Standard Operating Procedure:
Calculating R3m from Molecular Dynamic Simulation Data

Author: Kevin DeBoyace

January 2019

Abstract

This document will describe the procedure used to calculate the molecular descriptor R3m from individual molecule coordinates retrieved from a molecular dynamics (MD) simulation run in Materials Studio.

Contents

1 Introduction

This document will describe the procedure and the MATLAB code used to calculate the molecular descriptor R3m from individual molecule coordinates retrieved from a molecular dynamics (MD) simulation run in Materials Studio. It is assumed that the user has previously run a MD simulation starting with an amorphous cell built using a single molecule. This document will describe exporting the data from the simulation and applying the MATLAB code described to calculate the molecular descriptor R3m for each individual molecule in each of the selected frames from the simulation. The output is a large number of R3m values for which a distribution can be determined. This information is expected to be useful for determining if the R3m value calculated using only the SMILES file is representative of the amorphous form. Additionally, the width of the R3m distribution may help to identify molecules of interest (e.g. more flexible molecules may have a wider distribution). This data could help to explain outliers in the future. Similar methods may also be useful to explore the impact of the presence of solvent on the molecule conformation and subsequent R3m value.

2 Exporting the 3D Atomistic Files

After the MD simulation has been completed, the first step is to export several frames from the simulation. In this context, 'frame' refers to a single step of the simulation, and is a snapshot of the system. An example of a single frame of the simulation is shown in Figure 1.

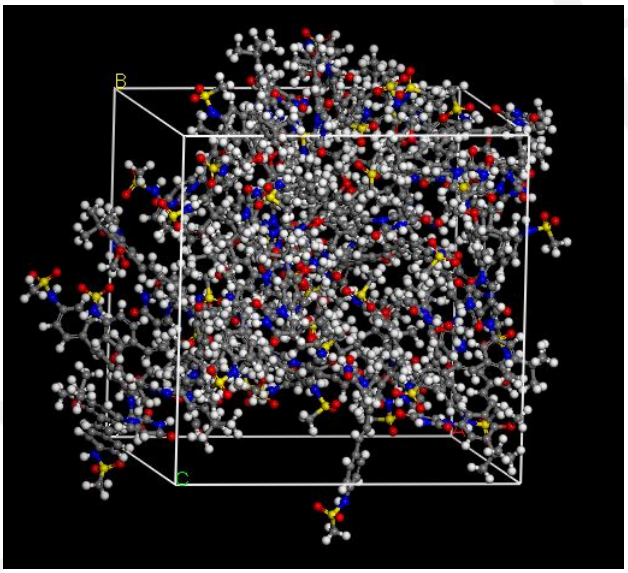

Figure 1: A frame of the MD simulation

The trajectory file which contains all of the simulation data will have a .xtd extension. Right click on the file and select 'Properties' to determine the file extension. The icon should look like the one shown in Figure 2.

Figure 2: Icon for the trajectory (.xtd) file.

Once the file is open, the user should open the 'animation' toolbar if it is not already available. Go to View > Toolbars > Animation. The toolbar should now appear, and should match Figure 3.

Figure 3: Animation toolbar

In the aimation toolbar, press the 'Animation mode' button dropdown arrow. This button will look like a looping arrow. From the dropdown, select 'Options'. A new pop-up dialog will appear as shown in Figure 4.

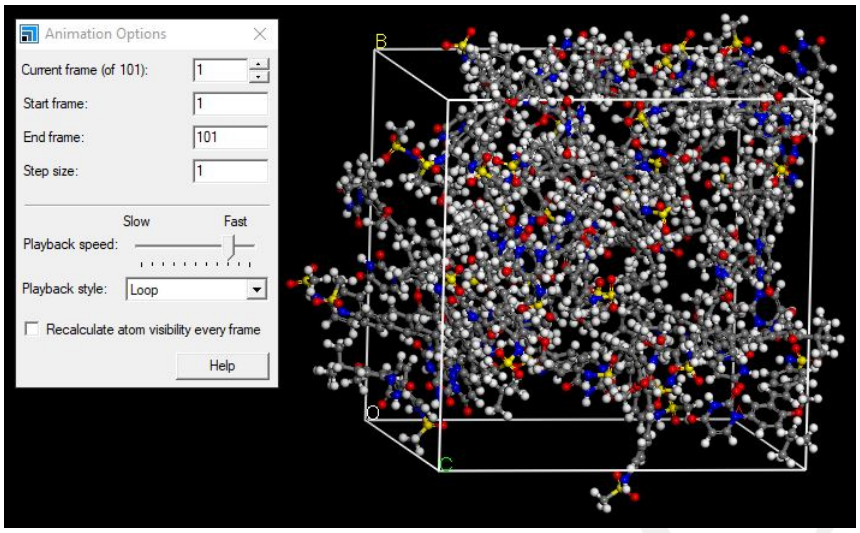

Figure 4: Animation Options dialog box shown alongside the 1st frame of the simulation.

