

---

# Molecular Trajectory Analysis

---

BY  
SUPER GROUP 1

## Group Members

EnrollmentNo	Name
MIT2019015	Aminul Islam
MIT2019016	Ashutosh Singh
MIT2019018	Harish Kumar Rajora
MIT2019025	Jafar Sarif
MIT2019027	Saurabh Singh Patel
MIT2019028	Rajeev Kumar Singh
MIT2019031	Shrikant Balaji Temburwar
MIT2019032	Sriyash Pravin Ingale
MIT2019033	Abhishek Yadav
MIT2019035	Ajay Bhagwanrao Garkal
MIT2019038	Nalla Praveen
MIT2019039	Abhinav Lahariya
MIT2019040	Abhishek Kumar
MIT2019041	Divay Sharma
MIT2019044	Thakkar Shrenik Nileshkumar
MIT2019046	Sumant Srivastav
MIT2019047	Rohit Takalkar
MIT2019048	Abhishek Barman
MIT2019061	Diwakar Mishra
MIT2019072	Mohammed Afzal Ansari
MIT2019074	Bhisikar Siddhant Sanjyot
MIT2019075	Sachin Gupta
MIT2019076	Abhishek Ranglal Gupta
MIT2019077	Pronika Gautam
MIT2019095	Nasalwai Nikhil Chakravarthy
MIT2019099	Musale Rampavan
MIT2019106	Ramji Jaiswal
MIT2019107	Dinesh Kumar Pal
MIT2019108	Bijendra Kaithwas
MIT2019109	Ajit Kumar
MIT2019113	Mukul Madaan
MIT2019115	Ansari Ramprasad
MIT2019116	Dharam Raj
MIT2019118	Harshita Sonkar
MIT2019120	Mohd Saif Khan
MIT2019122	Purnendu Sekhar Saha

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Problem statement . . . . .	4
1.2	Challenges . . . . .	4
1.3	Objective . . . . .	4
<b>2</b>	<b>Overall Description :</b>	<b>4</b>
2.1	Project Perspective: . . . . .	4
<b>3</b>	<b>Installation:</b>	<b>4</b>
<b>4</b>	<b>Project Functions :</b>	<b>5</b>
4.1	PDB generation: . . . . .	5
4.2	Mask: . . . . .	5
4.2.1	Algorithm: . . . . .	6
4.2.2	DFD: . . . . .	6
4.2.3	API Doc: . . . . .	7
4.3	Centre : . . . . .	8
4.4	Auto Imaging : . . . . .	8
4.4.1	Algorithm: . . . . .	8
4.4.2	DFD: . . . . .	8
4.4.3	API Doc: . . . . .	9
4.5	Superposition : . . . . .	10
4.6	Average Structure : . . . . .	10
4.6.1	Algorithm: . . . . .	10
4.6.2	DFD: . . . . .	10
4.6.3	API Doc: . . . . .	11
4.7	Root Mean Square Deviation (RMSD): . . . . .	12
4.7.1	Residue-Wise RMSD : . . . . .	12
4.7.2	Algorithm: . . . . .	12
4.7.3	API Doc: . . . . .	14
4.8	Distances : . . . . .	14
4.8.1	Algorithm: . . . . .	14
4.8.2	API Doc: . . . . .	15
4.9	Angles : . . . . .	16
4.9.1	Algorithm: . . . . .	16
4.9.2	DFD: . . . . .	17
4.9.3	API Doc: . . . . .	18
4.10	Dihedrals : . . . . .	18
4.10.1	Algorithm: . . . . .	18
4.10.2	DFD: . . . . .	19
4.10.3	API Doc: . . . . .	19
4.11	Ankush : . . . . .	21
4.11.1	Occurrences : . . . . .	21
4.11.2	Mamimum Residence Time: . . . . .	21
4.11.3	DFD: . . . . .	21
4.11.4	API Doc: . . . . .	22
4.12	Hydrogen Bonds : . . . . .	22
4.12.1	DFD: . . . . .	22
4.12.2	API Doc: . . . . .	23
<b>5</b>	<b>Technologies Used:</b>	<b>25</b>
5.1	Spark: . . . . .	25
5.2	SciSpark: . . . . .	25
5.3	Scala: . . . . .	26

<b>6</b>	<b>Design :</b>	<b>26</b>
6.1	Software Architecture: Incremental process model . . . . .	26
6.2	DFD Level 0: . . . . .	27
6.3	DFD Level 1: . . . . .	27
6.4	DFD Level 2: . . . . .	28
6.5	Class Diagram . . . . .	29
<b>7</b>	<b>Dependencies</b>	<b>30</b>
<b>8</b>	<b>Group Contribution:</b>	<b>30</b>
8.1	Work Distribution : . . . . .	30
<b>9</b>	<b>Performance Metrics:</b>	<b>31</b>

# 1 Introduction

## 1.1 Problem statement

To develop a set of programs that yields a great deal of information about the structure, dynamics, and interactions of biological macro molecules by performing trajectory analysis on the data generated by Molecular Dynamics Simulation.

## 1.2 Challenges

Huge amount of data is being generated in almost every field and it cannot be avoided, rather is essential for the advancement of the field. Analysis of this data requires intensive computing power. Molecular Simulation is a powerful tool for understanding the behaviour of natural systems. The simulation generates large amount of data while observing the spatial and temporal relationships. The challenge is to handle the analytical queries that are often compute Intensive and to provide the hardware architecture and corresponding software systems that could extract valuable knowledge from these large datasets. The storage of data is not a big deal nowadays but for data processing, map reduce is a trending programming model for processing large data sets with a parallel and distributed algorithms on a cluster. It provides efficient and fast solution for manipulation of large datasets.

## 1.3 Objective

1. Analysis of the input files which is given to the system as frames at regular intervals of time, which contain the position of atoms in the box as .crd file and information about atoms in topology file and our objective is to convert those to a common file with coordinates as well as information in one human readable file (.pdb).
2. Implementation of the following functionalities:
  - (a) Mask/select, centre, imaging, superposition.
  - (b) Average structure, PDB generation, RMSD, Residue-Wise RMSD.
  - (c) Distances, Angles, Dihedrals (backbone, specified set).
  - (d) Ankush: Occurrence, network interactions, Maximum Resistance Time.
  - (e) Hydrogen bonds: list, trajectory, and matrix.

# 2 Overall Description :

## 2.1 Project Perspective:

The following are the main features of this project:-

1. Easily Scalable - As the project is to be developed using Scala, it is easily scalable.
2. Fast processing - Using Spark, efficient parallelism can be achieved.
3. More functionalities - Various functionalities will be implemented in this project.
4. Developed an open-source software

# 3 Installation:

Steps to install software

1. Install jdk 1.8 in your system

## 2. Install Spark 2.3.0 in your Linux system

- (a) Downloading Scala
- (b) Verify Your Java Installation  
Open Command terminal and type `java -version`  
`Export JAVA_HOME=/usr/local/javacurrent`
- (c) Set Your Java Environment  
`Export JAVA_HOME=/usr/local/java-current`
- (d) Append the full path of Java compiler location to the System Path.  
`Export PATH=PATH :JAVA_HOME/bin/`
- (e) Execute the command `java -version` from the command prompt as explained above.
- (f) Install Scala
- (g) Finally, open a new command prompt and type `Scala -version` and press *Enter*.  
You should see the following  
`scala -version`
- (h) Installing Spark
  - Extracting Spark tar:  
`tar xvf spark-2.3.0-bin-hadoop2.6.tgz`
  - Verifying the Spark Installation  
`spark-shell`

## 3. Follow the following procedure to run the application:

- (a) Command to execute the jar:  
`./run.sh <crd_directory_path> <topology_file_path> <output_file> <number_of_crd> <module_name>`
- (b) The path for CRD, Topology and output file needs to be set up in the `run.sh` file.
- (c) `run.sh` is executed, that will take command line arguments of module name user wants to execute.
- (d) Output file will be created at given output path.

# 4 Project Functions :

## 4.1 PDB generation:

The Protein Data Bank (PDB) format provides a standard spatial representation of molecules. It is generated from Topology and Coordinate files. Topology file contains the molecular information and coordinate files contain the 3 dimensional spatial coordinates of atoms.

Algorithm:

1. Fetching the coordinates from CRD file using the nc2 library and storing it into an ArrayList.
2. Reading the Topology file as an Array (topoFileArray) where lines are the element of the array.
3. Extracting Atom Name, Molecule Name, Molecule Chain length and Termination Sequence from the topoFileArray.
4. Writing the Atom Name, Molecule Name, Molecule Chain, Termination Sequence and coordinates into the PDB file.

## 4.2 Mask:

Masking act as a filter which is executed and used in a way to select a particular atom out of the complete data set of atoms. It is basically weeding out the unrequired part and choosing the atoms as per the requirement and command of the user.

4.2.1 Algorithm:

1. Creating a Dataset of Atoms in the form of a row(atomName, atomIndex) from topology file.
2. Creating a Dataset of Coordinates (crdDS) in the form of a row(X, Y, Z, atomIndex) from coordinate file which is in NetCDF format.
3. Filtering the atomsDS based on the input given by the user and storing them in a dataset (atomsDS).
4. Joining the both datasets based on the atomIndex which gives the dataset of masked atoms coordinates.

