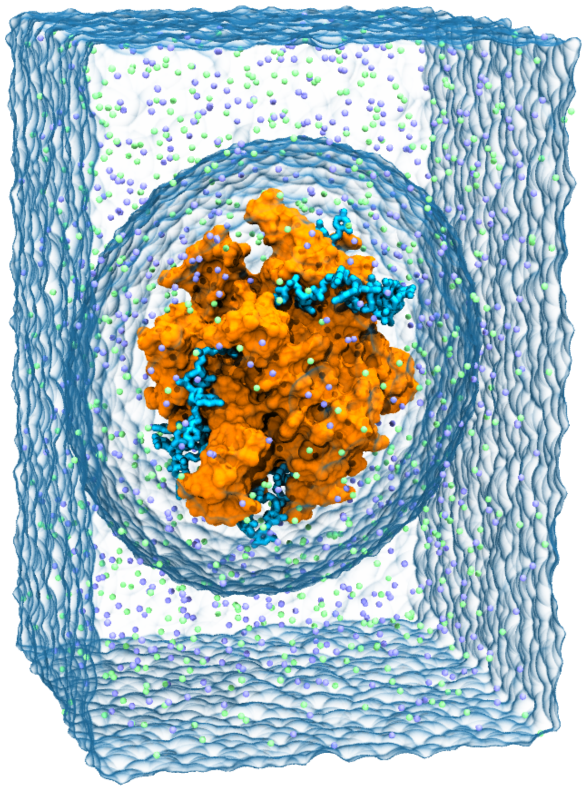
Aksimentiev Group
Department of Physics and
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign

Introduction to MD simulation of DNA–protein systems



Chris Maffeo
Rogan Carr
Aleksei Aksimentiev

Contents

1 Introduction

This is an advanced tutorial that will guide the reader through the processes of setting-up, performing and analyzing a molecular dynamics simulation of a DNA–protein containing system using (primarily) Tcl scripts. This can be a valuable approach because scripts provide an exact record of the process. Once written, a script can be copied and modified to avoid repeating work in similar projects.

We have provided complete versions of the scripts in the `complete` directory so you can check your work. We recommend that these are only used for reference after you've attempted to write the scripts yourself (even if you are transcribing more-or-less directly from the PDF); you will learn the most by making mistakes and taking the time to understand them!

The files associated with this tutorial can be downloaded at the following url: http://bionano.cpanel.engr.illinois.edu/tutorials/introduction-md-simulation-dna-protein-systems.

DNA systems

DNA is so famously known as the carrier of genetic information that the structural and dynamical aspects of the molecule are often neglected. However, most cellular processes that involve DNA cannot be understood without taking into account its physical properties and structure.

Single-stranded DNA (ssDNA) is a polymer composed of nucleotides. A nucleotide consists of one of four hydrophobic, ring-shaped bases (A, T, C or G) connected to a sugar ring, which is in turn bonded to a phosphate. The phosphate of one nucleotide can be connected to the sugar ring of another. When this process is repeated, ssDNA is formed.

If two strands have complementary sequences (A·T or C·G), they can anneal to form a double-stranded DNA (dsDNA) duplex stabilized by base-stacking and Watson-Crick hydrogen-bonding between the complementary bases. Canonical DNA (B-DNA) in electrolyte forms a right-handed double-helix. Traversing the duplex by one basepair corresponds to a rotation of about 34°. A DNA duplex—the smallest self-assembled unit of DNA—is used by the cell for packaging and protecting its genetic information. DNA-DNA and DNA-protein interactions can give rise to self-assembled structures; the DNA double-helix wraps twice around a histone to form the nucleosome, which in turn form aggregates that eventually form chromatin—the fiber that makes up the chromosome [1].

During DNA replication, the cell's machinery unravels these structures, forking dsDNA into a pair of single DNA strands at the last step. A protein called DNA polymerase moves along each unwound ssDNA strand to synthesize a complementary strand. After the DNA is unwound, but before the DNA polymerase arrives, single-stranded DNA binding protein (SSB) wraps up the ssDNA to prevent the strands from annealing, protect the nucleobases from chemical modifications and prevent the formation of hairpin structures in repetitive, self-complementary regions of DNA [2, 5].

Molecular Dynamics simulation

Small molecules that drive cellular processes can be studied using a variety of techniques. In experimental labs, optical traps can be used to apply and measure forces acting on single molecules. In the Aksimentiev group, models of single molecules can be manipulated in similar ways. We can apply and measure forces with a computational technique called Molecular Dynamics (MD) simulation.

In MD simulations, molecules are treated as collections of point particles which interact via a set of forces; Newton's equation ($\mathbf{F} = m\mathbf{a}$) is integrated to describe the temporal evolution of the system. MD simulations use a force field, which is a set of equations and parameters that together determine how any pair of point particles interact. The most popular force fields for MD simulation describe biomolecules as collections of atoms which are connected by harmonic bonds (two-body interactions), angles (three-body interactions) and dihedrals (four-body interactions) and interact through the Coulomb and van der Waals potentials. Given the positions and velocities for all the atoms in a system, NAMD (the MD package that we will be using) calculates new positions and velocities using the force on each atom with the equation in Figure 1.

Today, you will prepare a system for a steered molecular dynamics (SMD) simulation of ssDNA and SSB, running briefly to ensure that everything worked. Unfortunately, there is insufficient time to perform a long-timescale simulation, so a final trajectory of an equivalent simulation is provided. You will then perform simple analysis of the trajectory using VMD's Tcl interface.

You will probe the free energy landscape of DNA binding to SSB by pulling on DNA to force its dissociation from SSB. To be computationally economical, the DNA will be pulled along an unusual axis so that the DNA fit between

$$\begin{aligned} U_{\text{total}} &= \sum_{\text{bonds } i} k_i^{\text{bond}} \left(r_i - r_{0i}\right)^2 + \sum_{\text{angle } i} k_i^{\text{angle}} \left(\theta_i - \theta_{0i}\right)^2 \\ &+ \sum_{\text{dihedrals } i} k_i^{\text{dihed}} \begin{cases} \left[1 + \cos\left(n_i \phi_i - \gamma_i\right)\right] & n_i \neq 0 \\ \left(\phi_i - \gamma_i\right)^2 & n_i = 0 \end{cases} \\ &+ \sum_i \sum_{j>i} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^6\right] + \sum_i \sum_{j>i} \frac{q_i q_j}{4\pi\epsilon r_{ij}} \end{aligned}$$

Figure 1: The MD potential, where $\mathbf{F} = -\boldsymbol{\nabla} U$.

periodic images[1] of SSB when fully stretched. This axis was chosen by trial and error, rotating the extended DNA and adjusting the size of the unit cell until a suitable pathway was obtained. There may be more thoughtful ways of picking an axis, but this is a one-time task and we chose a quick guess-and-check approach. It is generally better to use a simulation system that is large enough to accommodate the DNA or to occasionally truncate the excess DNA.