Changing the value in the 'Current frame (of 101)' will update the position of the molecules. The total number of frames will always be 101. It is important to note that there are far more steps in the simulation than 101. As a result, to estimate the approximate location of the frame within the simulation, use percent of duration. For example, frame 40 will show a frame from the simulation at approximately 40% through the total time. The goal is to select multiple frames from the simulation once the system has equilibrated. For example, the first frame of the simulation is unlikely to be representative of the system that is being simulated because in this case the cell had a specified volume. Time is needed for the simulation to reach equilibrium, and hopefully, a more realistic cell volume. In this case, 40% of the way through the simulation corresponded with an equilibrated system. In total, 7 frames will be taken from the simulation, spaced periodically throughout the equilibrated range. In this case, frames 40, 50, 60, 70, 80, 90, and 100 will be exported. In this example, each frame contains coordinates for 40 individual molecules. Type '40' into the 'Current frame (of 100)' field and press the tab key. The selected frame should appear in the trajectory file. See Figure 5.

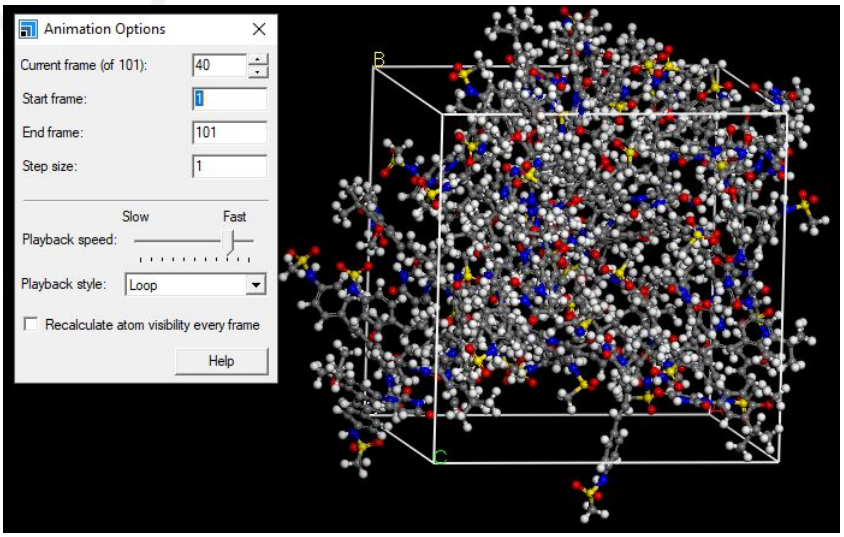

Figure 5: The 40th frame of the simulation.

The selected frame can now be exported by navigating to File > Export. Save the file as a .xsd

file in a new folder with a unique name. Repeat the export process for frames 50, 60, 70, 80, 90, and 100.

Repeat the process described above for each replicate simulation that was run, if any. For this example, the simulation had been run in triplicate. This resulted in 21 individual .xsd files.

3 Run the MATLAB Code

Once the frames (.xsd files) have been saved to the folder of your choice, the MATLAB code can be run.

> **IMPORTANT**: The user must have OpenBabel installed for the code to run successfully. This software is called within the MATLAB code to convert from an .xyz file to an .sdf file.

The primary function is 'GetAtomCoords.m'. When run, this function will prompt the user to select all appropriate files. Hold shift or ctrl and click to select multiple files when prompted. The function will then find the appropriate information within the file (such as atom coordinates), identify individual molecules, and subsequently calculate R3m for each of the molecules. In this example, the result will be 840 values for R3m which correspond to a distribution of conformations of the molecule of interest.
To run the function, open MATLAB and type 'GetAtomCoords' or '[outputvarname] = GetAtomCoords' into the MATLAB workspace. The user will be prompted to select the files. Once the files are selected, press OK. A dialog box will appear that will show the progress of the calculation. See Figure 6. The calculation may take several minutes to complete.

Figure 6: Once the code is running, a dialog will appear that will give the current progress of the calculation.

When the calculation is finished, a histogram plot of the distribution of R3m values will appear. This file will automatically be saved to the current directory as 'R3m_histogram.jpg' with a 600 dpi resolution. If unsure what the current directory is, type 'pwd' into the MATLAB workspace. An example of the resulting plot is shown in Figure 7. The user should change the file name before running the code again, since any file with the same name will be overwritten.

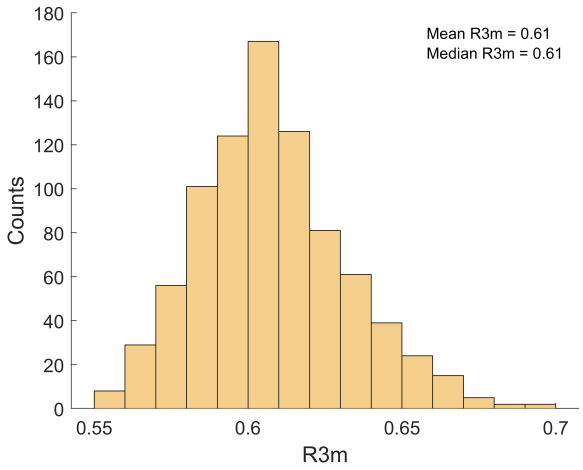

Figure 7: An example of the histogram that results from running the MATLAB code.

A matrix of R3m values will also be output to the MATLAB workspace as either 'ans' or the user

defined output variable name. It is recommended that the user rename the matrix as the drug name and save the workspace for future use. This way, the user will not need to re-run the code in the future.

A series of .xyz and .sdf files will also be output to the current working directory. These files will be all of the molecules from the final step in the code. In this case, a total of 40 of each file was output. Only the data from the last frame is output since these files are overwritten in each loop. The user can open these files in software such as Mercury. Example output conformations can be seen in Figure 8.

