PHAS0097:Astrophysics/Physics Project:

Watching Wiggling Biomolecules with Atomic Force Microscopy

Joshua Giblin-Burnham

Supervisor: Prof. Bart Hoogenboom

Theoretical Physics (MSci)

Faculty of Mathematical and Physical Sciences

Department of Physics and Astronomy

Word Count: 7480

University College London

April 2023

Submission of coursework for Physics and Astronomy course **PHAS0097/PHAS0096/PHAS0048** 2022/23

Please sign, date and return this form with your coursework by the specified deadline.

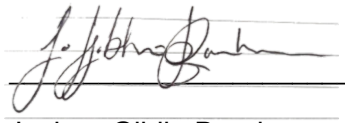
DECLARATION OF OWNERSHIP

I confirm that I have read and understood the guidelines on plagiarism, that I understand the meaning of plagiarism and that I may be penalised for submitting work that has been plagiarised.

I confirm that all work will also be submitted electronically and that this can be checked using the JISC detection service, Turnitin®.

I declare that all material presented in the accompanying work is entirely my own work except where explicitly and individually indicated and that all sources used in its preparation and all quotations are clearly cited.

Should this statement prove to be untrue, I recognise the right of the Board of Examiners to recommend what action should be taken in line with UCL's regulations.

Signed 

PrintName Joshua Giblin Burnham

Dated 24/04/23

Title	Date Received	Examiner	Examiner's Signature	Mark

Abstract

Atomic Force Microscopy (AFM) is a versatile three-dimensional topographic technique implementing a mechanical probe to raster-scan and image sample surfaces. The technique provides reliable nanometer measurements of materials[1–4] and has become a valuable tool with a diverse range of applications in areas such as materials physics, nanotechnology, electronics, and biology[3,5,6]. However, there are limited computational recreations of AFM imaging, and the area could benefit from greater tools to aid in interpreting surface characteristics. Consequently, this research presents novel computational modelling of AFM imaging, specifically for biomolecular samples. Currently, simulations of AFM imaging use a hard-sphere model and neglect tip indentation[7]. This research implemented Finite Element Modelling (FEM) and the commercial software ABAQUS to simulate AFM tip indentation and produce force curves that accounted for contact dynamics. Initial tests verified the accuracy of ABAQUS, focusing on the indentation of elastic half-spaces and spheres. The elastic half-space simulations showed good agreement with the theoretical models. In contrast, the results indicate that simple Hertzian models underestimate the elastic modulus of spherical samples and instead require Double Contact models. Moreover, a novel formulation of the Double Contact model for conical indenters demonstrated significant predictive power over a range of surface radii. Next, a FEM approach was applied to analyse the compression of simple hemispheres and periodic surfaces during AFM imaging. These simulations highlighted the dependency of the elastic behaviour on the contact radius and tip convolution. Our results indicated that larger indenters require larger forces to compress the sample to the same extent. In addition, Fourier analysis of the simulated AFM contours elucidated a possible novel trend, that larger indentation forces recover more of a surface's periodicity. Finally, a FEM simulation of AFM imaging was applied to simulate the appearance of B-DNA Dodecamer. Simulations used an assembly of the AFM tip and the biomolecule surface to produce individual indentations across the sample. Subsequently, contours of constant force are used to return an AFM image. These simulations show the viability of the FEM approach in reproducing the AFM dynamics and provide a wealth of extensions to be explored.

Contents

1 Introduction

1.1 Introduction to Atomic Force Microscopy

The Atomic Force Microscope (AFM) is a type of Scanning Probing Microscope[8] developed by Binnig, Quate and Gerbe at IBM, Zurich[9,10]. It uses a sharp probe tip to raster-scan a sample's topology and has become a versatile tool[3,5,6]. The technique has various applications, for example, in atomic imaging of crystal structures[11] and surface measurements of materials and polymer films[12–16]. In addition, AFM can image under natural conditions, such as in aqueous solutions and in real-time, allowing imaging of cell dynamics and biological processes. This includes imaging protein unfolding[17] and conformational changes[18], alongside, characterising microbial surfaces[19,20]. The ability to image and measure the physical properties of microbial surfaces can give important insight into microbiology, such as improving inhibition and cellular damage produced by antimicrobial compounds[19,21].

The imaging of AFM is based on detecting atomic forces acting between a sharp probe tip and the surface of a sample. A schematic of an AFM is shown in Figure 1. A cantilever holds the sharp tip used to probe the sample surface. A laser beam detection system using the position-sensitive photodiode (PSPD) enables the AFM system to monitor and record the deflection of the cantilever. The tip is raster-scanned across the surface of a sample, and interaction between the tip and surface causes deflection. Subsequently, the deflection is processed in the feedback system, designed to compensate for the change in topology and hold the deflective force constant during scanning. The tip deflection is compared to a reference force, producing an error signal that generates the feedback signal. The surface topography is then determined by mapping the contours of equal force[22].

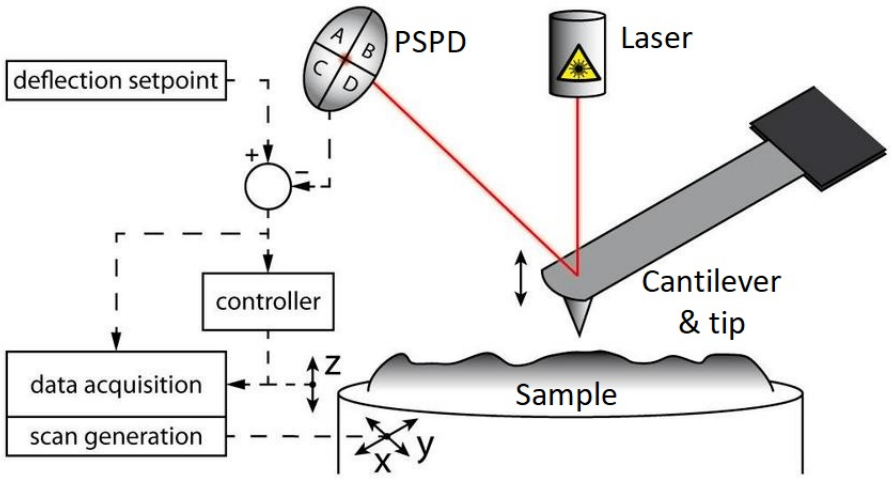


Figure 1: Schematic of AFM mechanism from Schitter *et al.*[23]

However, as with any experimental technique, AFM has limitations. Various effects can lead to ambiguity in images and image artefacts. A key source of error is a consequence of the resolution being directly dependent on imaging force and the probe geometry[24]. Imaging with large forces can dramatically reduce image resolution and damage the surface. Furthermore, probe geometry and its interaction with the sample is important to image contrast[24]. As shown in Figure 2**A** , an AFM image is a convolution of the probe geometry and the sample's topology. Therefore, the tip-sample convolution produces a trace of the tip geometry over the surface, broadening protrusions and narrowing holes in the surface.

Similarly to tip convolution, AFM can produce image artefacts due to tip interaction when scanning over the edge of a surface structure. As the tip loses contact with the sample, the deflective forces diminish over the edge. As a result, the probe must increase scan depth; however, the cantilever descends at a capped rate, so the tip gradually approaches the surface,

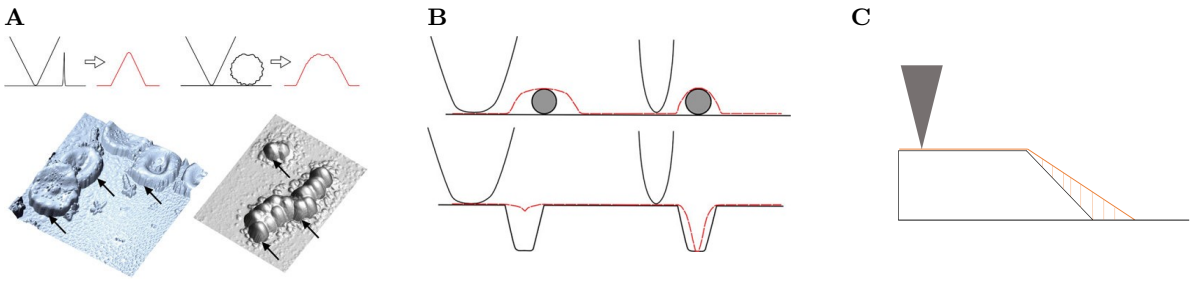


Figure 2: Illustrations of typical AFM artefacts taken or adapted from Eaton & West, Atomic Force Microscopy[10]. (A) Illustration of tip convolution artefact produced in AFM images. (B) Illustration of the widening/narrowing of surface features due to tip convolution. (C) Illustration of an elongated edge due to the parachuting effect.

minimising the risk of over-indenting. This causes a linearly protracted trace of the gradient and produces an elongated edge in the direction of the image scan, as shown in Figure 2**C** . This is known as parachuting.

Furthermore, other errors may arise due to environmental surroundings. For example, environmental vibrations can cause the probe to vibrate and produce artefacts and blur. Similarly, thermal drift is produced from prolonged usage, which causes the probe to expand/ contract thermally and produce deviations in the system. In this report, we describe computational simulations of AFM imaging to aid in interpreting images and artefacts. Moreover, we present some quantitative analysis of the compression produced in these simulations to explore the effect of imaging force and tip geometry in AFM imaging.

1.2 Previous Simulations of Atomic Force Microscopy Images

Currently, there are a limited number of AFM imaging simulations, leaving a prominent area for development. Recent work by Amyot R, Flechsig H *et al.*.[7] produced the BiomolecularAFMviewer in 2020. The BiomolecularAFMviewer, similar to this project, uses protein structural data to simulate AFM images (shown in Figure 3) and aids in interpreting experimental observations. In follow-up work in 2022, Amyot R, Flechsig H *et al.*.[25] used the software to reconstruct resolution-limited experimental images as an example of the application of the software. The modelling used a hard sphere model of the surface and indenter, evaluating the vertical height of the tip at the initial surface-surface contact. Therefore, the simulation does not account for indentation into the surface, surface deflection, off-axial forces that produce sliding and friction, or elastic properties of the surface. This approach could be improved by accounting for force curves and the indentation of the tip with elastic properties of the material.

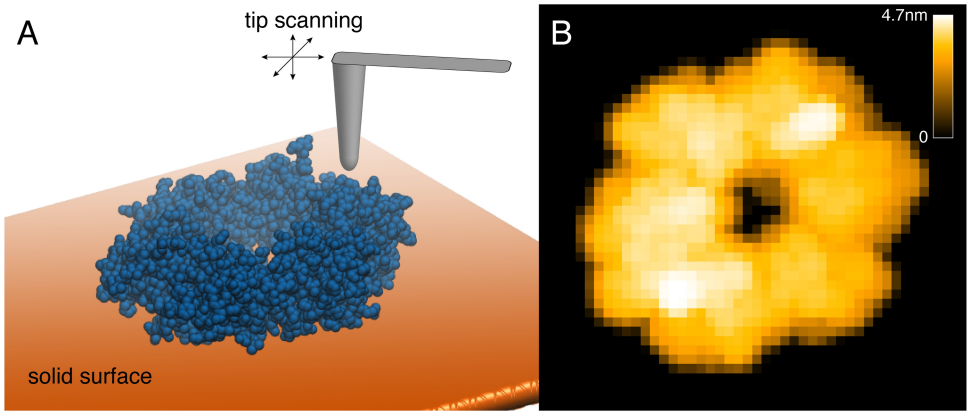


Figure 3: Graphic of simulation from BioAFMviewer[7]. (A) 3D geometry of simulation. (B) Simulated AFM image.

Other computational simulations of AFM have been used to produce such force curves that reflect these properties. However, these applications study quantitative AFM results as opposed to imaging[26–29]. Previous work has shown the viability of the commercial software ABAQUS and Finite Element Modelling (FEM) in the study of indention in AFM; Liu *et al.*[26] validated a FEM model for AFM indention with less than 10% error when comparing the simulated force-indentation curves with the experimental data. Similar analysis of experimental data with FEM done by Roduit *et al.*[30] and Han *et al.*[27] made use of the ABAQUS software to complete calculations. The work by Rajabifar *et al.*.[31] simulated the viscoelasticity contact between an AFM tip and a surface. This showed a fast and accurate use of FEM to simulate AFM indentation and the associated force curves.

1.3 A Finite Element Approach to Imaging

Consequently, our research presents novel FEM and computational methods to model and explore AFM images, specifically in biomolecular samples. The primary work of this research focuses on improving the modelling of previous AFM simulations from a hard sphere model to a model with tip surface interactions. FEM simulations enable us to simulate tip indentation and generate force curves that incorporate more intricate forces and dynamics. We employ several FEM simulations to investigate contact models of indentation and sample compression in AFM imaging. The overall goal was to produce more accurate images and artefacts.

FEM subdivides the geometry into small, discrete finite elements, producing a surface mesh as shown in Figure 4. The dynamics are approximated over these finite elements and result in a system of algebraic equations. The system is then modelled using the assembled equations for the finite elements and solutions are approximated via the calculus of variations and minimising an associated error function. Producing AFM images requires the calculation of contours of constant indentation force. Using FEM, the sample surface and probe tip geometry are recreated, and individual indentations across the sample are simulated. This provides 4-dimensional arrays of indenter position and corresponding indentation force. From this, contours of constant force can be used to return the surface contour.

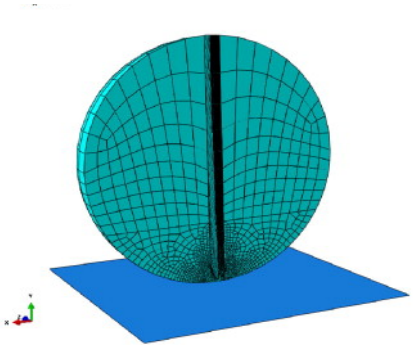


Figure 4: Graphic of FEM mesh for a simulation of sphere indentation by Zheng *et al.*[32]

Our implementation of FEM utilises the commercial engineering software ABAQUS. However, as ABAQUS is an engineering-focused software and simulating biological AFM imaging is a novel application for ABAQUS, initial tests verified its viability. The accuracy of ABAQUS was evaluated through tests that focused on elastic indentation, performed on both elastic half-spaces with varying depths and elastic spheres with varying radii. These tests are outlined in Section 2.2. Following these initial simulations, a FEM approach was applied to produce simulations of the AFM raster-scanning dynamic, outlined in Section 2.3. These simulations were used to analyse the compression of a simple hemisphere and periodic surfaces (Section 2.4). Finally, a FEM approach was applied to simulate the AFM appearance of simple biomolecules, outlined in Section 2.3.3.

2 Methodology

2.1 ABAQUS Finite Element Analysis

To provide accessible simulations of AFM imaging, our implementation of ABAQUS utilised Python scripts to produce simulations (see Appendix C). An ABAQUS model defines seven basic modules (shown in Figure 5): parts, properties, assembly, interactions, steps, loads/boundary conditions, and mesh. These moduli are defined within the Python code using predefined variables for the simulations, with geometric dimensions defined in nm and forces as pN. From this, a Python submission script is created and run by ABAQUS software. A brief outline of ABAQUS modelling follows.

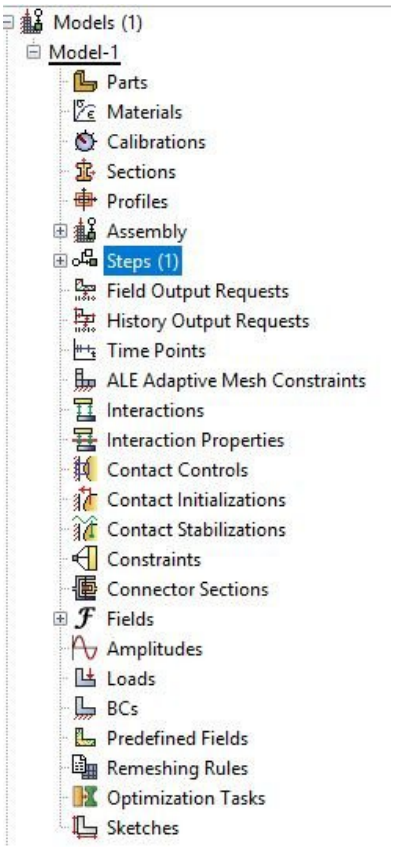


Figure 5: Display of model modules within ABAQUS GUI

The geometry of our simulation is the foundation of a model and is built as parts in the Parts module. Parts are formed from two-dimensional sketches that form a features profile. These profiles can be extruded, revolved, or swept to create part geometry or used directly to form a planar or axisymmetric part. Parts can be deformable, discrete rigid, analytical rigid, or Eulerian. The material properties and distribution for individual parts or sections of parts are set using the Properties module. The individual parts can then be joined and arranged in the Assembly module to create an assembly instance of the parts (known as part instances).

The behaviour of the part instances are defined in the interaction module, and various loads and boundary conditions can be applied to the part instances within the Load module. The Mesh module then allows the geometry to be coarse-grained and the surface subdivided into small, discrete finite elements. The steps module defines the sequence of one or more analysis steps and the increments of the analysis at which dynamics are propagated. The steps track the changes in the loading and boundary conditions of the model. The model can then be submitted, and the ABAQUS software runs the Finite Element Analysis.

2.2 Verification of Contact Models

Applying ABAQUS to assessing simple contact models of indentation provides a robust validation of the FEM approach. Three types of indenters were used for the simulations: conical, spherical, and spherically-capped conical. The behaviour of each indenter was characterized and compared with the theoretical indentation models: Hertz[33] and Dimitriadis[34] for spherical indenters, and Sneddon[27] for conical indenters (Details of the models are shown in the Appendix B). The simulations focused on the elastic indentation of both elastic half-spaces/planes of varying depths and elastic spheres of varying radii. For computational efficiency, asymmetric models centered around the indentation axis (y-axis) were used, as shown in Figure 6. Indenters were modelled as rigid (incompressible) parts restricted to the y-direction. The elastic half-spaces were fixed at the base, while the elastic spheres had a fixed, rigid base beneath them. The model surfaces were simulated as homogeneous, isotropic elastic materials with Young's modulus and Poisson ratio of $1000KPa$ and 0.3, respectively. These values were chosen as they are typical biomolecule values, and as such, dimensions were given in nm and pN (although somewhat arbitrary as analysis is dimensionless).

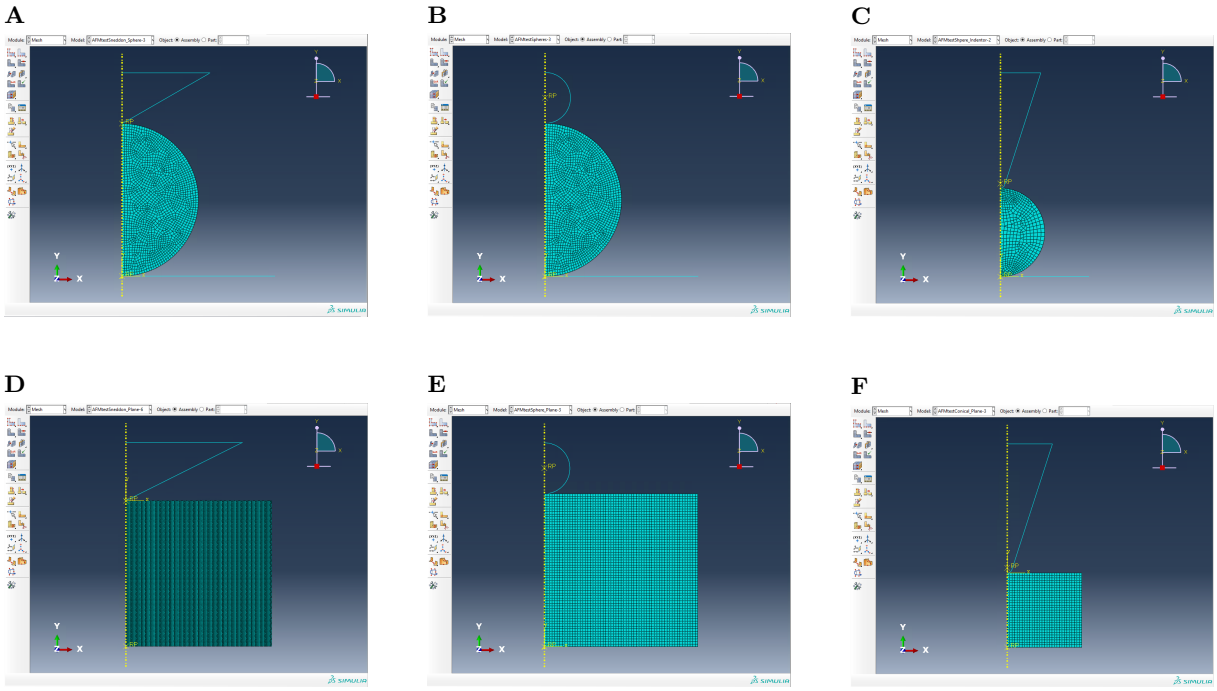


Figure 6: ABAQUS model assembly from elastic indentation tests. Elastic half-spaces were modelled asymmetrically using rectangles. Elastic spheres were modelled using semi-circles with a rigid base beneath. Models include (A) Conical indentation of an elastic sphere. (B) Spherical indentation of an elastic sphere. (C) Spherically-capped conical indentation of an elastic sphere. (D) Conical indentation of an elastic half-space. (E) Spherical indentation of elastic half-space. (F) Spherically-capped conical indentation of an elastic half-space.

The contact was modelled as a "surface-to-surface" type with "hard" (nonadhesive) properties in the normal direction and "rough" (non-slip) Coulomb friction in the tangential direction. Vertical force and indentation data was sampled via reference points at the centre of the indenter. As it is a rigid part, ABAQUS maps the forces and displacement of the surface of the part to the reference point. For elastic spheres, Double Contact Models[35,36] were required to account for more complex dynamics discussed in the results. Simulations were quasi-static computations using an implicit algorithm and approximately 30000 tetrahedral (R3D10) elements. The simulation data was exported to Python and scipy.curve_fit module was used to fit the desired contact model. Example scripts in Appendix C.

2.3 Simulation of AFM Imaging

2.3.1 General Scan Dynamics for Imaging

Following the application of FEM to single indentations, similar simulations were created to extend to AFM imaging. Independent ABAQUS simulations were produced in which individual indentations at positions across the surface's domain replicate the dynamics of an AFM raster-scan. Our AFM simulation subdivides the XY domain of the geometry into bins to produce the scan positions. The initial vertical heights of the indenter are then calculated at each position. Next, the independent ABAQUS simulations for the indentation at each position are produced, and corresponding vertical forces and displacements are extracted. This provided a four-dimensional array of indenter positions and corresponding force over the surface. Subsequently, computation of the contours for a given reference force produces the final AFM images shown in Figure 7. This methodology is computationally efficient as it utilises the initial heights computed from hard-sphere contact, resulting in the indentation data being consistently calculated to the same depth across the entire surface. This is as the hard-sphere contact represents the tangent points of the two surfaces' across the scan.

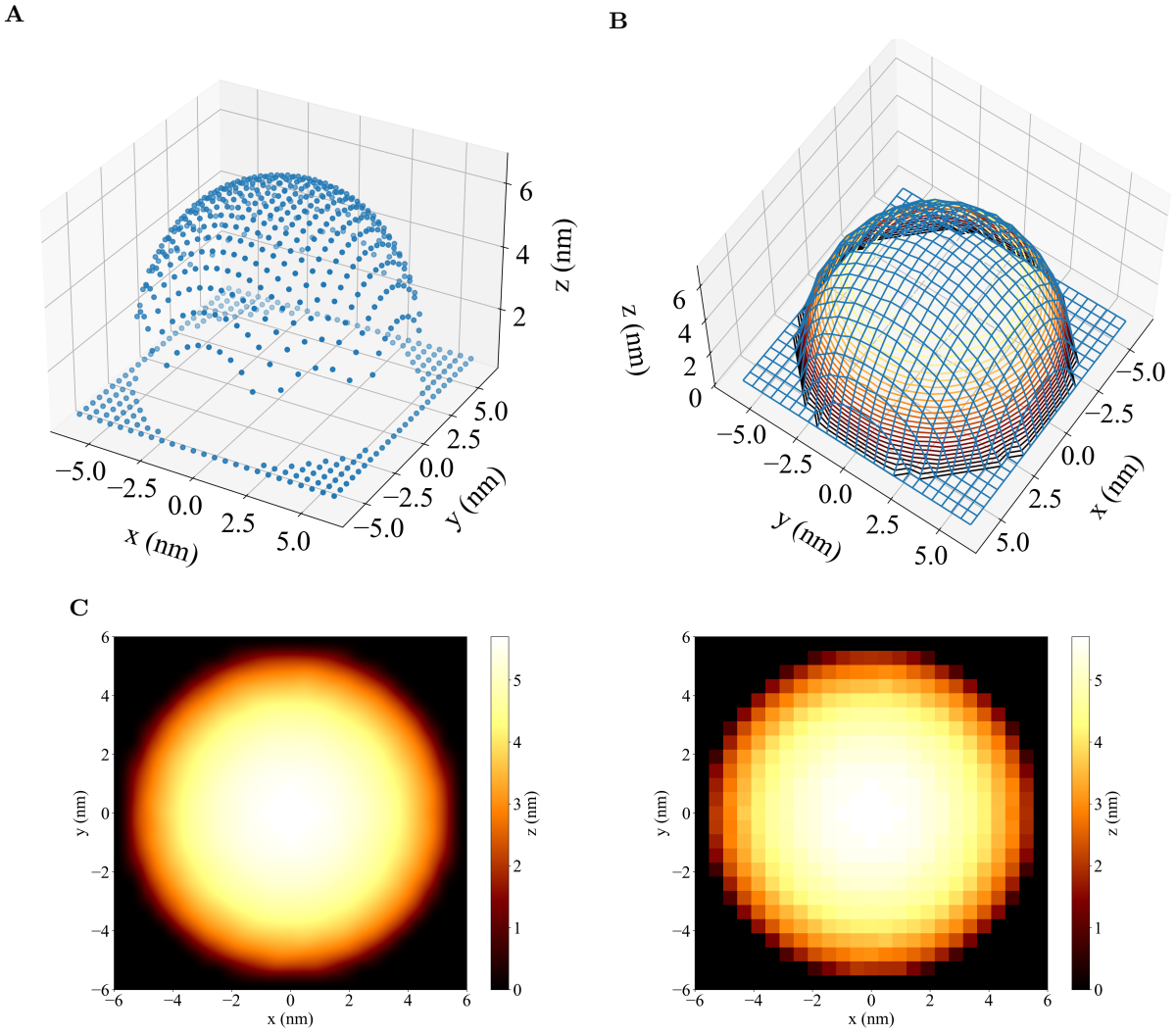


Figure 7: Plots for simulated AFM image on a hemisphere of radius r=5nm. $E = 1000KPa$, $\nu = 0.3$, and scan bins of 0.5nm. (A) Vertical initial scan positions for hemisphere. (B) Three-dimensional plot of force contours for 100pN for simulated AFM image. (C) Simulated 2D heatmap AFM image for 100pN force. Left: Interpolation. Right: Raw data

2.3.2 Image Processing and Interpolation

The pseudo-AFM images are produced from two-dimensional arrays of surface heights over the XY grid of scan positions. These heights map the surface contour of equal force over the scan. These contours are calculated from the indentation depth and force arrays produced by the ABAQUS/FEM simulations. For a given threshold/ reference force, the corresponding depths are found using a list comprehension (see Figure 8). The list comprehension iterates over the force data for each scan position and compares it with the reference force. The index for the value equal or greater than the reference force is found and the corresponding depth is stored. This is used to calculate the surface heights. If no values in the force array for a scan position are greater than the reference force, the function sets the corresponding contour value to the maximum indented depth. The contour data is then reshaped into the 2D grid representing the scan positions. Subsequently, the Matplotlib imshow function is used to produce the visualisation with a Colormap to illustrate the variation in surface heights. Normalisation of the colourmaps applies either linear scaling over the domain or power normalisation depending on detail contrast. Moreover, to increase pixel density images are interpolated using Mathplotlib imshow functions in-built bicubic interpolation.

```python
def ForceContours(U2, RF,forceRef, scanPos, baseDims, binSize):
    '''
    Function to calculate contours/z heights of constant force in simulation data for given threshold force.

        Parameters:
            U2 (arr)         - Array of indentors z displacement over scan position
            RF (arr)         - Array of reaction force on indentor reference point
            forceRef (float) - Threshold force to evaluate indentation contours at (pN)
            scanPos (arr)    - Array of coordinates [x,y,z] of scan positions to image biomolecule
            baseDims (list)  - Geometric parameters for defining base/ substrate structure [width, height, depth]
            binSize (float)  - Width of bins that subdivid xy domain during raster scanning/ spacing of the positions sampled over
        Return:
            X (arr) - 2D array of x coordinates over grid positions
            Y (arr) - 2D array of y coordinates over grid positions
            Z (arr) - 2D array of z coordinates of force contour over grid positions
    '''

    # Initialise dimensional variables
    xNum = int(baseDims[0]/binSize)+1
    yNum = int(baseDims[1]/binSize)+1

    # Initialise contour array
    forceContour = np.zeros(len(RF))

    # Loop over each reaction force array, i.e. each scan positions
    for i in range(len(RF)):

        # If maximum for at this position is greater than Reference force
        if np.max(RF[i]) > forceRef:
            # Return index of force threshold and store related depth
            j = [ k for k,v in enumerate(RF[i]) if v > forceRef][0]

            # Set surface height for reference height
            forceContour[i] = scanPos[i,2] + U2[i,j]

        # If no value above freshold set value at bottom height
        else:
            forceContour[i] = scanPos[i,2] + U2[i,-1]

    # Format x,y,z position for force contour
    X  = scanPos.reshape(yNum, xNum, 3)[:,:,0]
    Y  = scanPos.reshape(yNum, xNum, 3)[:,:,1]
    Z  = forceContour.reshape(yNum, xNum)

    return X, Y, Z
```

Figure 8: Code Snippet showing the calculation of force contours for AFM image. For a given threshold/ reference force, the corresponding depths are used to calculate the surface heights over the scan positions.

2.3.3 Application to Biomolecules

The application of these dynamics to biomolecules requires some simple modifications. Biological structures are produced using Protein Data Bank (PDB) files that specify the constituent atoms of a biomolecule and the corresponding coordinates. As the simplest approach, the biomolecule is modelled as an elastic material produced from the assembly of spheres (with van der Waals radius) of the individual atoms. The structure is assumed to be a continuous, homogeneous and isotropy material, with a typical biological Young's Modulus and Poisson ratio of 1000KPa and

0.3, respectively. The molecule is partially embedded in a rigid base/ substrate, and the scan positions are then calculated over the domain of the base. The embedded portion simulates a soft molecule absorbed onto a solid support and the molecule is then fixed at its base using boundary conditions. The AFM probe tip is modelled as a rigid capped conical indenter where the indentation is non-slip and without adhesion. The contact is modelled as "surface to surface" type, with the properties of hard contact in the normal direction and "frictionless" in the tangential direction to ensure no slip and no adhesion. Indentation data from the indenter is mapped and sampled via reference points at the centre. The computations were quasi-static with an implicit algorithm and using R3D10 tetrahedral elements (generally greater than 100,000). The elastic constitutive relations are integrated with the main body of the Abaqus code.

The initial scan positions must be calculated numerically as PDBs only provide atom positions, not surface structure. The XY scan positions are produced by creating a rectangular grid of positions over the base, each separated by a predefined bin size. The heights at each position are calculated by setting the tip above the sample and calculating the minimum vertical distance between the tip and the molecule's surface (corresponding to the tangent point of the two surface's). Figure 9 illustrates the calculations made by the code. For each scan position, a loop is used to calculate the radial distances to the atoms in the biomolecule (shown as $R_{Interact}$ in Figure 9). If the atom lies within the indenters boundary ($R_{Boundary}$), the atom and indenter could interact. Therefore, the vertical distances between the surface of the indenter and the "interacting atoms" of the molecule are computed. This produces an array of "dz" values which represents all differences in height between the tip and portion of molecule located within the boundary of the indenter. As illustrated in Figure 9, the minimum of these values gives the vertical distance corresponding to the position where the tip is in tangential contact. Subsequently, using this minimum dz value the tangent points for a scan location can be calculated. Repeating this for all scan locations provides an array of coordinates [x,y,z] of scan positions. For computational efficiency, this can be clipped to only include positions where the tip and molecule interact.

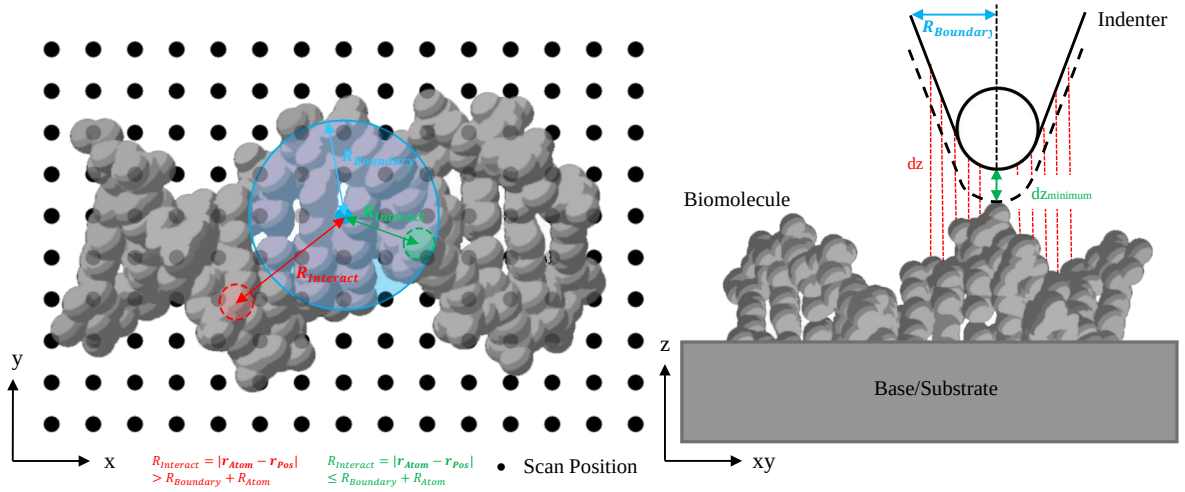


Figure 9: Schematic diagrams illustrating the calculation of initial scan heights in AFM code. Left: Illustrates the calculation atoms on the surface within the indenters boundary. Black dots represent the XY grid of scan positions. The calculation is restricted to the XY plane in which only atoms inside the radial extent of the indenters, $R_{Boundary}$ (blue), are calculated. The red atom represents an atom outside the extent and thus is omitted, as appose to the green atom, which is included. Right: Illustrates calculation of heights for each given position. An array of all distances (dz) between the indenters and molecules surfaces are calculated (red). The minima of these distances thus give the translation distance to place the indenter in tangential contact.

An outline of the simulation script is shown in Figure 10. The code has three main phases: In

the preprocessing phase, the desired PDB file is imported via the PDB function and used to process the biomolecule's structure in the MolecularStructure function. Subsequently, the base and indenter geometry is calculated, and the ScanGeometry function determines scan positions for the simulation. These variables and other predefined simulation variables are exported to CSV files using the ExportVariables function. Finally, the CSV files and the ABAQUS scripts are transferred to the remote server where the simulations are carried out. There are 3 ABAQUS scripts run in the submission phase. First, the SurfaceModel and RasterScan scripts are run sequentially using the RemoteCommand function. These scripts produce the ABAQUS surface model and input files for simulations at each scan position. Next, the BatchSubmission function creates a script to run the input files sequentially on the remote server. Once the simulations are complete, the data is stored in ODB files. Running the ODBAnalyis script extracts the indentation data and exports it to CSV files. Once completed, the simulation data can be transferred back to the local machine, where it is processed. In the postprocessing phase, the Data Processing function extracts data from the CSV files and formats the data to include all scan positions. The ForceContour function subsequently calculates the surface heights for a given reference force. Finally, this is plotted in an AFM image using ContourPlot function. (See Appendix C for complete code).

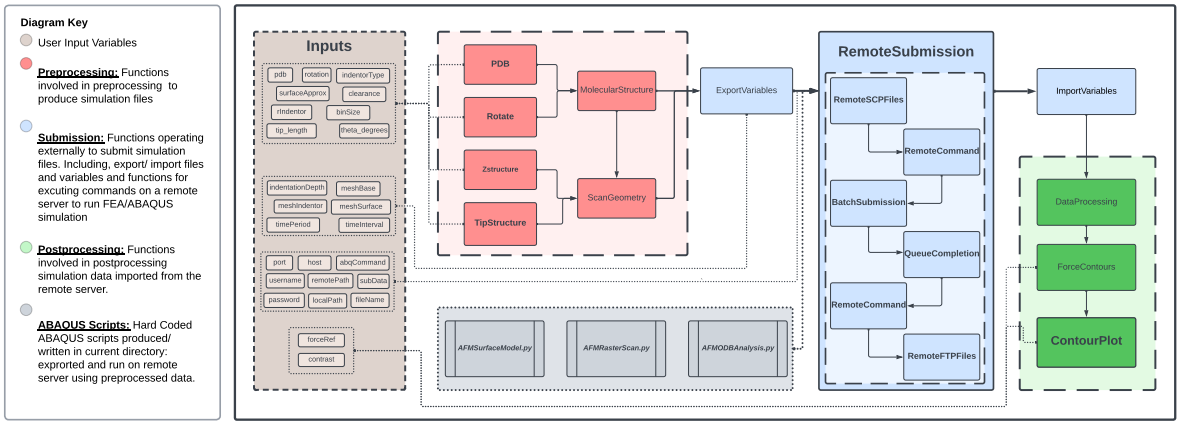


Figure 10: Flow diagram outline code functions and processing

2.4 Analysis of Surface Compression

2.4.1 Simulation Dynamics

This finite element approach can be extended to provide quantitative analysis of the indentation of surface features in AFM imaging. Of interest is the simulation of spherical and simple periodic structures. Within a natural setting, a hemisphere provides an analogy for various structures, and the deformation of a periodic structure provides a comparison for the analysis of DNA imaging. These structures were modelled as three-dimensional elastic parts in ABAQUS, with simulations focused on the compression produced from a single scan along the centre axis of the structures. Surfaces are assumed to be homogeneous and isotropic with a relative Young's modulus and Poisson ratio comparable to biomolecules as before. Indentations were simulated with a rigid, spherically capped conical indenter and "surface to surface" type contact. The contact was set as "hard", nonadhesive contact in the normal direction and "rough" Coulomb friction (non-slip) in the tangential direction. Boundary conditions fix the base of the structure, and vertical force and indentation data are mapped and sampled via reference points at the centre of the indenter. The simulations produced 2D force and indentation data over the central axis. The radial compression of spherical samples and the distortion in periodic structures are analysed as functions of the indentation force and the contact radius. The scan geometries are shown in Figure 11, and the following section details the techniques used to analyse the data. Example scripts are viewable in Appendix C.

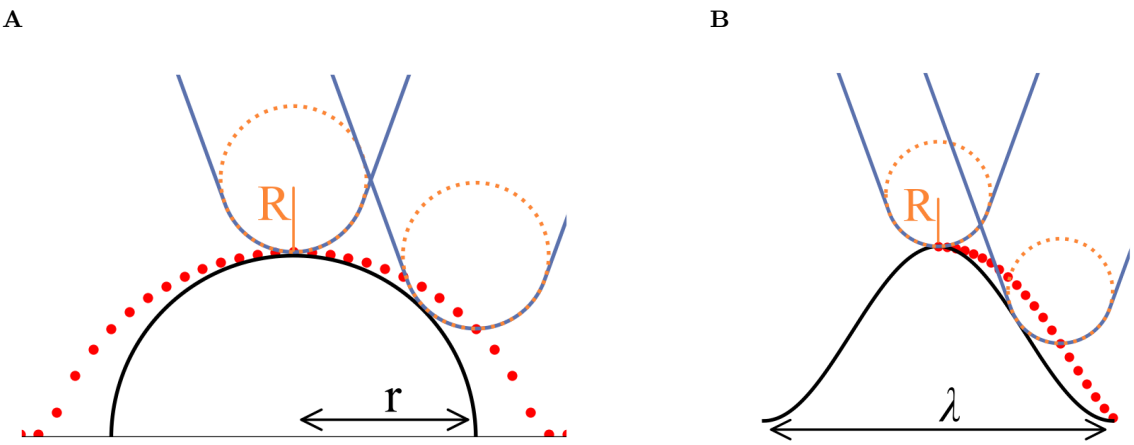


Figure 11: (A) Geometry of scan along the central axis of a hemisphere. Three-dimensional geometry is produced by rotating the indenter and semi-circle around the central z-axis. The hemisphere is shown in black with a radius r. Indenter geometry is shown in blue with a circular tip of radius R in orange. Red points indicate initial scan positions (Hard sphere contact points). (B) The geometry of the scan along the central axis of a plane wave structure for a half wavelength. Three-dimensional geometry is produced by rotating the indenter around the central z-axis and extruding the wave in the out-plane direction. Wave is shown in black with wavelength λ. Indenter geometry is shown in blue with a circular tip of radius R in orange. Red points indicate initial scan positions (Hard sphere contact points).

2.4.2 Heat Map and Force Contours

A heat map of the force over the scan domain was produced by processing data from indentation simulations. The heat map are produced by iterating over the course-grained x and z coordinates of the scan and mapping the corresponding force at each position into a 2D grid of the scan domain. This provides a grid of the force across a cross-section of the surface. A mask is used to exclude positions with no indentation data. Similarly, the force contours are calculated by iterating over each x coordinate and evaluating the force. The code finds the index for the value equal to the reference force and stores the corresponding XZ position. If the maximum force is below the threshold, the code masks that value in the contour array.

2.4.3 Full Width Half Maxima

Full-width half maximum (FWHM) measures the width of a peak in a spectrum at half of its maximum amplitude. FWHM characterise the resolution and peak shape of the contour. In Python, calculating the FWHM for the contour data is achieved by fitting a cubic spline to the contour data using the scipy.interpolate module. Once the spline is fitted, the FWHM can be calculated from the roots at half the maximum value. Finding the roots uses the UnivariateSpline.roots scipy function. This approach is advantageous because it provides a smooth representation of the data, which can help analyse noisy data sets.

2.4.4 Fourier Analysis

Fourier series express a periodic function as an infinite sum of sine and cosine waves, each modulated by an individual amplitude and frequency. Fitting data to a Fourier series allows for analysis of the spatial resolution of the contours produced. In Python, fitting data to a Fourier series is achieved using an explicit function for the series and scipy.optimise curve_fit function. Data used to fit the series was extrapolated from a tight spline of the raw contour data. This is used to ensure the smoothness of the fit and avoid overfitting due to a reduced number of data points. The curve_fit function returns the optimised coefficients and the covariance matrix. As the surface and contours are symmetric functions, Fourier analysis only requires the cosine terms given by,

$$F(x) \approx \sum_{n=0}^{\infty} A_n \cos\left(\frac{2\pi nt}{T}\right) \tag{1}$$

2.4.5 Volume Analysis

Volume analysis provides a quantitative metric for compression and distortion at varying indentation forces. Volume was calculated from splines fitted to the force contours using the Scipy UnivariateSplines class. This class provides an integral method that can be used to calculate the definite integral of the spline over a specified interval. The integral method uses numerical integration techniques to approximate the area under the curve, which can then be interpreted as the volume.

2.4.6 Youngs Modulus

The variation of fitted Young's Modulus over the scan positions illustrates the perceived elastic response of the surface. We use the same procedure used for contact models to fit the Hertz model in Python. Using the Scipy curve_fit function, the Hertz model is fitted for the indentation data at each scan position. Young's modulus is used as the fitting parameter—this returns the variation of fitted Young's modulus over the scan positions.

3 Results

3.1 FEM Verification of Contact Models

3.1.1 Elastic Half-space

The indentation of a spherical indenter into an elastic half-space is the simplest model to analyse; here, both the Hertz[33] and Dimitriadis models[34] are compared(Appendix B.1 and Appendix B.3). This is an important test, as the indentation in AFM experiments with blunt tips can be assumed to be spherical[37,38]. Similarly, analysing conical indenters into an elastic half-space is an important comparison for AFM experiments with sharp tips. Here the Sneddon model[27](Appendix B.4) is compared with the Hertz model. Moreover, the same analysis is applied to a spherically capped conical indenter. Illustrations of the indentation are shown in Figure 12**B** .

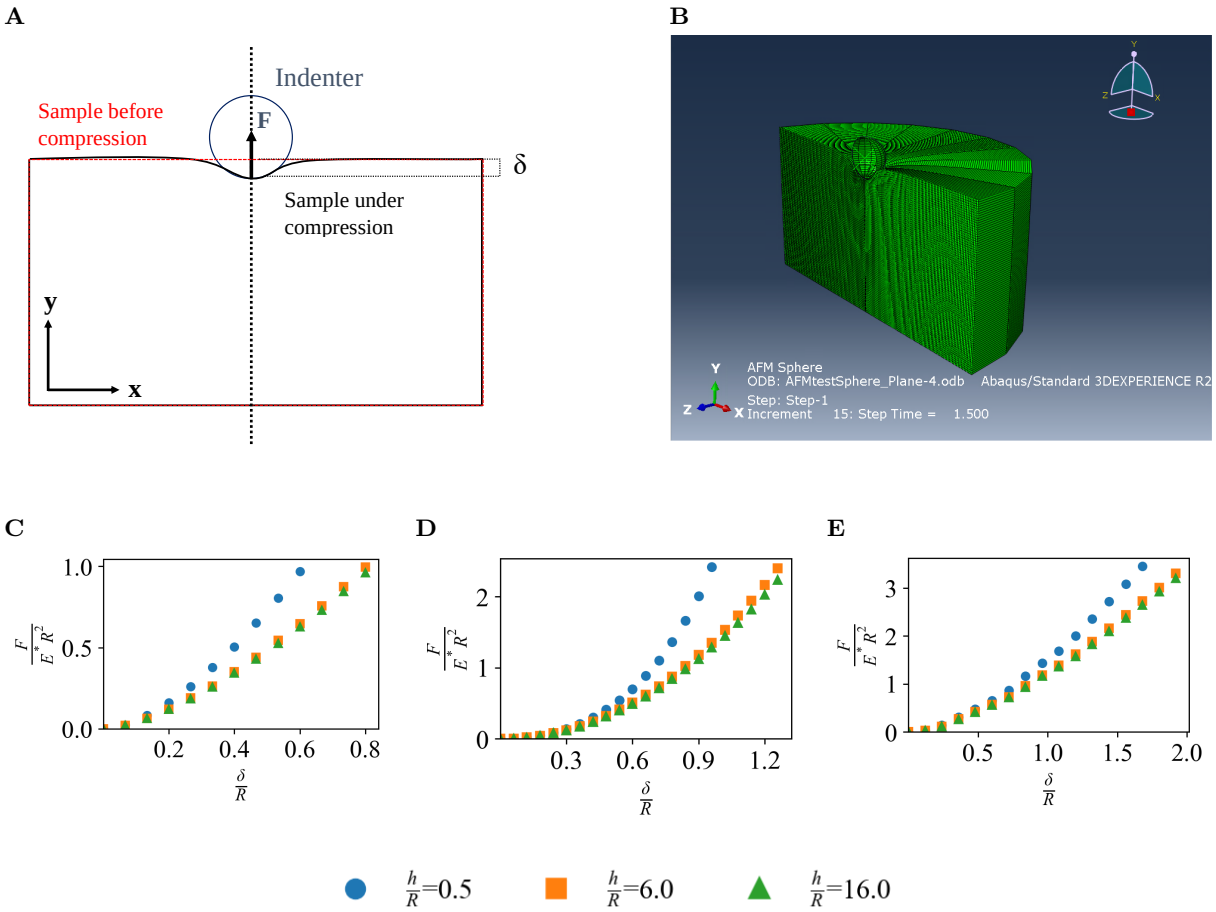


Figure 12: (A) Illustration of contact experienced by elastic half-space for indentation depth δ, and force, F. (B) Visualisation from GUI for the sphere-plane indentation where the asymmetric model is rotated 180 degrees around the central axis. (C)-(E) Force curves for indentation into an elastic half-space with varying plane depth and radius, h. Given in dimensionless units of force ($\frac{F}{E^* R^2}$) and indentation (δ/R). (C) Force curve for spherical indenter. (D) Force curve for conical indenter with half-angle of $\alpha = 60^o$. (E) Force curve for capped conical indenter with a half-angle of $\alpha = 20^o$. To normalise conical indenter data R is set as max contact radius $R = \delta_{max} \tan(\alpha)$.

As shown in Figure 12, these simulations gave the characteristic force curves for each indenter. For greater applicability, the simulation data is analysed in dimensionless units of force and relative indentation, which are scaled using the Hertz contact equation (given in Appendix B.1). The conical indenter shows the most significant curvature in its force curve. This is because contact forces are distributed more over the spherical surfaces than the sharper tip. Therefore,

the conical indenter produces more prominent deformation for the same force.

In comparison, the capped conical indenter produced a curve with transitional behaviour around $\delta/R = 1$, which is expected as the indenter moves from the spherical portion to the conical. Moreover, as the surface size (h) increases, the area bounded by the curves decreases. As the area under the curve represents the elastic energy of indentation, this behaviour illustrates the increased energy required to compress smaller planes. This is because the indention on smaller surfaces causes significant horizontal shear stress, and indention to the same depth cause more extensive compression. Therefore, this effectively produces greater stiffness and elastic energy.

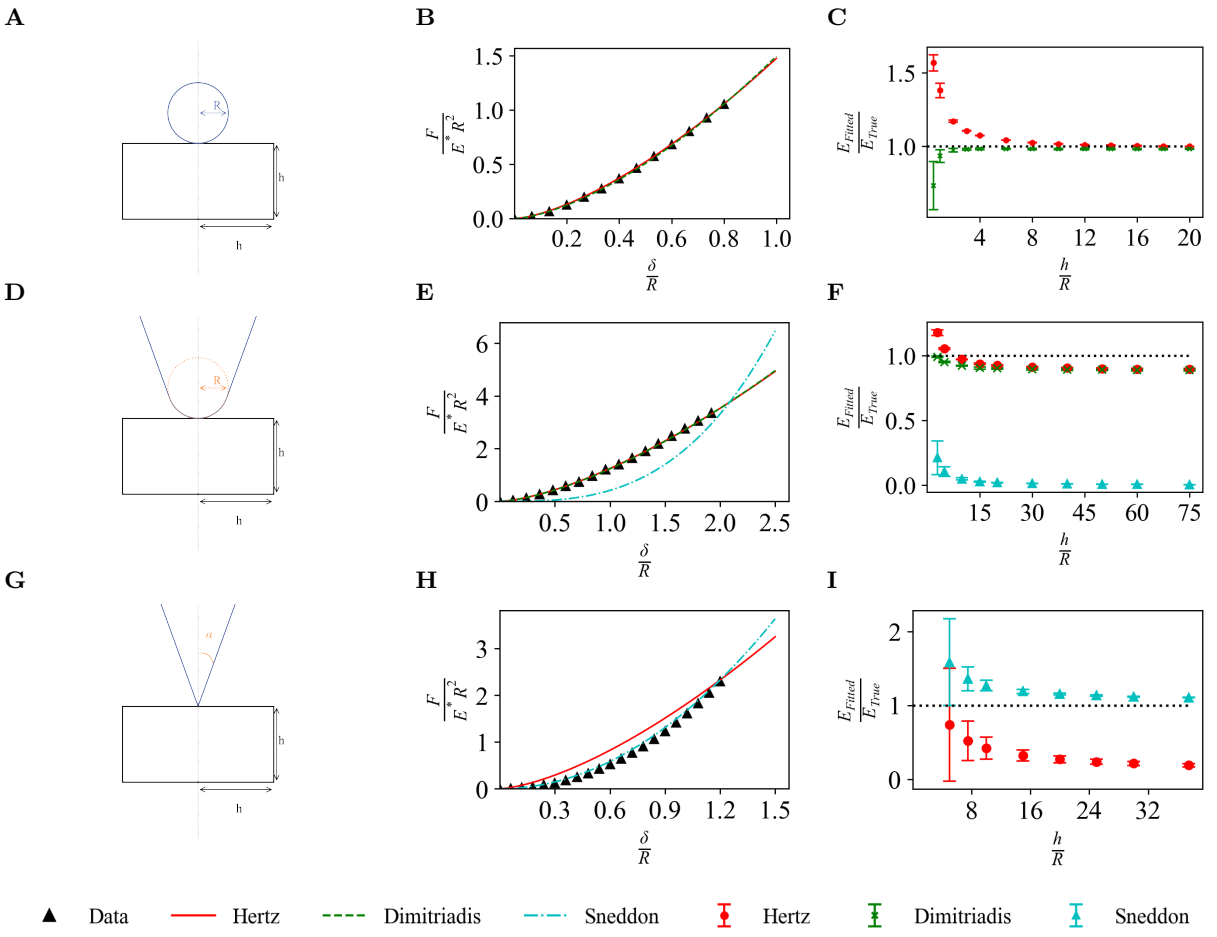


Figure 13: (A) Model assembly for spherical indentation of elastic half-space. (B) Plot for spherical indentation force curves fitted using the Hertz and Dimitriadis models. (C) Young's Modulus variation for spherical indentation into elastic half-space. (D) Model assembly for spherically-capped conical indentation of an elastic half-space. (E) Plot for capped indentation force curves fitted using the Hertz, Dimitriadis, and Sneddon models. (F) Young's Modulus variation for conical indentation into elastic half-space. (G) Model assembly for conical indentation of an elastic half-space. (H) Plot for conical indentation force curves fitted using the Hertz and Sneddon models. (I) Young's Modulus variation for conical indentation into elastic half-space. For conical indenter data normalisation $R = \delta_{max} \tan(\alpha)$.

Fitting the contact models to the simulated data for the spherical indenter, shown in Figure 13**B** , showed good adherence from both Hertz and Dimitriadis models. Moreover, comparing the fitted Young's modulus for increasing surface geometry, in Figure 13**C** , shows both models converged on the expected Young's modulus, confirming the accuracy of the simulation dynamics. However, there was a significant divergence from the predicted Young's modulus at small surface geometry and a greater error in the fit. This behaviour is expected as the indenter is a comparable size to the surface at smaller surface dimensions. Therefore, the force is distributed over a larger portion of the surface and can cause buckling and shear forces as the plane bulges out. In addition, this

can create larger reaction forces at the base. Consequently, this creates deviation from the theoretical models along with the contradiction of the assumption of an elastic half-space with an infinite horizontal/vertical extent.

Similarly, for the capped conical indenter, the Hertz and Dimitriadis model provided a much tighter fit to the data than the Sneddon model, which deviated as shown in Figure 13**E** . This indicates that the spherical behaviour dominates over the conical behaviour at these ranges. This qualitative behaviour is further reinforced by the variation of Young's modulus over surface depth/radius shown in Figure 13**F** . The Sneddon model converges on zero, indicating that the model diverges to a large extent and is a poor description of the indentation. In contrast, the Hertz and Dimitriadis models converged closely on the expected Young's modulus. Some deviation is expected as the models are for a pure spherical indenter, whereas the conical section of the capped indenter causes variation in the contact.

Lastly, the Sneddon model produced an expected tight fit for the conical indenter, whereas Hertz was a poor fit. As expected, the Hertz model failed to converge upon the correct Young's Modulus and produced a large underfit because the model is for smaller spherical deformation. However, the Sneddon model converges to the true value. These results all provided the expected behaviours and confirmed the accuracy of ABAQUS. This provides the basis to be applied to simulations of AFM imaging.

3.1.2 Elastic Sphere

The analysis of the elastic sphere is more complex, as it necessitates accounting for the indentation between the indenter and the surface and the indentation between the surface and the base. As shown in Figure 14**C** , the simulations give the characteristic force curve for the indenter and, as before, as the radius of the surface increases, the bounded area decreases. However, from Figure 14**D** , it can be seen that sampling data at the base of the sphere shows the surface is compressed. However, due to the extremely small indentation, the Hertzian behaviour closely approximates a linear relationship at the base, exhibiting a response consistent with Hooke's Law.

Moreover, Figure 14 shows the indentation of the elastic sphere into the base. Indentation force is distributed between the reaction at the base and the indenter, and part of the perceived indentation depth is due to compression at the base. Therefore, corrected values for the indentation depth is calculated by subtracting the surface compression at its base from the indenter's displacement. Similarly, the corrected forces are the sum of the reaction forces. However, as this research focuses on AFM imaging, the fitted force and indentation data are produced from the indenter data alone, and compression is accounted for using "Double Contact" models[35,36](Appendix B.5).

For these simulations, the radius of the elastic sphere is varied to test the model's accuracy across varying curvature. Figure 15**B** shows the difference in the corrected and indenter data. Fitting the contact models to the spherical indenter's data produced tight adherence from the Hertz, Dimitriadis, and Double Contact models. Similarly, for the capped conical indenter, shown in Figure 15**E** , the Hertz, Sneddon, and Double Contact models produced tight fits. In contrast, the Hertz model gave a poor fit for the conical indenter, while the Sneddon and Double Contact models produced tight fits.

Comparing the fitted Young's modulus variation over surface radius produced various convergent behaviours. The results indicate that the simple Hertzian model underestimates the elastic modulus as the data converged below the expected value for all simple contact models. In contrast, the double contact models converge to the actual value. This highlights the importance of the contribution of compression at the base of the sample. Considering only the indenter data reduces the force experienced by the indenter, reducing fitted Young's modulus.

