration**

- we select randomly an input vector from the input space
- we compute the Euclidean distance between the input vector and each neuron of the map
- we select the neuron with the minimal distance, which is called the **Best Matching Unit** (BMU)
- we modified the map with the following formula:

- Linear adjustment of the weights.
- Neighborhood function: regulates the influence of the BMU $(\beta_1, \beta 2)$ on the neighboring neurons.

$$M(t+1) = M(t) + \alpha(t) \cdot \Theta(t, \beta_1, \beta 2) \cdot (V - \Omega_{ij}(t))_{1 \leq i \leq X, 1 \leq j \leq Y}$$

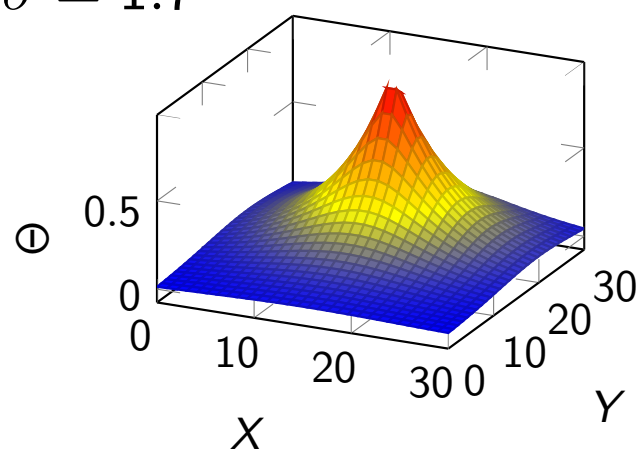
- Learning rate : weights the effect of the input vector during the training process.

The radius function

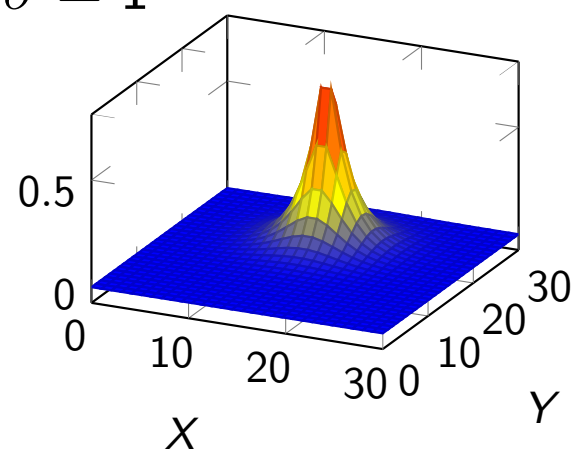
Neighborhood function Θ: the radius σ

$$\Theta(t, \beta_1, \beta_2) = \exp\left(-\frac{(i-\beta_1)^2 + (j-\beta_2)^2}{2\sigma^2(t)}\right)$$

$\sigma = 1.7$



$\sigma = 1$



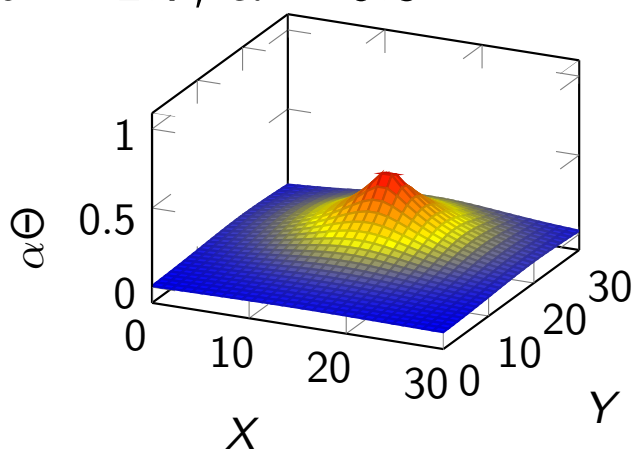
$$\sigma(t) = (\sigma(t_i) - \sigma(t_f)) \cdot \exp(-\frac{t}{\lambda}) + \sigma(t_f)$$

The learning rate

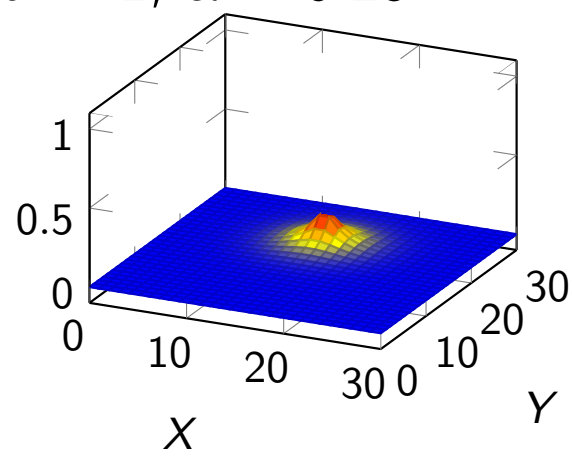
Neighborhood function Θ: the learning rate α

$$\alpha(t) \cdot \Theta(t, \beta_1, \beta_2)$$

$\sigma = 1.7$, $\alpha = 0.5$

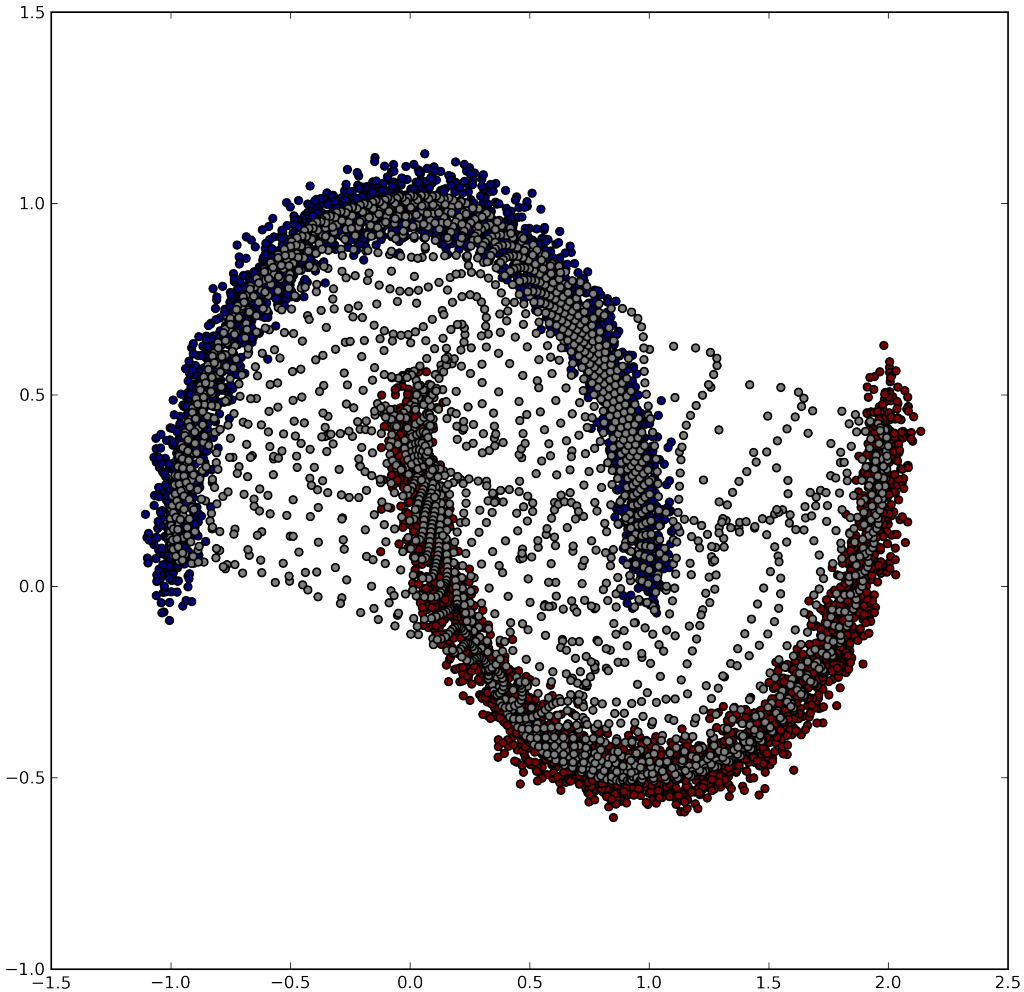


$\sigma = 1$, $\alpha = 0.25$



$$\alpha(t) = (\alpha(t_i) - \alpha(t_f)) \cdot \exp(-\frac{t}{\lambda}) + \alpha(t_f)$$

The trained SOM



Example of trained SOM with the moon dataset. The neurons are represented in the input space.

SOM parameters

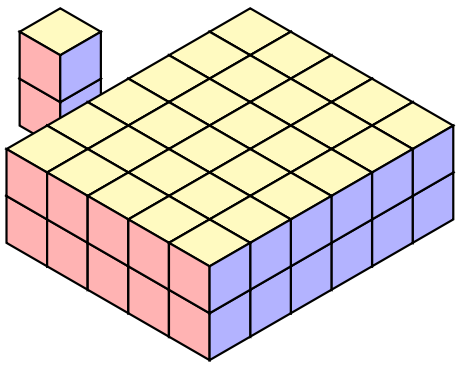
Map size A map size 50×50 is convenient to visualize the U-matrix and large enough to cluster large dataset.

Number of iterations Splited in **two phases**. The number of iterations is equal to the number of input data for the first phase and twice this number for the second phase.

Learning rate Starting from **0.5** and ending to **0.25** for the first phase and starting from **0.25** and ending to **0.0** for the second phase.

Radius Starting from **6.25** and ending to **3.0** for the first phase and starting from **4.0** and ending to **1.0** for the second phase.

How to visualize the SOM space?

