xPyder: a PyMOL plugin to analyze coupled residues and their networks in protein.

Marco Pasi[1], Matteo Tiberti[1], Alberto Arrigoni, Elena Papaleo

10th September 2018

1

1 Plotting matrices: tutorial for the impatient user

This is a quick-and-dirty, hands-on tutorial which guides the user through few essential steps to obtain a publication-ready figure with xPyder.

What you need:

- The PyMOL software and a proper installation of xPyder;
- A matrix in a xPyder-readable format (from now on, the "matrix file"). A simple text representation, with space-separated values and a newline character at the end of each matrix line will work fine.
- A PyMOL session in which the structures you need to plot on have been loaded. For structures composed of more than one chain, each one should have a proper and unique chain identifier. If this isn't the case you should use the PyMOL "alter" command to ensure this.

What you get:

- A visual representation of the relationships between residues encoded in the loaded matrix.

Here are the steps.

1. Open PyMOL and load the session or structure(s) you need to plot on. No other structure should be present. Load xPyder from the Plugins menu.
2. On the Matrix tab, click on the "Choose matrix file..." button. Choose the matrix file and click Ok. Click the Load matrix button. The matrix will be loaded and an histogram plot of the data distribution will be shown on the right.

[1]the authors equally contributed to this work

3. (Optional, but highly recomended) Activate the Sequence distance and Matrix values filters, by clicking on their respective checkboxes in the Filters control section, on the right panel of the xPyder interface. The red boxes at left of the Filters' names will turn green, meaning that the filters are now active.

4. Click on the "Plot" button

Done! A number of red and blue cylinders have now appeared on the reference object, joining couples of residues whose matrix elements were positive or negative, respectively. You may save your session for future reference or obtain an image by using the standard save PyMOL command, maybe after rendering it with your favourite ray tracer. If you followed step 3, some of the least meaningful matrix values have been filtered out, and do not appear in the final representation.

2 What is xPyder

xPyder is a PyMOL plugin originally conceived to analyze the cross correlations between residues as measured from molecular dynamics (MD) simulations, that is in fact capable of analyzing pairs of communicating or interacting residues, or any pairwise metric that is representable as a 2D matrix.

xPyder takes advantage of PyMOL's excellent molecular viewing capabilities to display in a simple and customizable way the correlations on the 3D structure, providing structural insight in the dynamical information contained in the correlation matrices that is otherwise hard to obtain, and to produce publication-quality images to capitalize on such information. In order to take full advantage of the often very large amount of information contained in the correlation matrices, xPyder features a full-fledged set of cascading filters that allow the user to refine the plotted data on the basis of their value or of the features of the structure they will be plotted on (such as secondary structure and distance between residues), as well as on the basis of user-defined PyMOL selections. Furthermore, these filters are independent modules that are easy to implement, and that plug in to xPyder to provide their functionality, making it possible for the users to add custom filters that exactly match their needs, giving xPyder the definitive customizability that is required for innovative scientific analysis and imaging applications. Finally, xPyder extends and speeds up the analysis pipeline by integrating, in its first version, two useful analyses, "Chained" and "Delta", making them easily accessible through its simple interface.

3 Installation

3.1 Requirements

xPyder is very easy to install, and has few dependencies (in most cases, just PyMOL). This is the required software list for xPyder to work properly.

- python 2.7
  (http://www.python.org)
- PyMOL $\geq$1.4.0
- numpy libraries $\geq$ 1.5.1
  (http://sourceforge.net/projects/numpy/files/NumPy/)
- matplotlib libraries $\geq$1.1
  (http://sourceforge.net/projects/matplotlib/files/matplotlib/)
- networkx libraries $\geq$ 1.5 (already included in the installation - you don't need to download them)
- Tkinter libraries >8.4 with ttk support. As some Python 2.6 distribution don't come with ttk included, we provided a fallback included in the xPyder package. This is usually the case for the MacPorts OS X install.

In our installation tests, we found the local Python and PyMOL and python environment to vary considerably among different operating systems and linux distrubitions. For example, you may need to install the corresponding Python 2.6 libraries if your PyMOL distribution uses version 2.6.

If PyMOL 1.4 is not available for your operative system as a pre-packaged installer from Schrodinger, you can install it in other ways.

- In linux you should refer to the standard package infrastructure of your distribution or compile it from scratch.
- In OSX you can install the version distributed with MacPorts
  (http://www.macports.org/).
- In Windows, you can follow the tutorial on the PyMOL wiki
  (http://www.pymolwiki.org/index.php/Windows_Install).
  Briefly, it involves downloading and installing Python 2.7 from the official site and the unofficial pre-compiled PyMOL installer from Christoph Gohlke
  (http://www.lfd.uci.edu/~gohlke/pythonlibs/#pymol).

3.2 Installation

Once PyMOL and libraries are present on your system, the installation procedure is straightforward.

1. If you have git installed in your system, clone the xPyder repository on your local machine:

   ```
   git clone https://github.com/ELELAB/xpyder.git
   ```

   Alternatively, you can manually download and decompress the zip package from the GitHub repository in a local directory. In either case you should end up with a directory containing the files required for xPyder to function.

2. Move the directory obtained in the previous step to a known location (For example, `/home/myuser/xPyder`; this will be where support files for xPyder will be stored.

3. Enter the just created directory and open the `xPyder.py` file with your favorite text editor, as vim, Notepad, Sublime text or others.

