Chapter 1

Introduction to the Probe Particle Model (PPM)

1.1 Probe Particle Model

With the achievement of high resolution images using CO functionalised tips that probe Pauli repulsive forces in the near domain of the sample, ascertaining the origin of artifacts initially purported to be intermolecular interactions in images was elusive. Within this context, Hapala et al [1] developed a simplistic model based on molecular mechanics to provide further insight into the mechanism of high resolution imaging. This simplification was only possible due to the fact that the origin of the high resolutions obtained with CO functionalised tips has been determined to be mainly due to the flexibility of the probe particle attached to the tip. As the sample is approached, the probe particle (Oxygen atom) experiences lateral shifts due to the Pauli repulsive forces in the near contact regime of the sample. This relaxation with respect to the sample, however, does not affect the electronic configuration of the tip and sample. For that matter, modelling the tip-sample interactions based on lennard Jones potentials and electrostatics would , to a good approximation, simulate all the tip-sample interactions provided that relaxations of the probe particle with respect to the tip is incorporated.

CO functionalised tips are modelled within the probe particle model to consist of two atoms, the probe particle (PP) and tip particle (TP). Accordingly, the probe particle is the Oxygen atom and the tip particle which functions as a tip base is the Carbon atom. As required for high resolution AFM experiments, the functionalised tip is modelled to be chemically inert to allow the sample to be probed in the near contact regime where Pauli repulsive forces are dominant. More importantly, this tip is allowed to relax due to an effective potential due to the tip particle and the surface. All of these considerations for the PPM, makes it possible to simulate NC-AFM images of various surfaces or molecules.

1.1.1 The Lennard Jones Force Field

The lennard Jones force field implemented in this model is calculated within the PPM code as the sum of the pairwise Lennard Jones forces between the PP and the atoms of the sample surface. This is expressed as,

$$\vec{F}_{\alpha\beta} = \vec{R}\left(\frac{12B_{\alpha\beta}}{r^{14}} - \frac{6A_{\alpha\beta}}{r^{8}}\right) \tag{1.1}$$

where the r^{14} and r^{8} terms approximate the Van der Waal attractive forces and the Pauli repulsive forces respectively. Based on empirical atomic parameters the

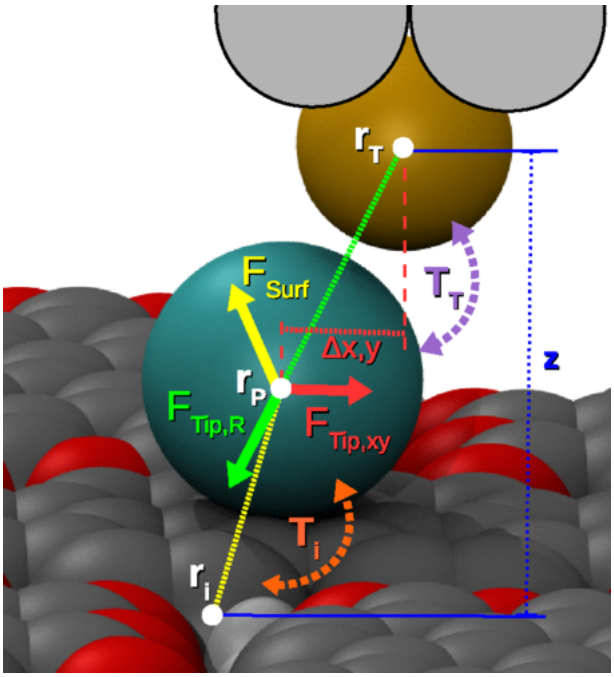

FIGURE 1.1: A schematic view of the simplistic Probe Particle Model based on molecular mechanics. The sample surface (red and grey spheres forming PTCDA molecular assemblies) are being imaged using a Probe Particle (torquoise sphere) attached to a Tip Particle (gold sphere). [1]

Lennard Jones parameters $A_{\alpha\beta}$ and $B_{\alpha\beta}$, determine the equilibrium bond distance and energy.

1.1.2 The Electrostatic Force Field

AFM experiments with CO functionalised tips have been found to produce distorted images due to deflections of the probe particle as it approaches the potential energy basin of a sample surface. While these deflections have been ascertained to be due to the Pauli repulsive forces, it has been determined that electrostatic forces also induce further distortions to the image caused by additional deflections of the CO tip. Within this context, the inclusion of electrostatic force fields to the PPM improves the accuracy with which NC-AFM experiments are simulated. Additionally, this would even allow the PPM to simulate NC-AFM images of charged systems. The charge distribution on the probe particle is modelled as a Gaussian distribution given by

$$Q_{PP} = \int \rho_{PP}(\vec{r}) d\vec{r}, \tag{1.2}$$

with ρ_{PP} and Q_{PP} being the gaussian probability distribution function and the total charge on the PP respectively. Accordingly, the total charge Q_{PP} is obtained by integrating over the entire 3D space defined by the coordinate r.

More specifically, the electrostatic force experienced by the PP is calculated as a multiplicative convolution of ρ_{PP} and the surface potential $V_{surf}(r)$. Hence the electrostatic force due to the sample surface F_{el} is given by,

$$F_{el}(R) = \frac{\delta}{\delta R} \int \rho_{PP} V_{surf}(\vec{r} - \vec{R}) d\vec{r} \tag{1.3}$$

Modelling the interaction/bond between the TP and PP requires the use of two potentials. The TP-PP bond F_{bond}, is characterised by a lateral harmonic potential and a radial Lennard Jones potential. Relaxations of the PP are made to occur in the probe particle model by allowing the PP to minimise the resultant force by moving freely within the present force fields. Thus, F_{tot} is such that,

$$F_{tot} = F_{el} + F_{LJ} + F_{bond} = 0 \tag{1.4}$$

Comparative studies of PPM simulations and DFT calculations have provided insight into the mechanism responsible for high resolution images using CO functionalised tips. It has been determined that the tip experiences repulsive forces within the near contact domain of the sample surface responsible for tip relaxations. Specifically, these repulsive forces have been found to be mainly due to the overlap of electronic orbitals (Pauli repulsion) and a slight contribution caused by electrostatic forces. As mentioned in Sec 1.1.2, the relaxation of the probe particle into the local minima of the potential energy landscape leads to increased resolutions along the ridges and saddle points. The PPM finds particular relevance in its ability to simulate NC-AFM images while accurately reproducing these features.

1.1.3 3D Force Mapping with Probe Particle Model

As mentioned in Sec. 1.1, the PPM is a simplistic model based on molecular mechanics used to simulate NC-AFM images. Despite the fact that NC-AFM images could be simulated using more direct methods such as Density Functional Theory (DFT), the PPM is extensively preferred within the literature due to its much reduced computational cost. DFT calculations would take tens of thousands of computational hours on a supercomputer - a particular characteristic that would hamper its wide use for simulating NC-AFM images.

The PPM code, on the other hand, simulates NC-AFM images based on molecular mechanics with a superimposed electrostatic potential while utilizing input atomic positions, unit vector lengths and electric potentials precalculated using DFT. This has a lower computational cost within the order of dozens of CPU hours. Fig. 1.2 shows simulated NC-AFM images of a PTCDA molecule on Cu as a function of tip-sample approach.