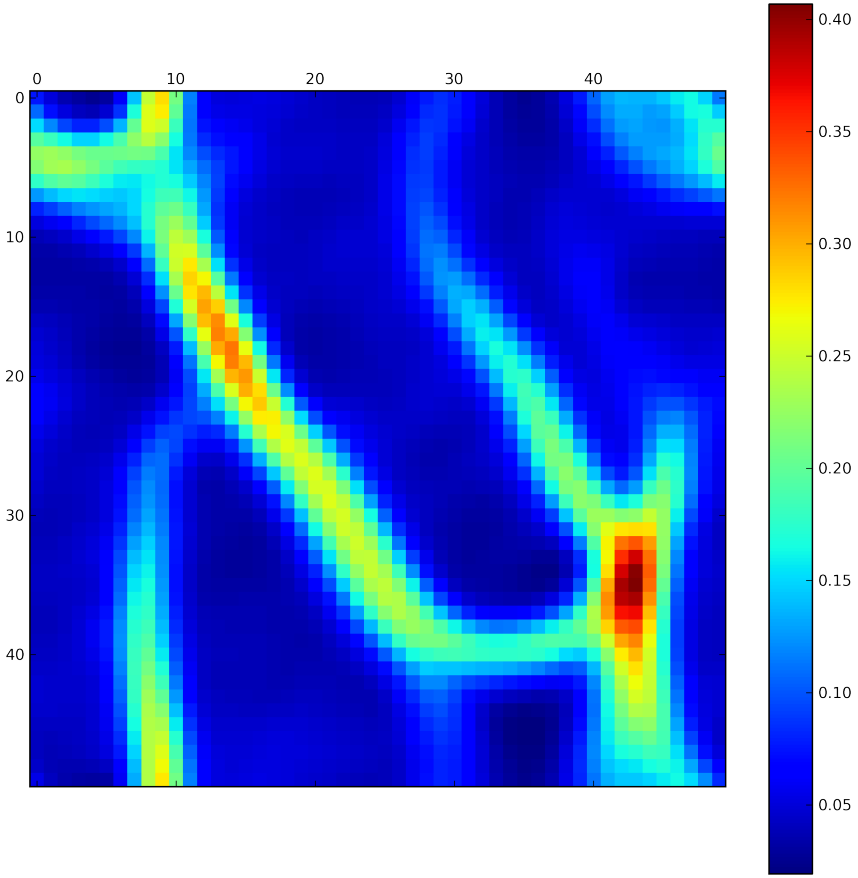


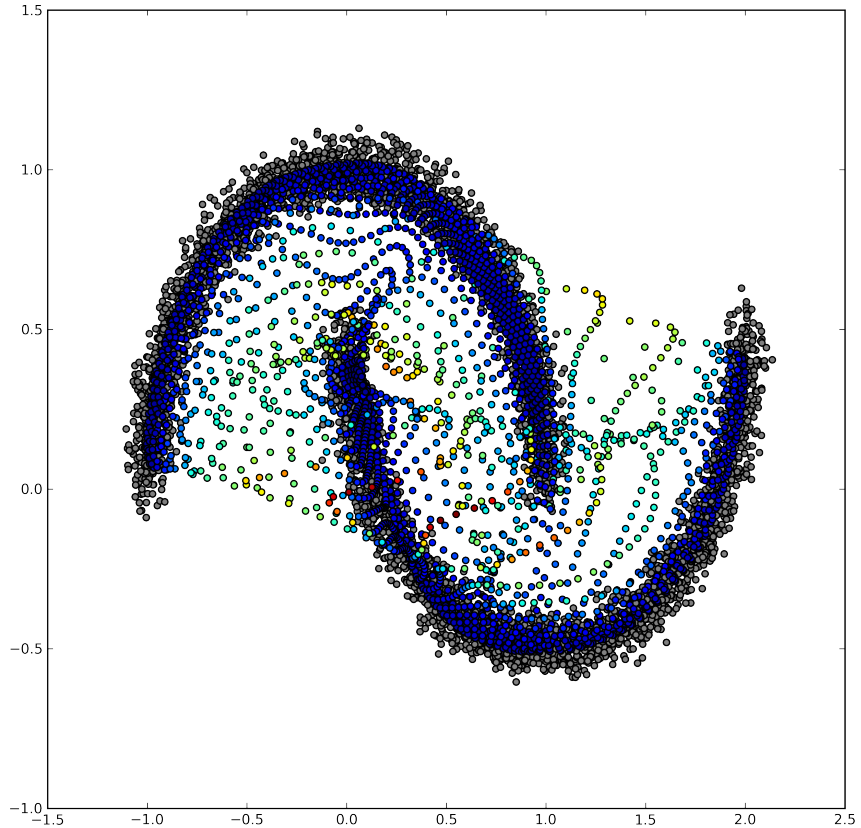
The U-matrix (Unified distance matrix) is the mean euclidean distance of each neuron with their eight neighbors.

$$U = \frac{1}{8} \sum_{\mu in N(\nu)} d(\nu, \mu)$$

It gives the topology of the map. It allows the identification of “natural“ clusters in the map.

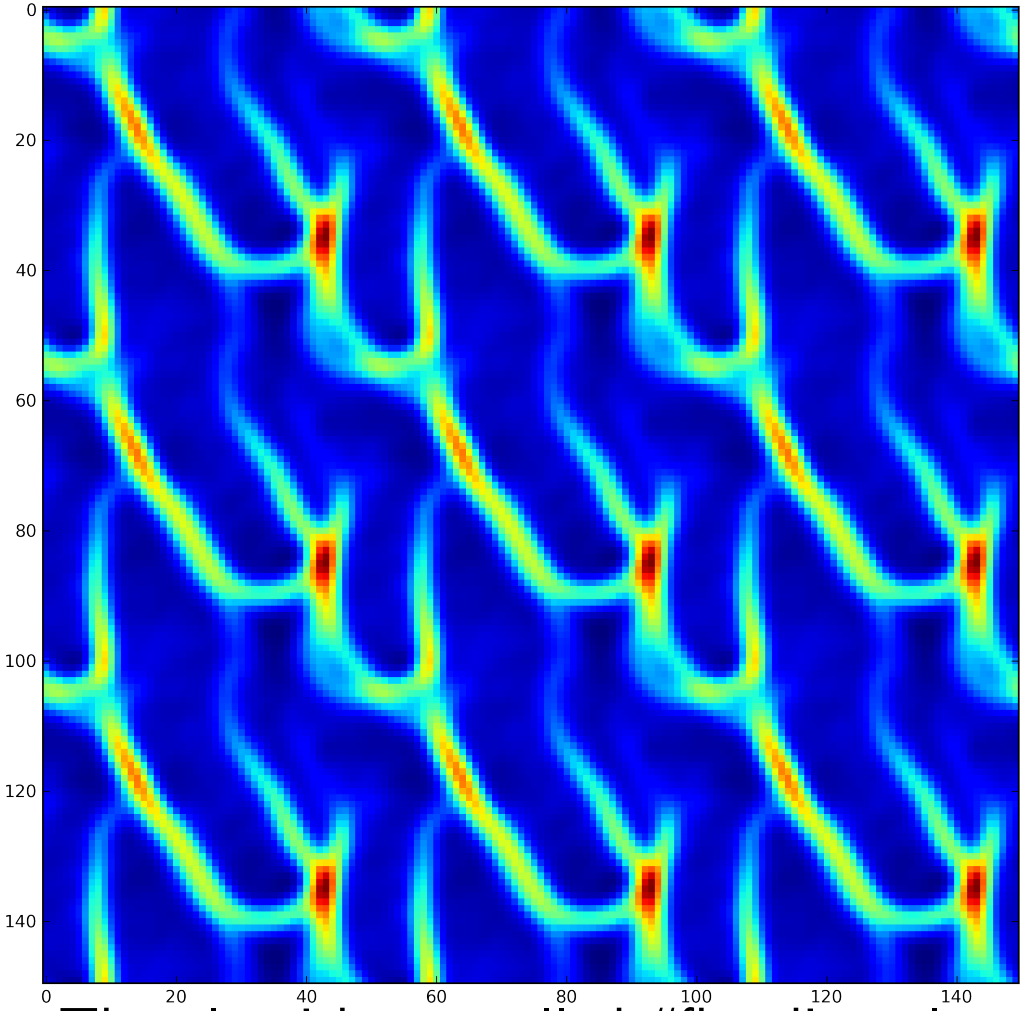
The U-matrix

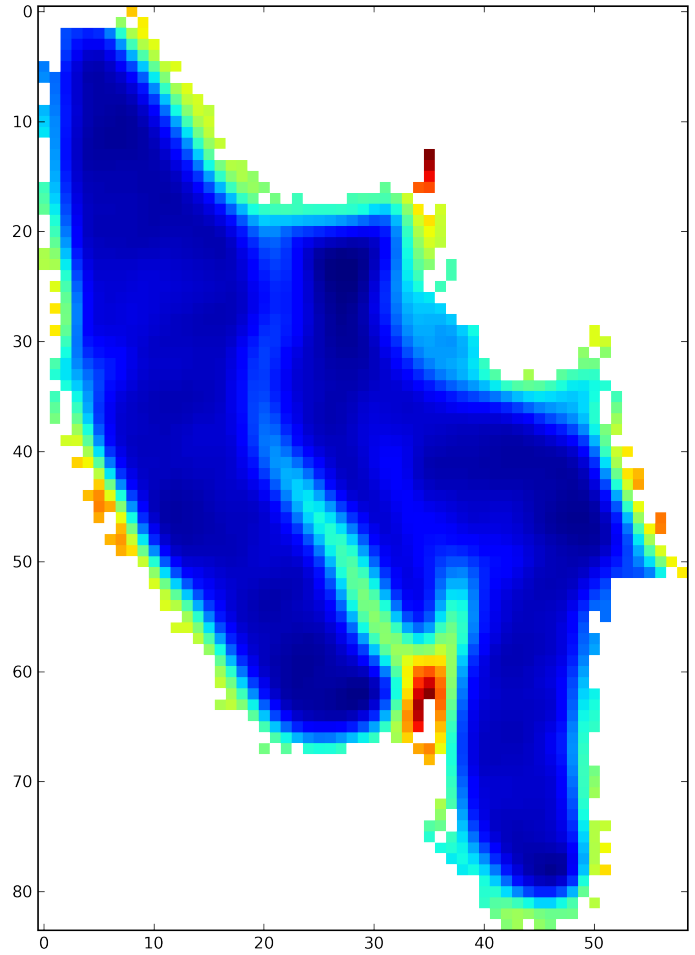




The U-matrix is a very conveniant tool to display the topology of the SOM. However the periodicity is not easily readable on the map.

Dealing with the donut!

