NPCoronaPredict (UnitedAtom and CoronaKMC) user guide

July 23, 2024

Chapter 1

Introduction

This document is designed to provide an overview of how to run the NPCoronaPredict pipeline, consisting of the UnitedAtom and CoronaKMC packages, for the prediction of nanoparticle (NP) to protein adsorption energies and the composition of the corona for an NP immersed in a particular environment. It is assumed that you have access to the publications describing the theory behind these tools and so this document will focus more on practical details about running the software packages.

Throughout, we use the convention that the substrate is a "nanoparticle" and the adsorbates are "proteins" to fit with the initial usage of UnitedAtom, where a "protein" is an assembly of "amino acids" (AA). In general, the protein can be any molecule which can be modelled as a set of component beads.

1.1 New Users First Steps

1. Follow the instructions in Section 1.2 to install and compile everything you need.
2. Calculate a protein - NP adsorption energy to get used to the GUI as described in Section 1.2.1 and the general input and output
3. Simulate a corona using the command-line tools in Section 1.3.1
4. Compute a whole set of proteins and NPs in Section 2.1.1

1.2 Installation

The most recent version of the code can be obtained using git from a repository stored on github:

```
git clone https://github.com/ucd-softmatterlab/NPCoronaPredict.git
```

UnitedAtom is written in C++ with all source files stored in the src directory and can be compiled using the command

```
make clean; make
```

which will require a C++ compiler and the Boost libraries to be installed.

The current version of CoronaKMC is implemented in Python 3 and is the CoronaKMC.py script. An older version (CoronaKMC-P2.py) is provided for Python 2.7 users but is no longer maintained and is lacking a significant amount of functionality compared to more recent versions.

Some optional GUI tools are included, NPDesigner and UAQuickRun. To install these, install QT:

```
sudo apt install qtchooser qtbase5-dev qt5-qmake
```

and run the commands

```
qmake -qt=qt5
make
```

in the folder of each GUI tool you want to build.

A complete installation from scratch would therefore look like:

```
mkdir NPCoronaPredict
git clone https://github.com/ucd-softmatterlab/NPCoronaPredict.git
cd NPCoronaPredict
sudo apt install libboost-all-dev libboost-filesystem-dev wget perl parallel
sudo apt install build-essential libsqlite3-dev libpng-dev libjpeg-dev
sudo apt install python3 python3-pip
pip3 install numpy scipy matplotlib argparse Bio
make clean
```

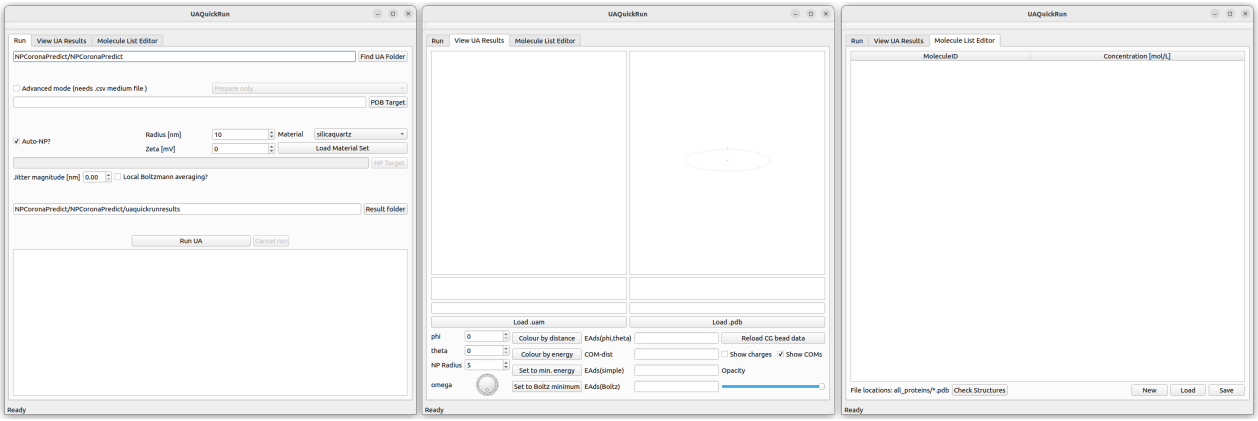

Figure 1.1: The three tabs of UAQuickRun showing the basic functionality for running and viewing UA output.

```
make
sudo apt install qtchooser qtbase5-dev qt5-qmake
cd UAQuickRun
qmake -qt=qt5
make
cd ../NPDesigner
qmake -qt=qt5
make
cd ../
```

which would then prepare you for running most of the tools. For new users, it's suggested you follow the quick tutorial in Section 1.2.1 to get started.

1.2.1 My first UA Run

For new users, it's recommended you use the UAQuickRun utility included in the repository so you can get an immediate feel for how things work without having to edit many text files.

First, make sure UA is compiled and python is installed, and you've installed QT following all the instructions above. If all goes well, you can then run the program either by double-clicking its icon in the UAQuickRun folder, or navigating to this folder in a command window and running

```
./UAQuickRun
```

and if not you likely have an issue with QT - you might need to install more packages to get all the correct headers. Assuming it works, this will bring up a simple GUI for running UA as shown in Figure 1.2.1.

The UA folder should be auto-set – if not, use the "Find UA Folder" button to find the "/home/whatever/UnitedAtom/unitedatom" folder. To begin, we need to find a protein structure. This can be conveniently done using the "Molecule List Editor" tab - this has more advanced uses, but for now we'll use it just to demonstrate fetching structures. Navigate to this tab and rightclick on the main window, then select "Add Molecule" from the popup. Double click "NewMolecule" and type in either a UniProtID or a PDB 4-letter code, then press enter. You'll see this entry is probably highlighted in red, meaning the software didn't find a matching structure (a file named MoleculeID.pdb located in all_proteins). If you don't have any in mind, "P02768" is human serum albumin. Now click "CheckStructures", which will check either the PDB databank (for four letter codes) or the AlphaFold database (for anything else) for your missing protein. Assuming it finds it, it'll download the structure and the highlighting will turn green. Don't worry about the New/Load/Save buttons or setting a concentration - that's for advanced use when you want to do a corona prediction.

Next, go to the "Run" tab. Click the PDB Target button and navigate to the protein you just downloaded in the all_proteins folder - this will tell UA which protein to run computations for. Leave "Auto-NP" checked and select a radius and zeta potential, then choose a material from the drop-down list e.g. carbon black. If you don't like the look of any of these, click the "Load Material Set" button and select the file "surface-pmfp/MaterialSetPMFP.csv" to load in some extra options. Finally enter a target folder to contain your results, and click Run UA. Assuming you compiled UA and have python correctly installed, after a brief delay (possibly a few minutes, if you picked a large protein) it'll tell you that the run is done and print some output to the big textbox.

Now you can move to the "View UA Results" tab. Use the "load .uam" button to find and load in the file generated by UA - you can see the path in the textbox on the UA Setup tab, e.g:

```
Info: Saving map to:

/home/myusername/NPCoronaPredict/NPCoronaPredict/uaquickrunresults/np1R_10_ZP_0/P02768_10_0.ua
```

and then load in the PDB from the same location as before. This will produce results similar to that shown in Figure 1.2.1. The left-hand graphic shows a heatmap plot of the data stored in the .uam results file, with strongly binding

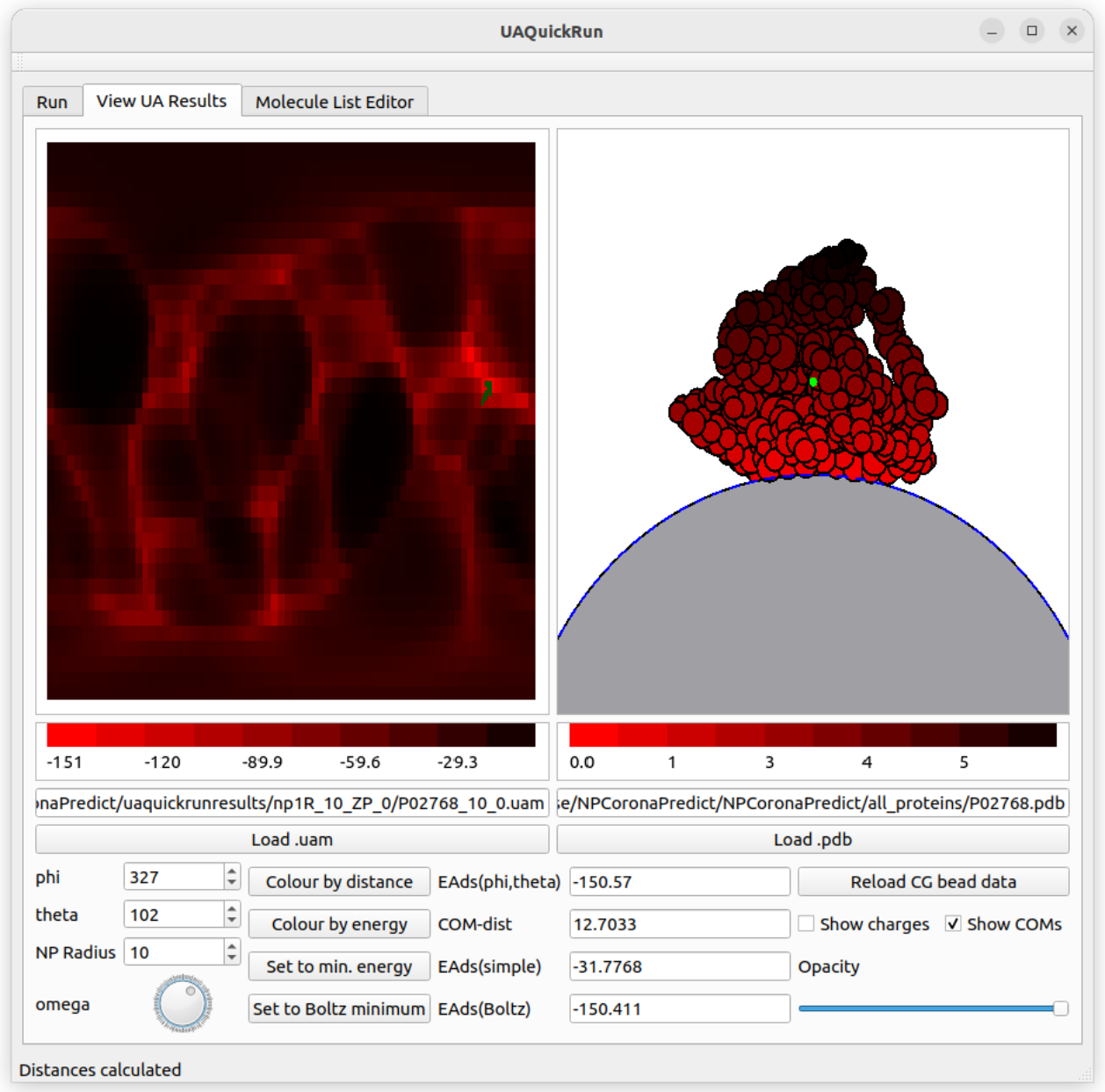

Figure 1.2: Example output from UAQuickRun for HSA adsorbing to a 10 nm nanoparticle.

protein-NP conformations shown in red (the colour scale at the bottom indicates values in kBT), with a green arrow pointing towards the currently selected orientation (pair of phi, theta values, see Section A.1 for details on how these are defined). Clicking on this image sets the orientation to be a new value, or finer control can be achieved by altering the phi, theta boxes directly. The right-hand window shows the biomolecule-NP complex at that orientation. By default, all the biomolecule beads are shown in black, its recommended to click "colour by distance" to apply shading to these beads based on their distance to the NP in the current orientation - this shading will stay fixed if you change orientation until you click this button again to update it. You can rotate the protein along the z axis with the omega dial - for these simple spherical NPs, this value omega doesn't alter the binding energy. The four text boxes in the center column show the values of the adsorption energy at the current orientation, the distance between the COM of the protein and the NP at that orientation, the simple orientation average energy, and the Boltzmann orientational average energy - these two averages will be discussed more later. Finally, the "set to min. energy" can be used to reset phi, theta to the values corresponding to a minimum of energy, while "set to boltz minimum" attempts to set the orientation to one in which each bead is as close as possible to its average distance from the surface of the NP. The third column contains a few options for controlling how the protein is plotted, in particular checking "Show charges" will outline charged residues based on their charge. Each residue is assigned a radius and charge based on its tag in the PDB file and the bead data file loaded into memory - you can change these definitions if needed by clicking "Reload CG bead data", if for example you disagree with certain bead radii or charges. The show COMs box plots a green dot for the centre of mass of the protein and a blue dot for the COM of the NP , and the opacity slider removes the central fill for CG protein beads - this can help visualise more complex cases where the NP is partly docked into the protein.