4.2.2 DFD:

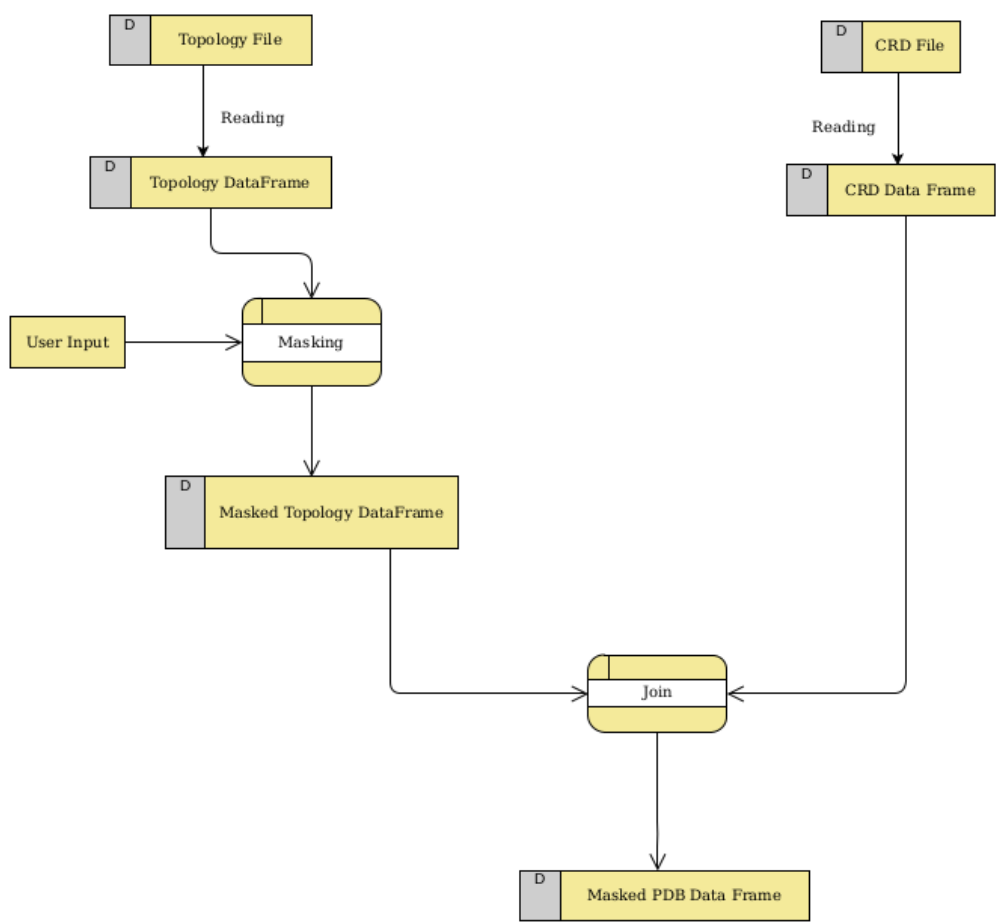


Figure 1: DFD for Masking

4.2.3 API Doc:

C

Masking

Mask

class Mask extends AnyRef

Linear Supertypes

Filter all members

Instance Constructors

new Mask(constant: CONSTANTS, spark: SparkSession)

Value Members

def getAutoImagedDataFrame(): DataFrame

def getCoordinateDataFrame(): DataFrame  
This is used to get coordinate file DataFrame

def getCoordinatesWithWaterDataFrame(): DataFrame

def getListOfFiles(dir: String): List[File]  
To get list of CRD files in given directory

def getMaskedAtomDataFrame(input: String): DataFrame  
This is used to perform masking by atom name

Figure 2: API for Masking

def getMaskedByAtomId(input: String): DataFrame This is used to perform masking by AtomId
def getMaskedByAtomRange(input: String): DataFrame This is used for masking atoms in range
def getMaskedByFrameRange(input: String): DataFrame This is used for masking frames in range
def getMaskedByMoleculeId(input: String): DataFrame This is used to perform masking by MoleculeId
def getMaskedByMoleculeRange(input: String): DataFrame This is used for masking molecules in range
def getMaskedMoleculeDataFrame(input: String): DataFrame This is used to perform masking by Molecule name
def getProteinDataBank(): DataFrame This is used to get protein Data Bank DataFrame
def getTopologyDataFrame(): DataFrame This is used to get topology DataFrame
def getTopologyWithWaterDataFrame(): DataFrame
def maskAtomByIds(atomsId: String, PDB: DataFrame): Dataset[Row]
def maskMoleculeByIds(moleculesId: String, PDB: DataFrame): Dataset[Row]
def maskedByAtomMoleacule(dataFrame: DataFrame): DataFrame This is used to get Masked Atom DataFrame

Figure 3: API for Masking



4.3 Centre :

Move all the atoms to so that the center of the atoms in the masked atoms is centered at a specific location.

4.4 Auto Imaging :

Automatically center and image a trajectory with periodic boundaries. The atoms of the anchor region will be centred, all fixed molecules will be imaged only if imaging brings them closer to the anchor molecule.

4.4.1 Algorithm:

- 1. Read CRD and Topology files.
- 2. ACalculate box center for the given box dimensions in CRD data using findBoxCenter() API
- 3. Calculate the residue center using centerAnchorMolecule() API
- 4. Image all the molecules such that they lie in the box, using imageMolecules() API.
- 5. Generate a PDB for each frame with auto imaged coordinates.

4.4.2 DFD:

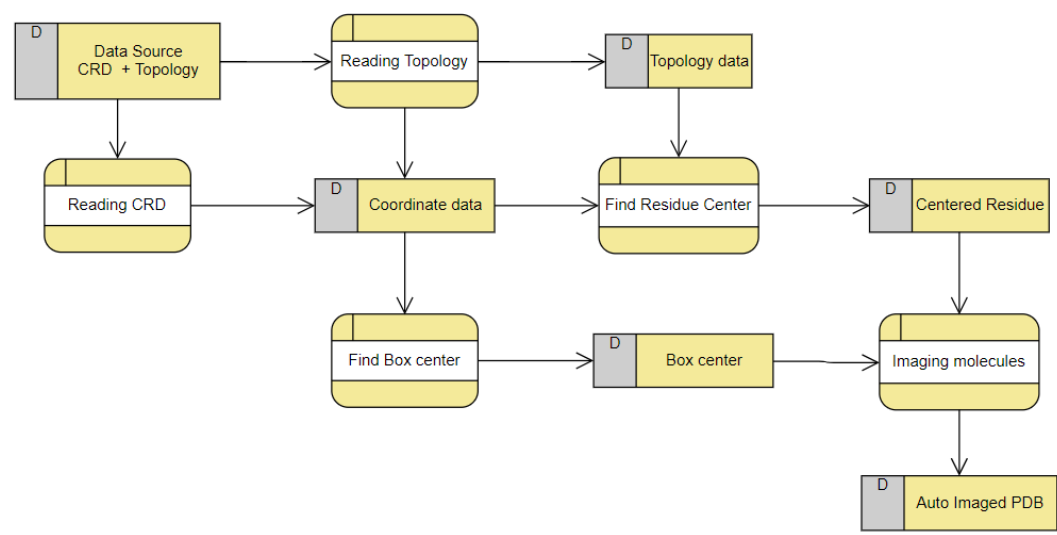



Figure 4: DFD for Autoimaging

4.4.3 API Doc:



AutoImage

abstract class **AutoImage** extends AnyRef

Provide functions to Auto Image molecules in each frame

Center each anchor molecule at box center Image remaining molecules inside the box

Linear Supertypes

Filter all members

Instance Constructors

new **AutoImage**()

Abstract Value Members

abstract def **centerAnchorMolecule**(pdbFile: String, boxCenter: Array[Double]): Unit

centerAnchorMolecule() does the centering of the anchor molecule

**pdbFile**

file path to get coordinates

**boxCenter**

coordinates

abstract def **findBoxCenter**(topologyFile: String): Array[Double]

findBoxCenter() finds the center of the box

**topologyFile**

file to read box dimensions

**returns**

Box center coordinates

abstract def **imageMolecules**(pdbFile: String, boxCenter: Array[Float], boxDimensions: Float): Unit

Images the water molecules which goes out of the box dimensions after the centering

**pdbFile**

file path to get coordinates

**boxCenter**

coordinates

**boxDimensions**

for periodic boundary conditions

Figure 5: API for Auto-Imaging

## 4.5 Superposition :

Superposition is used for visualisation purposes as we try to overlap the structure of the molecule of every frame to the reference frame and as a result we obtain the path followed by the molecule throughout those frames.

## 4.6 Average Structure :

1. Average Structure is to find the average structure of the protein across all the Frames.
2. Since coordinates are being averaged over many frames the resulting structure may appear distorted.

Calculating the average of coordinates of all the atoms in different frames.

### Steps to be followed :

1. Calculate the average of input coordinates.
2. If the number of atoms in frame are less than the total number of atoms in all frames, the topology will be stripped to match using match command.
3. After which we need to remove global rotational and translation movement as they can introduce errors in the average structure. It can be removed using RMSD module.

### 4.6.1 Algorithm:

1. Take the autoimaged crd files and store it in a dataframe.
2. Remove the water molecules from the dataframe.
3. Group the dataframe based on the atomId.
4. Calculate the average of x,y, z co-ordinates.
5. join the output data frame with topology file and write into PDB.

### 4.6.2 DFD:

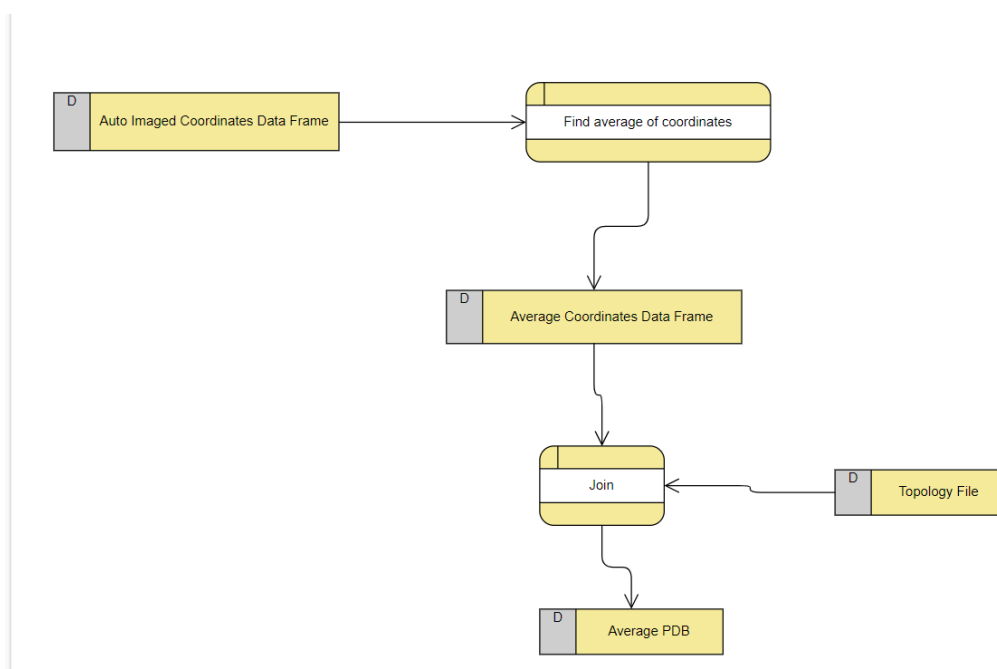



Figure 6: DFD for Average Structure

4.6.3 API Doc:



Average

abstract class **Average** extends AnyRef

Calculating the average of coordinates of the masked atoms in different frames.