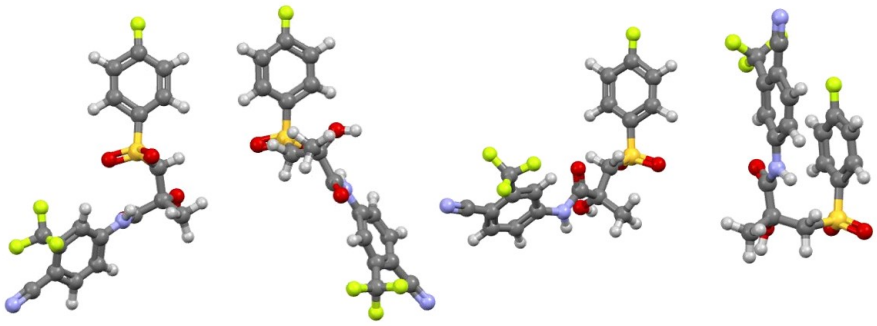

Figure 8: Example output conformations.

4 MATLAB code

The MATLAB code used to perform the calculations described above is included below. The code can be obtained online at the MathWorks file exchange via this link.

4.1 GetAtomCoords.m

```
function [r3m_out] = GetAtomCoords()
%GetAtomCoords_multi - This function allows the user to select data exported from
%multiple Materials Studio files (.xsd files converted to .txt format) and
%calculate R3m values for each molecule in each frame. A distribution of
%R3m values is then plotted as histogram.
%   This function allows the user to select text files to
%   determine the distribution of R3m values that result from
%   conformational differences in molecular structures as determined from
%   Molecular Dynamics Simulations performed using Materials Studio. The
%   user must export .xsd files from the MD run at multiple frames and
%   then save the files in .txt format. This function can then be used.
%
%   See also: R3mCalculate_auto, AtomConnection, AtomicWeighting,
%   EuclidDistance, InfluenceDistanceMat, MolecInfluenceMatrix, MolecMatrix
%
%   Code Author: Kevin DeBoyace
%                Duquesne University
%
%   Last Updated:   January 2019
%
%   NOTE: The Materials Studio data must be saved in .txt format. To
%   accomplish this:
```

```
%       (1) Export the trajectory file from Materials Studio
%       (2) Open the file in notepad and save in .txt format
%       (alternatively, run a script from the command line to batch convert
%       the files).
%       (3) Move the file to the appropriate directory (e.g. your MATLAB
%       directory
%       (4) Run this function
%
%
%% IMPORT MULTIPLE FILES
[files_in, path_in] = uigetfile('*.txt','Select the INPUT DATA FILE(s)', 'Multiselect', 'on');

%% Wait bar
w = waitbar(0,'Initiating...');

%% FOR 1 FILE SELECTED
if isstr(files_in) == 1;

    file_char = ([path_in files_in]); % Import multiple files
    filename = file_char;
    filecontent = fileread(filename);

    %% Wait bar
    w = waitbar(0,'Wait');

    %% Set number of molecules in system
    num_mol = 40;

    %% Get Data
    data1 = regexp(filecontent, '<Atom3d ID=.....', 'Match');
    data2 = regexp(filecontent, '<Bond ID=.....', 'Match');

    % Atom ID
    atom_id = extractBetween(data1, '<Atom3d ID="', '"');
    atom_id = str2double(atom_id); % convert to numeric

    % Molecule ID
    molecule_id = extractBetween(data1, ' Parent="', '"');
    molecule_id = str2double(molecule_id); % convert to numeric

    min_mol = min(molecule_id);
    max_mol = max(molecule_id);

    % Coordinates
    xyz = extractBetween(data1, ' XYZ="', '"');
    coords = regexp(xyz, ',', 'split');
    coords = vertcat(coords{:});
    coords = str2double(coords); % convert to numeric
    clear xyz

    % Get Box limits
```

```
74      max_x = max(coords(:,1));
75      min_x = min(coords(:,1));
76      max_y = max(coords(:,2));
77      min_y = min(coords(:,2));
78      max_z = max(coords(:,3));
79      min_z = min(coords(:,3));
80
81      % Get Cell Parameters
82      data3 = regexp(filecontent, 'AVector
            =.............................', 'Match');
83      cell_param = extractBetween(data3, '"', ',0,0"');
84      cell_param = str2double(cell_param);
85
86      % Get Bond ID
87      forBond = regexp(filecontent, '<Bond ID =.........', 'Match');
88      bond_id = extractBetween(forBond, '<Bond ID="', '"');
89      clear forBond
90
91      % Get Connections
92      connections = extractBetween(data2, 'Connects="','"');
93      connect = regexp(connections, ',', 'split'); %column 1 = x, column
            2 = y, column 3 = z
94      connect = vertcat(connect{:});
95      connect = str2double(connect); % convert to numeric
96      num_connect = size(connect,1)/num_mol;
97
98      % Get Atoms
99      forAtom = regexp(filecontent, 'Components =........', 'Match');
100     atom = extractBetween(forAtom, 'Components="','"');
101     clear forAtom
102
103     connect_actual = 1:num_connect;
104
105     minMol = min(molecule_id);
106     maxMol = max(molecule_id);
107
108     % Save to .xyz format
109     for hh = minMol:maxMol;
110         waitbar(hh/maxMol, w, 'Calculating...');
111         mol1 = find(molecule_id == hh);
112         atom_id_1 = atom_id(mol1);
113         coords_1 = coords(mol1,:);
114         connect_1 = connect(connect_actual,:);
115         connect_actual = connect_actual(end):connect_actual(end)+
                num_connect;
116         atom_1 = atom(mol1);
117         for ii = 1:size(atom_id_1,1);
118             % ii = atom row of interest
119             [row, ~] = find(atom_id_1(ii) == connect_1); %  row in
                    connect where atom is found (column 1)
120             connected = connect_1(row,:);
121             % Get coordintes for relevant atoms
122             for jj = 1:size(connected,1);
123                 for kk = 1:size(connected,2);
124                     coord_find(jj,kk) = {coords_1(find(atom_id_1==
                            connected(jj,kk)),:)};
125                 end
126             end
```

```
            % XYZ format
            % number of elements
            % comment line
            % <elemente> <X> <Y> <Z>
            % ...

            % First line: num_atoms;
            % comment line
            % atom_1; coords_1;
            num_atoms = size(atom_1,1);
            atom_label = char(atom_1);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % NOTE: data normalized to 1 (i.e. relative distance is the same, but overall
            % distance is no longer representative. Must convert back to Angstroms
            % HOW? ---> USE CELL VOLUME
            coords_conv = coords_1*cell_param; % Use cell parameter to convert back to angstroms
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            file_xyz = ['testOut',num2str(hh),'.xyz'];
            fileID = fopen(file_xyz, 'wt');
            fprintf(fileID, '%d\n', num_atoms);
            fprintf(fileID, 'NAME\n');
            for ii = 1:size(atom_1,1);
                %fprintf(fileID, '%s\t', '%i\t %i\t% i\t\n', atom_label, coords_1);
                fprintf(fileID, '%s\t %i\t %i\t% i\t\n',atom_label(ii,:), coords_conv(ii,:));
            end
            fclose(fileID);
            clear fileID
        end
        %% Convert from xyz to sdf to allow calculation of R3m
        % Run Open Babel ot convert
        % see https://openbabel.org/docs/dev/Command-line_tools/babel.html
        % -c = center coordinates
        % -o = specify output format (e.g. -osdf --> output to sdf)
        copyfile(['testOut',num2str(hh),'.xyz'], 'testOut.xyz'); %Copy file to run in Babel
        %!obabel testOut.xyz -c -O Out.sdf
        [yy, zz] = system('obabel testOut.xyz -c -O Out.sdf');
        clear yy zz
        copyfile('Out.sdf', ['testOut',num2str(hh),'.sdf']); % Copy output file from Babel to new file name
        %% Calculate R3m
        pathname = cd;
        file_sdf = ['testOut',num2str(hh),'.sdf'];
        filename = [pathname,'\', file_sdf]; % stitch together for full path
        [r3m] = R3mCalculate_auto(filename);
        r3m_out((hh-minMol+1),1) = r3m; % Output r3m value to vector


end

%% FOR MULTIPLE FILES SELECTED
```

```
176 else
177     files_in = files_in';
178     for aa = 1:size(files_in,1);
179
180         file_char = ([path_in files_in{aa}]); % Import multiple files
181         filename = file_char;
182         filecontent = fileread(filename);