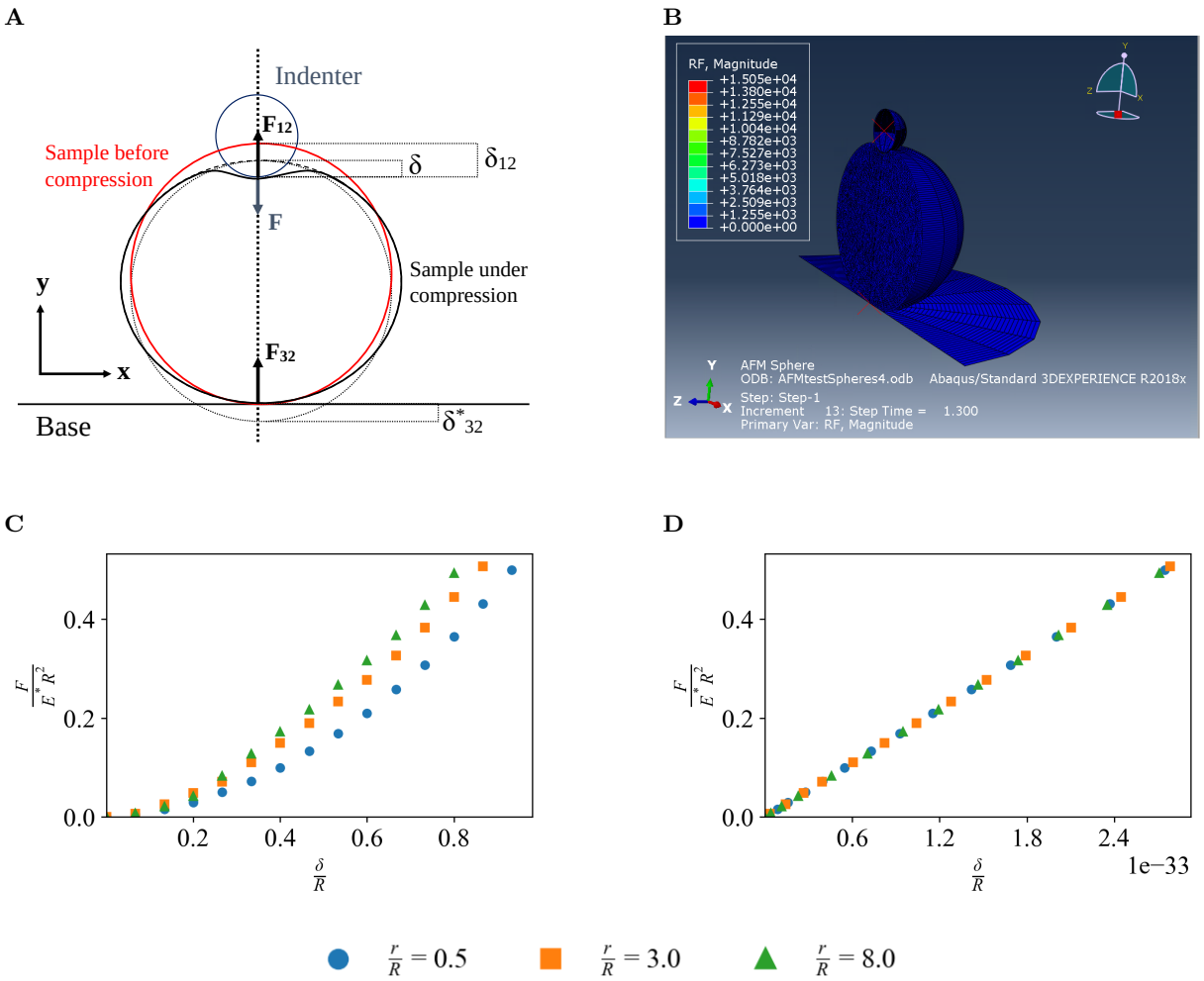


Figure 14: (A) Illustration of double contact experienced by spherical sample. For indentation δ, indenter displacement δ_{12}, surface compression δ^*_{32}, indentation force F, indenter reaction force F_{12}, and base reaction force F_{32}. (B) GUI visualisation for the sphere-sphere indentation where the asymmetric model is rotated 180 degrees around the central axis. (C) Force curve for spherical indentation δ/R into the elastic sphere of varying radius, r/R. (D) Force curve for compression of the surface into the base. Conical and capped indenter data is not shown, however, shows similar trends as before.

As shown in Figure 15**C** , the double contact models converged on the expected value for a spherical indenter. However, the error in the fit fluctuates as the surface radius increases. At small surface radii, excessive amounts of compression cause significant error and deviation from the theoretical models. The error decreases as the radius of the surface increases and compressive effects lessen; however, the error increases again at large surface radii as the geometry approximates an elastic half-space, and the double contact model is no longer valid. In contrast, errors in the Hertz and Dimitridias models decrease as the surface curvature decreases and the surface approximates an elastic half-space.

The Hertz model converged close to zero for the conical indenter shown in Figure 15**I** . This indicates that the model diverges to large extents, which is expected as the Hertz model is for spherical indenters and an elastic half-space. On the other hand, the Sneddon model for the conical indenter converged to around 0.5. As discussed before, this is due to the compression of the sample, which reduced the reaction force at the tip. In contrast, applying a novel formulation of the Double Contact model for conical indenters (shown in Appendix B.7) produces a tight fit to the actual value across all surface radii.

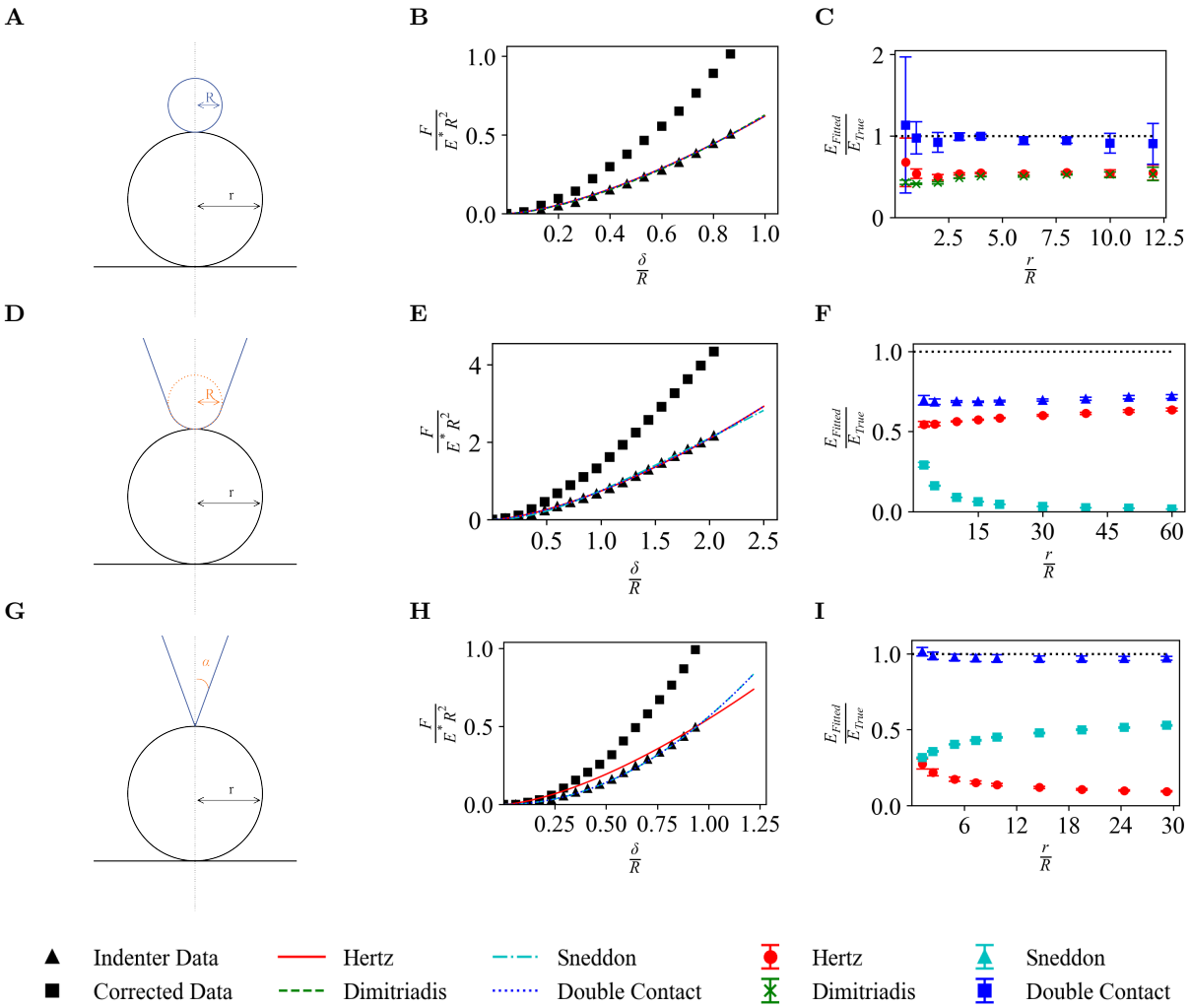


Figure 15: (A) Model assembly for spherical indentation of the elastic sphere. (B) Plot for spherical indentation force curves fitted using the Hertz, Dimitridias, and Double Contact models. (C) Youngs Modulus variation for spherical indentation into an elastic sphere. (D) Model assembly for spherically-capped conical indentation of an elastic sphere. (E) Plot for capped indentation force curves fitted using the Hertz, Sneddon, and Double Contact models. (F) Youngs Modulus variation for capped conical indentation into an elastic sphere. (G) Model assembly for conical indentation of an elastic sphere. (H) Plot for conical indentation force curves fitted using the Hertz, Sneddon, and Double Contact models. (I) Youngs Modulus variation for conical indentation into an elastic sphere. For conical indenter data normalisation, $R = \delta_{max} \tan(\alpha)$. Conical and capped indenters have the same dimensions as used in half-space simulations.

In comparison, for the capped indenter, the Double Contact model converged around 0.75. This offset is expected because the non-spherical portion of the indenter produces less curvature and deviation from the theoretical model. Similarly, the deviation produced by compression at the base leads to a similar convergence in the Hertz model. In comparison, the Sneddon model shows poor fit as it converges to 0. This indicates that the spherical indentation is dominant at these depths. Overall, these results further validated our ABAQUS modelling.

3.2 FEM Applied to Compression Simulation

The finite element approach presented in this research can be extended to analyse the compression of surfaces in AFM imaging. Spherical and periodic structures provide the simplest structures to consider. These structures were modelled as three-dimensional, homogeneous elastic parts with elastic modulus and Poisson ratio comparable to biomolecules. The simulations focused on the compression produced when scanning along the centre axis of the structures, generating 2D force and indentation data over the restricted central axis. Subsequently, the radial compression of spherical samples and distortion in periodic structures are analysed as a function of the indentation force. This elucidates the responses and characteristics produced in the surfaces under AFM imaging and the apparent appearance of the structure.

3.2.1 Analysis of Hemisphere Structure

As presented in Figure 17, the hemisphere simulations provided 2D heat maps of the indention force across a surface cross-section. This displays the force experienced at each depth along the scan line. The raw data from the simulation, shown in 16**A** , produces a course-grained distribution of indentation force over the scan. Using interpolation methods detailed in Section 2.4, the data can be smoothed to produce a continuous distribution shown in Figure 16**B** . From the raw data, contours of constant force can be extracted to analyse the perceived shape at varying forces as presented in Figure 16**C** .

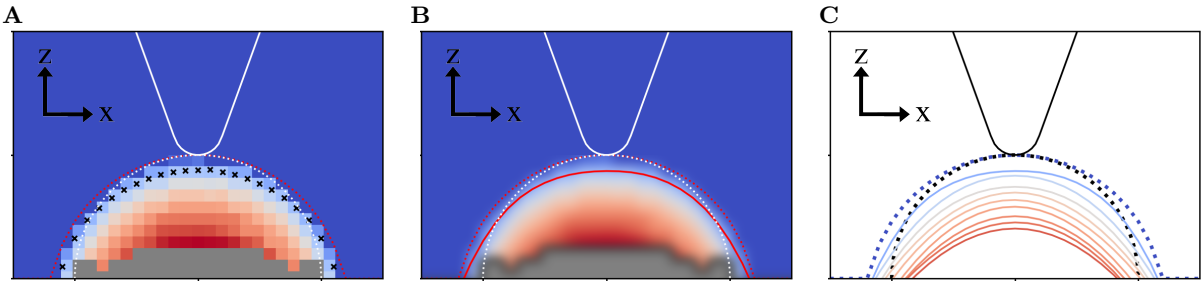


Figure 16: Illustrations of data produced from compression simulations for indenter $R/r = 0.2$. (A) Raw two-dimensional heat map of indentation force over the scanning axis for the hemisphere structure. Including the points of constant force, $\frac{F}{E^* R^2} = 0.227$, shown as black crosses. These are points used to produce the force contours. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (B) Interpolated two-dimensional heat map of indentation force over the scanning axis for the hemisphere structure. Including overlayed contour of constant force, $\frac{F}{E^* R^2} = 0.227$, shown in solid red. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (C) Two-dimensional plot of force contours for varying indentation/ reference forces over the scanning axis of the hemisphere structure. Force shown varies within the limit of $0.227 < \frac{F}{E^* R^2} < 3.867$. The indenter and surface are black, with the initial zero force/ hard sphere boundary in dashed blue.

Analysis of the compression, shown in Figure 17**B** and 17**C** , revealed that increasing the indenter-to-surface ratio results in lower indentation forces. As the surface-indenter ratio/contact radius increases, the forces are spread over a larger area, resulting in less deformation for the same force. Combined with greater tip convolution, this increases the broadening of the surface's appearance. This is reflected in fitted Young's modulus as shown in Figure 17**D** . As indentation force, and therefore fitted Young's modulus, varies as a function of the contact radius, the variation in Young's modulus over the scan position emulates the scan geometry. Therefore, larger indenters produce a flatter fitted Young's modulus. This highlights that increasing the indenter-surface ratio is analogous to scanning a surface with greater Young's modulus. Physically, this corresponds to greater effective stiffness experienced at larger indenter-surface ratios, as forces are distributed over a larger surface area and larger forces are required for equivalent compression. Interestingly, due to the flattening produced, larger indenter-surface ratios provide

a greater estimation of the true Young's modulus over the scan positions.

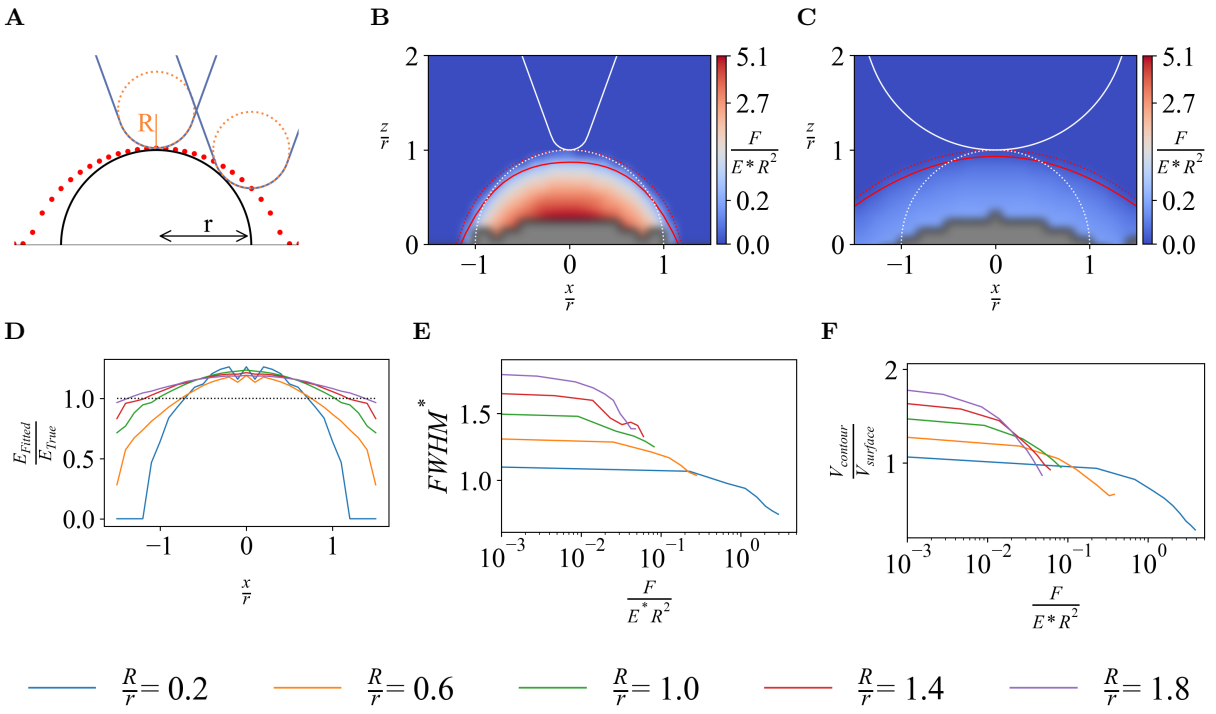


Figure 17: (A) Geometry of scan along the central axis of a hemisphere. Three-dimensional geometry is produced by rotating the indenter and semi-circle around the central z-axis. The hemisphere is shown in black with a radius r. Indenter geometry is shown in blue, with the circular tip of radius R in orange. Red points indicate initial scan positions (Hard sphere contact points). (B) Interpolated two-dimensional heat map of indentation force over the scanning axis for hemisphere structure with indenter $R/r = 0.2$. Including overlayed contour of constant force, $\frac{F}{E^* R^2} = 0.227$, shown in solid red. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (C) Interpolated two-dimensional heat map of indentation force over the scanning axis for hemisphere structure with indenter $R/r = 1.4$. Including overlayed contour of constant force, $\frac{F}{E^* R^2} = 0.227$, shown in solid red. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (D) Fitted Young's modulus over scan positions for each indenter radius (R/r). (E) Relative FWHM of the contour divided by FWHM of true geometry ($FWHM^* = \frac{FWHM}{FWHM_{True}}$) variation over contour force for each indenter radius(R/r). (F) Volume variation over contour force in spherical structures for each indenter radius(R/r).

The Full Width Half Maxima (FWHM) provides information about the spread of the contour. The FWHM value reflects the surface's compression and is used as an indicator flattening of the contour. As shown in Figure 17**E** , FWHM decreases asymptotically as the indentation force decreases. As the force decreases to zero, each indenter converges on the corresponding FWHM of the hard sphere boundary. Conversely, as the force increases, the sample is indented to greater depth producing tighter, compressed force contours and narrowing the surface's appearance. For larger indenters, this produces force contours with FWHM closer to the true FWHM. However, this does not directly show that the higher force provides a more accurate recreation of the sample surface. The compression of the sample can produce elliptical contours that do not represent the hemisphere yet share the same FWHM. For a smaller indenter, which approximates the surface geometry more closely at low force, the increased compression at higher forces produces an FWHM that diverges from the true surface FWHM. Furthermore, the nonlinear behaviour of the FWHM highlights that the compression producing the force contours is not a simple linear relationship.

Furthermore, the apparent volume can quantify compression over various contour forces. As shown in Figure 17**F** , the volume decreases exponentially as the indentation force increases. This further validates the behaviour shown by the FWHM. As the indentation force increases,

greater compression is produced, reducing the apparent appearance. In addition, as indenter size decreases, larger forces are required to achieve the same relative decrease in volume. This is expected as smaller indenters have smaller contact areas; therefore, larger forces are required to compress the sample to the same extent.

Overall, this analysis highlights three main characteristics of the hemisphere under compression. First, the Youngs modulus shows the dependence of elastic response/behaviour on the contact radius and tip convolution. The FWHM is an important analysis to demonstrate the nonlinear compression and widening of the perceived surface. Finally, the volume highlights that the larger indenters/contact areas require larger forces to compress the sample to the same extent as smaller indenters.

3.2.2 Analysis of a Simple Periodic Structure

Expanding upon the simulation of a hemisphere, next, we analyse the deformation of a periodic structure that provides a comparison for the analysis of DNA imaging. As presented in Figure 18, the simulations produced 2D heat maps of the indention force across the cross-section of the surface. Comparison of cross sections for different indenter ratios, shown in Figure 19**B** and Figure 19**C** , shows that increasing indenter-to-surface ratios produces lower indentation forces. As with the hemisphere, due to the large contact radius, the forces are distributed over a large area; consequently, the compression is smaller for the same force. Moreover, evaluating the variation of the fitted Young's modulus over the scan positions, shown in Figure 19**D** , highlights the relation between elastic response/ force and the contact area. Unlike for the hemisphere, the dependence on the tip-surface convolution produces an inverse relationship in Young's modulus at larger surface-tip ratios. At the surface trough, the indenter has a maximum contact radius and the maximum surface interaction. Consequently, the indenter experiences a larger effective stiffness and Young's modulus. However, the contact radius decreases for scans away from wave trough, and Young's modulus decreases proportionally.

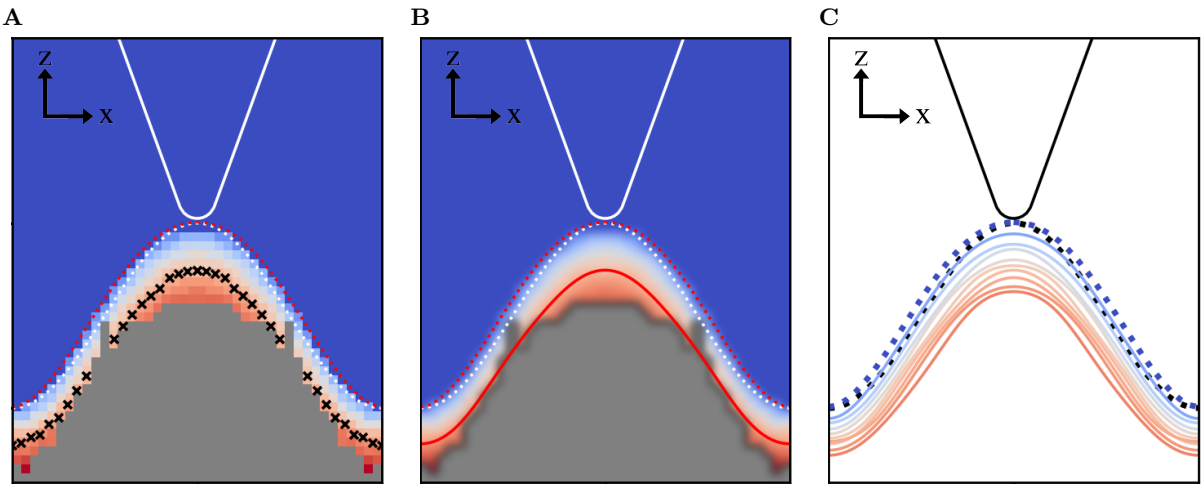


Figure 18: Illustrations of data produced from compression simulations for indenter $R/\lambda = 0.05$. (A) Raw two-dimensional heat map of indentation force over the scanning axis for the periodic structure. Including the point of constant force, $\frac{F}{E^* R^2} = 0.227$, shown as black crosses. These are points used to produce the force contours. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red.(B) Interpolated two-dimensional heat map of indentation force over the scanning axis for the periodic structure. Including overlayed contour of constant force, $\frac{F}{E^* R^2} = 0.227$, shown in solid red. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (C) Two-dimensional plot of force contours for varying indentation/ reference forces over the scanning axis of the periodic structure. Force shown varies within the limit of $0.227 < \frac{F}{E^* R^2} < 3.867$. The indenter and surface are black with initial zero force/ hard sphere boundary in dashed blue.

For greater quantitative analysis of the compression in the simulation, both Fourier analysis

and FWHM are employed. Fitting the contour points using a Fourier series allows us to break down the oscillatory signature of the simulated surface appearance. Furthermore, the individual contribution can highlight the deviation and behaviour of the surface when indented. The Fourier components are shown in Figure 20**A** . The zeroth component of the Fourier series represents a linear vertical offset. The increasing trend corresponds to an increased trough height for larger indenter-surface ratios. The first component corresponds to surface periodicity. Only this component is expected for the contour that matches the geometry of the surface, as shown by the black bar in Figure 20**A** . The decreasing amplitude of this component corresponds to an increase in surface distortion, as the amplitude is shared proportionally with higher-order terms. The second component is the significant component producing the widening wave peak, and higher-order terms refine the curvature of the contour.

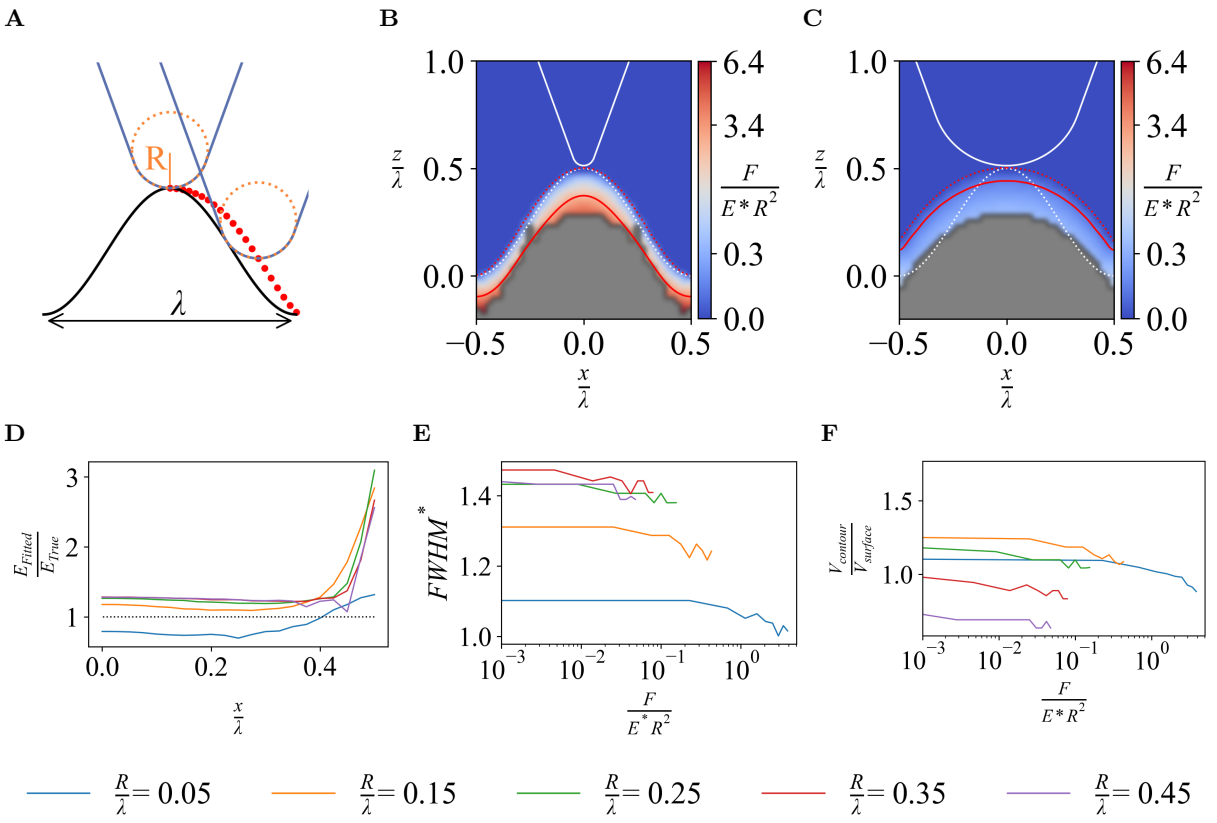


Figure 19: (A) Geometry of scan along the central axis of a hemisphere. Three-dimensional geometry is produced by rotating the indenter and extruding the wave. Wave is shown in black with wavelength λ. Indenter geometry is shown in blue with a circular tip of radius R in orange. Red points indicate initial scan positions (Hard sphere contact points). (B) Interpolated two-dimensional heat map of indentation force over the scanning axis for periodic structure with indenter $R/\lambda = 0.05$. Including overlayed contour of constant force, $\frac{F}{E^*R^2} = 0.227$, shown in solid red. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (C) Interpolated two-dimensional heat map of indentation force over the scanning axis for periodic structure with indenter $R/\lambda = 0.45$. Including overlayed contour of constant force, $\frac{F}{E^*R^2} = 0.227$, shown in solid red. The indenter is solid white, and the surface is dotted white. Points of zero force/ hard sphere contact is shown in dotted red. (D) Fitted Young's modulus over scan positions for each indenter radius (R/λ). (E) Relative FWHM of the contour divided by FWHM of true geometry ($FWHM^* = \frac{FWHM}{FWHM_{True}}$) variation over contour force for each indenter radius(R/λ). (F) Volume variation over contour force in spherical structures for each indenter radius(R/λ).

Therefore, the deviation from the true surface geometry and accuracy of the imaging can be quantified by analysing the variation of the first Fourier component over a range of contour forces. Analysing the absolute A1 for each contour force, shown in 20**B** , indicates how well we extract information from the surface topography. A lower indenter-surface ratio produces

less apparent structure deviation as the scan/tip more closely follows the surface geometry. Moreover, A1 is generally constant over the range of contour forces with only some decrease in the component for the smaller indenter at larger forces.

However, in contrast, the relative component A_1^* (i.e. individually normalised series where $A_1 + A_2 + A_3 + ... = 1$) shows that although the absolute value of the first component may be constant throughout the forces range, the actual percentage of the series that the first component represents does vary. As shown in Figure 20**C** , when contour force increases, a greater percentage of the periodicity of the surface is recovered. This indicates that the apparent resolution extracted from the AFM image increases with indentation force. This could be due to the increased proportion of the indenter in contact with the surface and the elastic behaviour becoming more linear for deeper indentations. As a result, the indentations are subject to more influence from the surfaces. Therefore, variations in the surface geometry are better resolved in the contour at a higher force.

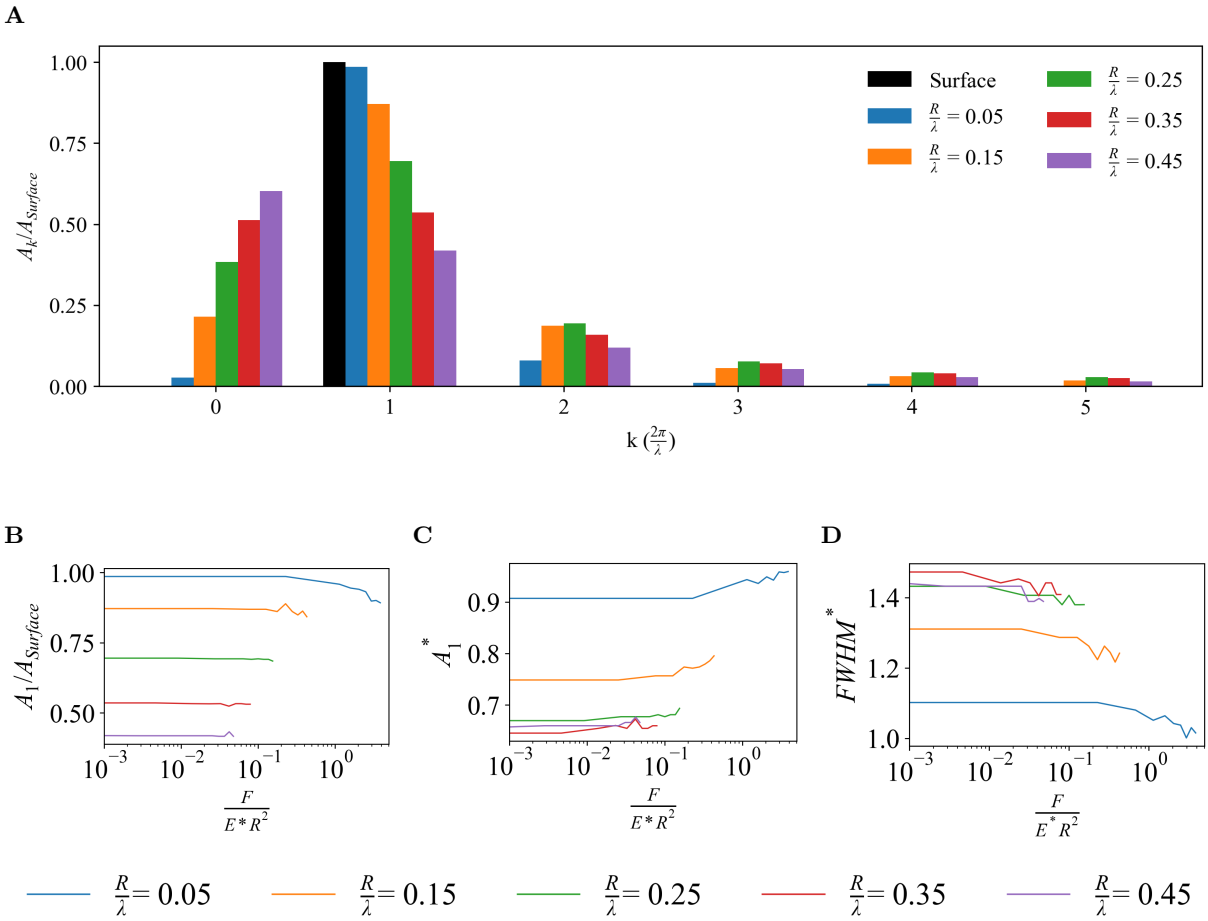


Figure 20: Analysis of force contours for periodic structure. (A) Fourier Series Component for force contours at $\frac{F}{E*R^2} = 0.227$. (B) Variation of absolute component A_1 (Normalised by true surface amplitude/Fourier component) over contour force for a range of indenters ($\frac{R}{\lambda}$). (C) Variation of relative component A_1^* (A_1 coeffiecent normalised by series terms - $A_1^* = \sum_{k\neq 0} A_k$) over contour force for range of indenters ($\frac{R}{\lambda}$). (D) Relative FWHM of the contour divided by FWHM of true geometry ($FWHM^* = \frac{FWHM}{FWHM_{Surface}}$) variation over contour force for each indenter radius($\frac{R}{\lambda}$).

This behaviour is supported by the FWHM shown in Figure 17**E** . As the indentation force increases, FWHM decreases asymptotically, similarly to the absolute component $A1$, which represents the compression of the sample. This produces contours with FWHM closer to the true FWHM. Comparing this with the trends highlighted in Fourier analysis further reinforces that larger forces enable better resolution of the underlying structure. The initial constant trend of the FWHM may also show that at low forces, the compression around the midpoint of the

wave is low, and the indenter requires a large force to indent the slope region.

As shown in Figure 17**F** , the volume variation over indentation force is less prominent than for the hemisphere. For each indenter, there is only a slight reduction in volume as the indentation force increases. Moreover, unlike the hemisphere, the apparent volume is not directly proportional to the indenter's radius. Although a larger indenter produces greater tip convolution, which widens the peak, as volume is measured from the peak to the trough, the composite reduction in wave amplitude and depth results in a smaller volume overall as the indenter radius increases from $R/\lambda = 0.15$. In contrast, $R/\lambda = 0.15$ produces force contours closer to the topology of the surfaces, which is reflected in the apparent volume.

Overall, these analyses highlight similar characteristics to the hemisphere under compression. Young's modulus shows the dependence of elastic response/behaviour on the contact radius and tip convolution. Similarly, the volume highlights that the larger indenters/contact areas require larger forces to compress the sample to the same extent as smaller indenters. However, the FWHM and Fourier analysis elucidate a possible novel feature. The Fourier analysis demonstrates that larger indentation forces recover more surface periodicity.

3.3 FEM Applied to AFM Image Simulation of B-DNA

Finally, the FEM approach was applied to biomolecules to produce simulations of AFM imaging of DNA strands. As shown in Figure 21, we produced simulated images of a B-DNA DODECAMER/ DNA strand (PDB: 1bna). As detailed in the methodology, the biological surfaces were produced using a PDB file specifying the atoms' coordinates in the molecule. The simulation modelled 598 individual atoms using the van der Waals radius of the atoms. The biomolecule was produced in ABAQUS as an assembly of spheres for individual atoms. Atoms were merged to produce a composite sphere model of the biomolecule. The structure was modelled as a homogeneous elastic material with Young's modulus and Poisson ratio (1000 kPa and 0.3, respectively). The molecule was partially embedded in a rigid base/ substrate with the bottom 20% cleaved and fixed at the base using boundary conditions. The scan positions were then calculated in the base domain. The base itself was modelled as a rigid cube with width 52 Å and length 76 Å divided into bins of 4 Å producing 108 individual scan positions/ ABAQUS simulations. The AFM probe tip was modelled as a capped conical indenter, which was rigid and incompressible. The indenter had a radius of 4 Å and a conical half-angle 5^o. The dynamics of an AFM raster scan were achieved by performing a set of individual indentation simulations across a surface domain. The AFM image was evaluated for a contour force of 0.1 pN. The code also allows hard-sphere model images to be calculated and produced along with the FEM shown in Figure 22**B** .

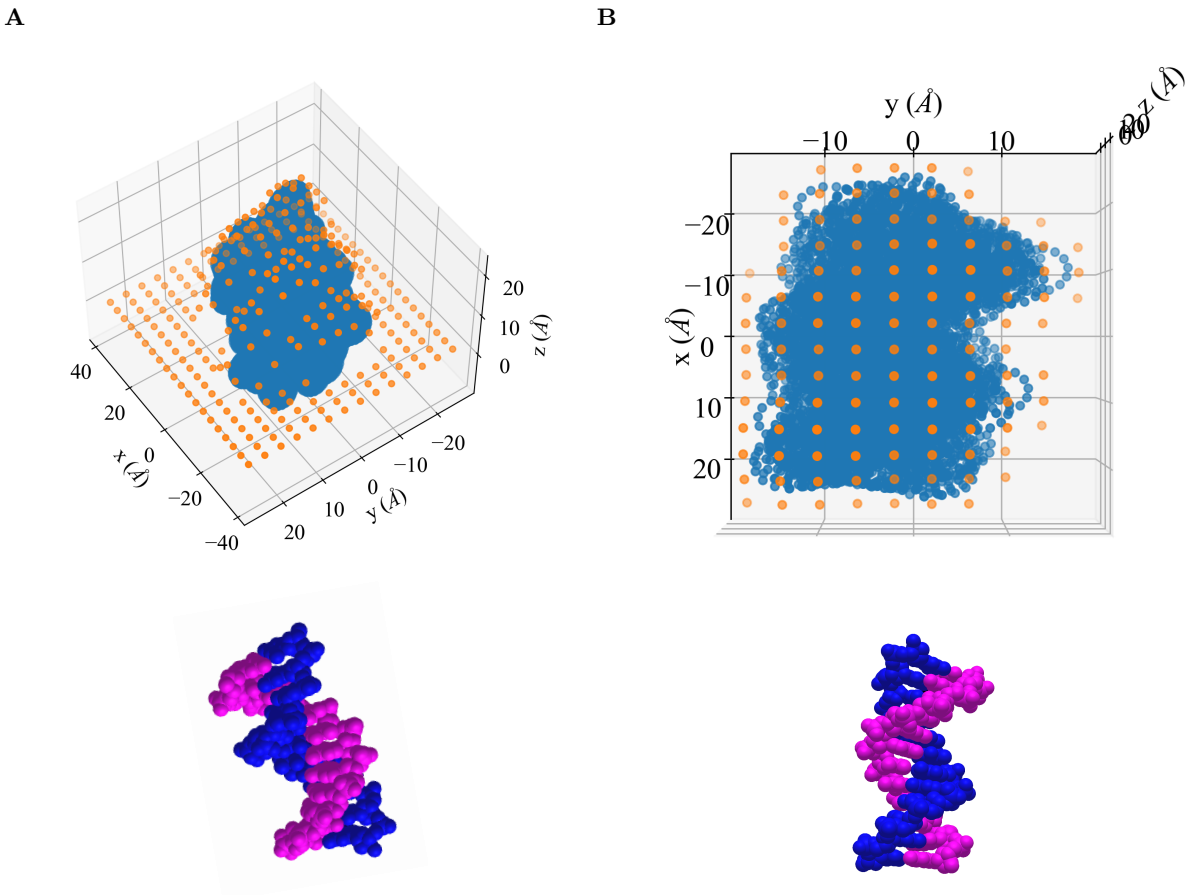


Figure 21: Illustration of biomolecule (blue) and scan positions for simulation (orange). Sphere model of PDB below each to aid in visualisation. (A) Side view of the assembly with all scan positions. (B) Top view of the assembly with clipped scan points (only points where tip interacts)

As presented in Figure 22**A** , our simulation has reproduced the expected appearance of the DNA double helix. This provides an example of the viability of the FEM approach to AFM imaging. We see distinct differences in the simulations produced with ABAQUS compared to the hard-

sphere model in Figure 22**B** . The indentation produces with FEM shows greater sensitivity for protruding features, whereas the deeper structure requires greater indentation/force to recover the surface topology. Similarly, simulating another orientation of the B-DNA, shown in Figure 22**C** , revealed the utility of the FEM approach. The base for these simulations was divided into bins of 10 Å producing 38 individual scan positions/ ABAQUS simulations. The AFM probe tip was modelled with a radius of 10 Å and a conical half-angle 10^o. The FEM simulation reproduced more underlying helix structure than the hard-sphere model. The hard sphere model for this simulation gave a blurred image of the DNA as the large tip distorts the surface. The surface indentation produced by FEM appears to recover elements of the underlying structure. However, some of these variations originate from ABAQUS not performing calculations at the points of complex deformation. Within the code, these points are set to the maximum indentation depth so appear artificially deeper. This could pose challenges in repeatability.

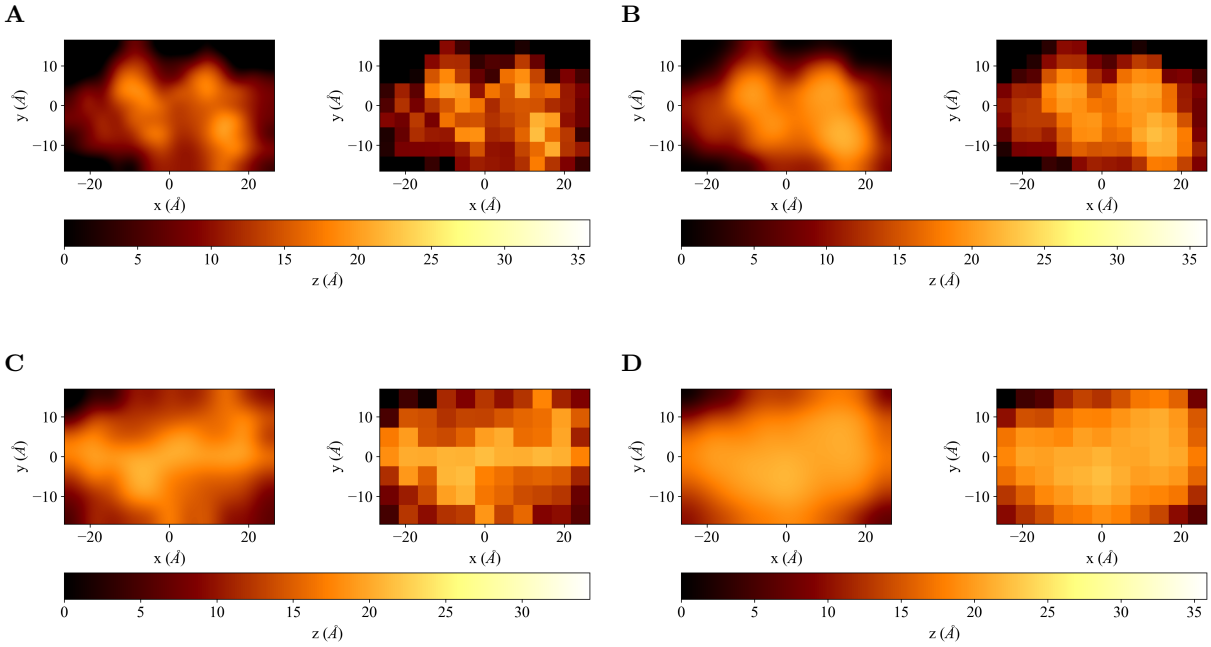


Figure 22: AFM images of a B-DNA DODECAMER with interpolated (left) and raw image(right) for each. (A) FEM Simulation with a tip radius of 4Å and spatial resolution of 4Å per pixel. (B) Hard sphere simulation with a tip radius of 4Å and spatial resolution of 4Å per pixel. (C) FEM simulation with a tip radius of $1nm$ and spatial resolution of $1nm$ per pixel.(D) Hard sphere simulation with a tip radius of $1nm$ and spatial resolution of $1nm$ per pixel.

4 Conclusion and Discussion

4.1 Conclusion

In conclusion, our FEM approach has demonstrated some novel and varied applications for the analysis of AFM imaging. Our analysis of the contact models for elastic half-spaces and spheres agreed with the theoretical models. Most prominently, the results highlight the under-fitting of the elastic modulus produced by simple Hertzian models for spherical samples. Moreover, our novel formulation of the Double Contact model for conical indenters demonstrated good predictive power over a range of surface radii.

Moreover, applying FEM to analyse the compression of hemispheres and simple periodic surfaces highlighted the quantitative power of this approach. These simulations highlighted the dependency of the elastic behaviour on the contact radius and tip convolution. Our results indicated that larger indenters require larger forces to compress the sample to the same extent. In addition, Fourier analysis of the simulated AFM contours elucidated a possible novel trend that larger indentation forces recover more of a surface's periodicity.

Finally, applying FEM to simulate the AFM appearance of B-DNA Dodecamer provides promising results. The code provides a range of parameters to allow simulations to emulate real AFM system specifications. These initial simulations showed the viability of this modelling and provided various extensions to be explored. However, this has shown some limitations to this approach, including the simulation times and scalability. For example, these simulations took around 48-72 hours for around 40 scan positions and one week for the 108 scan positions. These time scales limit the effectiveness of this approach for larger biomolecules. In addition, due to the numerical methods used by ABAQUS, deformation is limited and complex geometry simulations can often fail. This affects the repeatability and reliability of the simulation.

4.2 Discussion

These simulations have provided a good proof of concept, but large biomolecules would allow for better experimental comparison. Of interest would be the analysis of DNA compression during imaging and comparing it to experimental data. Furthermore, the analysis of human PARP-1 could add to current experimental efforts in the Hoogenboom lab. However, these require us to tackle the limitations discussed with scalability and time. Figure 23 shows the simulated hard sphere model for these two molecules to illustrate the scale. Greater computational resources may help mitigate the simulation time, and perhaps a courser grain may be required. This could help improve the repeatability issue produced by failed ABAQUS simulations caused by complex geometry.

Furthermore, an immediate improvement could involve simulations that account for biomolecule density, viscoelastic properties, and gravity. This could include modelling large biomolecules with multiple materials and properties and including adhesion in the model. In addition, introducing simulation of measurement and environmental errors, such as parachuting, noise, and thermal drift, could provide greater experimental applicability. Expanding the simulation to include in-situ conditions, such as liquid simulation, would provide an interesting extension in future work. This could be achieved using Abaqus/CFD, a Computational Fluid Dynamics software application which provides advanced computational fluid dynamics capabilities. In addition, Coulomb attraction of molecules can be included in ABAQUS.

Another extension could involve dynamic scans accounting for feedback and indentations based on threshold force. Using Abaqus/Explicit, more complex contact under transient loads can be modelled to include continuous raster scan such that change to a molecule in one indentation is carried over.

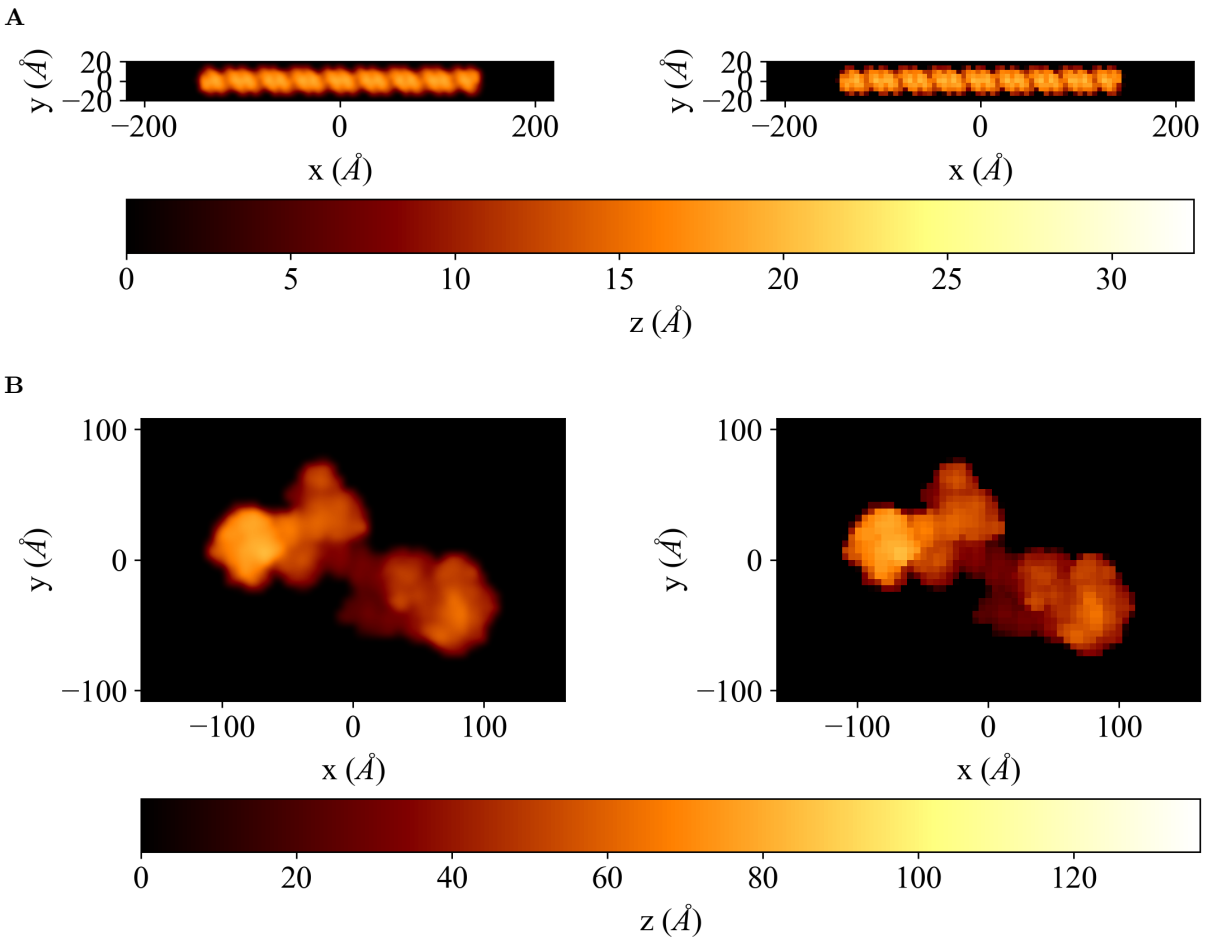


Figure 23: Hard sphere model simulations of AFM image (A) DNA strand (custom pdb). (B) Human PARP-1 (PDB:4dqy).

References

1. Nagashima, N., Matsuoka, S. & Miyahara, K. Nanoscopic hardness measurement by atomic force microscope. *JSME international journal. Ser. A, Mechanics and material engineering* **39,** 456–462 (1996).
2. Kempf, M., Göken, M. & Vehoff, H. Nanohardness measurements for studying local mechanical properties of metals. *Applied Physics A: Materials Science & Processing* **66** (1998).
3. Göken, M. & Kempf, M. Microstructural properties of superalloys investigated by nanoindentations in an atomic force microscope. *Acta Materialia* **47,** 1043–1052 (1999).
4. Yamamoto, A., Watanabe, A., Tsubakino, H. & Fukumoto, S. AFM observations of microstructures of deposited magnesium on magnesium alloys. *Materials science forum* **350,** 241–246 (2000).
5. Jalili, N. & Laxminarayana, K. A review of atomic force microscopy imaging systems: application to molecular metrology and biological sciences. *Mechatronics* **14,** 907–945 (2004).
6. Santos, N. C. & Castanho, M. A. An overview of the biophysical applications of atomic force microscopy. *Biophysical Chemistry* **107,** 133–149 (2004).
7. Amyot, R. & Flechsig, H. BioAFMviewer: An interactive interface for simulated AFM scanning of biomolecular structures and dynamics. *PLoS computational biology* **16,** e1008444 (2020).
8. Salapaka, S. M. & Salapaka, M. V. Scanning probe microscopy. *IEEE Control Systems Magazine* **28,** 65–83 (2008).
9. Binnig, G., Quate, C. F. & Gerber, C. Atomic Force Microscope. *Phys. Rev. Lett.* **56,** 930–933 (1986).
10. Eaton, P. & West, P. *Atomic force microscopy* (Oxford university press, 2010).
11. Yu, Y. *et al.* Atomic resolution imaging of halide perovskites. *Nano letters* **16,** 7530–7535 (2016).
12. Passeri, D. *et al.* Indentation modulus and hardness of polyaniline thin films by atomic force microscopy. *Synthetic metals* **161,** 7–12 (2011).
13. D'Antò, V. *et al.* Evaluation of surface roughness of orthodontic wires by means of atomic force microscopy. *The Angle Orthodontist* **82,** 922–928 (2012).
14. Dallaeva, D. *et al.* AFM imaging and fractal analysis of surface roughness of AlN epilayers on sapphire substrates. *Applied Surface Science* **312,** 81–86 (2014).
15. Acikgoz, O. & Baykara, M. Z. Speed dependence of friction on single-layer and bulk MoS2 measured by atomic force microscopy. *Applied Physics Letters* **116,** 071603 (2020).
16. Zeng, X., Peng, Y. & Lang, H. A novel approach to decrease friction of graphene. *Carbon* **118,** 233–240 (2017).
17. Hughes, M. L. & Dougan, L. The physics of pulling polyproteins: a review of single molecule force spectroscopy using the AFM to study protein unfolding. *Reports on Progress in Physics* **79,** 076601 (2016).
18. Moody, J. & Allen, S. Atomic force microscopy: applications in biology. *Chemical Biology Applications and Techniques, Larijani B., Rosser, CA, Woscholski, R., Eds. John Wiley & Sons, Ltd., Hoboken, NJ, USA,* 29–45 (2006).
19. Wright, C. J. & Armstrong, I. The application of atomic force microscopy force measurements to the characterisation of microbial surfaces. *Surface and Interface Analysis* **38,** 1419–1428 (2006).
20. Dufrêne, Y. F. Using nanotechniques to explore microbial surfaces. *Nature Reviews Microbiology* **2,** 451–460 (2004).
21. Tyagi, A. K. & Malik, A. In situ SEM, TEM and AFM studies of the antimicrobial activity of lemon grass oil in liquid and vapour phase against Candida albicans. *Micron* **41,** 797–805 (2010).
22. Maghsoudy-Louyeh, S., Kropf, M. & Tittmann, B. Review of progress in atomic force microscopy. *The Open Neuroimaging Journal* **12** (2018).
23. Schitter, G., Steininger, J., Heuck, F. & Staufer, U. Towards fast AFM-based nanometrology and nanomanufacturing. *Int. J. of Nanomanufacturing* **8,** 392–418 (2012).

24. Dufrêne, Y. F. Atomic force microscopy, a powerful tool in microbiology. *Journal of bacteriology* **184,** 5205–5213 (2002).
25. Amyot, R., Marchesi, A., Franz, C. M., Casuso, I. & Flechsig, H. Simulation atomic force microscopy for atomic reconstruction of biomolecular structures from resolution-limited experimental images. *PLoS computational biology* **18,** e1009970 (2022).
26. Liu, Y., Mollaeian, K. & Ren, J. Finite element modeling of living cells for AFM indentation-based biomechanical characterization. *Micron* **116,** 108–115 (2019).
27. Han, R. & Chen, J. A modified Sneddon model for the contact between conical indenters and spherical samples. *Journal of Materials Research* **36,** 1762–1771 (2021).
28. Kontomaris, S. & Malamou, A. Hertz model or Oliver & Pharr analysis? Tutorial regarding AFM nanoindentation experiments on biological samples. *Materials Research Express* **7,** 033001 (2020).
29. Senda, Y., Blomqvist, J. & Nieminen, R. M. Computational model for noncontact atomic force microscopy: energy dissipation of cantilever. *Journal of Physics: Condensed Matter* **28,** 375001 (2016).
30. Roduit, C. *et al.* Stiffness tomography by atomic force microscopy. *Biophysical journal* **97,** 674–677 (2009).
31. Rajabifar, B., Wagner, R. & Raman, A. A fast first-principles approach to model atomic force microscopy on soft, adhesive, and viscoelastic surfaces. *Materials Research Express* **8,** 095304 (2021).
32. Zheng, Q., Zhu, H. & Yu, A. Finite element analysis of the contact forces between a viscoelastic sphere and rigid plane. *Powder Technology* **226,** 130–142 (2012).
33. Kontomaris, S.-V. The Hertz model in AFM nanoindentation experiments: applications in biological samples and biomaterials. *Micro and Nanosystems* **10,** 11–22 (2018).
34. Determination of Elastic Moduli of Thin Layers of Soft Material Using the Atomic Force Microscope. *Biophysical Journal* **82,** 2798–2810. ISSN: 0006-3495 (2002).
35. Dokukin, M. E., Guz, N. V. & Sokolov, I. Quantitative study of the elastic modulus of loosely attached cells in AFM indentation experiments. *Biophysical journal* **104,** 2123–2131 (2013).
36. Glaubitz, M. *et al.* A novel contact model for AFM indentation experiments on soft spherical cell-like particles. *Soft Matter* **10,** 6732–6741 (2014).
37. Kontomaris, S., Stylianou, A., Nikita, K. & Malamou, A. Determination of the linear elastic regime in AFM nanoindentation experiments on cells. *Materials Research Express* **6,** 115410 (2019).
38. Chen, Z., Luo, J., Doudevski, I., Erten, S. & Kim, S. H. Atomic Force Microscopy (AFM) Analysis of an Object Larger and Sharper than the AFM Tip. *Microscopy and Microanalysis* **25,** 1106–1111 (2019).
39. Kontomaris, S.-V. & Malamou, A. The harmonic motion of a rigid cylinder on an elastic half-space. *European Journal of Physics* **41,** 015003 (2019).
40. Vinckier, A. & Semenza, G. Measuring elasticity of biological materials by atomic force microscopy. *FEBS letters* **430,** 12–16 (1998).
41. Korayem, M. & Taheri, M. Modeling of various contact theories for the manipulation of different biological micro/nanoparticles based on AFM. *Journal of nanoparticle research* **16,** 1–18 (2014).
42. Harding, J. & Sneddon, I. *The elastic stresses produced by the indentation of the plane surface of a semi-infinite elastic solid by a rigid punch* in *Mathematical Proceedings of the Cambridge Philosophical Society* **41** (1945), 16–26.
43. Johnson, K. L., Kendall, K. & Roberts, a. Surface energy and the contact of elastic solids. *Proceedings of the royal society of London. A. mathematical and physical sciences* **324,** 301–313 (1971).
44. Derjaguin, B. V., Muller, V. M. & Toporov, Y. P. Effect of contact deformations on the adhesion of particles. *Journal of Colloid and interface science* **53,** 314–326 (1975).