Linear Supertypes

Filter all members

Instance Constructors

new **Average**()

Abstract Value Members

abstract def **findAverageDS**(maskedDS: Dataset[String]): Dataset[String]

findAverageDS() will find the average of the masked atoms in all frames

**maskedDS**

cotains the masked molecular information

**returns**

DataSet containing the average of the masked atoms

abstract def **getMaskedDS**(): Dataset[String]

getMaskedDS() will be called to get the masked atomsDS from masking module

**returns**

Dat1aSet contains the masked molecular information

Figure 7: API for Average Structure

## 4.7 Root Mean Square Deviation (RMSD):

The root-mean-square deviation of atomic positions (or simply root-mean - square deviation, RMSD) is the measure of the average distance between the atoms (usually the backbone atoms). It basically measures the deviation of a target set of coordinates (i.e. a structure) to a reference set of coordinates, if RMSD is 0.0 this indicates a perfect overlap.

### 4.7.1 Residue-Wise RMSD :

The ability to graphically display the RMSD per Residue between two structure is a useful feature for demonstrating how well the structure is aligned.

### 4.7.2 Algorithm:

1. In bioinformatics, the root-mean-square deviation of atomic positions (or simply root-mean-square deviation, RMSD) is the measure of the average distance between the atoms (usually the backbone atoms) of superimposed proteins. A widely used way to compare the structures of biomolecules or solid bodies is by Kabsch Algorithm

$$\begin{aligned}\text{RMSD}(\mathbf{v}, \mathbf{w}) &= \sqrt{\frac{1}{n} \sum_{i=1}^n |v_i - w_i|^2} \\ &= \sqrt{\frac{1}{n} \sum_{i=1}^n ((v_{ix} - w_{ix})^2 + (v_{iy} - w_{iy})^2 + (v_{iz} - w_{iz})^2)}\end{aligned}$$

2. The algorithm starts with two sets of paired points, P and Q. Each set of points can be represented as an  $N \times 3$  matrix

#### 3. Kabsch Algorithm

- (a) Select a frame as a reference frame P.
- (b) Calculate centroid of matrix for each frame
- (c) Translates two matrices P, Q so that their centroids are equal to the origin of the coordinate system.

$$A = P^T Q$$

- (d) Or we could also perform singular value decomposition of A:

$$A = V S W^T$$

- (e) Calculates the optimal rotation matrix of two matrices P, Q and superimposes the two matrices

$$d = \text{sign}(\det(W V^T))$$

The optimal rotation matrix U is calculated

$$U = W \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} V^T$$

- (f) Implementation of Kabsch Algorithm was done using Chemistry Development Kit(CDK) Library.

DFD:

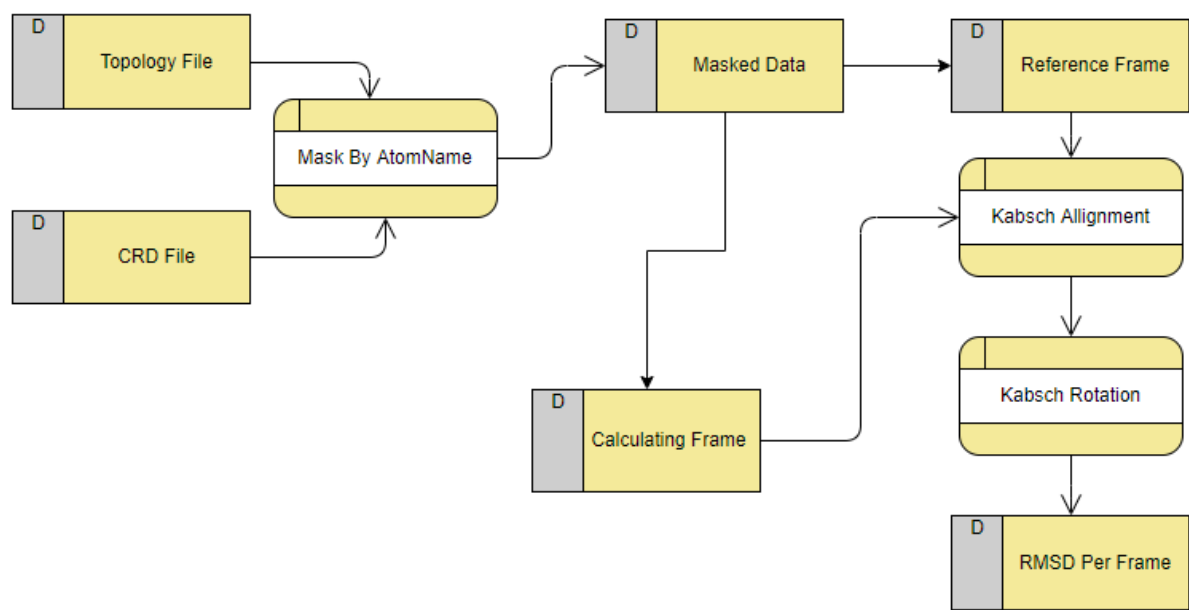


Figure 8: DFD for RMSD By Atom

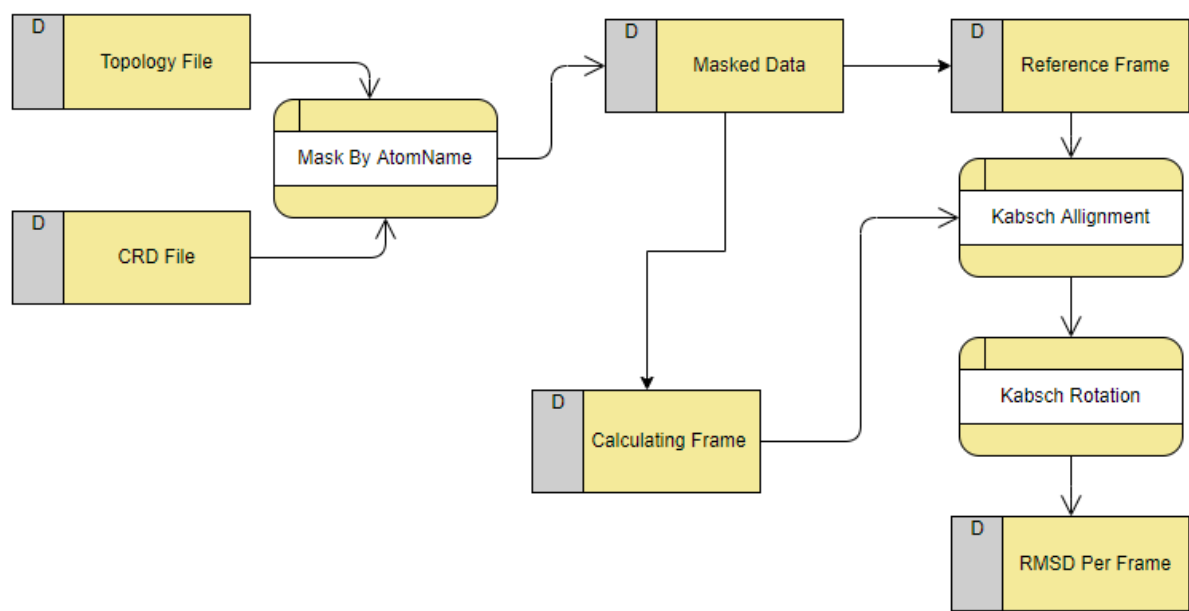


Figure 9: DFD for RMSD By Residue

### 4.7.3 API Doc:

```
//package org.openscience.cdk.geometry.alignment
import java.io.File
import org.openscience.cdk.AtomContainer
/**
 *The ability to graphically display the RMSD per Residue between two structure is a useful
 *feature for demonstrating how well the structure is aligned.
 */
abstract class RMSD{

    /**gets the coordinates of the first frame which is considered as the reference frame
     * @param Takes frame coordinate as array
     * @return returns AtomContainer
     */
    def makeContainer(CoordinateArr[]) : AtomContainer{

    }

    /**finds RMSD of calculating frame with respect to reference frame using CDK Library
     * @param Reference frame AtomContainer
     * @param Calculating frame AtomContainer
     * @return RMSD value
     */
    def findRMSD(refAtomContainer, nextAtomContainer): Double{

    }
}
```

### 4.8 Distances :

Distances can be calculated between two atoms or two molecules. For calculating the distance between two molecules, we can either use their geometric center or center of mass, thus selecting c-alpha atom of the molecule as the center of mass.

#### 4.8.1 Algorithm:

1. Mask the data as per the user input for the selected atoms/molecules in the form of frame no, id x, y, z
2. For every frame, there will be two rows (as 2 atoms or molecules are selected) in the same form
3. Iterate frame-wise and fetch pair of row containing the points of coordinates to the function which returns the distance.
4. Store the dataframe in the output file in the form of frame no. and distance.