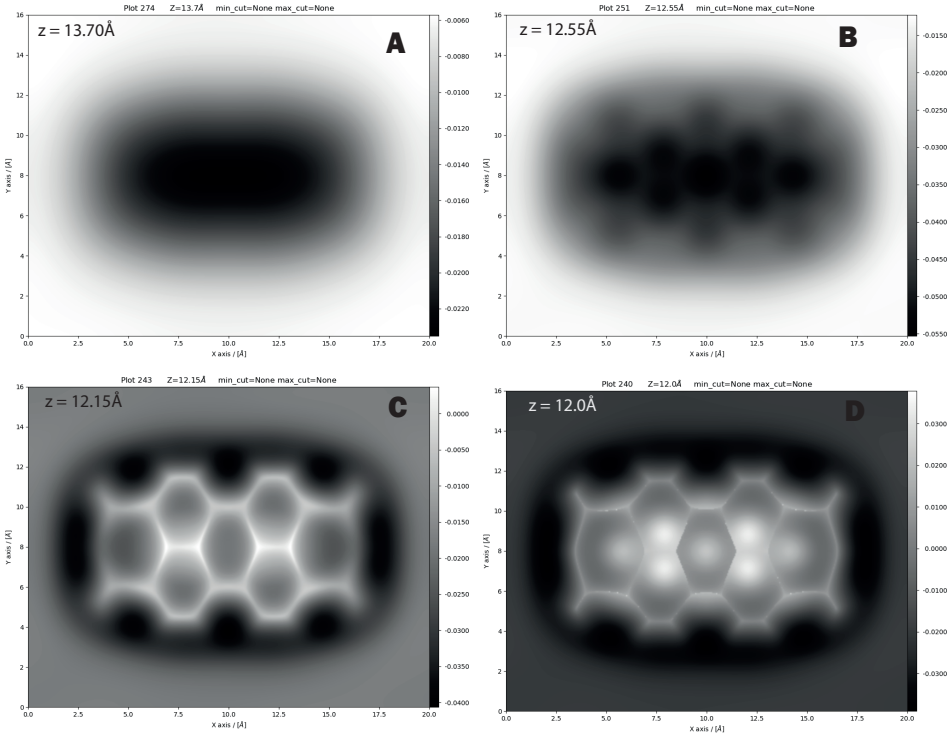

FIGURE 1.2: PPM simulated constant height NC-AFM force images of an isolated PTCDA molecule. **(D)** At far tip sample distances the structure of the molecule is not resolved due to dominant VdW forces. **(C)** As the molecule is further approched, the structure of the molecule becomes apparent as seen in **(B)** and **(A)** where contrast inversion occurs.

Chapter 2

Coordinate Systems in PPM

As detailed in Chapter 1, the PPM code simulates NC-AFM images by performing isolated calculations based on the mechanistic model. Due to the separate nature of the calculations, it is required that five distinct coordinate systems be defined for each calculation. The output of these calculations are stored in a three-dimensional (3D) grid with each voxel (a single data point in a 3D grid) having a corresponding output value. Accordingly, the coordinate systems are needed to present a framework for defining the specific positions of the generated voxels during the PPM calculation. The PPM itself implements three coordinate systems. These are the scanRange , gridN and the frequency shift coordinate systems.

Aside the PPM's coordinate systems, there are two other coordinates systems used in storing the electrostatic force field data as well as the atomic positions of a given sample. Specifically, the Atomic Positions Coordinate System (APCS) provides the specific means by which the positions of atoms are defined. The PPM defines its absolute origin based on this coordinate system. This origin is a very important component of the PPM code as it serves as the reference point from which all other coordinate systems are defined. Additionally, it is worth noting that while the APCS is not quantized, the electrostatic force field calculation utilizes a quantized coordinate system. The Electrostatics Coordinate System (ECS) is a 3D grid with regularly spaced coordinates with a corresponding electrostatic potential associated with each voxel. This data is stored in files known as cube files which serve as one of the primary input files for the PPM simulations. For the proper functioning of the PPM code, it is a basic requirement that the origin of the ECS be defined to be the same as the absolute origin defined in the APCS.

2.1 FFLJ, FFel and the GridN Cordinate System

The PPM calculates the Lennard-Jones (L-J) (based on molecular mechanics) and electrostatic force field (based on the electric potential in the cube file) associated with a sample and saves them both as $FFLJ$ and $FFel$ matrices respectively. But how this data is stored depends specifically on a coordinate system known as the gridN Coordinate System (gNCS). The gNCS specifies the positions of the voxels corresponding to both the calculated L-J force field and the electrostatic force field. This coordinate system is defined by four parameters, namely, gridA, gridB, gridC and gridN. GridA and gridB are orthogonal unit vectors that define the $X-Y$ plane over which scanning takes place (see Fig. 2.1). Accordingly, the gridC parameter is a unit vector orthogonal to gridA and gridB specifying the length of the grid in the Z direction. Thus, it is the gridC parameter that defines the Z-range over which simulations are run. In essence, this parameter controls the tip-sample distance of simulated AFM experiments. In this gNCS, the resolution and the computational

accuracy of the calculation is controlled by the gridN parameter. In particular, this parameter defines the dimensions of each voxel by specifying the number of the divisions in X, Y and Z directions within the grid. This coordinate system i.e. gNCS could be defined by exploiting a pre-set functionality of the PPM when working with input cube files. Specifically, the PPM extracts gridABCN values based on the electrostatics coordinate system defined in the cube file (which is the ECS). As a result, for the correct definition of the gNCS, it is essential that the origin defined in the cube file is the absolute origin.

The calculation of the pairwise L-J potentials between the probe particle and the atoms of the sample brings to bare another important parameter of the PPM. The $nPBC$ parameter controls the number of replications of the unit cell when calculating the pairwise potentials. The unit vectors of this unit cell (which is replicated in X and Y directions) is defined by the same gridA and gridB vectors used in the gridN coordinate system. It has also been determined that this parameter has an influence on the computational accuracy of the calculation. Specifically, larger nPBC values and hence longer simulation times have been found to be directly proportional to more accurate LJ calculations. These calculated L-J potentials are stored as $FFLJ$ matrices in the PPM. Furthermore, the calculated electrostatic forces experienced by the PP are stored as $FFel$ matrices using an approach based on a convolution of the charge density distribution of the PP and the electrostatic potential (supplied in the cube file). The electrostatic force field is sampled at every grid point in the gNCS.