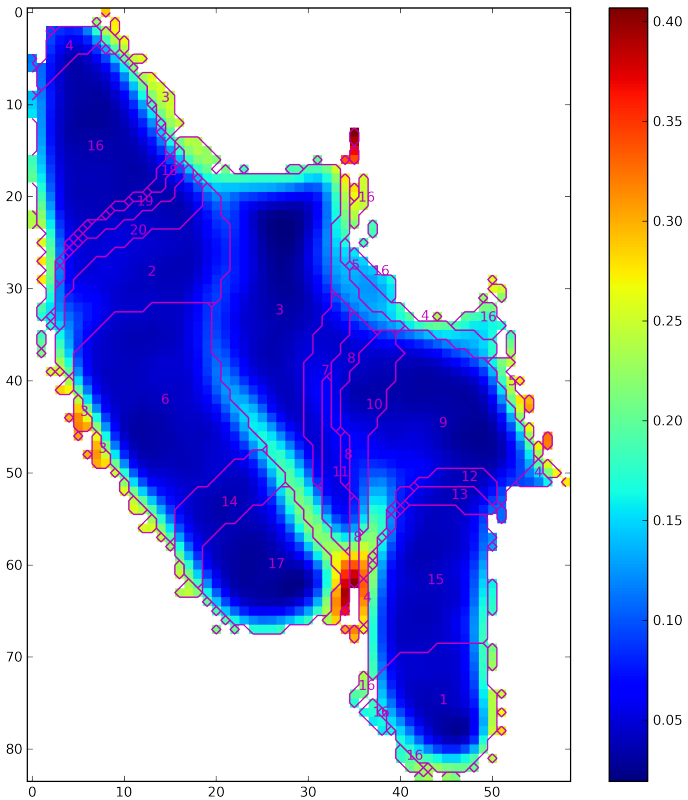


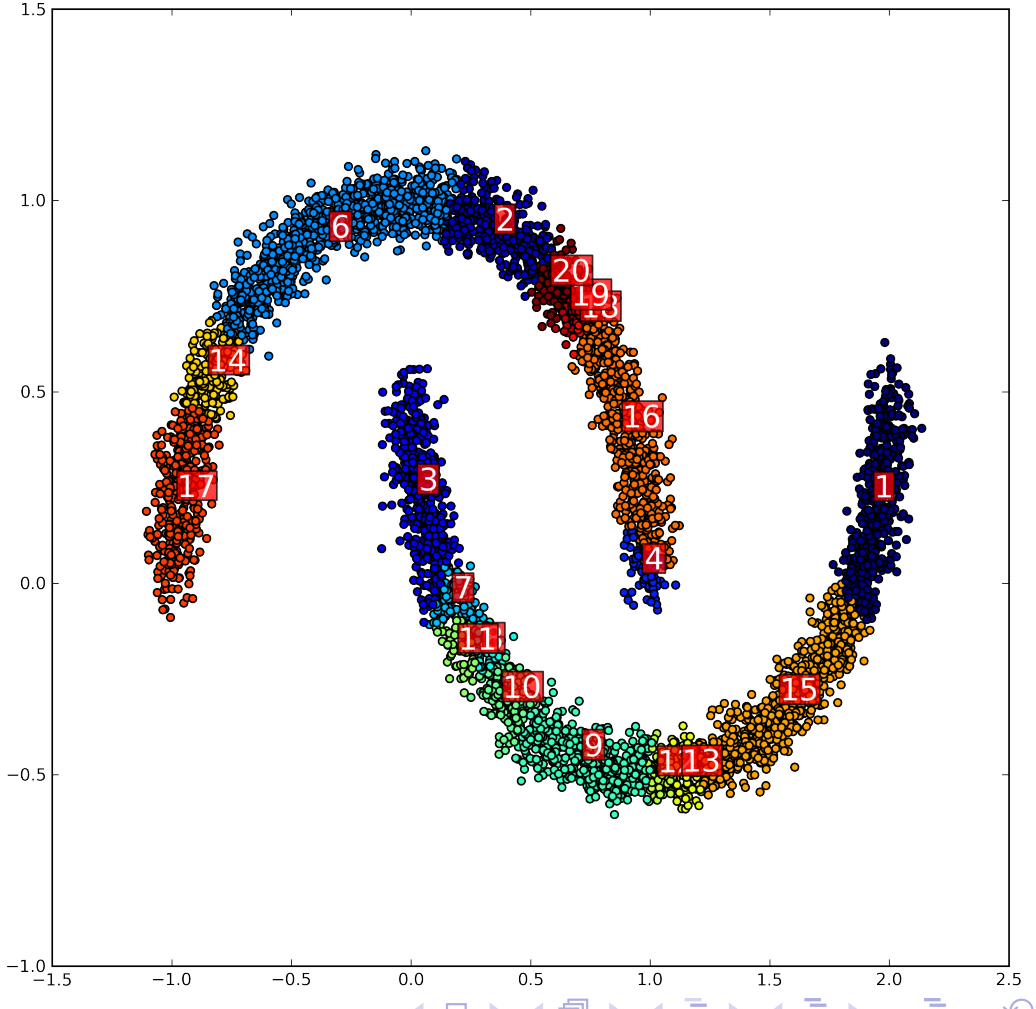


The algorithm – called "flooding algorithm" – starts from the global minimum of the U-matrix (many thanks to Mathias Ferber for this idea). It floods the map according to the relief of the U-matrix. This algorithm is inspired from the watershed algorithm.

Clustering the U-matrix

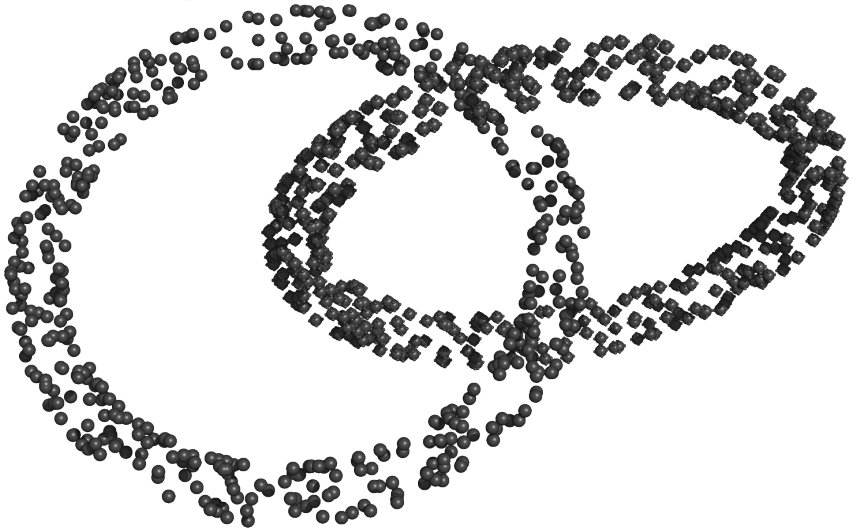
The main advantage of the flooding algorithm is to define cluster according to the flooding process. Each time the level goes down a new cluster is define. From this point of view we can define "natural cluster" without dealing with threshold definition!