183
184         % Set number of molecules in system
185         num_mol = 40;
186
187         % Get Data
188         data1 = regexp(filecontent, '<Atom3d ID
                =.....................................................................
                ', 'Match')';
189         data2 = regexp(filecontent, '<Bond ID
                =.....................................................', '
                Match')';
190
191         % Get Atom ID
192         atom_id = extractBetween(data1, '<Atom3d ID="', '"');
193         atom_id = str2double(atom_id); % convert to numeric
194
195         % Get Molecule ID
196         molecule_id = extractBetween(data1, ' Parent="', '"');
197         molecule_id = str2double(molecule_id); % convert to numeric
198
199         % Get Coordinates
200         xyz = extractBetween(data1, ' XYZ="', '"');
201         coords = regexp(xyz, ',', 'split');
202         coords = vertcat(coords{:});
203         coords = str2double(coords); % convert to numeric
204         clear xyz
205
206         % Get Box limits
207         max_x = max(coords(:,1));
208         min_x = min(coords(:,1));
209         max_y = max(coords(:,2));
210         min_y = min(coords(:,2));
211         max_z = max(coords(:,3));
212         min_z = min(coords(:,3));
213
214         % Get Cell Parameters
215         data3 = regexp(filecontent, 'AVector
                =.............................', 'Match')';
216         cell_param = extractBetween(data3, '"', ',0,0"');
217         cell_param = str2double(cell_param);
218
219         % Get Bond ID
220         forBond = regexp(filecontent, '<Bond ID =.....', 'Match')';
221         bond_id = extractBetween(forBond, '<Bond ID="', '"');
222         clear forBond
223
224         % Get Connections
225         connections = extractBetween(data2, 'Connects="','"');
226         connect = regexp(connections, ',', 'split'); %column 1 = x,
                column 2 = y, column 3 = z
227         connect = vertcat(connect{:});
```

```
228         connect = str2double(connect); % convert to numeric
229         num_connect = size(connect,1)/num_mol;
230 
231         %NOTE: Connections appear to refer to bond_id
232 
233         % Get Atoms
234         forAtom = regexp(filecontent, 'Components=.....', 'Match')';
235         atom = extractBetween(forAtom, 'Components="','"');
236         clear forAtom
237 
238         % Select only relevent connections (b/c numatoms ~= num
                connections)
239         connect_actual = 1:num_connect;
240 
241         % Identify min and max molecule id numbers for iteration
242         minMol = min(molecule_id);
243         maxMol = max(molecule_id);
244         allMol = unique(molecule_id); % identify all unique values in '
                molecule_id'
245 
246         %% Save to .xyz format
247         for hh = 1:size(allMol,1)
248             mol1 = find(molecule_id == allMol(hh));
249             atom_id_1 = atom_id(mol1);
250             coords_1 = coords(mol1,:);
251             connect_1 = connect(connect_actual,:);
252             connect_actual = connect_actual(end):connect_actual(end)+
                    num_connect;
253             atom_1 = atom(mol1);
254 
255             for ii = 1:size(atom_id_1,1);
256                 [row, ~] = find(atom_id_1(ii) == connect_1); %  row in
                        connect where atom is found (column 1)
257                 connected = connect_1(row,:);
258                 % Get coordintes for relevant atoms
259                 for jj = 1:size(connected,1);
260                     for kk = 1:size(connected,2);
261                         coord_find(jj,kk) = {coords_1(find(atom_id_1==
                                connected(jj,kk)),:)};
262                     end
263                 end
264 
265                 % XYZ format:
266                 % number of elements
267                 % comment line
268                 % <elemente> <X> <Y> <Z>
269                 % ...
270 
271                 num_atoms = size(atom_1,1);
272                 atom_label = char(atom_1);
273 
274                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
275                 % NOTE: data normalized to 1 (i.e. relative distance is
                        the same, but overall
276                 % distance is no longer representative. Must convert
                        back to
277                 % Angstroms.
278                 % HOW? ---> use cell volume
```

```
279                 coords_conv = coords_1*cell_param; % Use cell parameter
                        to undo normalization and covert to Angstroms
280                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
281                 file_xyz = ['Out',num2str(hh),'.xyz'];
282
283                 fileID = fopen(file_xyz, 'wt');
284                 fprintf(fileID, '%d\n', num_atoms);
285
286                 fprintf(fileID, 'NAME\n');
287
288                 for ii = 1:size(atom_1,1);
289                     fprintf(fileID, '%s\t %i\t %i\t% i\t\n',atom_label(
                            ii,:), coords_conv(ii,:));
290                 end
291                 fclose(fileID);
292
293                 clear fileID
294
295             end
296
297             %% Convert from xyz to sdf to allow calculation of R3m
298             % Run Open Babel to convert
299             % see https://openbabel.org/docs/dev/Command-line_tools/
                    babel.html
300             % -c = center coordinates
301             % -o = specify output format (e.g. -osdf --> output to sdf)
302
303             copyfile(['Out',num2str(hh),'.xyz'], 'Out.xyz'); %Copy file
                    to run in Babel
304
305             %!obabel Out.xyz -c -O Out.sdf
306             [yy, zz] = system('obabel Out.xyz -c -O Out.sdf'); % Use
                    OpenBable to convert from .xyz to .sdf
307             clear yy zz
308
309             copyfile('Out.sdf', ['Out',num2str(hh),'.sdf']); % Copy
                    output file from Babel to new file name
310
311             %% Calculate R3m
312             pathname = cd;
313             %file taken in by function
314
315             file_sdf = ['Out',num2str(hh),'.sdf'];
316             filename = [pathname,'\', file_sdf];
317             % Call other function: R3mCalculate_aut
318             [r3m] = R3mCalculate_auto(filename); % Calculate R3m for
                    each molecule
319
320             % r3m_out((hh-minMol+1),aa) = r3m;
321             r3m_out(hh,aa) = r3m;
322
323         end
324
325     waitbar(aa/(size(files_in,1)), w, 'Calculating.    Please Wait...');
            % update waitbar
326
327     end
328
```

```
329 end
330 close(w) % close waitbar
331
332 %% Plot a histogram of calculated R3m values
333 col = rand(1,3); % Random color
334 figure;
335 hold on
336 histogram(r3m_out, 'FaceColor', col, 'FaceAlpha', 0.5); % Plot
      histogram
337 mean_r3m = mean(mean(r3m_out)); % Calculate mean
338 median_r3m = median(median(r3m_out)); % Calculate median
339 set(gca, 'FontSize', 12);
340 % Display Mean on plot
341 text(0.7,0.95,['Mean R3m = ',num2str(round(mean_r3m,2))], 'Units', '
      Normalized','FontSize', 10);
342 % Display Median on plot
343 text(0.7,0.9,['Median R3m = ',num2str(round(median_r3m,2))], 'Units', '
      Normalized','FontSize', 10);
344 ylabel('Counts', 'FontSize', 14);
345 xlabel('R3m', 'FontSize', 14);
346
347 print(gcf, 'R3m_histogram', '-djpeg', '-r600') % Save histogram as jpg
      at 600 dpi
348
349 end
```

