Protein Structure Networks with Bio3D

Overview

The Bio3D package (version 2.1 and above) contains functionality for the creation and analysis of protein structure networks. Here we will introduce this functionality by building and analyzing networks obtained from atomic correlation data derived from normal mode analysis (NMA) and molecular dynamics (MD) simulation data (**Figure 1**).

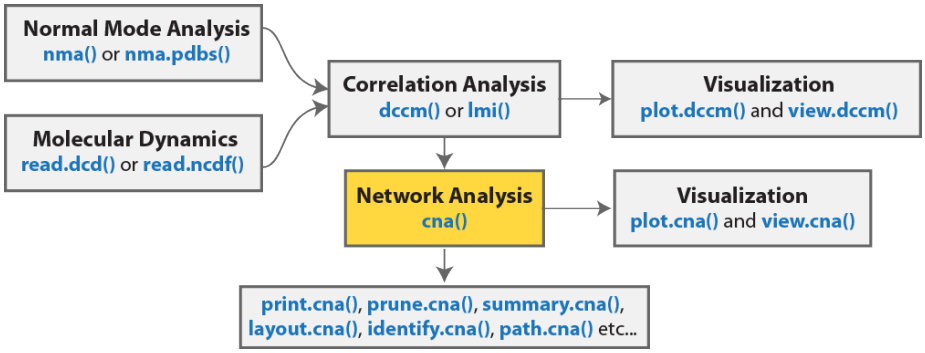


Figure 1: **Overview of a typical Bio3D network analysis protocol with key functions detailed in blue**. Normal mode and/or molecular dynamics input first undergoes **correlation analysis** (with the functions **dccm()** or **lmi()**). The output from these functions is typically a matrix of residue-by-residue cross-correlations. **Correlation network analysis** can then be performed (with the **cna()** function) to generate a correlation network with residues as nodes that are linked by weighted edges that are proportional to their degree of correlated motion. The **cna()** function also performs a **community clustering analysis** to identify *communities* of highly correlated residues. The full residue network and coarse-grained community network can be visualized and analyzed further. This methodology can provide valuable insight not readily available from the measures of correlation alone.

Basic network generation and visualization

Bio3D contains extensive functionality for NMA and MD analysis. An introduction to each of these areas is detailed in separate NMA and MD vignettes available online. In this section we will focus on correlation network analysis derived from NMA, and in subsequent sections MD.

The code snippet below first loads the Bio3D package, then reads an example PDB structure from the RCSB online database. We then follow the flow in Figure 1 by performing NMA, then dynamic cross-correlation analysis, and finally network analysis with the **cna()** function.

```
library(bio3d)
pdb <- read.pdb("1rv7")
modes <- nma(pdb)
cij <- dccm(modes)
net <- cna(cij)
```

To enable quick examination of the network analysis results we provide **cna** specific variants of the **print()**, **summary()** and **plot()** functions, as demonstrated below:

```
print(net)
```

```
##
## Call:
##   cna(cij = cij)
##
## Structure:
##  - NETWORK NODES#:   198     EDGES#: 458
##  - COMMUNITY NODES#: 11  EDGES#: 13
##
##  + attr: network, communities, community.network,
##         community.cij, cij, call
```

```
# Summary information as a table
summary(net)$tbl
```

```
## id size                                members
##  1   29               c(1:9, 89:103, 194:198)
##  2   28 c(10, 22:33, 83:88, 124:130, 184:185)
##  3   27                        c(11:21, 60:75)
##  4   31                c(34:39, 42:59, 76:82)
##  5    1                                   40
##  6    1                                   41
##  7   13          c(104:110, 121:123, 181:183)
##  8   25                   c(111:120, 159:173)
##  9   29          c(131:134, 141:158, 174:180)
## 10    6                               135:140
## 11    8                               186:193
```

```
# Plot both the 'full' all-residue network and simplified community network
par(mfcol = c(1, 2), mar = c(0, 0, 0, 0))
plot(net, pdb, full = TRUE)
plot(net, pdb)
```