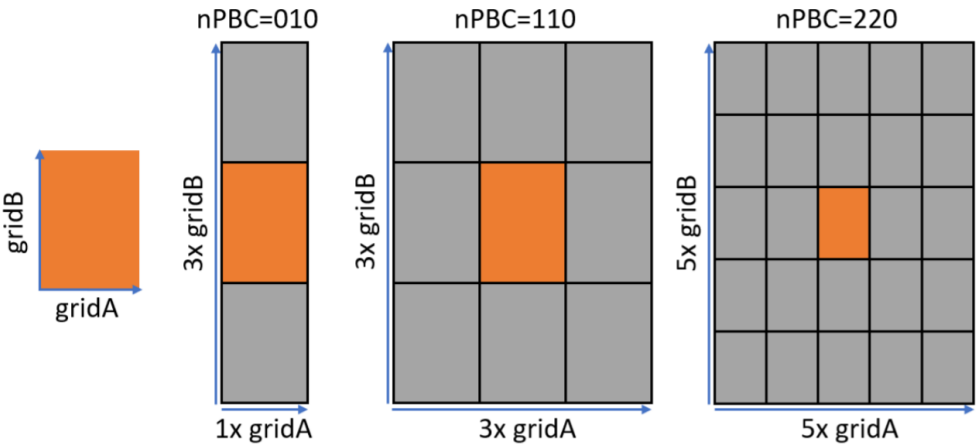

FIGURE 2.1: An illustration of the replication of the unit cell for generating the FFLJ matrices. For a specific unit cell (orange unit cell), FFLJ matrices are generated based on repeating the specified unit cell (containing gNCS grid points) as denoted by the grey rectangles. [2]

2.2 Relaxed Scan and ScanRange Coordinate System

It has been introduced in Sec. 1.1.2 that the probe particle relaxes to its minimum force configuration as it experiences different force contributions within the domain of the sample upon sample approach. Justifiably so, it is required that a separate coordinate system be defined to accurately simulate the relaxation process. The Scan Range Coordinate System (SCRS) is formulated based on a distinct set of parameters. They are the $scanMin[XYZ]$, $scanMax[XYZ]$ and the $scanStep[XYZ]$ parameters. In

particular, the *scanMin* and *scanMax* parameters determine the scanning range of the PPM simulation in the X, Y and Z directions. To control the resolution of the scanning process, the *scanStep* parameter controls the grid point spacing in the X, Y and Z directions. The *scanMin* and *scanMax* parameter are set by simulating within physically realistic regimes of the sample motivated by a wealth of AFM experiments [1]. More importantly, the PPM simulation accounts for the relaxation of the probe particle through a well-defined scheme. With the tip particle held at a fixed position, the probe particle is allowed to move into a configurational state that minimises the resultant force. Specifically, the tip particle is moved towards the sample constrained by the $scanMin[Z]$, $scanMax[Z]$ and *scanStep* parameters, with the probe particle being allowed to relax at each discrete approach. The result of this relaxation is stored in an *OutFz.npy* matrix which is essentially the total vertical force acting on the probe particle at every X, Y coordinate position specified by the SRCS. Accordingly, this output matrix is three-dimensional in shape commensurate with the expected 3D volumetric data acquired in real space 3D force spectroscopy.

2.3 Frequency shift (Δf) and the Frequency Shift Coordinate System

In real AFM experiments, it has been shown that force and energy curves are generated as part of a post-data acquisition process. In contrast, the PPM calculates and plots frequency shift data based on the calculated vertical force (*OutFz.npy*) experienced by the tip. Similar to the *FFel.npy* and *FFLJ.npy* matrices, the PPM stores the calculated frequency shift data into a $df.npy$ matrix whose voxel positions are specified by the frequency shift coordinate system (dfCS). This coordinate system is formulated based on the $scanMin[XYZ]$, $scanMax[XYZ]$ and $scanStep[XYZ]$ and is defined relative to the absolute origin.

2.4 PPM Input Files

The PPM has the functionality to utilize input files in different file formats. The format of the input file depends on the properties being probed for a given sample. For example, in running simulations that do not consider electrostatic interactions, one can use input files in the $input.xyz$ format which holds information on the atomic numbers and atomic positions of the sample being imaged. In this case, the *FFel* matrix would be a three-dimensional array of zeros. Another situation more relevant to our purposes is working with input files that include electrostatics data. These files are used for simulating NC-AFM images that consider electrostatic contributions to the total vertical force *OutFz.npy*. These kind of input files can be stored in th the cube file format ($input.cube$) or xsf file format ($input.xsf$).

Cube files are file formats that originate from the Gaussian Software Package. These files hold information on the atomic positions and volumetric data corresponding to a certain property of a given sample. In this case where the influence of electrostatics on sub-molecular contrast is being mapped, this volumetric data would describe the electrostatic potential of the sample. Cube files can be visualized and manipulated using 'The Open Visualization Tool (OVITO)'. Fig 2.2 shows a cube file for a PTCDA molecule visualized using OVITO showing the electrostatic potential at a specific isolevel. Another approach to incorporating electrostatics into the

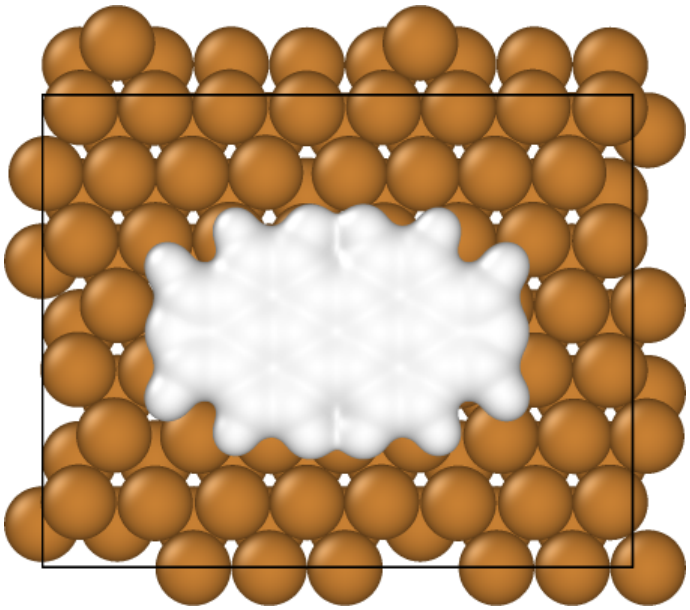

FIGURE 2.2: An image generated from OVITO showing the isosurface (at $isolevel = -0.1$) for a PTCDA molecule on a Cu surface

PPM is by using a pre-calculated Hartree potential of the surface or sample stored in an $input.xsf$ file format.

Chapter 3

Simulation Setup

3.1 Downloading and Installing the Simulation

Currently, the PPM code runs within the python3 environment. While this means one can just install python3 for running the PPM, it is most useful to use 'Anaconda' which gives the flexibility to change environments when needed. For Windows users,

- Download and install the 'Windows Subsystem for Linux (WSL)' found in the Microsoft Windows store.
- Open the WSL terminal and follow the instructions here to install 'Anaconda'.

Once Anaconda is installed, we can setup the python environment consisting of all the necessary modules (numpy, scipy, matplotlib, pip, Pillow) for the PPM simulation by doing the following,

- Within the WSL terminal, do

```
conda create -n PPM3 python=3 numpy scipy matplotlib joblib
    imageio pip pillow
```

Download and install the simulation on the probe particle model Wiki page found here.

3.2 Original PPM code

After downloading and extracting the PPM zip file, a number of directories can be found in this folder. To quick start this tutorial, our focus will be on the '*examples*' directory found in this folder. For each simulation, for instance Graphene, there are three files. These are:

- The *run.sh* file which is a shell scipt that executes the simulation.
- The main input file which holds information on the atomic geomtries of the molecule (in this case Graphene.xyz)
- The *params.ini* file which is used to define parameters for the PPM simulation.
- The *clean.sh* which resets the simulation by clearing the directory.

3.3 Visualizing Input Files

Input files need to be visualized to ensure that basic requirements are met for the correct functioning of the PPM simulation. These basic requirements include:

- Ensuring that the molecule being imaged is at the center of the unit cell
- Visualizing the electrostatic isosurface to check consistency with the molecule
- Checking if the voxel grid unit cell extends to about 10Å above the highest atom of the molecule. This would ensure that there is enough room above the highest atom to observe the evolution of contrast.