45. Burnham, N. A. & Kulik, A. J. in *Handbook of Micro/Nano Tribology* 247–271 (CRC Press, 2020).
46. Carpick, R. W., Ogletree, D. F. & Salmeron, M. A general equation for fitting contact area and friction vs load measurements. *Journal of colloid and interface science* **211,** 395–400 (1999).
47. Pietrement, O. & Troyon, M. General equations describing elastic indentation depth and normal contact stiffness versus load. *Journal of colloid and interface science* **226,** 166–171 (2000).
48. Sun, Y., Akhremitchev, B. & Walker, G. C. Using the adhesive interaction between atomic force microscopy tips and polymer surfaces to measure the elastic modulus of compliant samples. *Langmuir* **20,** 5837–5845 (2004).

Appendix

A Notation

δ - Indentation depth of the indenter into surface

δ_{12} - Indentation displacement of the indenter

δ^*_{32} - Indentation depth of the surface into base

F Total force of indentation

F_{12} - Contact force experienced by indenter

F_{32} - Contact force experienced by base

R_{12} - Tip-surface contact radius

R_1/R - Indenter radius

R_2/r - Surface radius

$\frac{1}{R_{12}} = \frac{1}{R_1} + \frac{1}{R_2}$,

h - Surface radius/depth

α - Indenter principle angle

E_2/E - Young's modulus

ν_2 - Poisson's ratio of the sample

B Models of Indentation in Atomic Force Microscopy

B.1 Hertz Model

The Hertz model is the most prevalent model used for AFM indention and describes contact forces for isotropic and homogeneous materials, denoted as elastic half-spaces, which present a linearly elastic response to forces[37] and are assumed to extend infinitely in all directions with a flat boundary on the top surface.[39]. This necessitates the AFM tip radius be at least ten times smaller than the sample dimension for the assumption to hold. In addition, it is assumed that the contract is smooth and continuous with no friction or adhesion[37].

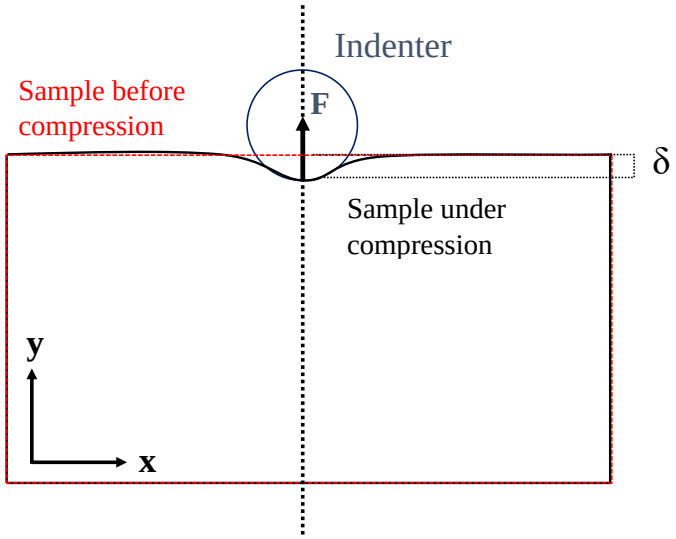


Figure 24: Illustration of contact experienced by elastic half-space. Where δ is indentation depth of the indenter into surface, F is total force of indentation.

Subsequently, the Hertz model provides the relation between the applied force, F, and the indentation depth, δ,[28,33]

$$F_{Hertz}(\delta) = \frac{4}{3}\frac{E_2}{(1-\nu_2^2)}\sqrt{R_{12}}\delta^{3/2} \tag{2}$$

$$\frac{1}{R_{12}} = \frac{1}{R_1} + \frac{1}{R_2}$$

where R_{12} is the tip-surface contact radius, R_1 is indenter radius, R_2 is surface radius (inf for a plane) E_2 Young's modulus and ν_2 is the Poisson's ratio of the sample. This relation allows calculation of Young's modulus as a fitting parameter for experimental measurements of force curves[35,36,40]. In addition, taking the logarithm of this equation returns a linear equation:

$$\log\left(F_{Hertz}(\delta)\right) = \frac{3}{2}\log(\delta) + \log\left(\frac{4}{3}\frac{E_2}{(1-\nu_2^2)}\sqrt{R_{12}}\right) \tag{3}$$

Subsequently, this equation can be used to cast indentation data in dimensional units of for force ($\frac{F}{E^* R^2}$) and indentation (δ/R), such that:

$$\frac{F_{Hertz}(\delta)}{E^* R_{12}^2} = \frac{4}{3}(\frac{\delta}{R_{12}})^{3/2} \tag{4}$$

$$F_{Dimensionless} = \frac{4}{3}(\delta_{Dimensionless})^{3/2} \tag{5}$$

where $E^* = \frac{E_2}{(1-\nu_2^2)}$. Therefore, $F_{Dimensionless} = \frac{F_{Hertz}}{E^* R_{12}^2}$ and $\delta_{Dimensionless} = \frac{\delta}{R_{12}}$.

B.2 Other Hertzian Model

The Hertz model ignores adhesive and frictional forces and can only be applied for parabolic tips (and approximately for a spherical tip given that $h \ll R$). However, various other 'Hertzian' contact models[41], including the models of Sneddon[27,42], JKR[43] , DMT[44], BCP[45], COS[46], PT[47], and SUN[48], use the same base dynamics but incorporate various other adhesive forces, tip geometry and dynamics, and constraints to obtain more advanced models of contact mechanics. Some other models used are discussed below.

B.3 Dimitriadis Model

A more complex model that is applied to spherical indenters for soft materials in AFM experiment is the model by Dimitriadis *et al*[34].

$$\begin{aligned} F_{Dimitriadis}(\delta) =& \frac{4}{3}\frac{E_2}{(1-\nu_2^2)} R_{12}^{1/2}\delta^{3/2}\left[1 - \frac{2\alpha_0}{\pi}\chi + \left(\frac{2\alpha_0}{\pi}\chi\right)^2\right. \\ & \left. - \left(\frac{2\alpha_0}{\pi}\chi\right)^3 + \left(\frac{2\alpha_0}{\pi}\chi\right)^4 - \frac{16}{\pi^2}\beta_0\left(\frac{2\pi}{15} - \frac{3}{5}\alpha_0\chi\right)\chi^3\right] \end{aligned} \tag{6}$$

where R_{12} is the radius of contact between the tip and the surface $\frac{1}{R_{12}} = \frac{1}{R_1} + \frac{1}{R_2}$, R_1 is the indenter radius, R_2 is the surface radius (inf for a plane) Young's modulus E_2 and ν_2 is the Poisson's ratio of the sample. And

$$\chi = \frac{\sqrt{R_{12}\delta}}{h}$$

$$\alpha_0 = -0.347\frac{3-2\nu_2}{1-\nu_2}$$

$$\beta_0 = 0.056\frac{5-2\nu_2}{1-\nu_2}$$

B.4 Sneddon Model

Applying the Hertz model to conical indenters, we recover the Sneddon model[27] given as

$$F_{Sneddon}(\delta) = \frac{2}{\pi}\frac{E_2}{(1-\nu_2^2)} tan(\alpha)\delta^2 \tag{7}$$

where α is indenter principle angle, R_2 is surface radius (inf for a plane) E_2 Young's modulus and ν_2 is the Poisson's ratio of the sample. In addition, taking the logarithm of this equation returns a linear equation:

$$\log\left(F_{Sneddon}(\delta)\right) = 2\log(\delta) + \log\left(\frac{2}{\pi}\frac{E_2}{(1-\nu_2^2)} tan(\alpha)\right) \tag{8}$$

B.5 Double Contact Model

Indentation into elastic spheres presents a more complex analysis as we must account for both the indentation between the indenter and the surface and the indentation between the surface and the base. As seen in Figure 25, part of the indenter of the displacement is due to the compression of the elastic sphere into the base δ^*_{32}. Moreover, the reaction force between the indenter and sphere will be enhanced as the indenter creates reaction force both due to its indentation into the sample and due to the compression of the sample between the indenter and base. To account for this, we consider double contact models[35,36].

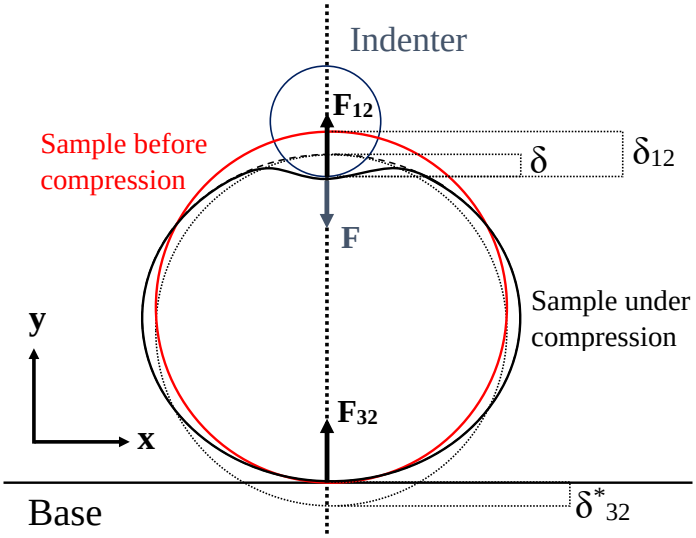


Figure 25: Illustration of double contact experienced by spherical sample. Where δ is indentation depth of the indenter into surface, δ_{12} is total displacement of the indenter, δ^*_{32} indentation depth of the surface into base, F is total force of indentation, F_{12} is contact force experienced by indenter, F_{32} contact force experienced by base. In these models, the corrected values for indentation depth are calculated by subtracting the displacement/ compression of the surface at its base from the displacement of the indenter. For the corrected force, we sum the reaction force between the indenter and surface and the force between the base and surface. This is as the force due to the indenter will be distributed between both the reaction at the base and the indenter.

In these models, the corrected values for indentation depth are calculated by subtracting the displacement/ compression of the surface at its base from the displacement of the indenter.

$$\delta = \delta_{12} - \delta^*_{32} \tag{9}$$

where δ indentation depth of the indenter into surface, δ_{12} is indentation displacement of the indenter, δ^*_{32} indentation depth of the surface into base. For the corrected force, we sum the reaction force between the indenter and surface and the force between the base and surface. This is as the force due to the indenter will be distributed between both the reaction at the base and the indenter.

$$F = F_{12} + F_{32} \tag{10}$$

where F is total force of indentation, F_{12} is contact force experienced by indenter, F_{32} contact force experienced by base.

B.6 Hertz Double Contact

Surface - Spherical Indenter Indentation (Hertz)

$$\delta_{12}(F) = \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\frac{F}{\sqrt{R_{12}}}\right)^{2/3} \tag{11}$$

Spherical Surface

$$\frac{1}{R_{12}} = \frac{1}{R_1} + \frac{1}{R_2}$$

Flat Surface

$$R_{32} = R_2$$

Calculating Double Contact

$$\delta_{12}(F) = \delta(F) + \delta_{32}^*(F)$$

$$= \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\frac{F}{\sqrt{R_{12}}}\right)^{2/3} + \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\frac{F}{\sqrt{R_{32}}}\right)^{2/3}$$

$$= \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\cdot F\right)^{2/3}\left[\frac{1}{R_{12}^{1/3}} + \frac{1}{R_{32}^{1/3}}\right]$$

$$= \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\cdot F\right)^{2/3}\left[\frac{R_{12}^{1/3} + R_{32}^{1/3}}{(R_{12}R_{32})^{1/3}}\right]$$

$$F(\delta_{12}) = \frac{4}{3}\frac{E_2}{(1-\nu_2^2)}\left[\frac{(R_{12}R_{32})^{1/3}}{R_{12}^{1/3} + R_{32}^{1/3}}\right]^{3/2}\delta_{12}^{3/2} \tag{12}$$

where δ indentation depth of the indenter into surface, δ_{12} is indentation displacement of the indenter, δ_{32}^* indentation depth of the surface into base and F is contact force experienced by indenter. R_{12} is the tip-surface contact radius, R_{32} is the surface-base contact radius, R_1 is indenter radius, and $R_2$is surface radius. E_2 Young's modulus and ν_2 is the Poisson's ratio of the sample.

B.7 Hertz-Sneddon Double Contact

Using a similar methodology for Sneddon and Hertz we formulate our own double contact model for conical indenters:

Model for Surface - indenter (Sneddon)

$$\delta_{12}(F) = \left(\frac{\pi}{2}\frac{(1-\nu_2^2)}{E_2}\frac{F}{tan(\alpha)}\right)^{1/2}$$

Model for Surface-Base Indention (Hertz)

$$\delta_{32}(F) = \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\frac{F}{\sqrt{R_{32}}}\right)^{2/3}$$

Flat Surface

$$R_{32} = R_2$$

Calculating Double Contact

$$\delta(F) = \delta_{12}(F) + \delta^*_{32}(F)$$

$$= \left(\frac{\pi}{2}\frac{(1-\nu_2^2)}{E_2}\frac{F}{tan(\alpha)}\right)^{1/2} + \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}\frac{F}{\sqrt{R_{32}}}\right)^{2/3}$$

Let the conical term dominate, and for the spherical term $2/3 \approx 1/2$

$$= \left(\frac{3}{4}\frac{(1-\nu_2^2)}{E_2}F\right)^{1/2}\left(\frac{1}{R_{32}}^{1/4} + (\frac{2\pi}{3}\frac{1}{tan(\alpha)})^{1/2}\right) \tag{13}$$

$$F(\delta_{12}) = \frac{4}{3}\frac{E_2}{(1-\nu_2^2)}\left[\frac{1}{\left(\frac{1}{R_{32}}^{1/4} + (\frac{2\pi}{3}\frac{1}{tan(\alpha)})^{1/2}\right)}\right]^2 \delta_{12}^2 \tag{14}$$

where δ indentation depth of the indenter into surface, δ_{12} is indentation displacement of the indenter, δ^*_{32} indentation depth of the surface into base and F is contact force experienced by indenter. R_{12} is the tip-surface contact radius, R_{32} is the surface-base contact radius, R_1 is indenter radius, and $R_2$is surface radius. E_2 Young's modulus and ν_2 is the Poisson's ratio of the sample.

C ABAQUS Script

C.1 Contact Model Script Example

Imports

In [ ]:
```
from platform import python_version
print(python_version())

# ----------------------------------------------------System Imports-----------------------------------------------------
import os
import sys
import time
import subprocess

# ------------------------------------ Possible modules to pip instal-------------------------------------------------
#!{sys.executable} -m pip install pip install pyabaqus==2022
#!{sys.executable} -m pip install scp
#!{sys.executable} -m pip install paramiko


# ----------------------------------------------Server commands-----------------------------------------------------
import paramiko
from scp import SCPClient
host = "128.40.163.27"
port = 22
username = "giblnbrnhm_j"
password = "axenub13"

# ----------------------------------------------Mathematical Imports-----------------------------------------------
# Importing relevant maths and graphing modules
import numpy as np
import pandas as pd
import math
from numpy import random
from random import randrange
from scipy.optimize import curve_fit

import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from matplotlib.ticker import MaxNLocator

linewidth = 5.92765763889 # inch

plt.rcParams["figure.figsize"] = (1.61*linewidth/3, linewidth/3)
plt.rcParams['figure.dpi'] = 256
plt.rcParams['font.size'] = 16
plt.rcParams["font.family"] = "Times New Roman"

plt.rcParams['mathtext.fontset'] = 'custom'
plt.rcParams['mathtext.rm'] = 'Times New Roman'
plt.rcParams['mathtext.it'] = 'Times New Roman:italic'
plt.rcParams['mathtext.bf'] = 'Times New Roman:bold'

# For displaying images in Markdown
from IPython.display import Image

# ----------------------------------------------Optimization modules-----------------------------------------------
import numba
from numba import prange
```

Abaqus Simulation

Main Abaqus Script

In [ ]:
```
%%writefile AFMtestplane_Plane.py

# ----------------------------------------------Load Modules-------------------------------------------------------
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
#from abaqus import getInput
executeOnCaeStartup()

# ----------------------------------------------Set variables-----------------------------------------------------
depth = [3, 5 ,10,15,20,30,40,50,60,70,80,90,100]
R_indentor = 5

# Simulation variables
timePeriod = 1.5
timeInterval = 0.1
N = int(timePeriod/ timeInterval)

# Set array for data
RF = np.zeros([len(depth),N+1])
```

```
U2 = np.zeros([len(depth),N+1])



#  ----------------------------------------------Model-----------------------------------------------------------------
for i, r in enumerate(depth):
    modelName = jobName = 'AFMtestplane_Plane-'+str(i)
    model = mdb.Model(name=modelName)


# ----------------------------------------------Set Parts-------------------------------------------------------------
    # Create Indentor part
    model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,R_indentor),point2=(0,-R_indentor))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0),point1=(0,R_indentor),point2=(0,-R_indentor),
                                        direction = CLOCKWISE)
    model.Part(name='indentor', dimensionality=AXISYMMETRIC, type= DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseWire(sketch = model.sketches['indentor'])

    # Create plane part
    model.ConstrainedSketch(name = 'plane', sheetSize=1.0)
    model.sketches['plane'].ConstructionLine(point1=(0,r),point2=(0,-r))
    model.sketches['plane'].rectangle(point1=(0,r),point2=(2*r,-r))
    model.Part(name='plane', dimensionality=AXISYMMETRIC, type=DEFORMABLE_BODY)
    model.parts['plane'].BaseShell(sketch = model.sketches['plane'])

# ----------------------------------------------Set Geometry----------------------------------------------------------
    # Create geometric sets for referencing
    model.parts['plane'].Set(faces= model.parts['plane'].faces, name='plane_edges')
    model.parts['indentor'].Set(edges= model.parts['indentor'].edges, name='indentor_edges')

    # Create gemoetric surface for contact
    model.parts['plane'].Surface(name='plane_surface_top',
                                 side1Edges = model.parts['plane'].edges.getSequenceFromMask(mask=('[#1]', ), ) )
    model.parts['plane'].Set(name='plane_surface_bottom',
                             edges = model.parts['plane'].edges.getSequenceFromMask(mask=('[#4]', ), ) )
    model.parts['indentor'].Surface(name='indentor_surface',
                                    side1Edges = model.parts['indentor'].edges.getSequenceFromMask(mask=('[#1]', ), ))

    # Create reference points
    point = model.parts['indentor'].ReferencePoint((0, 0, 0))
    model.parts['indentor'].Set(referencePoints = (model.parts['indentor'].referencePoints[point.id],),
                                name= 'indentor_centre')
    point = model.parts['plane'].ReferencePoint((0, -r, 0))
    model.parts['plane'].Set(referencePoints = (model.parts['plane'].referencePoints[point.id],), name='plane_base')

    model.rootAssembly.regenerate()


# ----------------------------------------------Set Properties--------------------------------------------------------
    # Assign materials
    model.Material(name='plane_material')
    model.materials['plane_material'].Elastic(table=((1000, 0.3), ))
    model.HomogeneousSolidSection(name='section', material='plane_material', thickness=None)
    model.parts['plane'].SectionAssignment( region = model.parts['plane'].sets['plane_edges'], sectionName='section')


# ----------------------------------------------Set Assembly----------------------------------------------------------
    model.rootAssembly.Instance(name='plane', part = model.parts['plane'],dependent=ON)
    model.rootAssembly.Instance(name='indentor', part = model.parts['indentor'], dependent=ON)
    model.rootAssembly.translate(instanceList = ('indentor',) , vector = (0,2*r+R_indentor,0) )
    model.rootAssembly.translate(instanceList = ('plane',)  , vector = (0,r,0) )

    model.rootAssembly.DatumCsysByDefault(CARTESIAN)


# ----------------------------------------------Set Steps-------------------------------------------------------------
    step = model.StaticStep(name='Step-1', previous='Initial', description='', timePeriod=timePeriod,
                            timeIncrementationMethod=AUTOMATIC, maxNumInc=int(1e5),
                            initialInc=1.0, minInc=1e-15, maxInc=1)
    field = model.FieldOutputRequest('F-Output-1', createStepName='Step-1', variables=('RF', 'TF', 'U'),
                                     timeInterval = timeInterval)

    model.steps['Step-1'].control.setValues(allowPropagation=OFF, resetDefaultValues=OFF,
                              timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 25.0, 6.0, 3.0, 50.0))


# ----------------------------------------------Set Interactions------------------------------------------------------
    model.ContactProperty(name = 'Contact Properties')
    model.interactionProperties['Contact Properties'].TangentialBehavior(formulation =ROUGH)
    model.interactionProperties['Contact Properties'].NormalBehavior(pressureOverclosure=HARD)

    model.RigidBody(name= 'indentor_constraint',
                    bodyRegion = model.rootAssembly.instances['indentor'].sets['indentor_edges'],
                    refPointRegion = model.rootAssembly.instances['indentor'].sets['indentor_centre'])

    model.SurfaceToSurfaceContactStd(name    = 'surface-indentor',
                                     createStepName = 'Initial',
                                     master = model.rootAssembly.instances['indentor'].surfaces['indentor_surface'],
                                     slave  = model.rootAssembly.instances['plane'].surfaces['plane_surface_top'],
                                     interactionProperty = 'Contact Properties',
                                     sliding = FINITE)

# ----------------------------------------------Set Loads-------------------------------------------------------------
    # Create surface boundary conditions
    model.DisplacementBC(name='Base-BC',createStepName='Initial',
                         region= model.rootAssembly.instances['plane'].sets['plane_surface_bottom'],
                         u1=SET, u2=SET, ur3= SET)

    # Create indentor boundary conditions
    model.DisplacementBC(name='Indentor-U',createStepName='Step-1',
                         region= model.rootAssembly.instances['indentor'].sets['indentor_centre'],
                         u1=SET, u2=-R_indentor, ur3 = SET )


# ----------------------------------------------Set Mesh--------------------------------------------------------------
    # Assign an 'plane' type to the part instance- seed and generate
```

```
    model.rootAssembly.regenerate()
    model.parts['plane'].seedPart(size = 0.5)
    model.parts['plane'].setElementType(model.rootAssembly.instances['plane'].sets['plane_edges'],
                                        elemTypes =(mesh.ElemType(elemCode=TRI,secondOrderAccuracy = ON),) )
    model.parts['plane'].setMeshControls(regions= model.rootAssembly.instances['plane'].sets['plane_edges'].vertices,
                                         elemShape=TRI, technique=FREE)
    model.parts['plane'].generateMesh()

    model.parts['indentor'].seedPart(size = 0.5)
    model.parts['indentor'].generateMesh()

# ----------------------------------------------Set Submission-------------------------------------------------------
    # Create an analysis job for the model and submit it.
    job = mdb.Job(name=jobName, model=modelName, description='AFM plane')
    job.writeInput()
    job.submit()
    job.waitForCompletion()

# ----------------------------------------------Set Data extraction--------------------------------------------------

    # Opening the odb
    odb    = session.openOdb(jobName +'.odb', readOnly=True)
    region = odb.rootAssembly.nodeSets.values()[1]

    # Extracting Step 1, this analysis only had one step
    step1 = odb.steps.values()[0]

    j,k = 0, 0
    # Creating a for loop to iterate through all frames in the step
    for x in odb.steps[step1.name].frames:
        # Reading stress and strain data from the model
        fieldRF = x.fieldOutputs['RF'].getSubset(region= region)
        fieldU  = x.fieldOutputs['U'].getSubset(region= region)

        # Storing Stress and strain values for the current frame
        for rf in fieldRF.values:
            RF[i,j] = rf.data[1]
            j+=1

        for u in fieldU.values:
            U2[i,k] = u.data[1]
            k+=1

# Writing to a .csv file
np.savetxt("U2_Results.csv", U2 , delimiter=",")
np.savetxt("RF_Results.csv", RF , delimiter=",")

# Close the odb
odb.close()

# Saves file
mdb.saveAs('AFMtestSphere-Plane.cae')
```

Submissions:

In [ ]:
```
# !abaqus fetch job=AFMtestSphere_Plane
# !abaqus cae -noGUI AFMtestSphere_Plane.py
```

In [ ]:
```
t0 = time.time()

localPath = os.getcwd()
remotePath = '/home/giblnbrnhm_j@MECHENG2012/ABAQUS/Spherical-Plane_Indentation/'
script = 'AFMtestSphere_Plane.py'
command  = 'cd ' + remotePath + '\n /opt/abaqus2018/abq2018 cae -noGUI ' + script + ' & \n'

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(host, port, username, password)

# Make working directory
stdin, stdout, stderr = ssh_client.exec_command('mkdir ' + remotePath)
lines = stdout.readlines()
print(lines)

# SCPClient Uploading content into remote directory
scp_client = SCPClient(ssh_client.get_transport())
scp_client.put(script, recursive=True, remote_path=remotePath)
scp_client.close()

# Execute abaqus command
stdin, stdout, stderr = ssh_client.exec_command(command)
lines = stdout.readlines()
print(lines)

ssh_client.close()

t1 = time.time()
print(t1-t0)
```

In [ ]:
```
t0 = time.time()
localPath = os.getcwd()
remotePath = '/home/giblnbrnhm_j@MECHENG2012/ABAQUS/Spherical-Plane_Indentation/'

ssh_client =paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(host, port, username, password)

ftp_client=ssh_client.open_sftp()
ftp_client.get(remotePath+'U2_Results.csv', localPath+'\\U2_Results.csv')
ftp_client.get(remotePath+'RF_Results.csv', localPath+'\\RF_Results.csv')
ftp_client.close()
ssh_client.close()
```

```
t1 = time.time()
print(t1-t0)
```

Data Processing

In [ ]:
```
# ----------------------------------------------------Variables-----------------------------------------------------
U2 = np.array(np.loadtxt('U2_Results.csv', delimiter=","))
RF = np.array(np.loadtxt('RF_Results.csv', delimiter=","))

# Set force cut off to fit at set maximum indentation force as apposed to depth
RFmax = np.max(RF, axis=0).min()
RF = np.ma.masked_less_equal(RF, RFmax)
U2 = np.ma.masked_array(U2, mask = np.ma.getmask(RF))

depth = np.array([2.5, 5 ,10,15,20,30,40,50,60,70,80,90,100])
v = 0.3
R_indentor = 5
E = 1000
E_eff = E/(1-v**2)
markers = ['o','s','^','H','D','X']

# ----------------------------------------------------Plot 1-------------------------------------------------------
fig, ax = plt.subplots(1,1)
for i in range(0,len(U2),5):
    ax.plot(-U2[i]/R_indentor, -RF[i]/(E_eff*R_indentor**2), markers[int(i/4)], ms = 5, label = r'$\frac{h}{R}$='+str(depth[i]/R_indentor))
ax.set_xlabel(r'$\frac{\delta}{R}$')
ax.set_ylabel(r'$\frac{F}{E^*R^2}$')
ax.set_xlim([0, ax.get_xlim()[1]])
ax.set_ylim([0, ax.get_ylim()[1]])
# ax.legend(frameon=False, loc = [0,0.4])

plt.gca().xaxis.set_major_locator(MaxNLocator(prune='lower'))
plt.locator_params(axis='x', nbins=5)
plt.locator_params(axis='y', nbins=4)

fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\Masters Project\\Figure\\Sphere-Plane-Force_Curve-Indenter.png', bbox_inches = 'tight')

plt.show()
```

Parameter Fit

Spherical Indentor (Hertz)

$$F(\delta) = \frac{4}{3}\frac{E}{(1-\nu^2)}R_{eff}^{1/2}\delta^{3/2}$$

for Flat Surface

$$R_{eff} = R_{Indentor}$$

In [ ]:
```
E_hertz = np.zeros(len(depth))
err_hertz =  np.zeros(len(depth))

x = np.linspace(0,5,40)

for i, r in enumerate(depth):
    def F_Hertz(U,E):
        return (4/3) * (E/(1-v**2)) * np.sqrt(R_indentor) * U**(3/2)

    u2, rf = abs(U2[i]),abs(RF[i])
    popt, pcov = curve_fit(F_Hertz, u2, rf)

    E_hertz[i] = popt
    err_hertz[i]= pcov

# fig, ax = plt.subplots(len(depth), 1, figsize = (20,40))
# for i, r in enumerate(depth):
#     ax[i].plot(u2, rf, 'x', color = 'b', label = 'Cylinder radius/ depth r='+str(r))
#     ax[i].plot(x, F_Hertz(x,popt), '-', color = 'k', label = 'Hertz fit r='+str(r))
#     ax[i].plot(x, F_Hertz(x,popt+pcov[0]), ':', color = 'g', label = 'Hertz Overfit r='+str(r))
#     ax[i].plot(x, F_Hertz(x,popt-pcov[0]), ':', color = 'r', label = 'Hertz Underfit r='+str(r))

#     ax[i].set_xlabel('Indentation depth')
#     ax[i].set_ylabel('Force')
#     ax[i].set_title('Spherical Indentor Force curve variation')
#     ax[i].legend()

# plt.subplots_adjust(hspace = 0.5)
# plt.show()
```

Complex Spherical Indentor (Dimitriadis):

https://doi.org/10.1016/S0006-3495(02)75620-8

$$F(\delta) = \frac{4}{3}\frac{E}{(1-\nu^2)}R_{eff}^{1/2}\delta^{3/2} \left[1 - \frac{2\alpha_0}{\pi}\chi + \left(\frac{2\alpha_0}{\pi}\chi\right)^2 - \left(\frac{2\alpha_0}{\pi}\chi\right)^3 + \left(\frac{2\alpha_0}{\pi}\chi\right)^4 -\frac{16}{\pi^2}\beta_0 \left(\frac{2\pi}{15} - \frac{3}{5}\alpha_0\chi \right)\chi^3 \right]$$$$\chi = \frac{\sqrt{R_{eff}\delta}}{h}$$$$\alpha_0 = -0.347\frac{3-2\nu}{1-\nu}$$$$\beta_0 = 0.056\frac{5-2\nu}{1-\nu}$$

In [ ]:
```
E_dim = np.zeros(len(depth))
err_dim =  np.zeros(len(depth))

x = np.linspace(0,5,40)

for i, r in enumerate(depth):
    alpha = -0.347*(3-2*v)/(1-v)
    beta = 0.056*(5-2*v)/(1-v)

    def F_Dim(U,E):
        chi = np.sqrt(U*R_indentor)/(2*r)
```

```
        return (4/3) * (E/(1-v**2)) * np.sqrt(R_indentor) * U**(3/2)*(1-(2*alpha*chi/np.pi)+(2*alpha*chi/np.pi)**2-(2*alpha*chi/np.pi)**3+(2*alpha*ch

    u2, rf = abs(U2[i]),abs(RF[i])
    popt, pcov = curve_fit(F_Dim, u2, rf)

    E_dim[i] = popt
    err_dim[i]= pcov

# fig, ax = plt.subplots(len(depth), 1, figsize = (20,40))
# for i, r in enumerate(depth):
#     ax[i].plot(u2, rf, 'x', color = 'b', label = 'Cylinder radius/ depth r='+str(r))
#     ax[i].plot(x, F_Dim(x,popt), '-', color = 'k', label = 'Dimitriadis fit r='+str(r))
#     ax[i].plot(x, F_Dim(x,popt+pcov[0]), ':', color = 'g', label = 'Dimitriadis Overfit r='+str(r))
#     ax[i].plot(x, F_Dim(x,popt-pcov[0]), ':', color = 'r', label = 'Dimitriadis Underfit r='+str(r))

#     ax[i].set_xlabel('Indentation depth')
#     ax[i].set_ylabel('Force')
#     ax[i].set_title('Spherical Indentor into Cylinder Force curve variation')
#     ax[i].legend()

# plt.subplots_adjust(hspace = 0.5)
# plt.show()
```

Comparison

In [ ]:
```
k = 3
r = depth[k]
x = np.linspace(0,5,40)

u2, rf = abs(U2[i]), abs(RF[k])

# ------------------------------------------------------Plot 1------------------------------------------------------
fig, ax = plt.subplots(1, 1)
ax.plot(u2/R_indentor, rf/(E_eff*R_indentor**2), '^', ms = 5,                    color = 'k', label = 'Data')
ax.plot(x/R_indentor, F_Hertz(x,E_hertz[k])/(E_eff*R_indentor**2), '-', lw = 1, color = 'r', label = 'Hertz ')
ax.plot(x/R_indentor, F_Dim(x,E_dim[k])/(E_eff*R_indentor**2),    '--', lw = 1, color = 'g', label = 'Dimitriadis ')

ax.set_xlabel(r'$\frac{\delta}{R}$')
ax.set_ylabel(r'$\frac{F}{E^*R^2}$')
ax.set_xlim([0, ax.get_xlim()[1]])
ax.set_ylim([0, ax.get_ylim()[1]])
# ax.legend(frameon=False, loc = [0,0.55])

plt.gca().xaxis.set_major_locator(MaxNLocator(prune='lower'))
plt.locator_params(axis='x', nbins=6)
plt.locator_params(axis='y', nbins=4)

fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\Masters Project\\Figure\\Sphere-Plane-Contact_Models.png', bbox_inches = 'tight')

plt.show()


# ------------------------------------------------------Plot 2------------------------------------------------------
fig, ax = plt.subplots(1, 1)
ax.plot((u2[1:]/R_indentor), (rf[1:]/(E_eff*R_indentor**2)), '^', ms = 5,                    color = 'k', label = 'Data')
ax.plot((x[1:]/R_indentor), (F_Hertz(x,E_hertz[k])[1:]/(E_eff*R_indentor**2)), '-', lw = 1, color = 'r', label = 'Hertz ')
ax.plot((x[1:]/R_indentor), (F_Dim(x,E_dim[k])[1:]/(E_eff*R_indentor**2)),    '--', lw = 1, color = 'g', label = 'Dimitriadis ')

ax.set_xlabel(r'log|$\frac{\delta}{R}$|')
ax.set_ylabel(r'log|$\frac{F}{E^*R^2}$|')
ax.set_xscale('log')
ax.set_yscale('log')
# ax.legend(frameon=False, loc = [0,0.45])

fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\Masters Project\\Figure\\Sphere-Plane-Contact_Models-log.png', bbox_inches = 'tight')

plt.show()
```

Youngs Modulus

In [ ]:
```
def G1_hertz(x, a, b, c, d):
    return a/abs(-b * x + c) + d

def G2_hertz(x, a, b, c, d):
    return a*np.exp(-b * x + c) + d

popt1_hertz, pcov1_hertz = curve_fit(G1_hertz, depth, E_hertz, bounds = ([0,0, 0,900],[1000,0.5, 0.5,1100]))
popt2_hertz, pcov2_hertz = curve_fit(G2_hertz, depth, E_hertz, bounds = ([0,0, 0,900],[1000,2, 2,1100]))

err1_hertz = np.diag(pcov1_hertz)
err2_hertz = np.diag(pcov2_hertz)

print('Hertxz Fits: Reciprocal',popt1_hertz[-1],err1_hertz[-1],'\n Exponential', popt2_hertz[-1], err2_hertz[-1])

def G1_dim(x, a, b, c, d):
    return -a/abs(-b * x + c) + d

def G2_dim(x, a, b, c, d):
    return -a*np.exp(-b * x + c) + d

popt1_dim, pcov1_dim = curve_fit(G1_dim, depth, E_dim, bounds = ([0,0, 0,900],[1000,0.5, 0.5,1100]))
popt2_dim, pcov2_dim = curve_fit(G2_dim, depth, E_dim, bounds = ([0,0, 0,900],[1000,2, 2,1100]))

err1_dim = np.diag(pcov1_dim)
err2_dim = np.diag(pcov2_dim)

print('Dimitriadis: Reciprocal',popt1_dim[-1],err1_dim[-1],'\n Exponential', popt2_dim[-1], err2_dim[-1])
```

In [ ]:
```
#print('Fitted Modulus for Hertz fit',E_hertz, err_hertz)
x = np.linspace(0,100,100)
```

```
# ----------------------------------------------------Plot 1----------------------------------------------------
fig, ax = plt.subplots(1,1)
ax.errorbar(depth/R_indentor, E_hertz/E,  yerr=err_hertz/E,  fmt='o', color = 'r', lw =1, ms = 3, capsize=3, label = 'Hertz')
ax.errorbar(depth/R_indentor, E_dim/E,    yerr=err_dim/E,    fmt='x', color = 'g', lw =1, ms = 3, capsize=3, label = 'Dimitriadis' )

ax.plot(np.linspace(0,depth[-1],10)/R_indentor, E*np.ones(10)/E, ':', color = 'k')

ax.set_xlabel(r'$\frac{h}{R}$')
ax.set_ylabel(r'$\frac{E_{Fitted}}{E_{True}}$')
ax.set_xlim([0, ax.get_xlim()[1]])
ax.set_ylim([ax.get_ylim()[0], ax.get_ylim()[1]])
# ax.legend(frameon=False)

plt.gca().xaxis.set_major_locator(MaxNLocator(prune='lower'))
plt.locator_params(axis='x', nbins=6)
plt.locator_params(axis='y', nbins=4)

fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\Masters Project\\Figure\\Sphere-Plane-Youngs_Modulus.png', bbox_inches = 'tight')

plt.show()
```

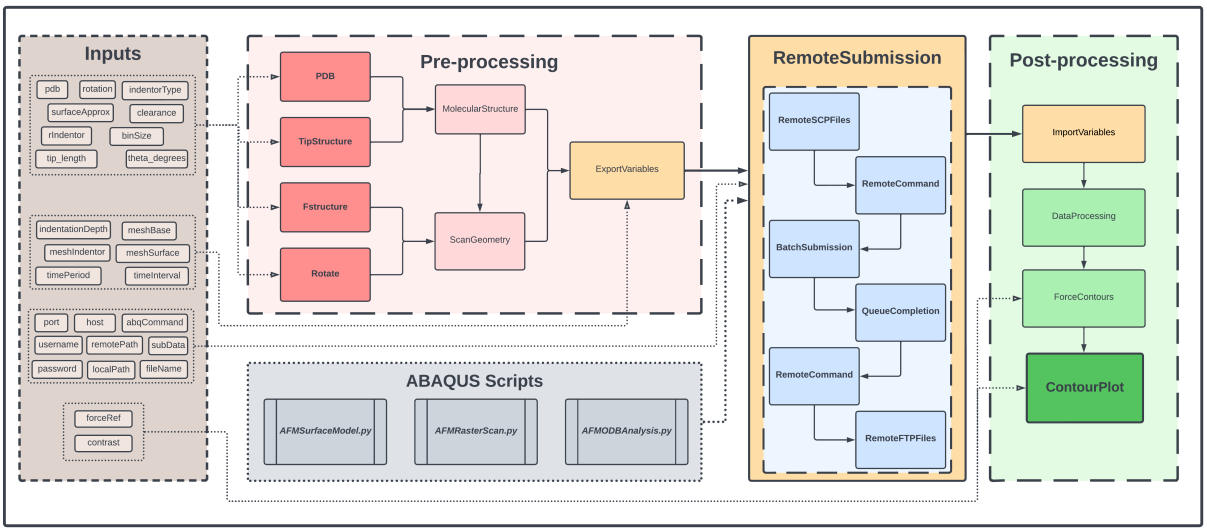
In [ ]:

C.2 Wave ABAQUS Simulation Code

Introduction

Authors: J. Giblin-Burnham

Flow Chart



Code Framework

Imports

In [ ]:
```
from platform import python_version
print(python_version())

# ----------------------------------------------------System Imports-----------------------------------------------------
import os
import sys
import time
import subprocess
from datetime import timedelta

import paramiko
from scp import SCPClient

# ---------------------------------------------------Mathematical Imports----------------------------------------------------
# Importing relevant maths and graphing modules
import numpy as np
import pandas as pd
import math
from numpy import random
from random import randrange
from scipy.interpolate import UnivariateSpline
from scipy.optimize import curve_fit

import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from matplotlib.ticker import MaxNLocator

linewidth = 11.69 # inch

plt.rcParams["figure.figsize"] = (linewidth/3, 1/1.61*linewidth/3)
plt.rcParams['figure.dpi'] = 256
plt.rcParams['font.size'] = 20
plt.rcParams["font.family"] = "Times New Roman"

plt.rcParams['mathtext.fontset'] = 'custom'
plt.rcParams['mathtext.rm'] = 'Times New Roman'
plt.rcParams['mathtext.it'] = 'Times New Roman:italic'
plt.rcParams['mathtext.bf'] = 'Times New Roman:bold'

# For displaying images in Markdown and  animation import
from IPython.display import Image
from IPython.display import clear_output

# ----------------------------------------------------Optimization modules----------------------------------------------------
import numba
from numba import prange
```

ABAQUS Scripts

ABAQUS is run automatically using seperate python scripts defined below. These are transfered to remote server and run using bash commands to produce input files before running simulations.

Raster Scan AFM Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to create ABAQUS input files at each position in the raster scan. These are used to performm the independent analysis/ simulation at each position.

In [ ]:
```
%%writefile AFMtestRasterScan.py

# ----------------------------------------------Load Modules-----------------------------------------------------------------
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
```

```
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
#from abaqus import getInput
executeOnCaeStartup()

# ----------------------------------------------Set variables-----------------------------------------------------
with open('indenterType.txt', 'r') as f:
    indenterType = f.read()

variables = np.loadtxt('variables.csv', delimiter=",")
elasticProperties =  np.loadtxt('elasticProperties.csv', delimiter=",")

waveDims = np.loadtxt('waveDims.csv', delimiter=",")
wavePos  = np.loadtxt('wavePos.csv', delimiter=",")
tipDims  = np.loadtxt('tipDims.csv', delimiter=",")
rackPos  = np.loadtxt('rackPos.csv', delimiter=",")

timePeriod, timeInterval, binSize, indentionDepth, meshIndenter, meshSurface = variables
rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
waveLength, waveAmplitude, waveWidth, groupNum = waveDims

#  ----------------------------------------------Model-----------------------------------------------------
modelName = 'AFMtestRasterScan'
model = mdb.Model(name=modelName)

# ----------------------------------------------Set Parts-----------------------------------------------------
# Create Surface part
model.ConstrainedSketch(name = 'surface', sheetSize=1.0)
model.sketches['surface'].ConstructionLine( point1=(-waveLength*groupNum/2,0), point2=(waveLength*groupNum/2,0) )
model.sketches['surface'].Spline(points= tuple(map(tuple,wavePos)))
model.sketches['surface'].Line(point1 = tuple(wavePos[0]), point2 = tuple(wavePos[0]-np.array([0, 2*waveAmplitude])))
model.sketches['surface'].Line(point1 = tuple(wavePos[0]-np.array([0, 2*waveAmplitude])),
                               point2 = tuple(wavePos[-1]-np.array([0, 2*waveAmplitude])))
model.sketches['surface'].Line(point1 = tuple(wavePos[-1]-np.array([0, 2*waveAmplitude])), point2 = tuple(wavePos[-1]))

model.Part(dimensionality=THREE_D, name='surface', type= DEFORMABLE_BODY)
model.parts['surface'].BaseSolidExtrude(depth=waveWidth, sketch=model.sketches['surface'])

if indenterType == 'Capped':
    # Create Capped-Conical Indentor
    sketch = model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,-rIndentor),point2=(0,z_top))
    model.sketches['indentor'].Line(point1=(r_int,z_int), point2=(r_top,z_top))
    model.sketches['indentor'].Line(point1=(0,-rIndentor), point2=(0,z_top))
    model.sketches['indentor'].Line(point1=(0,z_top), point2=(r_top,z_top))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0), point1=(r_int,z_int), point2=(0,-rIndentor),
                                               direction =CLOCKWISE)
    model.Part(name='indentor', dimensionality=THREE_D, type= DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseShellRevolve(angle=360.0, flipRevolveDirection=OFF, sketch=model.sketches['indentor'])

else:
    # # Create Spherical Indentor part
    model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,rIndentor),point2=(0,-rIndentor))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0),point1=(0,rIndentor),point2=(0,-rIndentor),
                                               direction = CLOCKWISE)
    model.sketches['indentor'].Line(point1=(0,rIndentor),point2=(0,-rIndentor))
    model.Part(name='indentor', dimensionality=THREE_D, type=DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseShellRevolve( angle=360.0, flipRevolveDirection=OFF, sketch=model.sketches['indentor'])


# ----------------------------------------------Set Geometry-----------------------------------------------------
# Create geometric sets for faces and cells
model.parts['surface'].Set(faces= model.parts['surface'].faces.getSequenceFromMask(mask=('[#1]', ), ),
                           name='surface_faces')
model.parts['surface'].Set(cells= model.parts['surface'].cells.getSequenceFromMask(mask=('[#1]', ), ),
                         name='surface_cells')
model.parts['surface'].Set(faces= model.parts['surface'].faces.getSequenceFromMask(mask=('[#4]', ), ),
                           name='surface_base')


if indenterType == 'Capped':
    # Spherically Capped
    model.parts['indentor'].Set(faces= model.parts['indentor'].faces.getSequenceFromMask(mask=('[#7]', ), ),
                                name='indentor_faces')
else:
    # Spherical
    model.parts['indentor'].Set(faces= model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1]', ), ),
                                name='indentor_faces')

# Create gemoetric surface for contact
model.parts['surface'].Surface(name='surface_surface',
                               side1Faces = model.parts['surface'].faces.getSequenceFromMask(mask=('[#1]', ), ) )

if indenterType == 'Capped':
    # Spherically Capped
    model.parts['indentor'].Surface(name='indentor_surface',
                                    side1Faces = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#7]', ), ))
else:
    # Spherical
    model.parts['indentor'].Surface(name='indentor_surface',
                                    side1Faces = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1]', ), ))

# Create reference points
point = model.parts['surface'].ReferencePoint((0, 0, waveWidth/2))
model.parts['surface'].Set(referencePoints=(model.parts['surface'].referencePoints[point.id],),
                           name='surface_centre')
point = model.parts['indentor'].ReferencePoint((0, 0, 0))
model.parts['indentor'].Set(referencePoints = (model.parts['indentor'].referencePoints[point.id],),
                            name = 'indentor_centre')


# ----------------------------------------------Set Properties-----------------------------------------------------
# Assign materials
elastic = (tuple(elasticProperties), )
viscoelastic = ((0.0403,0,0.649),(0.0458,0,1.695),)

# Surface material assignment optional add viscoelastic
model.Material(name='surface_material')
model.materials['surface_material'].Elastic(table = elastic)
# model.materials['surface_material'].Viscoelastic(domain = FREQUENCY, frequency = PRONz, table = viscoelastic )
model.HomogeneousSolidSection(name='section', material='surface_material', thickness=None)
model.parts['surface'].SectionAssignment(region=model.parts['surface'].sets['surface_cells'],sectionName='section')

# ----------------------------------------------Set Assembly-----------------------------------------------------
model.rootAssembly.Instance(name='surface', part = model.parts['surface'],dependent=ON)
model.rootAssembly.Instance(name='indentor', part = model.parts['indentor'], dependent=ON)
model.rootAssembly.DatumCsysByDefault(CARTESIAN)
```

```
# Position base
model.rootAssembly.translate(instanceList = ('surface',), vector=(0.0,0.0, -waveWidth/2))

# ----------------------------------------------Set Steps----------------------------------------------
model.StaticStep(name='Indentation', previous='Initial', description='', timePeriod=timePeriod,
                 timeIncrementationMethod=AUTOMATIC, maxNumInc=int(1e5), initialInc=0.1, minInc=1e-20, maxInc=1)
model.steps['Indentation'].control.setValues(allowPropagation=OFF, resetDefaultValues=OFF,
                                             timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 25.0, 6.0, 3.0,
                                                                 50.0))
field = model.FieldOutputRequest('F-Output-1', createStepName='Indentation', variables=('RF', 'TF', 'U'),
                                 timeInterval = timeInterval)


# ----------------------------------------------Set Interactions----------------------------------------------
model.ContactProperty(name ='Contact Properties')
model.interactionProperties['Contact Properties'].TangentialBehavior(formulation =ROUGH)
model.interactionProperties['Contact Properties'].NormalBehavior(pressureOverclosure=HARD)

model.RigidBody(name = 'indentor_constraint',
                bodyRegion = model.rootAssembly.instances['indentor'].sets['indentor_faces'],
                refPointRegion = model.rootAssembly.instances['indentor'].sets['indentor_centre'])


model.SurfaceToSurfaceContactStd(name = 'surface-indentor',
                                 createStepName = 'Initial',
                                 master = model.rootAssembly.instances['indentor'].surfaces['indentor_surface'],
                                 slave  = model.rootAssembly.instances['surface'].surfaces['surface_surface'],
                                 interactionProperty = 'Contact Properties',
                                 sliding = FINITE)

# ----------------------------------------------Set Loads----------------------------------------------
# Create surface boundary conditions
model.DisplacementBC(name = 'Surface-BC', createStepName = 'Initial',
                     region = model.rootAssembly.instances['surface'].sets['surface_base'],
                     u1 = SET, u2 = SET, u3 = SET, ur1 = SET, ur2 = SET, ur3 = SET)

# Create indentor boundary conditions
model.DisplacementBC(name = 'Indentor-UC', createStepName = 'Indentation',
                     region = model.rootAssembly.instances['indentor'].sets['indentor_centre'],
                     u1  = SET, u2  = -indentionDepth, u3  = SET,
                     ur1 = SET, ur2 = SET,             ur3 = SET)


# ----------------------------------------------Set Mesh----------------------------------------------
#Assign an element type to the part instance- seed and generate
model.rootAssembly.regenerate()

elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D15, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D10, elemLibrary=STANDARD, secondOrderAccuracy=ON, distortionControl=DEFAULT)
cells     = model.parts['surface'].cells.getSequenceFromMask(mask=('[#1 ]', ), )

model.parts['surface'].seedPart(size=meshSurface, minSizeFactor=0.1)
model.parts['surface'].setMeshControls(regions=cells, elemShape=TET, technique=FREE)
model.parts['surface'].setElementType(regions=(cells,), elemTypes=(elemType1, elemType2, elemType3))
model.parts['surface'].generateMesh()


elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=STANDARD)
faces     = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1 ]', ),)

model.parts['indentor'].seedPart(size= meshIndenter, minSizeFactor=0.25)
model.parts['indentor'].setMeshControls(regions=faces, elemShape=TRI)
model.parts['indentor'].setElementType(regions=(faces,), elemTypes=(elemType1, elemType2))
model.parts['indentor'].generateMesh()

# ----------------------------------------------Set Submission----------------------------------------------
for i in range(len(rackPos)):
    jobName = 'AFMtestRasterScan-Pos'+str(int(i))
    model.rootAssembly.translate(instanceList = ('indentor',),vector=(rackPos[i,0],rackPos[i,1]+rIndentor,0))
    job = mdb.Job(name=jobName, model=modelName, description='AFM Sphere')
    job.writeInput()
    model.rootAssembly.translate(instanceList = ('indentor',),vector=(-rackPos[i,0], -rackPos[i,1]-rIndentor, 0))

mdb.saveAs('AFMRaster.cae')
```

ODB Analysis Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to analysis odb files for each position in the raster scan. Extracting indentation data.

In [ ]:

```
%%writefile AFMtestODBAnalysis.py

# ----------------------------------------------Load Modules----------------------------------------------
import sys
from odbAccess import *
from types import IntType
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
#from abaqus import getInput
executeOnCaeStartup()

# ----------------------------------------------Set variables----------------------------------------------
variables = np.loadtxt('variables.csv', delimiter=",")
rackPos    = np.loadtxt('rackPos.csv', delimiter=",")

timePeriod, timeInterval, binSize, indentionDepth, meshIndenter, meshSurface = variables

N = int(timePeriod/ timeInterval)+1
RF = np.zeros([len(rackPos),N])
U2 = np.zeros([len(rackPos),N])

# ----------------------------------------------Set Data extraction----------------------------------------------
for i in range(len(rackPos)):
    jobName = 'AFMtestRasterScan-Pos' + str(int(i))
    try :
        # Opening the odb
        odb    = openOdb(jobName +'.odb', readOnly=True)
        region = odb.rootAssembly.nodeSets.values()[1]
    except:
        with open('Errors.txt', 'a') as f:
            f.write('ERROR for'+str(i)+'\n')
```

```
    else:
        # Extracting Step 1, this analysis only had one step
        step1 = odb.steps.values()[0]

        j,k = 0, 0
        # Creating a for loop to iterate through all frames in the step
        for x in odb.steps[step1.name].frames:
            # Reading stress and strain data from the model
            fieldRF = x.fieldOutputs['RF'].getSubset(region= region)
            fieldU  = x.fieldOutputs['U'].getSubset(region= region)

            # Storing Stress and strain values for the current frame
            for rf in fieldRF.values:
                RF[i,j] = np.sqrt(rf.data[1]**2)
                j+=1

            for u in fieldU.values:
                U2[i,k] = u.data[1]
                k+=1

# Writing to a .csv file
np.savetxt('U2_Results.csv', U2 , delimiter=",")
np.savetxt('RF_Results.csv', RF , delimiter=",")

# Close the odb
odb.close()
```

Simulation Code Functions

Functionalised code to automate scan and geometry calculations, remote server access, remote script submission, data anlaysis and postprocessing required to produce AFM image.

Pre-Processing Function

Functions used in preprocessing step of simulation, including calculating scan positions and exporting variables.

Tip Functions

Functions to produce list of tip structural parameters, alongside function to calculates and returns tip surface heights from radial position r.