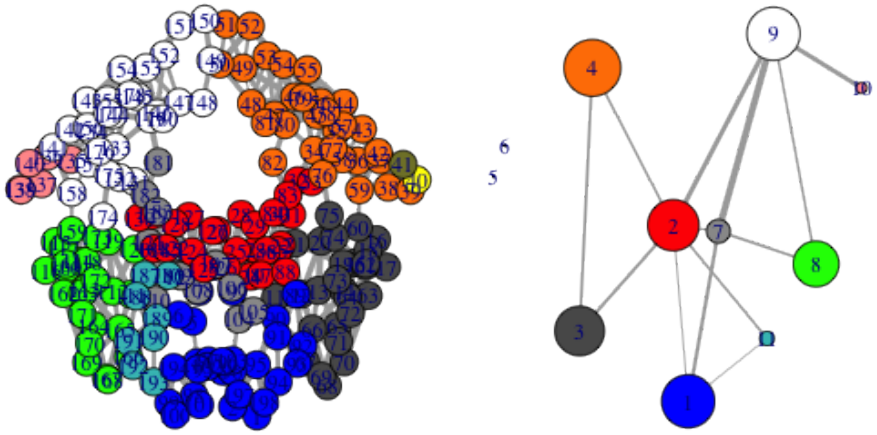


Figure 2: **Full all-residue network and simplified community network**

Note that the above code plots both a *full* all-residue network as well as a more coarse grained community network - see **Figure 2**. In this example, and by default, the Girvan-Newman clustering method was used to map the full network into communities of highly intra-connected but loosely inter-connected nodes that were in turn used to derive the simplified network displayed in **Figure 2**. We will discus community detection in further detail below and simply highlight here that a comparison of the distribution and connectivity of communities between similar biomolecular systems (e.g. in the presence and absence of a binding partner or mutation) has the potential to highlight differences in coupled motions that may be indicative of potential allosteric pathways. This approach has previously been applied to investigate allostery in tRNA–protein complexes, kinesin motor domains, G-proteins, thrombin, and other systems (refs).

Another useful function for the inspection of community structure is the **view.cna()** function, which permits interactive network visualization in the VMD molecular graphics program (see **Figure 3**).

```
# View the network in VMD
view.cna(net, pdb, launch = TRUE)
```

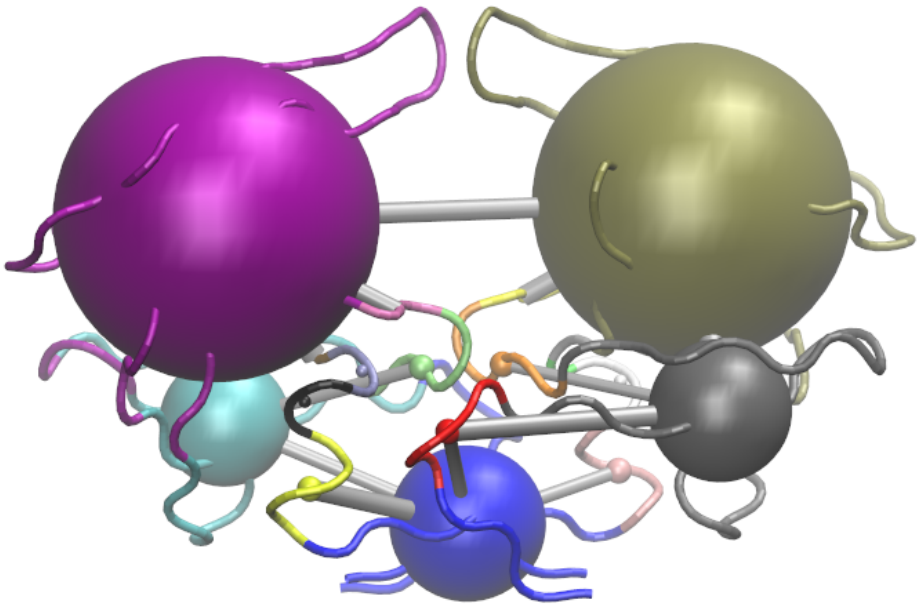


Figure 3: Example of **view.cna()** output

One of the main advantages of correlation network analysis is that it often facilitates the interoperation of correlated motions that can be challenging to rationalize from the output of a dynamic correlation map alone (e.g. the output

of the dccm() function). For example, the identified communities of highly intra-connected residues can be tied into the visualization of the raw correlation data (**Figure 4**) and cross referenced to the network visualizations presented in **Figures 2** and **3**:

```
plot.dccm(cij, margin.segments = net$communities$membership)
```