DFD:

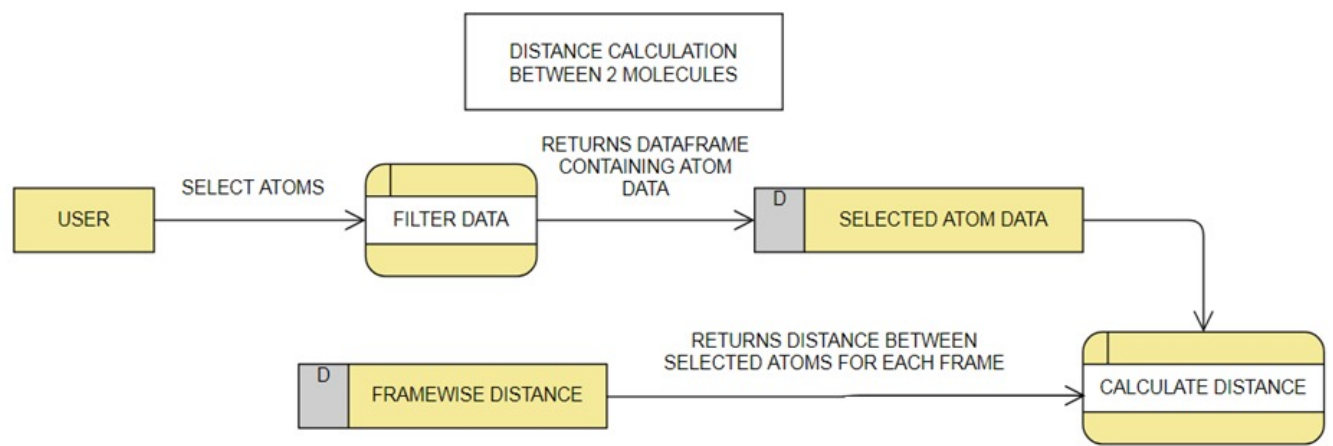


Figure 10: DFD for Molecular Distance

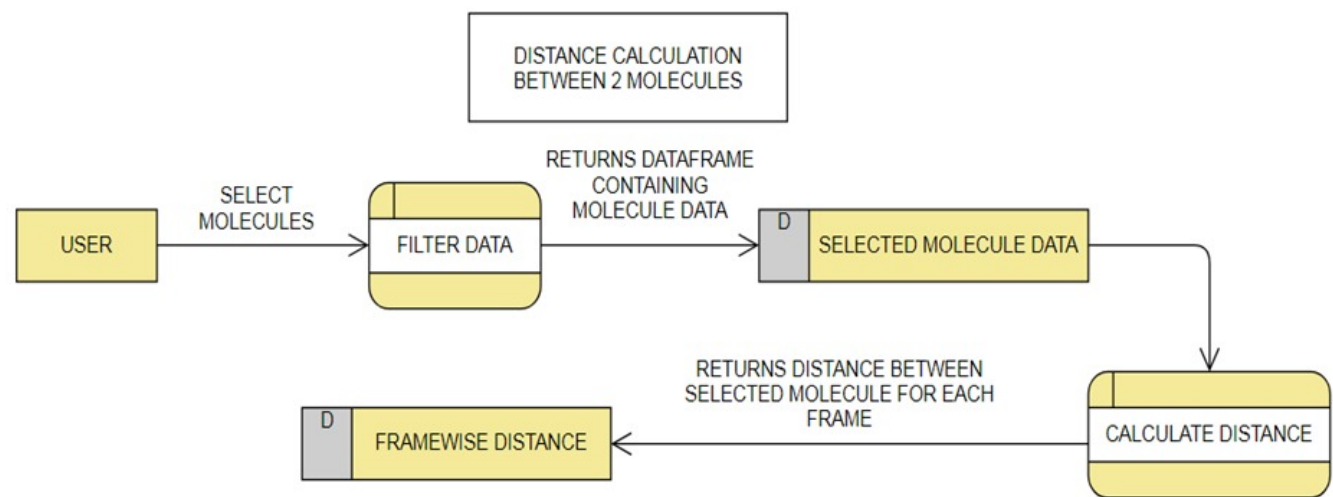


Figure 11: DFD for Atomic Distance

4.8.2 API Doc:

Value Members	
<pre>def calculateDistance(point1: Row, point2: Row): Double</pre>	
<b>point1</b>	- Row containing ,frame_no, id and x,y,z coordinates of first molecule/atom
<b>point2</b>	- Row containing ,frame_no, id and x,y,z coordinates of second molecule/atom
<b>returns</b>	Distance of the given two atoms/molecules in every frame
<pre>def calculateDistanceBetweenAtoms(atomData: Array[Row]): Unit</pre>	
<b>atomData</b>	Data containing the coordinates of the selected molecules in each frame
<pre>def calculateDistanceBetweenMolecules(moleculeData: Array[Row]): Unit</pre>	
<b>moleculeData</b>	Data containing the coordinates of the selected molecules in each frame

Figure 12: API for Distances



## 4.9 Angles :

The angle ( in degrees ) between atoms is calculated. We will use concept of vectors to calculate angle from base atom to other two atoms.

### 4.9.1 Algorithm:

Angle Calculation for Atoms:

1. Take dataset from masking module.
2. Ask user to enter the atoms for which he/she wants to find angle (3 atoms).
3. Make second atom as base atom and find vectors accordingly.
4. Calculate angle using following cosine formula.

$$\theta = \cos^{-1} \left( \frac{u.v}{\|u\|.\|v\|} \right)$$

where u,v are vectors.

Angle Calculation for Molecules:

1. Take dataset from masking module.
2. Ask user to enter the molecules for which the user wants to find angle (3 molecules).
3. Find geometric center of entered molecules.
4. Make second molecule as base molecule and find vectors accordingly.
5. Calculate angle using the same cosine formula for angle calculation of atoms.