In [ ]:
```
def TipStructure(rIndentor, theta_degrees, tip_length):
    '''
    Produce list of tip structural parameters. Change principle angle to radian. Calculate tangent point where
    sphere smoothly transitions to cone for capped conical indentor.

        Parameters:
            theta_degrees (float) - Principle conical angle from z axis in degrees
            rIndentor (float)     - Radius of spherical tip portion
            tip_length (float)    - Total cone height

        Returns:
            tipDims (list) - Geometric parameters for defining capped tip structure
    '''
    theta = theta_degrees*(np.pi/180)

    # Intercept of spherical and conical section of indentor (Tangent point)
    r_int, z_int = rIndentor*abs(np.cos(theta)), -rIndentor*abs(np.sin(theta))
    # Total radius/ footprint of indentor/ top coordinates
    r_top, z_top = (r_int+(tip_length-r_int)*abs(np.tan(theta))), tip_length-rIndentor

    return [rIndentor, theta, tip_length, r_int, z_int, r_top, z_top]
```

In [ ]:
```
def Fconical(r, r0, r_int, z_int, theta, R, tip_length):
    '''
    Calculates and returns spherically capped conical tip surface heights from radial  position r. Uses radial coordinate along
    xz plane from centre as tip is axisymmetric around z axis (bottom of tip set as zero point such z0 = R).

        Parameters:
            r (float/1D arr)   - xz radial coordinate location for tip height to be found
            r0 (float)         - xz radial coordinate for centre of tip
            r_int (float)      - xz radial coordinate of tangent point (point where sphere smoothly transitions to cone)
            z_int (float)      - Height of tangent point, where sphere smoothly transitions to cone (defined for tip centred at spheres
                                 center, as calculations assume tip centred at indentors bottom the value must be corrected to, R-z_int)
            theta (float)      - Principle conical angle from z axis in radians
            R (float)          - Radius of spherical tip portion
            tip_length (float) - Total cone height

        Returns:
            Z (float/1D arr)- Height of tip at xz radial coordinate
    '''

    ### Constructing conical and spherical parts boundaries of tip using arrays for computation speed

    # ------------------------------------------------Spherical Boundary-------------------------------------------------
    # For r <= r_int, z <= z_int : (z-z0)^2 +  (r-r0)^2 = R^2 --> z = z0  + ( R^2 - (r-r0)^2 )^1/2

    # Using equation of sphere compute height (points outside sphere radius are complex and return nan,
    # nan_to_num is used to set these points to max value R). The heights are clip to height of tangent point, R-z_int.
    # Producing spherical portion for r below tangent point r_int and constant height R-zint for r values above r_int.

    z1 = np.clip( np.nan_to_num(R - np.sqrt(R**2 - (r-r0)**2), copy=False, nan=R ), a_min = 0 , a_max = R-abs(z_int))
    # z1 = np.clip( np.where( np.isnan( R - np.sqrt(R**2 - (r-r0)**2) ) , R, R - np.sqrt(R**2 - (r-r0)**2) ), a_min = 0 , a_max = R-np.abs(z_int))

    # ------------------------------------------------Conical Boundary---------------------------------------------------
    # r > r_int, z > z_int : z = m*abs(x-x0);  where x = r, x0 = r0 + r_int,  m = 1/tan(theta)

    # Using equation of cone (line) to compute height for r values larger than tangent point r_int (using where condition)
    # For r values below r_int the height is set to zero

    z2 =np.where(abs(r-r0)>=r_int, (abs(r-r0)-r_int)/abs(np.tan(theta)), 0)

    # ------------------------------------------------Combing Boundaries-------------------------------------------------
    # For r values less than r_int, combines spherical portion with zero values from conical, producing spherical section
    # For r values more than r_int, combines linear conical portion with R-z_int values from spherical, producing cone section
    Z = z1 + z2

    # Optional mask values greater than tip length
    # Z = np.ma.masked_greater(z1+z2, tip_length )
    return Z
```

In [ ]:
```
def Fspherical(r, r0, r_int, z_int, theta, R, tip_length):
    '''
    Calculates and returns spherical tip surface heights from radial  position r. Uses radial coordinate along xz plane from
    centre as tip is axisymmetric around z axis (bottom of tip set as zero point such z0 = R).

        Parameters:
            r (float/1D arr)   - xz radial coordinate location for tip height to be found
            r0 (float)         - xz radial coordinate for centre of tip
            r_int (float)      - xz radial coordinate for tangent point (point where sphere smoothly transitions to cone)
            z_int (float)      - Height of tangent point (point where sphere smoothly transitions to cone)
            theta (float)      - Principle conical angle from z axis in radians
            R (float)          - Radius of spherical tip portion
            tip_length (float) - Total cone height
```

```
        Returns:
            Z (float/1D arr)- Height of tip at xz radial coordinate
    '''
    # Simple spherical equation: (z-z0)^2 +  (r-r0)^2 = R^2 --> z = z0  - ( R^2 - (r-r0)^2 )^1/2
    return ( R - np.sqrt(R**2 - (r-r0)**2) )
```

Scan Functions

Calculate scan positions of tip over surface and vertical set points above surface for each position. In addition, function to plot and visualise molecules surface and scan position.

In [ ]:
```
def waveSin(x, waveDims):
    '''Function defining wave shaped surface'''
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims
    A      = waveAmplitude/2
    omega  = 2*np.pi/waveLength
    phi    = -np.pi/2
    return A*(np.sin(omega*x+phi)+1)
```

In [ ]:
```
def ScanGeometry(indentorType, tipDims, waveDims, Nb, clearance):
    '''
    Produces array of scan locations and corresponding heights/ tip positions above surface in Angstroms (x10-10 m).The scan positions are produced creating a
    straight line along the centre of the surface with positions spaced by the bin size. Heights, at each position, are calculated for conical indentor by set
    tip above sample and calculating vertical distance between of tip and molecules surface over the indnenters area. Subsequently, the minimum vertical distance
    corresponds to the position where tip is tangential. Spherical indentors are calculated explicitly.

        Parameters:
            indentorType (str) - String defining indentor type (Spherical or Capped)
            tipDims (list)     - Geometric parameters for defining capped tip structure
            waveDims (list)    - Geometric parameters for defining base/ substrate structure [wavelength, amplitude, width]
            Nb (int)           - Number of scan positions along x axis of base
            clearance (float)  - Clearance above molecules surface indentor is set to during scan

        Returns:
            rackPos (arr) - Array of coordinates [x,z] of scan positions to image biomolecule
    '''
    #  -------------------------------------Set Rack Positions from Scan Geometry-------------------------------------------
    # Set variables
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims
    [rIndentor, theta, tip_length, r_int, z_int, r_top, z_top] = tipDims

    # Intialise array of raster scan positions
    rackPos = np.zeros([Nb,2])

    # Create linear set of scan positions over base, along x axis, for half a wave length
    rackPos[:,0] = np.linspace(-waveLength/2, 0, Nb)


    # ------------------------------------------------Set Indentor variables-----------------------------------------------
    # Set indentor height functions and indentor radial extent/boundry for initial height calculation.
    if indentorType == 'Capped':
        # Extent of conical indentor is the radius of the top portion
        rBoundary = r_top
        F = Fconical

    else:
        # Extent of spherical indentor is the radius
        rBoundary = rIndentor
        F = Fspherical

    #  ---------------------------------------------Calculate Rack Positions -------------------------------------------------
    for i, rPos in enumerate(rackPos[:,0]):
        # Array of radial positions along indentor radial extent. Set indentor position/ coordinate origin at surface height
        # (z' = z + surfaceHeight) and calculate  vertical heights along the radial extent of indentor at position.
        r0 = np.linspace(rPos-rBoundary, rPos+rBoundary, 1000)
        z0 = F(r0, rPos, r_int, z_int, theta, rIndentor, tip_length) + waveAmplitude

        # Using equation of sphere compute top heights of atoms surface along indentors radial extent (points outside sphere
        # radius are complex and return nan, nan_to_num is used to set these points to the min value of bases surface z=0).
        z  = waveSin(r0, waveDims)

        # The difference in the indentor height and the surface at each point along indenoter extent, produces a dz
        # array of all the height differences between indentor and surface within the indentors boundary around this position.
        # Therefore, z' = -dz  gives an array of indentor positions when each individual part of surface atoms contacts the tip portion above.
        # Translating from z' basis (with origin at z = surfaceHeight) to z basis (with origin at the top of the base) is achieved by
        # perform translation z = z' + surfaceheight. Therefore, these tip position are given by  dz = surfaceheight - dz'. The initial height
        # corresponds to the maximum value of dz/ min value of dz' where the tip is tangential to the surface. I.e. when dz' is minimised
        # all others dz' tip positions will be above/ further from the surface. Therefore, at this position, the rest of the indentor wil
        # not be in contact with the surface and it is tangential.

        rackPos[i,1] = waveAmplitude - abs((z0-z).min()) + clearance

    return rackPos
```

File Import/ Export Function

In [ ]:
```
def ExportVariables(rackPos, variables, waveDims, wavePos, tipDims, indentorType, elasticProperties):
    '''
    Export simulation variables as csv and txt files to load in abaqus python scripts.

        Parameters:
            rackPos (arr)           - Array of coordinates [x,z] of scan positions to image biomolecule
            variables (list)        - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor,
                                      indentionDepth, surfaceHeight]
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [wavelength, amplitude, width]
            wavePos (arr)           - Positions on wave used to define spline in ABAQUS
            tipDims (list)          - Geometric parameters for defining capped tip structure
            indentorType (str)      - String defining indentor type (Spherical or Capped)
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''

    np.savetxt("elasticProperties.csv", elasticProperties, fmt='%s', delimiter=",")
    np.savetxt("variables.csv", variables, fmt='%s', delimiter=",")
    np.savetxt("rackPos.csv", rackPos, fmt='%s', delimiter=",")
    np.savetxt("wavePos.csv", wavePos, delimiter=",")
    np.savetxt("waveDims.csv", waveDims, fmt='%s', delimiter=",")
    np.savetxt("tipDims.csv", tipDims, fmt='%s', delimiter=",")

    with open('indentorType.txt', 'w', newline = '\n') as f:
        f.write(indentorType)
```

In [ ]:
```
def ImportVariables():
    '''
    Import simulation geometry variables from csv files.

        Return:
            variables (list)        - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor,
                                      indentionDepth, surfaceHeight]
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [wavelength, amplitude, width, group number]
            rackPos (arr)           - Array of coordinates [x,z] of scan positions to image biomolecule
    '''
    variables          = np.loadtxt('variables.csv', delimiter=",")
    waveDims           = np.loadtxt('waveDims.csv', delimiter=",")
    rackPos            = np.loadtxt('rackPos.csv', delimiter=",")

    return variables, waveDims, rackPos
```

Remote Functions

Functions for working on remote serve, including transfering files, submitting bash commands, submiting bash scripts for batch input files and check queue statis.

File Transfer`

In [ ]:
```
def RemoteSCPFiles(host, port, username, password, files, remotePath):
    '''
    Function to make directory and transfer files to SSH server. A new Channel is opened and the files are transfered.
    The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            files (str/list) - File or list of file to transfer
            remotePath (str) - Path to remote file/directory
    '''
    # SHH to clusters
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    stdin, stdout, stderr = ssh_client.exec_command('mkdir -p ' + remotePath)

    # SCPClient takes a paramiko transport as an argument- Uploading content to remote directory
    scp_client = SCPClient(ssh_client.get_transport())
    scp_client.put(files, recursive=True, remote_path = remotePath)
    scp_client.close()

    ssh_client.close()
```

Bash Command Submission

In [ ]:
```
def RemoteCommand(host, port, username, password, script, remotePath, command):
    '''
    Function to execute a command/ script submission on the SSH server. A new Channel is opened and the requested command is executed.
    The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            script (str)     - Script to run via bash command
            remotePath (str) - Path to remote file/directory
            command (str)    - Abaqus command to execute and run script
    '''
    # SSH to clusters using paramiko module
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # Execute command
    stdin, stdout, stderr = ssh_client.exec_command('cd ' + remotePath + ' \n '+ command +' '+ script +' & \n')
    lines = stdout.readlines()

    ssh_client.close()

    for line in lines:
        print(line)
```

Batch File Submission

In [ ]:
```
def BatchSubmission(host, port, username, password, fileName, subData, rackPos, remotePath, **kwargs):
    '''
    Function to create bash script for batch submission of input file, and run them on remote server.
        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            fileName (str)   - Base File name for abaqus input files
            subData (str)    - Data for submission to serve queue [walltime, memory, cpus]
            rackPos (arr)    - Array of coordinates [x,z] of scan positions to image biomolecule (can be clipped or full)
            remotePath (str) - Path to remote file/directory

            kwargs:
                Submission ('serial'/ 'paralell') - optional define whether single serial script or seperate paralell submission to queue {Default: 'serial'}
    '''
    # For paralell mode create bash script to runs for single scan location, then loop used to submit individual scripts for each location which run in paralell
    if 'Submission' in kwargs and kwargs['Submission'] == 'paralell':
        lines = ['#!/bin/bash -l',
                 '#$ -S /bin/bash',
                 '#$ -l h_rt='+ subData[0],
                 '#$ -l mem=' + subData[1],
                 '#$ -pe mpi ' + subData[2],
                 '#$ -wd /home/zcapjgi/Scratch/ABAQUS',
                 'module load abaqus/2017 ',
                 'ABAQUS_PARALLELSCRATCH = "/home/zcapjgi/Scratch/ABAQUS" ',
                 'cd ' + remotePath,
                 'gerun abaqus interactive cpus=$NSLOTS mp_mode=mpi job=$JOB_NAME input=$JOB_NAME.inp scratch=$ABAQUS_PARALLELSCRATCH resultsformat=odb'
                ]

    # Otherwise, create script to run serial analysis consecutively with single submission
    else:
        # Create set of submission comands for each scan locations
        jobs   = ['gerun abaqus interactive cpus=$NSLOTS mp_mode=mpi job='+fileName+str(int(i))+' input='+fileName+str(int(i))+'.inp scratch=$ABAQUS_PARALLELSCRATCH resultsformat=odb'
                  for i in range(len(rackPos))]

        # Produce preamble to used to set up bash script
        lines = ['#!/bin/bash -l',
                    '#$ -S /bin/bash',
                    '#$ -l h_rt='+ subData[0],
                    '#$ -l mem=' + subData[1],
                    '#$ -pe mpi ' + subData[2],
                    '#$ -wd /home/zcapjgi/Scratch/ABAQUS',
                    'module load abaqus/2017 ',
                    'ABAQUS_PARALLELSCRATCH = "/home/zcapjgi/Scratch/ABAQUS" ',
                    'cd ' + remotePath ]
        # Combine to produce total  script
        lines+=jobs

    # Create script file in current directory by writing each line to file
    with open('batchScript.sh', 'w', newline = '\n') as f:
        for line in lines:
            f.write(line)
            f.write('\n')

    # SSH to clusters
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

```
    ssh_client.connect(host, port, username, password)

    stdin, stdout, stderr = ssh_client.exec_command('mkdir -p ' + remotePath)

    # SCPClient takes a paramiko transport as an argument- Uploading content to remote directory
    scp_client = SCPClient(ssh_client.get_transport())
    scp_client.put('batchScript.sh', recursive=True, remote_path = remotePath)
    scp_client.close()

    # If paralell mode, submit  individual scripts for individual scan locations
    if 'Submission' in kwargs and kwargs['Submission'] == 'paralell':
        for i in range(len(rackPos)):
            # Job name set as each input file name as -N jobname is used as input variable in script
            jobName = fileName+str(int(i))

            # Command to run individual jobs
            batchCommand = 'cd ' + remotePath + ' \n qsub -N '+ jobName +' batchScript.sh \n'

            # Execute command
            stdin, stdout, stderr = ssh_client.exec_command(batchCommand)
            lines = stdout.readlines()
            print(lines)

    # Otherwise submit single serial scripts
    else:
        # Job name set as current directory name (change / to \\ for windows)
        jobName = remotePath.split('/')[-1]
        batchCommand = 'cd ' + remotePath + ' \n qsub -N '+ jobName +' batchScript.sh \n'

        # Execute command
        stdin, stdout, stderr = ssh_client.exec_command(batchCommand)
        lines = stdout.readlines()
        print(lines)

    ssh_client.close()
```

Queue Status Function

In [ ]:
```
def QueueCompletion(host, port, username, password):
    '''
    Function to check queue statis and complete when queue is empty.
        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)       -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
    '''
    # Log time
    t0 = time.time()
    complete= False

    while complete == False:
        # SSH to clusters
        ssh_client = paramiko.SSHClient()
        ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh_client.connect(host, port, username, password)

        # Execute command to view the queue
        stdin, stdout, stderr = ssh_client.exec_command('qstat')
        lines = stdout.readlines()

        # Check if queue is empty
        if len(lines)==0:
            print('Complete')
            complete = True
            ssh_client.close()

        # Otherwis close and wait 2 mins before checking again
        else:
            ssh_client.close()
            time.sleep(120)

    # Return total time
    t1 = time.time()
    print(t1-t0)
```

File Retrieval

In [ ]:
```
def RemoteFTPFiles(host, port, username, password, files, remotePath, localPath):
    '''
    Function to transfer files from directory on SSH server to local machine. A new Channel is opened and the files are transfered.
    The function uses FTP file transfer.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)       -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            files (str )     - File to transfer
            remotePath (str) - Path to remote file/directory
            localPath (str)  - Path to local file/directory
    '''
    # SSH to cluster
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # FTPClient takes a paramiko transport as an argument- copy content from remote directory
    ftp_client=ssh_client.open_sftp()
    ftp_client.get(remotePath+'/'+files, localPath +'\\'+ files)
    ftp_client.close()
```

Remote Terminal

In [ ]:
```
def Remote_Terminal(host, port, username, password):
    '''
    Function to emulate cluster terminal. Channel is opened and commands given are executed. The command's input
    and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)       -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption
                               if passphrase is not given.
    '''
    # SHH to cluster
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # Create channel to keep connection open
    ssh_channel = ssh_client.get_transport().open_session()
    ssh_channel.get_pty()
    ssh_channel.invoke_shell()
```

```
    # While open accept user input commands
    while True:
        command = input('$ ')
        if command == 'exit':
            break

        ssh_channel.send(command + "\n")

        # Return bash output from command
        while True:
            if channel.recv_ready():
                output = ssh_channel.recv(1024)
                print(output)
            else:
                time.sleep(0.5)
                if not(ssh_channel.recv_ready()):
                    break
    # Close cluster connection
    ssh_client.close()
```

Submission Functions

Function to run simulation and scripts on the remote servers. Files for variables are transfered, ABAQUS scripts are run to create parts and input files. A bash file is created and submitted to run simulation for batch of inputs. Analysis of odb files is performed and data transfered back to local machine. Using keyword arguments invidual parts of simulation previously completed can be skipped.

In [ ]:
```
def RemoteSubmission(host, port, username, password, remotePath, localPath,  csvfiles, abqfiles, abqCommand, fileName, subData, rackPos, **kwargs):
    '''
    Function to run simulation and scripts on the remote servers. Files for variables are transfered, ABAQUS scripts are run to create parts and input files.
    A bash file is created and submitted to run simulation for batch of inputs. Analysis of odb files is performed and data transfered back to local machine.
    Using keyword arguments can submitt the submission files in parrallel.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - Username to authenticate as (defaults to the current local username)
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            remotePath (str) - Path to remote file/directory
            localPath (str)  - Path to local file/directory
            csvfiles (list)  - List of csv and txt files to transfer to remote server
            abqfiles (list)  - List of abaqus script files to transfer to remote server
            abqCommand (str) - Abaqus command to execute and run script
            fileName (str)   - Base File name for abaqus input files
            subData (str)    - Data for submission to serve queue [walltime, memory, cpus]
            rackPos (arr)    - Array of scan positions and initial height [x,z] to image
            kwargs           - Passes "Submmission" if present to batchSubmission function

    '''
    #  ----------------------------------------------File Transfer----------------------------------------------------------
    # Transfer scripts and variable files to remote server
    RemoteSCPFiles(host, port, username, password, csvfiles, remotePath)
    RemoteSCPFiles(host, port, username, password, abqfiles, remotePath)

    print('File Transfer Complete')

    #  ----------------------------------------------Input File Creation----------------------------------------------------
    t0 = time.time()
    print('Producing Input Files ...')

    # Produce simulation and input files
    script = 'AFMtestRasterScan.py'
    RemoteCommand(host, port, username, password, script, remotePath, abqCommand)

    t1 = time.time()
    print('Input File Complete - ' + str(timedelta(seconds=t1-t0)) )

    #  ----------------------------------------------Batch File Submission--------------------------------------------------
    t0 = time.time()
    print('Submitting Batch Scripts ...')

    # Submit bash scripts to remote queue to carry out batch abaqus analysis
    BatchSubmission(host, port, username, password, fileName, subData, rackPos, remotePath, **kwargs)

    t1 = time.time()
    print('Batch Submission Complete - '+ str(timedelta(seconds=t1-t0)) + '\n' )
```

In [ ]:
```
def DataRetrieval(host, port, username, password, scratch, wrkDir, localPath, csvfiles, dataFiles, indentorRadius, **kwargs):
    '''
    Function to retrieve simulation data transfered back to local machine. Using keyword arguments to change to compilation of simulations data.

        Parameters:
            host (str)          - Hostname of the server to connect to
            port (int)          - Server port to connect to
            username (str)      - Username to authenticate as (defaults to the current local username)        -
            password (str)      - Used for password authentication; is also used for private key decryption if passphrase is not given.
            remotePath (str)    - Path to remote file/directory
            localPath (str)     - Path to local file/directory
            csvfiles (list)     - List of csv and txt files to transfer to remote server
            datafiles (list)    - List of abaqus script files to transfer to remote server
            indentorRadius (arr) - Array of indentor radii of spherical tip portion varied for seperate  simulations
            kwargs:
                Compile(int)    - If passed, simulation data is compiled from seperate sets of simulations in directory in remote server to combine
                                  complete indentations. Value is set as int representing the range of directories to compile from (directories must have
                                  same root naming convention with int denoting individual directories)

        Return:
            variables (list) - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshIndentor, indentionDepth]
            TotalU2 (arr)    - Array of indentors z displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
            TotalRF (arr)    - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
            NrackPos (arr)   - Array of initial scan positions for each indenter [Ni, Nb, [x, z] ]
    '''
    #  ----------------------------------------------Remote Variable--------------------------------------------------------
    # Import variables from remote server used for the simulations
    for file in csvfiles:
        RemoteFTPFiles(host, port, username, password, file, scratch+wrkDir+'/IndenterRadius7', localPath)

    # Set simulation variables used to process data
    variables, waveDims, rackPos  = ImportVariables()
    timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface = variables

    # Set array size variables
    Nb, Nt = int((waveLength/2)/(binSize) + 1), int(timePeriod/ timeInterval)+1

    #  ----------------------------------------------Initialise data arrays-------------------------------------------------
    NrackPos = np.zeros([len(indentorRadius), Nb, 2])
    TotalRF  = np.zeros([len(indentorRadius), Nb, Nt])
    TotalU2  = np.zeros([len(indentorRadius), Nb, Nt])


    #  ----------------------------------------------Compiled data retrieval------------------------------------------------
    if 'Compile' in kwargs.keys():
        # Set base directory name to loop over
        wrkDir = '/ABAQUS/TipConvolutionTests/Test_Tip_Convolution'

        # For number of directories set to compile
        for n in range(kwargs['Compile']):
```

```
            for index, rIndentor in enumerate(indentorRadius):
                # Set path to file
                remotePath = scratch + wrkDir + str(n) + '/IndenterRadius'+str(int(rIndentor))

                # Check file is available
                try :
                    RemoteFTPFiles(host, port, username, password, dataFiles[0], remotePath, localPath)
                    RemoteFTPFiles(host, port, username, password, dataFiles[1], remotePath, localPath)
                    RemoteFTPFiles(host, port, username, password, dataFiles[2], remotePath, localPath)

                except:
                    None

                else:
                    # If files are available load data in to temperary variable
                    U2 = np.array(np.loadtxt(dataFiles[0], delimiter=","))
                    RF = np.array(np.loadtxt(dataFiles[1], delimiter=","))
                    NrackPos[index] = np.array(np.loadtxt(dataFiles[2], delimiter=","))

                    # Loop through data and store indentations with less zeros/ higher sums of forces
                    for i in range(len(RF)):
                        if np.all(TotalRF[index,i]==0)  == True or np.count_nonzero(TotalRF[index,i]==0)>np.count_nonzero(RF[i]==0) or sum(RF[i]) > sum(TotalRF[index,i]):
                            TotalU2[index,i]  = U2[i]
                            TotalRF[index,i]  = RF[i]

    #  ----------------------------------------Single Directory retrieval-----------------------------------------------------
    else:
        # For each indentor
        for index, rIndentor in enumerate(indentorRadius):
            # Define path to file
            remotePath = scratch + wrkDir + '/IndenterRadius'+str(int(rIndentor))

            # Retrive data files and store in curent directory
            try :
                RemoteFTPFiles(host, port, username, password, dataFiles[0], remotePath, localPath)
                RemoteFTPFiles(host, port, username, password, dataFiles[1], remotePath, localPath)
                RemoteFTPFiles(host, port, username, password, dataFiles[2], remotePath, localPath)

            except:
                None

            else:
                # Load and set data in array for all indentors
                TotalU2[index]  = np.array(np.loadtxt(dataFiles[0], delimiter=","))
                TotalRF[index]  = np.array(np.loadtxt(dataFiles[1], delimiter=","))
                NrackPos[index] =  np.array(np.loadtxt(dataFiles[2], delimiter=","))

    return variables, TotalU2, TotalRF, NrackPos
```

Post-Processing Functions

Function for postprocessing ABAQUS simulation data, loading variables from files in current directory and process data from simulation in U2/RF files. Process data from scan position to include full data range over all scan positions. Alongside, function to plot and visualise data. Then, calculates contours/z heights of constant force in simulation data for given threshold force and visualise. Produce data analysis for simulation data.

Data Plot

Function to produces scatter plot of indentation depth and reaction force to visualise and check simulation data.

In [ ]:
```
def DataPlot(NrackPos, TotalU2, TotalRF, Nb, Nt, n):
    '''
    Produces scatter plot of indentation depth and reaction force to visualise and check simulation data.

        Parameters:
            NrackPos (arr) - Array of initial scan positions for each indenter [Ni, Nb, [x, z] ]
            TotalU2 (arr)  - Array of indentors z displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
            TotalRF (arr)  - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
            Nb (int)       - Number of scan positions along x axis of base
            Nt(int)        - Number of frames in  ABAQUS simulation/ time step
            n (int)        - Index of indenter data to plot corresponding to indices in indenterRadius

    '''
    # Force Curves for all the data
    fig, ax = plt.subplots(1,1)
    for i in range(len(TotalRF)):
        ax.plot(TotalU2[i],TotalRF[i])


    # Initialise array for indentor force and displacement
    tipPos   = np.zeros([Nb*Nt, 2])
    tipForce = np.zeros(Nb*Nt)

    # Initialise count
    k = 0

    # Loop over array indices
    for i in range(Nb):
        for j in range( Nt ):
            #  Set array values for tip force and displacement
            tipPos[k]   = [NrackPos[n,i,0], NrackPos[n,i,1] + TotalU2[n,i,j]]
            tipForce[k] = TotalRF[n,i,j]

            # Count array index
            k += 1

    # Scatter plot indentor displacement over scan positions
    fig ,ax = plt.subplots(1,1)
    ax.plot(tipPos[:,0], tipPos[:,1], '.')

    ax.set_xlabel(r'x (nm)', labelpad = 25)
    ax.set_ylabel(r'y (nm)', labelpad = 25)
    ax.set_title('Tip Position for Raster Scan')
    plt.show()


    # Scatter plot of force over scan positions
    fig, ax = plt.subplots(1,1)
    ax.plot(tipPos[:,0], tipForce, '.')

    ax.set_xlabel(r'x (nm)', labelpad = 25)
    ax.set_ylabel('F (pN)',labelpad = 25)
    ax.set_title('Force Scatter Plot for Raster Scan')
    plt.show()
```

AFM Image Function

Calculate contours/z heights of constant force in simulation data for given threshold force and visualise.Function to load variables from fil~es in current directory and process data from simulation in U2/RF files

In [ ]:
```
def ForceGrid2D(X, Z, U2, RF, rackPos, courseGrain):
    '''
    Function to produce force heat map over scan domain.
```

```
        Parameters:
            X (arr)             - 1D array of postions over x domain of scan positions
            Z (arr)             - 1D array of postions over z domain of scan positions, discretised into bins of courseGrain value
            U2 (arr)            - Array of indentors y indentor position over scan ( As opposed to displacement into surface given from simulation and used elsewhere)
            RF (arr)            - Array of reaction force on indentor reference point
            rackPos (arr)       - Array of coordinates (x,z) of scan positions to image biomolecule [Nb,[x,z]]
            courseGrain (float) - Width of bins that subdivid xz domain of raster scanning/ spacing of the positions sampled over

        Return:
            forceGrid (arr)        - 2D Array of force heatmap over xz domain of scan i.e. grid of xz positions with associated force [Nx,Nz]
            forceGridmask (arr)    - 2D boolean array giving mask for force grid with exclude postions with no indentation data [Nx,Nz]
    '''
    #  ----------------------------------------------------Force Grid calculation-----------------------------------------------------------
    # Intialise force grid array
    forceGrid = np.zeros([len(X),len(Z)])

    # For all x and y coordinates in course grained/binned domain
    for i in range(len(X)):
        for j in range(len(Z)):

            # For each indentation coordinate
            for k in range(U2.shape[1]):

                # Set corresponding forcee grid array value to force value for that position
                if U2[i,k] == Z[j]:
                    # Set y values for corresponding x position in scan.
                    forceGrid[i,j] = RF[i,k]


    #  ----------------------------------------------------Create Force Grid mask-----------------------------------------------------------
    # Initialise mask array, 0 values include 1 excludes
    forceGridmask = np.zeros([len(X),len(Z)])

    # For scan positions in force array/ same as positions in X array
    for i in range(len(RF)):

        # Check how maNz non-zero values there are for each postion
        k = [ k for k,v in enumerate(forceGrid[i]) if v != 0]

        # If there are non zero values
        if len(k)!=0:
            # Mask all grid values upto the first non zero force value position
            for j in range(k[0]):
                forceGridmask[i,j] = 1

        # If all force values are zero
        else:
            # Mask all y positions in force grid for those forces
            k = [ k for k,v in enumerate(forceGrid[i]) if Z[k] == U2[i,0] ]
            for j in range(k[0]):
                forceGridmask[i,j] = 1


    return forceGrid, forceGridmask
```

In [ ]:
```
def ForceContour2D(U2, RF, rackPos, forceRef):
    '''
    Function to calculate contours/z heights of constant force in simulation data for given threshold force.

        Parameters:
            U2 (arr)            - Array of indentors y indentor position over scan ( As opposed to displacement into surface given from simulation and used elsewhere)
            RF (arr)            - Array of reaction force on indentor reference point
            rackPos (arr)       - Array of coordinates (x,z) of scan positions to image biomolecule [Nb,[x,z]]
            forceRef (float)    - Threshold force to evaluate indentation contours at (pN)

        Return:
            forceContour (arr)     - 2D Array of coordinates for contours of constant force given by reference force across scan positons
            forceContourmask (arr) - 2D boolean array giving mask for force contour for zero values in which no reference force
    '''
    #  ----------------------------------------------------Force Contour Calculation-----------------------------------------------------------
    # Initialise arrays
    forceContour = np.zeros([len(RF),2])
    forceContourmask = np.zeros([len(RF), 2])

    # For scan positions in force array/ same as positions in X array
    for i in range(len(RF)):

        # If maximum at this position is greater than Reference force
        if np.max(RF[i]) >= forceRef:

            # Return index at which force is greater than force threshold
            j = [ k for k,v in enumerate(RF[i]) if v >= forceRef][0]

            # Store corrsponding depth/ Y position and X position for the index
            forceContour[i] = np.array([ rackPos[i,0], U2[i,j] ])

        else:
            # Otherwise position not above force reference, therefore set mask values to 1
            forceContourmask[i] = np.ones(2)

    return forceContour, forceContourmask
```

Hertzian Force Interpolation over grid

In [ ]:
```
def F_Hertz(U, E, rIndentor, elasticProperties):
    R_eff = rIndentor
    v = elasticProperties[1]
    return (2/3) * (E/(1-v**2)) * np.sqrt(R_eff) * U**(3/2)
```

In [ ]:
```
def ForceInterpolation(Xgrid, Zgrid, U2, RF, rackPos, rIndentor, elasticProperties):
    '''
    Calculate a 2D force heatmap over the xz domain, produced from interpolated forces using Hertz model.

        Parameters:
            Xgrid (arr)             - 2D array/ grid of postions over xz domain of scan positions
            Zgrid (arr)             - 2D array/ grid of postions over xz domain of scan positions
            U2 (arr)                - Array of indentors y displacement in time over scan position and  for one indenter [Ni, Nb, Nt]
            RF (arr)                - Array of reaction force in time on indentor reference point over scan position  and for one indenter [Nb, Nt]
            rackPos (arr)           - Array of initial scan positions for one indenter [Nb, [x, z]]
            rIndentor (float)       - Indentor radius of spherical tip portion varied for seperate  simulations
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]

        Return:
            E_hertz (arr)      - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
            F (arr)            - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
    '''
    # Initialise array to hold elastic modulus
    E_hertz = np.zeros(len(rackPos))

    # Fit Hertz equation to force/indentation for each x scan positon, use lambda function to pass constant parameters(rIndentor/ elasticProperties )
    for i, value in enumerate(rackPos):
        u2, rf        = abs(U2[i]), abs(RF[i])
        popt, pcov    = curve_fit(lambda x, E: F_Hertz(x, E, rIndentor, elasticProperties), u2, rf)

        # Produce array of fitted elastic modulus over scan positions for each indentor
        E_hertz[i]  = popt
```

```
    # Use Elastic modulus over scan position to produce continous spline
    ESpline = UnivariateSpline(rackPos[:,0],  E_hertz, s=2)
    # From spline interpolate youngs modulus over x domain
    E = ESpline(Xgrid)

    # Create spline for initial scan positions
    rackSpline = UnivariateSpline(rackPos[:,0], rackPos[:,1], s = 0.001)
    # Calculate initial scan positions of x domain using scan position spline
    Zinit = rackSpline(Xgrid)

    # Use Hertz Eq to interpolate force over xz grid: (Yinit-Ygrid) gives indentation depths over grid
    F = F_Hertz(Zinit - Zgrid, E, rIndentor, elasticProperties)

    return F, E_hertz
```

Fourier, FWHM and Volume

$$g(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos\left(\frac{2\pi mt}{T}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi nt}{T}\right) \quad g(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos\left(\frac{2\pi mt}{T}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi nt}{T}\right)$$

$$= \sum_{m=0}^{\infty} a_m \cos\left(\frac{2\pi mt}{T}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi nt}{T}\right) \qquad = \sum_{m=0}^{\infty} a_m \cos\left(\frac{2\pi mt}{T}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi nt}{T}\right)$$

In [ ]:
```
def Fourier(x, waveDims, *a):
    '''Function to calculate Fourier Series for array of coefficence a'''
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    fs = waveAmplitude/2*np.copy(x)**0

    for k in range(len(a)):
        fs += a[k]*np.cos((2*np.pi*k*x)/waveLength)

    return fs
```

In [ ]:
```
def FWHM_Volume_Fourier(forceContour, NrackPos, X0, Nf, Ni, indentorRadius,  waveDims):
    '''
    Calculate Fourier series components, Full Width Half Maxima and Volume for Force Contours of varying reference force using splines

        Parameters:
            forceContour (arr)      - 2D Array of coordinates for contours of constant force given by reference force across scan positons
                                      for all indentor and reference force [Nf,Ni, Nb, [x,z]] (With mask applied).
            NrackPos (arr)          - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
            X0 (arr)                - Array of x positions along the scan
            Nf                      - Number if reference force values
            Ni                      - Number if indentor radii/ values
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [wavelength, amplitude, width, Number of oscilations/ groups in wave]

        Return:
            FWHM (arr)          - Array of full width half maxima of force contour for corresponding indentor and reference force [Nf,Ni]
            Volume (arr)        - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
            A (arr)             - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
    '''

   #  ----------------------------------------------Calculate Volume and Fourier-----------------------------------------------------
    # Intialise arrays for to store volume, FWHM and Fourier Series component A, for each indentor size and reference forces
    FWHM, Volume, A = np.zeros([Nf+1, Ni]), np.zeros([Nf+1, Ni]), np.zeros([Nf+1, Ni, Nmax])


    # Loop  for each indentor size and reference forces, using contours to assess volume
    for n in range(Ni):
        # --------------------Set first values as hardsphere boundary for each fwhm and volume -------------------------------------
        Fx, Fz = NrackPos[n,:,0], NrackPos[n,:,1]
        # Connect contour points smoothly with a spline
        forceSpline = UnivariateSpline(Fx, Fz, s = 0.01)

        A[0,n], pcov = curve_fit(lambda x, *a: Fourier(x, waveDims, *a), X0, forceSpline(X0), p0 =tuple(np.zeros(Nmax)))
        Volume[0,n] = forceSpline.integral(-waveDims[0]/2, 0)
        FWHM[0,n] = abs( -waveLength/2 - UnivariateSpline(Fx, Fz - (Fz.min() + Fz.max())/2, s = 1).roots()[0])


        # ------------------------------------Set fwhm and volume values for each force contour-----------------------------------
        for m  in range(Nf):

            # Extract xz compontents of force contour - compressed removees masked values
            Fx, Fz = forceContour[m,n,:,0].compressed(), forceContour[m,n,:,1].compressed()

            # Use try loop to avoid error for contours that cannot produce splines
            try:
                # Half maxima can be calculated by finding roots of spline that is tanslated vertically so half beneath x axis
                FWHM[m,n] = abs( -waveLength/2 - UnivariateSpline(Fx, Fz - (Fz.min() + Fz.max())/2, s = 1).roots()[0])
            except:
                None

            try:
                # Connect contour points smoothly with a spline, can fail, use try to avoid code breaking
                forceSpline = UnivariateSpline(Fx, Fz-Fz.min(), s = 0.01)
            except:
                None
            else:
                # Volume can be found by integrating contour spline over bounds
                Volume[m,n] = forceSpline.integral(-waveDims[0]/2, 0)
                # Calculate Fourier components
                A[m+1,n], pcov = curve_fit(lambda x, *a: Fourier(x, waveDims, *a), X0, forceSpline(X0)+Fz.min(), p0 =tuple(np.zeros(Nmax)))

    return  FWHM, Volume, A
```

Postprocessing

In [ ]:
```
def Postprocessing(TotalU2, TotalRF, NrackPos, Nb, Nmax, courseGrain, refForces, indentorRadius, waveDims, elasticProperties, **kwargs):
    '''
    Calculate a 2D force heatmap produced from simulation over the xz domain.

        Parameters:
            TotalU2 (arr)           - Array of indentors y displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
            TotalRF (arr)           - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
            NrackPos (arr)          - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
            Nb (int)                - Number of scan positions along x axis of base
            Nmax (int)              - Maximum number of terms in fourier series of force contour
            courseGrain (float)     - Width of bins that subdivid xz domain of raster scanning/ spacing of the positions sampled over
            refForces (arr)         - Array of threshold force to evaluate indentation contours at (pN)
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [wavelength, amplitude, width, Number of oscilations/ groups in wave]
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]

        Return:
            X (arr)             - 1D array of postions over x domain of scan positions
            Z (arr)             - 1D array of postions over z domain of scan positions, discretised into bins of courseGrain value
```

```
            forceGrid (arr)    - 2D Array of force heatmap over xz domain of scan i.e. grid of xz positions with associated force
                                 for all indentors and reference force [Nf, Ni, Nb, Nz] (With mask applied).
            forceContour (arr) - 2D Array of coordinates for contours of constant force given by reference force across scan positons
                                 for all indentor and reference force [Nf,Ni, Nb, [x,z]] (With mask applied).
            FWHM (arr)         - Array of full width half maxima of force contour for corresponding indentor and reference force [Nf,Ni]
            Volume (arr)       - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
            A (arr)            - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
            E_hertz (arr)      - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
            F (arr)            - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
    '''
    #  ----------------------------------------Initialise  Variables for force grid-----------------------------------------------
    Nf = len(refForces)
    Ni = len(indentorRadius)

    # Convert indentation data to indentor Y displacement and discretise values into bins of width given by course grain value
    zIndentor = (TotalU2 + NrackPos[:,:,1, None])
    U2 = courseGrain*np.round(zIndentor/courseGrain)

    # Set X arrays of scan positions
    X = NrackPos[0,:,0]

    # Produce Y arrays over all Y domain of indentor position for all indentors
    Z = np.round( np.arange( U2.min(initial=0), U2.max() + courseGrain, courseGrain )*10)/10


    #  ------------------------------------------------------Set force grid and force contour----------------------------------------
    # Initialise arrays for all indentor size and reference forces
    forceContour, forceContourmask = np.zeros([Nf, Ni , Nb, 2]),     np.zeros([Nf, Ni , Nb, 2])
    forceGrid,    forceGridmask    = np.zeros([Nf, Ni, Nb, len(Z)]), np.zeros([Nf, Ni, Nb, len(Z)])

    # Set force grid and force contour for each indentor and refence force
    for m  in range(Nf):
        for n in range(Ni):
            forceGrid[m,n],    forceGridmask[m,n]    = ForceGrid2D(    X, Z, U2[n], TotalRF[n], NrackPos[n], courseGrain)
            forceContour[m,n], forceContourmask[m,n] = ForceContour2D(zIndentor[n], TotalRF[n], NrackPos[n], refForces[m])


    # Mask force grid excluding postions with no indentation data [Nx,Nz] and mask force contour for zero values in which below reference force
    forceGrid    = np.ma.masked_array(forceGrid, mask=forceGridmask)
    forceContour = np.ma.masked_array(forceContour, mask=forceContourmask)


    #  ---------------------------------------Calculate Hertz fit and interpolate force from the fit--------------------------------
    # Initialise grid arrays over xz domain
    X0 = np.linspace(-waveDims[0]/2, 0, 250)
    Z0 = np.linspace(Z[0], waveDims[1], 250)
    Xgrid, Zgrid = np.meshgrid(X0,Z0)

    # Initialise array holding Fitted Elastic modulus and Interpolated force
    E_hertz   = np.zeros([Ni,Nb])
    F = np.zeros([Ni, len(X0), len(Z0)])

    # For each indentor calculate interpolated force heat maps
    for n, rIndentor in enumerate(indentorRadius):
        F[n], E_hertz[n] = ForceInterpolation(Xgrid, Zgrid, TotalU2[n], TotalRF[n], NrackPos[n], rIndentor, elasticProperties)


   #  ------------------------------------------------Calculate Volume and Fourier-------------------------------------------------------
    # Initialise arrays for to store volume, FWHM and Fourier Series component A, for each indentor size and reference forces
    FWHM, Volume, A =  FWHM_Volume_Fourier(forceContour, NrackPos, X0, Nf, Ni, indentorRadius,  waveDims)

    # Mask values equal to zero
    FWHM   = np.ma.masked_equal(FWHM, 0)
    Volume = np.ma.masked_equal(Volume, 0)

    return X, Z, forceContour, forceGrid, Volume, FWHM, A, E_hertz, F
```

Simulation Function

Final simulation function

In [ ]:

```
def AFMSimulation(host, port, username, password, scratch, wrkDir, localPath, abqCommand, fileName, subData,
                  indentorType, indentorRadius, theta_degrees, tip_length, indentionDepths, waveDims,
                  refForces, courseGrain, Nmax, binSize, clearance, meshSurface, meshIndentor,
                  timePeriod, timeInterval, elasticProperties, **kwargs):
    '''
    Final function to automate simulation. User inputs all variables and all results are outputted. The user gets a optionally get a surface plot of scan positions.
    Produces a heatmap of the AFM image, and 3D plots of the sample surface for given force threshold.

        Parameters:
            host (str)              - Hostname of the server to connect to
            port (int)              - Server port to connect to
            username (str)          - Username to authenticate as (defaults to the current local username)        -
            password (str)          - password (str) – Used for password authentication; is also used for private key decryption if passphrase is not given.
            scratch                 - Path to remote scratch directory
            wrkDir (str)            - Working directory extension
            localPath (str)         - Path to local file/directory
            abqCommand (str)        - Abaqus command to execute and run script
            fileName (str)          - Base File name for abaqus input files
            subData (str)           - Data for submission to serve queue [walltime, memory, cpus]

            indentorType (str)      - String defining indentor type (Spherical or Capped)
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            theta_degrees (float)   - Principle conical angle from z axis in degrees
            tip_length (float)      - Total cone height
            indentionDepths (arr)   - Array of maximum indentation depth into surface
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [wavelength, amplitude, width, Number of oscilations/ groups in wave]

            refForces (float)       - Threshold force to evaluate indentation contours at, mimics feedback force in AFM (pN)
            courseGrain (float)     - Width of bins that subdivid xz domain of raster scanning/ spacing of the positions sampled over
            Nmax (int)              - Maximum number of terms in fourier series of force contour
            binSize (float)         - Width of bins that subdivid xz domain during raster scanning/ spacing of the positions sampled over
            clearance (float)       - Clearance above molecules surface indentor is set to during scan
            meshSurface (float)     - Value of indentor mesh given as bin size for vertices of geometry in Angstrom (x10-10 m)
            meshIndentor (float)    - Value of indentor mesh given as bin size for vertices of geometry in Angstrom (x10-10 m)
            timePeriod(float)       - Total time length for ABAQUS simulation/ time step (T)
            timeInterval(float)     - Time steps data sampled over for ABAQUS simulation/ time step (dt)
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]

            kwargs:
                Submission ('serial'/ 'paralell') - Type of submission, submit pararlell scripts or single serial script for scan locations {Default: 'serial'}
                Main (bool)         - If false skip preprocessing step of simulation {Default: True}
                SurfacePlot (bool) - If false skip surface plot of biomolecule and scan positions, set as indenter radius you wish to plot {Default: False}
                Queue (bool)       - If false skip queue completion step of simulation {Default: True}
                Analysis (bool)    - If false skip odb analysis step of simulation {Default: True}
                Retrieval (bool)   - If false skip data file retrivial from remote serve {Default: True}
                Compile(int)       - If passed, simulation data is compiled from seperate sets of simulations in directory in remote server to combine
                                     complete indentations. Value is set as int representing the range of directories to compile from (directories must have
                                     same root naming convention with int denoting individual directories)                    -
                Postprocess (bool) - If false skip postprocessing step to produce AFM image from data {Default: True}
                DataPlot (bool)    - If false skip scatter plot of simulation data {Default: True}
                Symmetric          - If false skip postprocessing step to produce AFM image from data {Default: True}

        Returns:
            X (arr)             - 1D array of postions over x domain of scan positions, discretised into bins of courseGrain value [Nx]
            Z (arr)             - 1D array of postions over z domain of scan positions, discretised into bins of courseGrain value [Nz]
            TotalU2 (arr)       - Array of indentors z displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
```

```
        TotalRF (arr)      - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
        NrackPos (arr)     - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
        forceGrid (arr)    - 2D Array of force heatmap over xz domain of scan i.e. grid of xz positions with associated force [Nx,Nz] (With mask applied).
        forceContour (arr) - 2D Array of coordinates for contours of constant force given by reference force across scan positons (With mask applied).
        FWHM (arr)         - Array of full width half maxima of force contour for corresponding indentor and reference force [Nf,Ni]
        Volume (arr)       - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
        A (arr)            - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
        E_hertz (arr)      - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
        F (arr)            - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
'''
# Set intial time
T0 = time.time()

#  ------------------------------------------------------Main Simulations-------------------------------------------------------
if 'Main' not in kwargs.keys() or kwargs['Main'] == True:
    t0 = time.time()

    # For each indentor radius prdoduce main simulation and submit
    for index, rIndentor in enumerate(indentorRadius):
        print('Indentor Radius - ', rIndentor)

        #  ---------------------------------------Raster Scan Positions -----------------------------------------------------
        # Calculate tip geometry to create indentor and calculate scan positions over molecule for imaging
        indentionDepth = clearance + indentionDepths[index]

        # Set tip dimensions
        tipDims = TipStructure(rIndentor, theta_degrees, tip_length)

        # Set surface dimensions
        waveLength, waveAmplitude, waveWidth, groupNum = waveDims
        Nw = 70 # int(15*groupNum+1)
        wavePos      = np.zeros([Nw, 2])
        wavePos[:,0] = np.linspace(-waveLength*groupNum/2, waveLength*groupNum/2, Nw )
        wavePos[:,1] = waveSin(wavePos[:,0], waveDims)

        Nb, Nt = int((waveLength/2)/(binSize) + 1), int(timePeriod/ timeInterval)+1

        # Calculate scan positions
        rackPos = ScanGeometry(indentorType, tipDims, waveDims, Nb, clearance)

        # Option plot for scan positions for visualisation
        if 'SurfacePlot' in kwargs.keys() and kwargs['SurfacePlot'] == rIndentor:
            SurfacePlot(rackPos, Nb, waveDims, wavePos, tipDims, binSize, clearance)


        #  ---------------------------------------------Export Variable-----------------------------------------------------
        # Set list of simulation variables and export to current directory
        variables = [timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface]
        ExportVariables(rackPos, variables, waveDims, wavePos, tipDims, indentorType, elasticProperties)


        #  ---------------------------------------------Remote Submission---------------------------------------------------
        remotePath = scratch + wrkDir +'/IndenterRadius'+str(int(rIndentor))

        abqfiles = ('AFMtestRasterScan.py','AFMtestODBAnalysis.py')
        csvfiles = ( "rackPos.csv", "variables.csv", "waveDims.csv", "wavePos.csv", "tipDims.csv", "indentorType.txt", "elasticProperties.csv")

        RemoteSubmission(host, port, username, password, remotePath, localPath,  csvfiles, abqfiles, abqCommand, fileName, subData, rackPos, **kwargs)

    t1 = time.time()
    print('Main Submission Complete - ' + str(timedelta(seconds=t1-t0)) + '\n')


#  ------------------------------------------------------Queue Status-----------------------------------------------------------
if 'Queue' not in kwargs.keys() or kwargs['Queue'] == True:
    t0 = time.time()
    print('Simulations Processing ...')

    # Wait for completion when queue is empty
    QueueCompletion(host, port, username, password)

    t1 = time.time()
    print('ABAQUS Simulation Complete - '+ str(timedelta(seconds=t1-t0)) + '\n' )


#  ---------------------------------------------ODB Analysis Submission-------------------------------------------------------
if 'Analysis' not in kwargs.keys() or kwargs['Analysis'] == True:
    t0 = time.time()
    print('Running ODB Analysis...')


    # For each indentor radius
    for index, rIndentor in enumerate(indentorRadius):
        print('Indentor Radius:', rIndentor)

        # ODB analysis script to run, extracts data from simulation and sets it in csv file on server
        script     = 'AFMtestODBAnalysis.py'
        remotePath = scratch + wrkDir + '/IndenterRadius'+str(int(rIndentor))

        RemoteCommand(host, port, username, password, script, remotePath, abqCommand)

    t1 = time.time()
    print('ODB Analysis Complete - ' + str(timedelta(seconds=t1-t0)) + '\n' )


#  ---------------------------------------------File Retrieval----------------------------------------------------------------
if 'Retrieval' not in kwargs.keys() or kwargs['Retrieval'] == True:
    t0 = time.time()
    print('Running File Retrieval...')

    # Retrieve variables used for given simulation (in case variables redefined when skip kwargs used)
    dataFiles = ('U2_Results.csv','RF_Results.csv', 'rackPos.csv')
    csvfiles  = ( "rackPos.csv", "variables.csv","waveDims.csv", "tipDims.csv")

    variables, TotalU2, TotalRF, NrackPos = DataRetrieval(host, port, username, password, scratch, wrkDir, localPath, csvfiles, dataFiles, indentorRadius, **kwargs)

    # Export simulation data so it is saved in current directory for future use (save as a 2d array instead of 3d)
    np.savetxt("variables.csv", variables, fmt='%s', delimiter=",")
    np.savetxt("TotalU2.csv", TotalU2.reshape(TotalU2.shape[0], -1), fmt='%s', delimiter=",")
    np.savetxt("TotalRF.csv", TotalRF.reshape(TotalRF.shape[0], -1), fmt='%s', delimiter=",")
    np.savetxt("NrackPos.csv", NrackPos.reshape(NrackPos.shape[0], -1), fmt='%s', delimiter=",")

    t1 = time.time()
    print('File Retrevial Complete' + '\n')


#  ------------------------------------------------- Post-Processing-------------------------------------------------------------
if  'Postprocess' not in kwargs.keys() or kwargs['Postprocess'] == True:

    t0 = time.time()
    print('Running Postprocessing...')

    # Check if simulation files are accessible in curent directory to use if pre=processing skipped
    try:
        variables, waveDims, rackPos  = ImportVariables()
        waveLength, waveAmplitude, waveWidth, groupNum = waveDims
        timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface = variables
        Nb, Nt = int((waveLength/2)/(binSize) + 1), int(timePeriod/ timeInterval)+1
```

```
        # Load saved simulation data and reshape as data sved as 2d array,  true shape is 3d
        TotalU2  = np.array(np.loadtxt('TotalU2.csv', delimiter=",")).reshape(len(indentorRadius), Nb, Nt)
        TotalRF  = np.array(np.loadtxt('TotalRF.csv', delimiter=",")).reshape(len(indentorRadius), Nb, Nt)
        NrackPos = np.array(np.loadtxt('NrackPos.csv', delimiter=",")).reshape(len(indentorRadius), Nb, 2)

    # If file missing prompt user to import/ produce files
    except:
        print('No Simulation files available, run preprocessing or import data' + '\n')


    #  ------------------------------------------------- Data-Processing-----------------------------------------------
    # Visualise data if set in kwargs
    if 'DataPlot' in kwargs.keys():
        n =  kwargs['DataPlot']
        DataPlot(NrackPos, TotalU2, TotalRF, Nb, Nt, n)

    # Process simulation data to produce heat map, analyse force contours, full width half maximum, volume and youngs modulus
    X, Z, forceContour, forceGrid, Volume, FWHM, A, E_hertz, F = Postprocessing(TotalU2, TotalRF, NrackPos, Nb, Nmax, courseGrain, refForces, indentorRadius,
                                                                    waveDims, elasticProperties, **kwargs)

    t1 = time.time()
    print('Postprocessing Complete' + '\n')


    # Return final time of simulation and variables
    T1 = time.time()
    print('Simulation Complete - ' + str(timedelta(seconds=T1-T0)) )
    return X, Z, TotalU2, TotalRF, NrackPos, forceContour, forceGrid, Volume, FWHM, A, E_hertz, F

else:
    # Return final time of simulation
    T1 = time.time()
    print('Simulation Complete - ' + str(timedelta(seconds=T1-T0)) )
    return None, None, None, None, None, None, None, None, None, None
```

Plot Functions

Code to plot results of the simulation

Illustrative Surface Plot

In [ ]:

```
def SurfacePlot(rackPos, Nb, waveDims, wavePos, tipDims, binSize, clearance):
    '''
    Plot the surfaces and scan positions to visualise and check positions.

        Parameters:
            rackPos (arr)      - Array of coordinates [x,z] of scan positions to image biomolecule
            Nb (int)           - Number of scan positions along x axis of base
            waveDims (list)    - Geometric parameters for defining base/ substrate structure [Wavelength, Amplitude, Width, Number of oscilations/ groups in wave ]
            wavePos            - Positions on wave used to define spline in ABAQUS
            tipDims (list)     - Geometric parameters for defining capped tip structure
            clearance (float)  - Clearance above molecules surface indentor is set to during scan
    '''

    #  ------------------------------------------------Raster Scan positions ----------------------------------------------------
    # Set tip and wave dimensional variables
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    # Raster scan grid positions for spherical and capped tip
    rackPosSphere = ScanGeometry('Spherical', tipDims, waveDims, Nb, clearance)
    rackPosCone = ScanGeometry('Capped', tipDims, waveDims, Nb, clearance)

    # Set index for scan position to plot tip at
    i,j = 0,-7

    # Produce array for the x extent
    x = np.linspace(-waveLength, 0, 1000)
    X = np.linspace(-r_top, r_top, 1000)

    # Produce spherical tip with polar coordinates
    x1 = rIndentor*np.cos(np.linspace(-np.pi, np.pi, 1000))
    y1 = rIndentor*np.sin(np.linspace(-np.pi, np.pi, 1000))


    #  ------------------------------------------------Plot Points ----------------------------------------------------------
    # Create figure for scan positions
    fig, ax = plt.subplots(1,1, figsize= (11.69/3, 8.27/3 ))

    # Plot spherical and conical scan position as points
    ax.plot(rackPosCone[:,0], rackPosCone[:,1], '.', color = 'r')
    # Plot lines for wave base and points used to define it in abaqus wavePos
    ax.plot(x, waveSin(x, waveDims), 'k')

    # Plot the geometry of spherical and conical tip at index i
    ax.plot(X+rackPosCone[i,0], Fconical(X, 0, r_int, z_int, theta, rIndentor, tip_length)+rackPosCone[i,1], color=  '#5a71ad')
    ax.plot(x1+rackPosCone[i,0], y1+rIndentor+rackPosCone[i,1], ':', color = '#fc8535')

    # Plot the geometry of spherical and conical tip at index j
    ax.plot(X+rackPosCone[j,0], Fconical(X, 0, r_int, z_int, theta, rIndentor, 8)+rackPosCone[j,1], color=  '#5a71ad')
    ax.plot(x1+rackPosCone[j,0], y1+rIndentor+rackPosCone[j,1], ':', color = '#fc8535')

    # Set axis labels to create desired layout
    ax.set_xlabel(r'$x/\lambda$', labelpad = 5, color='white')
    # ax.set_ylabel(r'$z/\lambda$', rotation = 10, labelpad = 0,   color='white')
    ax.set_xlim(-waveLength, 0)
    ax.set_ylim(-1, ((2)*waveAmplitude))
    ax.axes.set_aspect('equal')

    # Place axis on right for desired spacing
    ax.yaxis.set_label_position("right")
    ax.yaxis.tick_right()

    # Make axis invisible
    ax.spines['top'].set_color('white')
    ax.spines['bottom'].set_color('white')
    ax.spines['left'].set_color('white')
    ax.spines['right'].set_color('white')
    ax.tick_params(colors='white')

    # Annotating Diagram
    ax.text(-waveLength/2-2, waveAmplitude + rIndentor/2, 'R', color = '#fc8535')
    ax.annotate('', color = '#fc8535',
                xy=(-waveLength/2, waveAmplitude), xycoords='data',
                xytext=(-waveLength/2, waveAmplitude+rIndentor), textcoords='data',
                arrowprops=dict(arrowstyle="-", connectionstyle="arc3", color='#fc8535'))

    ax.text(-waveLength/2, 0, r'$\lambda$')
    ax.annotate('', xy=(-waveLength, -0.5), xycoords='data',
                xytext=(0, -0.5), textcoords='data',
                arrowprops=dict(arrowstyle="<->", connectionstyle="arc3", color='k'))

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-SetUp.png', bbox_inches = 'tight')
```

```
    plt.show()
```

Contour Plot

Interpolate

In [ ]:
```
def ContourPlot(X, Z, rackPos, forceGrid, forceContour, refForce, clearance, A, N, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force heatmap produced from simulation over the xz domain for single indenter and refereance force.