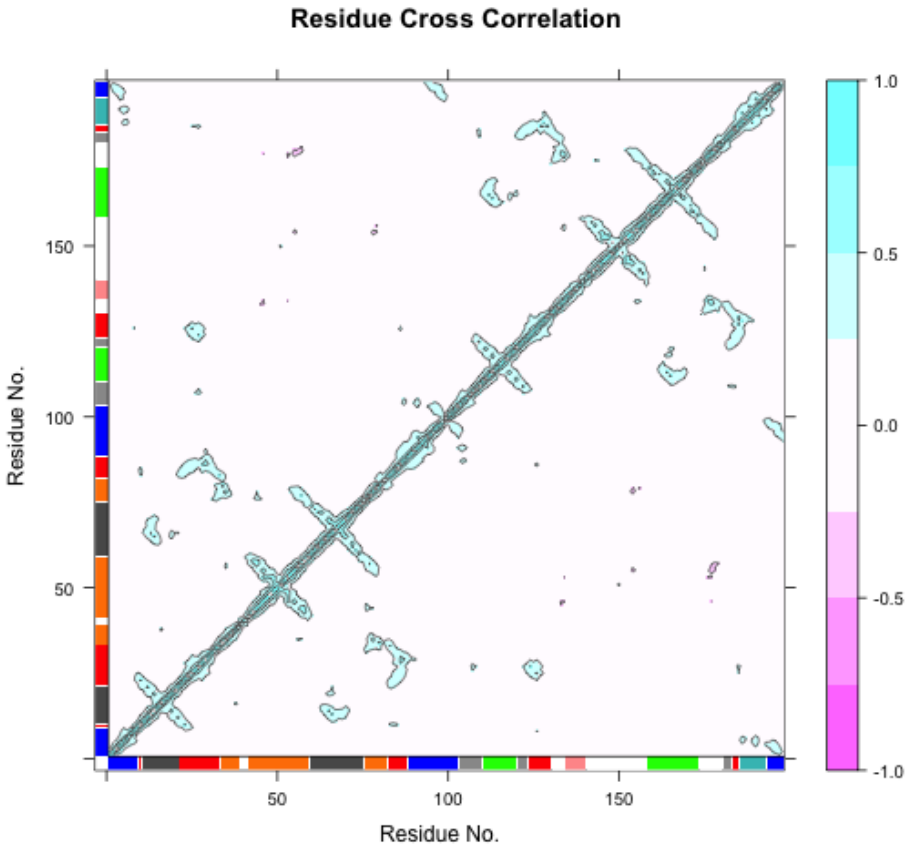


Figure 4: DCCM plot. Note that unique colors are assigned to communities and that these correspond to those annotated in all previous Figures and those used by VMD.

To more fully appreciate the results of network analysis one should become somewhat familiar with the structure of the object returned from the **cna()** function. This will allow you to generate customized visualizations beyond those in **Figures 2** to **4** and also enable you to more fully examine the effect of different parameters on network structure and topology.

The structure of cna() network objects The output of the **cna()** function is a list object of class cna with both network and community structure attributes. These are summarized at the bottom of the **print(net)** results above and can be explicitly listed with the **attributes(cna)** command:

```
attributes(net)
```

```
## $names
## [1] "network"           "communities"       "community.network"
## [4] "community.cij"     "cij"               "call"
##
## $class
## [1] "cna"
```

Here we introduce each of these components in turn:

- **$network**: The *full* protein structure network typically with a node per residue and connecting edges weighted by their corresponding correlation values (i.e. input cij values optionally filtered by the user defined 'cutoff.cij'). This filtered cij matrix is also returned in the output as **$cij**, as listed below.
- **$communities**: A community clustering object detailing the results of clustering the full $network. This is itself a list object with a number of attributes (see $attributes(net$communities) $ for details). Notable attributes include $ $communities membership, communities modularity and **$communities$algorithm**. These components detail the nodes (typically residues) belonging to each community; the modularity (which represents the difference in probability of intra- and intercommunity connections for a given network division); and the clustering algorithm used respectively.
- **$cij**: The filtered correlation matrix used to build the full $network.
- **$community.network**: A coarse-grained community network object with the number of nodes equal to the number of communities present in **$communities** (i.e. the community clustering of the full network) and edges based on the inter-community coupling as determined by the input 'collapse.method' (by default this is the maximum coupling between the original nodes of the respective communities).
- **$community.cij**: A cij matrix obtained by applying a "collapse.method" on $cij. The rows and columns match the number of communities in $communities. The values are based on the cij couplings of inter-communities residue. (collapse.method available are: max, mean, trimmed and median).