4.9.2 DFD:

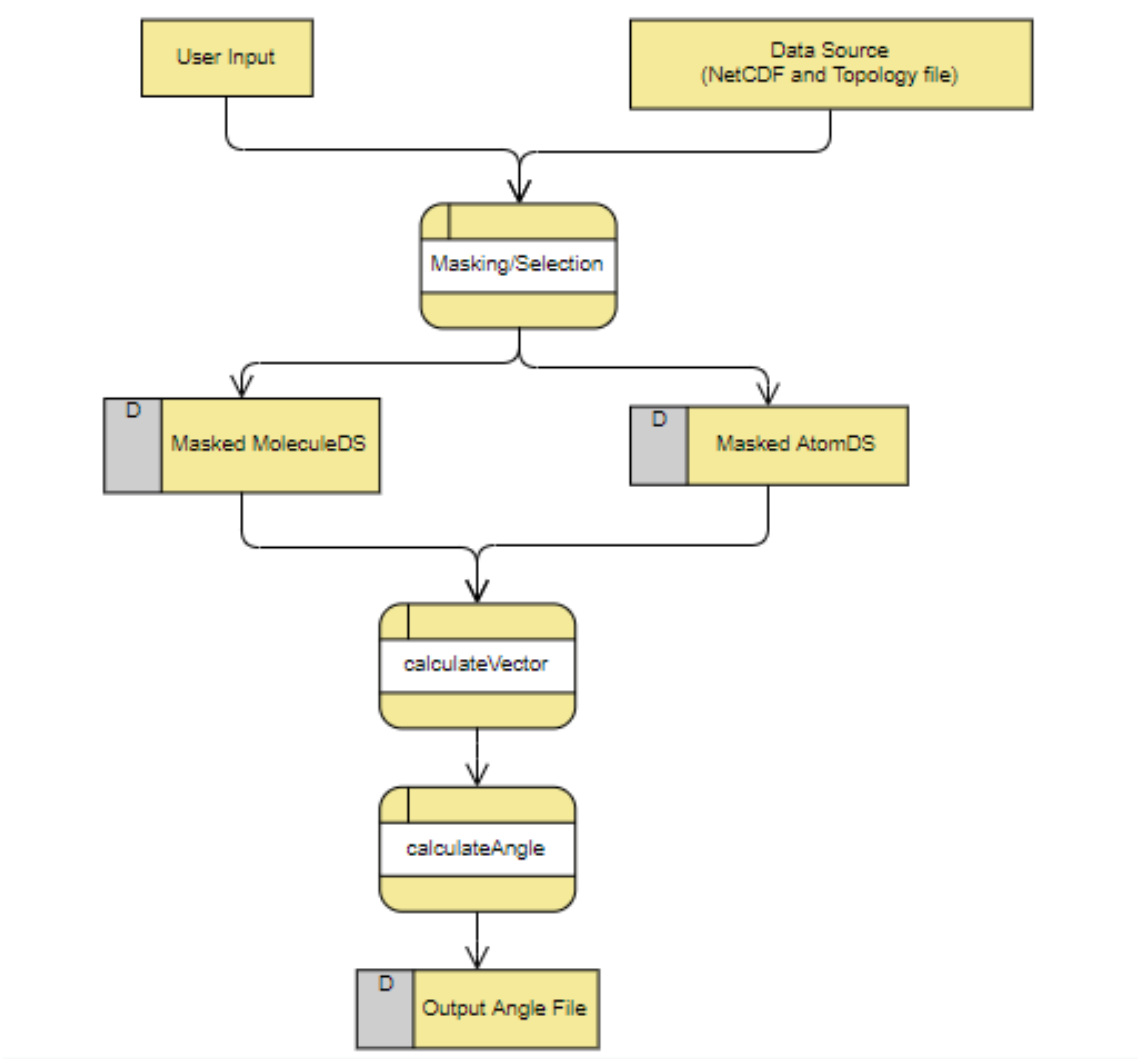


Figure 13: DFD for Angle Calculation

4.9.3 API Doc:

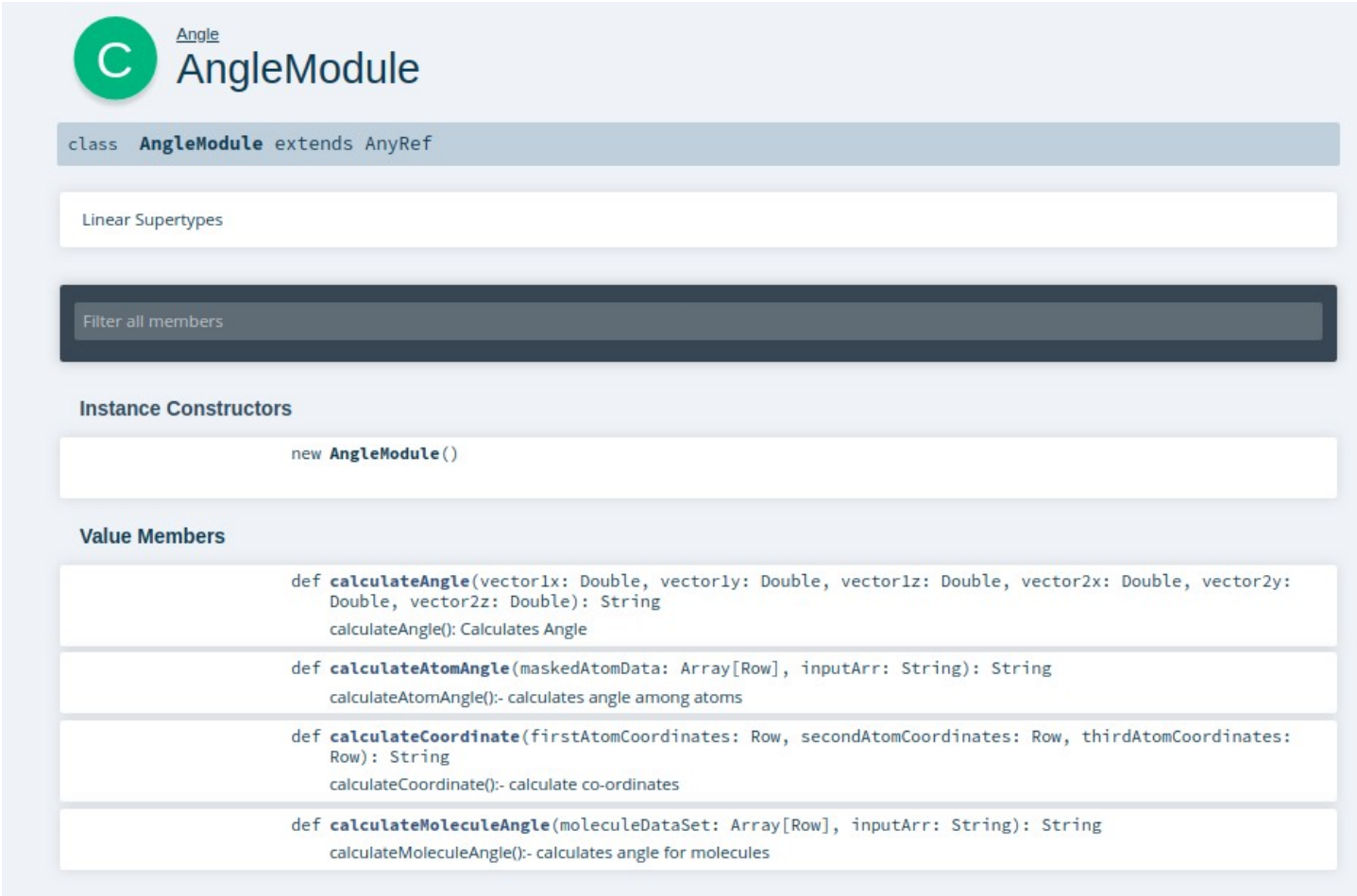


Figure 14: API for Angles

4.10 Dihedrals :

Dihedral Angle is basically calculating angle between 2 planes formed using 4 atoms, 2 of the atoms act as the backbone of the structure and the angle formed between the plane formed by the backbone and first atom, backbone and fourth atom is stated as the Dihedral Angle.

4.10.1 Algorithm:

1. Extract the Masked data as per the user input for the required peptide atoms
2. Fetch the coordinates from the Dataframe with the help of SQL query for the desired atoms in separate arrays
3. Calculate equation for two separate planes to calculate the dihedral angle between them i.e. phi, psi, omega
4. Export the output into different files as per the angle type.

4.10.2 DFD:

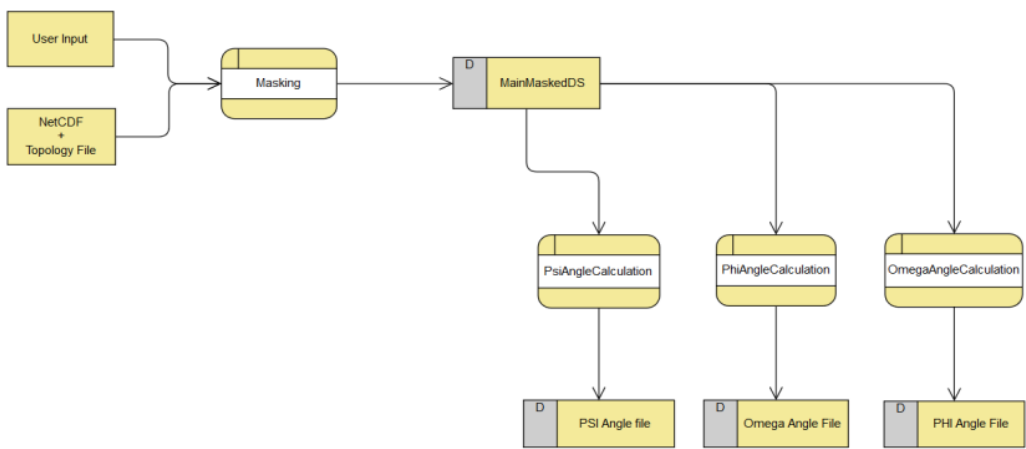


Figure 15: DFD for Dihedrals

4.10.3 API Doc:

C

DihedralAngle

abstract class **DihedralAngle** extends AnyRef

Dihedral Angle is basically calculating angle between 2 planes formed using 4 atoms, 2 of the atoms act as the backbone of the structure and the angle formed between the planes formed by the backbone and first atom, backbone and fourth atom is stated as the Dihedral Angle.

Linear Supertypes

Filter all members

Instance Constructors

new **DihedralAngle**()

Abstract Value Members

abstract def **Angle**(a0\_x: Float, a0\_y: Float, a0\_z: Float, a2\_x: Float, a2\_y: Float, a2\_z: Float, a3\_x: Float, a3\_y: Float, a3\_z: Float): Float  
Angle() to determine the angle between two planes

abstract def **CosFromDotProduct**(aDP: Float, aLine1Len: Float, aLine2Len: Float): Float  
CosFromDotProduct() will be called to calculate the angle PHI

abstract def **CrossProduct**(v1\_x: Float, v1\_y: Float, v1\_z: Float, v2\_x: Float, v2\_y: Float, v2\_z: Float): (Float, Float, Float)  
CrossProduct() crossProduct between two vectors

abstract def **DihedralAngle**(vA\_x: Float, vA\_y: Float, vA\_z: Float, vB\_x: Float, vB\_y: Float, vB\_z: Float, vC\_x: Float, vC\_y: Float, vC\_z: Float, vD\_x: Float, vD\_y: Float, vD\_z: Float): Float  
DihedralAngle() to extend the range of the cos function from {-pi to pi}

Figure 16: API for Dihedral Angles

19

Abstract Value Members	
abstract def	<b>Angle</b> (a0_x: Float, a0_y: Float, a0_z: Float, a2_x: Float, a2_y: Float, a2_z: Float, a3_x: Float, a3_y: Float, a3_z: Float): Float Angle() to determine the angle between two planes
abstract def	<b>CosFromDotProduct</b> (aDP: Float, aLine1Len: Float, aLine2Len: Float): Float CosFromDotProduct() will be called to calculate the angle PHI
abstract def	<b>CrossProduct</b> (v1_x: Float, v1_y: Float, v1_z: Float, v2_x: Float, v2_y: Float, v2_z: Float): (Float, Float, Float) CrossProduct() crossProduct between two vectors
abstract def	<b>DihedralAngle</b> (vA_x: Float, vA_y: Float, vA_z: Float, vB_x: Float, vB_y: Float, vB_z: Float, vC_x: Float, vC_y: Float, vC_z: Float, vD_x: Float, vD_y: Float, vD_z: Float): Float DihedralAngle() to extend the range of the cos function from {-pi to pi}
abstract def	<b>Distance3d</b> (x1: Float, y1: Float, z1: Float, x2: Float, y2: Float, z2: Float): Float Distance3d() distance between two coordinates
abstract def	<b>DotProduct3d</b> (v1_x: Float, v1_y: Float, v1_z: Float, v2_x: Float, v2_y: Float, v2_z: Float): Float DotProduct3d() dot product between two vectors
abstract def	<b>RadToDeg</b> (Radians: Float): Float RadToDeg() for simple conversion purpose
abstract def	<b>VectorSubtract</b> (v1_x: Float, v1_y: Float, v1_z: Float, v2_x: Float, v2_y: Float, v2_z: Float): (Float, Float, Float) VectorSubtract() subtraction between two vectors
abstract def	<b>getMaskedDS</b> (path: String): (Array[Row], Int) getMaskedDS() will have the masked data set
abstract def	<b>omegaAngleCalculation</b> (mainMaskedDS: Array[Row], count: Int, path: String): Unit omegaAngleCalculation() will be called to calculate the angle Omega
abstract def	<b>phiAngleCalculation</b> (mainMaskedDS: Array[Row], count: Int, path: String): Unit psiAngleCalculation() will be called to calculate the angle PSI
abstract def	<b>psiAngleCalculation</b> (mainMaskedDS: Array[Row], count: Int, path: String): Unit phiAngleCalculation() will be called to calculate the angle PHI

Figure 17: API for Dihedral Angles

4.11 Ankush :

To find the networks in which the solvent molecules(water) and polar solute atoms(O, N) have interactions. Interaction is said to be existing when the distance between the solvent molecules and the solute molecules is less than the cutoff distance, given by the user and there should exist at least two solute atoms interacting with the same solvent. There exists many combinations of networks from a network with at least two solute molecules. Moreover, networks with the same number of solute and different solvents molecules are considered as one network.

Steps to be followed :

- 1. Classification of the atoms in two sections i.e solute and solvent.
- 2. Strip everything but the Polar atoms (O, N) from the solute part.
- 3. Analysis of the cutoff distance between the two sets.
- 4. Working and efficient algorithm to return the required factors.

4.11.1 Occurrences :

To find whether the given network occurs in the given percentage of frames

4.11.2 Mamimum Residence Time:

To find the maximum time for which a network exists.

