

```
#####
####Code S1: Julia script for computing the weighted network from a trajectory##
#####
function compute_distances(matrix,adj, layer,num_amino)
    cutoff_sq = 49.0      # 7*7 cutoff distance for amino acids interactions

    for i in 1:num_amino
        for j in (i+1):num_amino
            distances = sum((matrix[i, :] .- matrix[j, :]).^2)
            if distances <= cutoff_sq
                adj[i,j] +=1
            end
        end
    end

    return
end

function read_pdb_file(filename)
    num_layers = 87300          # total number of relaxed structures
    counter = 0
    layer = 1

    # Dictionary for three letters to one letter codes
    three_to_one_letter = Dict{
        "ALA" => "A", "ARG" => "R", "ASN" => "N", "ASP" => "D", "CYS" => "C", "GLN" =>
"Q", "GLU" => "E", "GLY" => "G", "HSD" => "H",
        "ILE" => "I", "LEU" => "L", "LYS" => "K", "MET" => "M", "PHE" => "F", "PRO" =>
"P", "SER" => "S", "THR" => "T", "TRP" => "W",
        "TYR" => "Y", "VAL" => "V",
        "ASX" => "B", # Aspartic acid or Asparagine
        "GLX" => "Z"  # Glutamic acid or Glutamine
    }

    # Read the PDB file only first frame for number of amino acids
    open(filename, "r") do file
        for line in eachline(file)
            if startswith(line, "ATOM")
                counter += 1
            else
                break
            end
        end
    end
    num_amino = counter
    coordinates = zeros(num_amino,3)          # number of amino acids
    adjacency = zeros{Int64,num_amino,num_amino} # Store x, y, z coordinates
    names_aa = Array{String}(undef,num_amino) # Store adjacency matrix accumulated

    counter = 0
    # Read the PDB file only first frame for names of residues
    open(filename, "r") do file
        for line in eachline(file)
            if startswith(line, "ATOM")
                counter += 1
                atom_name = strip(line[13:16])
                residue_name = strip(line[18:20])
                chain_id = strip(line[22:22])
                residue_id = parse{Int, strip(line[23:26])}
                names_aa[counter] = residue_name
            end
        end
    end
end
```

```

        if counter == num_amino
            break
        end
    end

end

end

counter = 0
# Read the PDB file
open(filename, "r") do file
    for line in eachline(file)
        if startswith(line, "ATOM")
            counter += 1
            atom_name = strip(line[13:16])
            residue_name = strip(line[18:20])
            chain_id = strip(line[22:22])
            residue_id = parse(Int, strip(line[23:26]))
            x_coord = parse(Float64, strip(line[31:38]))
            y_coord = parse(Float64, strip(line[39:46]))
            z_coord = parse(Float64, strip(line[47:54]))

            # Append x, y, z coordinates to the matrix
            coordinates[counter,1] = x_coord
            coordinates[counter,2] = y_coord
            coordinates[counter,3] = z_coord

            if counter == num_amino # Compute adjacencies only when one reaches
the final atom of the frame
                compute_distances(coordinates, adjacency, layer, num_amino)
                coordinates = zeros(num_amino,3) # Reset x, y, z coordinates, not
necessary can be avoided
                if mod(layer,num_layers) == 0 # Print out adjacencies when the last
frame is reached
                    for i in 1:num_amino
                        for j in i:num_amino
                            if adjacency[i,j] > 0
                                #println(three_to_one_letter[names_aa[i]], "_", i,
"\t" ,three_to_one_letter[names_aa[j]], "_", j, "\t",
1.0*adjacency[i,j]/(num_layers*1.0))
                                println(i, "\t" ,j, "\t",
1.0*adjacency[i,j]/(num_layers*1.0))
                            end
                        end
                    end
                    adjacency = zeros(Int64,num_amino,num_amino) # Store adjacency
matrix accumulated
                    counter = 0
                    layer += 1
                end
            end
        end
    end

return
end

# Usage example: the following example contains the pdb frames for Calpha atoms
# run the script with "julia process-pdb-file-aggregated.jl"
read_pdb_file("CA-three-runs.pdb")

```

```
#####
##Code S2: Python script for computing the autocorrelation function#
#####
import numpy as np
import matplotlib.pyplot as plt
plt.switch_backend('TkAgg')

def autocorrelation(x, lag):
    n = len(x)
    mean_x = np.mean(x)
    numerator = np.sum((x[:n-lag] - mean_x) * (x[lag:] - mean_x))
    denominator = np.sum((x - mean_x)**2)
    return numerator / denominator

def normalized_autocorrelation(x, lag):
    acf_k = autocorrelation(x, lag)
    acf_0 = autocorrelation(x, 0)
    return acf_k / acf_0

# Example time series collected from VMD from instance
time_series = np.loadtxt('lid-sbd.dat')

# Compute normalized autocorrelation for various lags
lags = range(0, 12220) # You can adjust the range of lags
nacf_values = [normalized_autocorrelation(time_series, lag) for lag in lags]

# Plot the results
plt.plot(lags, nacf_values, marker='o', linestyle='-')
plt.xlabel('Lag')
plt.ylabel('Normalized Autocorrelation')
plt.title('Normalized Autocorrelation Function')
plt.grid(True)
plt.show()

np.savetxt('nacf-lid-sbd.dat', np.column_stack((lags, nacf_values)), fmt='%d %f')
```