4. Look for a line starting with "`INSTALLDIR`". It should be around line 18.

5. replace `INSTALLDIR_PLACEHOLDER` with the absolute path of the xPyder installation directory. Do not touch anything else. For exemple, modify

   ```
   INSTALLDIR=r"INSTALLDIR_PLACEHOLDER"
   ```

   to

   ```
   INSTALLDIR=r"/home/myuser/xPyder"
   ```

6. Save the file and exit the editor.

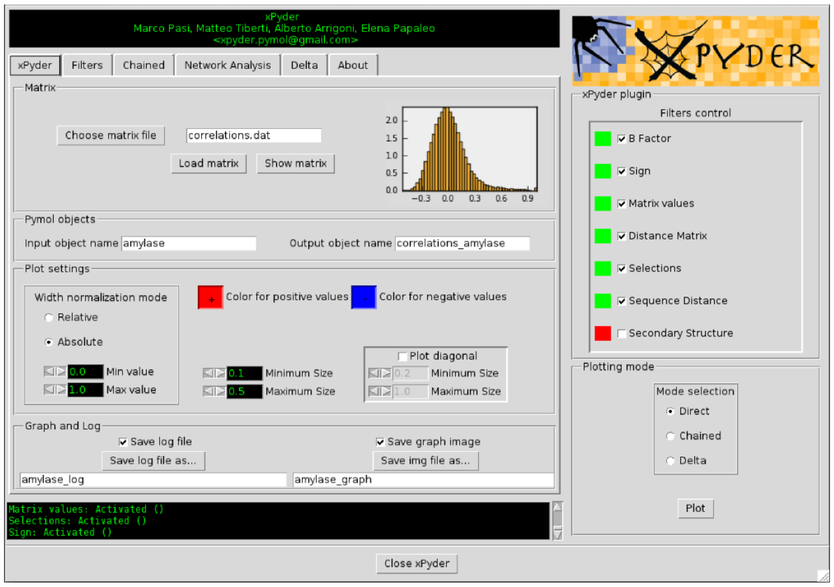
7. Install `xPyder.py` as you would have done with any other PyMOL plugin. Open PyMOL, click on the Plugins menu, Manage Plugins, Install. Choose xPyder.py. Restart PyMOL as required.

4 The Plugin

4.1 The xPyder main tab

4.1.1 Matrix section

Allows the user to load the matrix ($M_{i,j}$) that encodes the relationship to be plotted on the 3D structure. The matrix must be a $N \times N$ symmetrical matrix, where N corresponds to the number of residues in the analyzed PyMOL object (**Input object**, see on). The matrix file must be a plain-text file consisting of N lines, with N whitespace-separated floating-point numbers on each line. The



Matrix: correlations.dat

Input object name: amylase
Output object name: correlations_amylase

Width normalization mode: ○ Relative [✓ ● Absolute]
Min value: 0.0
Max value: 1.0

+ Color for positive values 🟥
- Color for negative values 🟦

Minimum Size: 0.1
Maximum Size: 0.5

☐ Plot diagonal
Minimum Size: 0.2
Maximum Size: 1.0

✓ Save log file: amylase_log
✓ Save graph image: amylase_graph

```
Matrix values: Activated ()
Selections: Activated ()
Sign: Activated ()
```

🟩	✓ B Factor
🟩	✓ Sign
🟩	✓ Matrix values
🟩	✓ Distance Matrix
🟩	✓ Selections
🟩	✓ Sequence Distance
🟥	☐ Secondary Structure

Mode selection: [✓ ● Direct] ○ Chained ○ Delta

Figure 1: xPyder's main tab

entry in the i-th row and the j-th column of the matrix will correspond to the j-th number present on line i.

The order of the matrix elements is assumed to be that of the residues in the corresponding PyMOL structure, i.e. field j of line i of the matrix file describes correlations between residues i and j in the PyMOL object. The user should make sure of this correspondence when using xPyder. An easy way to do this is by using the **Save log file** functionality (see below).

Choose matrix file: The file containing the matrix can be specified either by using the file selection dialog that appears by clicking on the button, or by typing its full path in the adjacent field;

Load matrix: This loads the matrix from the specified file, and plots a histogram of the loaded values; by clicking the histogram, a larger version of the histogram appears in a popup window, featuring the useful **matplotlib** toolbar, which allows, among other features, to save the histogram to an image file (for more information see the matplotlib help http://matplotlib.sourceforge.net/users/navigation_toolbar.html).

4.1.2 Pymol objects

The user can specify via the **Input object name** field, to which PyMOL object or selection the loaded correlations refer to. The field must contain a valid PyMOL object or selection name, and the number of residues in the object/selection will be used to validate the matrix input (see above, **Matrix**).

Since the correspondence between the matrix and the PyMOL object/selection is done merely on the basis of the order of residues, such correspondence should be checked by the user (see below, **Save log file).**

The 3D plot of the relationship encoded in the matrix is saved in the PyMOL object specified by **Output object name**: if such object exists, the plot is appended to the object as its next state; otherwise the object is created.

4.1.3 Plot and visualization options

Allows the user to access xPyder's visualization options. These have been designed in order to allow the user to customize the colors and the size of the cylinders used for the plot of inter-residue relationships (off-diagonal values of the matrix), and of the spheres used for the plot of intra-residue relationships (the values on the diagonal of the matrix).

Minimum/Maximum size: allows the user to define the size of the cylinders to be plotted: relationships with absolute value of m_{min} will be displayed with cylinders of width **Minimum size**, while relationships with absolute value of m_{max} will be displayed with cylinders of width **Maximum size**; the sizes for the values in between are linearly interpolated.

Width normalization mode: allows the user to define how xPyder defines the values of m_{min} and m_{max}. If this option is set to "Absolute", these values are specified by the user using the Min value and Max value fields. The values extracted from the matrix file are therefore processed in order for their absolute value to be included in the specified range. If the loaded matrix contains values which lie out of this range, their absolute value is coerced to the nearest of the two extremes just before plotting them, and a warning is issued to inform the user. Please note that for Delta matrix plots (see below), the range in Absolute mode is implicitly set to $[0; 2m_{max}]$. If otherwise this option is set to "Relative" (default), xPyder automatically identifies the minimum and maximum (absolute) values of the matrix and assigns them as $m_{min}$and m_{max}.

Color for positive/negative values: by pressing the "+" and "-" buttons two menus are displayed that allow the choice of plot colors.

Plot diagonal: if checked, the user can select a range size for the sphere representation of the elements that lie on the diagonal of the loaded matrix (please see **Minimum/Maximum size** above).

4.1.4 Graph and Log

The user can optionally save a 2D graph of the obtained 3D plot in an encapsulated postscript (eps) file by checking the **Save graph image** checkbox. The graph is drawn by finding a best 2D layout for the 3D plot by using the

Fruchterman-Reingold algorithm (see the relevant page on the networkx website: http://networkx.lanl.gov/reference/generated/networkx.spring_layout.html).