[1]Most all-atom MD simulations employ periodic boundary conditions, which allows long-range electrostatic interactions to be efficiently calculated in Fourier space. All other pair interactions in the system are computed between nearest set if periodic images.

2 Setting up a simulation

Suppose your experimental collaborators have been pulling on DNA bound to SSB, and are hoping that you can help them interpret their results. Normally, a new project begins with extensive review of the literature about the system, in this case SSB (PDB accession code: 1EYG). Briefly, SSB is a homotetramer known to bind ssDNA in two modes that depend on ion concentration: SSB_{35} and SSB_{65} [4]. The subscript denotes the approximate number of nucleotides occluded in each mode. SSB_{35} binds ssDNA cooperatively, and can form indefinitely long protein clusters; SSB_{65} binds ssDNA with limited cooperativity. Both binding modes are believed to have functional roles in the cell, but SSB_{65} is the preferred binding mode at physiological ion conditions *in vitro*. Your collaborators have data on the force at various levels of extension, but can't see the geometry of SSB in the experiment, and no one knows if DNA binds to SSB in the 35 or 65-nt mode under tension. Although they about a million of times more slowly than can currently be done in MD simulation, it might be informative to replicate their experiment *in silico* to see, e.g. whether the DNA unravels from both ends at the same rate, or if unraveling proceeds processively from one end.

When reviewing the literature, particular attention must be paid to any crystallographic articles reporting structures that will provide the initial configuration of the system. It is good to carefully examine the structure to obtain as complete an understanding of the protein as possible before simulating; many trivial errors can be avoided early by doing so. For example, portions of a biomolecule sometimes cannot be resolved from the electron density obtained by x-ray scattering.

Load the PDB 1EYG into VMD. This x-ray structure contains two ~ 30 nucleotide ssDNA fragments bound to SSB [4]. The structure depicts a homotetramer with folds that accommodate the DNA, which is held in place through a mix of base-stacking and electrostatic interactions. Models of SSB_{35} and SSB_{65}, made by extending the DNA fragments in the crystal structure, are shown in Fig. 2a and b.

Note that the x-ray structure does not contain hydrogen atoms. In general, hydrogen atoms cannot be resolved in x-ray crystallography. For most biomolecules, hydrogen atoms can easily be added to the heavier atoms. However, some amino acids can gain or lose a hydrogen atom with a likelihood that depends on the pH and the local chemical environment so the placement of hydrogen atoms can be ambiguous. The propensity of a molecule to gain or lose a hydrogen is usually expressed in terms of a pK value, which can be interpreted as the pH where the molecule has an extra hydrogen half of the time. The most ionizable amino acids are thus histidine (pK of $\sim$6) and cysteine (pK of $\sim$8.5). However, under the right conditions, aspartic and glutamic acids can acquire

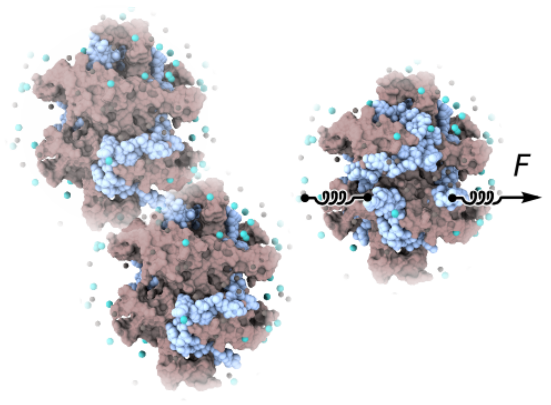


Figure 2: Models of ssDNA bound to SSB. (a) The ends of ssDNA bound to SSB_{35} extend from opposite sides of SSB, allowing unlimited cooperativity. (b) The ends of the ssDNA bound to SSB_{65} extend on the same side of SSB, allowing only limited cooperativity. A method proposed to remove DNA from SSB is illustrated with cartoon springs. The DNA is represented as light-blue van der Waals spheres; the surface of SSB is shown in pink; K^+ and Cl^- are shown as small brown and cyan spheres, respectively.

an extra hydrogen, and tyrosine, lysine and arginine can lose a hydrogen. The terminal amino can also lose its charge (pK of $\sim$9). These ionizable residues are often found around catalytic and ion-binding sites, which should be carefully examined to determine the appropriate protonation state. Otherwise, it is usually sufficient to examine only histidine residues. Histidine usually carries no net charge, but the hydrogen atom can be found on either of the nitrogen atoms in its five-pointed ring. In some cases, a second hydrogen will bind to positively charge the residue.

In VMD, the histidines can be highlighted using the selection text “resname HIS” and the surrounding environment can be highlighted in a second representation with “within 5 of resname HIS”. Using different size Licorice representations may be helpful here. You can look for possible hydrogen bonds with nearby oxygen atoms (red) to determine whether the hydrogen should be placed on the ND1 or NE2 atom by renaming the residue from “HIS” to “HSD” or “HSE”, respectively. For SSB, the qualitative placement of the proton is ambiguous, so no action is required. A more rigorous approach would be to build the protein structure file (PSF), which contains charge information, using HSD and HSE, and then selecting the system with lower electrostatic energy (e.g. by using VMD’s “measure energy” command). However, this more quantitative approach may not work well if the residue is near the solvent.

Finally, cysteine residues in close proximity may form a disulfide bond, which acts as a chemical crosslink that stabilizes many protein structures. These residues can be examined with the selection "resname CYS". You should find that SSB has no cysteine residues.