These files can be visualized using the open-source 'Open Visualization Tool (OVITO)' software. Let us demonstrate this using an example cube file of a PTCDA molecule in Fig 3.1. To load a cube file into OVITO,

1. Click the folder icon as indicated by the arrow (1) in Fig 3.1.
2. Select the preferred cube file (*PTCDA_example.cube* in this example)
3. Click 'Open' as indicated by the arrow (3)

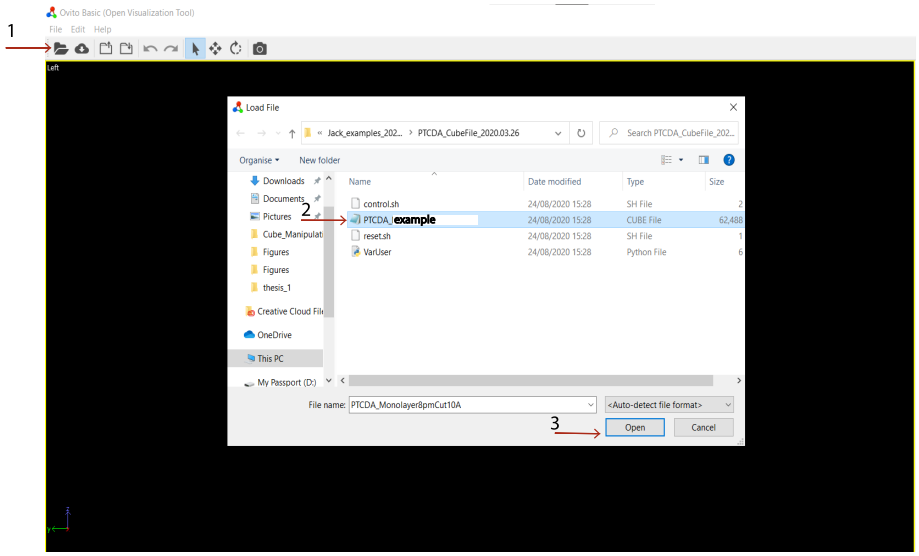

FIGURE 3.1

For the specific example file chosen, the result is shown in Fig. 3.2 The electrostatic isosurface of this molecule can be visualized by doing the following

1. Click the 'Add modification' option as indicated by the red arrow in Fig. 3.2
2. Navigate to 'Create isosurface' and select

The result of this step is shown in Fig. 3.3. Note that the isosurface generated is for a specific isolevel of -0.1Å. The red rectangle in Fig. 3.3 highlights the area where the isosurface could be modified.

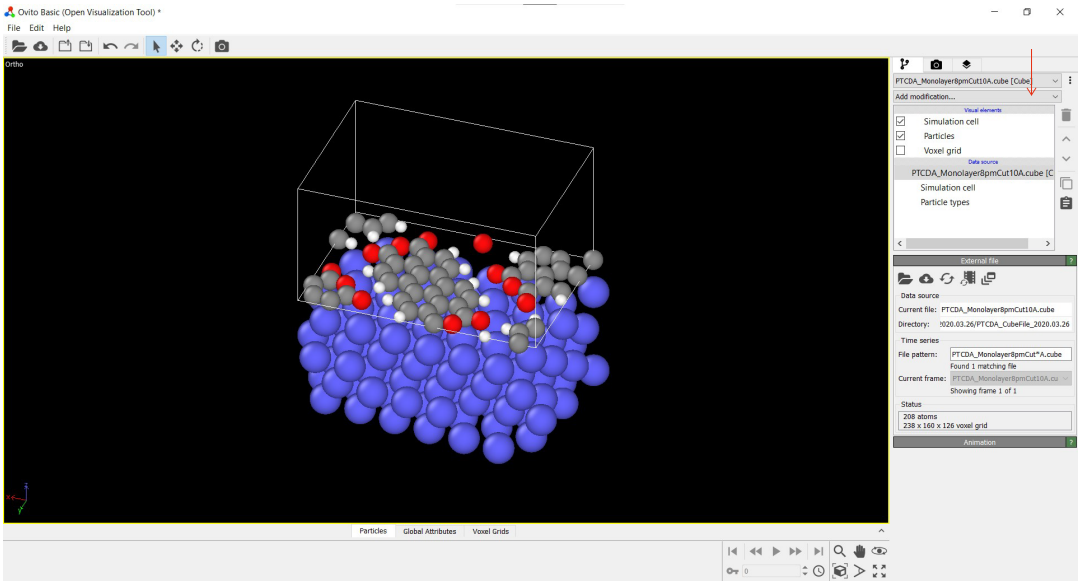

FIGURE 3.2

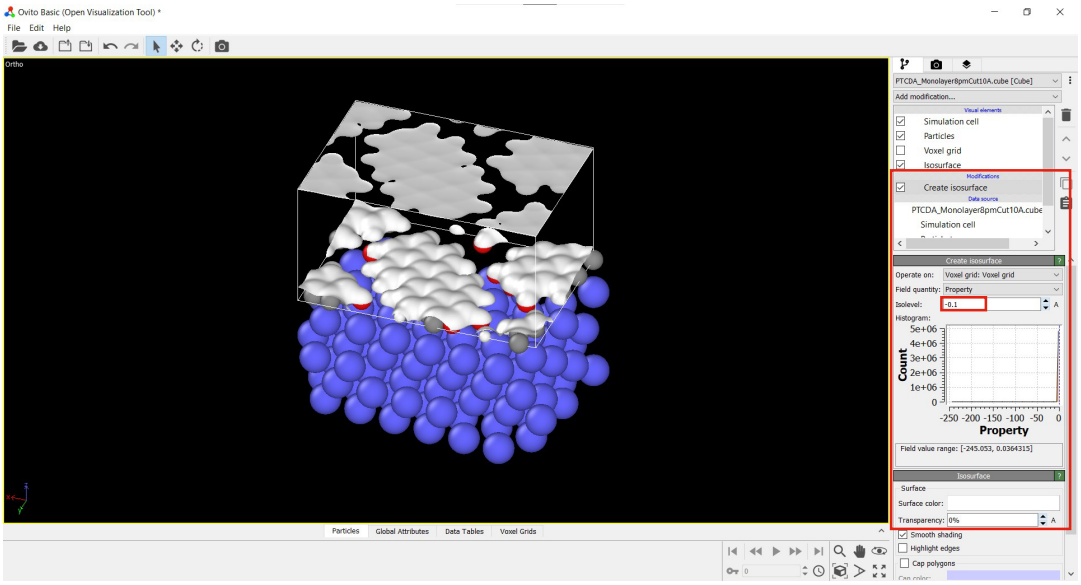

FIGURE 3.3

3.3.1 Effect of changing the Isolevel

Fig. 3.4 shows changes in the isosurface as the isolevel is varied. From C-A, it can be seen that the shape of the isosurface increasingly approaches the shape of the molecule as the the isolevel is increased. At 0.1, the isosurface resolves the structure of the molecule.