4.11.3 DFD:

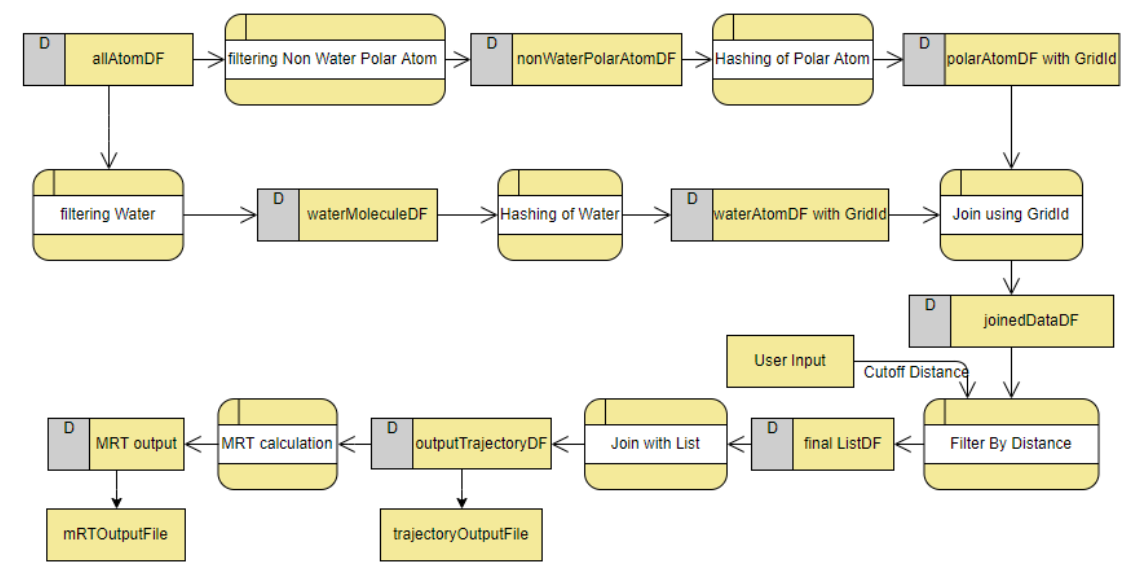


Figure 18: DFD for Ankush

4.11.4 API Doc:

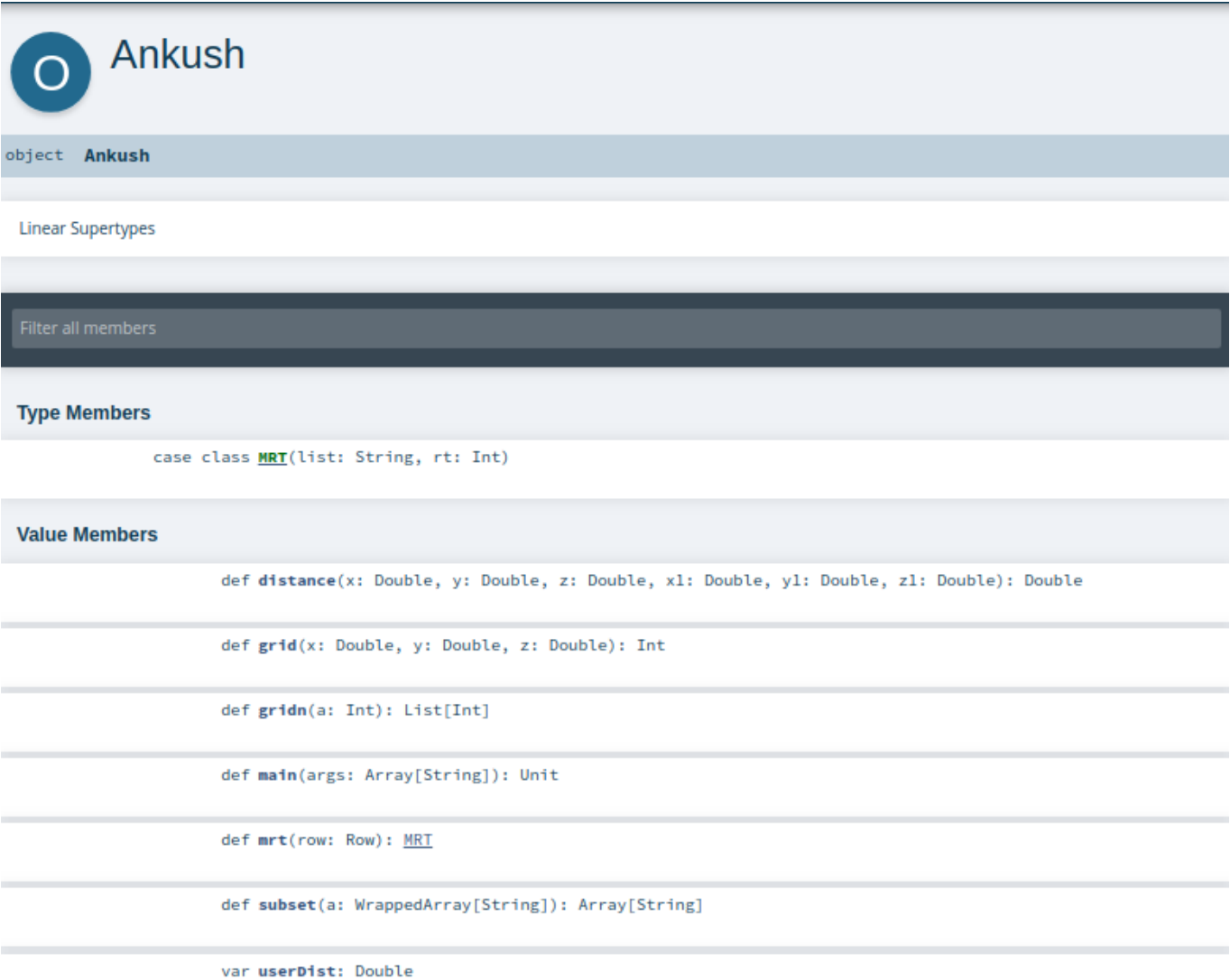


Figure 19: API for Ankush

4.12 Hydrogen Bonds :

A hydrogen bond is defined as being between an acceptor atom A, a donor hydrogen atom H, and a donor atom D. If the A to D distance is less than the distance cut off and the A-H-D angle is greater than the angle cut off, a hydrogen bond is considered formed.

**Potential hydrogen bond donor and acceptor are searched as follows :**

1. We mask donor $\langle$ dmask  $\rangle$ and acceptor  $\langle$ amask  $\rangle$ molecules separately.
2. We take on donor molecule and calculate distance between donor molecule and acceptor molecule using distance module and sort them in increasing order.
3. We calculate angle between donor and acceptor molecules and sort them in decreasing order.
4. We mask potential donor and acceptor using the cutoff criteria .

The number of hydrogen bonds present at each frame will be determined and written to the file specified by out.

4.12.1 DFD:

Bond Length[Å] between hydrogen and acceptor	1.5 to 2.5
Bond Length[Å] between donor and acceptor	2 to 3.5
Bond angle	120

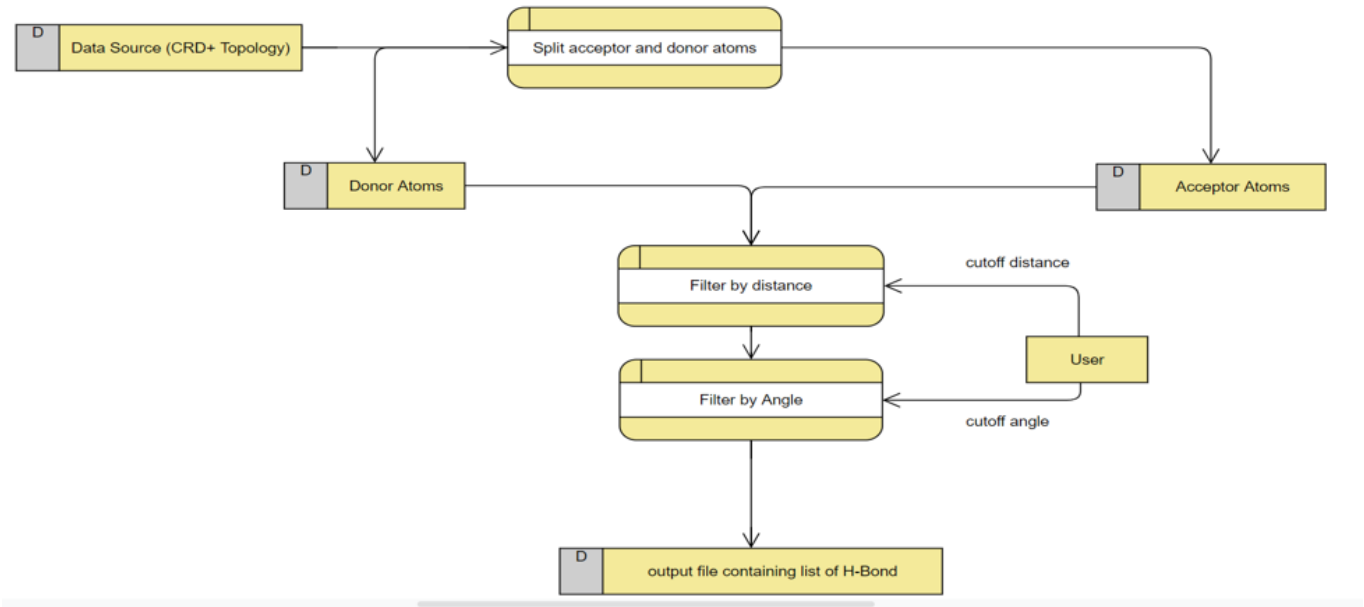


Figure 20: DFD for Hydrogen Bond

4.12.2 API Doc:



## HydrogenBond

`abstract class HydrogenBond extends AnyRef`

hydrogen bond is a type of attractive (dipole-dipole) interaction between an electronegative atom and a hydrogen atom bonded to another electronegative atom.

order to determine the hydrogen bond we'll be using a simple geometric criteria , i.e the distance between the molecules must be less than the cutoff distance and angle between the molecules mustn't be less than the cutoff distance .

Linear Supertypes

Filter all members

### Instance Constructors

`new HydrogenBond()`

### Abstract Value Members

```
abstract def filterAcceptorsbyAngle(maskedDonar: Dataset[String], filtersAcceptorsByDistance: Dataset[String]): Dataset[String]
```

**maskedDonar**are which hydrogen atoms are bounded to donar atoms.

**filtersAcceptorsByDistance**contains the data of donars which satisfies the cutoff distances.

**returns** the dataset which satisfies the angle

```
abstract def filterAcceptorsbyDistance(maskeddonar: Dataset[String], maskedacceptor: Dataset[String], cutoffDistance: Double): Dataset[String]
```

**maskeddonar**are which hydrogen atoms are bounded to donar atoms.

**maskedacceptor**are which can make a covalent bond with donar atoms.

**cutoffDistance**specifies the cutoff distances to make a bond between donar and acceptor

```
abstract def getMaskedDS(): Dataset[String]
```

`getMaskedDS()` will be called to get the masked atomsDS from masking module

**returns** Dat1aSet contains the masked molecular information

```
abstract def makeOutputfile(hydrogenBond: Dataset[String]): Unit
```

**hydrogenBond**dataset contains the list of molecules with hydrogenbonds

```
abstract def splitDonarAcceptor(maskeddata: Dataset[String]): (Dataset[String], Dataset[String])
```

**maskeddata**masked dataset contains molecular data which contains hydrogen bonds

**returns** a tuple of datasets which contains hydrogen bonds.

Figure 21: API for Hydrogen Bonds

## 5 Technologies Used:

### 5.1 Spark:

Apache Spark is open source, wide range data processing engine, that provides high-level API for Java, Scala, Python, and R to write code and it also provide multiple high-level tools for structured data processing, machine learning, graph processing and streaming such as spark SQL, MLlib, GraphX, Stream Processing. Key feature of spark is faster due to in memory computation and parallel processing that make spark more faster. Spark provide support for reading data from various files formats due to use of MapReduce. Spark works on RDD that provide support for in memory computation.