If you render 1EYG structure with the Trace or New Cartoon representation, you may notice several missing residues in some of the loops of some of the protein monomers. We have provided the PDB file `ssb65.pdb`, which has these missing residues added through homology modeling and has the DNA from the SSB_{65} model. Our strategy will be to pull one end of the DNA using the SMD feature of NAMD. SMD acts by attaching a spring to a group of atoms and moving the other end of the spring at a constant user-defined rate. Unfortunately, only once instance of SMD can be used per simulation, so the other end of the DNA will be held fixed with NAMD "constraints" (these are actually per-atom harmonic restraints). We can do a little better by using "moving constraints" to move the restraints at the same rate and in the opposite direction as the SMD spring so that no net momentum should be imparted to the system by the applied forces. However, with 70 nucleotides fully stretched (about 7 Å per nt), we would need a prohibitively large system to house the dissociated DNA. One solution to this technical challenge is to stop the simulation occasionally and remove excess DNA. Though challenging to do, such a process can be automated through shell scripts that invoke VMD and NAMD in a loop. For this tutorial, you will instead pull the DNA along an unusual axis between periodic images of SSB.

In VMD, source the file `load-extended-dna.tcl`, which loads `ssb65.pdb`, rotates the coordinates so that the DNA ends lie along the SMD axis, sets the size of the simulation cell, writes the PDB `ssb-oriented.pdb`, and finally extends the ssDNA fully along the SMD axis for subsequent visualization. The **Periodic** tab of the **Graphical Representations** window enables you to show or hide periodic images of DNA. Observe how the DNA will fit between the periodic images of the protein. Note that we assume the DNA will lie along a line as it is removed from the protein. This will be more-or-less true at the rapid rate the DNA will be pulled (150 Å/ns), but there may be unwanted interactions between periodic images. The strategy employed in this tutorial is not recommended for research projects!

In order to perform an MD simulation using NAMD, you must have at least three files:

1. a PDB containing the coordinates and names of each atom;
2. a PSF containing information—mass, charge, and atom connectivity (bonds, angles, dihedrals and impropers) and atom type for van der Waals parameters—that will later be used by NAMD to determine what forces to apply to each atom; and

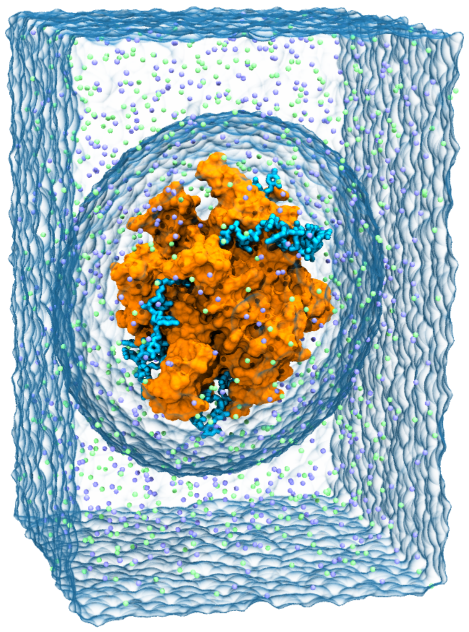
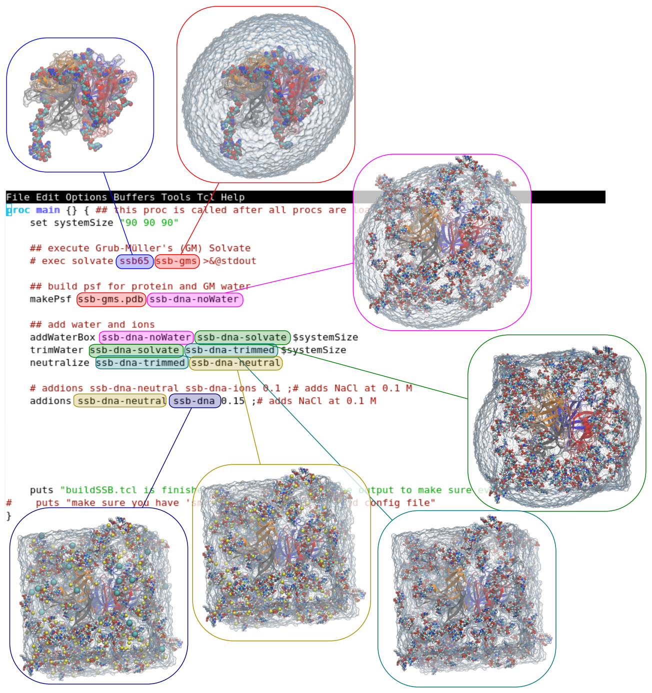


Figure 3: Final system containing SSB_{35}. The surface of water added with the solvate plugin of VMD is shown transparently, and indicates the size of the system as well as the size of the solvation shell from Grubmüller's Solvate. Ions are shown as light green and blue vdW spheres. The protein is shown using an orange surface, and the DNA is shown with in cyan with atomic detail.

3. a NAMD configuration file that instructs NAMD what and how to run the simulation.

The PDB `ssb-oriented.pdb` contains the protein and DNA structures in the desired initial configuration and will provide the basis for building the PDB and PSF. This can be done using the graphical AutoPSF plugin of VMD. However, this guide uses the latest version of the CHARMM force-field (CHARMM36), which uses a new patch to convert the default RNA into DNA (more on this later) that breaks the current version of AutoPSF. Thus, this tutorial will guide you through writing a simple Tcl script that is sourced from within VMD to produce the structures. In general, such scripts are more flexible than the built-in graphical interfaces, and also leave a precise record of how your system was built, which can be invaluable. The build process resembles that depicted in Fig. 4.



```
proc main {} { ## this proc is called after all procs are lo
    set systemSize "90 90 90"

    ## execute Grub-Müller's (GM) Solvate
    # exec solvate ssb65 ssb-gms >&@stdout

    ## build psf for protein and GM water
    makePsf ssb-gms.pdb ssb-dna-noWater

    ## add water and ions
    addWaterBox ssb-dna-noWater ssb-dna-solvate $systemSize
    trimWater ssb-dna-solvate ssb-dna-trimmed $systemSize
    neutralize ssb-dna-trimmed ssb-dna-neutral

    # addions ssb-dna-neutral ssb-dna-ions 0.1 ;# adds NaCl at 0.1 M
    addions ssb-dna-neutral ssb-dna 0.15 ;# adds NaCl at 0.1 M




    puts "buildSSB.tcl is finish                    utput to make sure ev
#    puts "make sure you have 's                       d config file"
}
```

Figure 4: An overview of the system assembly process. A typical compartmentalized system assembly script is depicted. The script was written to evoke explicitly named Tcl procedures that serve as logical wrappers. From top left, clockwise: the system is shown in its initial state, containing protein and DNA with no hydrogen atoms; structured water is added to the DNA and protein using Grubmüller's Solvate program (distinct from VMD's Solvate plugin); the DNA bound to the protein is cut into small pieces that are randomly distributed through the system; VMD's solvate plugin is used to add water that isn't too close to the protein; excess solvent is removed so that the system is a cube; the protein and DNA charge is neutralized as counterions and coions are added to the system at ~ 1 M concentration using the autoionize plugin of VMD (which also has a graphical interface).