Furthermore, a plain-text list of the cylinders and spheres plotted can be obtained by checking the **Save log file** checkbox. This list contains a line for each of the plotted cylinders, that describes the corresponding relationship $m_{i,j}$ using 5 fields:

1. the chain identifier of the first residue involved i;
2. the residue number (according to PyMOL) of the first residue i;
3. the chain identifier of the second residue involved j;
4. the residue number (according to PyMOL) of the second residue j;
5. the entity of the relationship $m_{i,j}$, as encoded in the matrix.

The log file can be a powerful tool to evaluate the correspondence between the loaded matrix and the PyMOL structure on which the 3D plot is performed.

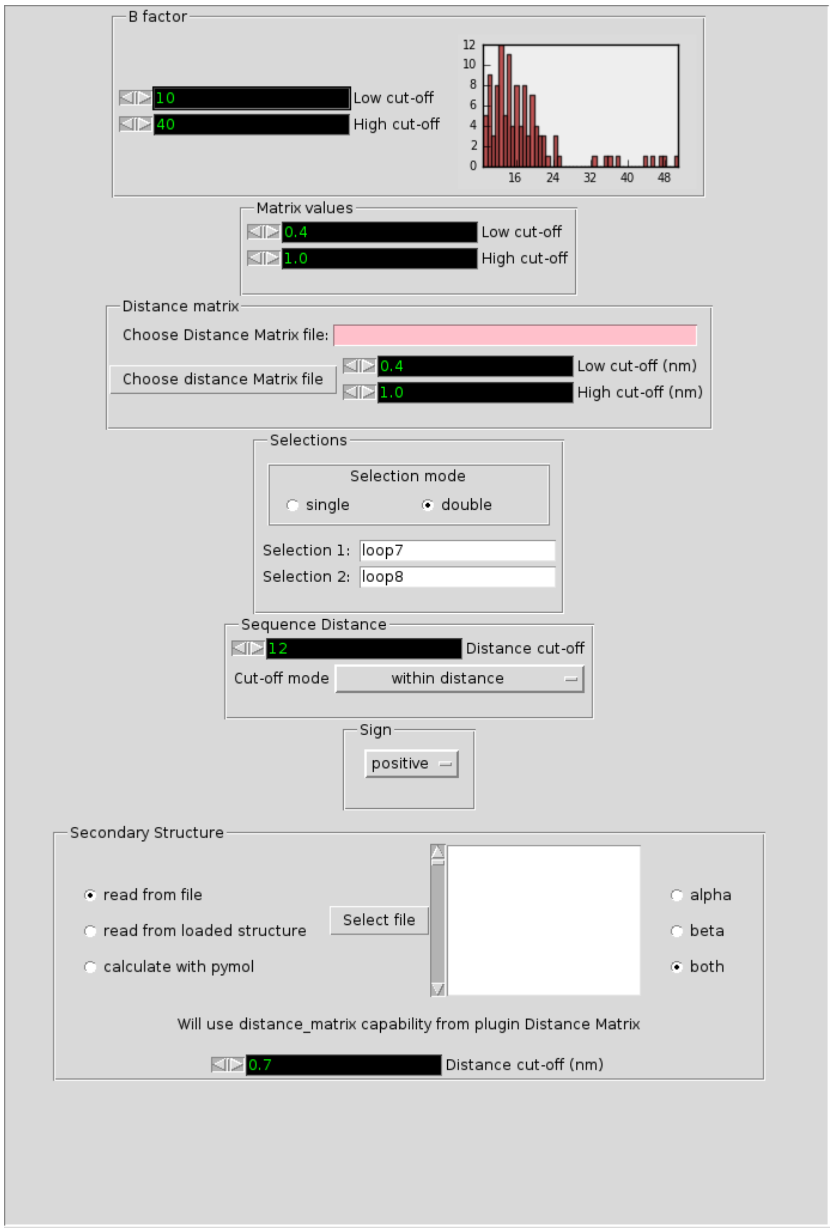
4.2 Filters

4.2.1 Filtering the loaded matrix

A square symmetric matrix of N elements describes $(N(N-1))/2$ unique relationships between distinct residues. As proteins easily comprise hundreds of residues this number gets quite high, and plotting the whole matrix on the 3D structure in terms of binary relationships usually turns out to be unpractical. Filtering out the least meaningful elements of the matrix is thus often more a necessity than an option, and it is useful to discern and point out the most important features of the relationships encoded by the matrix.

xPyder addresses this problem by implementing a modular filtering system, designed with expandability and flexibility in mind. Each filter acts independently on the loaded matrix, by deleting (zeroing out) those matrix elements that do not match the filters' criteria. They work like an array of superimposed cascading masks; only those elements that aren't blocked by any filter get to the plotting stage. The filtering system is modular, meaning that additional elements can be implemented and added to xPyder as external plugins, thus ensuring a high level of expandability. Filtering plugins are usually independent respect to one another, but can share their capabilities by registering to a common internal interface. The behavior of a plugin may thus depend by the status of another one; this will be described more in detail later when necessary.

From the user's point of view, filters are handled by the Filters control panel, on the right side of the interface, where each installed plugin is listed along with an activation/deactivation checkbox and a status light. By default all filters are off, and can be turned on by clicking on the respective checkboxes. The Filters



B factor

10 Low cut-off

40 High cut-off

Matrix values

0.4 Low cut-off

1.0 High cut-off

Distance matrix

Choose Distance Matrix file:

Choose distance Matrix file

0.4 Low cut-off (nm)

1.0 High cut-off (nm)

Selections

Selection mode

single [✓ double]

Selection 1: loop7

Selection 2: loop8

Sequence Distance

12 Distance cut-off

Cut-off mode within distance

Sign

positive

Secondary Structure

[✓ read from file]

read from loaded structure

calculate with pymol

Select file

alpha

beta

[✓ both]

Will use distance_matrix capability from plugin Distance Matrix

0.7 Distance cut-off (nm)

Figure 2: Matrix filters options

page contains a small box per filter, and this is where the user can modify the filtering parameters and behavior at will.

Plugins have three possible statuses, which are represented by three different colors of the status light.

- Turned off (red): the filter is turned off and will not be used
- Turned on and active (green): the filter is turned on and ready to work; it will be used.
- Turned on and inactive (yellow): the filter is turned on but is not ready to work. This is usually due to the fact that incorrect parameters have been specified by the user or that the plugin is awaiting for user intervention.