        Parameters:
            X (arr)                 - 1D array of x coordinates over scan positions
            Z (arr)                 - 1D array of z coordinates over scan positions
            rackPos (arr)           - Array of initial scan positions for indenter [Nb, [x, z] ]
            forceGrid (arr)         - 2D Array of force grid of xz positions
            forceContour( arr)      - 2D Array of coordinates for contours of constant force given by reference force
            refForce (float)        - Threshold force to evaluate indentation contours at
            clearance(float)        - Clearance above molecules surface indentor is set to during scan
            A (arr)                 - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
            N (int)                 - Number of fourier series terms included in fit
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)          - Geometric parameters for defining capped tip structure
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)        - Normalisation of cmap
            maxRF (float)           - Maximum Force value
            contrast (float)        - Contrast between high and low values in AFM heat map (0-1)
    '''

    #  ------------------------------------------------------Set Variable------------------------------------------------------
    # Set material properties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)

    # Tip variables
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
    # Surface variables
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    # Extract xz compontents of force contour - compressed removes masked values
    Fx, Fz = np.array(forceContour[:,0].compressed()), np.array(forceContour[:,1].compressed())
    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)


    # Increase padding to add above surface
    hPadding = 1
    # Produce spherical tip with polar coordinates
    x = np.linspace(-waveLength/2,waveLength/2, 100)


    #  ------------------------------------------------------2D Plots------------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3))


    # 2D heat map plot without interpolation, append two together to produce whole wavelength
    im = ax.imshow(np.ma.append(forceGrid[::-1],forceGrid, axis=0).T/F_dim, origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm= normalizer,
                   extent = (-1/2, 1/2, Z[0]/waveLength, Z[-1]/waveLength), interpolation_stage = 'rgba')

    # Plot fourier series fit for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(x/waveLength, waveSin(x+waveLength/2, waveDims)/waveLength, ':',                  color = 'w', lw = 1, label = 'Surface boundary')
    ax.plot(x/waveLength, Fourier(x+waveLength/2, waveDims,*A[:N])/waveLength,                color = 'r', lw = 1, label = 'Fitted Contour')

    ax.plot((rackPos[:,0]+waveLength/2)/waveLength, (rackPos[:,1]-clearance)/waveLength, ':', color = 'r', lw = 1, label = 'Hard Sphere boundary')
    ax.plot(rackPos[:,0]/waveLength, (rackPos[:,1][::-1]-clearance)/waveLength,':',           color = 'r', lw = 1, label = 'Hard Sphere boundary')


    # Plot indentor geometry
    ax.plot((x)/waveLength, (Fconical(x, 0, r_int, z_int, theta, rIndentor, tip_length)+rackPos[0,1])/waveLength, color = 'w', lw = 1, label = 'Indentor boundary')

    # Add 0 values in image for region above surface
    ax.imshow(np.zeros([10,10]), origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm= normalizer,
              extent = (-1/2, 1/2, waveAmplitude/waveLength, ((1+hPadding)*waveAmplitude)/waveLength) )

    # Set legend and axis Labels, limits and title
    ax.set_xlabel(r'$\frac{x}{\lambda}$')
    ax.set_ylabel(r'$\frac{z}{\lambda}$', rotation=0,  labelpad = 15)
    ax.set_xlim(-1/2, 1/2)
    ax.set_ylim(Z[0]/waveLength, ((1+hPadding)*waveAmplitude)/waveLength)
    ax.set_facecolor("grey")
    ax.axes.set_aspect('equal')

    # ------------------------------------------------Plot color bar ------------------------------------------------------
    cbar= fig.colorbar(im, ax= ax , orientation = 'vertical', fraction=0.035, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-ContourPlot-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

No Interpolate

In [ ]:
```
def ContourPlotNI(X, Z, rackPos, forceGrid, forceContour, refForce, clearance, A, N, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force heatmap produced from simulation over the xz domain for single indenter and refereance force.

        Parameters:
            X (arr)                 - 1D array of x coordinates over scan positions
            Z (arr)                 - 1D array of z coordinates over scan positions
            rackPos (arr)           - Array of initial scan positions for indenter [Nb, [x, z] ]
            forceGrid (arr)         - 2D Array of force grid of xz positions
            forceContour( arr)      - 2D Array of coordinates for contours of constant force given by reference force
            refForce (float)        - Threshold force to evaluate indentation contours at
            clearance(float)        - Clearance above molecules surface indentor is set to during scan
            A (arr)                 - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
            N (int)                 - Number of fourier series terms included in fit
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)          - Geometric parameters for defining capped tip structure
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)        - Normalisation of cmap
            maxRF (float)           - Maximum Force value
            contrast (float)        - Contrast between high and low values in AFM heat map (0-1)
    '''

    #  ------------------------------------------------------Set Variable------------------------------------------------------
    # Set material properties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)

    # Tip variables
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
    # Surface variables
```

```
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    # Extract xz compontents of force contour - compressed removes masked values
    Fx, Fz = np.array(forceContour[:,0].compressed()), np.array(forceContour[:,1].compressed())
    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)

    # Increase padding to add above surface
    hPadding = 1
    # Produce spherical tip with polar coordinates
    x = np.linspace(-waveLength/2,waveLength/2, 100)


    #  ------------------------------------------------------2D Plots-----------------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3))


    # 2D heat map plot without interpolation
    im = ax.imshow(np.ma.append(forceGrid[::-1],forceGrid, axis=0).T/F_dim, origin= 'lower', cmap='coolwarm', interpolation='none', norm= normalizer,
                   extent = (-1/2, 1/2, Z[0]/waveLength, Z[-1]/waveLength))

    # Plot fourier series fit for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(x/waveLength,            waveSin(x+waveLength/2, waveDims)/waveLength, ':',       color = 'w', lw = 1, label = 'Surface boundary')

    ax.plot((Fx+waveLength/2)/waveLength,  Fz/waveLength, 'x', ms = 3,                        color = 'k', lw = 1, label = 'Force contour for F= {0:.3f}'.format(refForce/F_dim))
    ax.plot(-(Fx+waveLength/2)/waveLength, Fz/waveLength, 'x', ms = 3,                        color = 'k', lw = 1, label = 'Force contour for F= {0:.3f}'.format(refForce/F_dim))

    ax.plot((rackPos[:,0]+waveLength/2)/waveLength, (rackPos[:,1]-clearance)/waveLength, ':', color = 'r', lw = 1, label = 'Hard Sphere boundary')
    ax.plot(rackPos[:,0]/waveLength, (rackPos[:,1][::-1]-clearance)/waveLength,':',          color = 'r', lw = 1, label = 'Hard Sphere boundary')


    # Plot indentor geometry
    ax.plot((x)/waveLength, (Fconical(x, 0, r_int, z_int, theta, rIndentor, tip_length)+rackPos[0,1])/waveLength, color = 'w', lw = 1, label = 'Indentor boundary')

    # Add 0 values in image for region above surface
    ax.imshow(np.zeros([10,10]), origin= 'lower', cmap='coolwarm', interpolation='none', norm= normalizer,
              extent = (-1/2, 1/2, waveAmplitude/waveLength, ((1+hPadding)*waveAmplitude)/waveLength) )

    # Set Legend and axis Labels, limits and title
    ax.set_xlabel(r'$\frac{x}{\lambda}$')
    ax.set_ylabel(r'$\frac{z}{\lambda}$', rotation=0,  labelpad = 15)
    ax.set_xlim(-1/2, 1/2)
    ax.set_ylim(Z[0]/waveLength, ((1+hPadding)*waveAmplitude)/waveLength)
    ax.set_facecolor("grey")
    ax.axes.set_aspect('equal')

    # ------------------------------------------Plot color bar ------------------------------------------------------------
    cbar= fig.colorbar(im, ax= ax , orientation = 'vertical', fraction=0.035, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-ContourPlotNI-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

Line

In [ ]:

```
def LineContourPlot(X, Z, rackPos, forceContour, refForces, clearance, A, N, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force contour lines produced from simulation over the xz domain for single indenter and range of reference force.

        Parameters:
            X (arr)                 - 1D array of x coordinates over scan positions
            Z (arr)                 - 1D array of z coordinates over scan positions
            RF(arr)                 - Array of reaction force on indentor reference point
            rackPos (arr)           - Array of initial scan positions for indenter [Nb, [x, z] ]
            forceContour( arr)      - 2D Array of coordinates for contours of constant force given by reference force
            refForces (float)       - Threshold force to evaluate indentation contours at (pN)
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            clearance(float)        - Clearance above molecules surface indentor is set to during scan
            A (arr)                 - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
            N (int)                 - Number of fourier series terms included in fit
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)          - Geometric parameters for defining capped tip structure
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)        - Normalisation of cmap
            maxRF (float)           - Maximum Force value
            contrast (float)        - Contrast between high and low values in AFM heat map (0-1)
    '''
    #  ------------------------------------------------------Set Variable-----------------------------------------------------------
    # Tip variables
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
    # Surface variables
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)

    cmap = mpl.cm.coolwarm



    # Produce spherical tip with polar coordinates
    x = np.linspace(-waveLength/2,waveLength/2, 100)
    # Increase padding to add above surface
    hPadding = 1

    #  ------------------------------------------------------2D Plots-----------------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3) )

    #  ------------------------------------------------------Interpolation-----------------------------------------------------------
    # Plot spline force for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(x/waveLength, waveSin(x+waveLength/2, waveDims)/waveLength, ':',                color = 'k',                lw = 2, label = 'Surface')
    ax.plot((rackPos[:,0]+waveLength/2)/waveLength, (rackPos[:,1]-clearance)/waveLength, ':', color = cmap(normalizer(0)), lw = 2, label = 'Hard\nSphere')
    ax.plot(rackPos[:,0]/waveLength, (rackPos[:,1][::-1]-clearance)/waveLength,':',          color = cmap(normalizer(0)), lw = 2)

    for m in range(len(refForces)):
        ax.plot(x/waveLength, Fourier(x+waveLength/2, waveDims,*A[m+1,:N])/waveLength,         color = cmap(normalizer(refForces[m]/F_dim)), lw = 1 )

    # Plot indentor geometry
    ax.plot((x)/waveLength, (Fconical(x, 0, r_int, z_int, theta, rIndentor, tip_length)+rackPos[0,1])/waveLength, color = 'k', lw = 1, label = 'Indentor')


    # Set Legend and axis Labels, limits and title
    ax.set_xlabel(r'$\frac{x}{\lambda}$')
    ax.set_ylabel(r'$\frac{z}{\lambda}$', rotation=0,  labelpad = 15)
    ax.set_xlim(-1/2, 1/2)
    ax.set_ylim(Z[0]/waveLength, ((1+hPadding)*waveAmplitude)/waveLength)
    ax.axes.set_aspect('equal')

    # ------------------------------------------Plot color bar ------------------------------------------------------------
```

```
    # plt.legend(frameon=False)
    cbar = plt.colorbar(plt.cm.ScalarMappable(cmap=cmap, norm=normalizer), fraction=0.035, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-LineContour-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

Force Interpolation Plot

In [ ]:
```
def FInterpolatePlot(X, Z, rackPos, F, clearance, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force heatmap interpolated from simulation over the xz domain.

        Parameters:
            X (arr)                 - 1D array of x coordinates over scan positions
            Z (arr)                 - 1D array of z coordinates over scan positions
            rackPos (arr)           - Array of initial scan positions for indenter [Nb, [x, z] ]
            F (arr)                 - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
            clearance(float)        - Clearance above molecules surface indentor is set to during scan
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)          - Geometric parameters for defining capped tip structure
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)        - Normalisation of cmap
            maxRF (float)           - Maximum Force value
            contrast (float)        - Contrast between high and low values in AFM heat map (0-1)
    '''

    #  ------------------------------------------------------Set Variable-----------------------------------------------------
    # Set material properties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)

    # Tip variables
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
    # Surface variables
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)


    # Increase padding to add above surface
    hPadding = 1

    # Produce spherical tip with polar coordinates
    x = np.linspace(-waveLength/2,waveLength/2, 100)


    #  ------------------------------------------------------2D Plots---------------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3))


    # 2D heat map plot without interpolation, append two together to produce whole wavelength
    im = ax.imshow(np.ma.append(F.T[::-1],F.T, axis=0).T/F_dim, origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm= normalizer,
                   extent = (-1/2, 1/2, Z[0]/waveLength, Z[-1]/waveLength), interpolation_stage = 'rgba')

    # Plot fourier series fit for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(x/waveLength, waveSin(x+waveLength/2, waveDims)/waveLength, ':',                 color = 'w', lw = 1, label = 'Surface boundary')
    ax.plot((rackPos[:,0]+waveLength/2)/waveLength, (rackPos[:,1]-clearance)/waveLength, ':', color = 'r', lw = 1, label = 'Hard Sphere boundary')
    ax.plot(rackPos[:,0]/waveLength, (rackPos[:,1][::-1]-clearance)/waveLength,':',          color = 'r', lw = 1, label = 'Hard Sphere boundary')


    # Plot indentor geometry
    ax.plot((x)/waveLength, (Fconical(x, 0, r_int, z_int, theta, rIndentor, tip_length)+rackPos[0,1])/waveLength, color = 'w', lw = 1, label = 'Indentor boundary')

    # Set legend and axis labels, limits and title
    ax.set_xlabel(r'$\frac{x}{\lambda}$')
    ax.set_ylabel(r'$\frac{z}{\lambda}$', rotation=0,  labelpad = 15)
    ax.set_xlim(-1/2, 1/2)
    ax.set_ylim(Z[0]/waveLength, ((1+hPadding)*waveAmplitude)/waveLength)
    ax.set_facecolor("grey")
    ax.axes.set_aspect('equal')

    # ------------------------------------------------Plot color bar ------------------------------------------------------------
    cbar= fig.colorbar(im, ax= ax , orientation = 'vertical', fraction=0.035, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-FInterpolate-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

Full Width Half Maxima

In [ ]:
```
def FWHMPlot(FWHM, indentorRadius, refForces, waveDims, elasticProperties):
    '''
    Function to plot Full Width Half Maxima of force contour for each indentor for varying reference force.

        Parameters:
            FWHM (arr)              - 2D array of y coordinates over grid positions
            indentorRadius (arr)    - 2D array of z coordinates of force contour over grid positions
            refForces (float)       - Threshold force to evaluate indentation contours at (pN)
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''

    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*indentorRadius**2)

    omega  = 2*np.pi/waveDims[0]
    phi    = -np.pi/2

    hsFWHM = (np.arcsin(0)-phi)/omega

    # ------------------------------------------------Plot ----------------------------------------------------------------------
    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, 1/1.61*linewidth/3))
    for n in range(len(indentorRadius)):
        # Plot fwhm for each indentor as they vary over reference force
        ax.plot(np.insert((refForces),0,0)/F_dim[n], FWHM[:,n]/hsFWHM, lw = 1, label = r'$\frac{R}{\lambda}$= '+str(indentorRadius[n]/waveDims[0]))

    # Set axis label and legend
    ax.set_xlabel(r'$\frac{F}{E^*R^2}$', labelpad=5)
    ax.set_ylabel(r'$FWHM^*$')
    ax.set_xscale('log')

    # plt.legend(frameon=False, loc = [0,0], labelspacing=0.2)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-FWHM.png', bbox_inches = 'tight')
```

```
    plt.show()

    fig2, ax2 = plt.subplots(1, 1, figsize = (linewidth, 1/20*linewidth))

    # create a legend for the legend plot
    ax2.legend(*ax.get_legend_handles_labels(), frameon=False, loc='center', ncol=5)
    # remove the axis from the legend plot
    ax2.axis('off')

    fig2.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Legend.png', bbox_inches = 'tight')
    plt.show()
```

Fourier

In [ ]:

```
def FourierPlot(X, Z, TotalRF, NrackPos, forceGrid,  forceContour,  refForce, m, indentorRadius, clearance, A, Nmax, N, waveDims, elasticProperties, contrast):
    '''
    Function to plot Full Width Half Maxima of force contour for each indentor for varying reference force.

        Parameters:
            X (arr)                 - 1D array of x coordinates over scan positions
            Z (arr)                 - 1D array of z coordinates over scan positions
            TotalRF(arr)            - Array of reaction force on indentor reference point
            NrackPos (arr)          - Array of initial scan positions for indenter [Nb, [x, z] ]
            forceGrid (arr)         - 2D Array of force grid of xz positions
            forceContour( arr)      - 2D Array of coordinates for contours of constant force given by reference force
            refForce (float)        - Threshold force to evaluate indentation contours at
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            clearance(float)        - Clearance above molecules surface indentor is set to during scan
            A (arr)                 - Array of Fourier components for force contour for corresponding indentor and reference force [Nf,Ni,Nb]
            N (int)                 - Number of fourier series terms included in fit
            Nmax (int)              - Maximum number of terms in fourier series of force contour
            waveDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            contrast (float)        - Contrast between high and low values in AFM heat map (0-1)
            m (int)                 -
    '''

    # ----------------------------------------------Setup Variables----------------------------------------------

    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    Ni = len(indentorRadius)

    # Surface variables
    waveLength, waveAmplitude, waveWidth, groupNum = waveDims

    # Set normalisation for colour map
    normalizer = mpl.colors.Normalize(0, contrast*(TotalRF/(E_eff*indentorRadius[:,None,None]**2)).max())

    # Fit surface to fourier series
    x = np.linspace(-waveLength, 0, 100)
    popt, pcov = curve_fit(lambda x, *a: Fourier(x, waveDims, *a), x, waveSin(x, waveDims), p0 =tuple(np.zeros(N)))

    # Bar chart variables
    width = 0.04
    f = np.array([(2*np.pi*k)/waveLength for k in range(N)])
    x_labels = [str(k) for k in range(N)]


    # ----------------------------------------------Plot 1----------------------------------------------
    fig, ax = plt.subplots(1, 1, figsize = (linewidth, 1/1.61*linewidth/2))
    ax.bar(f[:N] - width*(Ni)/2, abs(popt[:N])/(abs(popt[1])), color='k', width=width, label = 'Surface')
    for n in range(Ni):
        ax.bar(f[:N]+(n+1)*width - width*(Ni)/2, abs(A[m,n,:N])/(abs(popt[1])), width=width, label = r'$\frac{R}{\lambda}$ = '+str(indentorRadius[n]/waveLength))
    ax.set_xlabel(r'$k(\frac{2\pi}{\lambda})$', fontsize = 13)
    ax.set_ylabel(r'$A_k/A_{Surface}$', fontsize = 13)

    plt.xticks(f, x_labels, fontsize = 13)
    plt.yticks(fontsize = 13)
    plt.legend(frameon=False, ncol=2, fontsize = 13)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Fourier1.png', bbox_inches = 'tight')
    plt.show()


    # ----------------------------------------------Plot 1 b----------------------------------------------
    fig, ax = plt.subplots(1, 1, figsize = (linewidth, 1/1.61*linewidth/2))
    ax.bar(f[:N] - width*(Ni)/2, abs(popt[:N])/(abs(popt[1])), color='k', width=width, label = 'Surface')
    for n in range(Ni):
        ax.bar(f[:N]+(n+1)*width - width*(Ni)/2, abs(A[m,n,:N])/sum(abs(A[m,n,1:N])), width=width, label = r'$\frac{R}{\lambda}$ = '+str(indentorRadius[n]/waveLength))
    ax.set_xlabel(r'$k(\frac{2\pi}{\lambda})$', fontsize = 13)
    ax.set_ylabel(r'$A^*_k$', fontsize = 13)

    # ax.set_title('Fourier Series Coeffients')
    plt.xticks(f, x_labels, fontsize = 13)
    plt.yticks(fontsize = 13)
    plt.legend(frameon=False, ncol=2, fontsize = 13)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Fourier1b.png', bbox_inches = 'tight')
    plt.show()


    # ----------------------------------------------Plot 2----------------------------------------------
    fig, ax = plt.subplots(1, Ni, figsize = (linewidth, 1/1.61*linewidth))
    for n, rIndentor in enumerate(indentorRadius):

        # Force to set dimensionless plot
        F_dim = (E_eff*rIndentor**2)


        # Extract xz compontents of force contour - compressed removes masked values
        Fx, Fz = np.array(forceContour[n,:,0].compressed()), np.array(forceContour[n,:,1].compressed())

        # Plot of force heatmaps for data using imshow to directly visualise 2D array
        im = ax[n].imshow(forceGrid[n].T/F_dim, origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm=normalizer,
                          extent=(X[0]/waveLength, X[-1]/waveLength, Z[0]/waveLength, Z[-1]/waveLength), interpolation_stage = 'rgba', aspect='auto')

        # Plot spline force for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
        ax[n].plot(X/waveLength, Fourier(X, waveDims,*A[m,n][:N])/waveLength, color = 'r', lw = 3, label = 'Fitted Contour')
        ax[n].plot(X/waveLength, waveSin(X, waveDims)/waveLength, ':',        color = 'w', lw = 3, label = 'Surface boundary')


        # Plot Fourier Components
        for k in range(1,N+1):
            ax[n].plot(x/waveLength,  (Fourier(x, waveDims,*A[m,n,:k+1]) - (Z[-1]-Z[0]) - N + k)/waveLength, '--', lw=2, label = r'$A_{'+str(k)+'}$ Cosine Component')

        for k in range(1,N+1):
            ax[n].plot(x/waveLength, (A[m,n,k]*np.cos((2*np.pi*k*x)/waveLength)- (Z[-1]-Z[0]) - 1.5*N -4*k)/waveLength, ':', lw=2)

        # Set legend and axis labels, limits and title
        ax[n].set_title(r'$R/\lambda$ = {0}'.format(rIndentor/waveLength))
        ax[n].set_xlim(-waveLength/(2*waveLength), 0)
        ax[n].set_facecolor("grey")

    # plt.legend(fontsize = 20, loc = (1.1,0.8), facecolor='grey')
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Fourier2.png', bbox_inches = 'tight')
    plt.show()
```

```
    # ----------------------------------------------Plot 3----------------------------------------------------------
    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, 1/1.61*linewidth/3))
    for n, rIndentor in enumerate(indentorRadius):
        # Force to set dimensionless plot, add zero value to force array with insert
        F_dim = (E_eff*rIndentor**2)
        ax.plot(np.insert((refForces)/F_dim,0,0), abs(A[:,n,1])/np.sum(abs(A[:,n,1:N]), axis = 1), lw=1, label = r'$R/\lambda$ = '+str(rIndentor/waveLength))

    # Set axis labels
    ax.set_xlabel(r'$\frac{F}{E*R^2}$', labelpad=5)
    ax.set_ylabel(r'$A^*_1$')
    ax.set_xscale('log')

    # plt.legend(frameon=False, loc =[0,0.3], labelspacing=0.2)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Fourier3.png', bbox_inches = 'tight')
    plt.show()

    # ----------------------------------------------Plot 4----------------------------------------------------------
    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, 1/1.61*linewidth/3))
    for n, rIndentor in enumerate(indentorRadius):
        # Force to set dimensionless plot, add zero value to force array with insert
        F_dim = (E_eff*rIndentor**2)
        ax.plot(np.insert((refForces)/F_dim,0,0), abs(A[:,n,1])/(abs(popt[1])), lw=1, label = r'$\frac{R}{\lambda}$ = '+str(rIndentor/waveLength))

    # Set axis labels
    ax.set_xlabel(r'$\frac{F}{E*R^2}$', labelpad=5)
    ax.set_ylabel(r'$A_1/A_{Surface}$')
    ax.set_xscale('log')

    # plt.legend(frameon=False, labelspacing=0.2)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Fourier4.png', bbox_inches = 'tight')
    plt.show()
```

Volume

In [ ]:
```
def VolumePlot(Volume, indentorRadius, refForces, waveDims, elasticProperties):
    '''
    Function to plot volume under force contour for each indentor for varying reference force.

        Parameters:
            Volume (arr)            - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            refForces (float)       - Threshold force to evaluate indentation contours at, mimics feedback force in AFM (pN)
            waveDims (list)         - Geometric parameters for defining wave base/ substrate structure [wavelength, amplitude, width, Group number]
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''
    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)

    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*indentorRadius**2)

    # Calculate volume of surface portion
    x = np.linspace(-waveDims[0]/2,0,200)
    surfaceVolume = UnivariateSpline(x, waveSin(x,waveDims)).integral(-waveDims[0]/2,0)

    # Plot Volume variation over indentation force
    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, 1/1.61*linewidth/3))
    for n, rIndentor in enumerate(indentorRadius):
        # Plot for each indentor variation of volume over reference force, add zero value to force array with insert (for hard sphere, F=0)
        ax.plot(np.insert((refForces),0,0)/F_dim[n], Volume[:,n]/surfaceVolume,  linewidth = 1, label = r'$\frac{R}{\lambda}$= '+ str(rIndentor/waveDims[0]))

    # Set axis label and legend
    ax.set_xlabel(r'$\frac{F}{E*R^2}$', labelpad=5)
    ax.set_ylabel(r'$\frac{V_{contour}}{V_{surface}}$')
    ax.set_xscale('log')
    # plt.legend(frameon=False,  ncol=2, loc = [0,0], labelspacing=0.2)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Volume.png', bbox_inches = 'tight')
    plt.show()
```

Youngs Modulus

In [ ]:
```
def YoungPlot(E_hertz, indentorRadius, NrackPos, waveDims, elasticProperties):
    '''
    Function to plot elastic modulus over scan position for each indentor.

        Parameters:
            E_hertz (arr)           - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            NrackPos (arr)          - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
            waveDims (list)         - Geometric parameters for defining wave base/ substrate structure [wavelength, amplitude, width, Group number]
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''
    # Set material Propeties
    E_true, v = elasticProperties

    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, 1/1.61*linewidth/3) )
    for n, rIndentor in enumerate(indentorRadius):
        # Plot for each indentor variation of elastic modulus over scan positions
        ax.plot(-NrackPos[n,:,0]/waveDims[0], np.ma.masked_less(E_hertz[n], 0)[::-1]/E_true,  lw = 1, label = r'$\frac{R}{\lambda}$='+ str(rIndentor/waveDims[0]))

    # Expected elastic modulus value
    ax.plot(-NrackPos[0,:,0]/waveDims[0], NrackPos[1,:,0]**0, ':', color = 'k', lw = 1)

    # Set axis label and legend
    ax.set_xlabel(r'$\frac{x}{\lambda}$', labelpad=15)
    ax.set_ylabel(r'$\frac{E_{Fitted}}{E_{True}}$')
    # ax.Legend(frameon=False,  ncol=2, labelspacing=0.2)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Wave-Youngs.png', bbox_inches = 'tight')
    plt.show()
```

Simulation Interface

In [ ]:
```
#  ----------------------------------------------Remote Variables-----------------------------------------------------
# host = "kathleen.rc.ucl.ac.uk"
host = "myriad.rc.ucl.ac.uk"
port = 22
username = "zcapjgi"
password = "ucl.Giblj003.315/Burnj003.315"
home     = '/home/zcapjgi'
scratch  = '/scratch/scratch/zcapjgi'

bashCommand = 'qsub'
abqCommand  = 'module load abaqus/2017 \n abaqus cae -noGUI'

# host = "128.40.163.27"
# port = 22
# username = "giblnbrnhm_j"
# password = "axenub13"
```

```
# home = '/home/giblnbrnhm_j@MECHENG2012'

# abqCommand  = '/opt/abaqus2018/abq2018 cae -noGUI'
# abqCommand  = '/opt/abaqus614/Commands/abq6141 cae -noGUI'
```

In [ ]:
```
#  ----------------------------------------Submission Variables----------------------------------------------------
fileName   = 'AFMtestRasterScan-Pos'
subData    = ['72:0:0', '100G', '1']
wrkDir     = '/ABAQUS/WaveSurfaceTest/WaveSimulation1'
localPath  = os.getcwd()


#  ----------------------------------------Setup Variables---------------------------------------------------------
# Surface variables
waveLength    = 20
waveAmplitude = 10
waveWidth     = 4*10
groupNum      = 3
waveDims      = [waveLength, waveAmplitude, waveWidth, groupNum]

E_true, v = 1000, 0.3
E_eff = E_true/(1-v**2)

# Indentor variables
indentorType  =  'Capped' # 'Spherical'
indentorRadius = np.array([i for i in range(1, int(waveLength/2), 2)])    # angstrom - - 1 nm radius tip
theta_degrees = 20                                                        # degrees
tip_length    = 2*waveAmplitude                                           # angstrom

# Scan variables
clearance = 0.2                                          # angstrom
indentionDepths = [4,4,4,4,4] #(indentorRadius*3/2).clip(max = 2)            # angstrom
binSize = 0.5                                            # angstrom
contrast = 1
courseGrain = 0.5
refForces = np.arange(250,4500,500)
Nmax = 250

# ABAQUS variable
timePeriod = 1.5     # s
timeInterval = 0.1  # s
meshSurface = 1.2    # angstrom
meshIndentor = 0.8  # angstrom
elasticProperties = np.array([E_true, v])


Nf, Ni, Nb, Nt = len(refForces), len(indentorRadius), int((waveLength/2)/(binSize) + 1), int(timePeriod/ timeInterval)+1
print('Indentation Depths - ', indentionDepths, 'Indenter Radii - ', indentorRadius, '\n')

#  ----------------------------------------Simulation Script-------------------------------------------------------
X, Z, TotalU2, TotalRF, NrackPos, forceContour, forceGrid, Volume, FWHM, A, E_hertz, F = AFMSimulation(
    host, port, username, password, scratch, wrkDir, localPath, abqCommand, fileName, subData,
    indentorType, indentorRadius, theta_degrees, tip_length, indentionDepths, waveDims,
    refForces, courseGrain, Nmax, binSize, clearance, meshSurface, meshIndentor, timePeriod, timeInterval,
    elasticProperties,
    Submission = 'serial',
    Main = False,
    # SurfacePlot = 1,
    Queue = False,
    Analysis = False,
    Retrieval = False,
    # Compile  = 2,
    Postprocess = True)
    # DataPlot  = 0,)
```

Data Plots

Illustration

In [ ]:
```
rIndentor = 3

# Set tip dimensions
tipDims = TipStructure(rIndentor, theta_degrees, tip_length)

# Set surface dimensions
waveLength, waveAmplitude, waveWidth, groupNum = waveDims
Nw = 70 # int(15*groupNum+1)
wavePos      = np.zeros([Nw, 2])
wavePos[:,0] = np.linspace(-waveLength*groupNum/2, waveLength*groupNum/2, Nw )
wavePos[:,1] = waveSin(wavePos[:,0], waveDims)

Nb, Nt = int((waveLength/2)/(binSize) + 1), int(timePeriod/ timeInterval)+1

# Calculate scan positions
rackPos = ScanGeometry(indentorType, tipDims, waveDims, Nb, 0)

SurfacePlot(rackPos, Nb, waveDims, wavePos, tipDims, binSize, 0)
```

2D Force Heat Maps and Fitted Force

In [ ]:
```
contrast = 1
maxRF    = (TotalRF/(E_eff*indentorRadius[:,None,None]**2)).max()
normalizer = mpl.colors.PowerNorm(0.45, 0, contrast*maxRF)
```

Interpolate

In [ ]:
```
N = 15
n = 4
m = 8

tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

ContourPlot(X, Z, NrackPos[n], forceGrid[m,n], forceContour[m,n], refForces[m], clearance, A[m+1,n], N, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast)
```

No Interplolate

In [ ]:
```
N = 6
n = 0
m = 5
tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

ContourPlotNI(X, Z, NrackPos[n], forceGrid[m,n], forceContour[m,n], refForces[m], clearance, A[m+1,n], N, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast)
```

Contour Line Plot

In [ ]:
```
N = 6
n = 0
```

```
rIndentor =  indentorRadius[n]
tipDims = TipStructure(rIndentor, theta_degrees, tip_length)

LineContourPlot(X, Z, NrackPos[n], forceContour[:,n], refForces, clearance, A[:,n], N, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast)
```

Force Interpolation

In [ ]:
```
n = 4
tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

FInterpolatePlot(X, Z, NrackPos[n], F[n], clearance, waveDims, tipDims, elasticProperties, normalizer, maxRF, contrast)
```

In [ ]:
```
m = 1
contrast = 1
FInterpolatePlot(TotalRF, F, indentorRadius, waveDims, clearance, elasticProperties, contrast )
```

FWHM

In [ ]:
```
FWHMPlot(FWHM, indentorRadius, refForces, waveDims, elasticProperties)
```

Fourier Plot

In [ ]:
```
N = 6
m = 1
FourierPlot(X, Z, TotalRF[m], NrackPos, forceGrid[m],  forceContour[m],  refForces, m, indentorRadius, clearance, A, Nmax, N, waveDims, elasticProperties, contrast)
```

Volume Plot

In [ ]:
```
VolumePlot(Volume, indentorRadius, refForces, waveDims, elasticProperties )
```

Youngs Modulus

In [ ]:
```
YoungPlot(E_hertz, indentorRadius, NrackPos, waveDims, elasticProperties)
```

Bash Commands

In [ ]:
```
# RemoteCommand(host, port, username, password, '', home, 'qstat')
```

In [ ]:
```
# RemoteCommand(host, port, username, password, '', home, "qdel '*' ")
```

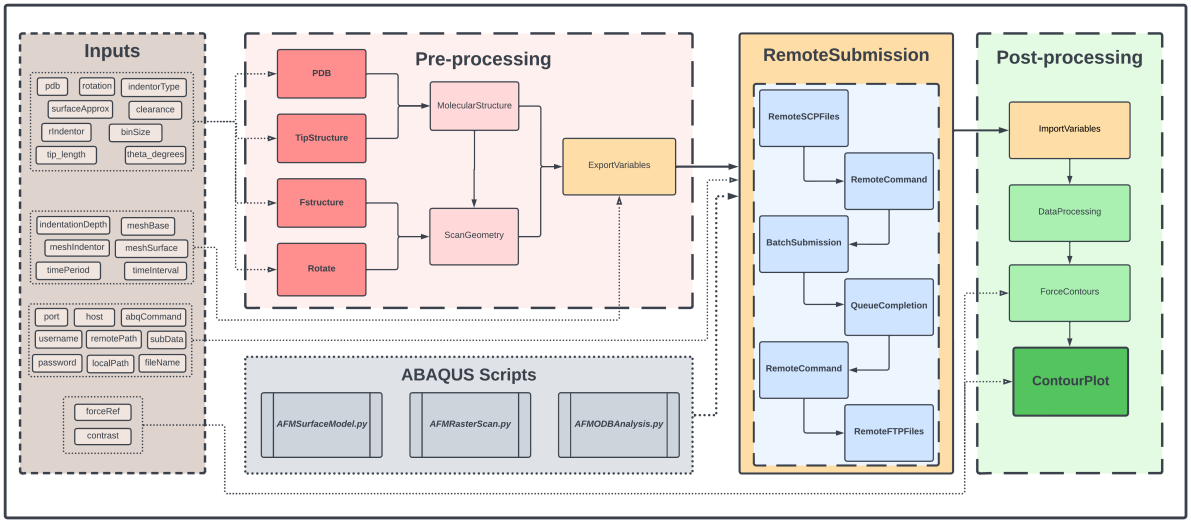
In [ ]:
```
# t0 = time.time()
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.023' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.cid' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.dat' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.lck' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.mdl' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.com' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.msg' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.prt' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.sim' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.sta' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.stt' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.tmp' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.SMABulk' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.SMAFocus' -delete")

# t1 = time.time()
# print(t1-t0)
```

C.3 Hemisphere ABAQUS Simulation Code

Introduction

Authors: J. Giblin-Burnham

Flow Chart



Code Framework

Imports

In [ ]:
```
from platform import python_version
print(python_version())

# ----------------------------------------------------System Imports-------------------------------------------------
import os
import sys
import time
import subprocess
from datetime import timedelta

import paramiko
from scp import SCPClient


# ----------------------------------------------------Mathematical Imports-------------------------------------------
# Importing relevant maths and graphing modules
import numpy as np
import pandas as pd
import math
from numpy import random
from random import randrange
from scipy.interpolate import UnivariateSpline
from scipy.optimize import curve_fit

import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from matplotlib.ticker import MaxNLocator

linewidth = 11.69 # inch

plt.rcParams["figure.figsize"] = (linewidth/3, (1/1.61)*linewidth/3)
plt.rcParams['figure.dpi'] = 256
plt.rcParams['font.size'] = 20
plt.rcParams["font.family"] = "Times New Roman"

plt.rcParams['mathtext.fontset'] = 'custom'
plt.rcParams['mathtext.rm'] = 'Times New Roman'
plt.rcParams['mathtext.it'] = 'Times New Roman:italic'
plt.rcParams['mathtext.bf'] = 'Times New Roman:bold'


# For displaying images in Markdown and  animation import
from IPython.display import Image
from IPython.display import clear_output

# ----------------------------------------------------Optimization modules-------------------------------------------
import numba
from numba import prange
```

ABAQUS Scripts

ABAQUS is run automatically using seperate python scripts defined below. These are transfered to remote server and run using bash commands to produce input files before running simulations.

Raster Scan AFM Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to create ABAQUS input files at each position in the raster scan. These are used to performm the independent analysis/ simulation at each position.

In [ ]:
```
%%writefile AFMtestRasterScan.py

# ----------------------------------------------Load Modules-------------------------------------------------------------
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
#from abaqus import getInput
executeOnCaeStartup()

# ----------------------------------------------Set variables-------------------------------------------------------------
with open('indentorType.txt', 'r') as f:
    indentorType = f.read()

elasticProperties =  np.loadtxt('elasticProperties.csv', delimiter=",")
variables = np.loadtxt('variables.csv', delimiter=",")
baseDims  = np.loadtxt('baseDims.csv', delimiter=",")
tipDims   = np.loadtxt('tipDims.csv', delimiter=",")
rackPos   = np.loadtxt('rackPos.csv', delimiter=",")

timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface, rSurface = variables
rIndentor, theta, tip_length, r_int, z_int, r_top, z_top    = tipDims

#  ----------------------------------------------Model-------------------------------------------------------------
modelName = 'AFMtestRasterScan'
model = mdb.Model(name=modelName)

# ----------------------------------------------Set Parts-------------------------------------------------------------
# Create Surface part
model.ConstrainedSketch(name = 'surface', sheetSize=1.0)
model.sketches['surface'].ConstructionLine( point1=(0,rSurface), point2=(0,-rSurface) )
model.sketches['surface'].Line( point1=(0,0), point2=(0,rSurface) )
model.sketches['surface'].Line( point1=(0,0), point2=(rSurface,0) )
model.sketches['surface'].ArcByCenterEnds(center=(0,0),point1=(0,rSurface),point2=(rSurface,0),direction = CLOCKWISE)
model.Part(name='surface', dimensionality=THREE_D, type= DEFORMABLE_BODY)
model.parts['surface'].BaseSolidRevolve( angle=360.0, flipRevolveDirection=OFF, sketch= model.sketches['surface'] )

if indentorType == 'Capped':
    # Create Capped-Conical Indentor
    sketch = model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,-rIndentor),point2=(0,z_top))
    model.sketches['indentor'].Line(point1=(r_int,z_int), point2=(r_top,z_top))
    model.sketches['indentor'].Line(point1=(0,-rIndentor), point2=(0,z_top))
    model.sketches['indentor'].Line(point1=(0,z_top), point2=(r_top,z_top))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0), point1=(r_int,z_int), point2=(0,-rIndentor),
                                               direction =CLOCKWISE)
    model.Part(name='indentor', dimensionality=THREE_D, type= DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseShellRevolve(angle=360.0, flipRevolveDirection=OFF, sketch=model.sketches['indentor'])

else:
    # # Create Spherical Indentor part
    model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,rIndentor),point2=(0,-rIndentor))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0),point1=(0,rIndentor),point2=(0,-rIndentor),
                                               direction = CLOCKWISE)
    model.sketches['indentor'].Line(point1=(0,rIndentor),point2=(0,-rIndentor))
    model.Part(name='indentor', dimensionality=THREE_D, type=DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseShellRevolve( angle=360.0, flipRevolveDirection=OFF, sketch=model.sketches['indentor'])

# Create Base part
model.ConstrainedSketch(name = 'base', sheetSize=1.0)
model.sketches['base'].rectangle(point1=(-baseDims[0]/2,-baseDims[1]/2), point2=(baseDims[0]/2,baseDims[1]/2))
model.Part(name='base', dimensionality=THREE_D, type= DEFORMABLE_BODY)
model.parts['base'].BaseSolidExtrude(sketch= model.sketches['base'], depth=baseDims[2])


# ----------------------------------------------Set Geometry-------------------------------------------------------------
# Create geometric sets for faces and cells
model.parts['surface'].Set(faces= model.parts['surface'].faces.getSequenceFromMask(mask=('[#1]', ), ),
                           name='surface_faces')
model.parts['surface'].Set(cells= model.parts['surface'].cells.getSequenceFromMask(mask=('[#1]', ), ),
                           name='surface_cells')
model.parts['surface'].Set(edges= model.parts['surface'].edges.getSequenceFromMask(mask=('[#2]', ), ),
                           name='surface_base')
model.parts['base'].Set(faces = model.parts['base'].faces.getSequenceFromMask(mask=('[#20]', ), ), name='base_faces')
model.parts['base'].Set(cells = model.parts['base'].cells.getSequenceFromMask(mask=('[#1]' , ), ), name='base_cells')

if indentorType == 'Capped':
    # Spherically Capped
    model.parts['indentor'].Set(faces= model.parts['indentor'].faces.getSequenceFromMask(mask=('[#7]', ), ),
                                name='indentor_faces')
else:
    # Spherical
    model.parts['indentor'].Set(faces= model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1]', ), ),
                                name='indentor_faces')

# Create gemoetric surface for contact
model.parts['surface'].Surface(name='surface_surface',
                               side1Faces = model.parts['surface'].faces.getSequenceFromMask(mask=('[#1]', ), ) )
model.parts['base'].Surface(name='base_surface',
                            side1Faces = model.parts['base'].faces.getSequenceFromMask(mask=('[#20]', ), ) )

if indentorType == 'Capped':
    # Spherically Capped
    model.parts['indentor'].Surface(name='indentor_surface',
                                    side1Faces = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#7]', ), ))
else:
    # Spherical
    model.parts['indentor'].Surface(name='indentor_surface',
```

```
                                    side1Faces = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1]', ), ))

# Create reference points
point = model.parts['surface'].ReferencePoint((0, 0, 0))
model.parts['surface'].Set(referencePoints=(model.parts['surface'].referencePoints[point.id],),
                           name='surface_centre')
point = model.parts['indentor'].ReferencePoint((0, 0, 0))
model.parts['indentor'].Set(referencePoints = (model.parts['indentor'].referencePoints[point.id],),
                            name = 'indentor_centre')
point = model.parts['base'].ReferencePoint((0, 0, 0))
model.parts['base'].Set(referencePoints = (model.parts['base'].referencePoints[point.id],), name= 'base_centre')


# ----------------------------------------------Set Properties-----------------------------------------------------
# Assign materials
elastic = (tuple(elasticProperties), )
viscoelastic = ((0.0403,0,0.649),(0.0458,0,1.695),)

# Surface material assignment
model.Material(name='surface_material')
model.materials['surface_material'].Elastic(table = elastic)
# model.materials['surface_material'].Viscoelastic(domain = FREQUENCY, frequency = PRONz, table = viscoelastic )
model.HomogeneousSolidSection(name='section', material='surface_material', thickness=None)
model.parts['surface'].SectionAssignment(region=model.parts['surface'].sets['surface_cells'],sectionName='section')

# Base material assignment
model.Material(name='base_material')
model.materials['base_material'].Elastic(table = ((1e15,0.4),))
model.HomogeneousSolidSection(name='base_section', material='base_material', thickness=None)
model.parts['base'].SectionAssignment(region = model.parts['base'].sets['base_cells'], sectionName='base_section')


# ----------------------------------------------Set Assembly-----------------------------------------------------
model.rootAssembly.Instance(name='surface', part = model.parts['surface'],dependent=ON)
model.rootAssembly.Instance(name='indentor', part = model.parts['indentor'], dependent=ON)
model.rootAssembly.Instance(name='base', part = model.parts['base'], dependent=ON)
model.rootAssembly.DatumCsysByDefault(CARTESIAN)
# Position base
model.rootAssembly.rotate(instanceList = ('base',) , axisPoint=(0,0,0), axisDirection= (1,0,0), angle = 90 )


# ----------------------------------------------Set Steps-----------------------------------------------------
model.StaticStep(name='Indentation', previous='Initial', description='', timePeriod=timePeriod,
                 timeIncrementationMethod=AUTOMATIC, maxNumInc=int(1e5), initialInc=0.1, minInc=1e-20, maxInc=1)

model.steps['Indentation'].control.setValues(allowPropagation=OFF, resetDefaultValues=OFF,
                                             timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 25.0, 6.0, 3.0,
                                                                 50.0))
field = model.FieldOutputRequest('F-Output-1', createStepName='Indentation', variables=('RF', 'TF', 'U'),
                                 timeInterval = timeInterval)


# ----------------------------------------------Set Interactions-----------------------------------------------------
model.ContactProperty(name ='Contact Properties')
model.interactionProperties['Contact Properties'].TangentialBehavior(formulation =ROUGH)
model.interactionProperties['Contact Properties'].NormalBehavior(pressureOverclosure=HARD)

model.RigidBody(name = 'indentor_constraint',
                bodyRegion = model.rootAssembly.instances['indentor'].sets['indentor_faces'],
                refPointRegion = model.rootAssembly.instances['indentor'].sets['indentor_centre'])

model.RigidBody(name = 'base_constraint',
                bodyRegion = model.rootAssembly.instances['base'].sets['base_faces'],
                refPointRegion = model.rootAssembly.instances['base'].sets['base_centre'])

model.SurfaceToSurfaceContactStd(name = 'surface-indentor',
                                 createStepName = 'Initial',
                                 master = model.rootAssembly.instances['indentor'].surfaces['indentor_surface'],
                                 slave  = model.rootAssembly.instances['surface'].surfaces['surface_surface'],
                                 interactionProperty = 'Contact Properties',
                                 sliding = FINITE)

model.SurfaceToSurfaceContactStd(name = 'base-sphere',
                                 createStepName = 'Initial',
                                 master = model.rootAssembly.instances['base'].surfaces['base_surface'],
                                 slave  = model.rootAssembly.instances['surface'].surfaces['surface_surface'],
                                 interactionProperty = 'Contact Properties',
                                 sliding = FINITE)


# ----------------------------------------------Set Loads-----------------------------------------------------
# Create base boundary conditions
model.DisplacementBC(name = 'Base-BC', createStepName = 'Initial',
                     region = model.rootAssembly.instances['base'].sets['base_faces'],
                     u1 = SET, u2 = SET, u3 = SET, ur1 = SET, ur2 = SET, ur3 = SET)

# Create surface boundary conditions
model.DisplacementBC(name = 'Surface-BC', createStepName = 'Initial',
                     region = model.rootAssembly.instances['surface'].sets['surface_base'],
                     u1 = SET, u2 = SET, u3 = SET, ur1 = SET, ur2 = SET, ur3 = SET)

# Create indentor boundary conditions
model.DisplacementBC(name = 'Indentor-UC', createStepName = 'Indentation',
                     region = model.rootAssembly.instances['indentor'].sets['indentor_centre'],
                     u1  = SET, u2  = -indentionDepth, u3  = SET,
                     ur1 = SET, ur2 = SET,             ur3 = SET)


# ----------------------------------------------Set Mesh-----------------------------------------------------
#Assign an element type to the part instance- seed and generate
model.rootAssembly.regenerate()

elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D15, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D10, elemLibrary=STANDARD, secondOrderAccuracy=ON, distortionControl=DEFAULT)
cells     = model.parts['surface'].cells.getSequenceFromMask(mask=('[#1 ]', ), )

model.parts['surface'].seedPart(size=0.9, minSizeFactor=meshSurface)
model.parts['surface'].setMeshControls(regions=cells, elemShape=TET, technique=FREE)
model.parts['surface'].setElementType(regions=(cells,), elemTypes=(elemType1, elemType2, elemType3))
model.parts['surface'].generateMesh()


elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=STANDARD)
faces     = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1 ]', ),)
```

```
model.parts['indentor'].seedPart(size=0.5, minSizeFactor= meshIndentor)
model.parts['indentor'].setMeshControls(regions=faces, elemShape=TRI)
model.parts['indentor'].setElementType(regions=(faces,), elemTypes=(elemType1, elemType2))
model.parts['indentor'].generateMesh()


model.parts['base'].seedPart(size = 2 )
model.parts['base'].setElementType(model.rootAssembly.instances['base'].sets['base_faces'],
                                   elemTypes = (mesh.ElemType(elemCode=QUAD, elemLibrary=STANDARD),))
model.parts['base'].setMeshControls(regions=model.rootAssembly.instances['base'].sets['base_faces'].vertices,
                                    elemShape=TET,technique=FREE)
model.parts['base'].generateMesh()

# ----------------------------------------------Set Submission-----------------------------------------------------------
for i in range(len(rackPos)):
    jobName = 'AFMtestRasterScan-Pos'+str(int(i))
    model.rootAssembly.translate(instanceList = ('indentor',), vector=(rackPos[i,0], rackPos[i,1]+rIndentor, 0))
    job = mdb.Job(name=jobName, model=modelName, description='AFM Sphere')
    job.writeInput()
    model.rootAssembly.translate(instanceList = ('indentor',), vector=(-rackPos[i,0], -rackPos[i,1]-rIndentor, 0))

mdb.saveAs('AFMRaster.cae')
```

ODB Analysis Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to analysis odb files for each position in the raster scan. Extracting indentation data.

In [ ]:

```
%%writefile AFMtestODBAnalysis.py

# ----------------------------------------------Load Modules-----------------------------------------------------------
import sys
from odbAccess import *
from types import IntType
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
#from abaqus import getInput
executeOnCaeStartup()

# ----------------------------------------------Set variables-----------------------------------------------------------
variables = np.loadtxt('variables.csv', delimiter=",")
rackPos    = np.loadtxt('rackPos.csv', delimiter=",")

timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface, rSurface = variables

N = int(timePeriod/ timeInterval)+1
RF = np.zeros([len(rackPos),N])
U2 = np.zeros([len(rackPos),N])

# ----------------------------------------------Set Data extraction-----------------------------------------------------
for i in range(len(rackPos)):
    jobName = 'AFMtestRasterScan-Pos'+str(int(i))
    try :
        # Opening the odb
        odb    = openOdb(jobName +'.odb', readOnly=True)
        region = odb.rootAssembly.nodeSets.values()[2]
    except:
        with open('Errors.txt', 'a') as f:
            f.write('ERROR for'+str(i)+'\n')
    else:
        # Extracting Step 1, this analysis only had one step
        step1 = odb.steps.values()[0]

        j,k = 0, 0
        # Creating a for loop to iterate through all frames in the step
        for x in odb.steps[step1.name].frames:
            # Reading stress and strain data from the model
            fieldRF = x.fieldOutputs['RF'].getSubset(region= region)
            fieldU  = x.fieldOutputs['U'].getSubset(region= region)

            # Storing Stress and strain values for the current frame
            for rf in fieldRF.values:
                RF[i,j] = np.sqrt(rf.data[1]**2)
                j+=1

            for u in fieldU.values:
                U2[i,k] = u.data[1]
                k+=1

# Writing to a .csv file
np.savetxt('U2_Results.csv', U2 , delimiter=",")
np.savetxt('RF_Results.csv', RF , delimiter=",")

# Close the odb
odb.close()
```

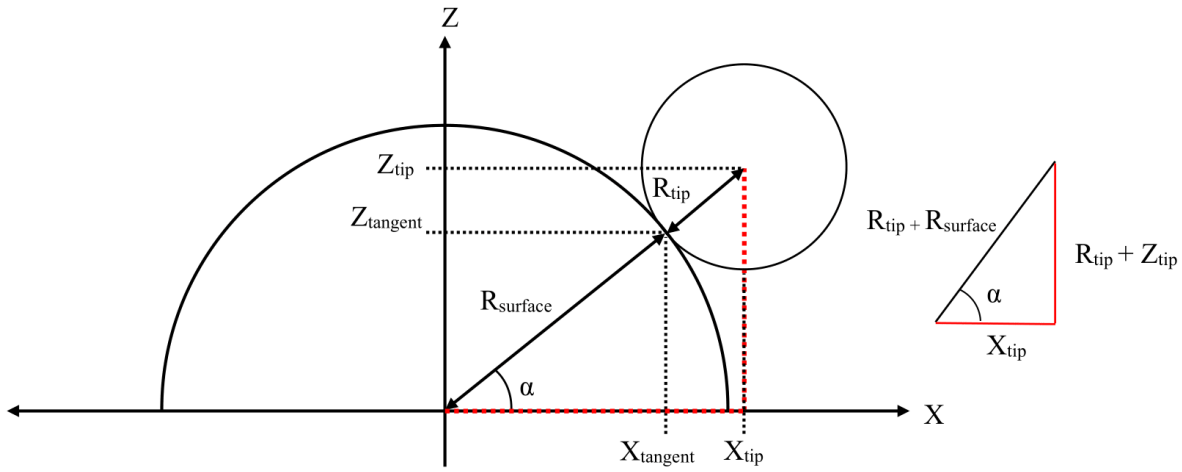
Simulation Code Functions

Functionalised code to automate scan and geometry calculations, remote server access, remote script submission, data anlaysis and postprocessing required to produce AFM image.

Pre-Processing Function

Functions used in preprocessing step of simulation, including calculating scan positions and exporting variables.

Scan positions can be calculated explicitly for a sphere:



From geometry: $$cos(\alpha) = \frac{X_{tip}}{R_{surface}+R_{tip}}$$

$$sin(\alpha) = \frac{Z_{tip}+R_{tip}}{R_{surface}+R_{tip}} $$
/ From trigonomerty: $$sin(\alpha) = \frac{\sqrt{(R_{surface}+R_{tip})^2-X_{tip}^2}}{R_{surface}+R_{tip}}$$

Therefore: $$ \frac{\sqrt{(R_{surface}+R_{tip})^2-X_{tip}^2}}{R_{surface}+R_{tip}} = \frac{Z_{tip}+R_{tip}}{R_{surface}+R_{tip}} $$

$$ Z_{tip} = \sqrt{(R_{surface}+R_{tip})^2-X_{tip}^2} - R_{tip} $$

Tip Functions

Functions to produce list of tip structural parameters, alongside function to calculates and returns tip surface heights from radial position r.

In [ ]:
```
def TipStructure(rIndentor, theta_degrees, tip_length):
    '''
    Produce list of tip structural parameters. Change principle angle to radian. Calculate tangent point where
    sphere smoothly transitions to cone for capped conical indentor.

        Parameters:
            theta_degrees (float) - Principle conical angle from z axis in degrees
            rIndentor (float)     - Radius of spherical tip portion
            tip_length (float)    - Total cone height

        Returns:
            tipDims (list) - Geometric parameters for defining capped tip structure
    '''
    theta = theta_degrees*(np.pi/180)

    # Intercept of spherical and conical section of indentor (Tangent point)
    r_int, z_int = rIndentor*abs(np.cos(theta)), -rIndentor*abs(np.sin(theta))
    # Total radius/ footprint of indentor/ top coordinates
    r_top, z_top = (r_int+(tip_length-r_int)*abs(np.tan(theta))), tip_length-rIndentor

    return [rIndentor, theta, tip_length, r_int, z_int, r_top, z_top]
```

In [ ]:
```
def Fconical(r, r0, r_int, z_int, theta, R, tip_length):
    '''
    Calculates and returns spherically capped conical tip surface heights from radial  position r. Uses radial coordinate along
    xz plane from centre as tip is axisymmetric around z axis (bottom of tip set as zero point such z0 = R).

        Parameters:
            r (float/1D arr)   - xz radial coordinate location for tip height to be found
            r0 (float)         - xz radial coordinate for centre of tip
            r_int (float)      - xz radial coordinate of tangent point (point where sphere smoothly transitions to cone)
            z_int (float)      - Height of tangent point, where sphere smoothly transitions to cone (defined for tip centred at spheres
                                 center, as calculations assume tip centred at indentors bottom the value must be corrected to, R-z_int)
            theta (float)      - Principle conical angle from z axis in radians
            R (float)          - Radius of spherical tip portion
            tip_length (float) - Total cone height

        Returns:
            Z (float/1D arr)- Height of tip at xz radial coordinate
    '''

    ### Constructing conical and spherical parts boundaries of tip using arrays for computation speed

    # ----------------------------------------------------Spherical Boundary-------------------------------------------------
    # For r <= r_int, z <= z_int : (z-z0)^2 +  (r-r0)^2 = R^2 --> z = z0  + ( R^2 - (r-r0)^2 )^1/2

    # Using equation of sphere compute height (points outside sphere radius are complex and return nan,
    # nan_to_num is used to set these points to max value R). The heights are clip to height of tangent point, R-z_int.
    # Producing spherical portion for r below tangent point r_int and constant height R-zint for r values above r_int.

    z1 = np.clip( np.nan_to_num(R - np.sqrt(R**2 - (r-r0)**2), copy=False, nan=R ), a_min = 0 , a_max = R-abs(z_int))
    # z1 = np.clip( np.where( np.isnan( R - np.sqrt(R**2 - (r-r0)**2) ) , R, R - np.sqrt(R**2 - (r-r0)**2) ), a_min = 0 , a_max = R-np.abs(z_int))

    # ----------------------------------------------------Conical Boundary---------------------------------------------------
    # r > r_int, z > z_int : z = m*abs(x-x0);  where x = r, x0 = r0 + r_int,  m = 1/tan(theta)

    # Using equation of cone (line) to compute height for r values larger than tangent point r_int (using where condition)
    # For r values below r_int the height is set to zero

    z2 =np.where(abs(r-r0)>=r_int, (abs(r-r0)-r_int)/abs(np.tan(theta)), 0)

    # ----------------------------------------------------Combing Boundaries-------------------------------------------------
    # For r values less than r_int, combines spherical portion with zero values from conical, producing spherical section
    # For r values more than r_int, combines linear conical portion with R-z_int values from spherical, producing cone section
    Z = z1 + z2
```

```
    # Optional mask values greater than tip length
    Z = np.ma.masked_greater(z1+z2, tip_length )
    return Z
```

Scan Functions

Calculate scan positions of tip over surface and vertical set points above surface for each position. In addition, function to plot and visualise molecules surface and scan position.

In [ ]:
```
def F_TipConvolution(x, rSurface, rIndentor):
    '''Function to calculate hard sphere tip convolution of tip and semi-sphere using mathematics above'''
    return ( np.sqrt( (rSurface + rIndentor)**2 - (abs(x).clip(max=rSurface+rIndentor))**2)- rIndentor).clip(min=0)
```

In [ ]:
```
def ScanGeometry(indentorType, tipDims, baseDims, Nb, clearance):
    '''
    Produces array of scan locations and corresponding heights/ tip positions above surface in Angstroms (x10-10 m).The scan positions are produced creating a
    straight line along the centre of the surface with positions spaced by the bin size. Heights, at each position, are calculated for conical indentor by set
    tip above sample and calculating vertical distance between of tip and molecules surface over the indnenters area. Subsequently, the minimum vertical distance
    corresponds to the position where tip is tangential. Spherical indentors are calculated explicitly.