Psfgen

VMD is unable to build a PSF without the help of the program "psfgen". Usually psfgen is accessed as a plugin from within VMD, and it unfortunately has similar commands and data structures that can make building a PSF difficult or confusing for the new student. It is essential to understand that psfgen has its own internal memory space, which is completely independent from the internal memory of VMD. Molecules are loaded into psfgen by reading data from files with special psfgen commands, and these molecules "know" nothing about the molecules loaded in VMD. Usually, a psfgen molecule is created one segment at a time, where a segment refers to a contiguous set of bonded atoms. Thus, the system building process involves loading your PDB into VMD, writing each segment to a temporary PDB that psfgen reads. At the end of this, only water and ions need to be added to the system to prepare it for simulation, and these tasks are easily accomplished at the command line using VMD plugins.

A crash-course in Tcl. Before we proceed it is necessary to review Tcl, a general purpose scripting language used by VMD. Below, Tcl commands are highlighted in blue and placeholders are italicized. Unless you feel quite comfortable with Tcl, we encourage you to try typing each code example in the Tk Console.

Tcl takes commands line-by-line and the usual format is:
`command-name argument1 argument2...`
For example, the command `puts "hello world"` will write "hello world" to the Tk Console or to `stdout`. Lines with multiple commands can be written with a semi-colon separating commands. Variables are assigned using:
`set` *`varname value`* and can be used in command arguments later by writing *`$varname`* or *`${varname}`* if special characters are present in *varname*.

Math can be performed with the `expr` command, as in `expr "5+2"`. Expressions involving integers will return integers and can cause unexpected behavior, for example try typing `expr "5/2"` into the Tk Console. You can insert the result of an `expr` command (or any other command) in another command's arguments by enclosing the `expr` command in brackets:
`puts "result: [expr "5+2"]"`. This is called *command substitution*, and it is very useful.

Double-quotes are used to create lists whose elements are delineated by white-space (spaces, tabs, newlines). While `set myList "a b c"` works as expected, the command `set myList a b c` fails because four arguments were provided to the `set` command, which only expects two arguments. Curly braces can also be used to make lists, `set myList {a b c}`. There is a vital difference between quoted lists and braced lists: variable expansion and command will be performed inside of quoted lists, but not braced lists. In addition, nested lists

are easy to write with braces but not quotes (the latter must be done using variables or the `list` command). Experiment with both kinds of lists in the Tk Console until you are comfortable with how these work. What do you think the output of the following commands will be?
`set var 1; puts "$var {2 [expr 2.0+$var]}"`.

Both kinds of lists can span multiple lines. In Tcl, blocks of code are technically multi-line lists (almost always written with braces). As with all high-level languages, Tcl contains *control statements* like `if`, `while`, and `for`. For example, you can create blocks of code that are conditionally executed as follows:

```
if { 1 == 1 } {
     puts "this is a block of code that can span multiple lines"
}
```

The first argument to the `if` statement is a conditional expression and the second argument is a block of code that will be evaluated when the conditional expression evaluates to a value other than 0.

We have only covered some of the basics, but it should be enough to get started writing the molecule building script. There are three online resources that are particularly useful when writing Tcl scripts for VMD:

1. The Tcl Reference Manual (http://tmml.sourceforge.net/doc/tcl/), which contains information about Tcl commands
2. The Tcl Text Interface section of the VMD user's guide (http://www.ks.uiuc.edu/Research/vmd/current/ug/node116.html), which explains the extra Tcl commands understood by VMD
3. The psfgen User's Guide (http://www.ks.uiuc.edu/Research/vmd/plugins/psfgen/ug.pdf), which describes how the psfgen plugin of VMD can be used.

Scripting the system building process. Open your favorite text editor and create a new file called `psfgen.tcl`. It is also a good idea to have a fresh VMD session open while you work on your script so that you can test commands in the Tk Console and also test the text used to create "atom selections" (more on this later). At any point you can test your script by typing `source psfgen.tcl` in the Tk Console. Afterwards, you may want to remove any molecules loaded with the VMD command `mol delete all` and reset psfgen's state with the psfgen command `resetpsf`.

1. **Prepare VMD and psfgen**
   The first step towards writing a PSF using a script is to tell VMD to use the psfgen plugin with the command `package require psfgen`[2]. Now, all

[2]Ordinarily the Tk Console automatically loads the psfgen package, but your script should contain this line in case you run the script from the command line with `vmd -dispdev text -e psfgen.tcl` or `vmd -dispdev text < psfgen.tcl`

the psfgen commands are available to the script. The next step is to read in the force-field topology files. These contain information about the atoms in each protein residue or nucleotide, including how they are bonded, what charge they have, and what "types" of atoms they are (this last bit of information is used by NAMD in conjunction with the force-field's parameter file to determine what forces to apply). The topology files can be read with the psfgen command `topology` *`path/to/file.rtf`*. Your topology files are located in the directory `charmm36.nbfix`. Make your script load all the files with the `.rtf` extension. Finally, make your script load the PDB into VMD with the command `mol new ssb-oriented.pdb`

Now VMD and psfgen are ready to build your DNA and protein. Again, this is done piece-by-piece using contiguous "segments" of bonded atoms. For example, SSB is a homotetramer comprised of four (identical) monomers, and each monomer will have its own segment. In preparation for adding the segment to psfgen (again, psfgen has its own memory of the molecule that you are building that is *completely* separate from the molecules loaded into VMD), your script should write a PDB for the first segment.