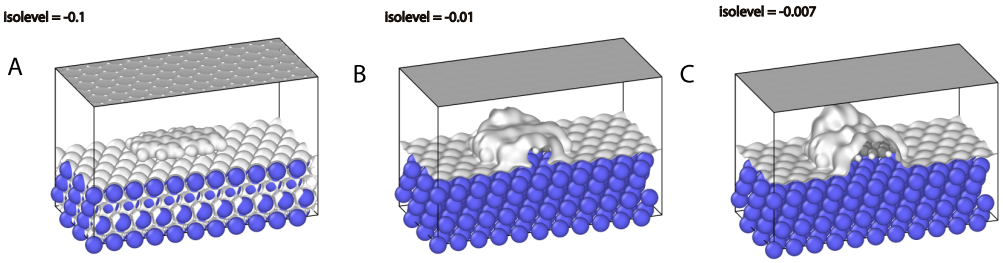

FIGURE 3.4

3.4 Running a PPM Simulation

3.5 Jack's code

While the base/original PPM has already been introduced, it is easier to use code written by Jack Henry to run the PPM simulation. This is preferrable since it gives the PPM additional functionalities such as generating different types of plots of the output file without having to write code for it manually.

3.5.1 The VarUser file

To successfully run a PPM simulation parameters defined within the *VarUser.py* file of Jack's code need to be understood. Starting with the SRCS (i) in Fig. 3.5, the variable *scanMin* specifies the minimum scanning distance of the simulated AFM tip in the X, Y and Z directions. Similarly, the *scanMax* variable determines the maximum scanning distance in the X, Y and Z directions. Additionally, the resolution of this scanning process is controlled by the *scanStep* variable. For example, in the z-direction, the range of scanning in the z-direction is from 22.5Å to 36.00Å with a scanning step of 0.01Å.

```
1   ''' Default PPM variables '''
2   #ScanRange Coordinate System (SRCS). Type=List ([X,Y,Z])
3   scanMin             = [   0.00  ,    0.00  ,   22.5  ]
4   scanMax             = [  20.00  ,   16.00  ,   36.00  ]          <-- (i)
5   scanStep            = [   0.05  ,    0.05  ,    0.01  ]
6   #GridN Coordinate System (gNCS). Type=List ([X,Y,Z])
7   gridN               = [    30,      30,     10]
8   gridA               = [    0.03998  ,     0.00000  ,     0.00000  ]   <-- (ii)
9   gridB               = [    0.00000  ,     0.08209  ,     0.00000  ]
10  gridC               = [    0.00000  ,     0.00000  ,     0.08209  ]
11  #FFLJ Periodic Boundary Conditions. Type=List ([X,Y,Z])
12  nPBC                = [ 0,   0,   0]                    #Recommended [3,3,0]
13  probeType           = 8                                 # atom type of ProbeParticle (to choose L-J
    potential ),e.g. 8 for CO, 24 for Xe
14  #tip       = 'dz2'
15  charge              = 0.05                               # effective charge of probe particle [e]
16  stiffness           = [ 0.20  ,   0.20  ,   20.00  ]     # [N/m] harmonic spring potential (x,y,R)
    components, x,y is bending stiffnes, R particle-tip bond-length stiffnes,
17  r0Probe             = [ 0.0  ,   0.0  ,    4.00  ]      # [AA] equilibirum position of probe
    particle (x,y,R) components, R is bond length, x,y introduce tip asymmetry
18  kCantilever         = 1800.0
19  f0Cantilever        = 23165.0
20  Amplitude           = 1
21  ''' Jack Addition Variables '''
22  ##### Adds zeros to the bottom of cube file electrostatics, relative to the initial
    file. This deals with the data that was not generated by DFT
23  CubeDataPad_Zpad_LJ         = 0         #Default=0
24  CubeDataPad_Zpad_EL         = 0         #Default=0
25  ##### Set gridABCN automatically by reading the .cube file enter True/False or nothing
26  CubeDataExtract             = True          <-- (iii)
27  ##### Set the scanMin, scanMax, and scanStep values to align with the gridN
    coordinate system from the .cube file. Takes into account the FF matrix truncation
28  AutoSet_scanMin             = [ True  ,   True  ,   True  ]
29  AutoSet_scanMax             = [ True  ,   True  ,   True  ]   <-- (iv)
30  AutoSet_scanStep            = [ True  ,   True  ,   True  ]
31  ##### RAM Reductions: Reduce RAM required by deleting some of the FF matrix. Set
    Z_range*(gridC/gridN[Z]) = {Top Of Surface}
32  FF_Truncate                 = CubeDataPad_Zpad_LJ
33  ##### Overwrite:  Overwrite specific FF matrices with zero's
34  Overwrite_FFEL              = [ False  ,   False  ,   False  ]
35  Overwrite_FFLJ              = [ False  ,   False  ,   False  ]
36  ##### SplitStep2: What x range (in angstroms) to carry out each relaxed scan
    (original Step 2) over
```

FIGURE 3.5: The VarUser file

Secondly, the calculuation of the LJ and the electrostatic forces depend on the gNCS (ii) shown in Fig. 3.5. As mentioned in Sec. **??**, the gridN parameter controls the number of divisions in the voxel grid unit cell used for the LJ and electrostatic forces calculation. For the PPM simulation to run properly SRCS and the gNCS have to be consistent with each other. Hence, for this specific example, gridN[X] (number of divisions in the x-direction) is calculated by dividing the scanning range by the scanstep in the x-direction. To obtain the values in the y and z direction, the same is applied. While this is a manual approach, the gNCS is always set automatically when dealing with cube files by exploiting the variable 'CubeDataExtract' (see (iii) in Fig. 3.5). Similarly, the variables shown in (iv) can be used to automatically set the SRCS based on the ECS defined in the cube file.

```
54  ###### Plot 2D images of OutFz.npy matrix in the x, y, or z planes
55  Fz_MakePlots          =  True          <----  (i)
56  Fz_cmap               = 'gray'
57  Fz_CustomCmap         = 'Gray_Caribbean/Gray_Caribbean_0.70.pymap'
58  Fz_interpolation      = 'bicubic'
59  Fz_Planes             = [ False , False , True ]   <----  (ii)
60  Fz_ZPlane_clims       = [ None, None]
61  Fz_YPlane_clims       = [ None, None]
62  Fz_XPlane_clims       = [ None, None]
63  ###### Lateral Force Images Angle: The angle defines the plotting axis, angle is
    relative to the PPM X axis
```

FIGURE 3.6: A section of the VarUser file highlighting the area to specify what planar images to generate.

The section of the VarUser file that controls plotting of planes of the 'OutFz.npy' matrix is shown in Fig. 3.6. By setting '*Fz_MakePlots*' to '*True*' at (i), the PPM simulation would output planar images of the molecule. Specifically, the variable '*Fz_Planes*'(shown in (ii)) controls the type of plane that should be generated. In this case, with '*Fz_Planes*' having the structure [X, Y, Z], setting its value to $[False, False, True]$ implies that only Z-plane images would be generated. To generate X, Y and Z plane images, set '*Fz_Planes*' to $[True, True, True]$.