        Parameters:
            indentorType (str) - String defining indentor type (Spherical or Capped)
            tipDims (list)     - Geometric parameters for defining capped tip structure
            baseDims (list)    - Geometric parameters for defining base/ substrate structure [width, height, depth]
            Nb (int)           - Number of scan positions along x axis of base
            clearance (float)  - Clearance above molecules surface indentor is set to during scan

        Returns:
            rackPos (arr) - Array of coordinates [x,z] of scan positions to image biomolecule
    '''
    #  ----------------------------------Set Rack Positions from Scan Geometry-----------------------------------------
    # Intialise array of raster scan positions
    rackPos = np.zeros([Nb,2])

    # Create linear set of scan positions over base, along x axis
    rackPos[:,0] = np.linspace(-baseDims[0]/2, baseDims[0]/2, Nb)

    [rIndentor, theta, tip_length, r_int, z_int, r_top, z_top] = tipDims

    # ------------------------------------------------For Conically-Capped tip-------------------------------------------------
    if indentorType == 'Capped':

        for i, rPos in enumerate(rackPos[:,0]):
            # Array of radial positions along indentor radial extent. Set indentor position/ coordinate origin at surface height
            # (z' = z + surfaceHeight) and calculate  vertical heights along the radial extent of indentor at position.
            r0 = np.linspace(rPos-r_top, rPos+r_top, 1000)
            z0 = Fconical(r0, rPos, r_int, z_int, theta, rIndentor, tip_length) + rSurface

            # Using equation of sphere compute top heights of atoms surface along indentors radial extent (points outside sphere
            # radius are complex and return nan, nan_to_num is used to set these points to the min value of bases surface z=0).
            z  = np.sqrt(rSurface**2 - (r0**2).clip(max = rSurface**2))

            # The difference in the indentor height and the surface at each point along indenoter extent, produces a dz
            # array of all the height differences between indentor and surface within the indentors boundary around this position.
            # Therefore, z' = -dz  gives an array of indentor positions when each individual part of surface atoms contacts the tip portion above.
            # Translating from z' basis (with origin at z = surfaceHeight) to z basis (with origin at the top of the base) is achieved by
            # perform translation z = z' + surfaceheight. Therefore, these tip position are given by  dz = surfaceheight - dz'. The initial height
            # corresponds to the maximum value of dz/ min value of dz' where the tip is tangential to the surface. I.e. when dz' is minimised
            # all others dz' tip positions will be above/ further from the surface. Therefore, at this position, the rest of the indentor wil
            # not be in contact with the surface and it is tangential.

            rackPos[i,1] = rSurface - abs((z0-z).min()) + clearance

    # ------------------------------------------------------Else Spherical tip-------------------------------------------------------
    else:
        # For spherical indentor use geometry to define scan positions
        rackPos[:,1] = F_TipConvolution(rackPos[:,0], rSurface, rIndentor)
        rackPos[:,1] += clearance

    return rackPos
```

File Import/ Export Function

In [ ]:
```
def ExportVariables(rackPos, variables, baseDims, tipDims, indentorType, elasticProperties ):
    '''
    Export simulation variables as csv and txt files to load in abaqus python scripts.

        Parameters:
            rackPos (arr)           - Array of coordinates [x,z] of scan positions to image biomolecule
            variables (list)        - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor,
                                      indentionDepth, surfaceHeight]
            baseDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)          - Geometric parameters for defining capped tip structure
            indentorType (str)      - String defining indentor type (Spherical or Capped)
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''
    np.savetxt("elasticProperties", elasticProperties, fmt='%s', delimiter=",")
    np.savetxt("rackPos.csv",       rackPos,           fmt='%s', delimiter=",")
    np.savetxt("variables.csv",     variables,         fmt='%s', delimiter=",")
    np.savetxt("baseDims.csv",      baseDims,          fmt='%s', delimiter=",")
    np.savetxt("tipDims.csv",       tipDims,           fmt='%s', delimiter=",")

    with open('indentorType.txt', 'w', newline = '\n') as f:
        f.write(indentorType)
```

In [ ]:
```
def ImportVariables():
    '''
    Import simulation geometry variables from csv files.

        Return:
            variables (list)        - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor,
                                      indentionDepth, surfaceHeight]
            baseDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            rackPos (arr)           - Array of coordinates [x,z] of scan positions to image biomolecule
    '''
    variables          = np.loadtxt('variables.csv', delimiter=",")
    baseDims           = np.loadtxt('baseDims.csv', delimiter=",")
    rackPos            = np.loadtxt('rackPos.csv', delimiter=",")

    return variables, baseDims, rackPos
```

Remote Functions

Functions for working on remote serve, including transfering files, submitting bash commands, submiting bash scripts for batch input files and check queue statis.

File Transfer`

In [ ]:
```
def RemoteSCPFiles(host, port, username, password, files, remotePath):
    '''
    Function to make directory and transfer files to SSH server. A new Channel is opened and the files are transfered.
    The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            files (str/list) - File or list of file to transfer
            remotePath (str) - Path to remote file/directory
    '''
    # SHH to clusters
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    stdin, stdout, stderr = ssh_client.exec_command('mkdir -p ' + remotePath)

    # SCPClient takes a paramiko transport as an argument- Uploading content to remote directory
    scp_client = SCPClient(ssh_client.get_transport())
    scp_client.put(files, recursive=True, remote_path = remotePath)
    scp_client.close()

    ssh_client.close()
```

Bash Command Submission

In [ ]:
```
def RemoteCommand(host, port, username, password, script, remotePath, command):
    '''
    Function to execute a command/ script submission on the SSH server. A new Channel is opened and the requested command is executed.
    The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            script (str)     - Script to run via bash command
            remotePath (str) - Path to remote file/directory
            command (str)    - Abaqus command to execute and run script
    '''
    # SSH to clusters using paramiko module
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # Execute command
    stdin, stdout, stderr = ssh_client.exec_command('cd ' + remotePath + ' \n '+ command +' '+ script +' & \n')
    lines = stdout.readlines()

    ssh_client.close()

    for line in lines:
        print(line)
```

Batch File Submission

In [ ]:
```
def BatchSubmission(host, port, username, password, fileName, subData, rackPos, remotePath, **kwargs):
    '''
    Function to create bash script for batch submission of input file, and run them on remote server.
        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            fileName (str)   - Base File name for abaqus input files
            subData (str)    - Data for submission to serve queue [walltime, memory, cpus]
            rackPos (arr)    - Array of coordinates [x,z] of scan positions to image biomolecule (can be clipped or full)
            remotePath (str) - Path to remote file/directory

            kwargs:
                Submission ('serial'/ 'paralell') - optional define whether single serial script or seperate paralell submission to queue {Default: 'serial'}
    '''
    # For paralell mode create bash script to runs for single scan location, then loop used to submit individual scripts for each location which run in paralell
    if 'Submission' in kwargs and kwargs['Submission'] == 'paralell':
        lines = ['#!/bin/bash -l',
                 '#$ -S /bin/bash',
                 '#$ -l h_rt='+ subData[0],
                 '#$ -l mem=' + subData[1],
                 '#$ -pe mpi ' + subData[2],
                 '#$ -wd /home/zcapjgi/Scratch/ABAQUS',
                 'module load abaqus/2017 ',
                 'ABAQUS_PARALLELSCRATCH = "/home/zcapjgi/Scratch/ABAQUS" ',
                 'cd ' + remotePath,
                 'gerun abaqus interactive cpus=$NSLOTS mp_mode=mpi job=$JOB_NAME input=$JOB_NAME.inp scratch=$ABAQUS_PARALLELSCRATCH resultsformat=odb'
                ]

    # Otherwise, create script to run serial analysis consecutively with single submission
    else:
        # Create set of submission comands for each scan locations
        jobs   = ['gerun abaqus interactive cpus=$NSLOTS mp_mode=mpi job='+fileName+str(int(i))+' input='+fileName+str(int(i))+'.inp scratch=$ABAQUS_PARALLELSCRATCH
                  for i in range(len(rackPos))]

        # Produce preamble to used to set up bash script
        lines = ['#!/bin/bash -l',
                    '#$ -S /bin/bash',
                    '#$ -l h_rt='+ subData[0],
                    '#$ -l mem=' + subData[1],
                    '#$ -pe mpi ' + subData[2],
                    '#$ -wd /home/zcapjgi/Scratch/ABAQUS',
                    'module load abaqus/2017 ',
                    'ABAQUS_PARALLELSCRATCH = "/home/zcapjgi/Scratch/ABAQUS" ',
                    'cd ' + remotePath ]
        # Combine to produce total  script
        lines+=jobs

    # Create script file in current directory by writing each line to file
```

```
    with open('batchScript.sh', 'w', newline = '\n') as f:
        for line in lines:
            f.write(line)
            f.write('\n')

    # SSH to clusters
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    stdin, stdout, stderr = ssh_client.exec_command('mkdir -p ' + remotePath)

    # SCPClient takes a paramiko transport as an argument- Uploading content to remote directory
    scp_client = SCPClient(ssh_client.get_transport())
    scp_client.put('batchScript.sh', recursive=True, remote_path = remotePath)
    scp_client.close()

    # If paralell mode, submit  individual scripts for individual scan locations
    if 'Submission' in kwargs and kwargs['Submission'] == 'paralell':
        for i in range(len(rackPos)):
            # Job name set as each input file name as -N jobname is used as input variable in script
            jobName = fileName+str(int(i))

            # Command to run individual jobs
            batchCommand = 'cd ' + remotePath + ' \n qsub -N '+ jobName +' batchScript.sh \n'

            # Execute command
            stdin, stdout, stderr = ssh_client.exec_command(batchCommand)
            output_line = stdout.readlines()
            print(output_line)

    # Otherwise submit single serial scripts
    else:
        # Job name set as current directory name (change / to \\ for windows)
        jobName = remotePath.split('/')[-1]
        batchCommand = 'cd ' + remotePath + ' \n qsub -N '+ jobName +' batchScript.sh \n'

        # Execute command
        stdin, stdout, stderr = ssh_client.exec_command(batchCommand)
        output_line = stdout.readlines()
        print(output_line)

    ssh_client.close()
```

Queue Status

In [ ]:

```
def QueueCompletion(host, port, username, password):
    '''
    Function to check queue statis and complete when queue is empty.
        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
    '''
    # Log time
    t0 = time.time()
    complete= False

    while complete == False:
        # SSH to clusters
        ssh_client = paramiko.SSHClient()
        ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh_client.connect(host, port, username, password)

        # Execute command to view the queue
        stdin, stdout, stderr = ssh_client.exec_command('qstat')
        lines = stdout.readlines()

        # Check if queue is empty
        if len(lines)==0:
            print('Complete')
            complete = True
            ssh_client.close()

        # Otherwis close and wait 2 mins before checking again
        else:
            ssh_client.close()
            time.sleep(120)

    # Return total time
    t1 = time.time()
    print(t1-t0)
```

File Retrieval

In [ ]:

```
def RemoteFTPFiles(host, port, username, password, files, remotePath, localPath):
    '''
    Function to transfer files from directory on SSH server to local machine. A new Channel is opened and the files are transfered.
    The function uses FTP file transfer.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            files (str )     - File to transfer
            remotePath (str) - Path to remote file/directory
            localPath (str)  - Path to local file/directory
    '''
    # SSH to cluster
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # FTPClient takes a paramiko transport as an argument- copy content from remote directory
    ftp_client=ssh_client.open_sftp()
    ftp_client.get(remotePath+'/'+files, localPath +'\\'+ files)
    ftp_client.close()
```

Remote Terminal

In [ ]:
```
def Remote_Terminal(host, port, username, password):
    '''
    Function to emulate cluster terminal. Channel is opened and commands given are executed. The command's input
    and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption
                               if passphrase is not given.
    '''
    # SHH to cluster
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # Create channel to keep connection open
    ssh_channel = ssh_client.get_transport().open_session()
    ssh_channel.get_pty()
    ssh_channel.invoke_shell()

    # While open accept user input commands
    while True:
        command = input('$ ')
        if command == 'exit':
            break

        ssh_channel.send(command + "\n")

        # Return bash output from command
        while True:
            if channel.recv_ready():
                output = ssh_channel.recv(1024)
                print(output)
            else:
                time.sleep(0.5)
                if not(ssh_channel.recv_ready()):
                    break
    # Close cluster connection
    ssh_client.close()
```

Submission Functions

Function to run simulation and scripts on the remote servers. Files for variables are transfered, ABAQUS scripts are run to create parts and input files. A bash file is created and submitted to run simulation for batch of inputs. Analysis of odb files is performed and data transfered back to local machine. Using keyword arguments invidual parts of simulation previously completed can be skipped.

In [ ]:
```
def RemoteSubmission(host, port, username, password, remotePath, localPath,  csvfiles, abqfiles, abqCommand, fileName, subData, rackPos, **kwargs):
    '''
    Function to run simulation and scripts on the remote servers. Files for variables are transfered, ABAQUS scripts are run to create parts and input files.
    A bash file is created and submitted to run simulation for batch of inputs. Analysis of odb files is performed and data transfered back to local machine.
    Using keyword arguments can submitt the submission files in parrallel.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - Username to authenticate as (defaults to the current local username)
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            remotePath (str) - Path to remote file/directory
            localPath (str)  - Path to local file/directory
            csvfiles (list)  - List of csv and txt files to transfer to remote server
            abqfiles (list)  - List of abaqus script files to transfer to remote server
            abqCommand (str) - Abaqus command to execute and run script
            fileName (str)   - Base File name for abaqus input files
            subData (str)    - Data for submission to serve queue [walltime, memory, cpus]
            rackPos (arr)    - Array of scan positions and initial heights [x,z] to image
            kwargs           - Passes "Submmission" if present to batchSubmission function

    '''
    #  ----------------------------------------------File Transfer----------------------------------------------------------
    # Transfer scripts and variable files to remote server
    RemoteSCPFiles(host, port, username, password, csvfiles, remotePath)
    RemoteSCPFiles(host, port, username, password, abqfiles, remotePath)

    print('File Transfer Complete')

    #  ----------------------------------------------Input File Creation-----------------------------------------------------
    t0 = time.time()
    print('Producing Input Files ...')

    # Produce simulation and input files
    script = 'AFMtestRasterScan.py'
    RemoteCommand(host, port, username, password, script, remotePath, abqCommand)

    t1 = time.time()
    print('Input File Complete - ' + str(timedelta(seconds=t1-t0)) )

    #  ----------------------------------------------Batch File Submission---------------------------------------------------
    t0 = time.time()
    print('Submitting Batch Scripts ...')

    # Submit bash scripts to remote queue to carry out batch abaqus analysis
    BatchSubmission(host, port, username, password, fileName, subData, rackPos, remotePath, **kwargs)

    t1 = time.time()
    print('Batch Submission Complete - '+ str(timedelta(seconds=t1-t0)) + '\n')
```

In [ ]:
```
def DataRetrieval(host, port, username, password, scratch, wrkDir, localPath, csvfiles, dataFiles, indentorRadius, **kwargs):
    '''
    Function to retrieve simulation data transfered back to local machine. Using keyword arguments to change to compilation of simulations data.

        Parameters:
            host (str)           - Hostname of the server to connect to
            port (int)           - Server port to connect to
            username (str)       - Username to authenticate as (defaults to the current local username)        -
            password (str)       - Used for password authentication; is also used for private key decryption if passphrase is not given.
            remotePath (str)     - Path to remote file/directory
            localPath (str)      - Path to local file/directory
            csvfiles (list)      - List of csv and txt files to transfer to remote server
            datafiles (list)     - List of abaqus script files to transfer to remote server
            indentorRadius (arr) - Array of indentor radii of spherical tip portion varied for seperate  simulations
```

```
            kwargs:
                Compile(int)     - If passed, simulation data is compiled from seperate sets of simulations in directory in remote server to combine
                                   complete indentations. Value is set as int representing the range of directories to compile from (directories must have
                                   same root naming convention with int denoting individual directories)

        Return:
            variables (list) - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshIndentor,
                               indentionDepth, surfaceHeight]
            TotalU2 (arr)    - Array of indentors z displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
            TotalRF (arr)    - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
            NrackPos (arr)   - Array of initial scan positions for each indenter [Ni, Nb, [x, z] ]
    '''
    #  ------------------------------------------------Remote Variable-----------------------------------------------------------
    # Import variables from remote server used for the simulations
    for file in csvfiles:
        RemoteFTPFiles(host, port, username, password, file, scratch+wrkDir+'/IndenterRadius1', localPath)

    # Set simulation variables used to process data
    variables, baseDims, rackPos  = ImportVariables()
    timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface, rSurface = variables

    # Set array size variables
    Nb, Nt= int(baseDims[0]/binSize)+1, int(timePeriod/ timeInterval)+1

    #  ------------------------------------------Initialise data arrays----------------------------------------------------------
    NrackPos = np.zeros([len(indentorRadius), Nb, 2])
    TotalRF  = np.zeros([len(indentorRadius), Nb, Nt])
    TotalU2  = np.zeros([len(indentorRadius), Nb, Nt])


    #  ------------------------------------------Compiled data retrieval---------------------------------------------------------
    if 'Compile' in kwargs.keys():
        # Set base directory name to loop over
        wrkDir = '/ABAQUS/HemisphereTests/Test_Hemisphere'

        # For number of directories set to compile
        for n in range(kwargs['Compile']):


            for index, rIndentor in enumerate(indentorRadius):
                # Set path to file
                remotePath = scratch + wrkDir + str(n) + '/IndenterRadius'+str(int(rIndentor))

                # Check file is available
                try :
                    RemoteFTPFiles(host, port, username, password, dataFiles[0], remotePath, localPath)
                    RemoteFTPFiles(host, port, username, password, dataFiles[1], remotePath, localPath)
                    RemoteFTPFiles(host, port, username, password, dataFiles[2], remotePath, localPath)

                except:
                    None

                else:
                    # If files are available load data in to temperary variable
                    U2 = np.array(np.loadtxt(dataFiles[0], delimiter=","))
                    RF = np.array(np.loadtxt(dataFiles[1], delimiter=","))
                    NrackPos[index] = np.array(np.loadtxt(dataFiles[2], delimiter=","))

                    # Loop through data and store indentations with less zeros/ higher sums of forces
                    for i in range(len(RF)):
                        if np.all(TotalRF[index,i]==0)  == True or np.count_nonzero(TotalRF[index,i]==0)>np.count_nonzero(RF[i]==0) or sum(RF[i]) > sum(TotalRF[index
                            TotalU2[index,i]  = U2[i]
                            TotalRF[index,i]  = RF[i]

    #  ----------------------------------------Single Directory retrieval--------------------------------------------------------
    else:
        # For each indentor
        for index, rIndentor in enumerate(indentorRadius):
            # Define path to file
            remotePath = scratch + wrkDir + '/IndenterRadius'+str(int(rIndentor))

            # Retrive data files and store in curent directory
            RemoteFTPFiles(host, port, username, password, dataFiles[0], remotePath, localPath)
            RemoteFTPFiles(host, port, username, password, dataFiles[1], remotePath, localPath)
            RemoteFTPFiles(host, port, username, password, dataFiles[2], remotePath, localPath)

            # Load and set data in array for all indentors
            TotalU2[index]  = np.array(np.loadtxt(dataFiles[0], delimiter=","))
            TotalRF[index]  = np.array(np.loadtxt(dataFiles[1], delimiter=","))
            NrackPos[index] =  np.array(np.loadtxt(dataFiles[2], delimiter=","))

    return variables, TotalU2, TotalRF, NrackPos
```

Post-Processing Functions

Function for postprocessing ABAQUS simulation data, loading variables from files in current directory and process data from simulation in U2/RF files. Process data from scan position to include full data range over all scan positions. Alongside, function to plot and visualise data. Then, calculates contours/z heights of constant force in simulation data for given threshold force and visualise. Produce data analysis for simulation data.

AFM Image Function

Function to produce force heat map over scan domain and calculate contours/z heights of constant force in simulation data for given threshold force.

In [ ]:

```
def ForceGrid2D(X, Z, U2, RF, rackPos, courseGrain, **kwargs):
    '''
    Function to produce force heat map over scan domain.

        Parameters:
            X (arr)             - 1D array of postions over x domain of scan positions
            Z (arr)             - 1D array of postions over z domain of scan positions, discretised into bins of courseGrain value
            U2 (arr)            - Array of indentors y indentor position over scan, discretised into bins of courseGrain value
                                  ( As opposed to displacement into surface given from simulation and used elsewhere)
            RF (arr)            - Array of reaction force on indentor reference point
            rackPos (arr)       - Array of coordinates (x,z) of scan positions to image biomolecule [Nb,[x,z]]
            courseGrain (float) - Width of bins that subdivid xz domain of raster scanning/ spacing of the positions sampled over

            kwargs:
                Symmetric - If false skip postprocessing step to produce AFM image from data {Default: True}

        Return:
            forceGrid (arr)        - 2D Array of force heatmap over xz domain of scan i.e. grid of xz positions with associated force [Nx,Nz]
            forceGridmask (arr)    - 2D boolean array giving mask for force grid with exclude postions with no indentation data [Nx,Nz]
            forceContour (arr)     - 2D Array of coordinates for contours of constant force given by reference force across scan positons
```

```
            forceContourmask (arr) - 2D boolean array giving mask for force contour for zero values in which no reference force
    '''
    #  ----------------------------------------------------Force Grid calculation-----------------------------------------------------
    # Intialise force grid array
    forceGrid = np.zeros([len(X),len(Z)])

    # For all x and y coordinates in course grained/binned domain
    for i in range(len(X)):
        for j in range(len(Z)):

            # For each indentation coordinate
            for k in range(U2.shape[1]):

                # If equal to Y position then set corresponding forcee grid array value to force value for that position
                if U2[i,k] == Z[j]:
                    # If symmetric kwargs, use same value for both x/-x position, i.e. x[i]  = -x[i] = x[-i-1]. Use data from position with indentation data with les
                    if 'Symmetric' in kwargs.keys() and kwargs['Symmetric'] == True and np.count_nonzero(RF[i]==0) <= np.count_nonzero(RF[-i-1]==0) :
                        forceGrid[i,j] = RF[i,k]
                        forceGrid[-i-1,j] = RF[i,k]

                    # Otherwise use on y values for corresponding x position in scan
                    else:
                        forceGrid[i,j] = RF[i,k]


    #  ----------------------------------------------------Create Force Grid mask-----------------------------------------------------
    # Initialise mask array, 0 values include 1 excludes
    forceGridmask = np.zeros([len(X),len(Z)])

    # For scan positions in force array/ same as positions in X array
    for i in range(len(RF)):

        # Check how maNz non-zero values there are for each postion
        k = [ k for k,v in enumerate(forceGrid[i]) if v != 0]

        # If there are non zero values
        if len(k)!=0:
            # Mask all grid values upto the first non zero force value position
            for j in range(k[0]):
                forceGridmask[i,j] = 1

        # If all force values are zero
        else:
            # Mask all y positions in force grid for those forces
            k = [ k for k,v in enumerate(forceGrid[i]) if Z[k] == U2[i,0] ]
            for j in range(k[0]):
                forceGridmask[i,j] = 1

    return forceGrid, forceGridmask
```

In [ ]:
```
def ForceContour2D(U2, RF, rackPos, forceRef, **kwargs):
    '''
    Function to calculate contours/z heights of constant force in simulation data for given threshold force.

        Parameters:
            U2 (arr)            - Array of indentors y indentor position over scan ( As opposed to displacement into surface given from simulation and used elsewhere
            RF (arr)            - Array of reaction force on indentor reference point
            rackPos (arr)       - Array of coordinates (x,z) of scan positions to image biomolecule [Nb,[x,z]]
            forceRef (float)    - Threshold force to evaluate indentation contours at (pN)

            kwargs:
                Symmetric - If false skip postprocessing step to produce AFM image from data {Default: True}

        Return:
            forceContour (arr)     - 2D Array of coordinates for contours of constant force given by reference force across scan positons
            forceContourmask (arr) - 2D boolean array giving mask for force contour for zero values in which no reference force
    '''
    #  ----------------------------------------------------Force Contour Calculation-----------------------------------------------------
    # Initialise arrays
    forceContour = np.zeros([len(RF),2])
    forceContourmask = np.zeros([len(RF), 2])

    # For scan positions in force array/ same as positions in X array
    for i in range(len(RF)):

        # If maximum at this position is greater than Reference force
        if np.max(RF[i]) >= forceRef:

            # Return index at which force is greater than force threshold
            j = [ k for k,v in enumerate(RF[i]) if v >= forceRef][0]

            if 'Symmetric' in kwargs.keys() and kwargs['Symmetric'] == True and np.count_nonzero(RF[i]==0) <= np.count_nonzero(RF[-i-1]==0) :
                # Store corrsponding depth/ Y position and X position for the index
                forceContour[i] = np.array([ rackPos[i,0], U2[i,j] ])
                forceContour[-i-1] = np.array([ rackPos[-i-1,0], U2[i,j] ])

            else:
                # Store corrsponding depth/ Y position and X position for the index
                forceContour[i] = np.array([ rackPos[i,0], U2[i,j] ])

        else:
            # If not symmetrical the position is automatically masked
            if 'Symmetric' not in kwargs.keys() or kwargs['Symmetric'] == False:
                # Otherwise position not above force reference, therefore set mask values to 1
                forceContourmask[i] = np.ones(2)

            # However, if symmetrical the position is only masked if the symmetric data is also less than force(as data can be take from either position to produce c
            elif 'Symmetric' in kwargs.keys() and kwargs['Symmetric'] == True and np.max(RF[-i-1]) < forceRef:
                # Otherwise position not above force reference, therefore set mask values to 1
                forceContourmask[i] = np.ones(2)

    return forceContour, forceContourmask
```

Force Interpolation Function

Calculate a 2D force heatmap over the xz domain, produced from interpolated forces using Hertz model.

In [ ]:
```
def F_Hertz(U, E, rIndentor, rSurface, elasticProperties):
    '''Hertzian fit for indentation data'''
    E_true, v =  elasticProperties
    R_eff = 1/(1/rSurface +1/rIndentor)
    return (2/3) * (E/(1-v**2)) * np.sqrt(R_eff) * U**(3/2)
```

In [ ]:
```
def ForceInterpolation(Xgrid, Zgrid, U2, RF, rackPos, rIndentor, rSurface, elasticProperties):
    '''
```

```
    Calculate a 2D force heatmap over the xz domain, produced from interpolated forces using Hertz model.

        Parameters:
            Xgrid (arr)             - 2D array/ grid of postions over xz domain of scan positions
            Zgrid (arr)             - 2D array/ grid of postions over xz domain of scan positions
            U2 (arr)                - Array of indentors y displacement in time over scan position and  for one indenter [Ni, Nb, Nt]
            RF (arr)                - Array of reaction force in time on indentor reference point over scan position  and for one indenter [Nb, Nt]
            rackPos (arr)           - Array of initial scan positions for one indenter [Nb, [x, z]]
            rIndentor (float)       - Indentor radius of spherical tip portion varied for seperate  simulations
            rSurface (float)        - Radius of semi-sphere surface
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]

        Return:
            E_hertz (arr)      - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
            F (arr)            - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
    '''
    # Initialise array to hold elastic modulus
    E_hertz = np.zeros(len(rackPos))

    # Fit Hertz equation to force/indentation for each x scan positon
    for i, value in enumerate(rackPos):
        u2, rf        = abs(U2[i]), abs(RF[i])
        popt, pcov    = curve_fit(lambda x, E: F_Hertz(x, E, rIndentor, rSurface, elasticProperties), u2, rf)

        # Produce array of fitted elastic modulus over scan positions for each indentor
        E_hertz[i]  = popt

    # Use Elastic modulus over scan position to produce continous spline
    ESpline = UnivariateSpline(rackPos[:,0],  E_hertz, s=1)
    # From spline interpolate youngs modulus over x domain
    E = ESpline(abs(Xgrid))

    # Create spline for initial scan positions
    rackSpline = UnivariateSpline(rackPos[:,0], rackPos[:,1], s = 0.001)
    # Calculate initial scan positions of x domain using scan position spline
    Zinit = F_TipConvolution(Xgrid, rSurface, rIndentor)

    # Use Hertz Eq to interpolate force over xz grid: (Yinit-Ygrid) gives indentation depths over grid
    F = F_Hertz(Zinit - Zgrid, E, rIndentor, rSurface, elasticProperties)

    return F, E_hertz
```

FWHM and Volume

In [ ]:

```
def FWHM_Volume(forceContour, NrackPos, Nf, Ni, indentorRadius, rSurface):
    '''
    Calculate Full Width Half Maxima and Volume for Force Contours of varying reference force using splines

        Parameters:
            forceContour (arr)      - 2D Array of coordinates for contours of constant force given by reference force across scan positons
                                      for all indentor and reference force [Nf,Ni, Nb, [x,z]] (With mask applied).
            NrackPos (arr)          - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
            Nf                      - Number if reference force values
            Ni                      - Number if indentor radii/ values
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            rSurface (float)        - Radius of semi-sphere surface

        Return:
            FWHM (arr)         - Array of full width half maxima of force contour for corresponding indentor and reference force [Nf,Ni]
            Volume (arr)       - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
    '''
    #  ----------------------------------------Calculate Volume and Full Width Half Maxima----------------------------------------
    # Intialise arrays for to store volume and FWHM, for each indentor size and reference forces
    FWHM, Volume = np.zeros([Nf+1, Ni]), np.zeros([Nf+1, Ni])

    # Loop  for each indentor size and reference forces
    for n, rIndentor in enumerate(indentorRadius):

        # Calculate a overestimate of boundary of integration for volume using geometry to find root of hard sphere
        a = 1.1*np.sqrt( (rSurface + rIndentor)**2 - rIndentor**2 )
        x = np.linspace(-a,a,100)

        # --------------------Set first values as hardsphere boundary for each fwhm and volume -------------------------------------
        # Create spline of hard sphere intial heights
        Fx, Fz = NrackPos[n,:,0], NrackPos[n,:,1]
        forceSpline = UnivariateSpline(Fx, Fz, s = 0.1)

        # Find definite bounds of integral using the spline
        b = UnivariateSpline(x, forceSpline(x), s = 0.1).roots()

        # Volume can be found by integrating contour spline over bounds - cannot integrate force spline itsels as integral will
        # only be over data range in that case
        Volume[0,n] =  UnivariateSpline(x, forceSpline(x), s = 0.01).integral(b[0], b[1])

        # Half maxima can be calculated by finding roots of spline that is tanslated vertically by half the maximum  y value
        FWHM[0,n] =  abs(UnivariateSpline(x, forceSpline(x)-rSurface/2, s = 0.1).roots()[0])


        # ------------------------------------Set fwhm and volume values for each force contour--------------------------------------
        for m  in range(Nf):
            # Extract xz compontents of force contour - compressed removees masked values
            Fx, Fz = forceContour[m,n,:,0].compressed(), forceContour[m,n,:,1].compressed()

            # Use try loop to avoid error for contours that cannot produce splines theat
            try:
                # Half maxima can be calculated by finding roots of spline that is tanslated vertically by half the maximum  y value
                FWHM[m,n] = abs(UnivariateSpline(Fx, Fz-Fz.max()/2, s = 0.1).roots()[1])
            except:
                None

            try:
                # Create a spline through contour points for integral
                forceSpline = UnivariateSpline(Fx, Fz, s = 0.1)

                # Find definite bounds of integral
                b = UnivariateSpline(x, forceSpline(x), s = 0.1).roots()

                # Volume can be found by integrating contour spline over bounds- cannot integrate force spline itsels as integral
                # will only be over data range in that case
                Volume[m,n] =  UnivariateSpline(x, forceSpline(x), s = 0.01).integral(b[0], b[1])

            except:
                None

    return FWHM, Volume
```

Postprocessing

In [ ]:
```
def Postprocessing(TotalU2, TotalRF, NrackPos, Nb, courseGrain, refForces, indentorRadius, baseDims, rSurface, elasticProperties, **kwargs):
    '''
    Calculate a 2D force heatmap produced from simulation over the xz domain.

        Parameters:
            TotalU2 (arr)           - Array of indentors y displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
            TotalRF (arr)           - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
            NrackPos (arr)          - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
            Nb (int)                - Number of scan positions along x axis of base
            courseGrain (float)     - Width of bins that subdivid xz domain of raster scanning/ spacing of the positions sampled over
            refForces (arr)         - Array of threshold force to evaluate indentation contours at (pN)
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            baseDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            rSurface (float)        - Radius of semi-sphere surface
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]

        Return:
            X (arr)            - 1D array of postions over x domain of scan positions
            Z (arr)            - 1D array of postions over z domain of scan positions, discretised into bins of courseGrain value
            forceGrid (arr)    - 2D Array of force heatmap over xz domain of scan i.e. grid of xz positions with associated force
                                 for all indentors and reference force [Nf, Ni, Nb, Nz] (With mask applied).
            forceContour (arr) - 2D Array of coordinates for contours of constant force given by reference force across scan positons
                                 for all indentor and reference force [Nf,Ni, Nb, [x,z]] (With mask applied).
            FWHM (arr)         - Array of full width half maxima of force contour for corresponding indentor and reference force [Nf,Ni]
            Volume (arr)       - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
            E_hertz (arr)      - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
            F (arr)            - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
    '''

    #  ----------------------------------------------------Initialise variables-------------------------------------------------------
    Nf = len(refForces)
    Ni = len(indentorRadius)

    # Convert indentation data to indentor Y displacement and discretise values into bins of width given by course grain value
    zIndentor = (TotalU2 + NrackPos[:,:,1, None])
    U2 = courseGrain*np.round(zIndentor/courseGrain)

    # Set X arrays of scan positions
    X = NrackPos[0,:,0]

    # Produce Y arrays over all Y domain of indentor position for all indentors
    Z = np.round(np.arange(U2.min(initial=0), U2.max()+courseGrain,courseGrain)*10)/10


    #  ----------------------------------------------------Set force grid and force contour---------------------------------------
    # Intialise arrays for all indentor size and reference forces
    forceContour, forceContourmask = np.zeros([Nf, Ni , Nb, 2]),     np.zeros([Nf, Ni , Nb, 2])
    forceGrid,    forceGridmask    = np.zeros([Nf, Ni, Nb, len(Z)]), np.zeros([Nf, Ni, Nb, len(Z)])

    # Set force grid and force contour for each indentor and refence force
    for m  in range(Nf):
        for n in range(Ni):
            forceGrid[m,n],    forceGridmask[m,n]    = ForceGrid2D(    X, Z, U2[n], TotalRF[n], NrackPos[n], courseGrain, **kwargs)
            forceContour[m,n], forceContourmask[m,n] = ForceContour2D(zIndentor[n], TotalRF[n], NrackPos[n], refForces[m], **kwargs)

    # Mask force grid excluding postions with no indentation data [Nx,Nz] and mask force contour for zero values in which below reference force
    forceGrid    = np.ma.masked_array(forceGrid, mask=forceGridmask)
    forceContour = np.ma.masked_array(forceContour, mask=forceContourmask)


    #  ----------------------------------------Calculate Volume and Full Width Half Maxima-----------------------------------------
    # Intialise arrays for to store volume and FWHM, for each indentor size and reference forces
    FWHM, Volume = FWHM_Volume(forceContour, NrackPos, Nf, Ni, indentorRadius, rSurface)

    # Mask values equal to zero
    FWHM   = np.ma.masked_equal(FWHM, 0)
    Volume = np.ma.masked_equal(Volume, 0)


    #  ----------------------------------------------------Calculate Hertz fit and interpolate force from the fit-----------------------------------------------------
    # Initialise grid arrays over xz domain
    X0     = np.linspace(-baseDims[0]/2, baseDims[0]/2, 250)
    Z0     = np.linspace(0, rSurface, 250)
    Xgrid, Zgrid = np.meshgrid(X0,Z0)

    # Initialise array holding Fitted Elastic modulus and Interpolated force
    E_hertz   = np.zeros([Ni,Nb])
    F = np.zeros([Ni, len(X0), len(Z0)])

    # For each indentor
    for n, rIndentor in enumerate(indentorRadius):
        F[n], E_hertz[n] = ForceInterpolation(Xgrid, Zgrid, TotalU2[n], TotalRF[n], NrackPos[n], rIndentor, rSurface, elasticProperties)

    return X, Z, forceContour, forceGrid, FWHM, Volume, E_hertz, F
```

Simulation Function

Final simulation function

In [ ]:
```
def AFMSimulation(host, port, username, password, scratch, wrkDir, localPath, abqCommand, fileName, subData,
                  indentorType, indentorRadius, theta_degrees, tip_length, indentionDepths, baseDims, rSurface,
                  refForces, courseGrain, binSize, clearance, meshSurface, meshIndentor, timePeriod, timeInterval,
                  elasticProperties, **kwargs):
    '''
    Final function to automate simulation. User inputs all variables and all results are outputted. The user gets a optionally get a surface plot of scan positions.
    Produces a heatmap of the AFM image, and 3D plots of the sample surface for given force threshold.

        Parameters:
            host (str)              - Hostname of the server to connect to
            port (int)              - Server port to connect to
            username (str)          - Username to authenticate as (defaults to the current local username)       -
            password (str)          - password (str) – Used for password authentication; is also used for private key decryption if passphrase is not given.
            scratch                 - Path to remote scratch directory
            wrkDir (str)            - Working directory extension
            localPath (str)         - Path to local file/directory
            abqCommand (str)        - Abaqus command to execute and run script
            fileName (str)          - Base File name for abaqus input files
            subData (str)           - Data for submission to serve queue [walltime, memory, cpus]

            indentorType (str)      - String defining indentor type (Spherical or Capped)
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
```

```
        theta_degrees (float)   - Principle conical angle from z axis in degrees
        tip_length (float)      - Total cone height
        indentionDepths (arr)   - Array of maximum indentation depth into surface
        baseDims (list)         - Dimension of base
        rSurface (float)        - Radius of semi-sphere

        refForces (float)       - Threshold force to evaluate indentation contours at, mimics feedback force in AFM (pN)
        courseGrain (float)     - Width of bins that subdivid xz domain of raster scanning/ spacing of the positions sampled over
        binSize(float)          - Width of bins that subdivid xz domain during raster scanning/ spacing of the positions sampled over
        clearance (float)       - Clearance above molecules surface indentor is set to during scan
        meshSurface (float)     - Value of indentor mesh given as bin size for vertices of geometry in nm (x10-9 m)
        meshIndentor (float)    - Value of indentor mesh given as bin size for vertices of geometry in nm (x10-9 m)
        timePeriod(float)       - Total time length for ABAQUS simulation/ time step (T)
        timeInterval(float)     - Time steps data sampled over for ABAQUS simulation/ time step (dt)
        elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]

        kwargs:
            Submission ('serial'/ 'paralell') - Type of submission, submit pararlell scripts or single serial script for scan locations {Default: 'serial'}
            Main (bool)        - If false skip preprocessing step of simulation {Default: True}
            SurfacePlot (bool) - If false skip surface plot of biomolecule and scan positions {Default: True}
            Queue (bool)       - If false skip queue completion step of simulation {Default: True}
            Analysis (bool)    - If false skip odb analysis step of simulation {Default: True}
            Retrieval (bool)   - If false skip data file retrivial from remote serve {Default: True}
            Compile(int)       - If passed, simulation data is compiled from seperate sets of simulations in directory in remote server to combine
                                 complete indentations. Value is set as int representing the range of directories to compile from (directories must have
                                 same root naming convention with int denoting individual directories)                    -
            Postprocess (bool) - If false skip postprocessing step to produce AFM image from data {Default: True}
            DataPlot (bool)    - If false skip scatter plot of simulation data {Default: True}
            Symmetric          - If false skip postprocessing step to produce AFM image from data {Default: True}

    Returns:
        X (arr)            - 1D array of postions over x domain of scan positions, discretised into bins of courseGrain value [Nx]
        Z (arr)            - 1D array of postions over z domain of scan positions, discretised into bins of courseGrain value [Nz]
        TotalU2 (arr)      - Array of indentors z displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
        TotalRF (arr)      - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
        NrackPos (arr)     - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
        forceGrid (arr)    - 2D Array of force heatmap over xz domain of scan i.e. grid of xz positions with associated force [Nx,Nz] (With mask applied).
        forceContour (arr) - 2D Array of coordinates for contours of constant force given by reference force across scan positons (With mask applied).
        FWHM (arr)         - Array of full width half maxima of force contour for corresponding indentor and reference force [Nf,Ni]
        Volume (arr)       - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
        E_hertz (arr)      - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
        F (arr)            - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
'''

# Set intial time
T0 = time.time()

#  ----------------------------------------------------Main Simulations---------------------------------------------------
if 'Main' not in kwargs.keys() or kwargs['Main'] == True:
    t0 = time.time()

    # For each indentor radius prdoduce main simulation and submit
    for index, rIndentor in enumerate(indentorRadius):
        print('Indentor Radius - ', rIndentor + '\n')

        #  ----------------------------------------Raster Scan Positions ---------------------------------------------
        # Calculate tip geometry to create indentor and calculate scan positions over molecule for imaging
        indentionDepth = clearance + indentionDepths[index]
        Nb, Nt= int(baseDims[0]/binSize)+1, int(timePeriod/ timeInterval)+1

        tipDims = TipStructure(rIndentor, theta_degrees, tip_length)
        rackPos = ScanGeometry(indentorType, tipDims, baseDims, Nb, clearance)

        # Option plot for scan positions for visualisation
        if 'SurfacePlot' in kwargs.keys() and kwargs['SurfacePlot'] == rIndentor:
            SurfacePlot(rackPos, Nb, baseDims, tipDims, rSurface, binSize, clearance)


        #  ---------------------------------------------Export Variable----------------------------------------------------
        # Set list of simulation variables and export to current directory
        variables = [timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface, rSurface]
        ExportVariables(rackPos, variables, baseDims, tipDims, indentorType, elasticProperties )


        #  ---------------------------------------------Remote Submission--------------------------------------------------
        remotePath = scratch + wrkDir +'/IndenterRadius'+str(int(rIndentor))

        abqfiles = ('AFMtestRasterScan.py','AFMtestODBAnalysis.py')
        csvfiles = ( "rackPos.csv", "variables.csv","baseDims.csv", "tipDims.csv", "indentorType.txt", "elasticProperties.csv" )

        RemoteSimulation(host, port, username, password, remotePath, localPath,  csvfiles, abqfiles, abqCommand, fileName, subData, rackPos, **kwargs)

    t1 = time.time()
    print('Main Submission Complete - ' + str(timedelta(seconds=t1-t0)) + '\n' )


#  ----------------------------------------------------Queue Status------------------------------------------------------
if 'Queue' not in kwargs.keys() or kwargs['Queue'] == True:
    t0 = time.time()
    print('Simulations Processing ...')

    # Wait for completion when queue is empty
    QueueCompletion(host, port, username, password)

    t1 = time.time()
    print('ABAQUS Simulation Complete - '+ str(timedelta(seconds=t1-t0)) + '\n')


#  ----------------------------------------ODB Analysis Submission---------------------------------------------------
if 'Analysis' not in kwargs.keys() or kwargs['Analysis'] == True:
    t0 = time.time()
    print('Running ODB Analysis...')


    # For each indentor radius
    for index, rIndentor in enumerate(indentorRadius):
        print('Indentor Radius:', rIndentor)

        # ODB analysis script to run, extracts data from simulation and sets it in csv file on server
        script     = 'AFMtestODBAnalysis.py'
        remotePath = scratch + wrkDir + '/IndenterRadius'+str(int(rIndentor))

        RemoteCommand(host, port, username, password, script, remotePath, abqCommand)

    t1 = time.time()
    print('ODB Analysis Complete - ' + str(timedelta(seconds=t1-t0)) + '\n')
```

```
    #  ------------------------------------------------File Retrieval----------------------------------------------------------
    if 'Retrieval' not in kwargs.keys() or kwargs['Retrieval'] == True:
        t0 = time.time()
        print('Running File Retrieval...')

        # Retrieve variables used for given simulation (in case variables redefined when skip kwargs used)
        dataFiles = ('U2_Results.csv','RF_Results.csv', 'rackPos.csv')
        csvfiles  = ( "rackPos.csv", "variables.csv","baseDims.csv", "tipDims.csv")

        variables, TotalU2, TotalRF, NrackPos = DataRetrieval(host, port, username, password, scratch, wrkDir, localPath, csvfiles, dataFiles, indentorRadius, **kwar

        # Export simulation data so it is saved in current directory for future use (save as a 2d array instead of 3d)
        np.savetxt("variables.csv", variables, fmt='%s', delimiter=",")
        np.savetxt("TotalU2.csv", TotalU2.reshape(TotalU2.shape[0], -1), fmt='%s', delimiter=",")
        np.savetxt("TotalRF.csv", TotalRF.reshape(TotalRF.shape[0], -1), fmt='%s', delimiter=",")
        np.savetxt("NrackPos.csv", NrackPos.reshape(NrackPos.shape[0], -1), fmt='%s', delimiter=",")

        t1 = time.time()
        print('File Retrevial Complete' + '\n')


    #  ------------------------------------------------- Post-Processing----------------------------------------------------------
    if  'Postprocess' not in kwargs.keys() or kwargs['Postprocess'] == True:

        t0 = time.time()
        print('Running Postprocessing...')

        # Check if simulation files are accessible in curent directory to use if pre=processing skipped
        try:
            variables, baseDims, rackPos  = ImportVariables()
            timePeriod, timeInterval, binSize, indentionDepth, meshIndentor, meshSurface, rSurface = variables
            Nb, Nt= int(baseDims[0]/binSize)+1, int(timePeriod/ timeInterval)+1

            # Load saved simulation data and reshape as data sved as 2d array,  true shape is 3d
            TotalU2  = np.array(np.loadtxt('TotalU2.csv', delimiter=",")).reshape(len(indentorRadius), Nb, Nt)
            TotalRF  = np.array(np.loadtxt('TotalRF.csv', delimiter=",")).reshape(len(indentorRadius), Nb, Nt)
            NrackPos = np.array(np.loadtxt('NrackPos.csv', delimiter=",")).reshape(len(indentorRadius), Nb, 2)

        # If file missing prompt user to import/ produce files
        except:
            print('No Simulation files available, run preprocessing or import data' + '\n')


        #  ------------------------------------------------- Data-Processing----------------------------------------------------------
        # Visualise data if set in kwargs
        if 'DataPlot' in kwargs.keys() and kwargs['DataPlot'] == True:
            n = 0
            DataPlot(NrackPos, TotalU2, TotalRF, Nb, Nt, n)

        # Process simulation data to produce heat map, analyse force contours, full width half maximum, volume and youngs modulus
        X, Z, forceContour, forceGrid, FWHM, Volume, E_hertz, F = Postprocessing(TotalU2, TotalRF, NrackPos, Nb, courseGrain, refForces,
                                                                                 indentorRadius,  baseDims, rSurface, elasticProperties, **kwargs)

        t1 = time.time()
        print('Postprocessing Complete' + '\n')


        # Return final time of simulation and variables
        T1 = time.time()
        print('Simulation Complete - ' + str(timedelta(seconds=T1-T0)) )
        return X, Z, TotalU2, TotalRF, NrackPos, forceContour, forceGrid, FWHM, Volume, E_hertz, F

    else:
        # Return final time of simulation
        T1 = time.time()
        print('Simulation Complete - ' + str(timedelta(seconds=T1-T0)) )
        return None, None, None, None, None, None, None, None, None, None, None
```

Plot Functions

Code to plot data from the simulation

Illustrative Surface Plot

In [ ]:
```
def SurfacePlot(rackPos, Nb, baseDims, tipDims, rSurface, binSize, clearance):
    '''
    Plot the surfaces and scan positions to visualise and check positions.

        Parameters:
            rackPos (arr)      - Array of coordinates [x,z] of scan positions to image biomolecule
            Nb (int)           - Number of scan positions along x axis of base
            baseDims (list)    - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)     - Geometric parameters for defining capped tip structure
            rSurface (float)   - Radius of spherical surface
            clearance (float)  - Clearance above molecules surface indentor is set to during scan
    '''

    #  ------------------------------------------------Raster Scan positions ----------------------------------------------------------
    # Set tip dimensional variables
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims
    Nb = int(baseDims[0]/binSize)+1

    # Raster scan grid positions for spherical and capped tip
    rackPosSphere = ScanGeometry('Spherical', tipDims, baseDims, Nb, clearance)
    rackPosCone = ScanGeometry('Capped', tipDims, baseDims, Nb, clearance)

    # Set index for scan position to plot tip at
    i, j = int(Nb/2), -6

    # Produce spherical tip with polar coordinates
    x = np.linspace(-7.5, 7.5, 1000)
    t = np.linspace(-np.pi, np.pi, 1000)
    x1 = rIndentor*np.cos(t)
    y1 = rIndentor*np.sin(t)


    #  ------------------------------------------------Plot Points ----------------------------------------------------------
    # Create figure for scan positions
    fig, ax = plt.subplots(1,1, figsize = (11.69/3, 8.27/3 ))

    # Plot spherical and conical scan position as points
    ax.plot(rackPosCone[:,0], rackPosCone[:,1], '.', color = 'r')
```

```
    # Plot lines for the semi-sphere surface and base using same array as indentor
    ax.plot(rSurface*np.cos(t[500:]),rSurface*np.sin(t[500:]),'k')
    ax.plot(x, 0*x, 'k')

    # Plot the geometry of spherical and conical tip at index i
    ax.plot(x+rackPosCone[i,0], Fconical(x, 0, r_int, z_int, theta, rIndentor, 7)+rackPosCone[i,1], color = '#5a71ad')
    ax.plot(x1+rackPosSphere[i,0], y1+rIndentor+rackPosSphere[i,1], ':', color = '#fc8535')

    # Plot the geometry of spherical and conical tip at index j
    ax.plot(x+rackPosCone[j,0], Fconical(x, 0, r_int, z_int, theta, rIndentor, 10)+rackPosCone[j,1], color = '#5a71ad')
    ax.plot(x1+rackPosSphere[j,0], y1+rIndentor+rackPosSphere[j,1], ':', color = '#fc8535')

    # Set axis labels, title and legend
    ax.set_xlim(-baseDims[0]/2, baseDims[0]/2)
    ax.set_ylim(0, (2*rSurface))
    ax.set_xlabel(r'$\frac{x}{r}$', color='white')
    ax.set_ylabel(r'$\frac{z}{r}$', rotation=0,  labelpad = 10, color='white')
    ax.axes.set_aspect('equal')


    # Place axis on right for desired spacing
    ax.yaxis.set_label_position("right")
    ax.yaxis.tick_right()

    ax.spines['top'].set_color('white')
    ax.spines['bottom'].set_color('white')
    ax.spines['left'].set_color('white')
    ax.spines['right'].set_color('white')
    ax.tick_params(colors='white')

    # Annotating Diagram
    ax.text( -1, rSurface + rIndentor/2, 'R', color = '#fc8535')
    ax.annotate('', color = '#fc8535',
                xy=(0, rSurface), xycoords='data',
                xytext=(0, rSurface+rIndentor), textcoords='data',
                arrowprops=dict(arrowstyle="-", connectionstyle="arc3", color='#fc8535'))

    ax.text(rSurface/2, 0.75, 'r')
    ax.annotate('', xy=(rSurface, 0.5), xycoords='data',
                xytext=(0, 0.5), textcoords='data',
                arrowprops=dict(arrowstyle="<->", connectionstyle="arc3", color='k'))


    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-SetUp.png', bbox_inches = 'tight')
    plt.show()
```

Data Plot

Function to produces scatter plot of indentation depth and reaction force to visualise and check simulation data.

In [ ]:
```
def DataPlot(NrackPos, TotalU2, TotalRF, Nb, Nt, n):
    '''
    Produces scatter plot of indentation depth and reaction force to visualise and check simulation data.