1.3 High-level pipelines

A frequent use case is the prediction of the corona for a simple nanoparticle (e.g. a sphere of a predefined material) in a medium consisting of proteins of known concentration. In this case, the provided NPCoronaPredict.py script is designed to automate the process and can be run using a command such as:

```
python3 NPCoronaPredict.py -r [NPRadius] -z [ZETA] \\
-m [MATERIALNAME] -p [OUTPUTFOLDERNAME] -o [PROTEINSET] -t [CORONASIMULATIONTIME]
```

where the first three options define the nanoparticle in question by radius (in nm), the excess surface potential of the NP beyond that included in the PMF (in mV) and the material from the list of predefined materials. The "-p" option defines the name of a folder in which to store the files generated. The "-o" option must be the path to a file containing a list of all biomolecules required to bepresent in the medium in the format,

```
#comment lines start with a # and are ignored during parsing
#all other lines are split by a comma into a protein name and a concentration in mol/L
protein1,1e-4
protein2,0.003
smallmolecule1,0.01
```

Each line in this file is required to have a matching structure stored in the all_proteins folder, e.g. "protein1.pdb", "protein2.pdb", "smallmolecule1.pdb". The -t option defines how many seconds of simulated time to run the corona simulation for. Further options are available and can be found using

```
python3 NPCoronaPredict.py --help
```

which lists all the available commands and details about what they do. This script is designed to allow rapid simulation of coronas for well-defined, single-bead NPs. To do so, perform the following steps:

1. Copy CG structures for UnitedAtom (from the Protein Databank/AlphaFold/MolToFragment/other) and place in the "all_proteins" folder (making this if it doesn't exist)
2. Create a medium file as described above. Remember that this is "molecule,concentration[M]" format and each molecule must have a matching molecule.pdb file in all_proteins.
3. Identify the material you want to use - the standard are all listed in MaterialSet.csv and the machine-learning extended set in surface-pmfp/MaterialSetPMFP.csv
4. Run NPCoronaPredict.py with a command string as shown above.

To save you some time, the NPCoronaPredict script will attempt to download some protein structures if it can't find them automatically. It can only do so if the name in the medium file looks like either "PDB-XXXX", where XXXX is the PDB ID code, or "AFDB-UNIPROTID", where UNIPROTID is the UniProtID e.g. P000001 for that protein. If it fails to find a match it will report this as an error Note that AlphaFoldDB contains most UniProtIDs unless they've been obsoleted or are longer than 1500 residues - these must be supplied manually. Here there's an important difference: the UAQuickRun GUI doesn't need the PDB- or AFDB- tags in front to find structures, while NPCoronaPredict does. This is so NPCoronaPredict only checks these if you explicitly tell it to.

1.3.1 My first corona prediction

Assuming you've already gone through the first tutorial above, you'll already have a protein downloaded into all_proteins (if not, pick a protein, get a .pdb structure, and save it there). For convenience, here we assume you have a structure named P02768.pdb present in this folder. We're now going to use the command-line level interface to run a corona prediction. The first step is preparing a medium file - the list of proteins and their abundances in the absence of an NP. Open up a text editor and make a file that looks like

```
#Molecule,Concentration
P02768,0.001
```

and save it somewhere in the NPCoronaPredict folder (or a subfolder, if you prefer), e.g. to "demo_medium.csv". You can also make these files using the Molecule List Editor in UAQuickRun and saving these out.

Now open a terminal window, navigate to the NPCoronaPredict folder and run

```
python3 NPCoronaPredict.py -r 10 -z 0 -m graphene -p corona_predict_demo -t 0.01 -o demo_mediu
```

This will call UnitedAtom, BuildCoronaParams and finally CoronaKMC to predict the corona abundances of P02768 on your NP (here, graphene of radius 10 nm, as a simple if unrealistic demonstration), letting the simulation run for 0.01 seconds and saving all the output to the folder corona_predict_demo. Try changing the material, timespan for corona predictions, or adding more proteins to demo_medium.csv. When it's done, check the resulting output folder - this will contain data about the evolution of the corona and its final contents.

Alternatively, if you want to keep using the GUI, try setting up this run in UAQuickRun. Prepare your list of molecules on the Molecule List Editor tab, making sure you give each a concentration. Run Check Structures and then

click "Save" to save out your medium file. Then on the Run tab, check "Advanced Mode" then click "Medium file" to find the medium file you just made. Then from the drop-down above this go from "Prepare only" to "Prepare + UA + BCP + KMC". After that, you can define your NP as before. Note that UAQuickRun doesn't have the ability to show corona results other than in the text box.

The above command only requests 0.01 seconds of corona simulation - this is a very short amount of time and nowhere near enough for the corona to reach equilibrium. Try increasing this! You can also try appending –steady to the command string and this will ask CoronaKMC to try to estimate the final state - this might be more useful for comparison to experiment.

Chapter 2

UnitedAtom

2.1 Introduction

In brief, UnitedAtom takes as input a set of potentials representing NP-AA interactions for a specific nanomaterial (e.g. anatase, amorphous carbon), a structure built from a set of AAs, and some variables defining a particular nanoparticle from the underlying nanomaterial (e.g. a 5 nm gold sphere with an extra applied surface potential of +10mV). The output is then a table of data representing the adsorption affinity of each orientation of the input structure relative to the surface of the NP, commonly plotted as a heatmap. Running UA is in principle straightforward. Store the protein/biomolecule structures of interest in a folder, make sure you have a set of PMFs and Hamaker constants describing the short range and long range AA-NP interactions respectively, edit the configuration file to select these, use the command

```
./UnitedAtom --config-file=myconfigfile.config
```

to start UnitedAtom and then go wait a while (typically, up to a few minutes per protein per NP) until its done.

2.1.1 My second UA run

The main advantage of running UA directly is that it allows much more flexibility in terms of parameters and the calculations performed. For these direct runs, you need to supply the configuration file mentioned above. To begin, either copy and paste the sample_config/sample_config.config file to the same folder as the UnitedAtom executable or use the GUI/NPCoronaPredict/RunUA.py to produce a configuration file for your preferred material. Then edit the pdb-target line to target either a specific protein or a whole folder of proteins - UA will run calculations for all of these. In this example, we'll ask UA to look at our target proteins on a set of NPs of the same material over a range of radii. Edit the nanoparticle-radius line to read

```
nanoparticle-radius = [1.0, 5.0, 10.0]
```

and we'll also add a slight extra surface potential via the zeta-potential option:

```
zeta-potential = [-0.001, 0.0, 0.001]
```

Note radii are in nm, zeta potential in V. Save the config file to myconfig.config and run

```
./UnitedAtom --config-file=myconfig.config
```

Then wait for a bit - it'll run UA for every protein on each of the nine NPs generated (R=1, zp = -1mV, R=1, zp = 0mV .... R=10, zp = 1mV). The resulting .uam files are plain-text and can be post-processed using e.g. tools/plotmap to visualise them.

2.1.2 What's it actually doing?

A run of UA works briefly as follows:

1. Read in the config file
2. Load in the protein structures
3. Assemble all NP radius-zeta combinations for the chosen material OR load in all target NPs
4. For each NP, build the total potential for each AA interacting with that NP.
5. For each protein, rotate to a specified orientation, sum over AA potentials to get the protein-NP potential, integrate over distance to get an adsorption energy. Repeat multiple times at slight perturbations of each orientation to produce an average.
6. When orientation averaging is done, write results out to a file.

7. Repeat for next protein, then for next NP, until everything is done.

The orientation θ, ϕ is defined such that the input co-ordinates are rotated by an angle of $-\phi$ around the z-axis, followed by a rotation of $\pi - \theta$ around the y axis. Note that because 3D rotations are non-commutative, you do not get the same results if you apply these in the opposite order and, moreover, the rotation around z still has an effect even for a spherical NP because it alters which residues are moved by the second rotation step. For cylinders and NP-complexes, a final rotation of ω is applied around the z axis after the θ rotation. This rotation is currently stored in the filename rather than the table of output for backwards compatability with post-processing scripts. This step is redundant for spherical NPs and is omitted for these.

2.2 Theory

The theory is covered in many other places, so here is just a quick overview. The basic, one-NP version of UA constructs a potential for each AA bead species S

$$U_S(r) = U_{surf}(S, r) + U_{el}(S, r) + U_H(S, r) \tag{2.1}$$

where r is the distance between the centre of the NP and the centre of the bead, U_{surf} is a short-range potential, U_{el} is an electrostatic potential, and U_H is a Hamaker-like potential. U_{surf} is a tabulated potential extracted from metadynamics simulations to include the very close range repulsion, some electrostatic terms, solvent effects, mid-range vdW attraction, etc. A correction factor is applied to map these from the tabulated form generally computed for a plane to an NP of finite radius of curvature. The electrostatic term is evaluated in the Debye-Huckel approximation and has simple forms for the spherical,

$$U_{el} = \psi_0 e^{-h/l} \frac{R}{h + R} \tag{2.2}$$

cylindrical (using the Bessel K function $K_0(r)$),

$$U_{el} = \psi_0 \frac{K_0((R + h)/l)}{K_0(R/l)} \tag{2.3}$$

and planar

$$U_{el} = \psi_0 e^{-h/l} \tag{2.4}$$

cases, where h is the distance from the surface, l is the Debye length , R is the radius of the NP and ψ_0 is the value of the potential at $h = 0$, which is typically and inaccurately referred to as the zeta potential. Finite cubes are evaluated using an expansion in spherical harmonics but are generally very well approximated by the planar expression, especially if the cube is large compared to the Debye length.

The Hamaker-like potential is an integral of the $1/r^6$ vdW potential over the volumes of the NP and bead, neglecting elements with a distance less than the exclusion distance r_e

$$U_H(S, r) \propto \int \int \frac{1}{r'^6} \Theta_H(r' - r_e) dV_{NP} dV_{AA} \tag{2.5}$$

where r' is the distance between two elements dV_{NP}, dV_{AA} and Θ_H is the Heaviside theta function which is zero for $r_e > r'$. This integral has a complex analytical solution for spherical NPs which is implemented into UA, and can be solved numerically for cylinders (also implemented). The result is a smooth function (see Figure 2.2) which avoids introducing a discontinuity into the potential which would be caused if this potential was only switched on if the bead centre is at a certain distance from the NP. Note that the Hamaker potential is generally quite small even without this exclusion effect, generally reaching only the order of a Hamaker constant for a typical residue size almost touching an NP of infinite extent, and the short-range repulsion inherent in the PMFs prevents the beads from reaching this close.

For a given protein consisting of a set of beads j with species S_j and NP-AA distances r_j, the total NP-protein potential is given by summation,

$$U_{AA-NP}(r_s) \tag{2.6}$$

where r_s is a measure of the NP-protein distance, generally the closest approach or COM.