2. **Load DNA into psfgen**
This can be done in a few lines by first creating an "atom selection" with `set sel [atomselect top "nucleic"]`. Here, the atom selection text, "nucleic" works just like it would in the **Graphical Representations** window and selects all the DNA atoms. The `atomselect` command returns a unique label that can be used as a Tcl command to query or manipulate the selected atoms. When a Tcl command is contained in square brackets, the command is executed and the bracketed-command is substituted with whatever it returns, in this case the unique atomselect label (something like atomselect0 or atomselect1). The code above saved that label in the variable `sel` (this variable could be called anything) for later use. Usually a PDB from the Protein Data Bank does not have its up-to-four-letter segment defined, but this is crucial for psfgen. The command `$sel set segid DNA` sets the segment name to "DNA" for the atoms in the selection. Finally `$sel writepdb tmp.pdb` writes the PDB file containing the atoms in the selection. Add the three lines described above to your script.

Once again, *adding or deleting molecules from VMD does not affect psfgen*. Similarly, *psfgen commands do not alter VMD's state.* Thus each monomer must be added to psfgen's picture of the structure using the command:

```
segment unique-segname {
      code-to-specify-residues-in-segment
}
```

The code argument to `segment` usually contains one psfgen command, `pdb tmp.pdb`, that tells psfgen to extract the bonded information for the segment from a PDB. A segment is almost always a set of covalently bonded atoms[3], such as a DNA molecule or protein monomer. Note that the segment command does not load coordinates into psfgen. Coordinates must be read into psfgen after the `segment` command using `coordpdb tmp.pdb`. At this point, psfgen should have your first segment molecule in its memory!

3. **Load protein into psfgen, one segment at a time**

   You can repeat the above steps for each protein segment using VMD before adding it to psfgen for the four monomers using suitable atom-selection text instead of `"nucleic"`. Most PDB files organize protein monomers into "chains" that are represented through a single letter. In the SSB structure, the chains for the four monomers are "A B C D". Try typing `chain A` into the **Graphical Representations** window to view just one monomer. In general, if you are building a system with multiple segments, you will need to find these chains. You will see how to find the chains in the PDB momentarily, but first we'll just assume that you know the chains so you can finish your script.

   You could adapt the code used to write the DNA segment to your protein by copying it once for each monomer, but there is a cleaner way that leverages the power of a programming language. You can use a loop to improve your script as follows, placing your `segment` and `coordpdb` commands inside the loop:

```
set chains {A B C D}
foreach chain $chains {
       set seg ${chain}PRO
       puts "Adding segment $seg to psfgen"
       ...
}
```

   The `foreach` command iterates through the elements in the list in `$chains`.

[3]there are some exceptions: multiple water molecules or ions are, for example, normally represented in a segment

In each iteration, the variable `chain` is set to the next list element before executing the code contained between the braces. The code between the braces should be very similar to the code used to add the DNA to psfgen, but the atomselection should be `"protein and chain $chain"`.

If you want to generalize your script so that it works for most proteins, you can obtain the list of chains for your `foreach` loop programatically. First, create an atomselection, `$sel`, containing at least one atom per chain that you would like to build. The command `set chains [$sel get chain]` will return the chain corresponding to every atom in the selection. The command `lsort -unique $chains` will return only the unique items in the original (likely very long) list of chains. Use the above information to make your script load the protein into psfgen, one chain at a time. The techniques learned here can be useful in other ways; for example, you could use the same approach to determine what amino acids are present in a protein.

Chains and fragments. VMD automatically adds an integer to each atom to identify each disjoint set of atoms as a unique *fragment*, which you can use instead of the chain. This is useful because the chain attribute will occasionally be missing from a PDB. However, be aware that residues missing from a PDB can fracture a single chain into multiple fragments.

4. **Apply patches in psfgen to make DNA instead of default RNA**
   At this point, if you wrote the PSF and PDB from psfgen's memory using the `writespsf` *`file.psf`* and `writespdb` *`file.pdb`* commands, you would end up with RNA, and not DNA (these differ chemically by only an OH group). You can try this to see firsthand the difference between RNA and DNA.

   Between the `segment` and `coordpdb` commands for the DNA[4], you should "patch" the RNA to turn it into DNA. This is easy, but requires a loop over each nucleotide. First, create a selection corresponding to the DNA and use it to create a list of resids with
   `set resids [lsort -unique [$sel get resid]]`. Then loop over those resids with `foreach r $resids { ... }` Inside that loop, apply the patch to the nucleotide "DEOX"[5] with `patch DEOX DNA:$r`.

   In addition to the bonds, the PSF specifies which atoms should have angles and dihedral angles applied. Unfortunately, patch statements do not usu-

[4]This might work after the coordpdb just fine, but we haven't tested it; in general it's best to apply patches *before* reading coordinates into psfgen.

[5]This patch is defined in the topology file for nucleic acids and changes the RNA structure currently in psfgen's memory into DNA.

ally specify changes to the angles and dihedrals, so you need to provide the psfgen command `regenerate angles dihedrals` to automatically reinsert the angles and dihedrals in the PSF.

5. **Write psfgen's PSF and PDB**
   At this point, the PSF is almost complete. However, the `tmp.pdb` files did not include all the atoms in the structures (namely hydrogen atoms were missing). Before you write the structure, you should tell psfgen to use the "internal coordinates" specified in the topology file to place the missing atoms in reasonable positions by issuing the command: `guesscoord`[6]. Psfgen flags the "occupancy" field of the atoms whose coordinates were guessed, which can be used to visualize these atoms by typing `occupancy > 0` in the **Graphical Representations** window. In some rare cases a side-chain might not be resolved and a bond can "pierce" an aromatic ring. This kind of steric clash will not usually be resolved through minimization and should be watched for.

   Finally, you are ready to write the PSF and PDB from psfgen's memory using the commands `writepsf psfgen.psf` and `writepdb psfgen.pdb`. Keep in mind that these are distinct from the VMD command (e.g. `$sel writepdb` *`file.pdb`*) used to write pdb files from atomselections in VMD's memory. Go ahead and source your script from within VMD with `source psfgen.tcl` from the Tk Console. Look at the resulting structures for anything unusual; you have just completed the trickiest part of building a structure, and things can easily go wrong. For example, if you forgot to load some atom coordinates into psfgen, they will be located at the origin, and unusually long bonds in your resulting structure are an indication that something may have gone wrong.

