Introduction

This section of the tutorial provides introductory information and some brief background necessary to comprehend the rest of the document.

About this tutorial

Welcome to charmmtutorial.org! This document is designed to teach newcomers the basics of setting up and running molecular simulations using the molecular simulation program CHARMM [1]. It has been written in conjunction with the CHARMMing [2] web portal. CHARMMing is a tool that provides a user-friendly interface for the preparation, submission, monitoring, and visualization of molecular simulations (i.e., energy minimization, solvation, and dynamics). The goal of this tutorial is to teach what is going on "behind the scenes" of the scripts that CHARMMing generates. This will help bridge the gap between using scripts developed by others and writing new CHARMM scripts to perform tasks. This tutorial is aimed at readers with some knowledge of molecular simulation (even if it's only classroom based or derived from using graphical tools such as CHARMMing), who have basic competency with the underlying physics, and who wish to use CHARMM to run a biomolecular simulation and analyze the resulting data.

These readers will primarily be advanced undergraduate or beginning graduate students. It does not aim to teach molecular simulation *per se*, but it does give background when needed to understand the examples with appropriate references given. The reader is not expected to know much about the practical details of simulation, but the basic principles of physical and biological chemistry are assumed to be known. To be specific, the reader is expected to know basic facts about:

Assumed biochemistry background

- Biopolymers [3] (e.g. proteins, carbohydrates, and nucleic acid chains) and their constituent subunits (amino acids, monosaccharides, and individual nucleic acids)
- Sources of structural data (e.g. the Protein Data Bank [4], Cambridge Structural Database [5]).

Assumed physics / physical chemistry background

- Atomic structure, ionic [6] and covalent [7] bonds, bond energy [8]
- The relationship of atomic structure to system energy
- Nonbonded forces [9], such as electrostatics [10] and van der Waals interactions [11]
- Some statistical mechanics [12], e.g., the reader should have some familiarity with the basic ensembles (in particular micro-canonical [13] and canonical ensemble [14]), know about relationships between macroscopic [15] and microscopic [16] properties (e.g., temperature [17] and average kinetic energy [18]), and have heard about the ergodic theorem [19].

Assumed computer background

This tutorial assumes that you have login ability to a Unix machine (this includes MacOS X). We further assume that CHARMM is already installed on this machine and you know the command to invoke it. If you just received the CHARMM distribution and need help installing it, here are some installation instructions.

Since CHARMM is a command line program, you need some familiarity with the Unix shell (the Unix command line), even on MacOS X! You should be able to navigate the directory hierarchy, copy and move files, and know how to use a text editor. At the time of writing this tutorial, one good Introduction to the Unix command line can be found here [20]; should this link be broken google for something like "introduction to the unix command line".

Suggested reading list

This list of texts is not definitive, but books that the authors have found useful.

Biochemistry

Material about properties of amino acids and nucleic acids, as well as the structure of proteins, DNA and RNA in, e.g.,

- Phillips, Kondev, and Theriot. *Physical Biology of the Cell*. Garland Science. ISBN 0815341636
- Elliott & Elliott. *Biochemistry and Molecular Biology*. Oxford University Press. ISBN 0199226717
- Berg, Tymoczko and Stryer. *Biochemistry*. W.H. Freeman & Comp. ISBN 0716787245

and similar tomes.

Physical Chemistry

Some general texts on physical chemistry contain quite good introductions to statistical mechanics/thermodynamics. In addition:

- Hill. *An Introduction to Statistical Thermodynamics*. Dover Publications. ISBN 0486652424
- Dill and Bromberg. *Molecular Driving Forces: Statistical Thermodynamics in Chemstry & Biology*. Garland Science. ISBN 0815320515
- Chandler. *Introduction to Modern Statistical Mechanics*. Oxford University Press. ISBN 0195042778

Molecular Simulation

- Allen and Tildsley. *Computer Simulations of Liquids*. Oxford University Press. ISBN 0198556454
- Becker, MacKerrell, Roux, and Wanatabe (ed.). *Computational Biochemistry and Biophysics*, CRC Press. ISBN 082470455X.
- Leach. *Molecular Modeling: Principles and Applications*. Prentice Hall. ISBN 0582382106
- Smit and Frenkel. *Understanding Molecular Simulation*. Academic Press. ISBN 0122673514

Unix computing and utilities

- Robbins. *UNIX In a Nutshell*. O'Reilly Media Inc. ISBN 0596100299

About the molecular simulation field

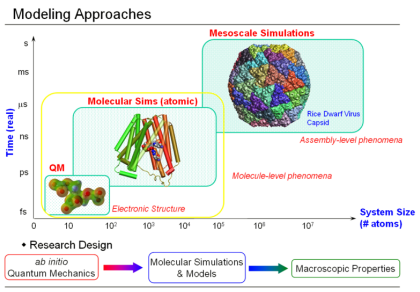


Illustration of the different size and timescales of modeling approaches.

Molecular simulations are performed for a wide variety of purposes. Often, they elucidate how subtle microscopic changes, such as the hydration of a protein interior, affect larger scale processes such as the folding of that protein. Molecular simulation is used across a breadth of disciplines in both organic and inorganic chemistry, however CHARMM, and therefore this tutorial, concentrates mainly on the study of systems of biological interest. Biomolecular simulations can provide insight into reactions that may be difficult to observe experimentally either due to the small size of the compounds involved or the rapid time scale of the event. A variety of techniques can be employed, from simple energy evaluations that can be performed with relatively few operations to long running molecular dynamics or monte carlo simulations using a complex system set up that can take months of computer time. The exact tools used will depend on the type of questions that the simulation (or group of simulations) is expected to answer. The end goal is to provide insight into the physical nature of a system.