2.3 The configuration file

2.3.1 A standard file

Below is a standard UA configuration file, suitale for running a basic calculation of all structures in the all_proteins folder for Anatase NPs of all combinations given by the "outer product" of the nanoparticle-radius and zeta-potential lists, in this case: 5nm at -0.001V, 5 nm at 0 V, 5nm at 0.001 V, 10nm at -0.001V, 10nm at 0 V, 10 nm at 0.001 V.

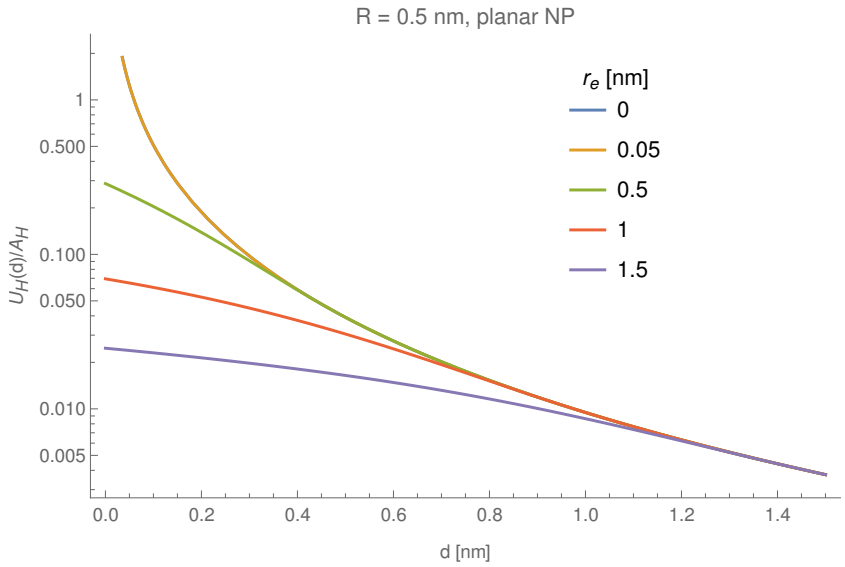

Figure 2.1: Plots of the Hamaker-like potential (in units of the Hamaker constant) used in UnitedAtom for an infinite planar NP and a residue of radius 0.5 nm with surface-surface distance d, shown for multiple values of the exclusion distance r_e.

```
#Autogenerated UA Config file, modified to show extra features
output-directory = UAOutput
pdb-target = all_proteins
nanoparticle-radius = [5.0, 10.0]
np-type = 1
pmf-directory = surface/TiO2-ana-101
hamaker-file = hamaker/TiO2_Anatase.dat
enable-surface
enable-core
enable-electrostatic
simulation-steps = 2000
potential-cutoff=5.0
potential-size = 1000
angle-delta = 5.0
bjerum-length=0.716
debye-length=0.785
temperature = 300.0
zeta-potential = [-0.001, 0, 0.001]
amino-acids          = [ ALA, ARG, ASN, ASP, CYS, GLN, GLU, GLY, HIS, ILE, LEU, LYS, MET, PHE,
amino-acid-charges   = [ 0.0, 1.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.5, 0.0, 0.0, 1.0, 0.0, 0.0
amino-acid-radii     = [ 0.323, 0.429, 0.362, 0.356, 0.352, 0.386, 0.376, 0.285, 0.302, 0.401,
```

In order:

- output-directory : Folder in which to save output files.
- pdb-target : Parent directory for pdb structures, all files with a .pdb file extension in all subfolders of this directory are used as input. Or a single file can be specified.
- nanoparticle-radius : A single value or list of NP radii in nm to scan over.
- np-type : Shape ID. 1 = sphere, 2 = cylinder (with planar PMFs), 3 = cube (with planar PMFs), 4 = tube/hollow cylinder (with cylindrical PMFs), 5 = dense cylinder (with cylindrical PMFs). This selects the integration method, Hamaker function and PMF scaling.
- pmf-directory : Folder containing XXX.dat files for each AA bead.
- hamaker-file : File containing Hamaker data for all beads for that surface.
- enable-surface : Switch to include short-range potential (should be always on)
- enable-core : Switch to include Hamaker potential (should be always on, use Hamaker Null.dat to disable)
- enable-electrostatic : Switch to include electrostatic potential (should be always on, use zeta = 0 to disable instead)

- simulation-steps, potential-cutoff, potential-size,angle-delta : Leave as default values. These are largely overwritten internally.
- bjerum-length : in principle the Bjerrum length, in practice does nothing except if a very specific other option is enabled and it almost surely won't be. The standard UA electrostatic expression does not include the Bjerrum length. If you are writing/reading anything and the electrostatic expression uses the Bjerrum length, you/the author are mistaken.
- debye-length : the Debye length for electostatic interactions. This actually is used.
- temperature : the "temperature", used as a scaling factor to change interactions. Experimental, best left at 300K unless you're sure you know what you're doing.
- zeta-potential : Single number or list, values for the "zeta" potential in V (so usually quite small, as -50 to +50 mV is the normal range). Note that internally this is actually the potential at the surface of the NP, the zeta potential will be smaller.

The last three lists define the AA (or other small molecule) beads present. The first list defines each bead as a three-character ID code - note that longer codes will have undefined behaviour, likely leading to errors. The next two lists contain the charge and radius associated with each bead tag as in the above example, e.g. ALA is defined with a charge of 0.0 and radius 0.323, ARG with charge +1.0 and radius 0.429, etc. Additionaly, the Hamaker file must contain an entry for each bead tag, and the PMF folder must contain a TAG.dat file for each bead, e.g. ALA.dat, ASP.dat.

2.3.2 Extra options

The above general-purpose configuration file should be fine for most cases, but in certain circumstances some extra options may be needed. It is recommended that you check Config.h to see all available options, but here are some extra switches which can be enabled:

- pmf-cutoff : Set the Lennard-Jones cutoff used in the calculation of the short-range potential. This is assumed to be 1 nm by default, but if you used CHARMM for the calculation of PMFs it should be set to 1.2 nm. Note that this is the LJ cutoff and not the maximum distance recorded in the PMF.
- calculate-mfpt : Use mean-first-passage-time theory to estimate escape rates for the exact potential. This is very slow and generally unused.
- np-target : This will be discussed more later. It overwrites the automatic generation of NPs and allows for multi-material or multi-bead NPs to be used.
- pmf-prefix : String used to differentiate between different surface potentials for the same bead as in "prefix-XXX.dat". Exists only for historical reasons.
- imaginary-radius : Used in testing of hollow NPs, can be ignored.
- bounding-radius : Used primarily for multicomponent NPs, sets the radius at which the NP is taken to be solid and infinitely repulsive. Might be useful to edit if you're doing weird things and the auto-generated radius isn't working.
- overlap-penalty : Value in kbT to penalise configurations in which a protein-bead overlaps an NP-bead. Used to mitigate problems with the multicomponent model by enforcing some anisotropy.
- overlap-radiusfactor : Number > 0 , Used to resize AA beads for determining overlap with multicomponent models. Just leave it at 1 unless you have a really good reason not to.
- recalculate-zp : If non-zero, automatically adjust input zeta potentials to take into account the size of the NP and solution conditions, assuming that the input ZP corresponds to bjerrum length = 1 nm, debye length = 1nm, NP radius = 1nm. This is the only time the Bjerrum length matters and in practice you should just set the zeta potentials to whatever they actually are and not enable this option.
- save-potentials : If nonzero saves out the protein-NP potentials. Note that AA-NP potentials are automatically saved out to the pot-dat folder if this exists regardless of this option.
- disorder-stategy : Enables extra options for dealing with disordered residues (those with b-factors between disorder-minbound and disorder-maxbound defined on the next two lines). If disorder-strategy = 0, these are treated as normal residues. If disorder-strategy = 1, these residues are moved to the centre of the protein. disorder-strategy = 2 disables these residues entirely. These are most useful for dealing with AlphaFold structures with unphysical loops.
- disorder-minbound
- disorder-maxbound

- omega-angles : Specify a list e.g. $[0, 90, 180]$ of the third-step rotation angle ω to apply. If this option is omitted then no rotation is applied for spherical NPs and for cylindrical the rotation is done in steps of 45 degrees up to 180. This is only needed for NPs which are not symmetric with respect to rotation about the z axis, e.g. cylinders and brush-decorated NPs.

2.4 Input data - Hamaker files

The Hamaker file for a given material has the following format,

```
#Name   kT        Joules        kJ/mol
ALA     6.933     2.872E-20     17.332
ARG     8.348     3.458E-20     20.869
ASN     8.995     3.726E-20     22.487
ASP     9.209     3.814E-20     23.022
```

where the first column contains the three-letter tag associated with a given AA bead and the remaining columns are the NP-Water-AA Hamaker constant. These files are read by HamakerConstants.h with columns 1–7 providing the tag and columns 8–17 the energy in kBT (T=300), with all other entries ignored (but kept for ease of interpretation).

2.5 Input data - Surface potential/PMFs

The surface potential defines the short-range interaction between the NP and the AAs and is, typically, a potential of mean force including solvent effects, although any tabulated potential with a short-range repulsion will suffice. All beads defined in the configuration file with name XXX must have a corresponding XXX.dat file in the relevant directory. These files are defined in Surface.h and must be either comma-separated in the form distance(nm),energy(kj/mol) or with the distance stored in the columns 1–8 and the energy in columns 9–20, again in nm and kJ/mol. Empty lines and those starting with a # are ignored. If the surface potential does not match this formatting you will probably get inconsistent results or an error. Comma formatting is strongly recommended!

Important: UnitedAtom internally assumes that the PMFs used for input are either an infinite plane or an infinite cylinder (of radius 0.75 nm, to be specific). These are then corrected to the actual NP radius. If you've created PMFs for a small repeat unit, then UA will not give meaningful results because it will be trying to apply a correction to something that's already been corrected. You can either manually de-correct your PMFs to map from single bead to the equivalent of a plane or use multicomponent UA (discussed later) to build a raspberry model of your NP made from the individual beads.

2.6 Input structures - PDB

The biomolecule to be scanned over is defined using a structure in the pdb format. The input file is read and each line is scanned in turn as implemented in PDBFile.h, with scanning continuing until the end of file or the first "ENDMDL" line is reached. Briefly, if columns 1 – 4 contain the text "ATOM" and columns 14–15 the text "CA" then a bead is added to the internal structure for that molecule, else, the line is discarded. The bead is defined based on the following entries:

- Column 18–20: bead ID tag
- Column 31–38: x-coordinate (Angstrom)
- Column 39–46: y-coordinate (Angstrom)
- Column 47–54: z-coordinate (Angstrom)
- Column 55–60: occupancy
- Column 61–66: B-factor

Of these, the bead ID tag is used to select interaction potentials for that bead from the PMF folder and Hamaker file, and the x/y/z coordinates define where the bead is located. The occupancy gives an overall scaling factor for the potentials generated by that bead and is used such that disordered structures don't overcount beads and to allow for beads with multiple possible protonation states (e.g. histidine can be modelled as a HIE, HID, HIP bead triplet with occupancies summing to unity). Thus, a single alanine bead can be represented by:

```
ATOM      1  CA  ALA A   1       0.000   0.000   0.000  1.00  0.00
```

to make an ALA-type bead at 0,0,0 with occupancy 1, while histidine might be

```
ATOM      1  CA  HIE A   1       0.000   0.000   0.000  0.62  0.00
ATOM      1  CA  HID A   1       0.000   0.000   0.000  0.16  0.00
ATOM      1  CA  HIP A   1       0.000   0.000   0.000  0.22  0.00
```

to produce a total of one residue but accounting for multiple possible charge states.

In general, these input structures can be taken directly from the PDB repository, from the AlphaFold prediction set or from I-TASSER output, or manually created. Two optional forms of pre-processing are suggested:

1) The protein can be rotated to "canonical form" using the ProteinSetRotate.py script, which rotates the protein such that the largest moment of intertia is associated with the z axis, second largest with the y axis, and the dipole moment along the x axis is negative. In practice, this generally ensures that a prolate protein is aligned with the z axis such that the $\theta = 90$ orientations are typically the most strongly adsorbing, creating a characteristic horizontal band in the heatmap plots.