        Parameters:
            NrackPos (arr) - Array of initial scan positions for each indenter [Ni, Nb, [x, z] ]
            TotalU2 (arr)  - Array of indentors z displacement in time over scan position and  for all indenter [Ni, Nb, Nt]
            TotalRF (arr)  - Array of reaction force in time on indentor reference point over scan position  and for all indenter [Ni, Nb, Nt]
            Nb (int)       - Number of scan positions along x axis of base
            Nt(int)        - Number of frames in  ABAQUS simulation/ time step
            n (int)        - Index of indenter data to plot corresponding to indices in indenterRadius

    '''
    # Force Curves for all the data
    fig, ax = plt.subplots(1,1)
    for i in range(len(TotalRF)):
        ax.plot(TotalU2[i],TotalRF[i])


    # Initialise array for indentor force and displacement
    tipPos   = np.zeros([Nb*Nt, 2])
    tipForce = np.zeros(Nb*Nt)

    # Initialise count
    k = 0

    # Loop over array indices
    for i in range(Nb):
        for j in range( Nt ):
            #  Set array values for tip force and displacement
            tipPos[k]   = [NrackPos[n,i,0], NrackPos[n,i,1] + TotalU2[n,i,j]]
            tipForce[k] = TotalRF[n,i,j]

            # Count array index
            k += 1

    # Scatter plot indentor displacement over scan positions
    fig ,ax = plt.subplots(1,1)
    ax.plot(tipPos[:,0], tipPos[:,1], '.')

    ax.set_xlabel(r'x (nm)', labelpad = 25)
    ax.set_ylabel(r'y (nm)', labelpad = 25)
    ax.set_title('Tip Position for Raster Scan')
    plt.show()


    # Scatter plot of force over scan positions
    fig, ax = plt.subplots(1,1)
    ax.plot(tipPos[:,0], tipForce, '.')

    ax.set_xlabel(r'x (nm)', labelpad = 25)
    ax.set_ylabel('F (pN)',labelpad = 25)
    ax.set_title('Force Scatter Plot for Raster Scan')
    plt.show()
```

Contour Plot

Interpolate

In [ ]:
```
def ContourPlot(X, Z, forceGrid, forceContour, refForces, baseDims, tipDims, rSurface, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force heatmap produced from simulation over the xz domain for single indenter and refereance force.
```

```
    Parameters:
        X (arr)                - 1D array of x coordinates over scan positions
        Z (arr)                - 1D array of z coordinates over scan positions
        forceGrid (arr)        - 2D Array of force grid of xz positions
        forceContour( arr)     - 2D Array of coordinates for contours of constant force given by reference force
        refForces (float)      - Threshold force to evaluate indentation contours at (pN)
        baseDims (list)        - Dimension of base
        tipDims (list)         - Geometric parameters for defining capped tip structure
        rSurface (float)       - Radius of semi-sphere
        elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
        normalizer (obj)       -  Normalisation of cmap
        maxRF (float)          - Maximum Force value
        contrast (float)       - Contrast between high and low values in AFM heat map (0-1)
    '''
    #  ----------------------------------------------------Set Variable----------------------------------------------------
    # Set material and tip  Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)
    # Produce spherical tip with polar coordinates
    x = np.linspace(-np.pi/2,np.pi/2, 100)

    # Extract xz compontents of force contour - compressed removes masked values
    Fx, Fz = np.array(forceContour[:,0].compressed()), np.array(forceContour[:,1].compressed())
    # Connect contour points smoothly with a spline
    forceSpline = UnivariateSpline(Fx, Fz, s = 0.1)

    # Increase padding to add above surface to show indentor
    hPadding = 1

    #  ----------------------------------------------------2D Plots----------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3))

    # 2D heat map plot without interpolation
    im = ax.imshow(forceGrid.T/F_dim, origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm= normalizer,
                   extent = (X[0]/rSurface, X[-1]/rSurface, Z[0]/rSurface, Z[-1]/rSurface), interpolation_stage = 'rgba')

    # Plot spline force for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(X/rSurface,  forceSpline(X)/rSurface,                                    color = 'r', lw = 1, label = 'Fitted Fource Contour')
    ax.plot(X/rSurface,  F_TipConvolution(X, rSurface, rIndentor)/rSurface, ':',     color = 'r', lw = 1, label = 'Hard Sphere boundary')
    ax.plot(np.sin(x),   np.cos(x), ':',                                             color = 'w', lw = 1, label = 'Surface boundary')

    # Plot indentor geometry
    ax.plot((7.5*2/np.pi*x)/rSurface, (Fconical(7.5*2/np.pi*x, 0, r_int, z_int, theta, rIndentor, tip_length)+rSurface)/rSurface,
            color = 'w', lw = 1, label = 'Indentor boundary')

    # Add 0 values in image for region above surface to have continous colour
    ax.imshow(np.zeros([10,10]), origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm= normalizer,
              extent = (X[0]/rSurface, X[-1]/rSurface, 1, ((1+hPadding)*rSurface)/rSurface) )

    # Set legend and axis labels, limits and title
    ax.set_xlabel(r'$\frac{x}{r}$')
    ax.set_ylabel(r'$\frac{z}{r}$', rotation=0,  labelpad = 15)
    ax.set_xlim(X[0]/rSurface, X[-1]/rSurface)
    ax.set_ylim(0, ((1+hPadding)*rSurface)/rSurface)
    ax.set_facecolor("grey")
    ax.axes.set_aspect('equal')

    # ------------------------------------------------Plot color bar ----------------------------------------------------
    # plt.legend(loc = [1.35,0.5], facecolor='#d1d1d1', edgecolor = '#d1d1d1')

    cbar= fig.colorbar(im, ax= ax , orientation = 'vertical', fraction=0.0315, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-ContourPlot-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

No Interpolation

In [ ]:

```
def ContourPlotNI(X, Z, forceGrid, forceContour, refForces, baseDims, tipDims, rSurface, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force heatmap produced from simulation over the xz domain for single indenter and refereance force.

        Parameters:
            X (arr)                - 1D array of x coordinates over scan positions
            Z (arr)                - 1D array of z coordinates over scan positions
            RF(arr)                - Array of reaction force on indentor reference point
            forceGrid (arr)        - 2D Array of force grid of xz positions
            forceContour( arr)     - 2D Array of coordinates for contours of constant force given by reference force
            refForces (float)      - Threshold force to evaluate indentation contours at (pN)
            baseDims (list)        - Dimension of base
            tipDims (list)         - Geometric parameters for defining capped tip structure
            rSurface (float)       - Radius of semi-sphere
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)       -  Normalisation of cmap
            maxRF (float)          - Maximum Force value
            contrast (float)       - Contrast between high and low values in AFM heat map (0-1)
    '''
    #  ----------------------------------------------------Set Variable----------------------------------------------------
    # Set material and tip  Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)
    # Produce spherical tip with polar coordinates
    x = np.linspace(-np.pi/2,np.pi/2, 100)

    # Extract xz compontents of force contour - compressed removes masked values
    Fx, Fz = np.array(forceContour[:,0].compressed()), np.array(forceContour[:,1].compressed())
    # Connect contour points smoothly with a spline
    forceSpline = UnivariateSpline(Fx, Fz, s = 0.1)

    # Increase padding to add above surface to show indentor
    hPadding = 1
```

```
    #  ------------------------------------------------------2D Plots-----------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3))

    # 2D heat map plot without interpolation
    im = ax.imshow(forceGrid.T/F_dim, origin= 'lower', cmap='coolwarm', interpolation='None', norm= normalizer,
                    extent = (X[0]/rSurface, X[-1]/rSurface, Z[0]/rSurface, Z[-1]/rSurface))

    # Plot spline force for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(Fx/rSurface, Fz/rSurface, 'x', ms = 3,                                      color = 'k', lw = 2, label = 'Force contour for F= {0:.3f}'.format(refForces
    ax.plot(X/rSurface,  F_TipConvolution(X, rSurface, rIndentor)/rSurface, ':',        color = 'r', lw = 1, label = 'Hard Sphere boundary')
    ax.plot(np.sin(x),   np.cos(x), ':',                                                color = 'w', lw = 1, label = 'Surface boundary')

    # Plot indentor geometry
    ax.plot((7.5*2/np.pi*x)/rSurface, (Fconical(7.5*2/np.pi*x, 0, r_int, z_int, theta, rIndentor, tip_length)+rSurface)/rSurface,
            color = 'w', lw = 1, label = 'Indentor boundary')

    # Add 0 values in image for region above surface to have continous colour
    ax.imshow(np.zeros([10,10]), origin= 'lower', cmap='coolwarm', interpolation='None', norm= normalizer,
              extent = (X[0]/rSurface, X[-1]/rSurface, 1, ((1+hPadding)*rSurface)/rSurface) )

    # Set legend and axis labels, limits and title
    ax.set_xlabel(r'$\frac{x}{r}$')
    ax.set_ylabel(r'$\frac{z}{r}$', rotation=0,  labelpad = 15)
    ax.set_xlim(X[0]/rSurface, X[-1]/rSurface)
    ax.set_ylim(0, ((1+hPadding)*rSurface)/rSurface)
    ax.set_facecolor("grey")
    ax.axes.set_aspect('equal')

    # ------------------------------------------Plot color bar ------------------------------------------------------------
    # plt.legend(loc = [1.35,0.5], facecolor='#d1d1d1', edgecolor = '#d1d1d1')

    cbar= fig.colorbar(im, ax= ax , orientation = 'vertical', fraction=0.0315, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-ContourPlotNI-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

Line

In [ ]:

```
def LineContourPlot(X, forceContour, refForces, rSurface, tipDims, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot fitted contour lines produced from simulation over the xz domain for single indenter and  range reference force.Interpolate/none.

        Parameters:
            X (arr)                 - 1D array of x coordinates over scan positions
            RF(arr)                 - Array of reaction force on indentor reference point
            forceContour( arr)      - 2D Array of coordinates for contours of constant force given by reference force
            refForces (float)       - Threshold force to evaluate indentation contours at (pN)
            rIndentor (arr)         - Array of indentor radii of spherical tip portion varied for seperate  simulations
            rSurface (float)        - Radius of semi-sphere
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)        -  Normalisation of cmap
            maxRF (float)           - Maximum Force value
            contrast (float)        - Contrast between high and low values in AFM heat map (0-1)
    '''
    #  ------------------------------------------------------Set Variable-------------------------------------------------
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)

    # Increase padding to add above surface to show indentor
    hPadding = 1

    # Set colourmap
    cmap = mpl.cm.coolwarm

    # Produce spherical tip with polar coordinates
    x = np.linspace(-np.pi/2,np.pi/2, 100)

    #  ------------------------------------------------------2D Plots-----------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize =  (11.69/3, 8.27/3))

    #  ------------------------------------------------------Interpolation------------------------------------------------
    # Plot spline force for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(X/rSurface,  F_TipConvolution(X, rSurface, rIndentor)/rSurface, ':', color = cmap(normalizer(0)), lw = 2, label = 'Hard\nSphere')
    ax.plot(np.sin(x),   np.cos(x), ':',                                         color = 'k', lw = 2, label = 'Surface')

    # Plot indentor geometry
    ax.plot((7.5*2/np.pi*x)/rSurface, (Fconical(7.5*2/np.pi*x, 0, r_int, z_int, theta, rIndentor, tip_length)+rSurface)/rSurface,
            color = 'k', lw = 1, label = 'Indentor')

    # Plot contour lines
    for m in range(len(refForces)):
        # Extract xz compontents of force contour - compressed removes masked values
        Fx, Fz = np.array(forceContour[m,:,0].compressed()), np.array(forceContour[m,:,1].compressed())
        try:
            # Connect contour points smoothly with a spline
            forceSpline = UnivariateSpline(Fx, Fz, s = 0.1)
        except:
            None
        else:
            ax.plot(X/rSurface,  forceSpline(X)/rSurface, color = cmap(normalizer(refForces[m]/F_dim)), lw = 1 )

    # Set legend and axis labels, limits and title
    ax.set_xlabel(r'$\frac{x}{r}$')
    ax.set_ylabel(r'$\frac{z}{r}$', rotation=0,  labelpad = 15)
    ax.set_xlim(X[0]/rSurface, X[-1]/rSurface)
    ax.set_ylim(0, ((1+hPadding)*rSurface)/rSurface)
    ax.axes.set_aspect('equal')

    # ------------------------------------------Plot color bar ------------------------------------------------------------
    # plt.legend( frameon=False)
    cbar = plt.colorbar(plt.cm.ScalarMappable(cmap=cmap, norm=normalizer), fraction=0.0315, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)
```

```
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-LineContour-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

Force Interpolation

In [ ]:
```
def FInterpolatePlot(X, Z, F, baseDims, tipDims, rSurface, elasticProperties, normalizer, maxRF, contrast):
    '''
    Function to plot a 2D force heatmap produced from simulation over the xz domain for single indenter and refereance force.

        Parameters:
            X (arr)                - 1D array of x coordinates over scan positions
            Z (arr)                - 1D array of z coordinates over scan positions
            F (arr)                - Array of interpolated force values over xz grid for all indentors and reference force [Ni, Nb, Nz]
            baseDims (list)        - Dimension of base
            tipDims (list)         - Geometric parameters for defining capped tip structure
            rSurface (float)       - Radius of semi-sphere
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
            normalizer (obj)       -  Normalisation of cmap
            maxRF (float)          - Maximum Force value
            contrast (float)       - Contrast between high and low values in AFM heat map (0-1)
    '''
    #  ----------------------------------------------------Set Variable----------------------------------------------------
    # Set material and tip  Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*rIndentor**2)
    # Produce spherical tip with polar coordinates
    x = np.linspace(-np.pi/2,np.pi/2, 100)

    # Increase padding to add above surface to show indentor
    hPadding = 1

    #  ----------------------------------------------------2D Plots----------------------------------------------------
    # Plot of force heatmaps using imshow to directly visualise 2D array
    fig, ax = plt.subplots(1, 1, figsize = (11.69/3, 8.27/3))

    # 2D heat map plot without interpolation
    im = ax.imshow(F/F_dim, origin= 'lower', cmap='coolwarm', interpolation='bicubic', norm= normalizer,
                   extent = (-baseDims[0]/(2*rSurface), baseDims[0]/(2*rSurface), 0, Z[-1]/rSurface), interpolation_stage = 'rgba')

    # Plot spline force for contour points, contour points themselves, surface boundary using polar coordinates, and hard sphere tip convolution
    ax.plot(X/rSurface,  F_TipConvolution(X, rSurface, indentorRadius[n])/rSurface, ':', color = 'r', lw = 1, label = 'Hard Sphere boundary')
    ax.plot(np.sin(x),   np.cos(x), ':',                                              color = 'w', lw = 1, label = 'Surface boundary')

    # Plot indentor geometry
    ax.plot((7.5*2/np.pi*x)/rSurface, (Fconical(7.5*2/np.pi*x, 0, r_int, z_int, theta, rIndentor, tip_length)+rSurface)/rSurface,
            color = 'w', lw = 1, label = 'Indentor boundary')

    # Set legend and axis Labels, limits and title
    ax.set_xlabel(r'$\frac{x}{r}$')
    ax.set_ylabel(r'$\frac{z}{r}$', rotation=0,  labelpad = 15)
    ax.set_xlim(X[0]/rSurface, X[-1]/rSurface)
    ax.set_ylim(0, ((1+hPadding)*rSurface)/rSurface)
    ax.set_facecolor("grey")
    ax.axes.set_aspect('equal')

    # ------------------------------------------Plot color bar ----------------------------------------------------------
    # plt.legend(Loc = [1.35,0.5], facecolor='#d1d1d1', edgecolor = '#d1d1d1')

    cbar= fig.colorbar(im, ax= ax , orientation = 'vertical', fraction=0.0315, pad=0.02)
    cbar.set_label(r'$\frac{F}{E*R^2}$', rotation=0)
    cbar.set_ticks(np.round(10*np.array([0, maxRF*0.25**(1/0.45), maxRF*0.75**(1/0.45), maxRF]))/10)
    cbar.ax.yaxis.set_label_coords(4, 0.6)

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-FInterpolate-'+str(rIndentor)+'.png', bbox_inches = 'tight')

    plt.show()
```

Full Width Half Maxima

In [ ]:
```
def FWHMPlot(FWHM, indentorRadius, refForces, rSurface, elasticProperties):
    '''
    Function to plot Full Width Half Maxima of force contour for each indentor for varying reference force.

        Parameters:
            FWHM (arr)           - 2D array of y coordinates over grid positions
            indentorRadius (arr) - 2D array of z coordinates of force contour over grid positions
            refForces (float)    - Threshold force to evaluate indentation contours at (pN)
            rSurface (float)     - Radius of semi-sphere
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''

    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)
    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*indentorRadius**2)

    # Calculate fwhm for hard sphere convolution, using geometric equation above
    halfMax = rSurface/2
    hsFWHM  = np.sqrt(3/4*rSurface**2) #np.array([ np.sqrt( (rSurface + indentorRadius[n])**2 - (halfMax+indentorRadius[n])**2 ) for n in range(len(indentorRadius))]

    # ------------------------------------------------Plot ----------------------------------------------------------------
    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, (1/1.61)*linewidth/3) )
    for n in range(len(indentorRadius)):
        # Plot fwhm for each indentor as they vary over reference force - add zero value to force array with insert (for hard sphere, F=0)
        ax.plot(np.insert((refForces),0,0)/F_dim[n], FWHM[:,n]/hsFWHM, lw = 1, label = r'$\frac{R}{r}$= '+str(indentorRadius[n]/rSurface))
        # ax.plot(refForces, hsFWHM[n]*refForces**0, ':', label = 'Hard Sphere Radius = '+str(indentorRadius[n]))

    # Set axis Label and legend
    ax.set_xlabel(r'$\frac{F}{E^*R^2}$', labelpad=5)
    ax.set_ylabel(r'$FWHM^*$')
    ax.set_xscale('log')
    # plt.legend(frameon=False)

    fig.subplots_adjust(bottom=0.15)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-FWHM.png', bbox_inches = 'tight')
    plt.show()
```

```
    fig2, ax2 = plt.subplots(1, 1, figsize = (linewidth, 1/20*linewidth))

    # create a legend for the legend plot
    ax2.legend(*ax.get_legend_handles_labels(), frameon=False, loc='center', ncol=5)
    # remove the axis from the legend plot
    ax2.axis('off')

    fig2.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-Legend.png', bbox_inches = 'tight')
    plt.show()
```

Volume

In [ ]:
```
def VolumePlot(Volume, indentorRadius, refForces, rSurface, elasticProperties):
    '''
    Function to plot volume under force contour for each indentor for varying reference force.

        Parameters:
            Volume (arr)            - Array of volume under force contour for corresponding indentor and reference force [Nf,Ni]
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            refForces (float)       - Threshold force to evaluate indentation contours at, mimics feedback force in AFM
            rSurface (float)        - Radius of semi-sphere
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''
    # Set material Propeties
    E_true, v = elasticProperties
    E_eff = E_true/(1-v**2)

    # Set constant to normalise dimensionaless forces
    F_dim = (E_eff*indentorRadius**2)

    # Calculate volume of surface portion
    surfaceVolume = 1/2*np.pi*rSurface**2


    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, (1/1.61)*linewidth/3))
    for n, rIndentor in enumerate(indentorRadius):
        # Plot for each indentor variation of volume over reference force- add zero value to force array with insert (for hard sphere, F=0)
        ax.plot(np.insert((refForces),0,0)/F_dim[n], Volume[:,n]/surfaceVolume, lw = 1, label = r'$\frac{R}{r}$= '+ str(rIndentor/rSurface))

    # Set axis label and legend
    ax.set_xlabel(r'$\frac{F}{E*R^2}$', labelpad=5)
    ax.set_ylabel(r'$\frac{V_{contour}}{V_{surface}}$')
    ax.set_xscale('log')
    # plt.legend(frameon=False)

    fig.subplots_adjust(bottom=0.15)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-Volume.png', bbox_inches = 'tight')
    plt.show()
```

Youngs Modulus

In [ ]:
```
def YoungPlot(E_hertz, indentorRadius, NrackPos, rSurface, elasticProperties):
    '''
    Function to plot elastic modulus over scan position for each indentor.

        Parameters:
            E_hertz (arr)           - Array of fitted elastic modulus value over scan positions for each indentor [Ni,Nb]
            indentorRadius (arr)    - Array of indentor radii of spherical tip portion varied for seperate  simulations
            NrackPos (arr)          - Array of initial scan positions for each indenter [Ni, Nb, [x, z]]
            rSurface (float)        - Radius of semi-sphere
            elasticProperties (arr) - Array of surface material properties, for elastic surface [Youngs Modulus, Poisson Ratio]
    '''
    E_true, v = elasticProperties

    fig, ax = plt.subplots(1, 1, figsize = (linewidth/3, (1/1.61)*linewidth/3) )
    for n, rIndentor in enumerate(indentorRadius):
        Nb2 = int(len(E_hertz[n])/2)
        Edata = np.append(E_hertz[n,Nb2:][::-1], E_hertz[n,Nb2+1:] )/E_true

        # Plot for each indentor variation of elastic modulus over scan positions
        ax.plot(NrackPos[n,:,0]/rSurface, Edata, lw=1, label = r'$\frac{R}{r}$='+ str(rIndentor/rSurface))

    # Expected elastic modulus value
    ax.plot(NrackPos[0,:,0]/rSurface, NrackPos[1,:,0]**0, ':', color = 'k', lw = 1)

    # Set axis label and legend
    ax.set_xlabel(r'$\frac{x}{r}$', labelpad=15)
    ax.set_ylabel(r'$\frac{E_{Fitted}}{E_{True}}$')
    # ax.Legend( frameon=False )

    fig.subplots_adjust(bottom=0.15)
    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\Hemisphere-Youngs.png', bbox_inches = 'tight')
    plt.show()
```

Simulation Interface

In [ ]:
```
#  ----------------------------------------Remote Variables----------------------------------------------------------
# host = "kathleen.rc.ucl.ac.uk"
host = "myriad.rc.ucl.ac.uk"
port = 22
username = "zcapjgi"
password = "ucl.Giblj003.315/Burnj003.315"
home     = '/home/zcapjgi'
scratch  = '/scratch/scratch/zcapjgi'

bashCommand = 'qsub'
abqCommand  = 'module load abaqus/2017 \n abaqus cae -noGUI'

# host = "128.40.163.27"
# port = 22
# username = "giblnbrnhm_j"
# password = "axenub13"
# home = '/home/giblnbrnhm_j@MECHENG2012'

# abqCommand  = '/opt/abaqus2018/abq2018 cae -noGUI'
# abqCommand  = '/opt/abaqus614/Commands/abq6141 cae -noGUI'
```

In [ ]:
```
#  ----------------------------------------Submission Variables----------------------------------------------------------
fileName = 'AFMtestRasterScan-Pos'
subData  = ['72:0:0', '100G', '1']
wrkDir   = '/ABAQUS/TipConvolutionTests/Test_Hemisphere5'
```

```
localPath  = os.getcwd()

#  ----------------------------------------Setup Variables-----------------------------------------------------------
# Surface variables
baseDims = [ 15, 15, 3 ]
rSurface = 5
E_true, v = 1000, 0.3
E_eff = E_true/(1-v**2)

# Indentor variables
indentorType  =  'Spherical' # 'Capped'
indentorRadius = np.array([i for i in range(1,2*rSurface,2)])    # angstrom - - 1 nm radius tip
theta_degrees = 20                                               # degrees
tip_length    = 50                                               # angstrom

# Scan variables
clearance = 0.1                                                  # angstrom
indentionDepths = [3.5,3.5,3.5,3.5,3.5] # (indentorRadius*3/2).clip(max = 2.5)            # angstrom
binSize = 0.5                                                    # angstrom
contrast = 1
courseGrain = 0.5
refForces = np.arange(250,4500,500)

# ABAQUS variable
timePeriod = 1.5     # s
timeInterval = 0.1   # s
meshSurface  = 0.8    # angstrom
meshIndentor = 0.4    # angstrom
elasticProperties = np.array([E_true, v])
print('Indentation Depths - ',indentionDepths, 'Indenter Radii - ', indentorRadius, '\n')

#  ----------------------------------------Simulation Script---------------------------------------------------------
X, Z, TotalU2, TotalRF, NrackPos, forceContour, forceGrid, FWHM, Volume, E_hertz, F = AFMSimulation(
    host, port, username, password, scratch, wrkDir, localPath, abqCommand, fileName, subData,
    indentorType, indentorRadius, theta_degrees, tip_length, indentionDepths, baseDims, rSurface,
    refForces, courseGrain, binSize, clearance, meshSurface, meshIndentor, timePeriod, timeInterval,
    elasticProperties,

    Submission = 'serial',
    Main = False,
    # SurfacePlot = 1,

    Queue     = False,

    Analysis  = False,
    Retrieval = False,
    Compile   = 6,

    Postprocess = True,
    DataPlot    = False,

    Symmetric   = True
)
```

Data Plots

Illustration

In [ ]:
```
rIndentor = 2

Nb, Nt= int(baseDims[0]/binSize)+1, int(timePeriod/ timeInterval)+1

tipDims = TipStructure(rIndentor, 20, tip_length)
rackPos = ScanGeometry(indentorType, tipDims, baseDims, Nb, clearance)

SurfacePlot(rackPos, Nb, baseDims, tipDims, rSurface, binSize, clearance)
```

2D Force Heat Maps and Fitted Force

In [ ]:
```
maxRF = (TotalRF/(E_eff*indentorRadius[:,None,None]**2)).max()
normalizer = mpl.colors.PowerNorm(0.45, 0, contrast*(maxRF))
```

Interpolate

In [ ]:
```
n = 4
m = 0
contrast = 1

tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

ContourPlot(X, Z, forceGrid[m,n], forceContour[m,n], refForces[m], baseDims, tipDims, rSurface, elasticProperties, normalizer, maxRF, contrast)
```

No Interplolate

In [ ]:
```
n = 4
m = 0
contrast = 1

tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

ContourPlotNI(X, Z, forceGrid[m,n], forceContour[m,n], refForces[m], baseDims, tipDims, rSurface, elasticProperties, normalizer, maxRF, contrast)
```

Line

In [ ]:
```
n = 4
tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

LineContourPlot(X, forceContour[:,n], refForces, rSurface, tipDims, elasticProperties, normalizer, maxRF, contrast)
```

Force Interpolation

In [ ]:
```
n = 4
contrast = 1
```

```
tipDims = TipStructure(indentorRadius[n], theta_degrees, tip_length)

FInterpolatePlot(X, Z, F[n], baseDims, tipDims, rSurface, elasticProperties, normalizer, maxRF, contrast)
```

Full Width Half Maximum

In [ ]:
```
FWHMPlot(FWHM, indentorRadius, refForces, rSurface, elasticProperties)
```

Volume Plot

In [ ]:
```
VolumePlot(Volume, indentorRadius, refForces, rSurface, elasticProperties )
```

Youngs Modulus

In [ ]:
```
YoungPlot(E_hertz, indentorRadius, NrackPos, rSurface, elasticProperties)
```

Bash Commands

In [ ]:
```
RemoteCommand(host, port, username, password, '', home, 'qstat')
```

In [ ]:
```
# RemoteCommand(host, port, username, password, '', home, "qdel '*' ")
```

In [ ]:
```
# t0 = time.time()
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.023' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.cid' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.dat' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.lck' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.mdl' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.com' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.msg' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.prt' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.sim' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.sta' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.stt' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.tmp' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.SMABulk' -delete")
# RemoteCommand( host, port, username, password, '', scratch, "find . -name '*.SMAFocus' -delete")

# t1 = time.time()
# print(t1-t0)
```

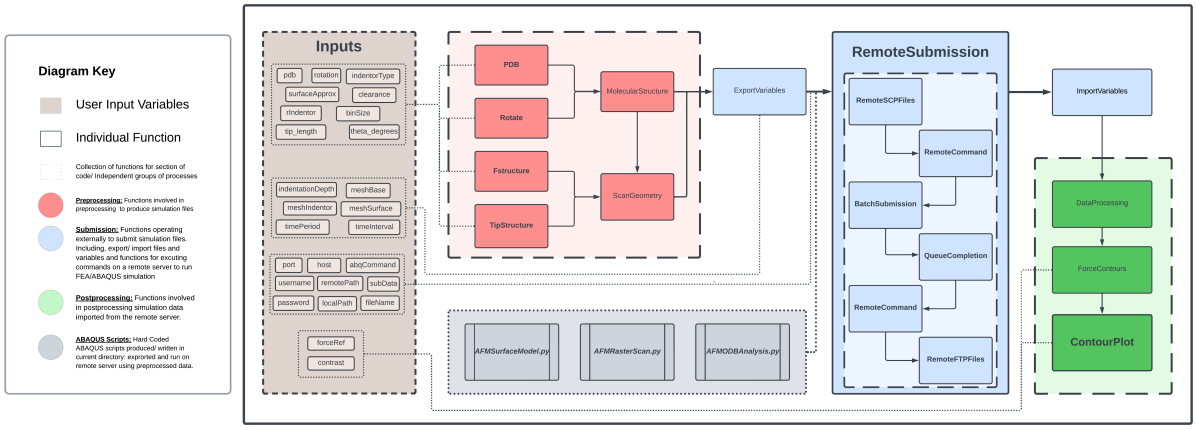
C.4 Main AFM ABAQUS Simulation Code

Introduction

Authors: J. Giblin-Burnham

To run the code for general use simply click the play button in the small box next to hidden cells in the 2. Code Framework section. This will set up all the functions so that you can easily produce AFM image simulations using the GUI in section 3. The GUI gives you the option to change all the variables in the final simulation function. Once you have selected your desired variables, hit shift+enter on your keyboard to run! To debug the functions or cells, simply click the drop down arrows next to subsections. To access the code behind the GUI, or hide it, double click anywhere on the interface.

Flow Chart



Code Framework

Imports

In [ ]:

```
# ----------------------------------------------------System Imports-----------------------------------------------------
import os
import sys
import time
import subprocess
from datetime import timedelta
from platform import python_version
print(python_version())

# ------------------------------------ Possible modules to pip instal-------------------------------------------------
# !{sys.executable} -m pip install py3Dmol
# !{sys.executable} -m pip install nglview
# !{sys.executable} -m pip install biopython
# !{sys.executable} -m pip install line_profiler
# !{sys.executable} -m pip install mendeleev
# !{sys.executable} -m pip install tensorflow
# !{sys.executable} -m pip install pip install pyabaqus==2022
# !{sys.executable} -m pip install scp
# !{sys.executable} -m pip install paramiko
# !{sys.executable} -m pip install scipy
# !{sys.executable} -m pip install matplotlib
# !{sys.executable} -m pip install numba

# ----------------------------------------------Server commands-----------------------------------------------------
import paramiko
from scp import SCPClient

# ----------------------------------------------Mathematical Imports-----------------------------------------------------
# Importing relevant maths and graphing modules
import numpy as np
import pandas as pd
import math
from numpy import random
from random import randrange

import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.mplot3d import axes3d
from matplotlib.ticker import MaxNLocator

linewidth = 5.92765763889 # inch

plt.rcParams["figure.figsize"] = (1.61*linewidth, linewidth)
plt.rcParams['figure.dpi'] = 256
plt.rcParams['font.size'] = 16
plt.rcParams["font.family"] = "Times New Roman"

plt.rcParams['mathtext.fontset'] = 'custom'
plt.rcParams['mathtext.rm'] = 'Times New Roman'
plt.rcParams['mathtext.it'] = 'Times New Roman:italic'
plt.rcParams['mathtext.bf'] = 'Times New Roman:bold'

# For displaying images in Markdown
from IPython.display import Image

# ----------------------------------------------Specific Imports-----------------------------------------------------
# PDB stuff:From video: youtube.com/watch?v=mL8NPpRxgJA&ab_channel=CarlosG.Oliver
from Bio.PDB import *
from Bio.PDB.PDBParser import PDBParser
from Bio.PDB import Entity

pdbl = PDBList()
```

```
parser = MMCIFParser()
pdb_parser = PDBParser(PERMISSIVE=1)

# Atomic properties amd molecule visualistion
from mendeleev import element
import nglview as nv
import py3Dmol

# ------------------------------------------------Optimization modules-------------------------------------------------
import numba
from numba import prange
```

ABAQUS Scripts

ABAQUS is run automatically using seperate python scripts defined below. These are transfered to remote server and run using bash commands to produce input files before running simulations.

Biological Surface Creation Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to create parts for simulation

In [ ]:
```
%%writefile AFMSurfaceModel.py
# ------------------------------------------------Load Modules-------------------------------------------------------
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from part import *
from section import *
from assembly import *
import odbAccess
import cProfile, pstats, io

# ------------------------------------------------Set variables------------------------------------------------------
atom_coord   = np.loadtxt('atom_coords.csv', delimiter=",")
atom_element = np.loadtxt("atom_elements.csv", dtype = 'str', delimiter=",")

keys     = np.loadtxt('atom_radius_keys.csv', dtype = 'str', delimiter=",")
values   = np.loadtxt('atom_radius_values.csv', delimiter=",")
atom_radius = {keys[i]:values[i] for i in range(len(keys))}

baseDims  = np.loadtxt('baseDims.csv', delimiter=",")
tipDims   = np.loadtxt('tipDims.csv', delimiter=",")
rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

with open('indentorType.txt', 'r') as f:
    indentorType = f.read()

#  ------------------------------------------------Model--------------------------------------------------------------
modelName = 'AFMSurfaceModel'
model = mdb.Model(name=modelName)

# ------------------------------------------------Create Atom Parts--------------------------------------------------
for atom in list(atom_radius.keys()):
    # Cretae base atoms (as spheres with vdw radius) to form the biomolecule
    r = atom_radius[atom]
    sketch = model.ConstrainedSketch(name = atom, sheetSize=1.0)
    model.sketches[atom].ConstructionLine(point1=(0,r),point2=(0,-r))
    model.sketches[atom].ArcByCenterEnds(center=(0,0),point1=(0,r),point2=(0,-r), direction = CLOCKWISE)
    model.sketches[atom].Line(point1=(0,r),point2=(0,-r))
    part = model.Part(name=atom, dimensionality=THREE_D, type=DEFORMABLE_BODY)
    model.parts[atom].BaseSolidRevolve( angle=360.0, flipRevolveDirection=OFF, sketch= model.sketches[atom] )


# ------------------------------------------------Create Molecule Assembly-------------------------------------------
# Centre coordinate system
model.rootAssembly.DatumCsysByDefault(CARTESIAN)

# Cretae biomolecule by looping through atom list and merge part instances of the individual atoms
for i, coord in enumerate(atom_coord):
    atom = atom_element[i]
    # For atoms part of molecule above the base
    if coord[2] >= -atom_radius[atom]:
        # Create new part instance for each atom in molecule and translate to position from coordinate list
        model.rootAssembly.Instance(name='instance'+str(i), part = model.parts[atom], dependent=ON)
        model.rootAssembly.translate(instanceList = ('instance'+str(i),) , vector = (coord[0],coord[1],coord[2]) )

# Merge the atomic part instances to make molecule part and part instance
Instances_List = list(model.rootAssembly.instances.keys())
model.rootAssembly.InstanceFromBooleanMerge(name='molecule',
                                            instances=([model.rootAssembly.instances[Instances_List[i]]
                                                        for i in range(len(Instances_List))] ),
                                            originalInstances=DELETE)
# Delete individual atoms
for atom in list(atom_radius.keys()):
    del model.parts[atom]

# ------------------------------------------Create Base/Substrate Part-----------------------------------------------
# Create base part using predefined base dimensions in baseDims, add width/height to accomidate radius of indenter
model.ConstrainedSketch(name = 'base', sheetSize=1.0)
model.sketches['base'].rectangle(point1=(-baseDims[0]/2-rIndentor,-baseDims[1]/2-rIndentor),
                                 point2=( baseDims[0]/2+rIndentor, baseDims[1]/2+rIndentor) )
model.Part(name='base', dimensionality=THREE_D, type= DEFORMABLE_BODY)
model.parts['base'].BaseSolidExtrude(sketch= model.sketches['base'], depth = baseDims[2])

# Create as base part instance
model.rootAssembly.Instance(name='base', part = model.parts['base'], dependent=ON)
model.rootAssembly.translate(instanceList = ('base',) , vector = (0,0,-baseDims[2]) )

# ------------------------------------------------Create Surface Part------------------------------------------------
# Create biomolecule surface by cut any of the molecule intersecting the base
model.rootAssembly.Instance(name='molecule', part = model.parts['molecule'], dependent=ON)
model.rootAssembly.InstanceFromBooleanCut(name= 'surface', instanceToBeCut=model.rootAssembly.instances['molecule'],
                                          cuttingInstances=(model.rootAssembly.instances['base'],),
                                          originalInstances=DELETE)
# Delete unclipped molecule part
del model.parts['molecule']

# ------------------------------------------------Create Tip Part----------------------------------------------------
# If set, create Capped-Conical Indentor using predefined dimensions in tipDims. Using rigid/ incompressible shell part
if indentorType == 'Capped':
```

```
    sketch = model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,-rIndentor),point2=(0,z_top))
    model.sketches['indentor'].Line(point1=(r_int,z_int), point2=(r_top,z_top))
    model.sketches['indentor'].Line(point1=(0,-rIndentor), point2=(0,z_top))
    model.sketches['indentor'].Line(point1=(0,z_top), point2=(r_top,z_top))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0), point1=(r_int,z_int), point2=(0,-rIndentor),
                                        direction =CLOCKWISE)
    model.Part(name='indentor', dimensionality=THREE_D, type= DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseShellRevolve(angle=360.0, flipRevolveDirection=OFF, sketch=model.sketches['indentor'])

# Otherwise create Spherical Indentor part using rIndentor only
else:
    model.ConstrainedSketch(name = 'indentor', sheetSize=1.0)
    model.sketches['indentor'].ConstructionLine(point1=(0,rIndentor),point2=(0,-rIndentor))
    model.sketches['indentor'].ArcByCenterEnds(center=(0,0),point1=(0,rIndentor),point2=(0,-rIndentor),
                                        direction = CLOCKWISE)
    model.sketches['indentor'].Line(point1=(0,rIndentor),point2=(0,-rIndentor))
    model.Part(name='indentor', dimensionality=THREE_D, type=DISCRETE_RIGID_SURFACE)
    model.parts['indentor'].BaseShellRevolve( angle=360.0, flipRevolveDirection=OFF, sketch=model.sketches['indentor'])


# ----------------------------------------------Export Part Files-----------------------------------------------------
model.parts['surface'].writeAcisFile( fileName = 'surface' )
model.parts['base'].writeAcisFile( fileName = 'base' )
model.parts['indentor'].writeAcisFile( fileName = 'indentor' )

# Save Model
mdb.saveAs(modelName +'.cae')
```

Raster Scan AFM Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to create ABAQUS input files at each position in the raster scan. These are used to performm the independent analysis/ simulation at each position.

In [ ]:

```
%%writefile AFMRasterScan.py
# ----------------------------------------------Load Modules-----------------------------------------------------
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
import subprocess
import os
import __main__
executeOnCaeStartup()


# ----------------------------------------------Set variables-----------------------------------------------------
# Import predefined variables from files set in current directory
variables = np.loadtxt('variables.csv', delimiter=",")

baseDims  = np.loadtxt('baseDims.csv', delimiter=",")
tipDims   = np.loadtxt('tipDims.csv', delimiter=",")

# scanPos    = np.loadtxt('scanPos.csv', delimiter=",")
clipped_scanPos  = np.loadtxt('clipped_scanPos.csv', delimiter=",")

timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor, indentionDepth, surfaceHeight = variables
rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

with open('indentorType.txt', 'r') as f:
    indentorType = f.read()

#  ----------------------------------------------Set Model-----------------------------------------------------
modelName =  'AFMRasterScan'

# Open surface model with parts and rename to simulation model
# Linux nomeclature - for Windows use '\\AFMSurfaceModel.cae'
mdb.openAuxMdb( pathName = os.getcwd() +'/AFMSurfaceModel.cae')
mdb.copyAuxMdbModel(fromName='AFMSurfaceModel', toName = modelName)
mdb.closeAuxMdb()

# Set model variable for briefity
model = mdb.models[modelName]


# ----------------------------------------------Set Parts-----------------------------------------------------
# Alternatively, model can be set and indiviadual parts can be imported however shapes detail can be lost in import
# model = mdb.Model(name=modelName)

# # Create Surface part
# surface = mdb.openAcis(fileName = 'surface' )
# model.PartFromGeometryFile(name = 'surface', geometryFile = surface , dimensionality =THREE_D,
#                            type = DEFORMABLE_BODY)

# # Create Base part
# base = mdb.openAcis(fileName = 'base' )
# model.PartFromGeometryFile(name= 'base', geometryFile = base, dimensionality = THREE_D,
#                            type = DEFORMABLE_BODY)

# # Create Indentor part
# indentor = mdb.openAcis(fileName = 'indentor' )
# model.PartFromGeometryFile(name= 'indentor', geometryFile = indentor, dimensionality = THREE_D,
#                            type = DISCRETE_RIGID_SURFACE)


# ----------------------------------------------Set Assembly-----------------------------------------------------
model.rootAssembly.DatumCsysByDefault(CARTESIAN)

# Regenerate part instances and orientate base and indentor
model.rootAssembly.Instance(name='surface', part = model.parts['surface'],dependent=ON)
model.rootAssembly.Instance(name='indentor', part = model.parts['indentor'], dependent=ON)
```

```
model.rootAssembly.Instance(name='base', part = model.parts['base'], dependent=ON)

model.rootAssembly.rotate(instanceList = ('indentor',), axisPoint = (0,0,0), axisDirection = (1,0,0), angle = 90)
model.rootAssembly.translate(instanceList = ('base',) , vector = (0,0,-baseDims[2]) )

# Delete duplicate of part instance
del model.rootAssembly.instances['molecule-1']
del model.rootAssembly.instances['surface-1']


# ----------------------------------------------Set Geometry-----------------------------------------------------
# Create geometric sets and surfaces for each parts faces and cells - these sets are used to reference in model set up

# Surface sets and gemoetric surface for contact
model.parts['surface'].Set(cells= model.parts['surface'].cells.getSequenceFromMask(mask=('[#1fff]', ), ),
                        name='surface_cells')

model.parts['surface'].Set( name='surface_base', faces=
                           model.parts['surface'].faces.getByBoundingBox(-baseDims[0]/2,-baseDims[1]/2,-0.5,
                                                                         baseDims[0]/2,baseDims[1]/2,0.5))
model.parts['surface'].Surface(name='surface_bottom', side1Faces =
                               model.parts['surface'].faces.getByBoundingBox(-baseDims[0]/2,-baseDims[1]/2,-0.5,
                                                                             baseDims[0]/2,baseDims[1]/2,0.5))
model.parts['surface'].Surface(name='surface_top', side1Faces =
                               model.parts['surface'].faces.getByBoundingBox(-baseDims[0]/2,-baseDims[1]/2,0,
                                                                             baseDims[0]/2,baseDims[1]/2, surfaceHeight))

# Base sets and surfaces
model.parts['base'].Set(faces= model.parts['base'].faces.getSequenceFromMask(mask=('[#20]', ),),name='base_faces')
model.parts['base'].Set(cells= model.parts['base'].cells.getSequenceFromMask(mask=('[#1]', ),), name='base_cells')
model.parts['base'].Surface(name='base_surface',
                            side1Faces = model.parts['base'].faces.getSequenceFromMask(mask=('[#10]', ), ) )

# Indentor sets and surfaces
if indentorType == 'Capped':
    # For Spherically Capped geometry
    model.parts['indentor'].Set(faces= model.parts['indentor'].faces.getSequenceFromMask(mask=('[#7]', ), ),
                                name='indentor_faces')
    model.parts['indentor'].Surface(name='indentor_surface',
                                    side1Faces = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#7]', ), ))
else:
    # For Spherical geometry
    model.parts['indentor'].Set(faces= model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1]', ), ),
                                name='indentor_faces')
    model.parts['indentor'].Surface(name='indentor_surface',
                                    side1Faces = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1]', ), ))
# Create reference points for indentor
point = model.parts['indentor'].ReferencePoint((0, 0, 0))
model.parts['indentor'].Set(referencePoints = (model.parts['indentor'].referencePoints[point.id],),
                            name = 'indentor_centre')

model.rootAssembly.regenerate()


# ----------------------------------------------Set Properties---------------------------------------------------
# Assign materials, using elastic and visoelastic properties
elastic = ((1000, 0.3), )
viscoelastic = ((0.0403,0,0.649),(0.0458,0,1.695),)

# Assign molecule surface material
model.Material(name='surface_material')
model.materials['surface_material'].Elastic(table = elastic)
# model.materials['surface_material'].Viscoelastic(domain = FREQUENCY, frequency = PRONY, table = viscoelastic )
model.HomogeneousSolidSection(name='section', material='surface_material', thickness=None)
model.parts['surface'].SectionAssignment(region=model.parts['surface'].sets['surface_cells'],sectionName='section')

# Assign base/substrate large (incompressible) material
model.Material(name='base_material')
model.materials['base_material'].Elastic(table = ((1e15,0.4),))
model.HomogeneousSolidSection(name='base_section', material='base_material', thickness=None)
model.parts['base'].SectionAssignment(region = model.parts['base'].sets['base_cells'], sectionName='base_section')


# ----------------------------------------------Set Steps--------------------------------------------------------
model.StaticStep(name='Indentation', previous='Initial', description='', timePeriod=timePeriod,
                 timeIncrementationMethod=AUTOMATIC, maxNumInc=int(1e5), initialInc=0.1, minInc=1e-20, maxInc=1)

model.steps['Indentation'].control.setValues(allowPropagation=OFF, resetDefaultValues=OFF,
                              timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 25.0, 6.0, 3.0, 50.0))

field = model.FieldOutputRequest('F-Output-1', createStepName='Indentation', variables=('RF', 'TF', 'U'),
                                 timeInterval = timeInterval)


# ----------------------------------------------Set Interactions-------------------------------------------------
# Set  Contact Behaviour
model.ContactProperty(name ='Contact Properties')
model.interactionProperties['Contact Properties'].TangentialBehavior(formulation = FRICTIONLESS) #PENALTY FRICTION
model.interactionProperties['Contact Properties'].NormalBehavior(pressureOverclosure=HARD)

# Set Ridged Indentor
model.RigidBody(name = 'indentor_constraint',
                bodyRegion = model.rootAssembly.instances['indentor'].sets['indentor_faces'],
                refPointRegion = model.rootAssembly.instances['indentor'].sets['indentor_centre'])

# Define surface-surface contact for each body
model.SurfaceToSurfaceContactStd(name = 'surface-indentor',
                                 createStepName = 'Initial',
                                 master     = model.rootAssembly.instances['indentor'].surfaces['indentor_surface'],
                                 slave = model.rootAssembly.instances['surface'].surfaces['surface_top'],
                                 interactionProperty = 'Contact Properties',
                                 sliding = FINITE)

model.SurfaceToSurfaceContactStd(name  = 'base-surface',
                                 createStepName = 'Initial',
                                 master      = model.rootAssembly.instances['base'].surfaces['base_surface'],
                                 slave  = model.rootAssembly.instances['surface'].surfaces['surface_bottom'],
                                 interactionProperty = 'Contact Properties',
                                 sliding = FINITE)

model.SurfaceToSurfaceContactStd(name = 'base-indentor',
                                 createStepName = 'Initial',
                                 master     = model.rootAssembly.instances['indentor'].surfaces['indentor_surface'],
                                 slave = model.rootAssembly.instances['base'].surfaces['base_surface'],
                                 interactionProperty = 'Contact Properties',
                                 sliding = FINITE)
```

```
# ----------------------------------------------Set Loads----------------------------------------------------------
# Create base boundary conditions
model.DisplacementBC(name = 'Base-BC', createStepName = 'Initial',
                     region = model.rootAssembly.instances['base'].sets['base_faces'],
                     u1 = SET, u2 = SET, u3 = SET, ur1 = SET, ur2 = SET, ur3 = SET)

# Create surface boundary conditions
model.DisplacementBC(name = 'Surface-BC', createStepName = 'Initial',
                     region = model.rootAssembly.instances['surface'].sets['surface_base'],
                     u1 = SET, u2 = SET, u3 = SET, ur1 = SET, ur2 = SET, ur3 = SET)

# Create indentor boundary conditions
model.DisplacementBC(name = 'Indentor-UC', createStepName = 'Indentation',
                     region = model.rootAssembly.instances['indentor'].sets['indentor_centre'],
                     u1 = SET, u2 = SET, u3 = -indentionDepth,
                     ur1 = SET, ur2 = SET, ur3 = SET)


# ----------------------------------------------Set Mesh----------------------------------------------------------
# Assign an element type to the part instance- seed and generate
model.rootAssembly.regenerate()

# Assign surface mesh using tetrahedral elements
elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D15, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D10, elemLibrary=STANDARD)
cells     =  model.parts['surface'].cells.getSequenceFromMask(mask=('[#1]', ), )

model.parts['surface'].seedPart(size=meshSurface, deviationFactor=0.01, minSizeFactor=0.9)
model.parts['surface'].setMeshControls(regions=cells, elemShape=TET, sizeGrowthRate=1.64, technique=FREE)
model.parts['surface'].setElementType(regions=(cells,), elemTypes=(elemType1, elemType2, elemType3))
model.parts['surface'].generateMesh()


# Assign indentor mesh using triangular elements
elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=STANDARD)
faces     = model.parts['indentor'].faces.getSequenceFromMask(mask=('[#1 ]', ),)

model.parts['indentor'].seedPart(size=meshIndentor, minSizeFactor=0.25)
model.parts['indentor'].setMeshControls(regions=faces, elemShape=TRI)
model.parts['indentor'].setElementType(regions=(faces,), elemTypes=(elemType1, elemType2))
model.parts['indentor'].generateMesh()


# Assign base mesh using tetrahedral elements
model.parts['base'].seedPart(size = meshBase )
model.parts['base'].setElementType(model.rootAssembly.instances['base'].sets['base_faces'],
                                   elemTypes = (mesh.ElemType(elemCode=QUAD, elemLibrary=STANDARD),))
model.parts['base'].setMeshControls(regions=model.rootAssembly.instances['base'].sets['base_faces'].vertices,
                                    elemShape=TET,technique=FREE)
model.parts['base'].generateMesh()


## ----------------------------------------------Set Submission----------------------------------------------------------
for i in range(len(clipped_scanPos)):
    # Translate indentor to raster scan position
    model.rootAssembly.translate(instanceList = ('indentor',),
                                 vector = (clipped_scanPos[i,0], clipped_scanPos[i,1], clipped_scanPos[i,2] + rIndentor ))
    model.rootAssembly.regenerate()

    # Create input file for simulation position
    jobName   = 'AFMRasterScan-Pos'+str(int(i))
    job = mdb.Job(name=jobName, model=modelName, description='AFM')
    job.writeInput()

    # Reset indentor position to centre
    model.rootAssembly.translate(instanceList = ('indentor',),
                                 vector = (-clipped_scanPos[i,0], -clipped_scanPos[i,1], -(clipped_scanPos[i,2] + rIndentor) ))

mdb.saveAs('AFMRasterScan.cae')
```

ODB Analysis Script

Create Python file in current directory (using magic command %%) used to run on ABAQUS to analysis odb files for each position in the raster scan. Extracting indentation data.

In [ ]:

```
%%writefile AFMODBAnalysis.py
# ----------------------------------------------Load Modules----------------------------------------------------------
import sys
import os
from os.path import exists

from odbAccess import *
from types import IntType
import numpy as np
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import *
from part import *
from material import *
from section import *
from assembly import *
from interaction import *
from mesh import *
from visualization import *
import visualization
import odbAccess
from connectorBehavior import *
import cProfile, pstats, io
import regionToolset
executeOnCaeStartup()

# ----------------------------------------------Set variables----------------------------------------------------------
# Import predefined variables from files set in current directory
variables        = np.loadtxt('variables.csv', delimiter=",")
# scanPos          = np.loadtxt('scanPos.csv', delimiter=",")
clipped_scanPos  = np.loadtxt('clipped_scanPos.csv', delimiter=",")

timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor, indentionDepth, surfaceHeight = variables

# Set array for indentation data: RF - Indentor reaction force , U2 - Indentor displacement, N - no. timesteps
N = int(timePeriod/ timeInterval)+1
RF = np.zeros([len(clipped_scanPos),N])
```

```
U2 = np.zeros([len(clipped_scanPos),N])

# ----------------------------------------------Set Data extraction-----------------------------------------------------
for i in range(len(clipped_scanPos)):
    # Log Analysis Progression in text file
    with open('Progress.txt', 'a') as f:
        f.write('File '+str(i)+'\n')

    jobName = 'AFMRasterScan-Pos'+str(int(i))

    # Check if odb file is veiwable and no corrupted data which my throw an error
    try :
        # Open odb and retrieve data for indentor reference point(2nd value in models nodal sets/ region)
        odb    = openOdb(jobName +'.odb', readOnly=True)
        region = odb.rootAssembly.nodeSets.values()[1]
    except:
        # If theres an error log odb file value in text file
        with open('Errors.txt', 'a') as f:
            f.write('ERROR for'+str(i)+'\n')
    else:
        # Extracting data for Step 1, this analysis only had one step
        step1 = odb.steps.values()[0]

        # Counting frames/ timesteps
        j,k = 0, 0

        # Creating a for loop to iterate through all frames in the step
        for x in odb.steps[step1.name].frames:

            # Reading force and displacement data from the model at reference point
            fieldRF = x.fieldOutputs['RF'].getSubset(region= region)
            fieldU  = x.fieldOutputs['U'].getSubset(region= region)

            # Storing reaction force and displacement values for the current frame
            for rf in fieldRF.values:
                RF[i,j] = np.sqrt(rf.data[2]**2)
                j+=1

            for u in fieldU.values:
                U2[i,k] = u.data[2]
                k+=1

# Writing to a .csv file
np.savetxt("U2_Results.csv", U2 , delimiter=",")
np.savetxt("RF_Results.csv", RF , delimiter=",")

# Close the odb
odb.close()
```

Simulation Script Functions

Functionalised code to automate scan and geometry calculations, remote server access, remote script submission, data anlaysis and postprocessing required to produce AFM image.

Pre-Processing Functions

Functions used in preprocessing step of simulation, including calculating scan positions and exporting variables.

Biomolecule PDB Function

Function to imports the relevant PDB file (and takes care of the directory) in which it is saved etc for the user, returning the structure and the view using a widget.

In [ ]:
```
### Function to get structure from pdb file 4 digit code
def PDB(pdbid):
    '''
    This function imports the relevant PDB file (and takes care of the directory) in which it is saved etc for the user,
    returning the structure and the view using a widget.

        Parameters:
            pdbid (str) - PDB (or CSV) file name of desired biomolecule

        Returns:
            Structure (class) - Class containing proteins structural data (Atom coords/positions and masses etc...)
            View (class)      - Class for visualising the protein
    '''
    # Retrieves PDB file from 4 letter code using Bio.python
    pdbl.retrieve_pdb_file(pdbid)

    ### Creating a folder on the Users system- location is the same as the Notebook file's
    split_pdbid = list(pdbid)
    structure_file_folder = str(split_pdbid[1]) + str(split_pdbid[2])

    # Retrieving file from the location it is saved in
    # Set `the_slashes` as '/' for MAC and Google CoLab or '//' for Windows
    cwd = str(os.getcwd())
    file_loc = cwd + '\\' + structure_file_folder + '\\' + pdbid + '.cif'

    # Defining structure i.e. '4 letter PDB ID code' and 'location'
    structure = parser.get_structure(pdbid, file_loc)

    # Plotting relevant structure using py3Dmol
    viewer_name = 'pdb:' + pdbid
    view = py3Dmol.view(query=viewer_name).setStyle({'cartoon':{'color':'spectrum'}})

    return(structure,view)
```

Tip Functions

Functions to produce list of tip structural parameters, alongside function to calculates and returns tip surface heights from radial position r.

In [ ]:
```
def TipStructure(rIndentor, theta_degrees, tip_length):
    '''
    Produce list of tip structural parameters. Change principle angle to radian. Calculate tangent point where
    sphere smoothly transitions to cone for capped conical indentor.

        Parameters:
            theta_degrees (float) - Principle conical angle from z axis in degrees
            rIndentor (float)     - Radius of spherical tip portion
            tip_length (float)    - Total cone height

        Returns:
            tipDims (list) - Geometric parameters for defining capped tip structure
```

```
    '''
    theta = theta_degrees*(np.pi/180)

    # Intercept of spherical and conical section of indentor (Tangent point)
    r_int, z_int = rIndentor*abs(np.cos(theta)), -rIndentor*abs(np.sin(theta))
    # Total radius/ footprint of indentor/ top coordinates
    r_top, z_top = (r_int+(tip_length-r_int)*abs(np.tan(theta))), tip_length-rIndentor

    return [rIndentor, theta, tip_length, r_int, z_int, r_top, z_top]
```

In [ ]:
```
def Zconical(r, r0, r_int, z_int, theta, R, tip_length):
    '''
    Calculates and returns spherically capped conical tip surface heights from radial  position r. Uses radial coordinate along
    xy plane from centre as tip is axisymmetric around z axis (bottom of tip set as zero point such z0 = R).

        Parameters:
            r (float/1D arr)   - xy radial coordinate location for tip height to be found
            r0 (float)         - xy radial coordinate for centre of tip
            r_int (float)      - xy radial coordinate of tangent point (point where sphere smoothly transitions to cone)
            z_int (float)      - Height of tangent point, where sphere smoothly transitions to cone (defined for tip centred at spheres
                                 center, as calculations assume tip centred at indentors bottom the value must be corrected to, R-z_int)
            theta (float)      - Principle conical angle from z axis in radians
            R (float)          - Radius of spherical tip portion
            tip_length (float) - Total cone height

        Returns:
            Z (float/1D arr)- Height of tip at xy radial coordinate
    '''

    ### Constructing conical and spherical parts boundaries of tip using arrays for computation speed
    # ----------------------------------------------------Spherical Boundary-----------------------------------------------
    # For r <= r_int, z <= z_int : (z-z0)^2 +  (r-r0)^2 = R^2 --> z = z0  + ( R^2 - (r-r0)^2 )^1/2

    # Using equation of sphere compute height (points outside sphere radius are complex and return nan,
    # nan_to_num is used to set these points to max value R). The heights are clip to height of tangent point, R-z_int.
    # Producing spherical portion for r below tangent point r_int and constant height R-zint for r values above r_int.

    z1 = np.clip( np.nan_to_num(R - np.sqrt(R**2 - (r-r0)**2), copy=False, nan=R ), a_min = 0 , a_max = R-abs(z_int))
    # z1 = np.clip( np.where( np.isnan( R - np.sqrt(R**2 - (r-r0)**2) ) , R, R - np.sqrt(R**2 - (r-r0)**2) ), a_min = 0 , a_max = R-np.abs(z_int))

    # ----------------------------------------------------Conical Boundary-------------------------------------------------
    # r > r_int, z > z_int : z = m*abs(x-x0);  where x = r, x0 = r0 + r_int,  m = 1/tan(theta)

    # Using equation of cone (line) to compute height for r values larger than tangent point r_int (using where condition)
    # For r values below r_int the height is set to zero

    z2 =np.where(abs(r-r0)>=r_int, (abs(r-r0)-r_int)/abs(np.tan(theta)), 0)

    # ----------------------------------------------------Combing Boundaries-----------------------------------------------
    # For r values less than r_int, combines spherical portion with zero values from conical, producing spherical section
    # For r values more than r_int, combines linear conical portion with R-z_int values from spherical, producing cone section
    Z = z1 + z2

    # Optional mask values greater than tip length
    # Z = np.ma.masked_greater(z1+z2, tip_length )
    return Z
```

In [ ]:
```
# @numba.njit
def Zspherical(r, r0, r_int, z_int, theta, R, tip_length):
    '''
    Calculates and returns spherical tip surface heights from radial  position r. Uses radial coordinate along xy plane from
    centre as tip is axisymmetric around z axis (bottom of tip set as zero point such z0 = R).

        Parameters:
            r (float/1D arr)   - xy radial coordinate location for tip height to be found
            r0 (float)         - xy radial coordinate for centre of tip
            r_int (float)      - xy radial coordinate for tangent point (point where sphere smoothly transitions to cone)
            z_int (float)      - Height of tangent point (point where sphere smoothly transitions to cone)
            theta (float)      - Principle conical angle from z axis in radians
            R (float)          - Radius of spherical tip portion
            tip_length (float) - Total cone height

        Returns:
            Z (float/1D arr)- Height of tip at xy radial coordinate
    '''
    # Simple spherical equation: (z-z0)^2 +  (r-r0)^2 = R^2 --> z = z0  - ( R^2 - (r-r0)^2 )^1/2
    return ( R - np.sqrt(R**2 - (r-r0)**2) )
```

Surface Functions

Functions to orientate biomolecule, extract atomic positions and elements from structural data and calculate bse/substrate dimensions.