Only plugins in the green status will be used to filter the matrix before plotting. To make the inactive filters active (i.e. to turn the yellow filters green) the user is invited to modify the filter's parameters according to the indications written next to the plugin's name in the Filters control panel, until the plugin's status light turns green. The status light is automatically updated in real-time upon user intervention.

xPyder comes with six default filters, which should cover the users' most common necessities. Please consider that, if not specified otherwise, the plugins' parameters are defined so that the users chooses what will be plotted, not what will be filtered out. For example, if for a given filter controlled by a parameter whose "meaningful values" range is defined by the user as residing between 0.2 and 0.6, all the matrix elements for which that parameter is lower than 0.2 and higher then 0.6 will be excluded from plotting.

4.2.2 Default filters

The following section describes how each of the default filters work. For each filter, default parameter values have been provided by the authors according to personal experience on a limited number of test systems; the user is invited to modify them as required.

In the following description, M will be the loaded relationship matrix, and m_{ij}will be the matrix element referring to residues i and j.

Sign Filter This is the simplest filter, designed to filter matrix values by sign. If positive or negative is chosen, only positive or negative (respectively) values of M will be plotted ($m_{ij} > 0$ or $m_{ij} < 0$).

Sequence distance filter This filter works by filtering the matrix elements depending on distance in protein sequence of the respective residues. Given I and J as the simple sequence numbers of residues i and j, if the cut-off mode is "Within distance", only matrix elements whose residue couples have a distance in sequence lower than the cut-off value c ($|I - J| < c$) will be considered for plotting. On the contrary, if the cut-off mode is "Over distance", only residue couples whose distance in sequence is higher or equal to the distance cut-off value ($|I - J| \geq c$) will be plotted.

Selections filter This filter works on PyMOL selections and objects, by excluding all the matrix elements whose residues are not part of the chosen selection(s). The two input boxes, Selections 1 and 2, are used to provide the plugin with user-defined selections by name. "all" is also a permitted value.

In Single mode, all the relationships between the residues included in the selection specified in Selection 1 are plotted. The content of Selection 2 is ignored.

In Double mode, all and only the relationships between each residue of Selection 1 and each residue of Section 2 are considered for plotting. This is particularly useful, for example, to identify the relationship between a residue and the rest of the protein. Notice that, in Double mode, the two selections must be strictly overlapping (falling back to Single mode) or completely non-overlapping.

Matrix values filter This filter allows to filter the input matrix by the matrix values itself. Only the input matrix elements m_{ij} whose values fall between the low cut-off l and the high cut-off h are plotted ($m_{ij} \epsilon [l; h]$).

Distance matrix filter The 3D distance filter acts on the input matrix depending to a C_α distance matrix R. It works similarly to the Relationships entity filter, only considering the r_{ij} element as mean of comparison instead of the input matrix element $m_{ij}(r_{ij} \epsilon [l; h])$. The distance matrix can be loaded from file, or, as happens by default, calculated on the loaded tridimensional structure. Distance are expressed in nm.

The distance matrix loaded on the 3D distance plugin is also used by the Secondary structure filter plugin.

Secondary structure filter The secondary structure filter has been designed to filter out the matrix elements which connect residues belonging to the same secondary structure element, defined as α-helix or β-sheet. This is particularly useful for correlation matrices, as residues belonging to the same secondary structure element usually feature a high level of motion correlation, which is useful to filter out to obtain a more compact and cleaner representation.

The user can choose to filter out correlations between residues involved in alpha or beta structural elements, or both, using the controls in the right side of the Secondary structure filter in the Filters tab.

As identifying secondary structure elements is not a trivial task, different working modes can be selected. If read from file is chosen, the user has to input a secondary structure definition file by clicking on the Load secondary structure button. The file may be in the DSSP [Kabsch W and Sander C, Biopolymers, 1983] or PDSSP output file format (which will be described shortly). If the loaded secondary structure file contains information for more than one definition, as often happens with PDSSP files, the user has to choose the secondary structure definition to be used; the numbers refer to the order of the structure present in the PDSSP file.

If "read from loaded structure" is chosen, the secondary structure definition already present in the reference PyMOL object will be used.

At last, PyMOL is able to calculate a secondary structure definition for the loaded structure using the util.ss command (see PyMOL documentation for details), as happens when using the "calculate using PyMOL" option. PyMOL authors stress that this is not a sound and failproof method and that the obtained results should be subject to proofreading before productive use. We suggest to use this method as a quick-and-dirty way to filter out the secondary structure elements, mainly useful for fast screening.

Of the described working modes, DSSP only is able to assign beta residues to β-sheets. This feature is, indeed, very useful, as it permits to filter out the matrix elements which join β-strands of the same β-sheet. In the cases in whichβ-sheets are not specially crafted sheets assignation algorithm is used. First, β-residues are considered as nodes of a graph, which are connected by edges if their distance in the 3D structure is inferior to the Distance cut-off defined in the filters' interface, or if they are part of the same $\beta-$strand. The connected components of the graph are then identified, which correspond to single interconnected β-sheets. All the matrix elements referring to all the possible $(i, j), i \neq j$ couples of nodes of each connected component are filtered out.

Please notice that the distance matrix used to build the graph is retrieved from the Distance matrix plugin, so that either the user-loaded matrix or the pdb-calculated matrix will be used. Both the distance matrix and the secondary structure elements should refer to the same structural ensemble to ensure a consistent behavior of the plugin.

The PDSSP file format is designed to host secondary structure data for one or more molecular dynamics simulation or crystal structures. It is composed of$1 + (n * 2)$ space-separated columns, and each row represents a protein residue. The first column is the residue number. The remaining columns have to be interpreted as couples, each couple representing a simulation or crystal structure. The first column of each couple hosts the type of secondary structure as described in the DSSP dictionary, with "~" representing unstructured regions. The second column is always a percentage value which represents the fraction of frames in which the given residue is found in the secondary structure written in the first column. For example, this PDSSP file:

```
1    H    100.00    H    100.00
2    H    100.00    H    100.00
3    H    100.00    H    100.00
4    H    90.59     H    90.23
5    H    84.82     H    81.17
6    H    69.86     ~    67.43
```

7	H	58.76	~	59.26
8	~	49.47	~	58.12
9	~	68.43	~	84.96

will represent the most retained secondary structures of a 9-residue peptide, in two different simulations.