2) Propka processing: propka3 (must be installed separately!) is called for each protein in the set and used to estimate protonation state probabilities for all residues based on an input pH value. The structure is then updated so that all possible protonation states for each residue are included with that weighting in the occupancy field, and HIS is also modified to split into HID and HIE states. The result is a structure which smoothly interpolates between possible charge states at a given pH. Note that the structure itself is not denatured at all and so care must be used for pH changes significantly far from neutral.

Note that both of the above are combined in the PreprocessProteins.py script.

For the purposes of UA, only lines with the ATOM and CA strings in the correct locations matter. It is strongly recommended that further metadata is included in the file where possible, since it's not a good idea to rely on filenames alone, especially when the same protein may have multiple structures. Provided that all the required structure is in place, UA does not care whether or not the three-letter code corresponds to a traditional AA code (as can be guessed from the use of HIE/HID/HIP above) or even an amino acid at all and so this same format can be used for molecules other than proteins, provided that PMFs are available for all constituent beads.

Coarse-graining an arbitrary molecule is not straightforward, but approximate models can be constructed using the MolToFragments.py tool located in the tools folder. This is a command-line Python script which takes an input like:

```
python3 MolToFragments.py -m NewMoleculeName -s "CCCCO" -M EqualParts
```

where "NewMoleculeName" is the name to give the output, the -s option takes in a SMILES code for the molecule in question and -M chooses the method to use. This script attempts to fragment the input molecule, match components to those listed in the associated FragmentLookup.csv file, and write out a .pdb file suitable for input to UA with the fragments found. For example, the input:

```
python3 MolToFragments.py
-s "CCCCCCCCCCCCCCCCC(=O)OCC(COP(=O)([O-])OCC[N+](C)(C)C)OC(=O)CCCCCCCCCCCCCCCC"
-m DPPC-EP -M EqualParts
```

produces a file DPPC-EP.pdb with contents:

```
HEADER DPPC-EP                                          06-Sep-23
TITLE     DPPC-EP
REMARK SMILES: CCCCCCCCCCCCCCCCC(=O)OCC(COP(=O)([O-])OCC[N+](C)(C)C)OC(=O)CCCCCCCCCCCCCCCC
REMARK Generated: 2023-09-06 12:08:27
REMARK Method: EqualParts
REMARK 0:ILESCA-AC:ILE:CCCC
REMARK 1:PENTANE:X4P:CCCCC
REMARK 2:PENTANE:X4P:CCCCC
REMARK 3:EST-AC:EST:COC(C)=O
REMARK 4:CHOL-JS:X4L:C[N+](C)(C)CCO
REMARK 5:DPPC-EPNew0:!!!:COC=O
REMARK 6:DPPC-EPNew1:!!!:CO[PH](=O)[O-]
REMARK 7:VALSCA-AC:VAL:CCC
REMARK 8:ILESCA-AC:ILE:CCCC
REMARK 9:ILESCA-AC:ILE:CCCC
REMARK 10:ILESCA-AC:ILE:CCCC
ATOM      0  CA  ILE A   1     -13.5     4.51    -1.76  1.00  0.00           C
ATOM      1  CA  X4P A   2     -9.68     2.04   -0.672  1.00  0.00           C
ATOM      2  CA  X4P A   3     -5.91    0.885     1.15  1.00  0.00           C
ATOM      3  CA  EST A   4     -1.07    0.855    0.949  1.00  0.00           C
ATOM      4  CA  X4L A   5     0.384    -8.21   -0.379  1.00  0.00           C
ATOM      5  CA  !!! A   6       2.1   -0.714     2.42  1.00  0.00           C
ATOM      6  CA  !!! A   7    -0.428    -3.87    0.802  1.00  0.00           C
ATOM      7  CA  VAL A   8       5.3    -1.54     2.22  1.00  0.00           C
ATOM      8  CA  ILE A   9      7.45    0.238    0.879  1.00  0.00           C
ATOM      9  CA  ILE A  10      9.87     2.35   -0.838  1.00  0.00           C
ATOM     10  CA  ILE A  11      11.6     5.59    -2.01  1.00  0.00           C
END
```

In this output, the remark lines give the CG bead number, long name of the matched fragment, bead code, and the SMILES code for that bead. Bead codes marked with "!!!" do not have an already-defined PMF, which must be

generated or an alternative method used (-M ForwardsMatching only produces known beads, if this is an issue). The default FragmentSet.csv file assumes that PMFs for a large set of beads are available. If it isn't, you should remove entries from this file to represent only the PMFs you have for the surface of interest (but be aware that ForwardsMatching may fail in this case).

The allowed methods are BRICS (uses the standard BRICS) algorithm, EqualParts (attempts to produce fragments with relatively consistent sizes), ForwardsMatching (iteratively builds the molecule up from known beads, does not allow for the generation of new ones) and Exhaustive (considers many different templates, scores and chooses the best, very slow and not recommended). Of the three quick algorithms, BRICS and EqualParts should always produce a CG model but it may need PMFs to be generated. ForwardsMatching may not always work (especially if there are aromatic structures or you have a very limited fragment library) but when it does the result will only use pre-existing PMFs. The other methods may require the generation of new PMFs and provide output suitable for use in the PMFP software package, see Section 2.11. Output from this package (assigned bead code and SMILES ID) can be added to FragmentSet.csv as needed to extend the range of molecules which can be parameterised.

2.7 Output: the .uam file

The main output of UA is a .uam file for each protein-NP pair, which has the following format:

#phi	theta	EAds/kbT=300	Error(Eads)/kbT=300	min_surf−surf−dist/nm	mfpt∗DiffusionCoeff/nmˆ2	EA
0.0	0.0	−3.73681	0.01376	0.32033	−1.00000e+00	−9.32087	5.32033
5.0	0.0	−3.73211	0.01528	0.31476	−1.00000e+00	−9.30914	5.31476
10.0	0.0	−3.72567	0.01541	0.30752	−1.00000e+00	−9.29309	5.30752
15.0	0.0	−3.72474	0.01652	0.30929	−1.00000e+00	−9.29075	5.30929
20.0	0.0	−3.72432	0.02027	0.30895	−1.00000e+00	−9.28971	5.30895
25.0	0.0	−3.72646	0.02086	0.31663	−1.00000e+00	−9.29504	5.31663

Each row contains to one set of orientational samples for that protein-NP pair, divided into bins of θ, ϕ values. These columns contain, in order:

- phi: Left-hand edge of the angular bin for ϕ values - note the average is generally $\phi + 2.5$
- theta: Left-hand edge of the angular bin for θ values - note the average is generally $\theta + 2.5$
- EAds/kbT=300: Average of the EAds value for that bin
- Error(Eads): Standard deviation NOT STANDARD ERROR for that bin. Be very aware that this is an SD for error propagation!
- min_surf-surf-dist : Distance between the closest bead centre and the surface of the NP. Yes, its bead centre and not surface-surface distance. This is due to how UA stores distances.
- mfpt*DiffusionCoeff : product of the estimated MFPT and an unknown diffusion coefficient - divide this value by your diffiusion coefficient to get the MFPT. This is usually set to -1 if the MFPT isn't calculated.
- EAds/kJMol : Adsorption energy again, but in units kJ/mol to make converting between temperatures easier.
- ProtSurf_NPCentre dist: Distance beteen the centre of the NP and the closest AA centre - in other words, min-surf-surf + the radius of the NP.

2.8 The RunUA.py interface, material sets and bead sets

To avoid the need to manually generate a configuration file each time, it's strongly recommended to use the RunUA.py interface where possible. This script takes a few common options and builds an appropriate configuration file (including calculation of appropriate Debye lengths), then automatically calls UA to run the adsorption energy calculation.

This script requires that both the material and set of AA beads are predefined (so that it can populate the configuration file with entries for both). The material dataset is read from the "MaterialSet.csv" and "surface-pmfp/MaterialSetPMFP.csv" to differentiate between manually constructed materials and the automatically generated set of materials generated using the PMFPredictor software package. In general, as an end-user only MaterialSet.csv will need editing. This file consists of a set of comma-separated lines of the form

```
materialname,pmf-folder,hamaker-file,shape-ID
```

e.g.

```
anatase100,surface/TiO2-ana-100,hamaker/TiO2_Anatase.dat,1
```

which registers a material with the name "anatase100", PMFs stored in "surface/TiO2-ana-100", a Hamaker file at "hamaker/TiO2_Anatase.dat" and a shape ID of "1" (meaning the PMFs will be mapped from an infinite plane to a sphere). Valid shape IDs are: 1 = plane-to-sphere 2 = plane-to-cylinder 3 = plane-to-cube 4 = 1.5 nm diameter cylinder to tube 5 = 1.5 nm diameter cylinder to cylinder where "tubes" are hollow and "cylinders" are solid.

The RunUA.py interface searches this dataset for a material with the name in the first column, then assigns Hamaker and PMF directories based on the remaining entries for this material. This script also automatically populates the amino-acids, amino-acid-charges, amino-acid-radii entries based on the chosen bead configuration, by default, "beadsets/StandardAABeadSet.csv", which looks like this:

```
#List of beads to include in UA autogenerated config files, providing the three-letter code, c
#BeadID,Charge[e],Radius[nm]
ALA,0.0,0.323
ARG,1.0,0.429
ASN,0.0,0.362
ASP,-1.0,0.356
CYS,0.0,0.352
GLN,0.0,0.386
```

Other files can be chosen to more easily vary bead parameters - this is useful if you have some beads defined only for certain surfaces or disagree with the radii assigned.

2.9 Extending UA

The golden rule: document everything you add! For beads include a SMILES code, for surfaces ideally provide a link to the input structure and details about metadynamics settings. Storage is cheap and metadata is invaluable, especially when it comes to trying to mix results from multiple groups.

2.9.1 New materials

Computing PMFs and Hamaker constants for a brand-new material is outside the scope of this guide, so here we simply assume you have a set of short-range potentials, plus Hamaker constants for this material with each of the 20 AAs (or a willingness to use Null.dat for these Hamakers). Make a folder for your material in the surface folder with a short but descriptive name, e.g. "Au-100-FCC" to denote gold, miller index 100, FCC crystal structure. Place all the computed PMFs in this folder with names ALA.dat, ARG.dat etc (one per bead). Add a file with the computed Hamaker constants and ideally the same or at least a similar name. For bonus points, update MaterialSets.csv so the material works with the wrapper scripts and also put a readme file in the folder describing what exactly the material represents, what forcefield was used, metadynamics settings, the solvent/pH/ionic strength, definition used for SSD, and convention used for amino acids (full vs side chain, convention for glycine and proline, histidine variants, any charge variants, etc).

2.9.2 New adsorbate beads

UA will throw an exception if it tries to load a bead for a material without that bead defined, but otherwise as long as you have the XXX.dat file in the surface folder and an entry in the Hamaker file for a material, this should go ok and you should be able to include new beads just by including these in the configuration file with their name, radius and charge. Remember to only use three letters for the ID tag and to make this match what you use in the PDB input file. To save headaches, if you're manually adding beads try to avoid starting the tag with an X (Y is probably also a bad idea) and make sure you remember what code is what bead. It's probably a good idea to make a .pdb file using a template similar to the one shown in Section 2.11 as this provides a record of the chosen tag, SMILES code, full molecule name and gives you something you can immediately use to test the molecule in UA.