Additional network annotation Both the **$network** and **$community.network** attributes are igraph package network objects and contain additional node (or vertex) and edge annotations that can be accessed with the **V()** and **E()** functions in the following way:

```
#' Data access
# list.graph.attributes(net$community.network)
head(V(net$network)$color)
```

```
## [1] "#0000FF" "#0000FF" "#0000FF" "#0000FF" "#0000FF" "#0000FF"
```

```
V(net$community.network)$size
```

```
##  [1] 29 28 27 31  1  1 13 25 29  6  8
```

```
E(net$community.network)$weight
```

```
##  [1] 0.4786 0.5872 0.4636 0.5601 0.5259 0.4920 0.6456 0.5263 0.5510 0.5395
## [11] 0.7708 0.5252 0.6315
```

```
V(net$community.network)$name
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11"
```

In addition to various Bio3D functions, the full set of igraph package features can be used to further analyze these network objects.

Additional community annotation The **$communities** attribute returned by the **cna()** function provides details of the community clustering procedure. This procedure aims to split the full network into highly correlated local substructures (referred to as communities) within which the network connections are dense but between which they are sparser. A number of community detection algorithms can be used to solve the graph-partitioning problem. The default Girvan-Newman edge-betweenness approach is a divisive algorithm that is based on the use of the edge betweenness as a partitioning criterion. "Betweenness" is calculated by finding the shortest path(s) between a pair of vertexes and scoring each of the edges on this/these path(s) with the inverse value of the number of shortest paths. (So if there was only one path of the shortest length, each edge on it would score 1 and if there were 10 paths of that length, each edge would score 1/10.) This is done for every pair of vertexes. In this way each edge accumulates a "betweenness" score for the whole network. The network is separated into clusters by removing the edge with the highest

"betweenness", then recalculating betweenness and repeating. The method is fully described in (Girvan and Newman, 2002 PNAS).

A useful quantity used to measure the quality of a given community partition is the **modularity** (detailed in **$communities$modularity** component of **cna()** output). The modularity represents the difference in probability of intra- and inter-community connections for a given network division. Modularity values fall in the range from 0 to 1, with higher values indicating higher quality of the community structure. The optimum community structures obtained for typical correlation networks fall in the 0.4 to 0.7 range.

```
head(net$communities$modularity)
```

```
## [1] -0.0062092 -0.0033897 -0.0016651 -0.0001935  0.0038083  0.0052087
```

```
max(net$communities$modularity)
```

```
## [1] 0.7922
```

To do: Demonstrate the clustering progress and choice of modularity cut-point as well as showing the as.hclust() and dendPlot() functions.

Network Generation From Molecular Dynamics Data

For this example section we apply correlation network analysis to a short molecular dynamics trajectory of Human Immunodeficiency Virus aspartic protease (HIVpr). This trajectory is included with the Bio3D package and stored in CHARMM/NAMD DCD format. To reduce file size and speed up example execution, non C-alpha atoms have been previously removed and the frames superposed. Note that typically an input trajectory would require superposition with the **fit.xyz()** function prior to correlation analysis with the **dccm()** function (see the Trajectory Analysis vignette for full details).

The code snippet below first sets the file paths for the example HIVpr starting structure (pdbfile) and trajectory data (dcdfile), then reads these files (producing the objects trj and pdb).

```
dcdfile <- system.file("examples/hivp.dcd", package = "bio3d")
pdbfile <- system.file("examples/hivp.pdb", package = "bio3d")

# Read MD data
dcd <- read.dcd(dcdfile)
pdb <- read.pdb(pdbfile)
```

```
inds <- atom.select(pdb, resno = c(24:27, 85:90), elety = "CA")
trj <- fit.xyz(fixed = pdb$xyz, mobile = dcd, fixed.inds = inds$xyz, mobile.inds = inds$xyz)
```

Once we have the superposed trajectory frames we can asses the extent to which the atomic fluctuations of individual residues are correlated with one another and build a network from this data:

```
cij <- dccm(trj)
net <- cna(cij)
plot(net, pdb)
```