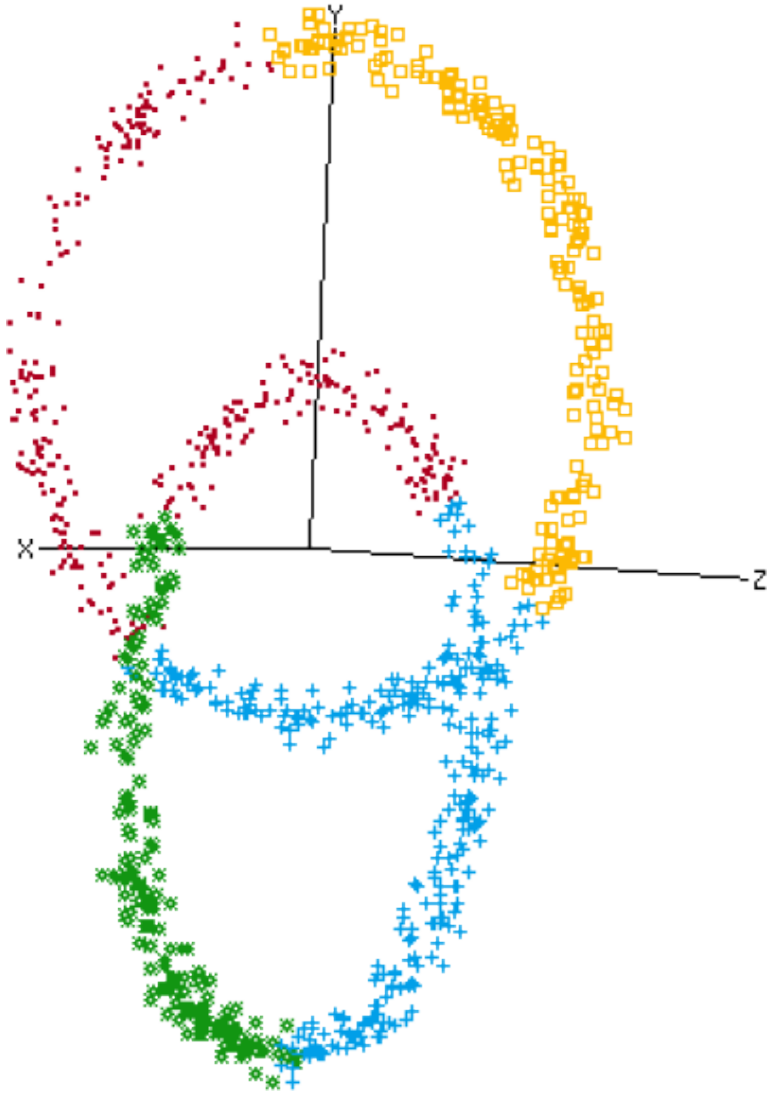


The chainlink benchmark

The input space

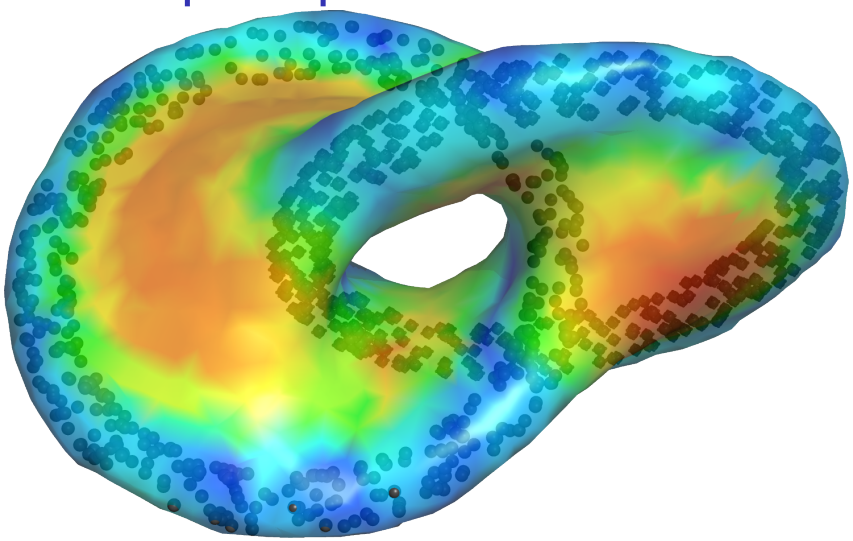


k-means clustering with $k = 4$

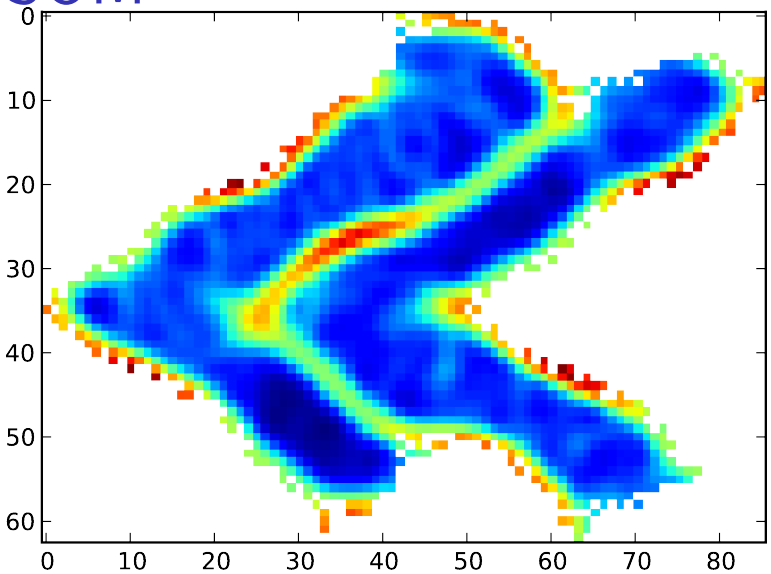


The chainlink benchmark

The input space

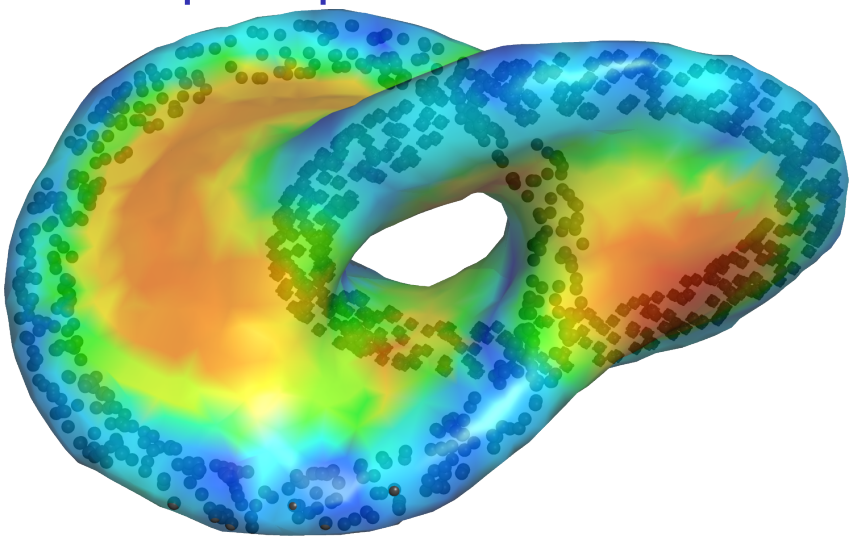


SOM

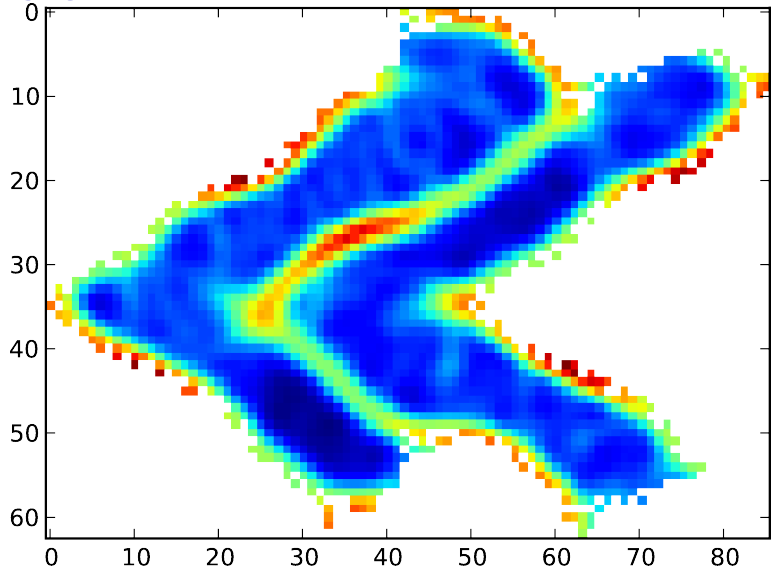


The chainlink benchmark

The input space



SOM



SOM preserves the topology of the input space

Plan

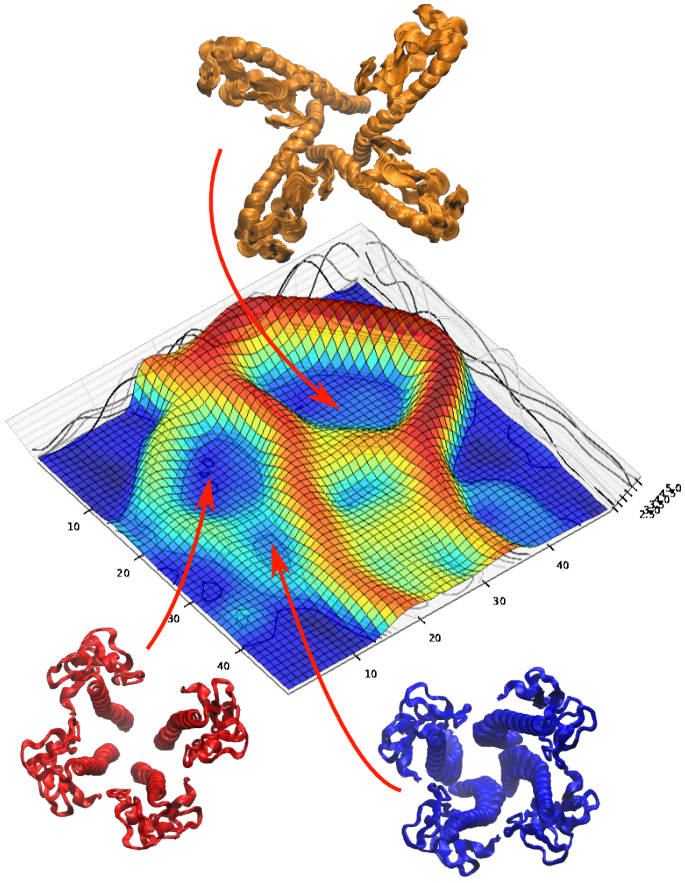
Introduction and theory

Application to protein conformation clustering

How to do in practice?

SOM basics

- SOM forms a semantic map where similar samples are mapped close together and dissimilar ones apart. This may be visualized by a U-Matrix (Euclidean distance between vectors of neighboring cells) of the SOM.
- Neurons are pointers to the input space. They form a discrete approximation of the distribution of training samples. More neurons point to regions with high training sample concentration and fewer where the samples are scarce.



$$\text{U-height}(\nu) = \frac{1}{8} \sum_{\mu \in N(\nu)} d(\nu, \mu)$$

How to describe a protein conformation?

Euclidean distance matrix

$A = (a_{ij})$; $a_{ij} = \|x_i - x_j\|_2^2$ where $\|x_i - x_j\|_2$ is the euclidean distance between the two atoms x_i, x_j

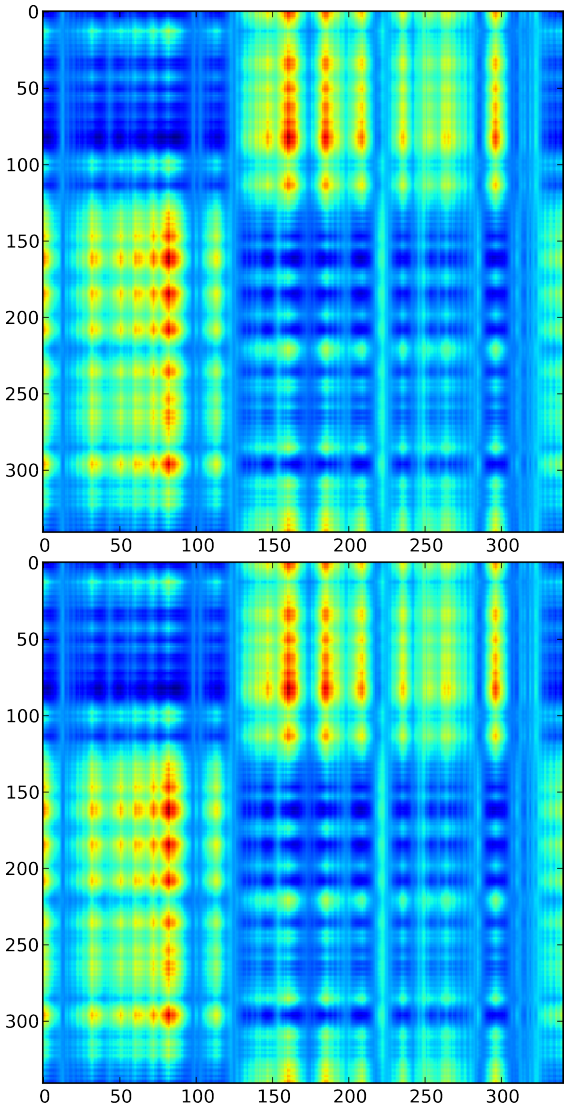
Spectral decomposition of distance matrix

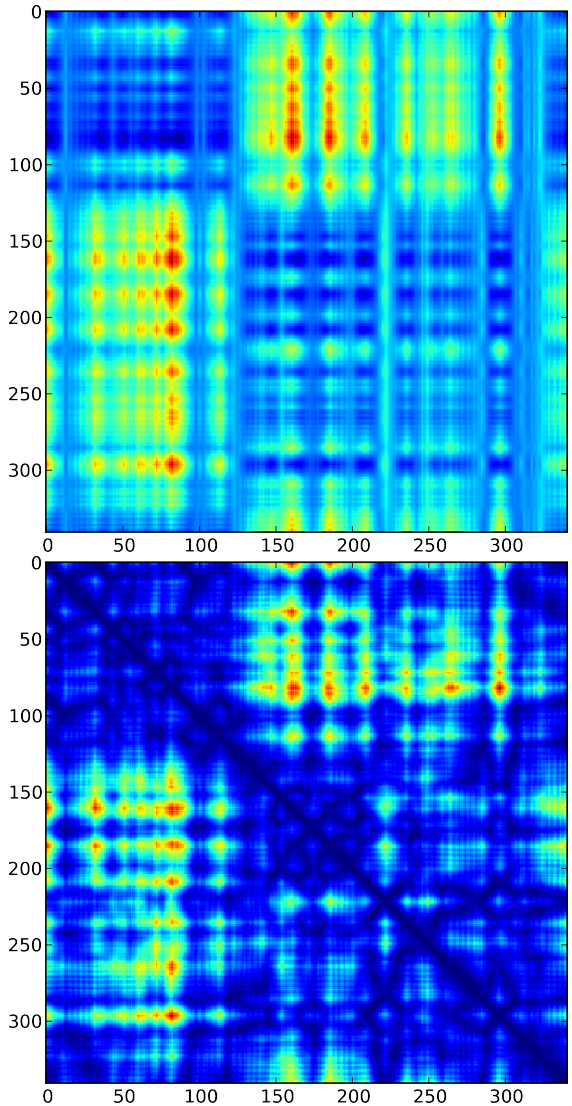
Decomposition based on eigenvalues of a square matrix A is called spectral decomposition. It allows us to express the original square matrix A of size N in $N \times N$ terms of its eigenvalues λ_k and corresponding eigenvectors v_k[1]

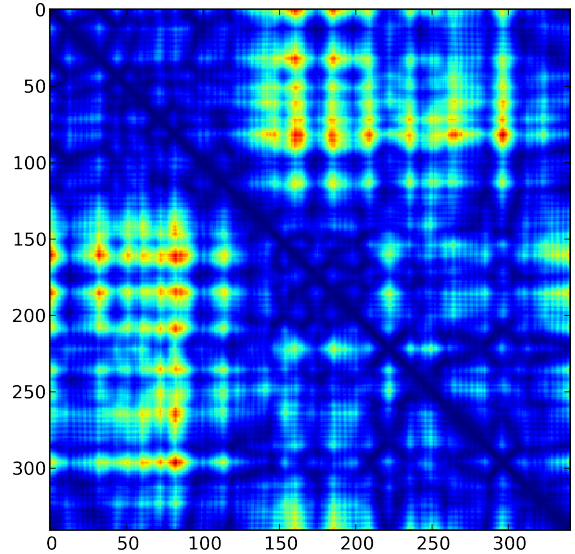
$$A = \sum_k \lambda_k v_k v_k^T$$

[1] J Struct Funct Genomics. 2009 March; 10(1): 67–81.

How many PCs to describe a distance matrix?







Matrix reconstruction of the original matrix above with 1, 2, 3 and 4 PCS.

Protein conformation descriptor

The distance matrix

The square distance matrix is too big to be an efficient descriptor: for a protein with 341 amino-acids (VanA), a C_α distance matrix gives a descriptor length of 57 970. With 25 000 snapshot the size of the input matrix is: $25\,000 \times 57\,970$

The first 4 PCs...

The first 4 PCs of the PCA are sufficient to describe the distance matrix. We obtain a descriptor with 1 364 elements and an input matrix with $25\,000 \times 1\,364$. The compression rate is 43.

Advantages

The clustering is not dependant of the alignment of the trajectory.