2.10 Multicomponent mode

For a reasonably isotropic NP the above operation mode suffices, with core-shell NPs approximated via custom materials with one set of surface potentials and a different set of Hamaker potentials. In general, though, a more flexible approach may be needed. This is achieved through adding an extra line to the configuration file:

```
np-target mynpfolder
```

If this line exists, UA no longer automatically generates NPs of the specified material/radius/shape. Instead, it reads in structures from this folder with each file corresponding to an NP-complex: a structure built from composite NP beads, analogously to how a protein is built from AA beads. This structure is constructed using NPFile.h which expects input of the form:

i.e. a comma separated string for each NP component, with every component on a new line. The columns are taken to be:

- x : x-coord of centre of NP component in nm

- y : y-coord of centre of NP component in nm
- z : z-coord of centre of NP component in nm
- R : Radius of NP component
- Zeta: electrostatic surface potential for the component (probably mV)
- Core scale: prefactor applied to Hamaker potentials
- Surface factor: prefactor applied to surface potentials
- Shape: Shape ID. Shapes other than spheres are not very well tested and note that cylinders are always aligned to the x-axis.
- Hamaker file: Path to the Hamaker file for that component
- Surface set: Path to the directory of surface potentials
- PMF cutoff: LJ cutoff for this component (to allow combining multiple methodologies)
- Correction type: Manually change the surface correction applied to the surface potentials for this bead. This can probably be set to the same as the shape, except for small beads such as in polymer chains where it should be set to 0 because these don't need to be corrected as one bead = one molecule.

As an example:

```
#x,y,z,R,zeta,corescale,surfacescale,shape,Hamaker,PMFSet,PMFCutoff,Correction
0,0,0,5.0,0,1,0,1,hamaker/Metal.dat,surface/Au/FCC/100/sca,1.2,1
1.085,4.159,2.554,0.5249,0,0,1,1,hamaker/PseudoPEG.dat,surface/PEG,1.2,0
1.137,4.536,2.702,0.525,0,0,1,1,hamaker/PseudoPEG.dat,surface/PEG,1.2,0
1.229,4.674,3.077,0.525,0,0,1,1,hamaker/PseudoPEG.dat,surface/PEG,1.2,0
1.028,4.944,3.286,0.525,0,0,1,1,hamaker/PseudoPEG.dat,surface/PEG,1.2,0
1.249,5.279,3.389,0.525,0,0,1,1,hamaker/PseudoPEG.dat,surface/PEG,1.2,0
```

will produce a 5nm gold NP decorated with a few PEG beads. Note the PMF cutoff is set to 1.2 nm because these potentials were computed using CHARMM and the PEGs use a correction type 0. The Hamaker terms for these beads can generally be set to 0, given their small size.

The scaling factors can be arbitrary real values, with a typical use being the construction of a hollow NP by subtracting a small sphere from a large sphere, or a diffuse region with lower density. For example:

```
#x,y,z,R,zeta,corescale,surfacescale,shape,Hamaker,PMFSet,PMFCutoff,Correction
0,0,0,5.0,0,1,1,1,hamaker/TiO2_Anatase.dat,surface/anatase-101,1.0,1
0,0,0,5.0,0,-1,-1,1,hamaker/Metal.dat,surface/Au/FCC/100/sca,1.2,1
0,0,0,6.0,0,1,1,1,hamaker/Metal.dat,surface/Au/FCC/100/sca,1.2,1
```

can be used to produce a 5nm anatase NP core with a 1nm coating of gold (i.e., a 6nm gold particle with scaling factors $+1$ and a 5nm gold particle with scaling factors -1. It's also possible to pre-define bead types and use these as a short-hand (new feature as of September 2023)

```
#bead type definitions:
#TYPE,radius,zeta,corescale,surfacescale,shape,Hamaker,PMFSet,PMFCutoff,Correction
TYPE,5.0,0,1,1,1,hamaker/TiO2_Anatase.dat,surface/anatase-101,1.0,1
#bead instance definitions
#type-index (starting from zero and corresponding to the above beads),x,y,z
0,0,0,0
0,5,0,0
```

would produce two beads of type 0 at $(0,0,0)$ and $(5,0,0)$, where type 0 has been pre-defined to be an anatase bead of radius 5nm and zero zeta potential.

Complexes can be built using the NPDesigner tool, the GenerateNP.py script (check the NPGenerator folder) or other such tools, or through manual construction (not recommended, unless you have a lot of spare time).

Caveats:

1) The surface of a complex NP is not well defined. Interpret binding energies with caution and try varying the bounding radius (that is, the region of space over which UA scans). By default, UA scans over an interval from an inner radius $R_{in} = \max(R_i)$, that is, the maximum radius of all existing NP beads centered at zero, to an outer radius $R_{out} = 2.0 + \max(|r_i| + R_i)$, where $|r_i|$ is the distance of the NP bead from $(0,0,0)$ - this is designed so that it captures a brush region if it exists and stops at the outer solid shell. The inner radius can be overridden by the config file if needed with the "bounding-radius" option.

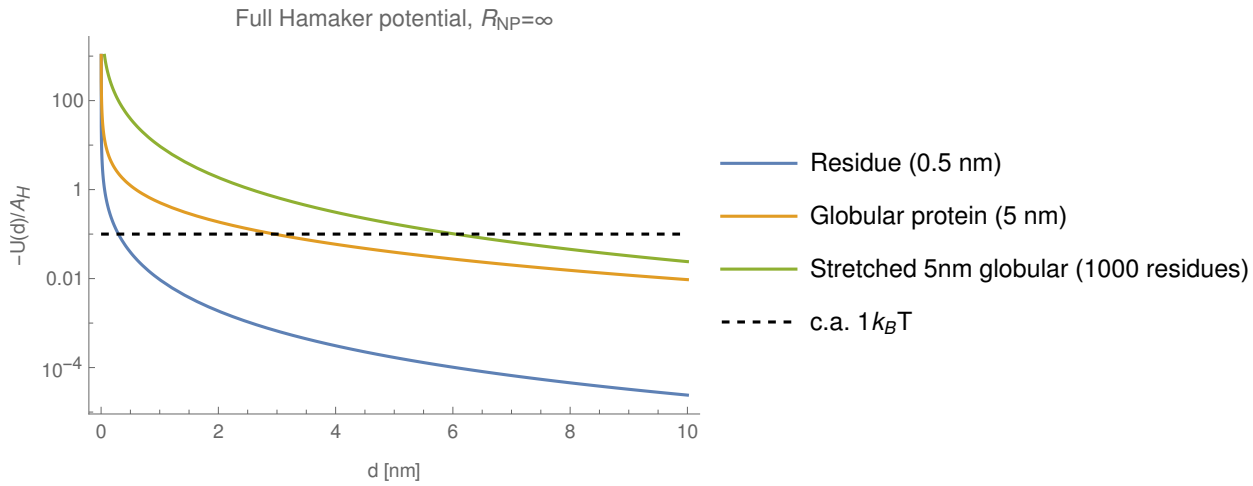

Figure 2.2: Plots of the total Hamaker potential between an NP of infinite radius and three cases: single residue ($R = 0.5$ nm), globular protein ($R = 5$ nm) and a "stretched globule" of 1000 residues of 0.5 nm radius each, producing the same total volume as the 5 nm protein, but all in the same plane. The dashed line indicates the point at which the potential is equal to $1k_BT$ for a typical Hamaker constant of c.a. $10k_BT$.

2) In the default mode, the potentials are generated for single bead AAs at the north pole of the NP-complex and summation is done over this. In other words, if we have a bead ALA, then UA generates a potential for ALA of the form

$$U_{ALA}(z) = \sum_{NPs} U_{NP-ALA}(\sqrt{x_{NP}^2 + y_{NP}^2 + (z - z_{NP})^2}) \tag{2.7}$$

which is valid for isotropic shells, but for bead-brush models this can lead to unexpected behaviour since it does not adequately model the interaction between off-axis NP/AA beads. This means that if there is a large enough gap for a single bead to sneak through, the entire protein will behave as if it can sneak into this space and dramatically overestimate the achievable binding energy. A hard-sphere repulsion can be switched on to reduce this effect, or you can disable the presummation of NPs by adding the switch "disable-np-summation" to your UA config file. With this mode enabled, the NP-AA potential is now calculated explicitly at each step of the simulation by summing over all AA and NP beads using their actual physical locations:

$$U_{NP-prot}(z) = \sum_i \sum_j U_{i,j}(r_i - r_j) \tag{2.8}$$

This will significantly increase the time taken for UA to run, it is highly recommended that you optimise your NP to eliminate trivial beads before enabling this option. However, this gives a much more realistic model of the interaction between complex NPs and a protein.

Polymeric NPs (those for which the PMFs represent a small repeat unit rather than an entire slab) can be built using a raspberry model, check GenerateNP.py for examples, or as beads decorating the surface of a homogenous inner bead. For reasons of speed it's best to replace small inner beads by larger ones if you can, especially if they're at a distance of > 1 nm from the surface of the NP (and so only interact through the Hamaker potential) - summation of the potentials over all beads in an NP can be very slow, especially for larger radii. The Hamaker potential is generally quite small (see Figure. 2.2 with $r_e = 0$ and Figure 2.10) - a typical Hamaker constant is on the order of $10k_BT$ at most, so for an infinite large NP bead at a distance of 1.5 nm from a residue bead (typical radius 0.5 nm) the contribution is on the order of $0.05k_BT$ per residue if the entire space below this distance is filled with NP material. From Figure 2.10 it can be seen that even if 1000 residues are laid out in a sheet and allowed to interact with an infinite mass of NP, the interaction is less than $1k_BT$ after a distance of 7 nm or so. Consequently, all beads with a closest-approach distance greater than 10 nm from the "top" of the NP (i.e., the plane defined by $z = R_{in}$ for a complex centered at $(0, 0, 0)$) are essentially negligible, since no matter how many there are the total contribution summed over all these beads is almost always less than $1k_BT$. Thus, if your NP is taking too long to run, you can speed things up by removing any bead for which $z + R_{in} - R_i > 10$ - even if you remove all such beads the interaction energy will change by far less than $1k_BT$.

2.11 PMFP

If you've looked at the material sets (or read this guide) you might have noticed some files are tagged with "pmfp" or "PMFP". This refers to surface and Hamaker data predicted using the PMFPredictor repository, which uses machine learning methods to significantly increase the amount of materials and small molecules available for input to UA. A script in that repository produces bindings for UA to make use of the predictions, but as they are less rigorously tested and to distinguish them from metadynamics results they are assigned a different namespace. In brief, the PMFP datasets include a "-pmfp" tag in material names where appropriate. The generated beads are assigned an auto-generated 3 letter code starting with an X (unless they correspond to a standard bead, in which they use the same code). This results in a folder

called “hamaker-pmfp” containing Hamaker files, “surface-pmfp” containing subdirectories with individual surfaces, a folder “pmfp-beadsetdef” containing a bead definition file “PMFP-BeadSet.csv” (three-letter code, radius, charge) and “PMFP-BeadNames.txt” which contains a lookup table for three-letter codes to the full name used in PMFPredictor. A folder “pmfp-singlebead-pdbs” is also supplied containing example PDBs for all the small molecules produced using PMFP, e.g.:

```
HEADER CAFF-AC                                          23-Mar-23       XX4D
TITLE    CAFF-AC to bead X4D
REMARK SMILES: Cn1cnc2n(C)c(=O)n(C)c(=O)c12
REMARK Generated: 2023-03-23 10:03:05
ATOM      1  CA  X4D A   1       0.000    0.000    0.000  1.00  0.00
END
```

which may be used to generate binding energies for small molecules and contains sufficient metadata to identify the molecule used.

2.12 Source files

In this section, the individual source files are discussed in more detail - if you need to modify UA or want to be sure how exactly something works, this is the place to start.