3.5.2 Executing a simulation

To run a PPM simulation, three files are required. They are the *VarUser.py* file, input cube file and a shell script (*control.sh*). The shell script is needed to execute the PPM simulation. After setting up the *VarUser.py* file as shown in Sec. 3.5.1, the simulation can be run using the following procedure,

- Create a new directory within the Probe Particle Model directory
- Copy the input cube file, the *VarUser.py* and *control.sh* within the new directory
- Using a linux terminal, navigate to this new directory
- Type the command *bash control.sh*

3.5.3 Running simulation for a range of parameters

In some cases one might want to run simulations for a range of parameters. To do this, the *run.sh* file (see Fig. 3.7) of the base PPM should be used. This *run.sh* file is the original shell scipt of the PPM code that executes the simulation. Note that

the *control.sh* file mentioned in 3.5.2 executes this *run.sh* file while also adding other functionalities to the PPM defined in the VarUser file. In Fig. 3.7, the separate steps of the PPM calculations have been highlighted showing,

1. Calculation of the LJ force field for the molecule
2. Relaxation process of the PP in the LJ force field. This relaxation process is calculated for stiffness $k = 0.16$ and effective charge $q = -0.05e$.
3. Plotting the results

The structure of this shell script gives the opportunity to run simulations for a range of parameters as shown by the red arrow in Fig. 3.7. To run a simulation for a range of k values, substitute the circled area with,

```
--krange 0.16 0.20 3
```

where the range of values takes the form of a numpy *linspace* function. Similarly, for a range of q values use,

```
--qrange -0.05 0.05 3
```

This means that the simulation will run for three q data values from $-0.05e$ to $0.05e$. Generally, the following format should be used,

```
--krange min max nK --qrange min max nQ
```

Additionally, we can choose to output constant height F(z) images or df images by including that option in the script as follows,

```
--krange min max nK --qrange min max nQ --Fz
```

or

```
--krange min max nK --qrange min max nQ --df
```

3.6 Working with Output files

3.6.1 The '*OutFz.npy*' file

The OutFz.npy file is a numpy array of the total force felt by the PP. Specifically, this numpy array has the characteristic shape (Z, Y, X). Note that this is not typical since numpy arrays usually have the shape (X, Y, Z). This is essentially analogous to 3D volumetric data generated in real AFM experiments. While it has been demonstrated how planar images may be generated from this, it is worth practising how to generate these plots manually.

3.6.2 Plotting Z-plane images

A basic python code can be used to plot z-plane images which requires an understanding of matrix slicing, using 'for loops' and general manipulation of numpy arrays. A useful introduction to using numpy arrays can be found here. In Fig. 3.8, a working example of python code for plotting Z-plane images for a PTCDA molecule

FIGURE 3.7: The *run.sh* file.

has been shown. Using the *linspace* function of the numpy module, the x and y data points for 2D planar image has been specified as shown in (ii) of Fig. 3.8. Since the scanStep variable controls the number of data points within a specified scanning range, there are 401 data points for scanRange[X] and scanRange[Y] (corresponding to a scanStep of 0.05Å).

```
1
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5
6  # ScanRange coordinate system parameters
7  scanMin      = [  0.00,  0.00, 17.65]
8  scanMax      = [    20,    20, 30.00]
9
10 #load OutFz matrix
11 load_matrix = np.load('OutFz.npy')
12
13 Axis   = np.linspace(scanMin[2], scanMax[2], load_matrix.shape[0])
14 plot_num = np.arange(0, 62, 1)
15
16 x = np.linspace(0, 20, 401)
17 y = np.linspace(0, 20, 401)
18 X, Y = np.meshgrid(x, y)
19
20 # A for loop for generating all the Z-planes for the specific scan range and
   resolution
21 for i in plot_num:
22     Fz1 = load_matrix[i,:,:]
23     Extent = (scanMin[0], scanMax[0], scanMin[1], scanMax[1])
24     plt.imshow(Fz1, extent=Extent, origin='lower', cmap='gray')
25     plt.xlabel('x/'r'$\AA$', fontsize=8)
26     plt.ylabel('y/'r'$\AA$', fontsize=8)
27     plt.title('Plot '+str(i)+'      Z='+str(np.round(Axis[i], 2)) + r'$\AA$' ,
       fontsize=10.0)
28     plt.axis(aspect='image')
29     plt.tight_layout()
30     plt.savefig('Fz_surface ' + str(i))
31     plt.close
```

(i)

FIGURE 3.8: An example of a Z-plane python code. The use of a for-loop on line 21 ensures that Z-plane images are generated at each specified tip-sample height.

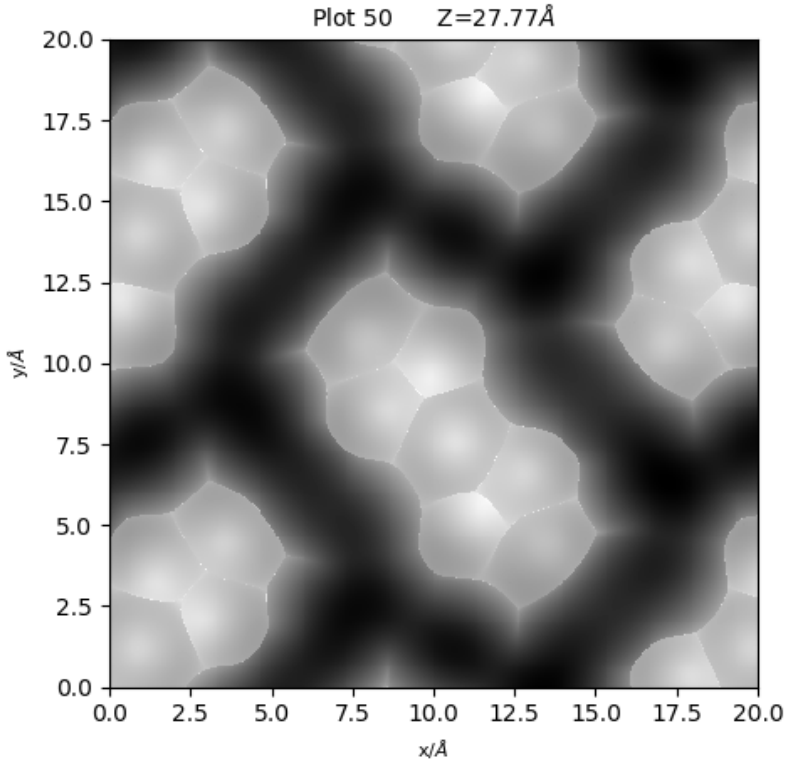

FIGURE 3.9: A corresponding Z-plane image of the shown python code.