Resistance mechanism to vancomycin antibiotic

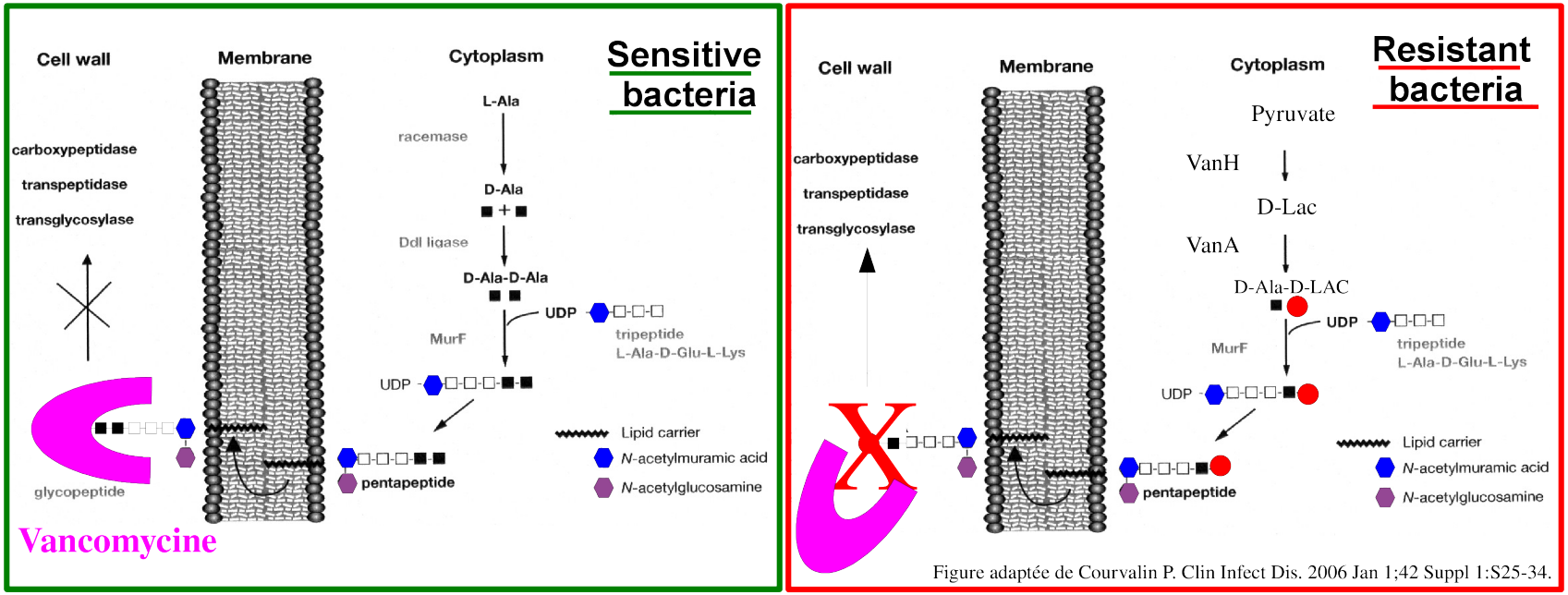
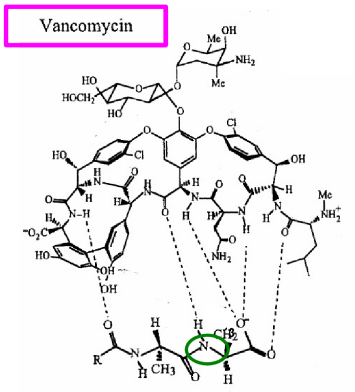
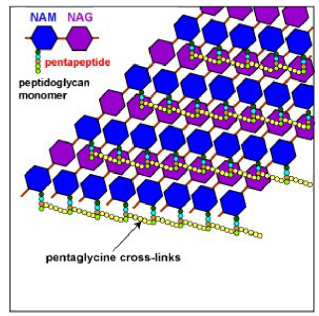


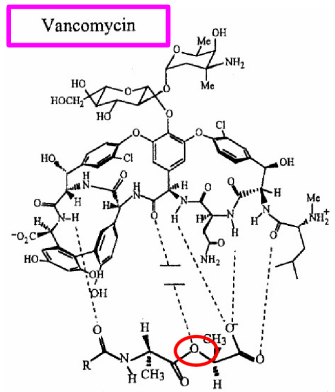
Figure adaptée de Courvalin P. Clin Infect Dis. 2006 Jan 1;42 Suppl 1:S25-34.



D-Ala-D-Ala

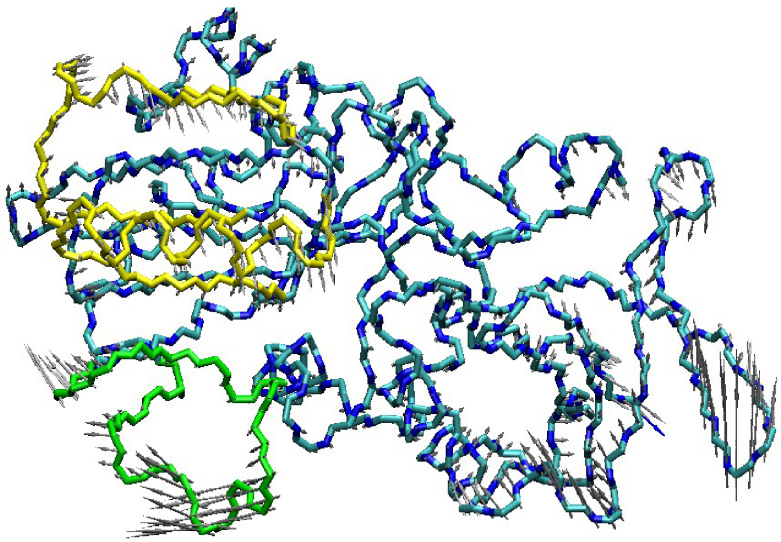


NAM : acide N-acétylmuramique
NAG : N-acétylglucosamine

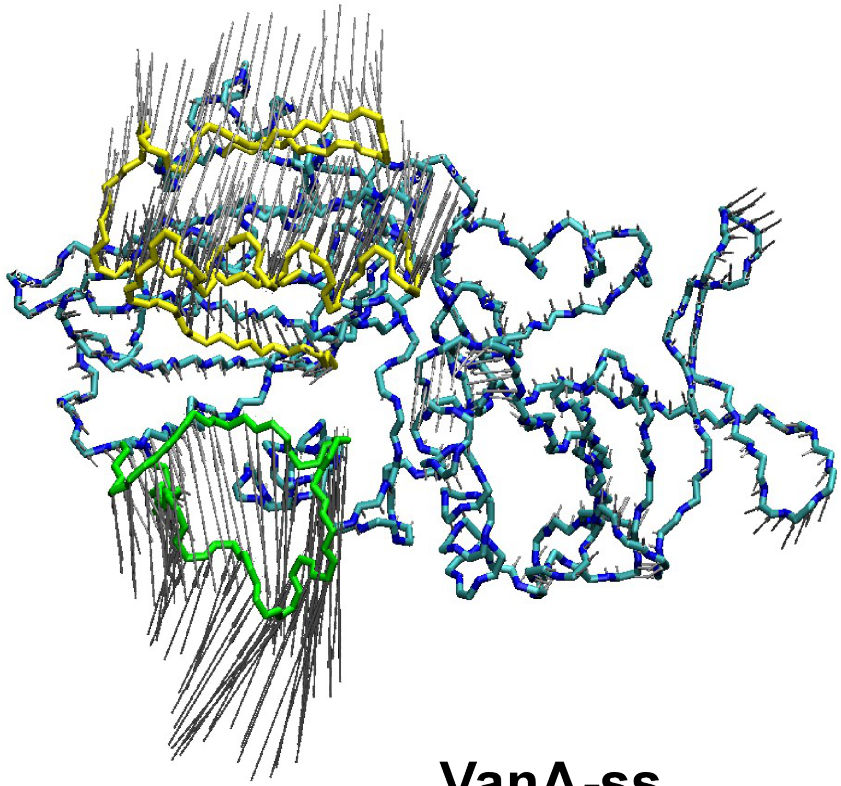


D-Ala-D-Lac

VanA molecular dynamics (25ns)



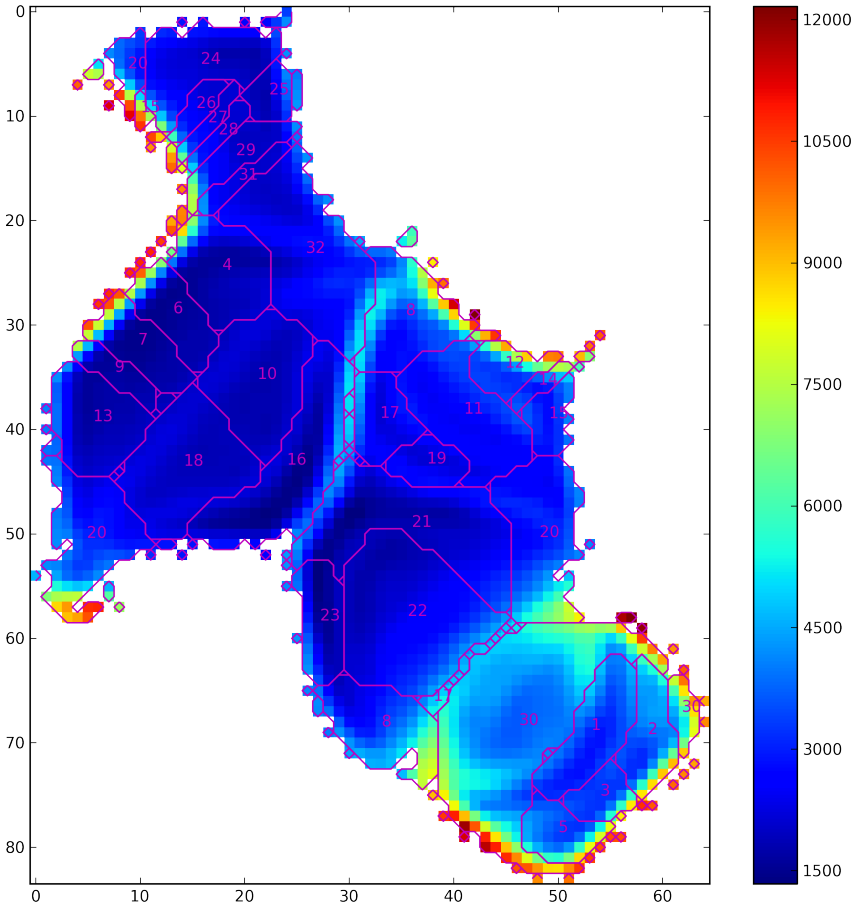
VanA



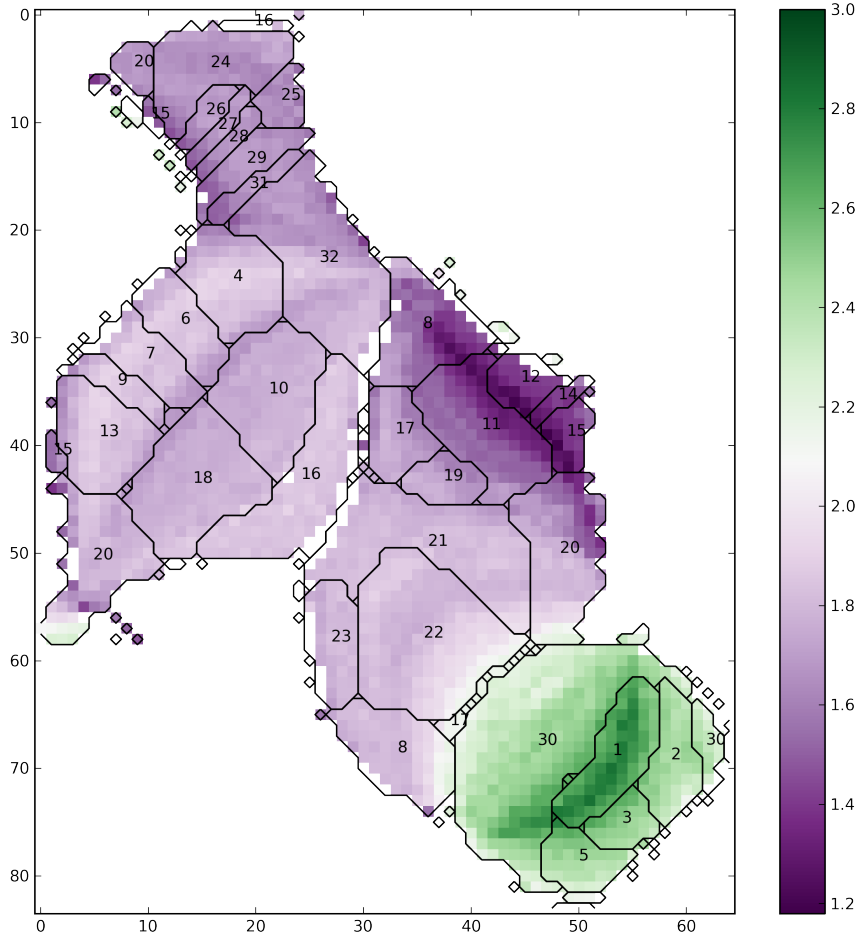
VanA-ss

PCA on VanA and VanA_{ss}.