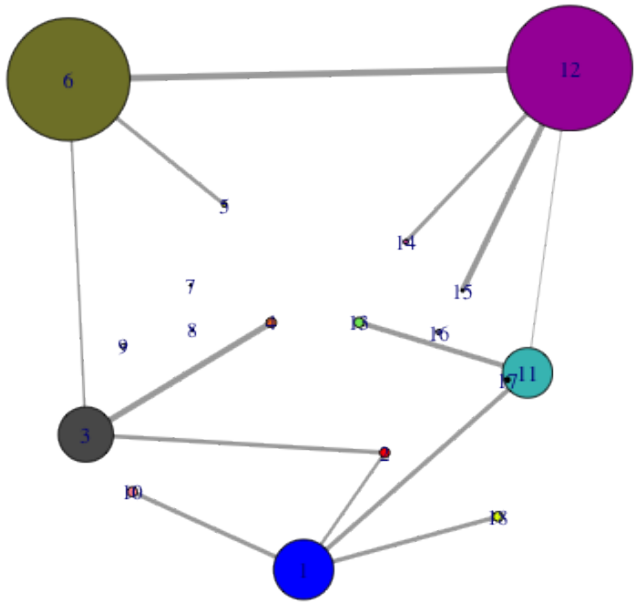
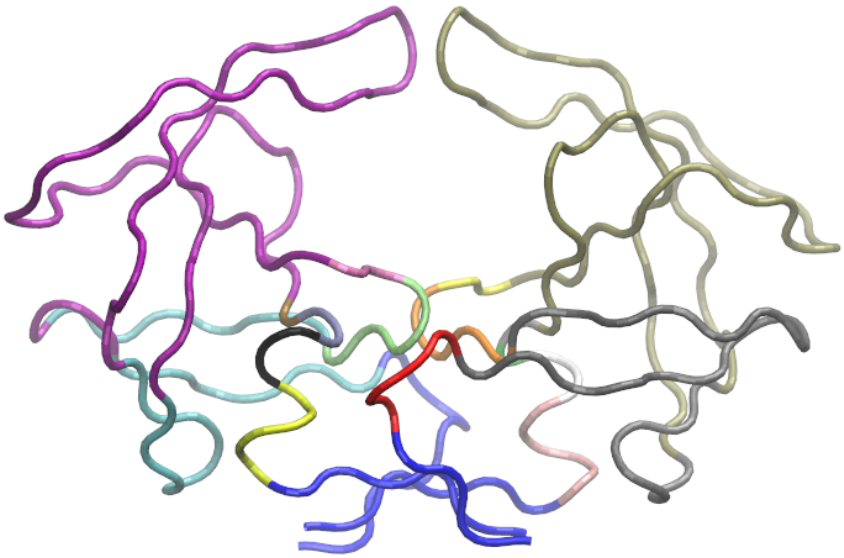


Figure 5: Network analysis of MD data

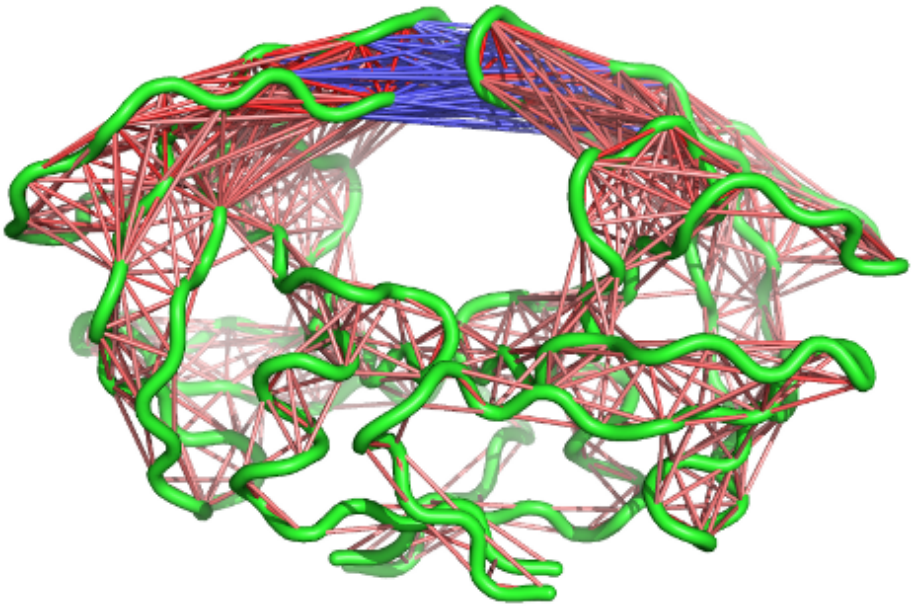
Note that the **dccm()** function can be slow to run on very large trajectories. In such cases you may want to use multiple CPU cores for the calculation by setting

the 'ncore' option to an appropriate value for your computer - see *help(dccm)* for details.

```
# View the correlations in pymol
view.dccm(cij, pdb, launch = TRUE)
```