Parsers:

- CommandLineParser.h - read command line options
- ConfigFileReader.h - read config files
- Config.h - process config files
- HamakerConstants.h - read Hamaker files
- StringFormat.h - string helper functions
- PDBFile.h - read PDB files
- TargetList.h - find PDB files
- NPFile.h - read NP input files
- NPTargetList.h - find NP files

Potentials:

- CubeESPotential.h - define the DH-like potential for cubes
- CubePotential.h - define the Hamaker-like potential for cubes (centre of planar face)
- CylinderPotential.h - define the Hamaker-like potential for infinite cylinders
- TubePotential.h - define the Hamaker-like potential for infinite hollow cylinders
- Surface.h - read PMFs and correct to target NP
- Potentials.h - Combine all the potentials, build tables for each AA on the given NP, summing over NPs as needed.

The main core of the program is contained in main.cpp, which handles the overall integration of potentials and orientational sampling. There are some inconsistencies in variable names, in particular: cutoff may mean either an LJ cutoff or the final point saved for a potential and “SSD” may mean surface-surface distance or surface-center distance depending on context. An attempt to standardise or at least comment on these has been made.

2.13 Post-processing and extra tools

UnitedAtom output by itself typically requires some further processing to make it useful, since the raw data tends to be too much to directly analyse.

Note: in all that follows, be very careful to remember that the UA file format stores LEFT HAND EDGES for angles in units degrees. You therefore need to shift and convert to radians, e.g.

```
theta += 2.5
theta = theta*pi/180.0
```

early on or use

```
sin((theta+2.5)*pi/180.0)
```

in all appropriate places.

A good first step is plotting a heatmap from the tables of binding energies. This can be done using the script in "tools/plotmap" or by importing the data into e.g. Mathematica and plotting contour plots (this can be better if you want a more smooth plot due to how tools/plotmap stores data, be aware that you can't just smooth it out using a Gaussian blur)

The next most common form of post-processing tends to be the evaluation of average adsorption energies as some form of weighted average,

$$\langle E \rangle = \frac{\sum_i w_i E_i}{\sum_i w_i} \tag{2.9}$$

where w_i has two common forms. For the "simple average", $w_i = \sin\theta_i$ while for the Boltzmann average, $w_i = \sin\theta_i e^{-E_i/kbT}$. The former weights all states equally, e.g. for early corona formation, while the latter weights more strongly adsorbing states more strongly and is closer to the thermodynamic equilibrium for an NP with infinite binding locations such that competition between orientations can be neglected.

In certain circumstances, it may be necessary to average across multiple .uam files. Be careful to identify what other weights need to be assigned and include them in w_i if this is physically required to make a sensible average, and don't just assume that you can "average the averages" afterwards - this won't always give physically meaningful results and it won't always be meaningful to take an average at all. For example, if you have an NP which is half strongly-binding on side a and half non-binding on side b, e.g. $E_a = -50, E_b = 0$, then the naive average is -25 which would suggest the NP is reasonably strongly binding everywhere. In this kind of complex case it can be better to average in a more sophisticated way depending on what exactly you want the binding affinity to represent. If, for example, you only want to think about a single instance of a single protein and you just want to know if it binds somewhere to the surface of an NP with j surface types of proportions α_j at all, include the surface weights as an extra term e.g.:

$$\langle E \rangle = \frac{\sum_i \sum_j \alpha_j \sin\theta_i e^{-E_{ij}/kbT} E_i}{\sum_i \sum_j \alpha_j \sin\theta_i e^{-E_{ij}/kbT}} \tag{2.10}$$

noting that its included in the denominator too and where E_{ij} is the adsorption energy of orientation i to surface j.

Conversely, if you want something more representative of an experiment where you have a bunch of copies of a protein floating around and want to know how many are adsorbing, it'd be more appropriate to simulate the corona formation for each type of surface separately to get the total number and surface coverage to each type of NP-surface , $\langle N \rangle = \sum_j \alpha_j N_j$, $\langle s \rangle = \sum_j \alpha_j s_j$. With those and the input concentration you can estimate an effective binding energy defined such that an isotropic protein binding to an isotropic NP would produce the same numbers and coverages. For a spherical NP, the effective expression is given by:

$$\langle E_{ads} \rangle = -k_B T ln \left(\frac{s}{1-s} \exp[3\frac{s}{1-s} + \frac{s^2}{(1-s)^2}] \right) / (C_{tot}) \tag{2.11}$$

where C_{tot} is the input concentration summed over all orientations (which for simplicity should be the same in each input simulation). Even for a single isotropic NP surface this approach can be useful to produce a more realistic orientational average as a better match to experiment - remember that since experiments extract a binding energy from observed bound-to-unbound ratios, their binding energies implicitly include the effects of surface crowding unless they're at extremely low concentration.

2.14 Common issues and debugging tips

UA is not foolproof, do not blindly accept your results as fact. Always test surfaces (especially new ones) against simple test molecules and conversely test molecules (especially AlphaFold/I-TASSER proteins) against the well-characterised surfaces to make sure these results are physically meaningful. In particular make sure you understand how orientations are defined with respect to the NP surface. It can be really useful to define simple dipole-like molecules with one strongly adsorbing end and one weakly adsorbing end as a simple test to make sure any post-processing script is giving sensible answers - for example, if you set the zeta potential to a large positive value (e.g. on the order volts) then a protein consisting of one LYS and one GLU should adsorb mostly strongly when GLU is on the surface.

A few rules of thumb:

- The most important factor by far is the number of residues directly in contact with the NP (distances < 0.6 nm or so.
- The integration free energy is in practice essentially a soft minimum, especially for strongly binding proteins - remember that a difference of $1k_BT$ in an exponential is a factor of 2.7 in relative probabilities, so if there's a strongly binding configuration this typically outweights most other factors.
- The short-range potential is typically the most important contribution, so you can usually get a feel for how strongly a protein will bind by looking at the most strongly binding PMFs and multiplying the depth of these by the number of protein-NP contacts.

- For typical electrostatic potentials of ±50 mV the electrostatic term typically only contributes a few kbT at most per charged residue in contact with the NP.
- The long range (Hamaker) potential is generally the least significant. This is because it only takes into account parts of the NP/AA which are at a distance greater than 1 nm from each other and by this point the $1/r^6$ potential has decayed to essentially zero.

Some common problems:

- Binding energy is (almost) zero everywhere (0.06 seems typical) with (effectively) zero standard deviation. This means that the potential for your molecule is zero everywhere and the small residual is due to rounding issues in the numerical integration. Check the potentials and the PDB structures - if the columns aren't aligned correctly then the occupancy column may be reading as 0 for all residues.
- Boost is complaining about file copying: this means you've renabled the code for saving config files (generally a good idea!) but you're using an older version of boost (generally a less good idea!) and one of their arbitrary breaking changes to do with how it handles overwriting files is breaking. Uncomment the alternative line or use the RunUA.py/PrepareKMCInput.py interfaces to save the config file automaticallly.
- The present version of UA won't automatically overwrite result files, so make sure your workflows save results to new folders as appropriate.

Chapter 3

CoronaKMC

3.1 Introduction

CoronaKMC runs a hard-sphere simulation of the (usually reversible) adsorption of spherical particles onto a substrate based on input concentrations, particle sizes and per-site rate constants. The most typical use case is for the adsorption of proteins onto nanoparticle and this documentation is written with that in mind.

3.2 How it works

CoronaKMC.py is the main script of interest here. It takes in a set of options and then performs the following

1. Prepare the list of all potential adsorbates
2. Begin the main loop
3. Use the Kinetic Monte Carlo algorithm to select events. Events can be adsorption of a protein from solution or desorption of a bound protein.
4. Continue until the preset time (simulation time, not wall time) is reached
5. Write out coverages, numbers, surface coordinates

3.3 Input

CoronaKMC is controlled almost entirely through command-line arguments, unless you need to make very specific tweaks. These arguments can be viewed by running "python3 CoronaKMC.py –help" and a summary of their meaning is given below:

- -r or –radius - the radius of the NP in nm.
- -p or –proteins - The input file defining all adsorbates, see later
- -s or –shape - shape of the NP, 1 = sphere, 2 = cylinder, 3 = plane, 4 = truncated sphere (i.e. a sphere but limited to only the upper part of the NP to save time)
- -t or –time - Number of hours of simulated time (internal time, not wall time). Generally only a few seconds are needed, this is in hours for historical reasons.
- –timedelta - Time step in seconds between showing updates, this will likely need some tweaking so you get updates reasonably frequently without being too quick to follow.
- –demo - enables the demonstration mode (a live updating graph and corona figure)
- -P or –projectname - name for the output folder used.
- -x or –npconc - concentration of NPs, leave zero for vanishing concentration. This is used to deplete proteins during simulation and so is appropriate for in vitro studies if the serum concentration is low. If proteins can be replenished from outside the NP concentration should be set to zero.
- -n or –numnp - number of NPs to run simultaneously, defaults to 1 and should be left at 1 unless you want to simulate protein depletion using the –npconc option (in which case it should be increased to reduce fluctuations). If you have npconc = 0 then just run multiple KMC runs at the same time instead for speed reasons.
- -D or –displace - activate the "displacement mode" if non-zero, where a protein can displace bound proteins if this is energetically favourable. See Section 3.7 for more details.

- -A or –accelerate - activate Dybeck style acceleration if set non-zero, with 1 giving the classic mode and 2 giving the mode with freezing of rate constants. This may need some tweaking to the code to give sensible results.
- -f or –fileid - optional string to append to output filenames to allow for multiple runs with similar configurations without overwriting each other (e.g. to get better averages).
- –steady : Enables steady-state mode. This rescales rate constants such that the dynamical evolution is meaningless, but the steady state should be reached in significantly fewer steps. Displacement mode is activated for a short period and is then disabled, since the rate-rescaling isn't compatible with a displacement mode steady state.

Some extra optional ones are listed below - these tend to be for debugging or more specialised cases.

- -l or –loadfile - load in a .kmc file from a previous run to precoat the NP in these proteins. Once these desorb they're lost forever. No sanity checks are made to make sure the NP between the two runs match!
- -d or –diffuse - enable surface diffusion of bound proteins, currently only possible for spherical NPs. This is extremely slow and largely irrelevant. Activate only if needed.
- -c or –coarse - this is mostly unneeded in the PrepareKMCInput.py pipeline but in effect allows for an orientational-averaging of all adsorbates to get an effective binding energy and area. The envisaged usage is that this lets you run a simulation of all orientations of a specific protein competing for binding and can back-calculate an effective binding energy and area which would produce the same number and surface coverage.
- -m or –meanfield - switches binding mode to mean-field approximation, probably not needed.
- -H or–hardsphere - switch collision detection to pure hard-sphere rather than hard-sphere-projection.
- R or –runningfile - saves extra data if enabled
- -b or –boundary - defaults to 1 to enable periodic boundary conditions for planar and cylindrical NPs. Set to 0 to disable these.

3.4 Adsorbate input file

In general the most important input is the list of potential adsorbates, which is a file with the following structure

```
CAFF–AC–P1:1.5707 -2.5 0.0011 0.429782 40223088.95 92160.946 -6.07866 0.4928
DGL–AC–P2:1.57079 -2.5 0.126 0.40580 37973099.80 410581.0240 -4.52706 0.4432
THRSCA–AC–P3:1.57079 -2.5 1.95 0.2893 27060588.72 15066871.28 -0.58558 0.2352
CASEIN–P4:2.5 -2.5 8.258e-09 2.64137 254665682.08 78662113.41 -1.17479 9.68
CASEIN–P4:2.5 -7.5 8.258e-09 2.65634 256171091.41 77399130.53 -1.19687 9.75
CASEIN–P4:2.5 -12.5 8.258e-09 2.68301 258856223.61 73540978.86 -1.25843 9.88
CASEIN–P4:2.5 -17.5 8.258e-09 2.72020 262602959.93 70165254.15 -1.31979 10.07
```

Note that unlike almost every other input file, these are space-delimited and not comma-delimited - this should probably be updated at some point. The columns are:

- Protein ID
- Concentration
- Effective radius
- KAds (per-site)
- KDesorb
- EAds = log(kads/kdesorb)
- Effective binding area

The protein ID has the format "name:theta-phi", with this convention allowing for different orientations of the same protein to be handled separately internally but summed together for output. That is, in the above example, the simulation prints out counts of adsorbed CAFF-AC, DGL-AC, THRSCA-AC and CASEIN, but CASEIN has every orientation treated separately and the simulation stores data about which orientations are actually bound.