3.6.3 Plotting Y-plane images

Similarly, the Y-plane images are can be generated by using the python code shown in Fig. 3.10. In line 10 of this code (red rectangle), the Y-plane to be plotted has been specified. This line of code is a simple illustration of numpy matrix slicing. Since numpy indexes data points of the OutFz matrix, the index corresponding to the particular y-value has to be calculated. For this example, to slice at $y = 10$Å, divide 10Å by 0.05Å(which is the scanStep). This gives 200 as shown in the rectangle in Fig. 3.10.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy import ndimage
4
5  scanMin      = [  0.00,  0.00, 17.65]
6  scanMax      = [    20,    20, 20.00]
7
8  load_matrix = np.load('OutFz.npy')
9
10 Fz1 = load_matrix[:,200,:]
11
12 x = np.linspace(0, 20, 401)
13 z = np.linspace(17.65, 20, 235)
14 X, Z = np.meshgrid(x, z)
15
16 Extent = (scanMin[0], scanMax[0], scanMin[2], scanMax[2])
17 Axis   = np.linspace(scanMin[1], scanMax[1], load_matrix.shape[1])
18
19 plt.imshow(Fz1, extent=Extent, origin='lower', cmap='rainbow')
20 plt.xlabel('x/'r'$\AA$', fontsize=6)
21 plt.ylabel('z/'r'$\AA$', fontsize=6)
22 plt.title('Y = ' +str(np.round(Axis[200], 2)) + r'$\AA$', fontsize=10)
23 plt.colorbar()
24 plt.axis(aspect='image')
25 #plt.savefig('Y_plane10')
26 #plt.close()
27 plt.show()
```

FIGURE 3.10: An example of a Y-plane python code

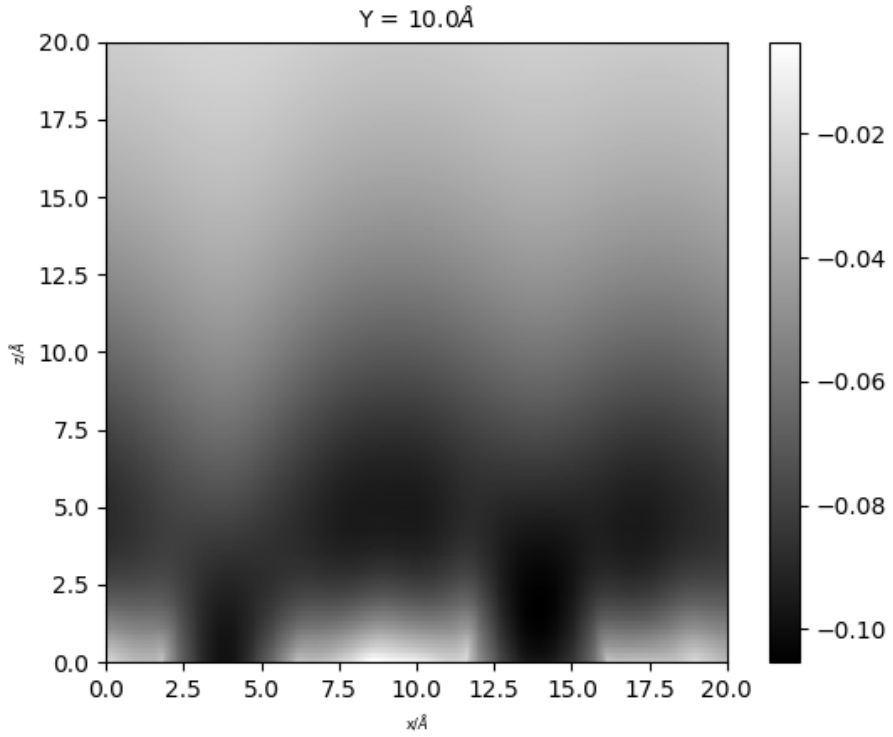

FIGURE 3.11: Corresponding Y-plane image of the above shown python code

3.7 Worked Example

We will demonstrate in a step-by-step manner a working example of a PPM simulation.

3.7.1 Step 1

As shown in Fig. 3.12,

- Make a directory containing the required the files
- Move this directory to the '*ProbeParticleModel*' folder.

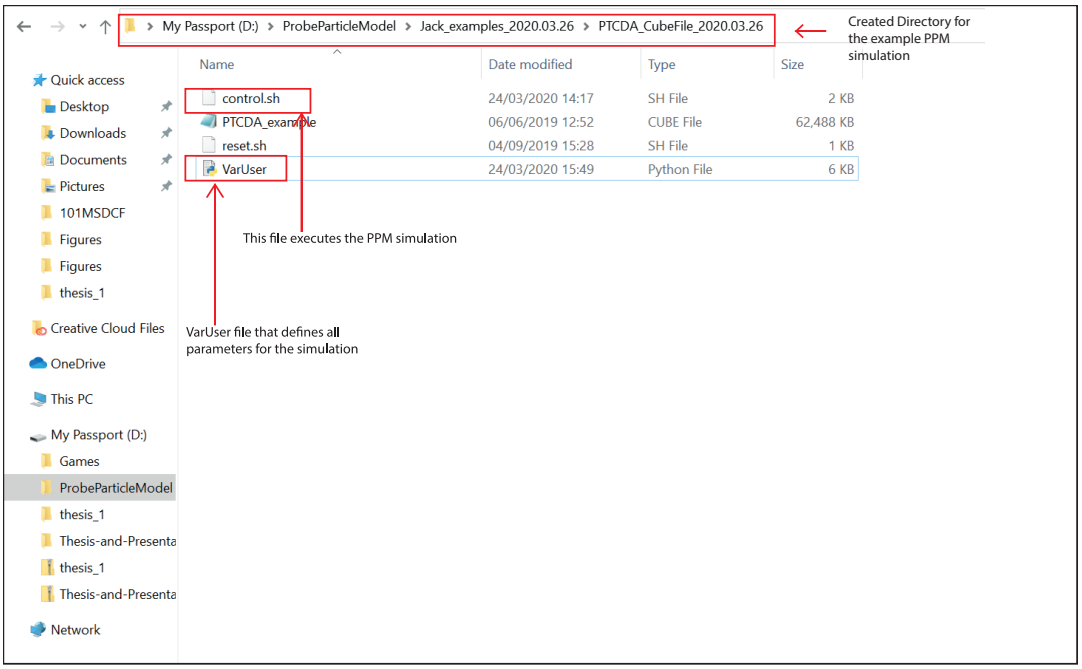

FIGURE 3.12: Setting up the simulation directory.

3.7.2 Step 2

We move on to set the parameters for the PPM Simulation. In this case, the gNCS and the SRCS will be set automatically as shown in Fig. 3.13

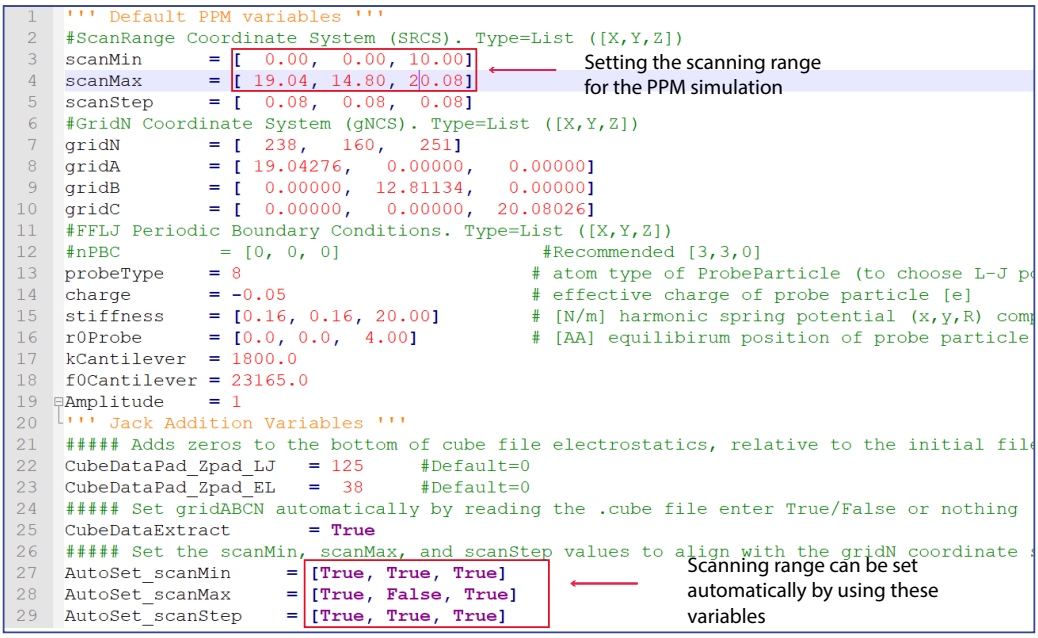

FIGURE 3.13: Setting up the VarUser file.

3.7.3 Step 3

To execute the PPM simulation, we open a linux terminal and do the following:

1. Navigate to the file directory containing the shell script (*control.sh*)
2. Make sure that the python3 environment consisting of all necessary modules indicated in Sec. 3.1 has been activated using