4.2 R3mCalculate_auto.m

```
1  function [R3m] = R3mCalculate_auto(filename)
2  %R3mCalculate - A function to calculate the R3m molecular descriptor.
3  %   This function calculates the molecular matrix (M), geometry matrix
       (G),
4  %   molecular influence matrix (H), and influence distance matrix (R).
       This
5  %   is accomplished by calling the following subfunctions: ImportSDF,
6  %   MolecMatrix, EuclidDistance, MolecInfluenceMatrix, and
7  %   InfluenceDistanceMat. A GUI is called to select the file type to
       import
8  %   (filetypegui). Next, the molecule is plotted using the PlotMolecule
9  %   function to ensure the 3D coordinates are reasonable. Finally, the
       R3m
10 %   value is calculated.
11 %
12 %   See also: IMPORTSDF, MOLECMATRIX, EUCLIDDISTANCE,
       MOLECINFLUENCEMATRIX,
13 %   INFLUENCEDISTANCEMAT, PLOTMOLECULE, ATOMICWEIGHTING, ATOMCONNECTION
14 %
15 %   References:
16 %       [1] Todeschini R, Consonni V. 2008. Handbook of molecular
17 %           descriptors. ed.: John Wiley & Sons.
18 %       [2] Consonni V, Todeschini R, Pavan M, Gramatica P 2002.
19 %           Structure/response correlations and similarity/diversity
20 %           analysis by GETAWAY descriptors. 2. Application of the
       novel
21 %           3D molecular descriptors to QSAR/QSPR studies. Journal of
22 %           chemical information and computer sciences  42(3):693-705.
23 %
24 %
```

```
%   Code Author: Kevin DeBoyace
%                Duquesne University
%
%   Date:    October 2018

%% Matrices
% M - Molecular Matrix
% G - Geometry Matrix
% H - Molecular Influence Matrix
% R - Influence Distance Matrix

% Molecular Matrix
[ M, Atoms, numatoms, Connectivity] = MolecMatrix( filename );

% Atomic weighting
[ weightedmass, MolecWeight, colorset ] = AtomicWeighting( Atoms );
% Geometry Matrix
[G] = EuclidDistance(M);
% Molecular Influence Matrix
[ H, Leverage ] = MolecInfluenceMatrix( M );
% Influence Distance Matrix
[ R ] = InfluenceDistanceMat( Atoms, Leverage, G );

%% Atom connections
% Find atoms 1, 2 and 3 bond distances away.
[ connections, topology_mat ] = AtomConnection( Connectivity, Atoms );

%% R3m calculation
%  R3m = sum(i to A-1)sum(j>i) R*wi*wj*delta(k;dij) k =1,2..., d
R3m = 0;
for ii = 1:size(Atoms,1)-1;
    for jj = ii+1:size(Atoms,1);
        if topology_mat(ii,jj) == 1;
            calc = R(ii,jj)*weightedmass(ii)*weightedmass(jj);
            R3m = calc + R3m;
        else
        end
    end
end

% R3m

end
```