B Factor filter Using this filter the user can plot only correlations between residues featuring B Factor values in a given range. The B Factor is taken from the corresponding column of the PDB file from which the reference structure was loaded (in fact, it is taken from the values parsed by PyMOL). By modifying the Low and High cut-offs, the user can tweak the allowed B Factor range to her own requirements. To help in the definition of the cut-off values, after the first plot is issued, a histogram representing the distribution of B Factor values in the reference structure is shown inside the filter's option box. This allows the user to modify the cut-off values accordingly to refine the representation in subsequent plots.

Acting on real B Factors, or on simulation-derived B Factors, the user can exclude from the analysis the regions that have high experimental uncertainty, or focus the analysis on highly flexible tracts of the protein. But the applicability of the filter goes behond simply filtering flexibility. In fact, it is a simple and common practice to replace the B Factor column in PDB files with whatever biochemical property may be of interest, as a means of providing results of a given analysis along with a structure, e.g. for colouring the structure accordingly. xPyder's B Factor filter exploits this, giving the user highly extensible filtering options thanks to cross-talk with other applications. Finally, in cases where the analysis tool that produces the values the user would like to use for filtering the correlation analysis does not provide output in the B Factor column of a PDB file, but instead for example in a text file, there exist several scripts that make these values readily available both for coloring and for filtering with the B Factor filter in xPyder. One such example is reported on the PyMOL Wiki (http://www.pymolwiki.org/index.php/Color) and reported here.

Consider that the analysis tool has provided the ouptut profile of interest in a plain text file (*output.dat*), containing on each line the value for each residue in order. After loading the corresponding PDB structure (*structure.pdb*) in PyMOL, the B Factor values of the resulting PyMOL object can be modified by using the `alter` command:

```
# 1) load the protein in the "reference" PyMOL Object
cmd.load("structure.pdb", "reference")

# 2) open the file containing the analysis output
in_file = open("output.dat", 'r')

# 3) read the data and store it, then close the file
stored.profile = []
for line in in_file.readlines():
    stored.profile.append( float(line) )
in_file.close()

# 4) clear out the old B Factors
alter reference, b=0.0

# 5) update the B Factors with the profile
alter reference and name CA, b=stored.profile.pop(0)
```

The `reference` PyMOL object now has re-defined B Factor values taken from the data in the *output.dat* file. In case the latter contains both residue number and the output profile, separeted by white space (i.e. a two-column file, of which only the second is of interest), step 3) of the above procedure becomes:

```
# 3) read the data and store it, then close the file
stored.profile = []
for line in in_file.readlines():
    # values is a list: [ residue_number , profile_value ]
    values = line.split()
    # keep only the second element
    profile_value = values[1]
    stored.profile.append( float(profile_value) )
in_file.close()
```

A script (data2bfactor.py) that performs this operation (and a similar operation on the occupancy field) can be downloaded from Robert Campbell's PyMOL script repository (http://pldserver1.biochem.queensu.ca/~rlc/work/pymol/).

4.3 Chains of relationships

Introduction: a short primer on graph theory

In mathematics and information science, a graph is an abstract representation that encodes relationships between objects of a given set by formalizing binary relationships between them. A graph is composed of nodes or vertices, which are the objects between which relationships are defined. Nodes are connected by edges or arcs, that define the relationships between them; two nodes are connected by an edge only if, in a given problem, a certain relationship exists between them. For example, we could build a graph with cities of a certain region as nodes and roads or railways connecting them as edges; two cities would then be connected only if direct travel between them is possible. In fact, a graph is often visualized as a bidimensional plot in which nodes are represented by dots and edges are represented by lines or arrows connecting them. Once the graph has been built, we can use several algorithms and techniques to answer more specific questions; In the cities example, we may ask ourselves what is the route with least exchanges between two non directly connected cities.

In a simple graph, such as the one used by xPyder to encode the binary relationships between residues, the edges are undirected (i.e. the relationships between nodes don't have a directionality), no more than one edge connects two given nodes (the relationships are binary) and there are no loops (no edges connecting a node with itself exist). Moreover, each edge is associated with a (positive) number that defines the strength of the relationship (the graph is weighted). Starting on this simple description, other peculiar metrics and properties of a graph can be derived.

The degree of a node is defined as the number of edges connected to it. As no loops exist in the relationships graph, a node can have at most a degree of $n - 1$, where n is the total number of nodes.

In graph theory, a wealth of problems deals with identifying specific paths between nodes. A path in a graph is a specific sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence. This means that one could travel from the first vertex of the path to the last along the graph, visiting all the nodes of the path by passing through edges connecting them.

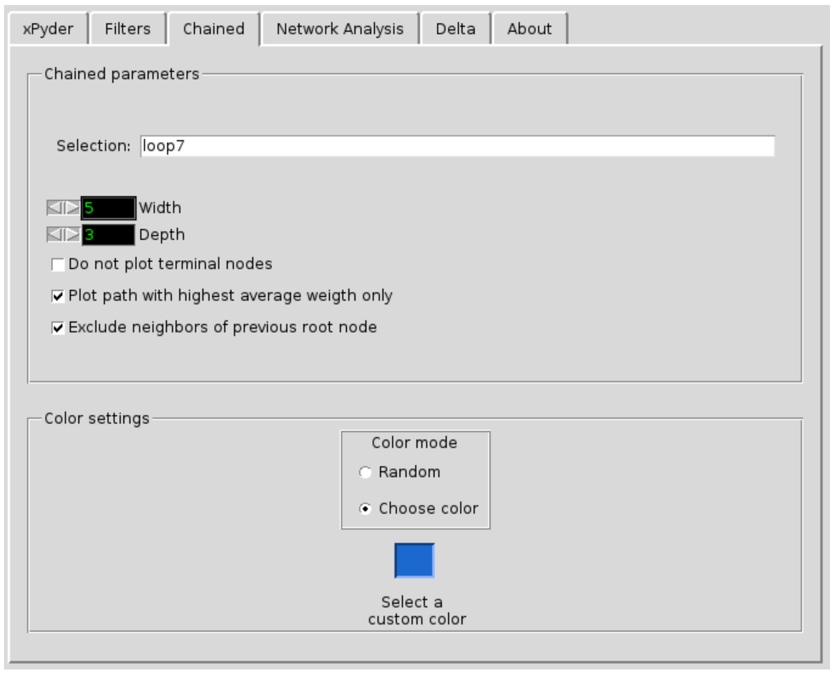
In the graph calculated by xPyder, two vertices may or may not be connected by edges, depending on the used filter options and the matrix values. This means that a graph may be formed in such a way that all the nodes are reachable from each other (is connected) or not. In the latter case, the graph is is composed by a number of connected components. A path always exists between two arbitrary node belonging to a connected component, while no paths exist between two nodes belonging to two different connected components.