In [ ]:
```
def Rotate(domain, rotation):
    '''
    Rotate coordinates of a domain around each coordinate axis by angles given.
        Parameters:
            domain (arr)    - Array of [x,y,z] coordinates in domain to be rotated (Shape: (3) or (N,3) )
            rotation (list) - Array of [xtheta, ytheta, ztheta] rotational angle around coordinate axis:
                              # xtheta(float), angle in degrees for rotation around x axis (Row)
                              # ytheta(float), angle in degrees for rotation around y axis (Pitch)
                              # ztheta(float), angle in degrees for rotation around z axis (Yaw)
        Returns:
             rotate_domain(arr) - Rotated coordinate array
    '''
    xtheta, ytheta, ztheta = (np.pi/180)*np.array(rotation)

    # Row, Pitch, Yaw rotation matrices
    R_x = np.matrix( [[1,0,0],[0,np.cos(xtheta),-np.sin(xtheta)],[0,np.sin(xtheta),np.cos(xtheta)]] )
    R_y = np.matrix( [[np.cos(ytheta),0,np.sin(ytheta)],[0,1,0],[-np.sin(ytheta),0,np.cos(ytheta)]] )
    R_z = np.matrix( [[np.cos(ztheta),-np.sin(ztheta),0],[np.sin(ztheta),np.cos(ztheta),0],[0,0,1]] )

    # Complete rotational matrix, from matrix multiplication
    R = R_x * R_y * R_z

    return np.array((R*np.asmatrix(domain).T).T)
```

In [ ]:
```
def MolecularStructure(structure, rotation, rIndentor, surfaceApprox):
    '''
    Extracts molecular data from structure class and returns array of molecules atomic coordinate and element names. Alongside, producing dictionary
    of element radii and calculating base dimensions. All distances given in Angstroms (x10-10 m).

        Parameters:
            structure (class)     - Class containing proteins structural data (Atom coords/positions and masses etc...)
            rotation (list)       - Array of [x,y,z] rotational angle around coordinate axis'
```

```
            surfaceApprox (float) - Percentage of biomolecule assumed to be not imbedded in base/ substrate. Range: 0-1

        Returns:
            atom_coord (arr)      - Array of coordinates [x,y,z] for atoms in biomolecule
            atom_element (arr)    - Array of elements names(str) for atoms in biomolecule
            atom_radius (dict)    - Dictionary containing van der waals radii each the element in the biomolecule
            surfaceHeight (float) - Maximum height of biomolecule in z direction
            baseDims (list)       - Geometric parameters for defining base/ substrate structure [width, height, depth]
    '''
    #  ------------------------------------Setup Molecule Elements-----------------------------------------------
    # Extract atom element list as array
    atom_list  = structure.get_atoms()
    atom_element = np.array([atom.element[0] + (atom.element[1:].lower() if len(atom.element) > 1 else '') for atom in atom_list])

    # Produce dictionary of element radii in angstrom (using van de waals/dreiding radius)
    atom_radius = dict.fromkeys(np.sort(atom_element))

    for i in list(atom_radius.keys()):
        atom_radius[i] = element(i).vdw_radius_uff/100 #element(i).vdw_radius_dreiding/100 # element(i).vdw_radius_mm3/100

    #  ------------------------------------Setup Molecule Geometry-----------------------------------------------
    # Extract atom coordinates list as array in angstrom
    atom_list  = structure.get_atoms()
    atom_coord = np.array([atom.coord for atom in atom_list])

    # Rotate coordinates of molecule
    atom_coord = Rotate(atom_coord, rotation)

    # Find extent of molecule extent
    surfaceMaxX, surfaceMinX = atom_coord[:,0].max(), atom_coord[:,0].min()
    surfaceMaxY, surfaceMinY = atom_coord[:,1].max(), atom_coord[:,1].min()
    surfaceMaxZ, surfaceMinZ = atom_coord[:,2].max(), atom_coord[:,2].min()

    surfaceWidthX = abs(surfaceMaxX-surfaceMinX)
    surfaceWidthY = abs(surfaceMaxY-surfaceMinY)
    surfaceWidthZ = abs(surfaceMaxZ-surfaceMinZ)

    # Centre molecule geometry in xy and set z=0 at the top of the base with percentage of height not imbedded
    atom_coord[:,0] = atom_coord[:,0] - surfaceMinX - surfaceWidthX/2
    atom_coord[:,1] = atom_coord[:,1] - surfaceMinY - surfaceWidthY/2
    atom_coord[:,2] = atom_coord[:,2] - surfaceMinZ - surfaceWidthZ*surfaceApprox

    #  ------------------------------------Setup Base/Surface Geometry-------------------------------------------
    # Calculate maximum surface height with added clearance. Define substrate/Base dimensions using biomolecules extent in x and y and radius of indentor
    surfaceHeight = 1.5*(atom_coord[:,2].max())
    baseDims      = np.rint([1.5*(surfaceWidthX+2*rIndentor), 1.5*(surfaceWidthY+2*rIndentor), 2*np.max(list(atom_radius.values()))+1])

    return atom_coord, atom_element, atom_radius, surfaceHeight, baseDims
```

Scan Functions

Calculate scan positions of tip over surface and vertical set points above surface for each position. In addition, function to plot and visualise molecules surface and scan position.

In [ ]:
```
def ScanGeometry(atom_coord, atom_radius, atom_element, indentorType, tipDims, baseDims, surfaceHeight, binSize, clearance):
    '''
    Produces array of scan locations and corresponding heights/ tip positions above surface in Angstroms (x10-10 m). Also return an array including
    only positions where tip interact with the sample. The scan positions are produced creating a rectangular grid over bases extent with widths bin size.
    Heightss, at each position, are calculated by set tip above sample and calculating vertical distance between of tip and molecules surface over the indnenters
    area. Subsequently, the minimum vertical distance corresponds to the position where tip is tangential.

        Parameters:
            atom_coord (arr)      - Array of coordinates [x,y,z] for atoms in biomolecule
            atom_radius (dict)    - Dictionary containing van der waals radii each the element in the biomolecule
            atom_element (arr)    - Array of elements names(str) for atoms in biomolecule
            indentorType (str)    - String defining indentor type (Spherical or Capped)
            tipDims (list)        - Geometric parameters for defining capped tip structure
            baseDims (list)       - Geometric parameters for defining base/ substrate structure [width, height, depth]
            surfaceHeight (float) - Maximum height of biomolecule in z direction
            binSize (float)       - Width of bins that subdivid xy domain during raster scanning/ spacing of the positions sampled over
            clearance (float)     - Clearance above molecules surface indentor is set to during scan

        Returns:
            scanPos (arr)         - Array of coordinates [x,y,z] of scan positions to image biomolecule and initial heights/ hard sphere boundary
            clipped_scanPos (arr) - Array of clipped (containing only positions where tip and molecule interact) scan positions and
                                    initial heights [x,y,z] to image biomolecule
    '''
    #  ------------------------------------Set Scan Positions from Scan Geometry-----------------------------------
    # Create rectangular grid of xy scan positions over base using meshgrid.
    x = np.linspace(-baseDims[0]/2, baseDims[0]/2, int(baseDims[0]/binSize)+1)
    y = np.linspace(-baseDims[1]/2, baseDims[1]/2, int(baseDims[1]/binSize)+1)

    # Produce xy scan positions of indentor, set initial z height as clearance
    scanPos = np.array([ [x[i], y[j], clearance] for j in range(len(y)) for i in range(len(x)) ])

    #  ------------------------------------Set Vertical Scan Positions Positions -----------------------------------
    # Extract tip dimensions
    rIndentor, theta, tip_length, r_int, z_int, r_top, z_top = tipDims

   # Set indentor height functions and indentor radial extent/boundry for z scanPos calculation.
    if indentorType == 'Capped':
        # Extent of conical indentor is the radius of the top portion
        rBoundary  =  r_top
        Zstructure = Zconical
    else:
        # Extent of spherical indentor is the radius
        rBoundary  = rIndentor
        Zstructure = Zspherical

    # Array of radial positions along indentor radial extent. Set indentor position/ coordinate origin at surface height
    # (z' = z + surfaceHeight) and calculate  vertical heights along the radial extent.
    r = np.linspace(-rBoundary, rBoundary, 100)
    zIndentor = Zstructure(r, 0, r_int, z_int, theta, rIndentor, tip_length) + surfaceHeight

    # Loop for each scan position
    for i, indentorPos in enumerate(scanPos):
        # Create temporary height array with initial maximum height difference at surface height
        dz = surfaceHeight*np.ones(1)

        # Looping for each atom position in the coordinate array
        for j, atomPos in enumerate(atom_coord):

            # Extract each atoms radius using radius dictionary
            rElement = atom_radius[atom_element[j]]

            # If atoms surface lies on/above the top of the base (z=0)
```

```
            if atomPos[2] >= -rElement:

                # Calculate radial distance from scan position to atom centre
                rInteract  = np.sqrt( (atomPos[0]-indentorPos[0])**2 + (atomPos[1]-indentorPos[1])**2 )

                # If any of the atoms surface lies within indentors boundary calculate vertical distance between the two
                if rInteract - rElement <= rBoundary:
                    # Using equation of sphere compute top heights of atoms surface along indentors radial extent (points outside sphere
                    # radius are complex and return nan, nan_to_num is used to set these points to the min value of bases surface z=0).
                    zElement  = np.nan_to_num((atomPos[2] + np.sqrt( rElement**2 - (r-rInteract)**2)), copy=False, nan=0 )

                    # Appending the difference in the indentor height and the atoms surface at each point along indenoter extent, produces a dz
                    #  array of all the height differences between indentor and surface atoms within the indentors boundary around this position.
                    dz  = np.append(dz, (zIndentor-zElement))
            else:
                None

        # Therefore, z' = -dz  gives an array of indentor positions when each individual part of surface atoms contacts the tip portion above.
        # Translating from z' basis (with origin at z = surfaceHeight) to z basis (with origin at the top of the base) is achieved by
        # perform translation z = z' + surfaceheight. Therefore, these tip position are given by  dz = surfaceheight - dz'. The initial height
        # corresponds to the maximum value of dz/ min value of dz' where the tip is tangential to the surface. I.e. when dz' is minimised
        # all others dz' tip positions will be above/ further from the surface. Therefore, at this position, the rest of the indentor wil
        # not be in contact with the surface and it is tangential.
        scanPos[i,2] = surfaceHeight - abs( dz.min() ) + clearance


    #  ----------------------------------------------Clip Scan position ------------------------------------------------------
    # Include only positions where tip interact with the sample. Scan position equal clearance, corresponds to indentor at base height
    # therfore, can't indent surface (where all dz' heights were greater than surface height )
    clipped_scanPos = np.array([ [ scanPos[i,0], scanPos[i,1], scanPos[i,2] ] for i in range(len(scanPos)) if scanPos[i,2] != clearance ])


    return scanPos, clipped_scanPos
```

In [ ]:

```
def SurfacePlot(atom_coord, atom_radius, atom_element, scanPos, clipped_scanPos):
    '''
    Plot the molecules atoms surfaces and scan positions to visualise and check positions.

        Parameters:
            atom_coord (arr)        - Array of coordinates [x,y,z] for atoms in biomolecule
            atom_radius (dict)      - Dictionary containing van der waals radii each the element in the biomolecule
            atom_element (arr)      - Array of elements names(str) for atoms in biomolecule
            scanPos (arr)           - Array of coordinates [x,y,z] of scan positions to image biomolecule and initial heights/ hard sphere boundary
            clipped_scanPos (arr)   - Array of clipped (containing only positions where tip and molecule interact) scan positions and
                                      initial heights [x,y,z] to image biomolecule
    '''
    # Set range of polar/ azimuthal angles for setting atoms surface positions
    polar     = np.linspace(-np.pi, np.pi, 16)
    azimuthal = np.linspace(0, np.pi, 16)

    # Initialise count variable
    k=-1

    # Create array of all atom surface positions includung embedded
    surfacatomPos = np.zeros([ len(atom_coord)*len(polar)*len(azimuthal), 3 ])
    # For each atom, loop over polar angles and azimuthal angles
    for i, r in enumerate(atom_coord):
        for phi in polar:
            for theta in azimuthal:
                # Count array index
                k+=1

                # Unpack coordinates of atom centre and atom radius
                x0, y0, z0 = r
                R = atom_radius[atom_element[i]]

                # Calculate surface coordinate using spherical coordinates
                surfacatomPos[k,0] = x0 - R*np.cos(phi)*np.sin(theta)
                surfacatomPos[k,1] = y0 - R*np.sin(phi)*np.sin(theta)
                surfacatomPos[k,2] = z0 - R*np.cos(theta)


    # Initialise count variables
    nB=0
    k=-1

    # Create array of all atom surface positions above base
    clipped_surfacatomPos = np.zeros([ len(atom_coord)*len(polar)*len(azimuthal), 3 ])
    for i, r in enumerate(atom_coord):

        #  Set atom radius for atoms with surface above base
        R = atom_radius[atom_element[i]]
        if r[2] >= -R:
            # Count atom
            nB+=1
            # For each atom, loop over polar angles and azimuthal angles
            for phi in polar:
                for theta in azimuthal:
                    # Count array index
                    k+=1

                    # Unpack coordinates of atom centre
                    x0, y0, z0 = r

                    # Calculate surface coordinate using spherical coordinates
                    clipped_surfacatomPos[k,0] = x0 - R*np.cos(phi)*np.sin(theta)
                    clipped_surfacatomPos[k,1] = y0 - R*np.sin(phi)*np.sin(theta)
                    clipped_surfacatomPos[k,2] = z0 - R*np.cos(theta)

    # Return number of atoms and scan positions
    print('Number of Atoms in Molecuel:', nB)

    # Plot Surface incuding imbedded portin and all scan positions
    fig1 = plt.figure()
    ax1 = plt.axes(projection='3d')
    ax1.scatter3D(surfacatomPos[:,0], surfacatomPos[:,1], surfacatomPos[:,2], label = 'All Atom Surfaces')
    ax1.scatter3D(scanPos[:,0], scanPos[:,1], scanPos[:,2], label = 'All Scan Positons')
    ax1.set_xlabel(r'x (${\AA}$)')
    ax1.set_ylabel(r'y (${\AA}$)')
    ax1.set_zlabel(r'z (${\AA}$)')
    fig1.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\AFMSimulationScanPos-'+pdb+'1.png', bbox_inches = 'tight')
    plt.show()

    # Plot clipped surface and clipped scan positions
    fig2 = plt.figure()
    ax2 = plt.axes(projection='3d')
```

```
    ax2.scatter3D(clipped_surfacatomPos[::8,0], clipped_surfacatomPos[::8,1], clipped_surfacatomPos[::8,2],  label = 'Clipped Atom Surfaces')
    ax2.scatter3D(clipped_scanPos[:,0], clipped_scanPos[:,1], clipped_scanPos[:,2],  label = 'Clipped Scan Positons')
    ax2.set_xlabel(r'x (${\AA}$)')
    ax2.set_ylabel(r'y (${\AA}$)')
    ax2.set_zlabel(r'z (${\AA}$)')
    ax2.view_init(90, 0)
    fig2.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\AFMSimulationScanPos-'+pdb+'2.png', bbox_inches = 'tight')
    plt.show()
```

Submission Functions

File Import/ Export

In [ ]:
```
def ExportVariables(atom_coord, atom_element, atom_radius, clipped_scanPos, scanPos, variables, baseDims, tipDims, indentorType):
    '''
    Export simulation variables as csv and txt files to load in abaqus python scripts.

        Parameters:
            atom_coord (arr)      - Array of coordinates [x,y,z] for atoms in biomolecule
            atom_element (arr)    - Array of elements names(str) for atoms in biomolecule
            atom_radius (dict)    - Dictionary containing van der waals radii each the element in the biomolecule
            clipped_scanPos (arr) - Array of clipped (containing only positions where tip and molecule interact) scan positions and
                                    initial heights [x,y,z] to image biomolecule
            scanPos (arr)         - Array of coordinates [x,y,z] of scan positions to image biomolecule and initial heights/ hard sphere boundary
            variables (list)      - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor,
                                    indentionDepth, surfaceHeight]
            baseDims (list)       - Geometric parameters for defining base/ substrate structure [width, height, depth]
            tipDims (list)        - Geometric parameters for defining capped tip structure
            indentorType (str)    - String defining indentor type (Spherical or Capped)
    '''

    np.savetxt("atom_coords.csv", atom_coord, delimiter=",")
    np.savetxt("atom_elements.csv", atom_element, fmt='%s', delimiter=",")

    np.savetxt("atom_radius_keys.csv", list(atom_radius.keys()), fmt='%s', delimiter=",")
    np.savetxt("atom_radius_values.csv", list(atom_radius.values()), delimiter=",")

    np.savetxt("clipped_scanPos.csv", clipped_scanPos, delimiter=",")
    np.savetxt("scanPos.csv", scanPos, fmt='%s', delimiter=",")

    np.savetxt("variables.csv", variables, fmt='%s', delimiter=",")
    np.savetxt("baseDims.csv", baseDims, fmt='%s', delimiter=",")
    np.savetxt("tipDims.csv", tipDims, fmt='%s', delimiter=",")

    with open('indentorType.txt', 'w', newline = '\n') as f:
        f.write(indentorType)
```

In [ ]:
```
def ImportVariables():
    '''
    Import simulation geometry variables from csv files.

        Return:
            atom_coord (arr)        - Array of coordinates [x,y,z] for atoms in biomolecule
            atom_element (arr)      - Array of elements names(str) for atoms in biomolecule
            atom_radius (dict)      - Dictionary containing van der waals radii each the element in the biomolecule
            variables (list)        - List of simulation variables: [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor,
                                      indentionDepth, surfaceHeight]
            baseDims (list)         - Geometric parameters for defining base/ substrate structure [width, height, depth]
            scanPos (arr)           - Array of coordinates [x,y,z] of scan positions to image biomolecule and initial heights/ hard sphere boundary
            clipped_scanPos (arr)   - Array of clipped (containing only positions where tip and molecule interact) scan positions and
                                      initial heights [x,y,z] to image biomolecule
    '''

    atom_coord      = np.loadtxt('atom_coords.csv', delimiter=",")
    atom_element    = np.loadtxt("atom_elements.csv", dtype = 'str', delimiter=",")

    keys            = np.loadtxt('atom_radius_keys.csv', dtype = 'str', delimiter=",")
    values          = np.loadtxt('atom_radius_values.csv', delimiter=",")
    atom_radius     = {keys[i]:values[i] for i in range(len(keys))}

    variables       = np.loadtxt('variables.csv', delimiter=",")
    baseDims        = np.loadtxt('baseDims.csv', delimiter=",")
    scanPos         = np.loadtxt('scanPos.csv', delimiter=",")
    clipped_scanPos = np.loadtxt('clipped_scanPos.csv', delimiter=",")

    return atom_coord, atom_element, atom_radius, variables, baseDims, scanPos, clipped_scanPos
```

Remote Functions

Functions for working on remote serve, including transfering files, submitting bash commands, submiting bash scripts for batch input files and check queue statis.

File Transfer`

In [ ]:
```
def RemoteSCPFiles(host, port, username, password, files, remotePath):
    '''
    Function to make directory and transfer files to SSH server. A new Channel is opened and the files are transfered.
    The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            files (str/list) - File or list of file to transfer
            remotePath (str) - Path to remote file/directory
    '''
    # SHH to clusters
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    stdin, stdout, stderr = ssh_client.exec_command('mkdir -p ' + remotePath)

    # SCPClient takes a paramiko transport as an argument- Uploading content to remote directory
    scp_client = SCPClient(ssh_client.get_transport())
    scp_client.put(files, recursive=True, remote_path = remotePath)
    scp_client.close()

    ssh_client.close()
```

Bash Command Submission

In [ ]:
```python
def RemoteCommand(host, port, username, password, script, remotePath, command):
    '''
    Function to execute a command/ script submission on the SSH server. A new Channel is opened and the requested command is executed.
    The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            script (str)     - Script to run via bash command
            remotePath (str) - Path to remote file/directory
            command (str)    - Abaqus command to execute and run script
    '''
    # SSH to clusters using paramiko module
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # Execute command
    stdin, stdout, stderr = ssh_client.exec_command('cd ' + remotePath + ' \n '+ command +' '+ script +' & \n')
    lines = stdout.readlines()

    ssh_client.close()

    for line in lines:
        print(line)
```

Batch File Submission

In [ ]:
```python
def BatchSubmission(host, port, username, password, fileName, subData, scanPos, remotePath, **kwargs):
    '''
    Function to create bash script for batch submission of input file, and run them on remote server.
        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)        -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            fileName (str)   - Base File name for abaqus input files
            subData (str)    - Data for submission to serve queue [walltime, memory, cpus]
            scanPos (arr)    - Array of coordinates [x,y] of scan positions to image biomolecule (can be clipped or full)
            remotePath (str) - Path to remote file/directory

            kwargs:
                Submission ('serial'/ 'paralell') - optional define whether single serial script or seperate paralell submission to queue {Default: 'serial'}
    '''
    # For paralell mode create bash script to runs for single scan location, then loop used to submit individual scripts for each location which run in paralell
    if 'Submission' in kwargs and kwargs['Submission'] == 'paralell':
        lines = ['#!/bin/bash -l',
                 '#$ -S /bin/bash',
                 '#$ -l h_rt='+ subData[0],
                 '#$ -l mem=' + subData[1],
                 '#$ -pe mpi ' + subData[2],
                 '#$ -wd /scratch/scratch/zcapjgi/ABAQUS',
                 'module load abaqus/2017 ',
                 'ABAQUS_PARALLELSCRATCH = "/scratch/scratch/zcapjgi/ABAQUS" ',
                 'cd ' + remotePath,
                 'gerun abaqus interactive cpus=$NSLOTS mp_mode=mpi job=$JOB_NAME input=$JOB_NAME.inp scratch=$ABAQUS_PARALLELSCRATCH resultsformat=odb'
                ]

    # Otherwise, create script to run serial analysis consecutively with single submission
    else:
        # Create set of submission comands for each scan locations
        jobs   = ['gerun abaqus interactive cpus=$NSLOTS mp_mode=mpi job='+fileName+str(int(i))+' input='+fileName+str(int(i))+'.inp scratch=$ABAQUS_PARALLELSCRATCH
                  for i in range(len(scanPos))]

        # Produce preamble to used to set up bash script
        lines = ['#!/bin/bash -l',
                    '#$ -S /bin/bash',
                    '#$ -l h_rt='+ subData[0],
                    '#$ -l mem=' + subData[1],
                    '#$ -pe mpi ' + subData[2],
                    '#$ -wd /home/zcapjgi/Scratch/ABAQUS',
                    'module load abaqus/2017 ',
                    'ABAQUS_PARALLELSCRATCH = "/home/zcapjgi/Scratch/ABAQUS" ',
                    'cd ' + remotePath ]
        # Combine to produce total  script
        lines+=jobs

    # Create script file in current directory by writing each line to file
    with open('batchScript.sh', 'w', newline = '\n') as f:
        for line in lines:
            f.write(line)
            f.write('\n')

    # SSH to clusters
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    stdin, stdout, stderr = ssh_client.exec_command('mkdir -p ' + remotePath)

    # SCPClient takes a paramiko transport as an argument- Uploading content to remote directory
    scp_client = SCPClient(ssh_client.get_transport())
    scp_client.put('batchScript.sh', recursive=True, remote_path = remotePath)
    scp_client.close()

    # If paralell mode, submit  individual scripts for individual scan locations
    if 'Submission' in kwargs and kwargs['Submission'] == 'paralell':
        for i in range(len(scanPos)):
            # Job name set as each input file name as -N jobname is used as input variable in script
            jobName = fileName+str(int(i))

            # Command to run individual jobs
            batchCommand = 'cd ' + remotePath + ' \n qsub -N '+ jobName +' batchScript.sh \n'

            # Execute command
            stdin, stdout, stderr = ssh_client.exec_command(batchCommand)
            lines = stdout.readlines()
            print(lines)

    # Otherwise submit single serial scripts
    else:
        # Job name set as current directory name (change / to \\ for windows)
        jobName = remotePath.split('/')[-1]
```

```
        batchCommand = 'cd ' + remotePath + ' \n qsub -N '+ jobName +' batchScript.sh \n'

        # Execute command
        stdin, stdout, stderr = ssh_client.exec_command(batchCommand)
        lines = stdout.readlines()
        print(lines)

    ssh_client.close()
```

Queue Status Function

In [ ]:
```
def QueueCompletion(host, port, username, password):
    '''
    Function to check queue statis and complete when queue is empty.
        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)       -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
    '''
    # Log time
    t0 = time.time()
    complete= False

    while complete == False:
        # SSH to clusters
        ssh_client = paramiko.SSHClient()
        ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh_client.connect(host, port, username, password)

        # Execute command to view the queue
        stdin, stdout, stderr = ssh_client.exec_command('qstat')
        lines = stdout.readlines()

        # Check if queue is empty
        if len(lines)==0:
            print('Complete')
            complete = True
            ssh_client.close()

        # Otherwis close and wait 2 mins before checking again
        else:
            ssh_client.close()
            time.sleep(120)

    # Return total time
    t1 = time.time()
    print(t1-t0)
```

File Retrieval

In [ ]:
```
def RemoteFTPFiles(host, port, username, password, files, remotePath, localPath):
    '''
    Function to transfer files from directory on SSH server to local machine. A new Channel is opened and the files are transfered.
    The function uses FTP file transfer.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)       -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption if passphrase is not given.
            files (str )     - File to transfer
            remotePath (str) - Path to remote file/directory
            localPath (str)  - Path to local file/directory
    '''
    # SSH to cluster
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # FTPClient takes a paramiko transport as an argument- copy content from remote directory
    ftp_client=ssh_client.open_sftp()
    ftp_client.get(remotePath+'/'+files, localPath +'\\'+ files)
    ftp_client.close()
```

Remote Terminal

In [ ]:
```
def Remote_Terminal(host, port, username, password):
    '''
    Function to emulate cluster terminal. Channel is opened and commands given are executed. The command's input
    and output streams are returned as Python file-like objects representing stdin, stdout, and stderr.

        Parameters:
            host (str)       - Hostname of the server to connect to
            port (int)       - Server port to connect to
            username (str)   - username to authenticate as (defaults to the current local username)       -
            password (str)   - password (str) - Used for password authentication; is also used for private key decryption
                               if passphrase is not given.
    '''

    # SHH to cluster
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(host, port, username, password)

    # Create channel to keep connection open
    ssh_channel = ssh_client.get_transport().open_session()
    ssh_channel.get_pty()
    ssh_channel.invoke_shell()

    # While open accept user input commands
    while True:
        command = input('$ ')
        if command == 'exit':
            break

        ssh_channel.send(command + "\n")

        # Return bash output from command
        while True:
            if channel.recv_ready():
                output = ssh_channel.recv(1024)
                print(output)
            else:
```

```
            time.sleep(0.5)
            if not(ssh_channel.recv_ready()):
                break
    # Close cluster connection
    ssh_client.close()
```

Remote/ Local Submission

Function to run simulation and scripts on the remote servers. Files for variables are transfered, ABAQUS scripts are run to create parts and input files. A bash file is created and submitted to run simulation for batch of inputs. Analysis of odb files is performed and data transfered back to local machine. Using keyword arguments invidual parts of simulation previously completed can be skipped.

In [ ]:
```
def LocalSubmission():
    ''' Submit Abaqus scripts locally'''
    !abaqus fetch job=AFMSurfaceModel
    !abaqus cae -noGUI AFMSurfaceModel.py

    !abaqus fetch job=AFMRasterScan
    !abaqus cae -noGUI AFMRasterScan.py

    !abaqus fetch job=AFMODBAnalysis
    !abaqus cae -noGUI AFMODBAnalysis.py
```

In [ ]:
```
def RemoteSubmission(host, port, username, password, remotePath, localPath,  csvfiles, abqfiles, abqCommand, fileName, subData, clipped_scanPos, **kwargs):
    '''
    Function to run simulation and scripts on the remote servers. Files for variables are transfered, ABAQUS scripts are run to create parts and input files.
    A bash file is created and submitted to run simulation for batch of inputs. Analysis of odb files is performed and data transfered back to local machine.
    Using keyword arguments invidual parts of simulation previously completed can be skipped.

        Parameters:
            host (str)              - Hostname of the server to connect to
            port (int)              - Server port to connect to
            username (str)          - Username to authenticate as (defaults to the current local username)        -
            password (str)          - password (str) – Used for password authentication; is also used for private key decryption if passphrase is not given.
            remotePath (str)        - Path to remote file/directory
            localPath (str)         - Path to local file/directory
            csvfiles (list)         - List of csv and txt files to transfer to remote server
            abqfiles (list)         - List of abaqus script files to transfer to remote server
            abqCommand (str)        - Abaqus command to execute and run script
            fileName (str)          - Base File name for abaqus input files
            subData (str)           - Data for submission to serve queue [walltime, memory, cpus]
            clipped_scanPos (arr)   - Array of clipped (containing only positions where tip and molecule interact) scan positions and  initial heights [x,y,z]
                                      to image biomolecule
            kwargs:
                submission ('serial'/ 'paralell') - Type of submission, submit pararlell scripts or single serial script for scan locations {Default: 'serial'}
                Transfer (bool)                   - If false skip file transfer step of simulation {Default: True}
                Part (bool)                       - If false skip part creation step of simulation {Default: True}
                Input (bool)                      - If false skip input file creation step of simulation {Default: True}
                Batch (bool)                      - If false skip batch submission step of simulation {Default: True}
                Queue (bool)                      - If false skip queue completion step of simulation {Default: True}
                Analysis (bool)                   - If false skip odb analysis step of simulation {Default: True}
                Retrieval (bool)                  - If false skip data file retrivial from remote serve {Default: True}
    '''
    #  ----------------------------------------------File Transfer-----------------------------------------------------------
    if 'Transfer' not in kwargs.keys() or kwargs['Transfer'] == True:

        # Transfer scripts and variable files to remote server
        RemoteSCPFiles(host, port, username, password, csvfiles, remotePath)
        RemoteSCPFiles(host, port, username, password, abqfiles, remotePath)

        print('File Transfer Complete')

    #  ----------------------------------------------Input File Creation-----------------------------------------------------
    if 'Part' not in kwargs.keys() or kwargs['Part'] == True:
        t0 = time.time()
        print('Creating Parts ...')

        # Create Molecule and Tip
        script = 'AFMSurfaceModel.py'
        RemoteCommand(host, port, username, password, script, remotePath, abqCommand)

        t1 = time.time()
        print('Part Creation Complete - ' + str(timedelta(seconds=t1-t0)) )

    if 'Input' not in kwargs.keys() or kwargs['Input'] == True:
        t0 = time.time()
        print('Producing Input Files ...')

        # Produce simulation and input files
        script = 'AFMRasterScan.py'
        RemoteCommand(host, port, username, password, script, remotePath, abqCommand)

        t1 = time.time()
        print('Input File Complete - ' + str(timedelta(seconds=t1-t0)) )

    #  ----------------------------------------------Batch File Submission---------------------------------------------------
    if 'Batch' not in kwargs.keys() or kwargs['Batch'] == True:
        t0 = time.time()
        print('Submitting Batch Scripts ...')

        # Submit bash scripts to remote queue to carry out batch abaqus analysis
        BatchSubmission(host, port, username, password, fileName, subData, clipped_scanPos, remotePath, **kwargs)

        t1 = time.time()
        print('Batch Submission Complete - '+ str(timedelta(seconds=t1-t0)) )

    if 'Queue' not in kwargs.keys() or kwargs['Queue'] == True:
        t0 = time.time()
        print('Simulations Processing ...')

        # Wait for completion when queue is empty
        QueueCompletion(host, port, username, password)

        t1 = time.time()
        print('ABAQUS Simulation Complete - '+ str(timedelta(seconds=t1-t0)) )

    #  ----------------------------------------------ODB Analysis Submission-------------------------------------------------
    if 'Analysis' not in kwargs.keys() or kwargs['Analysis'] == True:
        t0 = time.time()
        print('Running ODB Analysis...')

        # ODB analysis script to run, extracts data from simulation and sets it in csv file on server
        script = 'AFMODBAnalysis.py'
        RemoteCommand(host, port, username, password, script, remotePath, abqCommand)
```

```
        t1 = time.time()
        print('ODB Analysis Complete - ' + str(timedelta(seconds=t1-t0)) )

    #  ----------------------------------------------File Retrieval----------------------------------------------------------------
    if 'Retrieval' not in kwargs.keys() or kwargs['Retrieval'] == True:
        t0 = time.time()
        # Retrieve variables used for given simulation (in case variables redefined when skip kwargs used)
        csvfiles = ("clipped_scanPos.csv", "scanPos.csv","variables.csv","baseDims.csv", "tipDims.csv")
        dataFiles = ('U2_Results.csv','RF_Results.csv')

        # Files retrievals from remote server
        for file in csvfiles:
            RemoteFTPFiles(host, port, username, password, file, remotePath, localPath)
        RemoteFTPFiles(host, port, username, password, dataFiles[0], remotePath, localPath)
        RemoteFTPFiles(host, port, username, password, dataFiles[1], remotePath, localPath)

        t1 = time.time()
        print('File Retrevial Complete')
```

Post-Processing Functions

Function for postprocessing ABAQUS simulation data, loading variables from files in current directory and process data from simulation in U2/RF files. Process data from clipped scan positions to include full data range over all scan positions. Alongside, function to plot and visualise data. Then, calculates contours/z heights of constant force in simulation data for given threshold force and visualise.

Data Processing

Function to load variables from fil~es in current directory and process data from simulation in U2/RF files. Process data from clipped scanpositions to include full data range over all scan positions. Alongside, function to plot and visualise data.

In [ ]:
```
def DataProcessing(clipped_RF, clipped_U2, scanPos, clipped_scanPos, indentionDepth, timePeriod, timeInterval):
    '''
    Function to load variables from files in current directory and process data from simulation in U2/RF files. Process data from clipped scan positions
    to include full data range over all scan positions.
        Parameters:
            clipped_RF             - Array of indentors z displacement over clipped scan position
            clipped_U2             - Array of reaction force on indentor reference point over clipped scan positions
            scanPos (arr)          - Array of coordinates [x,y,z] of scan positions to image biomolecule and initial heights/ hard sphere boundary
            clipped_scanPos (arr)  - Array of clipped (containing only positions where tip and molecule interact) scan positions and
                                     initial heights [x,y,z] to image biomolecule
            indentionDepth (float) - Maximum indentation depth into surface
            timePeriod(float)      - Total time length for ABAQUS simulation/ time step (T)
            timeInterval(float)    - Time steps data sampled over for ABAQUS simulation/ time step (dt)

        Return:
            U2 (arr)      - Array of indentors z displacement over scan position
            RF (arr)      - Array of reaction force on indentor reference point
            N (int)       - Number of frames in ABAQUS simulation/ time step
    '''

    # Set number if frames in ABAQUS simulation step -  N = T/dt + 1 for intial frame
    N = int(timePeriod/ timeInterval)+1

    # Initialise reaction force RF and z indentation depth U2
    RF = np.zeros([len(scanPos),N])
    U2 = np.zeros([len(scanPos),N])

    # Loop over scan positions and clipped scanPos positions
    for i in range(len(scanPos)):
        for j in range(len(clipped_scanPos)):

            # If scan position is in clipped set points extract corresponding simulation for position
            if scanPos[i,0]==clipped_scanPos[j,0] and  scanPos[i,1]==clipped_scanPos[j,1] and scanPos[i,2]==clipped_scanPos[j,2]:
                RF[i] = abs(clipped_RF[j])
                U2[i] = clipped_U2[j]

            # Otherwise indentor does not contact molecules surface and force left as zero for linear indentor displacement
            else:
                U2[i] = np.linspace(0,-indentionDepth,N)

    return U2, RF, N
```

In [ ]:
```
def DataPlot(scanPos, U2, RF, N):
    '''
    Produces scatter plot of indentation depth and reaction force to visualise and check simulation data.

        Parameters:
            scanPos (arr) - Array of coordinates [x,y] of scan positions to image biomolecule
            U2 (arr)      - Array of indentors z displacement over scan position
            RF (arr)      - Array of reaction force on indentor reference point
            N (int)       - Number of frames in  ABAQUS simulation/ time step
    '''

    # Initialise array for indentor force and displacement
    tipPos   = np.zeros([len(scanPos)*N,3])
    tipForce = np.zeros(len(scanPos)*N)

    # print(scanPos.shape)
    # print(RF.shape, U2.shape)
    # print(tipPos.shape, tipForce.shape)

    # Initialise count
    k = 0

    # Loop over array indices
    for i in range(len(scanPos)):
        for j in range( N ):
            #  Set array values for tip force and displacement
            tipPos[k]   = [scanPos[i,0], scanPos[i,1] , U2[i,j]]
            tipForce[k] = abs(RF[i,j])

            # Count array index
            k+=1

    # Scatter plot indentor displacement over scan positions
    fig1 = plt.figure()
    ax1 = plt.axes(projection='3d')

    ax1.scatter3D(tipPos[:,0], tipPos[:,1], tipPos[:,2])

    ax1.set_xlabel(r'x coordinate/ Length (nm)')
```

```
    ax1.set_ylabel(r'y coordinate/ Width(nm)')
    ax1.set_zlabel(r'z coordinate/ Height (nm)')
    ax1.set_title('Tip Position for Raster Scan')
    plt.show()

    # Scatter plot of force over scan positions
    fig2 = plt.figure()
    ax2 = plt.axes(projection='3d')

    ax2.scatter3D(tipPos[:,0], tipPos[:,1], tipForce)

    ax2.set_xlabel(r'x coordinate/ Length (nm)')
    ax2.set_ylabel(r'y coordinate/ Width(nm)')
    ax2.set_zlabel('Force N')
    ax2.set_title('Force Scatter Plot for Raster Scan')
    ax2.view_init(50, 35)
    plt.show()
```

AFM Image Functions

Calculate contours/z heights of constant force in simulation data for given threshold force and visualise.

In [ ]:
```
def ForceContours(U2, RF,forceRef, scanPos, baseDims, binSize):
    '''
    Function to calculate contours/z heights of constant force in simulation data for given threshold force.

        Parameters:
            U2 (arr)         - Array of indentors z displacement over scan position
            RF (arr)         - Array of reaction force on indentor reference point
            forceRef (float) - Threshold force to evaluate indentation contours at (pN)
            scanPos (arr)    - Array of coordinates [x,y,z] of scan positions to image biomolecule
            baseDims (list)  - Geometric parameters for defining base/ substrate structure [width, height, depth]
            binSize (float)  - Width of bins that subdivid xy domain during raster scanning/ spacing of the positions sampled over
        Return:
            X (arr) - 2D array of x coordinates over grid positions
            Y (arr) - 2D array of y coordinates over grid positions
            Z (arr) - 2D array of z coordinates of force contour over grid positions
    '''

    # Initialise dimensional variables
    xNum = int(baseDims[0]/binSize)+1
    yNum = int(baseDims[1]/binSize)+1

    # Initialise contour array
    forceContour = np.zeros(len(RF))

    # Loop over each reaction force array, i.e. each scan positions
    for i in range(len(RF)):

        # If maximum for at this position is greater than Reference force
        if np.max(RF[i]) > forceRef:
            # Return index of force threshold and store related depth
            j = [ k for k,v in enumerate(RF[i]) if v > forceRef][0]

            # Set surface height for reference height
            forceContour[i] = scanPos[i,2] + U2[i,j]

        # If no value above freshold set value at bottom height
        else:
            forceContour[i] = scanPos[i,2] + U2[i,-1]

    # Format x,y,z position for force contour
    X  = scanPos.reshape(yNum, xNum, 3)[:,:,0]
    Y  = scanPos.reshape(yNum, xNum, 3)[:,:,1]
    Z  = forceContour.reshape(yNum, xNum)

    return X, Y, Z
```

In [ ]:
```
def ContourPlot(X, Y, Z, baseDims, forceRef, contrast, pdb):
    '''
    Function to plot force contor produced from simulation. Plots 3D wire frame image and a 2D AFM image.

        Parameters:
            X (arr)          - 2D array of x coordinates over grid positions
            Y (arr)          - 2D array of y coordinates over grid positions
            Z (arr)          - 2D array of z coordinates of force contour over grid positions
            baseDims (list)  - Geometric parameters for defining base/ substrate structure [width, height, depth]
            forceRef (float) - Threshold force to evaluate indentation contours at (pN)
            contrast (float) - Contrast between high and low values in AFM heat map (0-1)
    '''
    #  ----------------------------------------------------3D Plots-----------------------------------------------------
    # Plot 3D Contour Plot
    fig = plt.figure()
    ax = plt.axes(projection = "3d")
    ax.contour3D(X,Y, Z, 30, cmap='afmhot')
    # ax.plot_wireframe(X,Y, Z)
    ax.plot_surface(X,Y, Z, cmap='afmhot')

    ax.set_xlabel(r'x coordinate/ Length (${\AA}$)')
    ax.set_ylabel(r'y coordinate/ Width(${\AA}$)')
    ax.set_zlabel(r'z coordinate/ Height (${\AA}$)')
    ax.set_zlim(0,7)
    ax.set_title('Contour Plot for Force of {0}pN'.format(forceRef))
    ax.view_init(60, 35)
    # ax.view_init(90, 0)
    plt.show()


    #  ----------------------------------------------------2D Plots-----------------------------------------------------
    # 2D heat map/ contour plot with interpolation
    fig, ax = plt.subplots(1, 2)
    im = ax[0].imshow(Z, origin= 'lower', cmap='afmhot', interpolation='bicubic',vmin=0, vmax= (contrast)*np.max(Z),
                      extent=(-baseDims[0]/2,baseDims[0]/2,-baseDims[1]/2,baseDims[1]/2)  )
    ax[0].set_xlabel(r'x (${\AA}$)')
    ax[0].set_ylabel(r'y (${\AA}$)')
    ax[0].axes.set_aspect('equal')

    # 2D heat map/ contour plot without interpolation
    im = ax[1].imshow(Z, origin= 'lower', cmap='afmhot',vmin=0, vmax= (contrast)*np.max(Z),
                      extent=(-baseDims[0]/2,baseDims[0]/2,-baseDims[1]/2,baseDims[1]/2)  )
    ax[1].set_xlabel(r'x (${\AA}$)')
    ax[1].set_ylabel(r'y (${\AA}$)')
    ax[1].axes.set_aspect('equal')
    plt.subplots_adjust(wspace = 0.5)
```

```
    cbar= fig.colorbar(im, ax= ax.ravel().tolist(), orientation='horizontal')
    cbar.set_label(r'z (${\AA}$)')

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\AFMSimulationMolecule-'+pdb+'.png', bbox_inches = 'tight')
    plt.show()

#     N = 3
#     Z1 = np.copy(Z.T)
#     for i in range(N):
#         Z1 = np.append(Z1,Z1).reshape(2*Z1.shape[0],Z1.shape[1])

#     # 2D heat map/ contour plot with interpolation
#     fig, ax = plt.subplots(1, 1, figsize = (5,5*(2**N)*baseDims[1]/baseDims[0]) )
#     im = ax.imshow(Z1.T, origin= 'lower', cmap='afmhot', interpolation='bicubic',vmin=0, vmax= (contrast)*np.max(Z))
```

In [ ]:
```
def HardSphereAFM(scanPos, baseDims, binSize, clearance, contrast):
    '''
    Plot the molecules atoms surfaces and scan positions to visualise and check positions.

        Parameters:
            scanPos (arr)      - Array of coordinates [x,y,z] of scan positions to image biomolecule and initial heights/ hard sphere boundary
            baseDims (list)    - Geometric parameters for defining base/ substrate structure [width, height, depth]
            binSize (float)    - Width of bins that subdivid xy domain during raster scanning/ spacing of the positions sampled over
            clearance (float)  - Clearance above molecules surface indentor is set to during scan
            contrast (float)   - Contrast between high and low values in AFM heat map (0-1)
    '''
    # Initialise dimensional variables
    xNum = int(baseDims[0]/binSize)+1
    yNum = int(baseDims[1]/binSize)+1
    Z    = scanPos.reshape(yNum, xNum, 3)[:,:,2] - clearance

    #  ----------------------------------------------------2D Plots---------------------------------------------------------
    # 2D heat map/ contour plot with interpolation
    fig, ax = plt.subplots(1, 2)
    im = ax[0].imshow(Z, origin= 'lower', cmap='afmhot', interpolation='bicubic',vmin=0,  vmax= (contrast)*np.max(Z),
                      extent=(-baseDims[0]/2,baseDims[0]/2,-baseDims[1]/2,baseDims[1]/2)  )
    ax[0].set_xlabel(r'x (${\AA}$)')
    ax[0].set_ylabel(r'y (${\AA}$)')
    ax[0].axes.set_aspect('equal')

    # 2D heat map/ contour plot without interpolation
    im = ax[1].imshow(Z, origin= 'lower', cmap='afmhot',vmin=0,  vmax= (contrast)*np.max(Z),
                      extent=(-baseDims[0]/2,baseDims[0]/2,-baseDims[1]/2,baseDims[1]/2)  )
    ax[1].set_xlabel(r'x (${\AA}$)')
    ax[1].set_ylabel(r'y (${\AA}$)')
    ax[1].axes.set_aspect('equal')

    plt.subplots_adjust(wspace = 0.5)
    cbar= fig.colorbar(im, ax= ax.ravel().tolist(),  orientation='horizontal')
    cbar.set_label(r'z (${\AA}$)')

    fig.savefig('C:\\Users\\Joshg\\Documents\\UCL\\Masters Project\\Figure\\AFMSimulationMolecule-'+pdb+'_HS.png', bbox_inches = 'tight')
    plt.show()
```

Simulation Script

Final simulation function

In [ ]:
```
def AFMSimulation(host, port, username, password, remotePath, localPath, abqCommand, fileName, subData,
                  pdb, rotation, surfaceApprox, indentorType, rIndentor, theta_degrees, tip_length,
                  indentionDepth, forceRef, contrast, binSize, clearance, meshSurface, meshBase, meshIndentor,
                  timePeriod, timeInterval, **kwargs):
    '''
    Final function to automate simulation. User inputs all variables and all results are outputted. The user gets a optionally get a surface plot of scan positions.
    Produces a heatmap of the AFM image, and 3D plots of the sample surface for given force threshold.

        Parameters:
            host (str)             - Hostname of the server to connect to
            port (int)             - Server port to connect to
            username (str)         - Username to authenticate as (defaults to the current local username)        -
            password (str)         - password (str) – Used for password authentication; is also used for private key decryption if passphrase is not given.
            remotePath (str)       - Path to remote file/directory
            localPath (str)        - Path to local file/directory
            abqCommand (str)       - Abaqus command to execute and run script
            fileName (str)         - Base File name for abaqus input files
            subData (list)         - Data for submission to serve queue [walltime, memory, cpus]
            pdbid (str)            - PDB (or CSV) file name of desired biomolecule
            rotation (list)        - Array of [xtheta, ytheta, ztheta] rotational angle around coordinate axis'
            surfaceApprox (float)  - Percentage of biomolecule assumed to be not imbedded in base/ substrate. Range: 0-1
            indentorType (str)     - String defining indentor type (Spherical or Capped)
            rIndentor (float)      - Radius of spherical tip portion
            theta_degrees (float)  - Principle conical angle from z axis in degrees
            tip_length (float)     - Total cone height
            indentionDepth (float) - Maximum indentation depth into surface
            forceRef (float)       - Threshold force to evaluate indentation contours at, mimics feedback force in AFM (pN)
            contrast (float)       - Contrast between high and low values in AFM heat map (0-1)
            binSize(float)         - Width of bins that subdivid xy domain during raster scanning/ spacing of the positions sampled over
            clearance(type:float)  - Clearance above molecules surface indentor is set to during scan
            meshSurface (float)    - Value of indentor mesh given as bin size for vertices of geometry in Angstrom (x10-10 m)
            meshBase (float)       - Value of indentor mesh given as bin size for vertices of geometry in Angstrom (x10-10 m)
            meshIndentor (float)   - Value of indentor mesh given as bin size for vertices of geometry in Angstrom (x10-10 m)
            timePeriod(float)      - Total time length for ABAQUS simulation/ time step (T)
            timeInterval(float)    - Time steps data sampled over for ABAQUS simulation/ time step (dt)

            kwargs:
                Submission ('serial'/ 'paralell') - Type of submission, submit pararlell scripts or single serial script for scan locations {Default: 'serial'}
                Preprocess (bool)                 - If false skip preprocessing step of simulation {Default: True}
                SurfacePlot (bool)                - If false skip surface plot of biomolecule and scan positions {Default: True}
                HSPlot (bool)                     - If false skip Hard Sphere AFM plot of biomolecule {Default: True}
                Transfer (bool)                   - If false skip file transfer step of simulation {Default: True}
                Part (bool)                       - If false skip part creation step of simulation {Default: True}
                Input (bool)                      - If false skip input file creation step of simulation {Default: True}
                Batch (bool)                      - If false skip batch submission step of simulation {Default: True}
                Queue (bool)                      - If false skip queue completion step of simulation {Default: True}
                Analysis (bool)                   - If false skip odb analysis step of simulation {Default: True}
                Retrieval (bool)                  - If false skip data file retrivial from remote serve {Default: True}
                Postprocess (bool)                - If false skip postprocessing step to produce AFM image from data {Default: True}
                DataPlot (bool)                   - If false skip scatter plot of simulation data {Default: True}
    '''
    T0 = time.time()

    #  ----------------------------------------------------Pre-Processing---------------------------------------------------------
    if 'Preprocess' not in kwargs.keys() or kwargs['Preprocess'] == True:
        t0 = time.time()
```

```
        # Extract molecules structure, and produce array of atoom coordinates, element, radius and dimension of base/substrate
        structure, view = PDB(pdb)
        atom_coord, atom_element, atom_radius, surfaceHeight, baseDims = MolecularStructure(structure, rotation, rIndentor, surfaceApprox)

        # Calculate tip geometry to create indentor and calculate scan positions over molecule for imaging
        tipDims = TipStructure(rIndentor, theta_degrees, tip_length)
        scanPos, clipped_scanPos = ScanGeometry(atom_coord, atom_radius, atom_element, indentorType, tipDims, baseDims, surfaceHeight, binSize, clearance)

        # Set list of simulation variables and export to current directory
        variables = [timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor, indentionDepth, surfaceHeight]
        ExportVariables(atom_coord, atom_element, atom_radius, clipped_scanPos, scanPos, variables, baseDims, tipDims, indentorType)

        t1 = time.time()
        print('Preprocessing Complete - ' + str(timedelta(seconds=t1-t0)) )
        print('Number of Scan Positions:', len(clipped_scanPos))

        if 'HSPlot' in kwargs.keys() and kwargs['HSPlot'] == True:
            HardSphereAFM(scanPos, baseDims, binSize, clearance, contrast)

        # Option plot for surface visualisation
        if 'SurfacePlot' in kwargs.keys() and kwargs['SurfacePlot'] == True:
            SurfacePlot(atom_coord, atom_radius, atom_element, scanPos, clipped_scanPos)

    # Condition to skip preprocessing step if files already generated previously
    else:
        # Check if simulation files are accessible in curent directory to use if pre=processing skipped
        try:
            atom_coord, atom_element, atom_radius, variables, baseDims, scanPos, clipped_scanPos     = ImportVariables()
            timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor, indentionDepth, surfaceHeight = variables

        # If file missing prompt user to import/ produce files
        except:
            print('No Simulation files available, run preprocessing or import data')


    #  ------------------------------------------------------Remote Simulation-----------------------------------------------------------
    # SSH to remote cluster to perform ABAQUS simulation and analysis from scripts and data files
    csvfiles = ("atom_coords.csv","atom_elements.csv","atom_radius_keys.csv", "atom_radius_values.csv",
                "clipped_scanPos.csv", "scanPos.csv","variables.csv","baseDims.csv", "tipDims.csv", "indentorType.txt")
    abqfiles = ('AFMSurfaceModel.py', 'AFMRasterScan.py', 'AFMODBAnalysis.py')

    RemoteSubmission(host, port, username, password, remotePath, localPath, csvfiles, abqfiles, abqCommand, fileName, subData, clipped_scanPos, **kwargs)


    #  ------------------------------------------------------ Post-Processing-------------------------------------------------------------
    if  'Postprocess' not in kwargs.keys() or kwargs['Postprocess'] == True:

        # Check if all simulation files are accessible in curent directory for post-processing
        try:
            atom_coord, atom_element, atom_radius, variables, baseDims, scanPos, clipped_scanPos                 = ImportVariables()
            timePeriod, timeInterval, binSize, meshSurface, meshBase, meshIndentor, indentionDepth, surfaceHeight = variables
            clipped_U2 = np.array(np.loadtxt('U2_Results.csv', delimiter=","))
            clipped_RF = np.array(np.loadtxt('RF_Results.csv', delimiter=","))

        # If file missing prompt user to import/ produce files
        except:
            print('Missing Simulation files, run preprocessing or import data')


        #  ------------------------------------------------------ Data-Processing-------------------------------------------------------------
        # Process simulation data to include full range of scan positions
        U2, RF, N  =  DataProcessing(clipped_RF, clipped_U2, scanPos, clipped_scanPos, indentionDepth, timePeriod, timeInterval)
        if 'DataPlot' in kwargs.keys() and kwargs['DataPlot'] == True:
            DataPlot(scanPos, U2, RF, N)


        #  ------------------------------------------------------AFM Force Contour-----------------------------------------------------------
        # Return force contours and plot in AFM image
        X,Y,Z = ForceContours(U2, RF, forceRef, scanPos, baseDims, binSize)
        ContourPlot(X,Y,Z, baseDims, forceRef, contrast, pdb)

        if 'HSPlot' in kwargs.keys() and kwargs['HSPlot'] == True:
            HardSphereAFM(scanPos, baseDims, binSize, clearance, contrast)

    # Return final time of simulation
    T1 = time.time()
    print('Simulation Complete - ' + str(timedelta(seconds=T1-T0)) )
```

Simulation Interface

In [ ]:

```
#  ------------------------------------------Remote Sever Variables-----------------------------------------------------
# host = "kathleen.rc.ucl.ac.uk"
host = "myriad.rc.ucl.ac.uk"
port = 22
username = "zcapjgi"
password = "ucl.Giblj003.315/Burnj003.315"

home     = '/home/zcapjgi'
scratch  = '/scratch/scratch/zcapjgi'

bashCommand = 'qsub'
abqCommand  = 'module load abaqus/2017 \n abaqus cae -noGUI'

# host = "128.40.163.27"
# port = 22
# username = "giblnbrnhm_j"
# password = "axenub13"

# home = '/home/giblnbrnhm_j@MECHENG2012'

# abqCommand  = '/opt/abaqus2018/abq2018 cae -noGUI'
# abqCommand  = '/opt/abaqus614/Commands/abq6141 cae -noGUI'

#  ------------------------------------------Submission Variables-----------------------------------------------------
fileName = 'AFMRasterScan-Pos'
subData  = ['24:0:0', '5G', '64']
abqDir   = '/ABAQUS/AFM_Simulations/Simulation_Code-Test-8'

localPath  = os.getcwd()
remotePath = scratch + abqDir

#  ------------------------------------------Simulation Variables-----------------------------------------------------
# Surface variables                          #
```

```
pdb = '1bna'                                    # '1bna' 'bdna8' '4dqy' 'pyn1bna'
rotation = [0, 270, 60]                         # degrees
surfaceApprox = 0.2                             # arb
                                                #
# Indentor variables                            #
indentorType = 'Capped'                         #
rIndentor = 4 # 10                              # (x10^-10 m / Angstroms)
theta_degrees = 5                               # degrees
tip_length = 50                                 # (x10^-10 m / Angstroms)
                                                #
# Scan variables                                #
clearance = 0.5                                 # (x10^-10 m / Angstroms)
indentionDepth = clearance + 5                  # (x10^-10 m / Angstroms)
binSize  = 4                                    # (x10^-10 m / Angstroms)
forceRef = 1                                    # (x10^-10 N / pN)
contrast = 1.61                                 # arb
                                                #
# ABAQUS variable                               #
timePeriod   = 1.5                              # s
timeInterval = 0.1                              # s
meshSurface  = 1.2                              # (x10^-10 m / Angstroms)
meshBase     = 2                                # (x10^-10 m / Angstroms)
meshIndentor = 0.6                              # (x10^-10 m / Angstroms)


#  ----------------------------------------Simulation Script-----------------------------------------------------------
AFMSimulation(host, port, username, password, remotePath, localPath, abqCommand, fileName, subData,
              pdb, rotation, surfaceApprox, indentorType, rIndentor, theta_degrees, tip_length,
              indentionDepth, forceRef, contrast, binSize, clearance, meshSurface, meshBase, meshIndentor,
              timePeriod, timeInterval,

              Preprocess  = True,
              SurfacePlot = True,
              HSPlot      = True,

              Transfer    = False,
              Part        = False,
              Input       = False,
              Batch       = False,
              Queue       = False,
              Analysis    = False,
              Retrieval   = False,

              Postprocess = False,
              DataPlot    = False
             )
```

Bash Commands

In [ ]:
```
# host = "kathleen.rc.ucl.ac.uk"
# # host = "myriad.rc.ucl.ac.uk"
# port = 22
# username = "zcapjgi"
# password = "ucl.GibLj003.315/Burnj003.315"

# home     = '/home/zcapjgi'
# scratch  = '/scratch/scratch/zcapjgi'
```

In [ ]:
```
# RemoteCommand(host, port, username, password, '', home, 'qstat')
```

In [ ]:
```
# for i in range(383, len(clipped_scanPos)):
#     jobName = 'AFMRasterScan-Pos'+str(int(i))
#     RemoteCommand(host, port, username, password, '', remotePath,'qsub -N '+ jobName +' batchScript.sh')
```

In [ ]:
```
# RemoteCommand(host, port, username, password, '',home,"qdel '*' ")
```

In [ ]:
```
# t0 = time.time()

# fileType = ['*.023', '*.cid', '*.dat', '*.lck', '*.mdl', '*.com','*.SMABulk',
#             '*.msg', '*.prt', '*.sim', '*.sta', '*.stt', '*.tmp',  '*.SMAFocus']
# for file in fileType:
#     RemoteCommand(host, port, username, password, '',scratch, "find . -name "+ file +" -delete")

# t1 = time.time()
# print(t1-t0)
```

In [ ]:
```
# Remote_Terminal(host, port, username, password)
```