SOM analysis of VanA MD

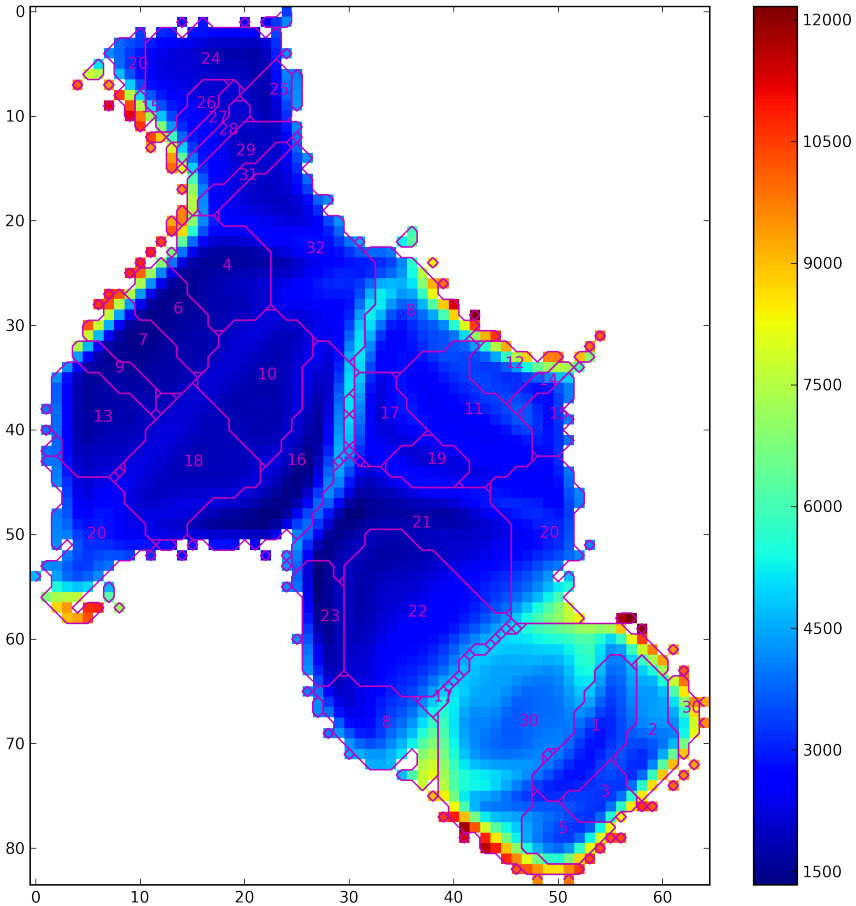


U-matrix of the VanA/VanA$_{ss}$ trajectory.

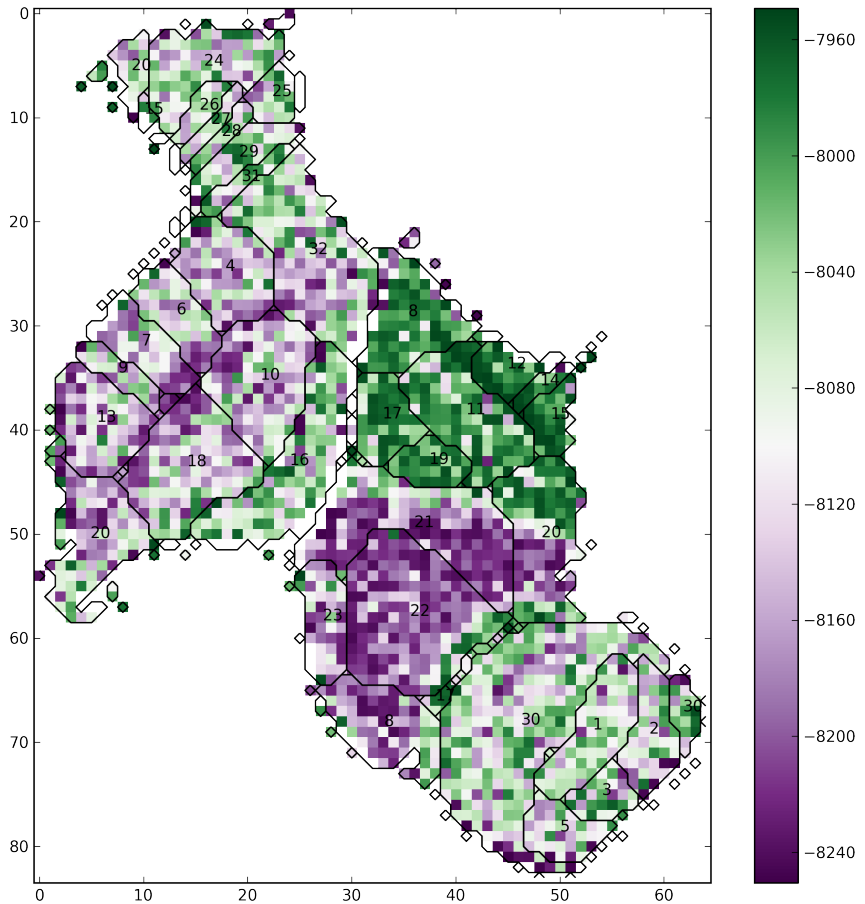


Projection of the RMSD on the SOM.

SOM analysis of VanA MD: projection of the MM-GBSA energies

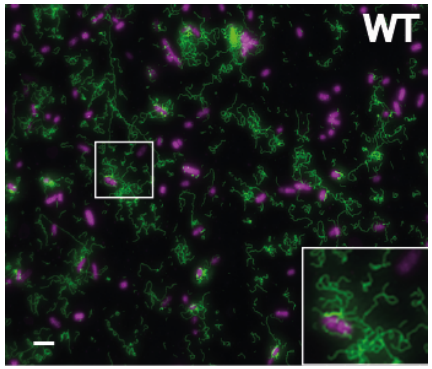


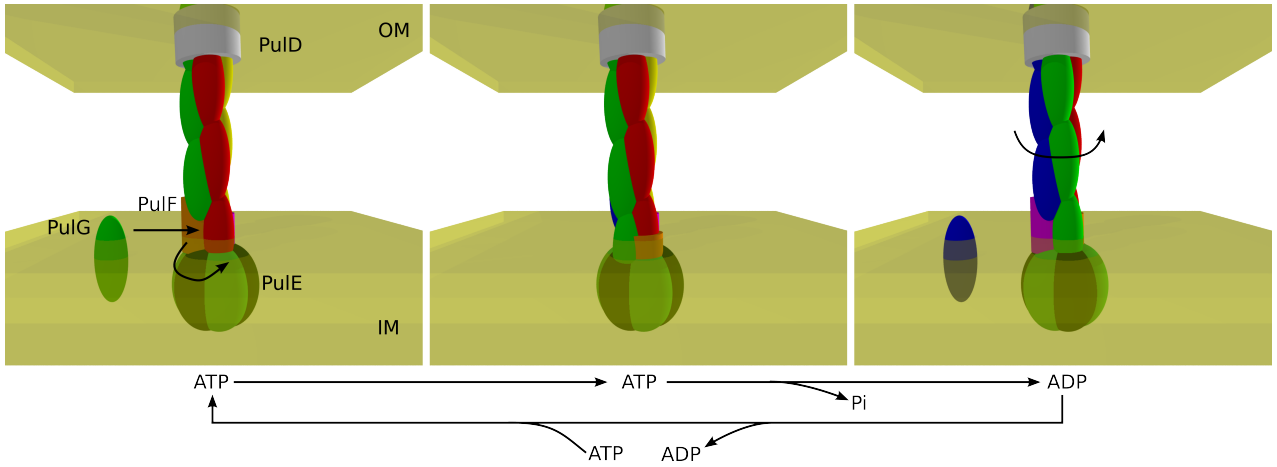
U-matrix of the VanA/VanA$_{ss}$ trajectory.

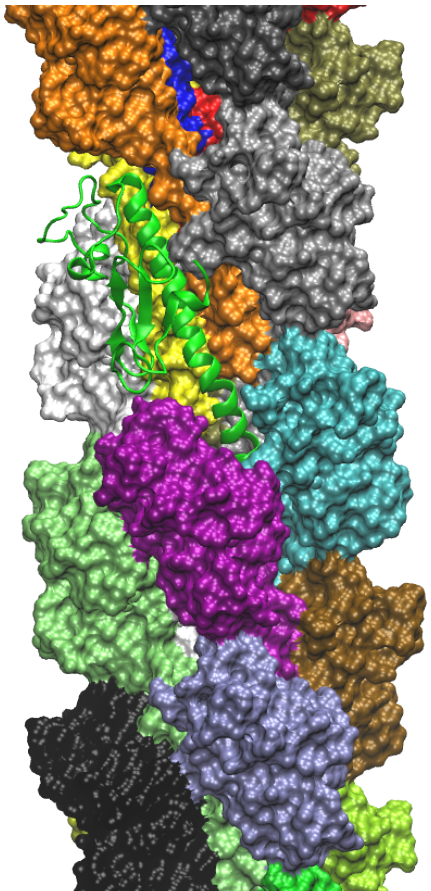


Projection of the MM-GBSA energies on the SOM.