The algorithm we implemented, inspired to the one available in the FlexServ web server [Camps *et al.*, Bioinformatics, 2009], is designed to identify chains or sequences of residues maximally connected to a given one of interest.

4.3.1 The algorithm

As introduced, the algorithm works on a graph representation of the input matrix M, considering the input matrix as a weight matrix. Residues I and J, or corresponding to matrix indeces i and j, are represented as nodes which are connected by an edge of weight m_{ij} if $m_{ij} \neq 0$. Matrix filtering (see previous sections) is performed before the construction of the graph, so that filtered out elements, which have value of 0 in the filtered matrix, are never represented. Negative matrix elements are also always filtered out.

Once the graph has been built, a starting node (root) is selected and all the edges connected to the root are sorted by weight. The list of nodes is filtered by discarding both those that have already been explored by the algorithm, and, if desired, those that are also connected to the root node of the previous iteration (if available). The first w edges of the list are selected, thus identifying the w most importantly related nodes to the root. Each of these residues is then considered as a new root and the procedure is recursively repeated for d iterations. The final result is a tree of connected nodes, with at most w branches at every node and of maximum depth d.



Selection: loop7

5 Width

3 Depth

☐ Do not plot terminal nodes

☑ Plot path with highest average weigth only

☑ Exclude neighbors of previous root node

Color mode

○ Random

● Choose color

Select a custom color

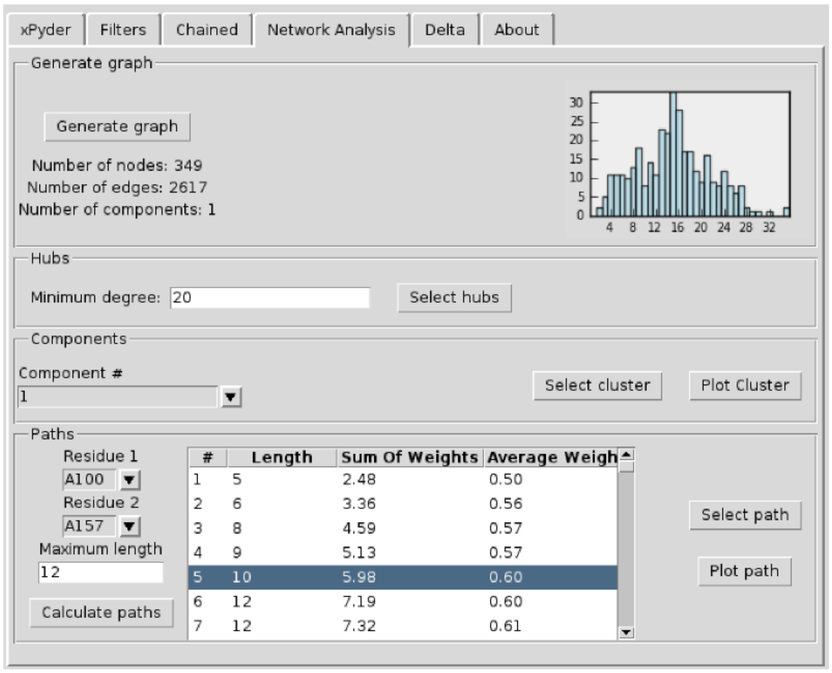
Figure 3: Chains of relationships

d and w, also known as depth and width, are two user-configurable parameters.

4.3.2 Use of chained analysis

Chained relationships are calculated in the Chained tab. The user has to input a valid selection name into the Selection box, which must contain the residue(s) which will be considered as root by the search algorithm. The calculation is repeated independently considering each residue of the selection as root and using the graph obtained by the filtered input matrix. Width and depth parameters, as explained before, can be easily defined in this tab. The user can choose between two coloring modes: random or single color. In the random node a unique random color is chosen to represent a tree calculated by the algorithm starting from a single residue. This is particularly useful to quickly visualize chained relationships starting from several residues at once. In single color mode all the calculated trees are plotted on the structure using the same, user-defined color. Apart from color, the rest of the plotting settings is configured in the first tab.

Please keep in mind that using a pre-filtered matrix is essential to obtain meaningful results from the chained relationships analysis. This is particularly evident when dealing with motion relationships, as one residue will probably feature high correlation with closer residues, especially when they belong to the



Number of nodes: 349
Number of edges: 2617
Number of components: 1

Minimum degree: 20

Component #: 1

Residue 1: A100
Residue 2: A157
Maximum length: 12

#	Length	Sum Of Weights	Average Weigh
1	5	2.48	0.50
2	6	3.36	0.56
3	8	4.59	0.57
4	9	5.13	0.57
5	10	5.98	0.60
6	12	7.19	0.60
7	12	7.32	0.61

Figure 4: Graph analysis

same secondary structure element.

4.4 Protein graph analysis

The study of binary interactions, as the defined in the matricial representation used by xPyder, can be insightful in detecting relationships between single residues or single residues and protein regions, but does not tell much about how information is transmitted in the protein structure. This can be performed by identifying peculiar paths of connected residues over the structure, using the input matrix as the source of information. Since we're dealing with a network of relationships encoded by a matrix which closely resembles the graph theory's adjacency matrix, the use of graph theory seems natural to deal with such a problem. When the interaction network is seen as an undirected graph, performing a number of analyses is possible to evaluate overall topological properties of the network as well as the presence of peculiar paths between couples of residues. This analysis is performed in the Graph analysis tab.