### 5.2 SciSpark:

SciSpark is implemented in a Java and Scala Spark environment. Although Spark also offers a Python environment and we recognize that most scientists are comfortable using Python as a programming language, this Spark environment was chosen to avoid the known latency issues related to the communication overhead involved with copying data from the worker JVMs to Python daemon processes in the PySpark environment. Furthermore, we wish to maximize on in-memory computation but in the PySpark environment the driver JVM writes results to local disk that are then read by the Python process.

Please see our latest paper "Palamuttam, Rahul, Renato Marroquín Mogrovejo, Chris Mattmann, Brian Wilson, Kim Whitehall, Rishi Verma, Lewis McGibbney, and Paul Ramirez. "SciSpark: Applying In-memory Distributed Computing to Weather Event Detection and Tracking." for more details. This document require the Adobe Reader. Download here if you do not have this browser plug-in installed.

### **PARTITION, EXTRACT, TRANSFORM, LOAD**

Scientific array-based data from the network Common Data Form (netCDF) and Hierarchical Data Format (HDF) files on local disks or from remote sources via protocols like OPeNDAP are loaded into SciSpark using a Partition, Extract, Transform and Load (PETaL) process. More specifically, the PETaL layer first partitions the files by time and/or by space (the latter to be added) then distributes the extraction of the data, transforms it into a data type usable in SciSpark, and loads it into the SciSpark engine to the compute nodes.

### **SRDDS**

In the same way Spark exploits Resilient Distributed Datasets (RDDs), SciSpark contributes and exploits the Scientific RDD (sRDD) that corresponds to a multi-dimensional array representing a scientific measurement (grid) subset by time, or by space. The RDD notion directly enables the reuse of array data across multi-stage operations and it ensures data can be replicated, distributed and easily reconstructed in different storage tiers, e.g., memory for fast interactivity, SSDs for near real time, and spinning disk for later operations.

SciSpark defines the Scientific Resilient Distributed Dataset (sRDD), a distributed-computing array structure that supports multidimensional data and processing of scientific algorithms in the MapReduce paradigm. This is currently achieved through a self-documented array class called the sciTensor that defines the data in the multi-dimensional format that scientists are accustomed to. SciSpark currently provides methods to create sRDDs that (1) loads data from network Common Data Form (netCDF) and Hierarchical Data Format (HDF) files into the Hadoop Distributed File System (HDFS); (2) preserve the logical representation of structured and dimensional data in ; and (3) create a partition function that divides the multidimensional array by time (to be expanded to space as well). sRDDs are cached (in-memory) in the SciSpark engine support data reuse between multi-staged analytics.

5.3 Scala:

Scala is a type-safe, modern multi-paradigm programming language that incorporates both object-oriented and functional programming into an extremely concise and logical manner. Scala has features like currying, type inference, immutability, lazy evaluation, pattern matching, higher order functions, polymorphism, abstraction, inheritance etc. It is better suited for distributed data and also supports parallel data processing. It offers number of native libraries for parallelism and also have well designed libraries for scientific computing and linear algebra.

6 Design :

6.1 Software Architecture: Incremental process model

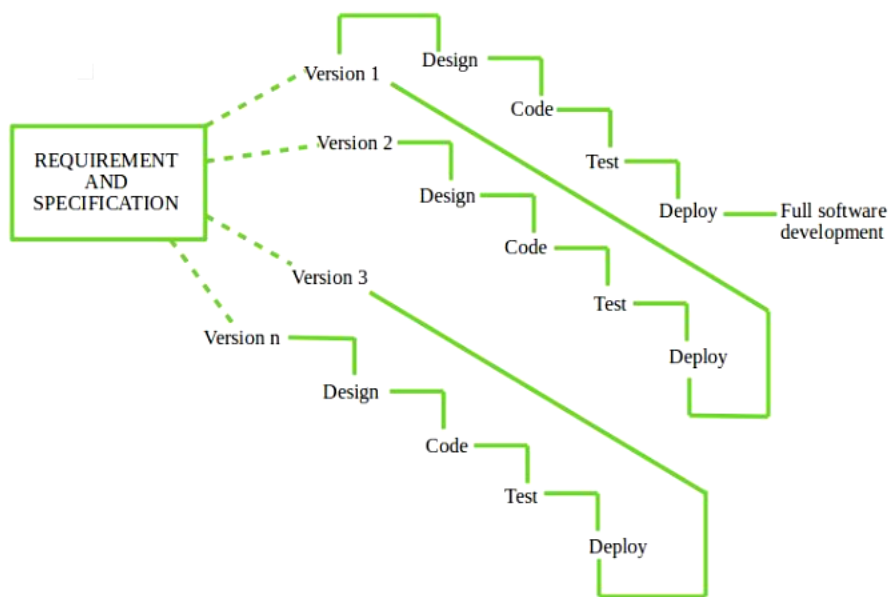


Figure 22: Software architecture model

Requirements of software are first broken down into several modules that can be incrementally constructed and delivered. At any time, the plan is made just for the next increment and not for any kind of long term plans. Therefore, it is easier to modify the version as per the need of the customer. Development Team first undertakes to develop core features (these do not need services from other features) of the system.

Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in successive versions. Each incremental version is usually developed using an iterative waterfall model of development.

As each successive version of the software is constructed and delivered, now the feedback of the customer is to be taken and these were then incorporated in the next version. Each version of the software have more additional features over the previous ones.

How we are implementing it:-

Till now we have used 'Staged Delivery Model of Incremental Process Model'. In 'Staged Delivery Model' we focus on developing one task of project at a time. In this model our initial tasks were to generate PDB file and perform Masking. For further development we are going to adapt Parallel Development Model in which we will work parallelly on several modules.