HIVP structure colored according to communities

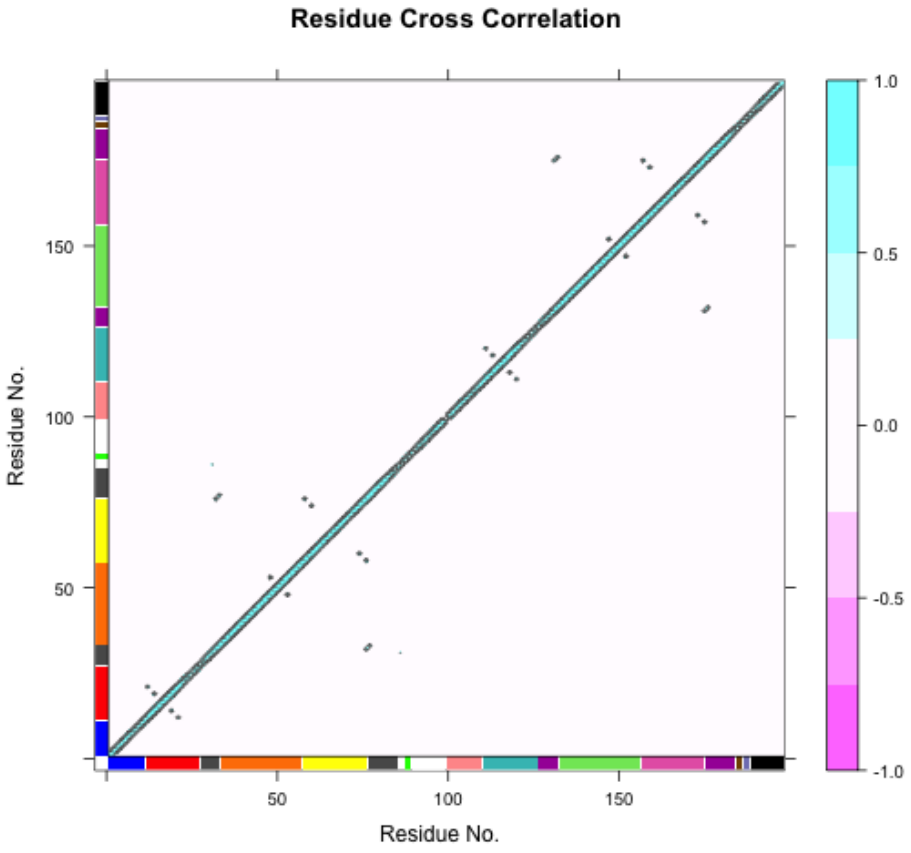


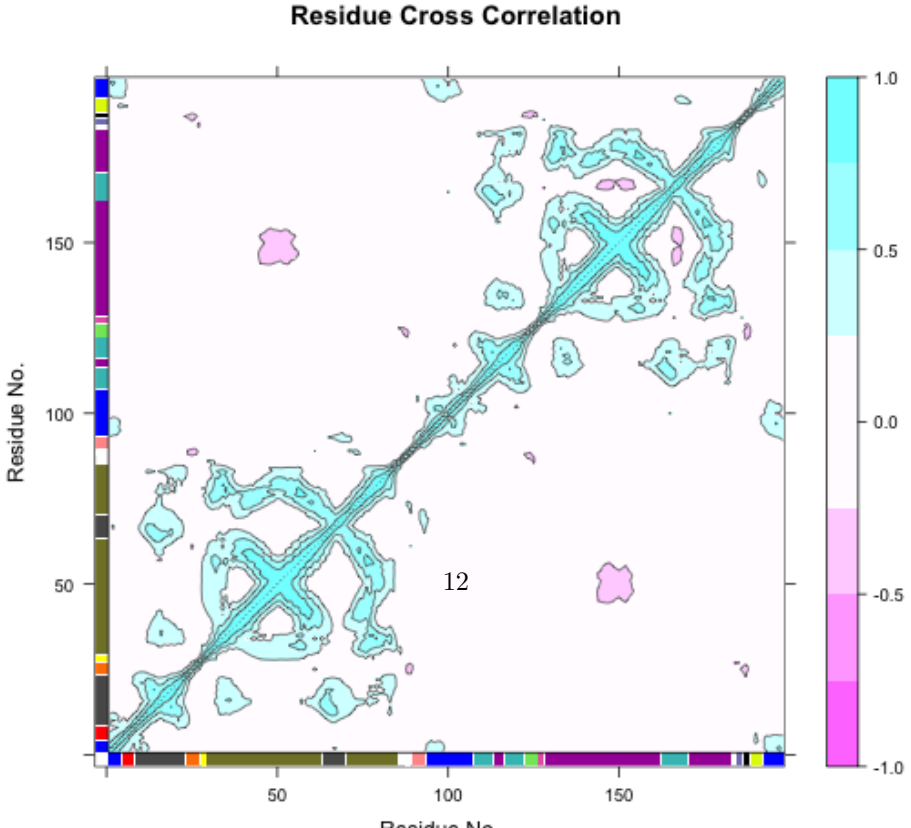
Using a contact map filter In the original Luthy-Shulten and co-workers approach to correlation network analysis [1] edges were only included between "in contact" nodes. Contacting nodes were defined as those having any heavy atom within 5.0 Å for greater than 75% of the simulation. The weight of these contact edges was then taken from the cross-correlation data. This approach effectively ignores long range correlations between residues that are not in physical contact for the majority of the trajectory. To replicate this approach one can simply define a contact map with the **cmap()** function and provide this to the **cna()** function along with the required cross-correlation matrix, e.g.:

```
cm <- cmap(trj, dcut = 5, scut = 0, pcut = 0.75, mask.lower = FALSE)
net.cut <- cna(cij, cm = cm)
```