pseudopilus: structure and function





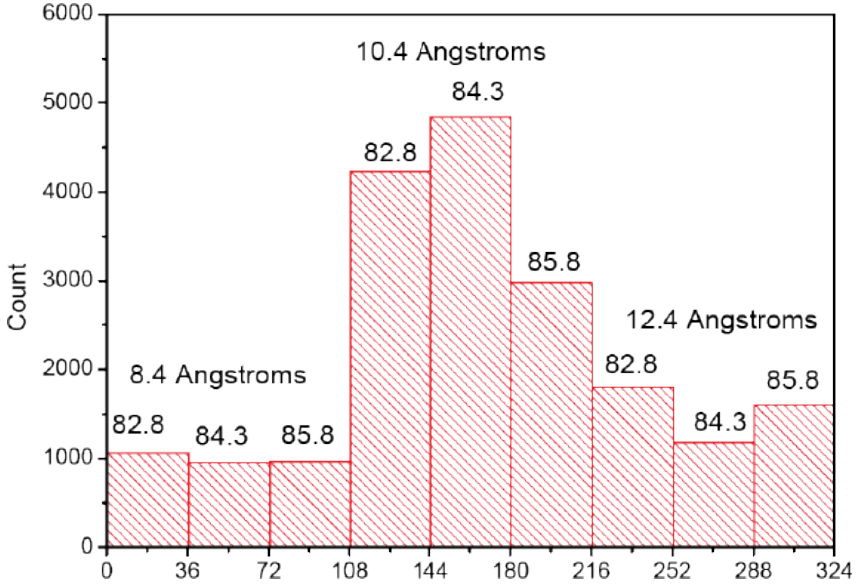


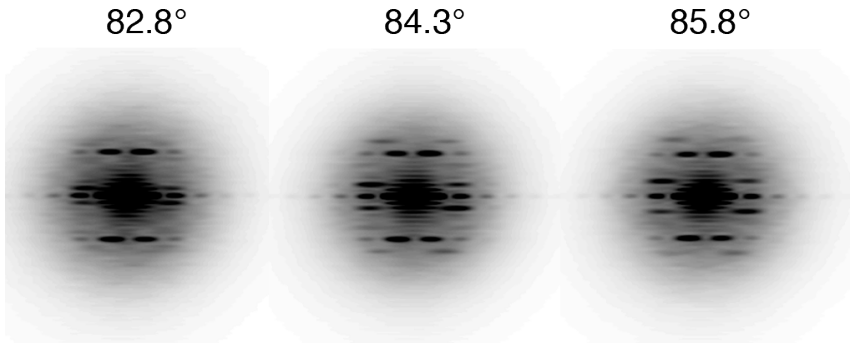
Conformational sampling

Pilus conformations generated by multi-stage minimization and molecular dynamic procedure in CNS using the CHARMM PARAM19 force field ...

From Campos et al. 2010

Variability of pulG filaments as seen by EM



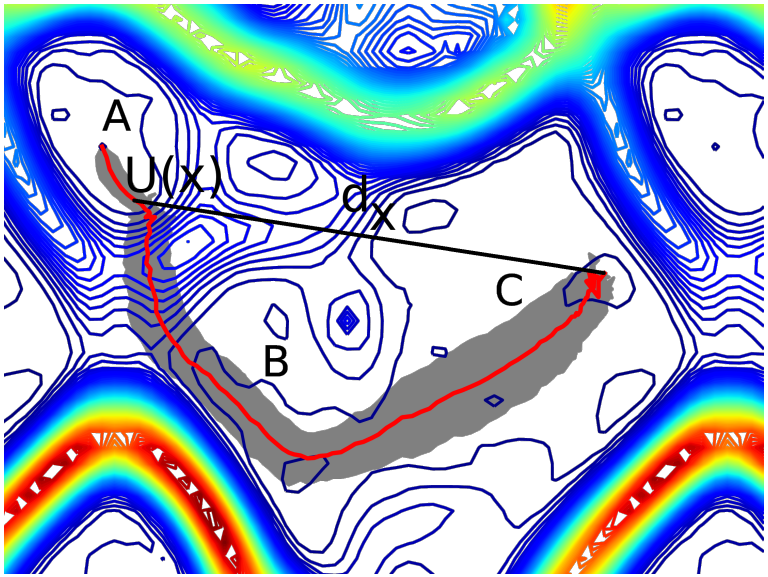


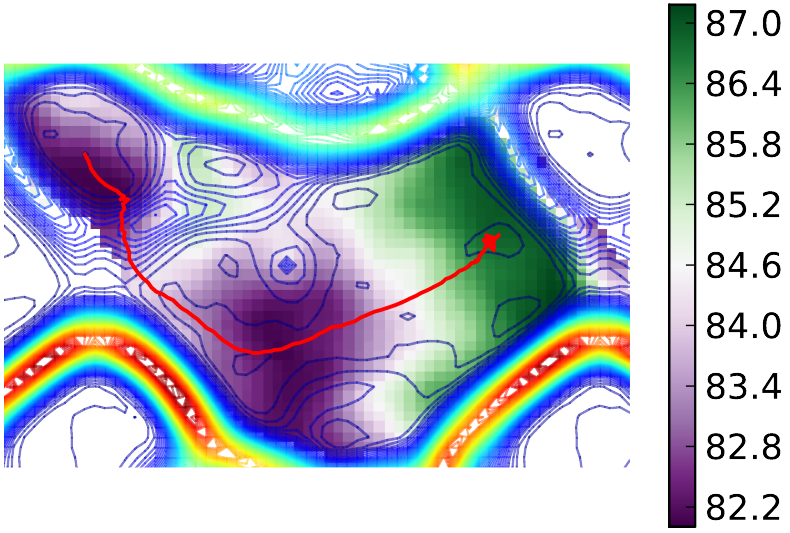
Conformational sampling

... during the multistage procedure a twist angle was randomly chosen between 81 and 88 degrees with uniform probability. 3901 models were obtained and clustered with SOM.
3D coordinates of one monomer and symmetry information were used as input vectors for the SOM.

From Nivaskumar, Bouvier et al. 2013

SOM clustering of pilus conformations





Projection of twist angle values

Path ($A \rightarrow C$) computation

MHMC algorithm:

$$P(x_{t+1} = x | x_t) = \min \left\{ \frac{\pi(x)Q(x)}{\pi(x_t)Q(x_t)}, 1 \right\}$$

with:

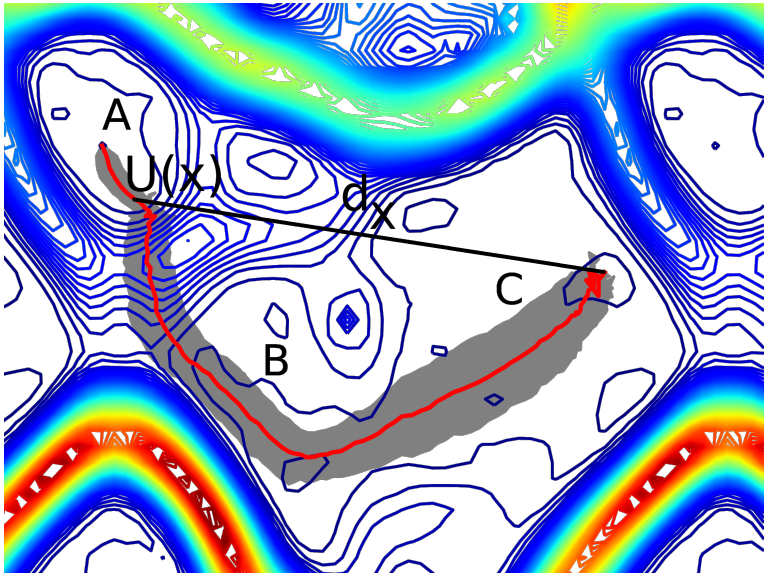
$$Q(x) = \exp(-d_x/k)$$

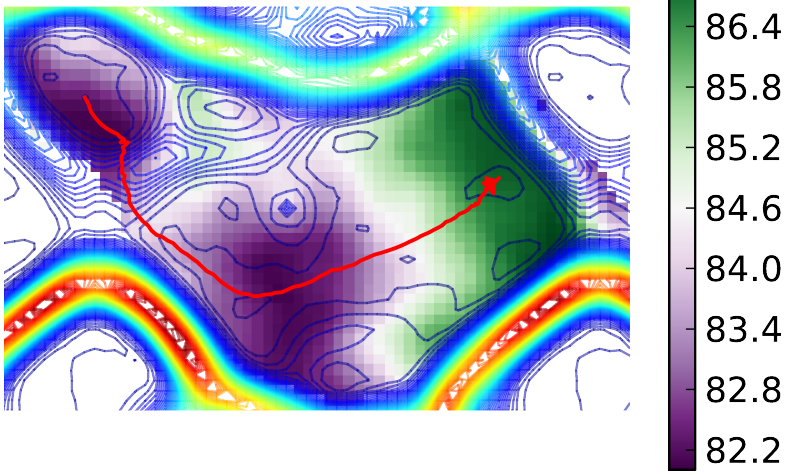
and:

$$\pi(x) = \exp(-U(x)/k')$$

An average on 100 possible paths was computed

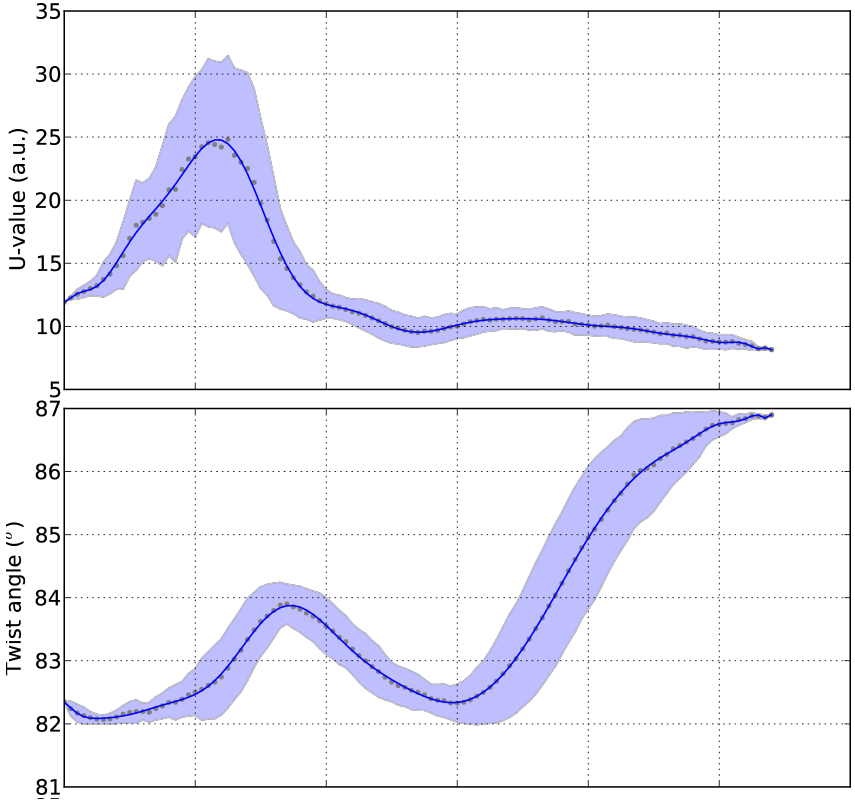
SOM clustering of pilus conformations



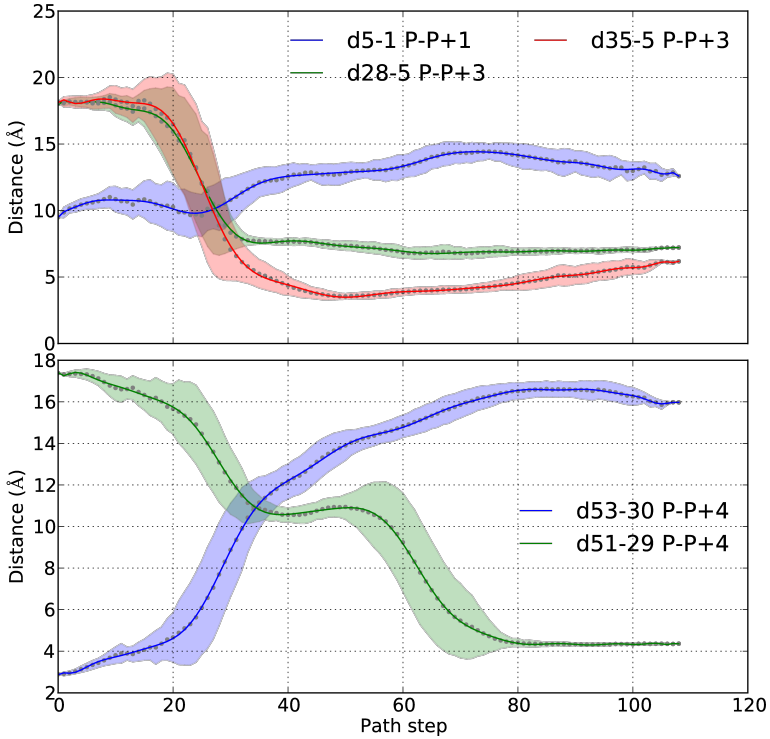


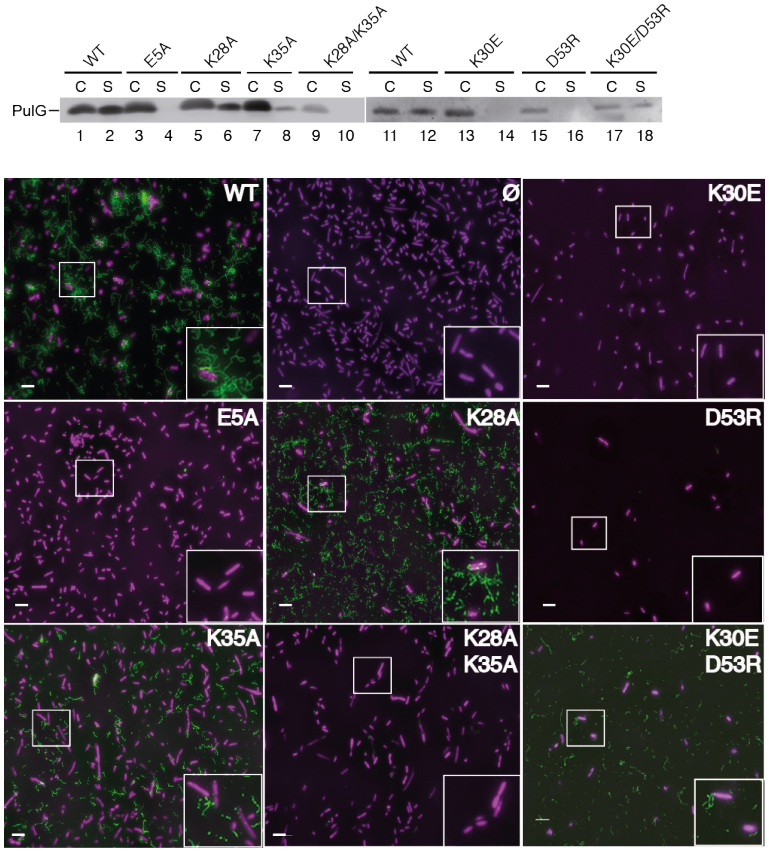
Projection of twist angle values

Path ($A \rightarrow C$) computation



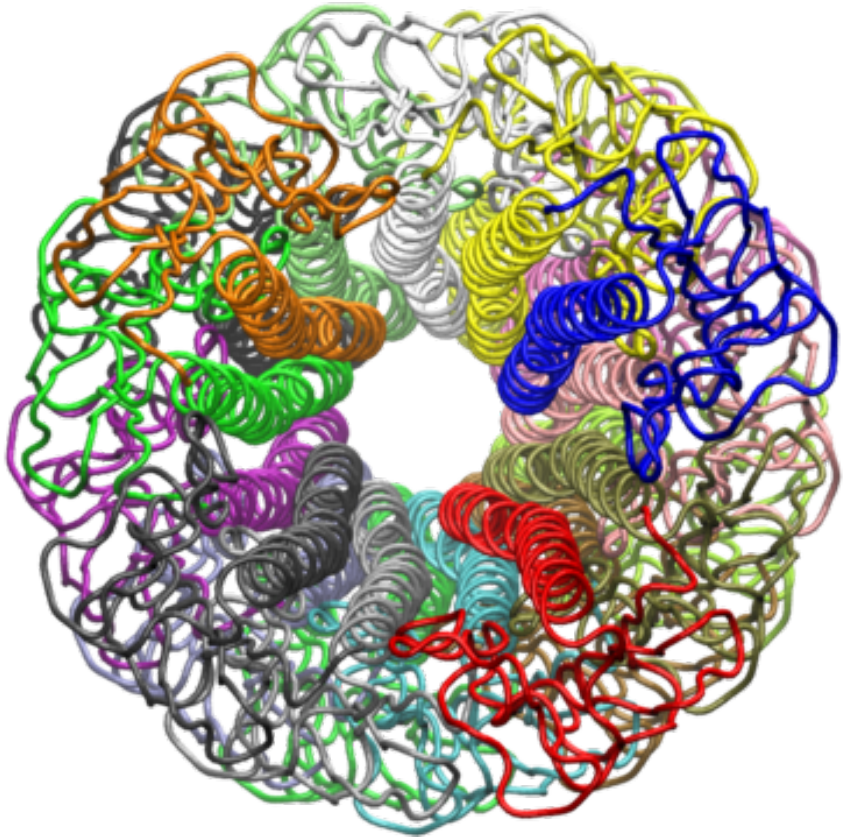
Experimental validation





From Nivaskumar, Bouvier et al. 2013

Reconstruction of the 3D structures from the path



Plan

Introduction and theory

Application to protein conformation clustering

How to do in practice?

Get the code

The code is on Github:
https://github.com/bougui505/SOM/tree/dev
And can be downloaded as a zip archive.
It's written in **python** and need **numpy** and **scipy** libraries.
Let's take a look on the code on Github.

Reading the trajectory

Home made implementation of dcd reader in python (named IO.py).

```
import IO
traj = IO.Trajectory('filename.dcd', struct='filename.pdb')
print traj.array.shape
(9249, 168)
#For a trajectory with 50001 frames and 168/3=56 atoms
print traj.struct.atoms
array([ (0, 1, 'CA', 'MET', 'X', 1, [26.659000396728516,
    32.76900100708008, 36.242000579833984],
    16.6599998474121, ''),
       (1, 2, 'CA', 'THR', 'X', 2, [28.715999603271484,
           29.84799957275390, 34.9510002136230],
           13.109999656677246, ''),
       (2, 3, 'CA', 'TYR', 'X', 3, [26.924999237060547,
           27.43899917602539, 32.5810012817382],
           8.869999885559082, ''),
...
      dtype=[('index', '<i4'), ('count', '<i4'), ('atomname'
          , '|S4'), ('resname', '|S3'), ('chain', '|S1'), ('
          resid', '<i2'), ('coord', '<f4', (3,)), ('beta', '
          <f4'), ('segid', '|S4')])
```

Computing the descriptors

A script called `makeVectorsFromdcd_PCA.py` in `application` does the job.

```
python makeVectorsFromdcd_PCA.py makeVectorsFromdcd_PCA.conf
```

with `makeVectorsFromdcd_PCA.conf`:

```
[makeVectors]
nframes: 9249 # Number of frames in the dcd file
structFile: 2pgb_hydro_ca.pdb
trajFile: newtraj78_210521_ca.dcd
projection: True
nProcess: 4 # Number of parallel process
```

The script creates 5 files:

- `eigenValues.npy`: contains the eigenvalues of the distance matrix.
- `eigenVectorsList.npy`: contains the eigenvectors.
- `meansList.npy`: contains the means over the lines of the distance matrix.
- `projections.npy`: contains the projections of the distance matrix on the first four eigenvectors (this is the file which will be used for the SOM training process).
- `reconstruction.npy`: contains the concatenation of the eigenvectors, means and projections, which is useful when one want to reconstruct structure from the descriptor.

Run the SOM

The easiest is to use ipython.

```
ipython
```

and

```
%pylab # To switch to pylab mode
Welcome to pylab, a matplotlib-based Python environment [
    backend: TkAgg].
For more information, type 'help(pylab)'.
import SOM2
inputmat = load('projections.npy') # Loading the input
    matrix
som = SOM2.SOM(inputmat) # Create the som object
som.learn(verbose=True) # Launch the learning in verbose
    mode with default parameters
...
0 8200 9249 88.66% 3.00045867515 0.250035282704 (5, 39)
0 8300 9249 89.74% 3.00041167015 0.250031666935 (7, 9)
...
# Displays the phase number (from 0), the iteration number,
    the total number of iteration, the progress, the radius,
     the learning rate and the BMU
```

However you can change the default, for example the learning rate:

```
som.learn(learning_rate=[lambda t, end_t, vector, bmu: som.
    _generic_learning_rate(t, end_t, 1, 0.5, 'exp'), lambda t, end_t, vector,
    bmu: som._generic_learning_rate(t, end_t, 0.5, 0., 'exp')], verbose = True)
```

SOM objects

The map is stored in: `som.smap`

```
print som.smap.shape
(50, 50, 224)
# 224 = 4*56 (56 is the number of atoms)
```

Don't forget to store the map in a numpy file:
`save('smap', som.smap)`, which will create the file `smap.npy` in the current working directory.

Analyzing the SOM

The easiest is to use ipython notebook:

`ipython notebook --pylab=inline` and to do some import:

```
import SOM2
import SOMTools # Tools to analyze SOMs
import SOMclust # Tool to cluster SOMs
import scipy
import IO # Tool to load the trajectory
pylab.rcParams['figure.figsize'] = 10, 10  # that's default
    image size for this interactive session
smap = load('smap.npy') # load the map already computed
inputmat = load('projections.npy') # load the inputmatrix
som = SOM2.SOM(inputmat) # into the som
som.smap = smap # and the smap too
traj = IO.Trajectory('newtraj78_210521_ca.dcd', struct='2
    pgb_hydro_ca.pdb') # and the trajectory
bmus = som.get_allbmus() # compute the Best Matching Units
print bmus.shape
(9249, 2) # Give the position in the SOM for each frame of
    the trajectory
umat = SOMTools.getUmatrix(smap) # compute the U-matrix
imatshow(umat) # To display the U-matrix
colorbar() # with a colorbar
```

Computing the density

The density is the number of input data per neuron. It can be easily calculated from the bmus:

```
X,Y,Z = smap.shape
density = zeros((X,Y), dtype=int)
for i,j in bmus:
    density[i,j]+=1
```

Computing the clusters

All is in SOMclust.py

```
clust = SOMclust.clusters(umat, bmus, smap) # will perform
    the flooding
clust.getclusters() # will perform the flooding based
    clustering
clust.plotclusters() # will plot the clusters
```

We can use directly the traj object to extract structures corresponding to clusters:

```
for e in unique(clust.labels):
    save('clust_%d'%e, traj.array[clust.labels == e])
```

and use the script `applications/npytodcd.py` to directly convert the npy file to a standard dcd file:

```
python npytodcd.py file.npy.
```

Projecting data onto the SOM

If you have data (RMSDs, energies) you want to project onto the map, you can use the BMUs to obtain the coordinate of each data on the SOM. This function below gives the mean value for each neuron.

```
def project(data):
    pmap = zeros((X,Y))
    for c,e in enumerate(bmus):
        i,j = e
        pmap[i,j]+=data[c]
    pmap = pmap / density
    return pmap
```