6.2 DFD Level 0:

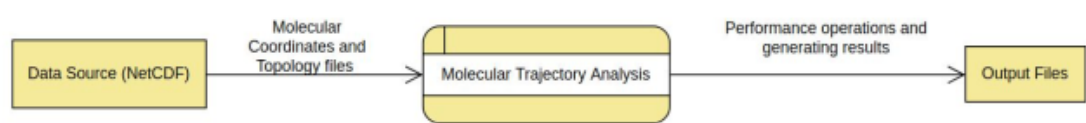


Figure 23: DFD Level 0

6.3 DFD Level 1:

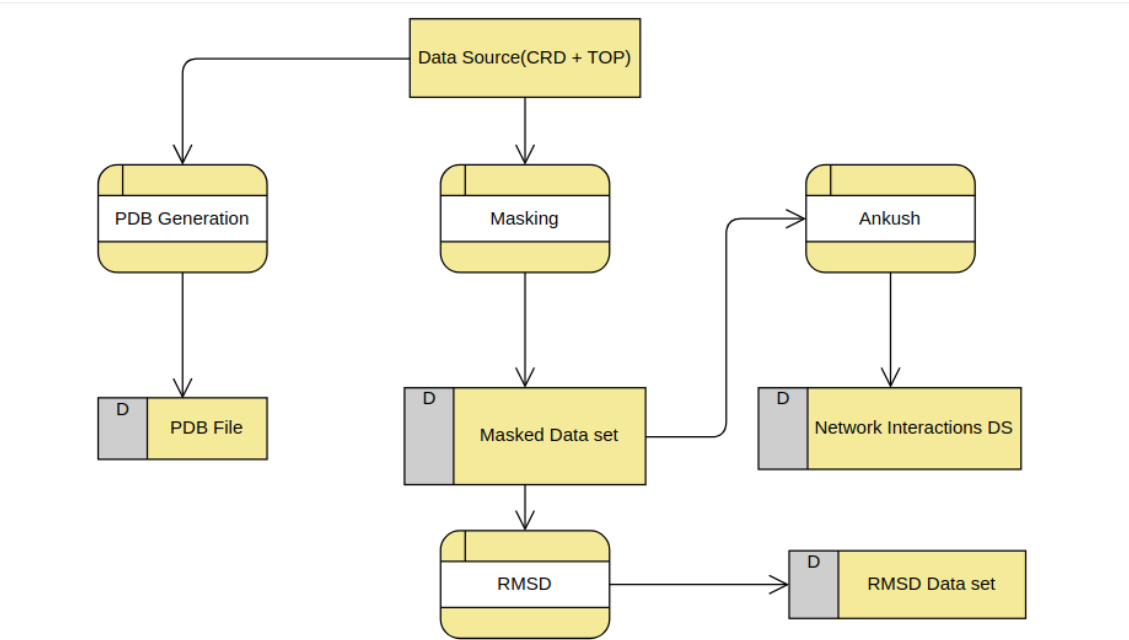


Figure 24: DFD Level 1

## 6.4 DFD Level 2:

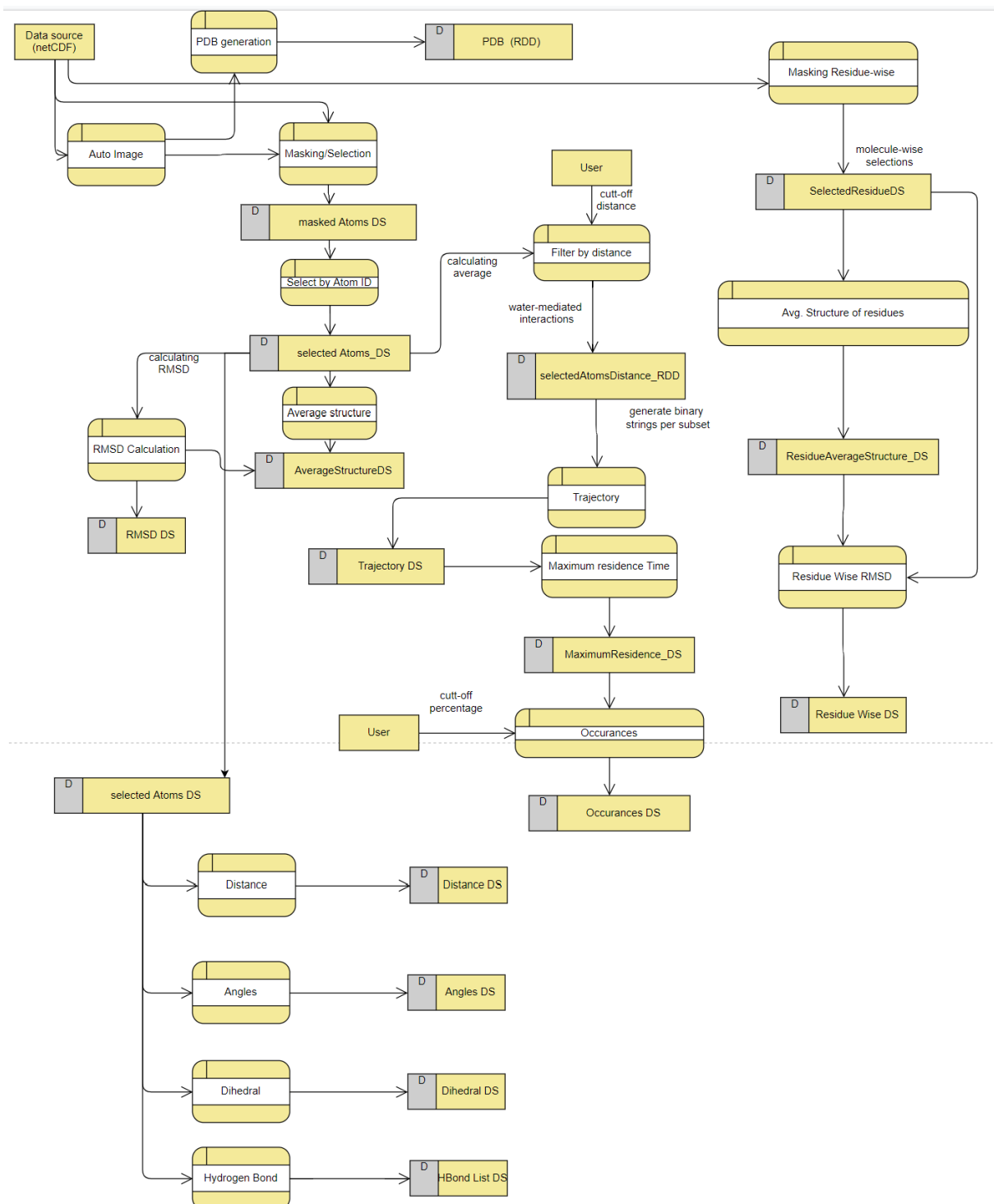


Figure 25: DFD Level 2

6.5 Class Diagram

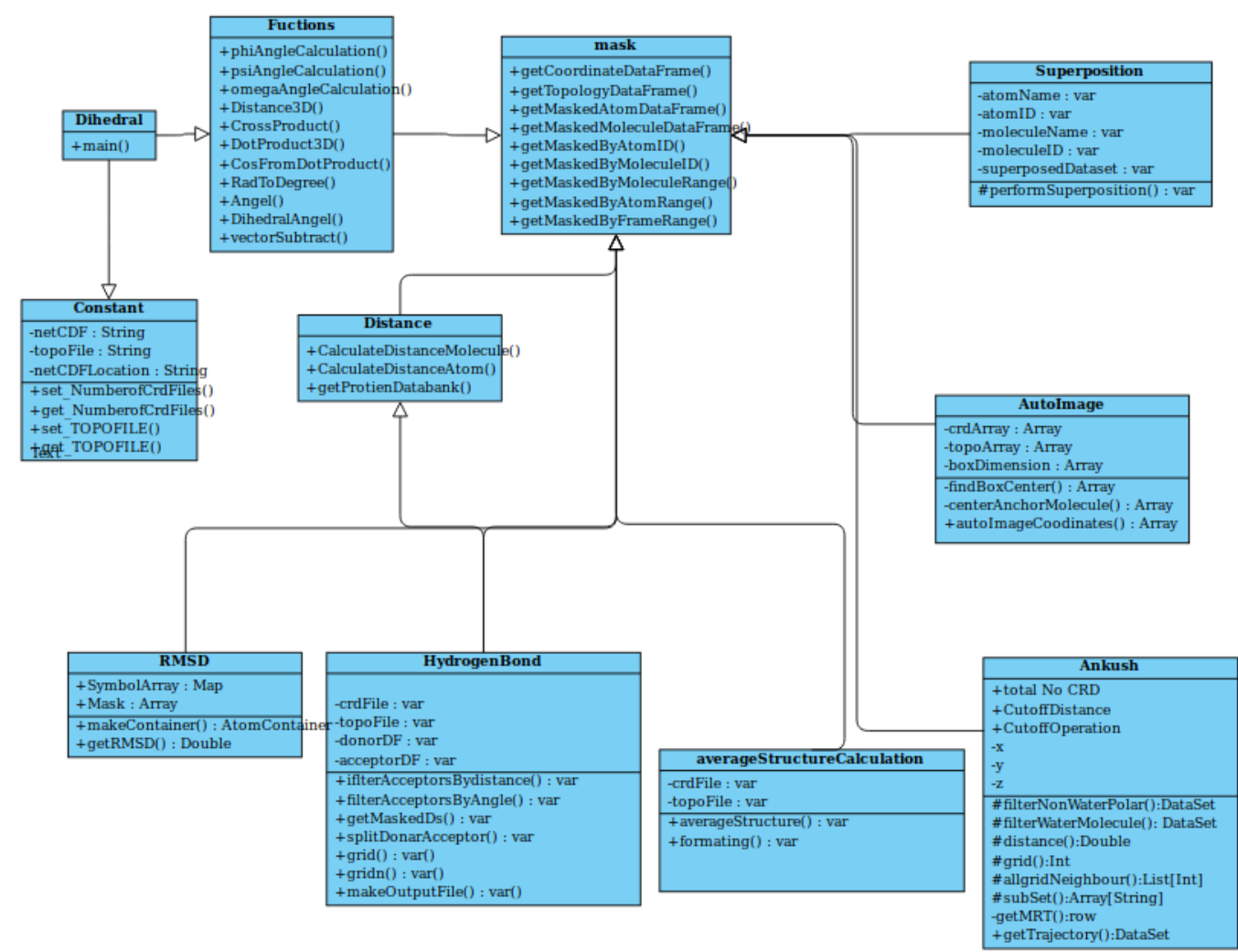


Figure 26: Class diagram

## 7 Dependencies

- NetCDF java library for reading NetCDF files.
- SciSpark for reading NetCDF files into Spark.
- CDK library for RMSD calculations.

## 8 Group Contribution:

### 8.1 Work Distribution :

Modules	Members	Enrolment ID	Team Name
Design	Rohit Takalkar Ram Prasad Saif Khan	MIT2019047 MIT2019115 MIT2019120	
Reading Input	Ajay Garkal Nikhil Ram Prasad	MIT2019035 MIT2019095 MIT2019115	
PDB Generation	Jafar Sarif Shriyas Ingale	MIT2019025 MIT2019032	Enforcers
Auto Image	Aminul Islam Nikhil Brijendra Kaithwas Ram Prasad	MIT2019038 MIT2019095 MIT2019108 MIT2019115	Bit Monks
Masking	Ajay Garkal Diwakar Mishra Purnendu Shekhar	MIT2019035 MIT2019061 MIT2019122	The Heapsters
Average Structure	Praveen Ram Pavan Dharam Raj	MIT2019038 MIT2019099 MIT2019116	Code Knights
Angles	Afzal Ansari Abhishek Gupta Pronika Gautam Ramji Jaiswal	MIT2019072 MIT2019076 MIT2019077 MIT2019106	Black Coders
Distances	Ashutosh Singh Harish Rajora Sumant Srivastav Rohit Takalkar	MIT2019016 MIT2019018 MIT2019046 MIT2019047	Dirty_Bits
Dihedrals	Siddhant Bhisikar Sachin Gupta Mukul Madaan Harshita Sonkar	MIT2019074 MIT2019075 MIT2019113 MIT2019118	The Outliers
RMSD	Abhinav Lahariya Abhishek Kumar Dinesh Pal Ajit Kumar Divay Sharma Thakkar Shrenik Abhishek Barman	MIT2019039 MIT2019040 MIT2019107 MIT2019109 MIT2019041 MIT2019044 MIT2019048	Runtime_Error  OnePunch
Hydrogen Bond	Saurabh Singh Rajeev Singh Abhishek Yadav Praveen Saif Khan	MIT2019027 MIT2019028 MIT2019033 MIT2019038 MIT2019120	BR0grammers
Ankush	Abhishek Gupta Ramji Jaiswal Mukul Madaan Saif Khan	MIT2019076 MIT2019106 MIT2019113 MIT2019120	
Integration	Jafar Sarif Shriyas Ingale Ajay Garkal	MIT2019025 MIT2019032 MIT2019035	
Testing	Harish Rajora Afzal Ansari Sachin Gupta	MIT2019018 MIT2019072 MIT2019074	
	Pronika Gautam Harshita Sonkar	MIT2019077 MIT2019118	
Documentation	Abhishekh Barman Dinesh Pal Purnendu Sekhar Saha	MIT2019048 MIT2019107 MIT2019122	

Figure 27: Work Distribution

9 Performance Metrics:

Modules	1k Frames	10k Frames	100k Frames
Reading	-----	6.0	1.0
Angle	15.0	12.2	3.2
Auto Imaging	9.0	9.2	-----
Average PDB	11.0	6.5	-----
RMSD	245.0	150.0	139.8
PDB Generation	1798.0	1620.0	-----
Masking	40.0	32.0	-----
Dihedral	44.3	29.0	-----
Distance	10.0	13.5	2.9
Hydrogen Bond	210	-----	-----
Ankush	200/600 frames	-----	-----

Figure 28: Performance Metrics

Resources in YARN cluster	Allocation
No. of Executors	35
No. of Executor cores	6
Driver Memory	200 GB
Executor Memory	45 GB

Figure 29: Resources used



## References

- [1] Cay S. Horstmann, *Scala for the Impatient*, Pearson Education.
- [2] O'Reilly, Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis*, Pearson Education
- [3] Mike Frampton, Andrew Szymanski, *Mastering Apache Spark*, PACKT Publishing
- [4] Websites referred  
<http://www.visual-paradigm.com>
- [5] Chemistry Development Kit  
<https://cdk.github.io/>
- [6] SciSpark  
<https://github.com/SciSpark/SciSpark>