4.3 AtomConnection.m

```
function [ connections, topology_mat ] = AtomConnection( Connectivity,
    Atoms )
%AtomConnection - A function which determines which atoms are connected
    by
%distances of 1, 2 and 3 bond distances.
%   This function takes in the connectivity table (without the bond
    order
%   column), and outputs a structure containing cell arrays, where each
%   column corresponds to the atom number, and each row corresponds to
    the
%   number of bond distances (e.g. col 2, row 3 represents the atoms
    which
```

```
%    are 3 bond distance away from atom 2.)
%
%    See also: ImportSDF, PlotMolecule, R3mCalculate
%
%    Author: Kevin DeBoyace & Shikhar Mohan
%    Updated: Jan 2019


connect = Connectivity(:,1:2);

for i = 1:size(Atoms,1)
    B = find(connect(:,1) == i);
    C = find(connect(:,2) == i);
    Temp.A{1,i} = [connect(B,:);connect(C,:)];
    Temp.A{1,i} = reshape(Temp.A{1,i},1,[]);
    for j = 1:size(Temp.A{1,i},2);
        if Temp.A{1,i}(j) == i
            Temp.A{1,i}(j) = 0; %Replace where row = value with zeros (
                Itself present in its own row)
        else
        end
    end
    %Delete zeros
    test = Temp.A{1,i};
    test(test==0) = [];
    Temp.A{1,i} = test;
end
% remove empty cells

clear B C i j test


% Working 2 bond connection - Need to delete atom self reference

for i=1:size(Atoms,1)
    for jj = 1:size(Temp.A{1,i},2)
        B = find(connect(:,1) == Temp.A{1,i}(jj));
        C = find(connect(:,2) == Temp.A{1,i}(jj));
         for kk = 1:size(connect(B,:),2)
                D{i, jj} = reshape([connect(B,:);connect(C,:)], 1, []);
        end
    end
end

% Place values into Temp structure
for i = 1:size(D,1);
    Temp.A{2,i} = [D{i,:}];
end
clear B C D E F i ii j jj k kk

% Delete atoms which are 1 bond distance away.
for i = 1:size(Temp.A,2)
    for k = 1:size(Temp.A{1,i},2)
        [~, col] = find(Temp.A{1,i}(k) == Temp.A{2,i});
        Temp.A{2,i}(col) = [];
    end
end
clear i k col row
```

```
65
66 % Delete where column = atom number (where atom itself is included in
       bond
67 % connections)
68
69 for i = 1:size(Temp.A,2)
70     if isempty(Temp.A{2,i}) == 0; % NEW
71         a = find(Temp.A{2,i} == i);
72         Temp.A{2,i}(a) = [];
73     elseif isempty(Temp.A{2,i}) == 1; % NEW
74             Temp.A{2,i}(a) = []; % NEW
75
76 end
77
78
79 %  3 Bond Distances
80
81 for i=1:size(Atoms,1) % NEW
82     for jj = 1:size(Temp.A{2,i},2)
83         if isempty(Temp.A{2,i}) == 0;
84             B = find(connect(:,1) == Temp.A{2,i}(jj));
85             C = find(connect(:,2) == Temp.A{2,i}(jj));
86              for kk = 1:size(connect(B,:),2)
87                 if connect(B,kk) ~= i & connect(B,kk) ~= jj
88                     D{i, jj} = reshape([connect(B,:);connect(C,:)], 1,
                           []);
89                 else
90                 end
91              end
92         else
93         end
94     end
95 end
96
97
98 % Place values into Temp structure
99 for i = 1:size(D,1);
100     Temp.A{3,i} = [D{i,:}];
101 end
102 clear B C D E F i ii j jj k kk a
103
104
105 % Delete atoms which are 1 bond distance away.
106 for i = 1:size(Temp.A,2)
107     for k = 1:size(Temp.A{1,i},2)
108         [~, col] = find(Temp.A{1,i}(k) == Temp.A{3,i});
109         Temp.A{3,i}(col) = [];
110     end
111 end
112 clear i k col row
113
114
115 % Delete atoms which are 2 bond distance away.
116 for i = 1:size(Temp.A,2)
117     for k = 1:size(Temp.A{2,i},2)
118         [~, col] = find(Temp.A{2,i}(k) == Temp.A{3,i});
119         Temp.A{3,i}(col) = [];
120
```

```
    end
end
clear i k col row

% Delete duplicate values and sort values
for ii = 1:size(Temp.A,1)
    for jj = 1:size(Temp.A, 2)
        Temp.A{ii,jj} = unique(Temp.A{ii,jj}); % Keeps only unique
            values.
    end
end

connections = Temp;


% Topological distance matrix
topology_mat = zeros(size(Connectivity,1)); %Preallocate matrix

    for ii = 1:size(Atoms,1)
        for jj = 1:size(connections.A{3,ii},2)
            %try
            if isempty(connections.A{3,ii}) == 0;
                topology_mat(ii,connections.A{3,ii}(jj)) = 1;
                topology_mat(connections.A{3,ii}(jj), ii) = 1;
            else
            end
        end
    end



end
```