3.5 Output

The most immediate output is seen when running the simulation

abc

where the first column is the time, the next N columns are N adsorbates (each summed over orientations if necessary), followed by a total count and the total surface coverage.

A number of files are created in the target output folder, either during the simulation (Running) or at the end of the run (Final):

- (Running) kmc_running_INPUTFILENAME_X.txt - protein counts
- (Running) kmc_running_INPUTFILENAME_X_coverage.txt - surface coverages
- (Final) FILENAME.kmc - adsorbed proteins at the end of the simulation
- (Final) FILENAME_finalcoords_X - adsorbed proteins, real-space coordinates
- (Final) kmc_FILENAME_X.txt - protein counts, saved at end

If the Boolean flag "doMovie" in the script is set to True then an extra folder "movie" will be created. This will be empty unless the –demo command line option is also enabled, in which case it will contain the figures generated by demonstration mode.

The .kmc file can be used as a precoating file for further simulations if needed via the -l option. This allows the simulation of a corona in one environment followed by transfer to another. This file can also be used with the tools/-CoronaKMCtoVMD.py script to generate a simple .tcl script which can be run in VMD for visualisation.

3.6 Connection to UA

A frequent use case is the adsorption of proteins to NPs, with the protein-NP adsorption affinity characterised using UA. This requires a method to convert the static binding energies output from UA to dynamic rate constants, together with estimations of the binding area and ideally accounting for multiple protein orientations. The most straightforward way to achieve this is to use the BuildCoronaParams.py script, which takes the following input:

- -f or –folder - folder containing UA output .uam files
- -r or –radius - target radius
- -z or –zeta - target zeta potential
- -s or –shape - target NP shape, 1 = sphere , 2 = cylinder , 3 = plane - this should match what was used for UA. Plane can be used for large spherical NPs to reduce simulation time, cylinder covers UA types 2,4 and 5 (cylinders and tubes)
- -p or –proteins - path to a file (protein serum file) containing pairs in the form "proteinID,proteinconcentration" (see below). Only molecules in this file will be added to the output file and only if their structure and .uam file can be located.
- -c or –coordfolder - Path to the directory containing all .pdb files, must contain a match for each molecule in the protein serum file
- -b or –beadset - Bead file - generally the default is fine, unless you have special beads like sugars or other small molecules
- -o or –outputname - File to which results should be saved. This can then be used as input to CoronaKMC.

The -r and -z arguments are used to locate the correct .uam files for a given protein in case there are multiple matches in a given folder. The -p option is used to determine the total concentration for all orientations of a given molecule and -c used to find the correct structure for the molecule.

This script performs the following steps:

1. Load in the set of molecules to include (serum file)
2. For each molecule in the serum file load in the .pdb and .uam files and compute for all orientations:
   (a) Projected area via convex hull
   (b) Effective concentration (input concentration multiplied by $\sin\theta$, normalised to sum to input concentration)
   (c) Adsorption rate constant (kinetic collision theory based on sphere-sphere collisions, normalised to give a per-site constant)

(d) Desorption rate constant (from adsorption rate and equilibrium constant based on UA adsorption energy)

3. Write out all results to a file which can be used as input to CoronaKMC.

If there is only a single atom present in the PDB file then only one orientation is used to simplify the output. The projected area is calculated from the input co-ordinates and the beadset file to assign a radius to each bead, then projecting these onto the surface of the NP, extracting the area of the convex hull of the projection. An equivalent radius is then found for a sphere which would project the same area. Note that the default operation is to assign a bead radius to all the atoms in an input structure, not only CA ones, but in most cases the overestimation is small because the size of a bead relative to the size of a protein is quite small. If this is really an issue the code can be altered to only use CA locations when placing beads or input structures can be pre-filtered to include only CA beads.

The adsorption and desorption rate constants are calculated using the following procedure. Note that since the collision rate is undefined for planar and cylindrical surfaces due to their infinite surface area, we approximate these using the expression for a sphere. The collision rate between two spheres in solution is given using Smoluchowski theroy,

$$k_s = 4\pi * (R_{NP} + R_P)\, N_a \frac{k_B T}{6\pi\eta} \left[\frac{1}{R_{NP}} + \frac{1}{R_p}\right] \tag{3.1}$$

where η is the viscosity of the medium and N_a is Avogadro's constant. The per-site collision rate is then given by,

$$k_{coll} = \frac{1000 k_s A_p}{4\pi R_{NP}^2} \tag{3.2}$$

where A_p is the projected area and the factor of 1000 arises from the conversion from m^3 to L. This value is recorded as the adsorption rate constant under the assumption that adsorption occurs with unit probability following a collision. We then obtain the desorption rate constant by requiring that

$$\frac{k_a}{k_d} = e^{-E_{ads}/k_B T} \tag{3.3}$$

such that if collisions occur with unit probability the desorption rate is given by,

$$k_d = e^{E_{ads}/k_B T} k_{coll} \tag{3.4}$$

i.e. a strongly negative adsorption energy leads to a very low desorption rate.

3.7 Displacement

In the standard form of CoronaKMC a single weakly-binding particle can block the adsorption of a large protein even if this would bind much more strongly. This can lead to the steady-state corona consisting almost entirely of these small molecules because large proteins never get a chance to bind. To counteract this, the -D switch enables a mode in which an incoming particle can displace a set of pre-bound proteins. The overlap detection works as before, but now instead of outright rejecting the incoming adsorbate the total change in binding energy ΔE is computed, $\Delta E = E_{new} - \sum_j E_{old,j}$ where j is all overlapping adsorbates. The probability for replacement is then given by a Boltzmann probability,

$$p = \frac{e^{-\Delta E/k_B T}}{1 + e^{-\Delta E/k_B T}} \tag{3.5}$$

i.e. if the two states have nearly the same energy there's a fifty-fifty probability of selecting either rather than forcing the new state as the classic Metropolis algorithm would do. For consistency in the limit of a single small adsorbate with $E_{old,j} = 0$, we apply this same acceptance probability even if the NP is otherwise bare (since this is implicitly covered with water and adsorption energies are defined relative to this). This results in an effective decrease in the adsorption rate constant,

$$k_a^* = \frac{e^{-\Delta E/k_B T}}{1 + e^{-\Delta E/k_B T}} k_{coll} \tag{3.6}$$

and so to maintain the same equilibrium constant k_d must be scaled by this same factor (this is done automatically by CoronaKMC if displacement is enabled),

$$k_d^* = \frac{e^{-\Delta E/k_B T}}{1 + e^{-\Delta E/k_B T}} k_d. \tag{3.7}$$

Note that if you're using BuildCoronaParams output, k_d^* is then given by,

$$k_d^* = \frac{1}{1 + e^{-\Delta E/k_B T}} k_{coll} \tag{3.8}$$

c.f. Eq.(3.4). The two values are essentially equivalent if $E_{ads} < -2k_B T$, but this produces a change in the rates in the limit $E_{ads} \to \infty$: $k_d \to \infty$ while $k_d^* \to k_{coll}$, but because the equilibrium constant is fixed the steady-state behaviour should be the same.

3.8 Tips and tricks

If you're trying to get a match to experimental, match the absolute experimental concentrations as best as you can – it can be shown that an overall dilution factor changes the steady-state corona composition and high concentrations tend to lead to an increase in more weakly binding proteins (see Rouse and Lobaskin, Biophys J, 2021 for details).

Appendix A

Rotations, random rotations, and sphere points

It is very very highly recommend that you at least read the wikipedia page on rotation matrices (https://en.wikipedia.org/wiki/Rotation_matrix) to get a feel for some of the issues with rotation in 3D, which can be quite counterintuitive. Some common problems are:

1. Rotations around different axes do not commute - in other words, if you rotate by an angle ϕ around the z axis and then θ around the y axis, you get a different result than if you had done the rotation of θ around y first and then ϕ around z. This is very important for UA and will be proven later.

2. Following from the above point, even though the biomolecule is placed on the z-axis, the rotation around z done first does make a difference even for a spherical isotropic NP. A rotation around z AFTER the rotation around y wouldn't matter for that specific case, but does in general.

3. Generating a uniform random rotation is difficult and needs a specialised algorithm. No, you can't just generate three Euler angles in the interval $[0, 2\pi]$ and do x/y/z rotations, or z/y/z, or whatever. Try it and see where an initial point (0,0,1) gets mapped to and plot the density - you'll get polar bunching and/or gimbal lock.

4. Generating random points on the surface of a sphere also needs to be done carefully - naive approaches tend to put more density on the poles.

These points are discussed in the following sections.

A.1 The UA rotation convention

One of the most common pitfalls with UA is misunderstanding how the orientation of a given biomolecule is defined and making this consistent with other software, which leads to misinterpretation of results and people making significant errors later.

We begin by considering a set of N points, each indexed i and with Cartesian co-ordinates x_i, y_i, z_i. These points are assumed to belong to the CG beads in a biomolecule of interest and are defined in the molecule's own reference frame, i.e. read directly from an input file. First, we center the molecule by applying the following transformations to each co-ordinate:

$$x_i \to x_i - \frac{\sum_j x_j}{N} \tag{A.1}$$

and likewise for y, z. Note that this is similar to the centre of mass, but we treat all beads as having the same mass, and in UA this transformation is applied only to the CG beads obtained from the CA atoms in an input .pdb file, so for external files make sure the displacement is computed only using these.

Next, we consider an NP approaching along the vector defined by ϕ, θ defined by the (new) centre of the biomolecule at $(0, 0, 0)$ and employing the physics convention such that $(x, y, z) = (r\cos\phi\sin\theta, r\sin\phi\sin\theta, r\cos\theta)$, where $r > 0$ is the length of the vector. This fixes the orientation of the protein relative to the surface of the NP, up to a rotation around that vector which is unimportant for a spherical isotropic NP. Note that any given ϕ, θ will define a different approach for the NP and thus a different orientation of the protein. For practical purposes, it's useful to instead consider the NP to be fixed in its frame at $(0, 0, 0)$ and have the biomolecule approach along the z axis from $+z$ to $-z$, normal to the surface, while keeping the biomolecue in the same relative orientation as it was in the original frame.