4.4.1 Calculating the graph

Analyzing the interactions graph first involves calculating such graph from the loaded matrix. This is performed by clicking on the Generate graph button in the Graph analysis tab. Clicking on the button performs the following operations: i) the loaded matrix is filtered according to the activated and working filters, plus filtering out the negative values; ii) the relationships weighted simple graph is generated from the filtered matrix. Residues are represented as nodes and the matrix-encoded relationships as edges, with edge weights equal to the corresponding matrix value; this means that two nodes are connected if the corresponding matrix value is non-zero. Diagonal elements of the matrix are always discarded. iii) the user interface is populated. Once the graph has been loaded additional analyses and plotting features can be performed, as detailed in the following sections.

Positive weights only are considered because positive-weighted graphs are simpler to treat. Anyway, one may need to use both positive and negative values of a metric (as happens in DCCMs) to obtain a more complete picture. In this case, an absolute values matrix should be considered for the graph analysis.

When the graph has been generated, a number of properties of the calculated graph are displayed, along with an histogram plot which display the distribution of vertex degrees of the graph, i.e. the number of nodes having a given degree value.

Important: Please notice that the graph is generated by clicking on the Generate graph button only. This means the the loaded graph is *not* automatically refreshed upon changing the filtering options; changing the filters without recalculating the graph will leave the Graph analysis tab and the rest of the program in an inconsistent state, as the two would refer to two differently filtered matrices. Loading another matrix, on the other hand, completely resets any previously calculated graph-related data, so that the graph has to be regenerated from scratch to perform the graph analysis.

4.4.2 Graph analysis

Hubs The Hubs section of the graph analysis permits to identify the hub nodes of the graph, defined as those having a degree $\geq k$where $k$is a user-defined value. The degree of a node is the number of edges connected to it, and hub nodes are those featuring a relatively high number of relationships when compared to the other ones, and thus likely to have a significant functional or structural role, depending on the plotted matrix. A meaningful value of k can be easily selected by evaluating the distribution of degree values on the histogram plot. Once that k has been defined by inserting the chosen value in the input box of the hubs section, the user can choose to select on the 3D structure the hub residues by clicking on the Select hubs button.

Connected components The user can choose to identify and plot the connected components of the graph. The components are listed by size in the list box under the Connected components section. The selection of the single component also permits to visualize few properties of the selected component. The user can choose to create a selection in the 3D structures of the residues corresponding to the nodes belonging to the selected connected component by clicking the Select component button. By clicking the Plot component button, otherwise, the relationships between the residues belonging to the selected component are plotted on the 3D structure in the form of cylinders interconnecting the alpha carbon of residues.

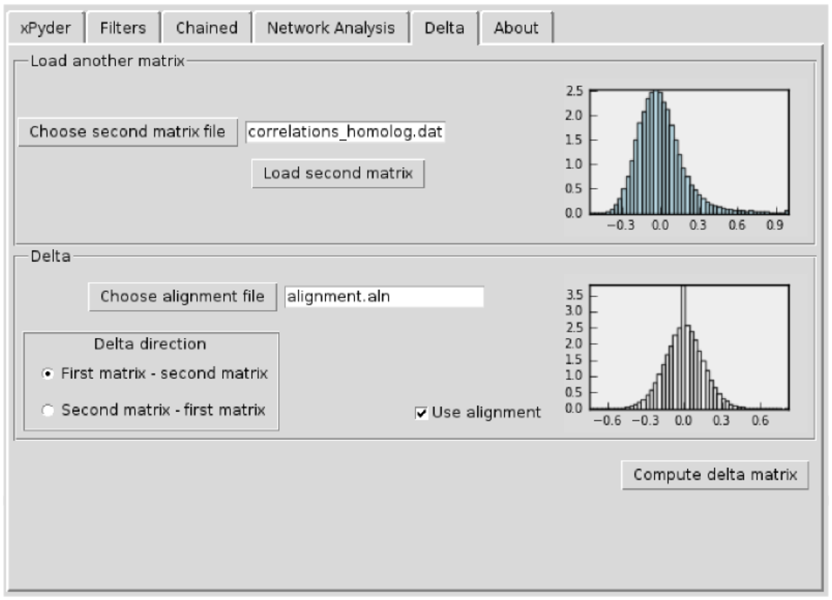
Pathways The paths analysis uses a slight variation of depth-first search algorithm to identify all the paths in the graph connecting two given nodes. The user needs to specify the two extremities of the paths to be searched and the maximum path length m; upon clicking on the Calculate paths button, all the possible paths of length $\leq m$ will be displayed on the table on the right. The computational cost of this procedure depends on the topology of the graph and on the chosen threshold m. Please consider that in a very connected graph several thousands of paths may exist between two given nodes, meaning that this functionality should be used on well filtered graphs and on data types which emphasize local connectivity.

Once the table has been filled, to user can choose to select or plot on the 3D structure the selected path by selecting one or more on them in the table and clicking on the appropriate buttons. By clicking table labels it is also possible to order the table elements according to length, average or cumulative weight.

4.5 Delta matrix

When comparing multiple matrices, the possibility to compute difference matrices, in a way similar to computing difference profiles (e.g. for RMSF profiles), focuses the analysis on the quantitative details of the comparison. The Delta correlation analysis of xPyder allows the user to compute and plot on the 3D structure the difference matrix (or delta matrix), to integrate structural insight into the detailed comparisons of the matrices, while providing the full set of xPyder's filtering features to customise the plot.

This analysis may prove useful when comparing matrices computed on the same structure (to compare two matrices resulting from two experiments on the same protein, e.g. the correlations from two MD simulations), or on different structures, for example two homologous proteins or a wild-type and a mutated variant. xPyder makes both analyses possible by allowing the user to specify an alignment to match the sequences of two homologues, in order to compute a meaningful difference matrix.



correlations_homolog.dat

alignment.aln

Delta direction: [✓ First matrix - second matrix] Second matrix - first matrix

[✓ Use alignment]

Figure 5: Delta matrix calculation

4.5.1 Load another matrix

Allows the user to load a second matrix ($M^{(2)}_{i,j}$) that will be used together with the first matrix ($M^{(1)}_{i,j}$, see above) for delta calculations:

$D_{i,j} = M^{(1)}_{i,j} - M^{(2)}_{i,j}$

The delta matrix will be plotted on the structure using the same plot settings for the first matrix. If the data in the second matrix was computed on the saime structure as the first matrix, the sizes of the two matrices should be identical; if the sizes are not identical, an aligment must be provided (see on).