4.4 AtomicWeighting.m

```
function [ weightedmass, MolecWeight, colorset ] = AtomicWeighting(
    Atoms )
%AtomicWeigthing A function to normalize atomic masses to carbon
%   This function is used to determine the appropriate atomic mass
%   weighting for the calculation of R3m. In the calculation, the
    atomic
%   mass is normalized to carbon.
%   This function also assigns a color to each atom for plotting (see
%   PlotMolecule) and calculates the molecular weight.
%
%   Author: Kevin DeBoyace
%   Updated: Jan 2019
%
%   See also: R3mCalculate, PlotMolecule

atomicweight = zeros(size(Atoms,1),1);

% Check if molecule contains Hydrogens. If not, display warning message
    .
ContainHydrogen = strcmp(Atoms, 'H');
if any(ContainHydrogen) ~= 1;
    h = warndlg('WARNING: This molecule is missing Hydrogens. Hydrogens
        should be added to accurately calculate molecular descriptors.
```

```
        ', 'WARNING');
    waitfor(h) %Pause code until user closes warning message.
else
end
% Calculate molecular weight and select atom colors for those Atoms which
% are present.
for b = 1:(size(Atoms,1));
    if strcmp(Atoms(b,:), 'O ') || strcmp(Atoms(b,:), 'O') == 1;
        atomicweight(b) = 15.9994;
        colorset(b,:) = [1 0 0];
    elseif strcmp(Atoms(b,:),'C ')|| strcmp(Atoms(b,:), 'C') == 1;
        atomicweight(b) = 12.0107;
        colorset(b,:) = [0 0 0];
    elseif strcmp(Atoms(b,:),'N ')|| strcmp(Atoms(b,:), 'N') == 1;
        atomicweight(b) = 14.0067;
        colorset(b,:) = [0 0 1];
    elseif strcmp(Atoms(b,:),'Cl') || strcmp(Atoms(b,:),'Cl ') == 1;
        atomicweight(b) = 35.453;
        colorset(b,:) = [0 0.3 0];
    elseif strcmp(Atoms(b,:),'S ')|| strcmp(Atoms(b,:), 'S') == 1;
        %atomicweight(b) = 32.065;
        atomicweight(b) = 32.066;
        colorset(b,:) = [1 1 0];
    elseif strcmp(Atoms(b,:), 'P ') || strcmp(Atoms(b,:), 'P') == 1;
        atomicweight(b) = 30.973762;
        colorset(b,:) = [1 0.5 0];
    elseif strcmp(Atoms(b,:), 'F ') || strcmp(Atoms(b,:), 'F') == 1;
        atomicweight(b) = 18.998;
        colorset(b,:) = [0.5 0.8 0.4];
    elseif strcmp(Atoms(b,:), 'H ') || strcmp(Atoms(b,:), 'H') == 1;
        atomicweight(b) = 1.00794;
        colorset(b,:) = [0.9 0.9 0.9];
    end
end

% Throw an error if an atom is not listed in the code above. If this error
% is thrown, the atom needs to be added to the portion of the code above to
% ensure correct calculation of Molecular weight and weightedmass.
for b = 1:(size(Atoms,1));
    if atomicweight(b) == 0;
        error('Error: Atom missing from list in code')
    end
end

atomicweight = atomicweight';
MolecWeight = sum(atomicweight);

% Masses are weighted with respect to the carbon atom : mass/ mass carbon
weightedmass = atomicweight / 12.0107;
weightedmass = weightedmass';

end
```