```
conda activate [environment name]
```

3. Type

```
bash control.sh
```

4. Open the directory of the Z-plane as shown in Fig. 3.14

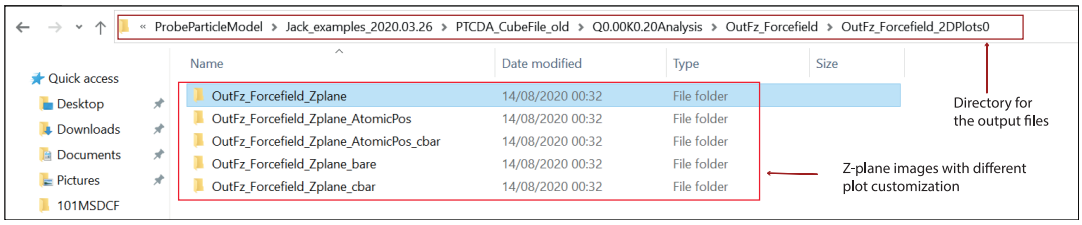

FIGURE 3.14: Directory for the output Z-plane images.

3.7.4 Step 4

The directory where the raw output files of the PPM ($FFLJ.npy, FFel\_i.npy, OutFz.npy$) have been shown where i is either $x, y,$ *or* z (see Fig. 3.15). In Fig. 3.15 the calculated LJ potentials and the electrostatic forces felt by the PP are stored in the directory where the *VarUser* file is found. However, the $OutFz.npy$ file is stored in another directory with the filename format, $Q[qvalue]K[kvalue]$.

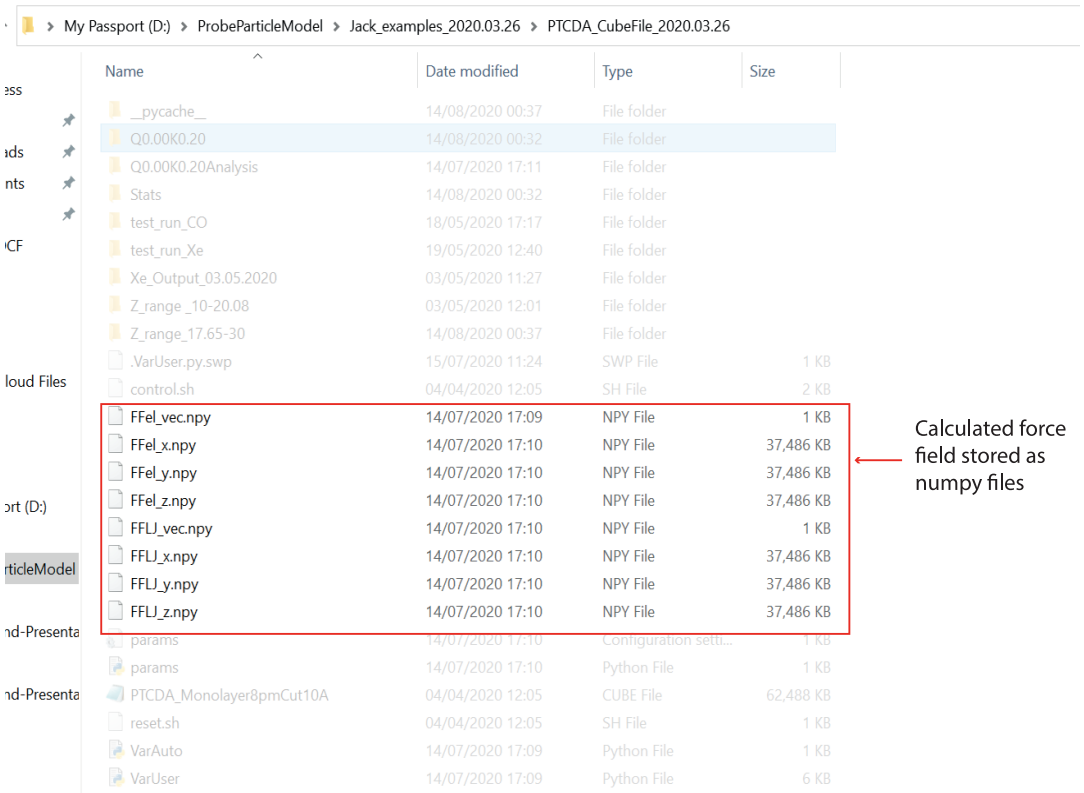

FIGURE 3.15

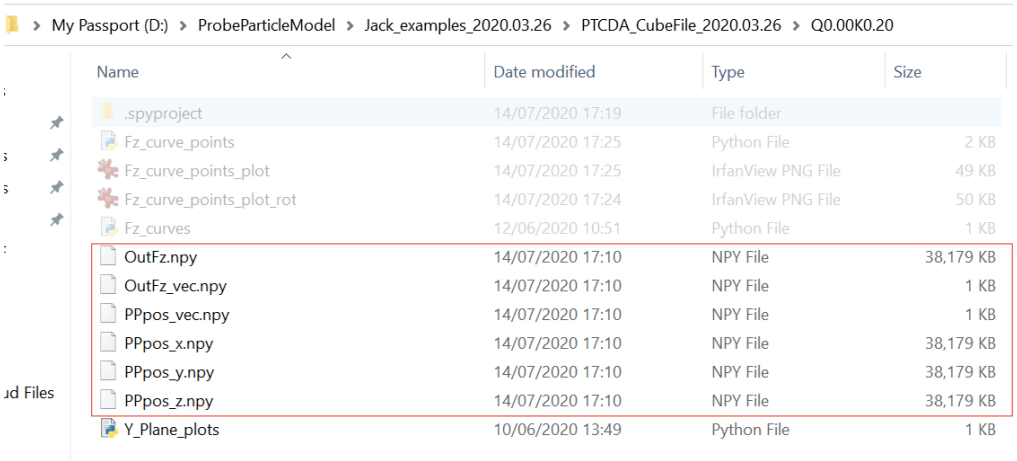

FIGURE 3.16

Bibliography

[1] Prokop Hapala et al. “Mechanism of high-resolution STM/AFM imaging with functionalized tips”. In: *Physical Review B - Condensed Matter and Materials Physics* 90.8 (2014). ISSN: 1550235X. DOI: 10.1103/PhysRevB.90.085421. arXiv: 1406.3562.

[2] Henry Jack. “Experimental and Thoeretical Study of Intermolecular Bonds”. In: (2019).