6. **Add solvent**
   The next step is to add water to the system.

[6]Psfgen will print many warnings that there are "poorly guessed coordinates" when the topology file doesn't explicitly specify the bond-lengths and angles expected for an atom. However, psfgen almost always guesses atomic coordinates adequately for a simulation and you can generally ignore these warnings.

Careful placement of water. The structure of water can be influenced 10 Å from a surface, and in this way can act as an extension of the protein. Moreover, the structure of the water around a protein can stabilize its conformation. We often use a pair of programs called Dowser and Grubmüller's Solvate (accessed from the command line with `dowserx` and `solvate`) to place individual water molecules in energetically favorable locations near the protein. Dowser places water molecules in cavities within a protein. These molecules often cannot be resolved in x-ray structures but can be essential for the structural stability of a protein. Grubmüller's Solvate places water molecules in energetically favorable locations around a protein, resulting in a tighter solvent–solute interface. This optional step may very slightly reduce the necessary equilibration time but requires an extra, slightly complicated step during system assembly. Note that Grubmüller's Solvate is distinct from the solvate plugin of VMD, which places pre-equilibrated water without considering the interaction between the water and the protein. Both Dowser and Solvate are executed prior to creation of the PSF. To make this tutorial more portable, these steps have been omitted.

Unstructured solvent can be added using VMD's graphical Solvate plugin, but you can also add two lines to your `psfgen.tcl` script as follows:

```
package require solvate
solvate psfgen.psf psfgen.pdb -minmax "{-57.5 -57.5 -75} {57.5
57.5 75}" -o solvate
```

The numbers in `minmax` specify the extent to which the solvent should reach, and have been chosen to allow the DNA to move between periodic images of SSB. Usually, you should provide enough solvent so that there is a minimum of 20–30 Å separation between the surfaces of periodic images of the solute. This criterion is a rule-of-thumb based on the observation that the structure of water is affected up to 10 Å from the surface of a protein. Furthermore, electrostatic interactions are (approximately) exponentially screened on the characteristic Debye length, which is ~10 Å in 100 mM monovalent electrolyte, and ~3 Å in 1 M.

Sometimes `solvate` adds a little too much water, and needs to be trimmed. This isn't the case for the box used in this tutorial, but you can consult section 2.2 of the psfgen User's Guide for detailed instructions should this problem ever arise.

7. **Add ions to system**
   DNA is highly charged (one negative electron charge per phosphate). Counterions are expected to, more-or-less, neutralize the DNA within a couple of Debye-lengths, so the system should be neutralized before additional ions are added to the appropriate concentration. This is very easily

achieved using the graphical Autoionize plugin of VMD, which also has a convenient scripting interface. You can make your script neutralize the system and then add ions to 1 M concentration with the following commands (the `-sc` option specifies the desired molarity):

```
package require autoionize
autoionize -psf solvate.psf -pdb solvate.pdb -sc 1 -o ssb
```

Extra precision when adding ions. The autoionize plugin of VMD adds ions in random position by substituting for water molecules based on the number of water molecules present in the system. However, the plugin doesn't account for water molecules removed, which can cause the ionic concentration to be a few percent larger than desired at high ionic strengths (e.g. 1 M). Changes of a few percent in the ion concentration are unlikely to have significant effect in most biological systems, but for those desiring higher precision, the following approach can be used. First the system should be neutralized, using for example the `-neutralize` option of autoionize instead of the `-sc` option. Alternatively, the program `cionize` may be used, which places ions sequentially in optimal positions according to Coulomb electrostatics. This may be especially important when creating very large structures (millions of atoms) that can only be simulated for a short time because it will take less time for the ion atmosphere to relax with `cionize`. The usual expression for the molality of an ionic species c_i (concentration by weight) is $c_\mathrm{i} = \frac{N_\mathrm{A} n_\mathrm{i}}{m_\mathrm{w}} = \frac{n_\mathrm{i}}{0.018 n_\mathrm{w}}$, where N_A is Avogadro's number, n is the number of atoms and m_w is the mass of water in kilograms, the subscripts i and w refer to ions and water, respectively[a]. n_w is the number of water molecules *after* ions have been added, which can be related to the number n_0 of water molecules before ions have been added. n_0 can be obtained with the command `[atomselect top "water and noh"] num`. Assuming monovalent electrolyte such as NaCl, $n_\mathrm{w} = n_0 - 2n_\mathrm{i}$ and thus $n_\mathrm{i} = \frac{0.018 n_0}{1+2\cdot 0.018 c_\mathrm{i}} c_\mathrm{i}$. Rounding this number to the nearest integer, ions can be added by using the autoionize option `-nions "{SOD` n_i`} {CLA` n_i`}"`. For a list of ion species known to autionize, type `autoionize` at the Tk Console without arguments.

[a]Note that we are using molal concentration here with the factor of 18 coming from the atomic mass of a water molecule. If one were to insist on using molar ion concentration, this factor might be too small; the density of the TIP3P water model is a few percent lower than actual water. Whereas for real water, molarity and molality should closely coincide, for TIP3P, these measures of concentration differ slightly. The autionize plugin uses a factor of 0.0187 so that c_i is provided as a molarity. Since the density of water depends slightly on simulation parameters (e.g. PME), we feel the most accurate way to report ion concentration is by specifying the molal concentration.

After adding the `solvate` and `autoionize` commands to your script, open VMD and source your script with `source psfgen.tcl` to execute all the commands. Load the resulting structure and make sure everything looks okay. Check that the system is neutral with the following command: `measure sumweights [atomselect top all] weight charge`. The command returns the total charge of the system in electron charges and it should return a value of magnitude significantly less than one. The `measure` command provides a lot of useful functionality to VMD, especially for analysis of simulation trajectories.

8. **Create PDB to specify constrained and SMD-forced atoms**
   The final step is to flag atoms in order to apply force using SMD and movingConstraints. In the provided NAMD configuration file, atoms in `ssb.pdb` that are flagged with a beta of 1 will have movingConstraints applied and atoms flagged with an occupancy of 1 will have SMD applied.

   In your favorite editor, open a file called `constrainDNA.tcl`. In this file, you must first load your PSF and PDB (`mol new ssb.psf` and `mol addfile ssb.pdb`). The next step is to set the occupancy and beta of each atom to 0 by creating the selection `set all [atomselect top all]` and `$all set` *`field`* `0`, where *`field`* is a placeholder for occupancy or beta.