4.5 EuclidDistance.m

```
function [G] = EuclidDistance(M)
%G - Determine Euclidean Distance of atoms in a molecule
%   G = Geometric Distance Matrix
%   This function determines the Euclidean distance between all of the
%   atoms in the molecule and outputs the data to the matrix G, which
    is
%   known as the geometric distance matrix.
%   See also: R3mCalculate, CheckCentroid,
%
% Author: Kevin DeBoyace
%         Wildfong Lab
%         Duquesne University
% Updated: Jan 2019


%G = pdist2(M,M,'euclidean'); % Performs same as below, but is slower

for ii = 1:size(M,1);
    for jj = 1:size(M,1);
        % Calculate Euclidean Distance
        G(ii,jj) = sqrt((M(ii,1)-M(jj,1)).^2 + (M(ii,2)-M(jj,2)).^2 +(M
            (ii,3)-M(jj,3)).^2);
    end
end

end
```

4.6 InfluenceDistanceMat.m

```
function [ R ] = InfluenceDistanceMat( Atoms, Leverage, G )
%InfluenceDistanceMat (R) - Calculation of the Influence Distance
    Matrix
%(R)
%   This function calculates the influence/ distance matrix (R). Its
    initial
%   application was for the calculation of 'R3m', but may be used to
%   calculate any of the R-indices. The function requires input of '
    Atoms',
%   'Leverage', and 'G' (the Molecular Geometry Matrix). These values
    come
%   from the functions 'ImportSDF', 'MolecInfluenceMatrix', and
%   'EuclidDistance', respectively.
%
% See Also: ImportSDF, MolecInfluenceMatrix, EuclidDistance,
    R3mCalculate
%
% Author: Kevin DeBoyace
%         Wildfong Lab
%         Duquesne University
% Updated: Jan 2019


for ii = 1:size(Atoms,1);
    for jj = ii:size(Atoms,1);
        if jj == ii
            R(ii,jj) =0;
```

```
        else
            R(ii,jj) = (sqrt(Leverage(ii)*Leverage(jj))/G(ii,jj));
            R(jj,ii) = (sqrt(Leverage(jj)*Leverage(ii))/G(jj,ii));
        end
    end
end

end
```

4.7 MolecInfluenceMatrix.m

```
function [ H, Leverage ] = MolecInfluenceMatrix( M )
% Molecular Influence Matrix
%   This function is used to calculate the molecular influence matrix (M)
%   from the 3D cartesian coordinates of a molecule.
%
% See Also: MolecMatrix, R3mCalculate
%
% Author: Kevin DeBoyace
%         Wildfong Lab
%         Duquesne University
% Updated: Jan 2019

H = M*pinv(M'*M)*M';
Leverage = diag(H);
end
```

4.8 MolecMatrix.m

```
function [ M, Atoms, numatoms, Connectivity] = MolecMatrix( filename )
%MolecMatrix Summary: Function to calculate the molecular matrix (M)
% This function extracts coordinates from a .sdf file and generates the
% molecular matrix (M). This function also extracts the connectivity table
% and creates a separate matrix for this information. The connectivity
% table contains the bond information.
%
% See Also: R3mCalculate, ImportSDF
%
% Author: Kevin DeBoyace
%         Wildfong Lab
%         Duquesne University
% Updated: Jan 2019

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
delimiter = ' ';
startRow = 2;
formatSpec = '%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';
fileID = fopen(filename,'r');

dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, '
    MultipleDelimsAsOne', true, 'ReturnOnError', false);
```

```
fclose(fileID);
for ii = 1:size(dataArray{1,1},1)
    for jj = 1:size(dataArray,2)
        dataArraynum(ii,jj) = str2double(dataArray{1,jj}{ii});
    end
end

%% Extract cartesian coordinates from file and place them in a single
    matrix.
dataArraynum_cut = dataArraynum(:,1:3); %Take first three columns
datarem = rem(dataArraynum_cut,1);
jj = 1;
for ii = 1:size(dataArraynum_cut, 1);
    if sum(datarem(ii,:)) == 0 || isnan(datarem(ii,1))
        temp_rem(ii) = 0;
    else
        temp_rem(ii) = 1;
        coords(jj,:) = dataArraynum_cut(ii,:);
        Atoms(jj,:) = dataArray{1,4}(ii); % Extract atom names
    end
    jj = jj+1;
end

for ii = 1:size(Atoms,1)
    temp_emp(ii) = isempty(Atoms{ii});
    try
        temp(ii) = str2num(Atoms{ii});
    catch
        temp(ii) = 0;
    end
end
[row,col] = find(temp ~= 0 | temp_emp == 1);
Atoms(col) = [];
coords(col,:) = [];

M = coords; % Matrix of cartesian coordinates
clear ii jj

numatoms = size(M,1);

%% Connectivity table
jj = 1;

for ii = 1:size(dataArraynum_cut,1)
    if (sum(datarem(ii,:))) == 0 && ~isnan(datarem(ii,1)) && sum(
        dataArraynum_cut(ii,1:3)~=0) == 3 %For those numbers which are
        part of connectivity or NaN --> delete
        e(jj,:) = dataArraynum_cut(ii,:); %Build matrix with
            connectivity data
        jj = jj + 1;
    else
        jj = jj;
    end
end

Connectivity = e;

clear ii jj d e f %Clear Temp variables
```

```
79
80 end
```