One could also simply multiple the correlation matrix by the contact map to zero out these long-range correlations. However, we currently prefer not to use this filtering step as we have found that this approach can remove potentially interesting correlations that might prove to be important for the types of long-range coupling we are most interested in exploring. For example, compare the plot of correlation values below to those in **Figure X**.

```
par(mfcol = c(1, 2), mar = c(0, 0, 0, 0))
plot.dccm((cij * cm), margin.segments = net.cut$communities$membership)
plot.dccm(cij, margin.segments = net$communities$membership)
```


Residue Cross Correlation
Residue No.
Residue No.
50
100
150
1.0
0.5
0.0
-0.5
-1.0


Residue Cross Correlation
Residue No.
50
100
150
1.0
0.5
0.0
-0.5
-1.0

NOTE: Need to fix potential error message if mask.lower=TRUE is used in the cmap() call. It is also possible to use a consensus of contact map and dynamical correlation matrices from replica simulations with the functions **cmap.filter()** and **dccm.mean()** - see their respective help pages for details.

Network visualization

The **layout.cna()** function can be used to determine the *geometric center* of protein residues and communities and thus provide 3D and 2D network and community graph layouts that most closely match the protein systems 3D coords.

```
layout3D <- layout.cna(net, pdb, k=3)
layout2D <- layout.cna(net, pdb, k=2)
```

Other network layout options are available via the igraph package, see *?igraph::layout* in R for details.

Identifying nodes/communities and their composite residues The **identify.cna()** function allows one to click on the nodes of a **plot.cna()** network plot to identify the residues within a give set of nodes. This can also be a useful first step to determine the placement of text labels for a plot -see *help(identify.cna)* for details.

```
# Click with the right mouse to label, left button to exit.
xy <- plot.cna(net)
identify.cna(xy, cna = net)
```

You can also make 3D plots using the rgl environment with the function **plot3d.cna()**. An example is shown in Figure 8.

```
plot3d.cna(net, pdb)
```

Color settings and node/edge labels The **vmd.colors()** function is used by default to define node color. This can be changed to any input color vector as demonstrated below. Note that the **vmd.colors()** function outputs a vector of colors that is ordered as they are in the VMD molecular graphics program and changing the colors will likely break this correspondence.

```
bwr.colors(20)[net$communities$membership]
```