   Now make the script set the beta field of the C1′[7] atom of the bottom-most nucleotide to 1. Use VMD interactively to figure out the resid of that nucleotide. Alternatively you can use the `lindex` and `lsort -unique -integer` commands to make your script programatically find the first resid. Similarly, set the occupancy field of the C1′ atom of the top-most nucleotide to 1. Lastly, write over `ssb.pdb` with `$all writepdb ssb.pdb`. Now source your script in VMD and verify that the proper atoms are flagged in `ssb.pdb`

3 Simulation

Simulations can be performed in the NPT (constant number of atoms, pressure, temperature), NVT (constant number of atoms, volume, temperature), or NVE (constant number of atoms, volume, energy) thermodynamic ensembles. Water is a nearly incompressible fluid, so small changes in the volume cause large changes in the pressure. When building a system, it is almost impossible to obtain a pressure close to atmospheric without simulating in the NPT ensemble. External forces (which in general do not conserve momentum) may interact badly with the barostat. Furthermore, since the external forces will do work and add energy to the system, it is best to perform the production simulation in the NVT ensemble. But how do you ensure that the pressure in this simulation will be close to atmospheric pressure?

1. **Perform NPT simulation to obtain the system volume**
   A good approach is to first run a short NPT simulation without external

[7] This atom is the carbon on the sugar that connects to the base. It is a good proxy for the center of mass of a nucleotide.

forces until the volume of the system stops changing, then use the volume obtained in the NPT simulation to start an NVT simulation using the correct system size. In this case, the terminal nucleotides must lie along the steered-molecular dynamics (SMD) pulling axis at the onset of the SMD simulation. Because the DNA ends may drift away from their initial positions in the NPT simulation, it is best to start the NVT simulation using the original coordinates rather than the NPT simulation's restart coordinates.

The solvent at the edges of the system may have clashes or small gaps that could perturb the solute conformation. Thus it is best to perform initial equilibration in the NVT ensemble with the solute conformation restrained (constrained in NAMD terminology). Once, the system is equilibrated, SMD simulation can begin.

The NPT simulation has already been performed on your behalf.

2. **Extract the average volume from the NPT simulation**
When starting an NVT simulation from an NPT simulation, it is common practice to simply use the "extended system" restart file. This practice isn't ideal because the system volume fluctuates during the NPT simulation; the volume of the production simulation would be randomly selected from these fluctuations. For the present system, the fluctuations are about 0.1 Å along each axis, which is smaller than the size of an atom. However, changing the system volume by even this tiny amount can result in significant changes of the pressure because water is nearly incompressible.

   A somewhat better approach is to find the average the volume during the NPT simulation, and scale your cellBasisVectors accordingly. From the Tk Console, run the following command to extract the average system volume: `source averageVolume.tcl`.

3. **Perform NVT equilibration simulation**
Enter the correct cellBasisVectors into `ssb-nvt.namd` and run this locally. This simulation will equilibrate your system with "constraints" (really restraints, but NAMD syntax is not always precisely descriptive) and SMD forces defined (but a value of 0 for the SMD velocity so the ends are merely held in place). Note that the cellBasisVectors are a little smaller than the initial size of the system, and water around the edges will be roughly twice the nominal density when you begin the NVT simulation. Using the speed of sound in water (1500 m/s) to estimate the timescale required for the uneven water density to propagate through the system, you should simulate at least 6.6 ps per 100 Å. While this simulation runs, have a careful look at `ssb-nvt.namd` and `ssb-smd.namd` to make sure you understand the configuration files well. Don't hesitate to ask questions

about the various options. You can also stop the simulation prematurely and move on to the next section; a complete trajectory is provided for the production simulation, so it is okay if the system is not fully equilibrated.

4. **Perform production simulation with pulling forces**
   Once the simulation is finished, you should run `ssb-smd.namd` for a moment to ensure that everything worked. `ssb-smd.namd` is the same as `ssb-nvt.namd`, except it uses an SMD velocity of 150 Å/ns and uses the system volume from the `restart.xsc` file if the **`ssb-nvt`** simulation.

4 Analysis

1. **Examine trajectory**
   Load and examine the SMD simulation trajectory in VMD (`mol new complete/ssb.psf`[8]`; mol addfile complete/output/ssb-smd.dcd` in the Tk Console). Watch how the DNA dissociates from SSB.

   In the limit of slow pulling, you can safely assume that the force due to `movingConstraints` is the same as the force due to SMD. In the provided simulation trajectory, the pulling velocity was extremely fast, and the above assumption may not hold. The only reliable way to extract the force due to `movingConstraints` is to use VMD to track the position of the C1′ atom. However, for simplicity we will assume the movingConstraints and SMD forces have equal magnitudes.

   The SMD force can be extracted from a NAMD log file using any number of scripting languages or utilities. If you are comfortable with a particular scripting language (e.g. awk, Perl, or Python), feel free to extract the force from the log file using that language. Presently, you will be guided through this task using Tcl.

2. **Extract SMD force from NAMD log file**
   The line you are trying to copy from the log file looks like this:
   `SMD  0 -2.88316 26.5742 -33.5497 150.378 261.271 -3359.08`.
   This line has the format: "SMD timestep posX posY posZ forceX forceY forceZ". Create a new tcl script called, `getForce.tcl`. First, set a variable to the axis direction as follows `set axis [vecnorm "92 115 60"]`. `vecnorm` is a handy VMD command that normalizes a vector. In this file, use the Tcl command `set ch [open complete/output/ssb-smd.log]` to open the file for reading. The `open` command returns a unique channel ID, which you set to the `ch` variable. The command `gets $ch line` will

[8]The provided trajectory was built with an old version of the solvate plugin and has a different number of atoms compared to your PSF

read a line from the file, setting it to the variable `line` and returning the number of characters on the line, or −1 if it reached the end of the file.

To step through the file, you can use a while loop, which executes a conditional statement and then executes the loop code as long as the conditional statement was true (0 in Tcl). For example, `while { [gets $ch line] >= 0 } { puts $line }` would simply copy the contents of the file to the Tk Console. Inside the loop, you must write code that checks if the line begins with "SMD " (note the extra space prevents the line beginning "SMDTITLE" from being printed).