Choose matrix file: The file containing the second matrix can be specified either by using the file selection dialog that appears by clicking on the button, or by typing its full path in the adjacent field;

Load second matrix: This loads the second matrix from the specified file, and plots a histogram of the loaded values;

4.5.2 Delta

In the case that the sizes of the first and second matrices don't match, an alignment must be provided so that xPyder can correctly match the values in the two matrices: $_{i,j} = M^{(1)}_{i,j} - M^{(2)}_{A(i),A(j)}$, where $A(i)$ is the position of $M^{(2)}$ matching position i in $M^{(1)}$ according to the alignment.

The provided alignment file should contain at least 2 sequences, referring to the data in the first and second matrix in this order. This means that the length of the first sequence in the alignement, excluding gaps, should match the size of the first matrix; the same holds for the second sequence/second matrix. Any further sequences in the alignement will be ignored.

The format of the alignment file is a loose version of the ALN/ClustalW2 format (http://www.ebi.ac.uk/2can/tutorials/formats.html#aln); we have dropped the requirement for the header (CLUSTAL W (version) ...) and the 60-residue limit for blocks. The consensus line below each block may be present and is ignored.

Please note that providing an alignement may be required even when the sizes of the matrices match, depending on the way the data in the two matrices was obtained (e.g. when comparing two proteins that align in such a way that only one of the two sequences has gaps).

Choose alignment file: The file containing the correctly formatted alignement may be specified either by using the file selection dialog that appears by clicking on the button, or by typing its full path in the adjacent field;

Delta direction: Allows the user to define which should be the order of the resulting delta matrix; please note that the two results are one the opposite of the other;

Use alignement: The user can force xPyder to use/not use the provided alignement, if it is feasible;

Compute delta matrix: This computes the delta matrix $D_{i,j}$, using an alignement if specified, and plots a histogram of the resulting delta values;

Plot: Performs the entire Delta analysis pipeline with a single click; in particular, it performs the following actions:

1. Load the second matrix (see "**Load second matrix**");
2. Compute the delta matrix (see "**Compute delta matrix**");
3. Plot the delta matrix.

5 Obtaining an xPyder matrix

As the xPyder main input format is an old-style plain ASCII matrix (see above for details), obtaining an xPyder-compatible file should be easy from common analysis tools such as those included in the gromacs or AMBER software suites. The following section briefly covers the topic of how to obtain the said matrices for the most used matricial metrics.

5.1 MD Trajectories

5.1.1 Dynamical cross-correlation matrices and Linear mutual information matrices

The Dynamical cross-correlation matrices (DCCM) [Hunenberger *et al.*, J Mol Biol 1995, 252, (4), 492-503] can be easily calculated in gromacs using the modified version of the g_covar tool available on the gromacs website, that works with gromacs trajectories of version <4. This program has been updated to work with gromacs 4.x trajectories, and is available in the xPyder website.

In Amber you may refer to the ptraj analysis tool (look for the 'matrix corr' option).

You may also use the Wordom MD analysis suite (http://wordom.sourceforge.net/). A tool (wordomdccm2dat) is available on the xPyder website to convert the Wordom output to the ASCII matrix format used by xPyder. This conversion tool (and those described in the following paragraphs) is a Python 2.7 script which requires the numpy package. The usage of this and other scripts is pretty straightforward and adequately described in the script documentation itself (script -h or script –help).

Wordom is also able to calculate another index of motion correlation, namely the Linear mutual information (LMI) [Lange *et al.*, 2006, 62, (4), 1053-1061], thus obtaining Linear mutual information matrices (LMIM). As these matrices are in the same format of DCCMs, they can readily be converted and plotted using xPyder, as explained before.

5.1.2 Protein Structure Network

For both xtc and dcd trajectory files you may use Wordom; the steps needed to obtain a complete Protein Structure Network (PSN) [Vishveshwara *et al.,* Curr Protein Pept Sc 2009, 10, (2), 146-160] analysis are clearly detailed in the Wordom manual. Once a Wordom output has been obtained, the user needs to generate the corresponding .dat file to be used in xPyder. This is performed using the wordompsn2dat script available on the xPyder website, which permits to extract the interaction frequency for the PSN at a given I_{min}(i.e. the number of frames in which the two residues are connected by an edge in the PSN, over the total number of frames) or the average interaction strength matrix.

5.2 Conformational ensembles (PDB)

The user may also need to analyse and plot the aforamentioned properties for a multi-pdb ensemble, such as one derived from an NMR ensemble available in the PDB databank or from a number of X-ray structures. In this case, the most straightforward way is to generate an MD trajectory from the PDB ensemble using Wordom, and then proceed as previously explained for the analysis. The process of generating an trajecory from a list of single PDB structures (i.e. each containing a single model) is covered in the Wordom manual, and will be shortly discussed here.

The user usually starts from a number of PDB files, named {conf1.pdb, conf2.pdb ... confi.pdb ... confn.pdb}. All the PDB files must differ only in terms of atomic coordinates: they must contain the same atoms in the same order, with the same name. It is advisable to remove all the non-protein atoms from the PDBs before using them. Once the PDB files have been correctly edited, use one of them to generate a trajectory:

```
wordom -mono -imol conf1.pdb -otrj traj.dcd
```

The obtained PDB files can be appended to the trajectory file:

```
wordom -amol conf2.pdb -otrj traj.dcd
```

The obtained trajectory can then be analyzed using the standard Wordom tools and plotted using xPyder, as previously explained for trajectories.

PDB containing structures resolved from NMR experiments consist of several model entries each representing a single protein structure. These files have to be split so that one pdb per model is present, before adding them to a trajectory. This can be performed using PyMOL, or more easily, by using the nmrsplit Python script that is available on the xPyder website. The script requires numpy and Biopython (http://biopython.org/wiki/Biopython) installed.

Another useful software to analyze PDB ensembles is Theseus (http://www.theseus3d.org/); it is designed to calculate optimal superimpositions of PDB structures using the maximum likelihood method and is able to output DCCM and covariance matrices. It is also capable of performing mode analysis on the calculated matrices, and the eigenvectors of the N-th principal component can be plotted using xPyder. The conversion of the matrices from the Theseus to the xPyder format is performed using the thesesudccm2dat and theseuspcs2dat available on the xPyder websites; the latter writes the principal component i as the diagonal values in an otherwise empty xPyder matrix.