To do this, we need to generate a rotation which maps the orientation vector $(\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$ to $(0, 0, -1)$. This can be achieved through a combination of two basic rotations, one around the y axis of an angle α:

$$R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \tag{A.2}$$

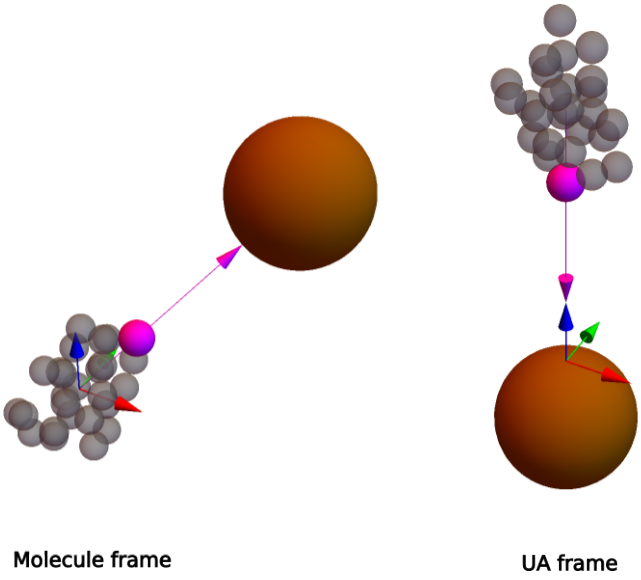

Figure A.1: Left: a biomolecule in its original frame read in from a co-ordinate file, with a vector $(\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$ and associated atom shown in magenta. Right: the molecule in the frame for UnitedAtom calculations, where the original vector has been mapped to $(0, 0, -1)$. The axes for the coordinate frame are shown by the origin symbol: x in red, y in green and z in blue.

and one around the z axis of an angle β:

$$R_z(\beta) = \begin{pmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.3}$$

It's useful to work backwards here: if we start with $(0, 0, -1)$ and want to map to our initial vector, a rotation around z won't get us anywhere so the first backwards rotation would have to be around y. Rotating around the same axis twice in a row is equivalent to rotation of a different angle around the same axis once, so the next should be around z. Thus, we write:

$$R_z(v) \cdot [R_y(u) \cdot (0, 0, -1)] = (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta) \tag{A.4}$$

where u, v are angles we need to find. Expanding out the left hand side and taking care with the order of matrix operations (remember: matrix operations are non-commutative), we find:

$$(-\cos v\sin u, -\sin u\sin v, -\cos u) = (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta) \tag{A.5}$$

This is already solvable by inspection: we can see that u is a function of θ alone and v is a function of ϕ alone. Taking care with the sin signs, we find that $u = \theta - \pi$ and $v = \phi$. We must now invert this procedure so instead of moving from $(0, 0, -1)$ to the target vector, we move from the target to $(0, 0, -1)$. The inverse of a rotation around an axis by u is a rotation around the same axis by $-u$. Thus, we can write:

$$R_y(u) \cdot (0, 0, -1) = R_z(-v) \cdot (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta) \tag{A.6}$$

and then,

$$(0, 0, -1) = R_y(-u) \cdot [R_z(-v) \cdot (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)] \tag{A.7}$$

From the definitions of u, v found above, we therefore have:

$$R_y(\pi - \theta) \cdot [R_z(-\phi) \cdot (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)] = (0, 0, -1) \tag{A.8}$$

Thus, if we apply a rotation of $-\phi$ around the z axis followed by $\pi - \theta$ around the y-axis to the biomolecule, we have mapped the configuration obtained with the initial co-ordinates and an NP along the vector $(\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$ to one where the same orientations are kept, but now the approach is along the z-axis and the NP is "below" the biomolecule. This procedure is shown in the top row of Figure A.1. We assume that the rotations are applied to the biomolecule in its original reference frame with the origin passing through the centre of the molecule, the origin symbol is displaced in these figures purely for clarity.

Note that the first rotation around z maps our target vector to be parallel to the x axis. If we apply these rotations in the wrong order (bottom row of Figure A.1) , we instead get a much different final outcome:

$$R_z(-\phi) \cdot [R_y(\pi - \theta) \cdot (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)] = \begin{pmatrix} \sin(\theta)\left(\cos^2(\phi)(-\cos(\theta)) + \cos(\phi)\cos(\theta) + \sin^2(\phi)\right) \\ \sin(\phi)\sin(\theta)(\cos(\phi)(\cos(\theta)+1) - \cos(\theta)) \\ -\cos(\phi)\sin^2(\theta) - \cos^2(\theta) \end{pmatrix} \tag{A.9}$$

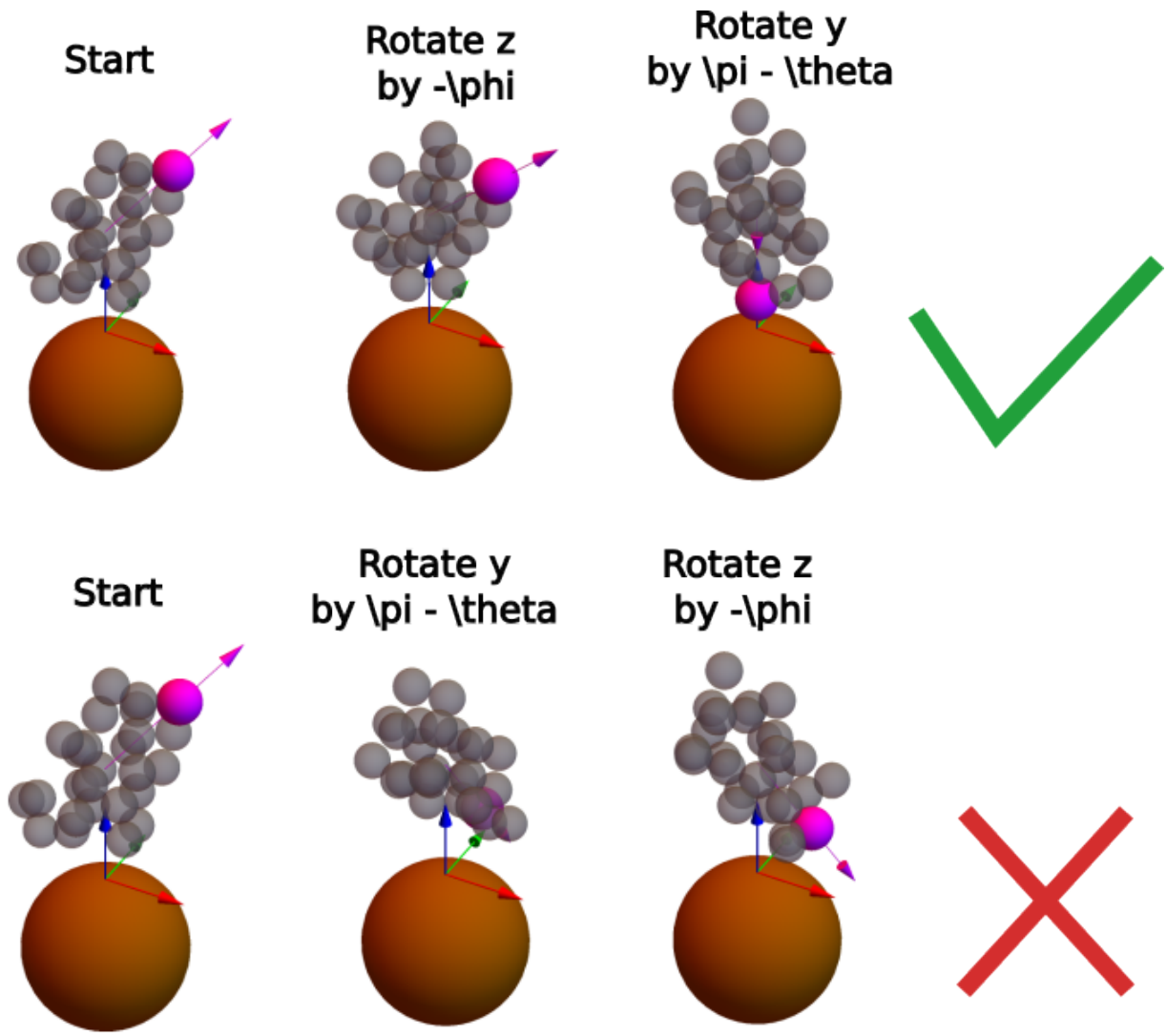

Figure A.2: The procedure used to rotate biomolecules for UnitedAtom based on their initial configuration, showing the two step process. The top row demonstrates the correct procedure, which maps an initial vector $(\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$ to $(0, 0, -1)$. The lower row demonstrates the results if the steps are carried out in the wrong order. Note that for clarity, we have displaced the symbol showing the origin from the centre of the biomolecule but that the rotation axes are understood to pass through the centre of the molecule.

which very clearly isn't the result we're looking for. This is demonstrated more in Figure A.1 and you can see this directly from the rotation matrices, for z then y we get:

$$R_y(\alpha) \cdot R_z(\beta) = \begin{pmatrix} \cos(\alpha)\cos(\beta) & -\cos(\alpha)\sin(\beta) & \sin(\alpha) \\ \sin(b) & \cos(\beta) & 0 \\ \sin(\alpha)(-\cos(\beta)) & \sin(\alpha)\sin(\beta) & \cos(\alpha) \end{pmatrix} \tag{A.10}$$

but for y then z it is:

$$R_z(\beta) \cdot R_y(\alpha) = \begin{pmatrix} \cos(\alpha)\cos(\beta) & -\sin(\beta) & \sin(\alpha)\cos(\beta) \\ \cos(\alpha)\sin(\beta) & \cos(\beta) & \sin(\alpha)\sin(\beta) \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \tag{A.11}$$

You can also get differently wrong results by forgetting the signs for the angles, using θ instead of $\pi - \theta$, trying to rotate around x, assuming that a rotation around z won't do anything ever, switching which axis has which angle, etc. An example is shown in Figure A.1, demonstrating the correct approach and the results obtained if the rotations are performed in the wrong order. All of these mistakes can lead to nonsense results, like binding energies of tens of kbT through a single point contact when each AA only adsorbs with an energy of a few kbT. It's always worth looking at the proposed orientation relative to the NP and counting the number of contacts - in most cases, the highest binding affinity is the one with a large surface-surface contact area.

The above generates the first two angles. For a spherical isotropic NP, as can be seen from the figures above, a further rotation around the z axis is possible but won't alter the overall binding energy. In general, however, it may be necessary to apply a final rotation around z of ω to correctly generate all possible conformations of the biomolecule relative to the NP.

As a final reminder, the orientation $(0, 0)$ only means "zero-rotation" in a frame of reference where the NP is placed directly **above** the biomolecule, i.e. at $(0, 0, z > \max(z_i))$. By convention, we usually take the NP to be at $(0, 0, 0)$ and the biomolecule placed above this, so an orientation $(0, 0)$ corresponds to a rotation of 0 around the z axis and 180 around the y axis.

A.2 Random rotation and point generation

From the previous section, it may be apparent that rotations in 3D are less trivial than they may appear. This is even more true for random rotations, which for UA are relevant when generating nanoparticles. Although it may seem straightforward to simply produce three random rotations in a row with uniformly distributed angles, this doesn't produce the desired result. A simple mechanism to generate the required rotation was proposed in [Arvo] (there are other methods, but this avoids the use of quaternions and only needs uniformly distributed random input). First, generate the matrix to rotate around the z-axis by a random amount $u_1 \in [0, 1]$:

$$R_1(u_1) = \begin{pmatrix} \cos(2\pi u_1) & -\sin(2\pi u_1) & 0 \\ \sin(2\pi u_1) & \cos(2\pi u_1) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.12}$$

Next, generate the "Householder matrix",

$$H = I - 2vv^T \tag{A.13}$$

where I is the three-by-three identity matrix,the vector v is given by,

$$v = (\cos(2\pi u_2)\sqrt{u_3}, \sin(2\pi u_2)\sqrt{u_3}, \sqrt{1 - x_3}) \tag{A.14}$$

and vv^T is the outer product of v with itself. The final rotation matrix is then given by $M = -HR$, which generates isotropic random rotations if u_1, u_2, u_3 are all uniformly distributed in $[0, 1]$.

Part of the reason for this complication is due to "bunching" of density at the poles. In short, if you generate a uniform grid of points ϕ, θ, the density in Cartesian coordinates is biased towards $x = 0, y = 0$. For generating random points on the surface, a specialised algorithm should be used. A simple one is generating ϕ uniform in $[0, 2\pi]$, generating a second variable u in $[0, 1]$ and calculating $\theta = acos(2u - 1)$.

If you want non-random points with a uniform distribution over the surface of the sphere, try the algorithm in Koay DOI: 10.1016/j.jocs.2010.12.003. This is implemented in NPDesigner (C++) for adding brush beads and likewise in NPGenerator (Python). Note that this algorithm requires the use of elliptic integrals which have different definitions between languages, e.g. in C++ the Boost library take k while Scipy for Python uses $m = k^2$ as the parameter. You may also have to make sure that the argument is real using the modulus transformations in DLMF.