The easiest way to do this is with a very simple *regular expression*[9]. Use an `if` statement and the command `regexp "^SMD " $line` to only print lines that begin with "SMD ". Check to see if it works in the Tk Console by sourcing your script.

There are several ways to extract the relevant information, but the easiest employs either the `lrange` *`list index1 index2`* or `lassign` *`list var1 var2 ...varN`* command. Use one of these techniques and the format of the SMD line (given above) to get the vector form of the force. Don't hesitate to read about these commands in the Tcl Reference Manual. By taking the dot product[10] between the force vector and the axis of pulling, you can obtain the magnitude of the force. Test to see if this works. Although NAMD usually describes force using kcal/mol Å, the force printed in the SMD output is in units of piconewtons.

Now that you have a basic script, you can open a file for *writing* (rather than reading as we have just done) using the command `set outCh [open` *`outfile.dat`* `w]`. It doesn't matter whether the file was pre-existing, but opening a file like this will erase its contents. Subsequent commands like `puts $outCh "some text or data"` will print "some text or data" into `outfile.dat`.

Thus, you can print the magnitude of the force along the SMD axis inside the loop to a data file of your choosing. Finally, after the close of the `while` loop, you should close both of the open file channels with the `close $ch` command. Now have a look at the resulting forces using your favorite plotting software! Note that you will need to employ heavy smoothing to see the signal emerge from the noise. The force that you obtain should be quite large. This is because the pulling velocity was extremely rapid. At

[9]Regular expressions are implemented in many scripting languages and provide a powerful method for querying and manipulating text. See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for more information about regular expressions in Tcl.

[10]`vecdot $v1 $v2` where `$vN` is a list of numbers like `"1.0 0.0 0.0"`

a slower rate of 1 Å/ns, the force is on the order of 100 pN, which is still much larger than the forces obtained in experiment.

If you have time, you can modify your script to print the work performed by the SMD spring. Recall that the work done by the spring is equal to the applied force times the displacement. You can approximate the work done between two SMD output statements by considering the average force and the displacement. The displacement along the pulling direction can be obtained by extracting the position of the SMD atom at each of the two times, taking the difference with the `vecsub` command, and taking the dot product between the force vector and the displacement vector with the `vecdot` command. The total work is just the cumulative sum of the resulting quantities.

Equilibrium information from non-equilibrium simulations. If you were to perform this simulation many times, you would be able to apply Jarzynski's equality [3] to obtain an estimate of the free energy change in removing the DNA from SSB from the work performed during the non-equilibrium trajectories.

$$e^{-\beta\,\Delta F} = \overline{e^{-\beta\,W}}$$

The bar denotes an ensemble average; β denotes $1/\mathrm{k_B T}$; ΔF is the change in free energy when the system is brought from one state to another; W is the work done during the change of state. Jarzynski's equality is a relatively recent development in statistical mechanics that has be experimentally validated. We find this development significant because it relates work performed during a non-equilibrium process (performed many times) to an equilibrium property of the system. There are other ways of obtaining free energies from MD simulations, including umbrella sampling, adaptive biasing force, and metadynamics. but we highlight Jarzynski's equality because it has applications in both experiment and simulation.

3. **Track the unraveling of DNA from SSB at either end**
Create a new script. Load the PSF. You can load the DCD with `mol addfile` *`file.dcd`* `waitfor all`. The option `waitfor all` makes the `mol` command wait to return until the entire DCD is read; by default `mol` will return after just the first frame of the DCD is read. Now, create a skeleton of a loop over the frames of the trajectory. The most straightforward way is to find the number of frames using `set nf [molinfo top get numframes]`. Then create a `for` loop that runs an index from 0 to $nf:

`for {set f 0} {$f < $nf} {incr f} {...}`.
Inside that loop, you will find and print the resids of the first and last nucleotides bound to the SSB.

Before the loop, create an atomselection to select the nucleotides bound to SSB: something like `set bound [atomselect top "nucleic within 5 of protein"]`. Of course, the nucleotides near the protein vary during the simulation. The selection text is evaluated during the current VMD frame when the `atomselect` line is read, which should be the last frame of the DCD. If you change frames afterwards (from the GUI, using `animate goto` *`frame`*, or using `$bound frame` *`frame`*), the selection will still point to the original atoms selected. To re-evaluate the selection text, you must change the frame for the selection `$bound frame` *`frame`* before executing `$bound update`. Use the above information to update the `$bound` at each frame inside the loop.

Now you must extract the first and last resids from `$bound`. This can be done with a combination of `$bound get resid`, `lindex` *`list index`* and `lsort -integer` *`list`*. Here, *index* is a 0-based integer (starts with 0). To select items from the end of the list, *index* can be `end` or `end-`*`n`*, where n is an integer.

Finally, add code to print this information, preferably to two different files each with time in the first column and the resid in the second.

How does unwinding proceed? When you plot the data in all-three files, do you see any correlations between DNA unbinding from either end and the applied force? Recall that these simulations were performed with a very fast pulling velocity (much faster than the usual, already-much-faster-than-experiment pulling velocities typically employed in production simulation).

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of The Cell.* Garland Science, New York & London, 4th edition, 2002.

[2] E. V. Bocharov, A. G. Sobol, K. V. Pavlov, D. M. Korzhnev, V. A. Jaravine, A. T. Gudkov, and A. S. Arseniev. From structure and dynamics of protein L7/L12 to molecular switching in ribosome. *J. Biol. Chem.*, 279:17697–17706, 2004.

[3] C. Jarzynski. Nonequilibrium equality for free energy differences. *Phys. Rev. Lett.*, 78:2690–2693, 1997.

[4] S. Raghunathan, A. G. Kozlov, T. M. Lohman, and G. Waksman. Structure of the DNA binding domain of *E. coli* SSB bound to ssDNA. *Nat. Struct. Mol. Biol.*, 7(8):648–652, 2000.

[5] W. A. Rosche, A. Jaworski, S. Kang, S. F. Kramer, J. E. Larson, D. P. Geidroc, R. D. Wells, and R. R. Sinden. Single-stranded DNA-binding protein enhances the stability of CTG triplet repeats in Escherichia coli. *J. Bacteriol.*, 178(16):5042–5044, 1996.