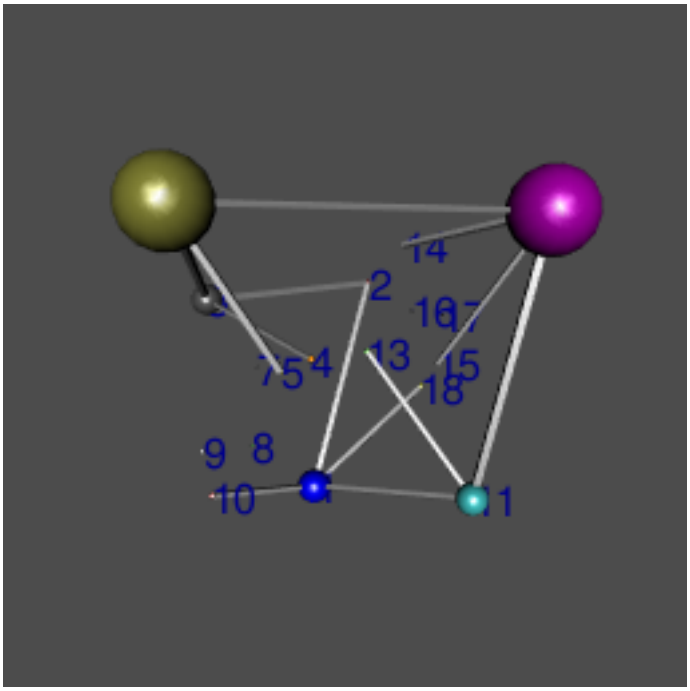


Figure 6: Interactive 3D community network plot

Useful options in **plot.cna()** to be aware of include the optional flags 'mark.groups' and 'mark.col', which let you to draw and color areas around groups of residues corresponding to the community clustering or any other residue partitioning of interest:

```
colbar.full <- vmd.colors()[net$communities$membership]
colbar.comms <- vmd.colors(max(net$communities$membership), alpha = 0.5)
plot(net, pdb, full = TRUE, mark.groups = grp.col, mark.col = colbar.comms)
```

Figure 7: $network colored according to communities grouping

Network manipulation The **prune.cna()** function allows one to remove communities under a certain size or with less less than a certain number of edges to other communities prior to visualization or further analysis. For example, to remove communities composed of only 2 residues:

```
net.pruned <- prune.cna(net, size.min = 3)
```

```
## Removing Nodes: 7, 8, 9, 16, 17, 5, 14, 15
## id   size    edges   members
## 5    2   1   28:29
## 7    1   0   86
## 8    1   0   87
## 9    2   0   88:89
## 14   2   1   127:128
## 15   1   1   184
## 16   2   0   185:186
## 17   2   0   187:188
```

```
par(mfcol = c(1, 2), mar = c(0, 0, 0, 0))
plot.cna(net)
plot.cna(net.pruned)
```

Applying this command to our network example, you can see how the communities composed by less than 3 residues are deleted from the graph.

Summary

In this vignette we have confined ourselves to introducing relatively simple correlation network generation protocols. However, the individual Bio3D functions for network analysis permit extensive customization of the network generation procedure. For example, one can chose different node representations (including all heavy atoms, residue center of mass, Calpha atoms for each residue, or collections of user defined atoms), different correlation methods (including Pearson and mutual information), different community detection methods (including Girvan-Newman betweenness, Random walk, and Greedy modularity optimization approaches), as well as customized network layouts and result visualizations. We encourage users to explore these options and provide feedback as appropriate.

Although the protocols detailed herein can provide compeling results in a number of cases, defining an optimal procedure for network generation is an area of active research. For example rearrangement of some residues in the communities and changes in the total number of communities may be observed among different community structures based on different independent MD simulations of the same system. These variations of community structures are partially due to the small changes in the calculated cij values that reflect statistical fluctuations among independent simulations. However, changes in mutual information or correlation values may still be relatively small and differences among independent community structures may be largely due to limitations of the modularity metric, a methodology that has been developed for binary adjacency matrices, where edges are counted without taking into account their lengths. We are currently

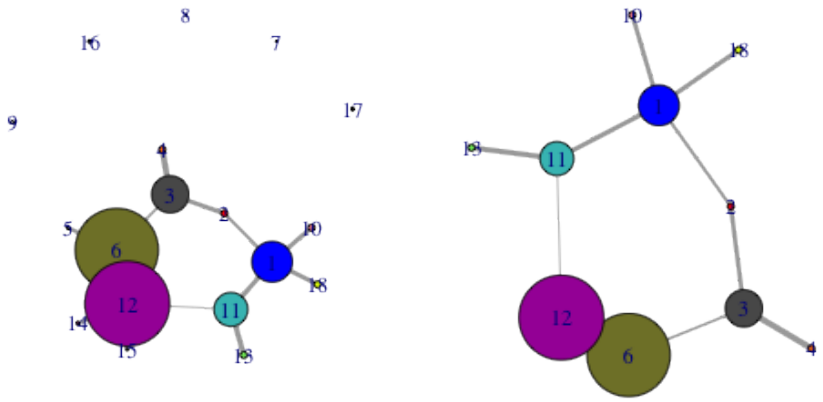


Figure 8: Original (left) and pruned (right) networks

developing a modularity methodology based on natural familial networks, where the strength of a community is defined by the number of close connections between members.

To analyze the interactions that generate the changes in the community network, we must focus on specific hydrophobic, ionic, and hydrogen-bonding interactions associated with the amino acid residues identified by the community network analysis.

References

[1] Sethi A, Eargle J, Black AA, Luthey-Schulten Z. Proc Natl Acad Sci U S A.2009, 106(16):6620-5.

TO BE FINISHED