Simulations may be performed at different **levels of theory**, depending on their goal. Perhaps the most familiar level is the classical all-atom representation of the system where interactions are modeled without using quantum mechanics [21]. Higher levels than this directly employ the quantum mechanical properties of the atoms (they are used indirectly even in classical simulations as force fields [22] are often parametrized from quantum mechanical data). Lower levels than the classical all-atom [23] generally use coarse-graining [24], i.e. multiple atoms are grouped together into a single point mass. In general, higher levels of theory yield more accurate results, but at the cost of computer time.

As computer power expands, so too does the range of questions that can be answered by simulation. Currently modelers are able to simulate tens to hundreds of thousands of atoms over a time scale of tens to hundreds of nanoseconds at the classical all atom level of theory. Recent simulations of microsecond length simulations of complex systems have recently been reported. As important biological processes such as protein folding take place on the order of microseconds, this is an important development. The increase in computer power predicted (indirectly) by Moore's Law is expected to continue for at least the next decade. Therefore, many previously intractable problems should be solvable in the near future.

About CHARMM

CHARMM (Chemistry at HARvard Molecular Mechanics) is a highly versatile and widely used molecular simulation program. It has been developed with a primary focus on molecules of biological interest, including proteins, peptides, lipids, nucleic acids, carbohydrates, and small molecule ligands, as they occur in solution, crystals, and membrane environments. The CHARMM program has been produced over the last thirty years by a vast team of developers lead by Martin Karplus's group at Harvard University. The program is distributed to academic research groups for a nominal fee; a commercial version is distributed by Accelrys.Information on acquiring CHARMM may be found on the CHARMM development project home page at www.charmm.org [1].

The most up to date reference for CHARMM is a 2009 article in the Journal of Computational Chemistry. (BR Brooks *et al*. CHARMM: The biomolecular simulation program. J Comp. Chem. (30)10 2009. [25])

Basic Information on running CHARMM

CHARMM is a command line program that runs on UNIX and UNIX-like systems (this is why, in the prerequisites section, we wanted you to have access to such a machine). Graphics are available (however they are not covered in this tutorial), but all interaction is done via text commands. Although CHARMM may be used interactively, most use is done via pre-written scripts (i.e. lists of commands that CHARMM executes). The following portion of the tutorial provides the basic information needed to use CHARMM's (powerful) scripting language effectively.

CHARMM can produce a number of files that may be input into third party programs for visualization or analysis (e.g., VMD includes the capability to read CHARMM coordinate and trajectory files). In general, this tutorial does not deal with these third party programs. However, here is a quick example of how to visualize CHARMM coordinate files with VMD: (**vmd -psf structure.psf -cor structure.crd**)

The best source of basic information about CHARMM and its capabilities are the aforementioned journal article and the resources given in the following subsection.

Sources of Further Information

- The CHARMM Web site http://www.charmm.org [1]
  - Information on acquiring CHARMM http://www.charmm.org/html/package/license.html [26]
  - Documentation http://www.charmm.org/documentation/current/index.html [27]
  - Discussion forums (where you can ask for help) http://www.charmm.org/ubbthreads [28]
- Parameter files for CHARMM force fields http://mackerell.umaryland.edu/CHARMM_ff_params.html [29]
- MD workshop at PSC http://www.psc.edu/general/software/packages/charmm/tutorial/index.php [30]

References

[1] http://www.charmm.org
[2] http://www.charmming.org
[3] http://en.wikipedia.org/wiki/Biopolymers
[4] http://www.pdb.org
[5] http://en.wikipedia.org/wiki/Cambridge_Structural_Database
[6] http://en.wikipedia.org/wiki/Ionic_bond
[7] http://en.wikipedia.org/wiki/Covalent
[8] http://en.wikipedia.org/wiki/Bond_energy
[9] http://en.wikipedia.org/wiki/Nonbonded_interactions
[10] http://en.wikipedia.org/wiki/Electrostatics
[11] http://en.wikipedia.org/wiki/Van_der_Waals_interactions
[12] http://en.wikipedia.org/wiki/Statistical_mechanics
[13] http://en.wikipedia.org/wiki/Microcanonical_ensemble
[14] http://en.wikipedia.org/wiki/Canonical_ensemble
[15] http://en.wikipedia.org/wiki/Macroscopic
[16] http://en.wikipedia.org/wiki/Microscopic
[17] http://en.wikipedia.org/wiki/Temperature
[18] http://en.wikipedia.org/wiki/Kinetic_energy
[19] http://en.wikipedia.org/wiki/Ergodic_hypothesis
[20] http://www.mhpcc.edu/training/vitecbids/UnixIntro/UnixIntro.html
[21] http://en.wikipedia.org/wiki/Quantum_chemistry
[22] http://en.wikipedia.org/wiki/Force_field_(chemistry)
[23] http://en.wikipedia.org/wiki/Molecular_dynamics
[24] http://en.wikipedia.org/wiki/Molecular_dynamics#Coarse-graining_and_reduced_representations
[25] http://www.ncbi.nlm.nih.gov/pubmed/19444816?ordinalpos=1&itool=EntrezSystem2.PEntrez.Pubmed.Pubmed_ResultsPanel.Pubmed_DefaultReportPanel.Pubmed_RVDocSum
[26] http://www.charmm.org/html/package/license.html
[27] http://www.charmm.org/documentation/current/index.html
[28] http://www.charmm.org/ubbthreads
[29] http://mackerell.umaryland.edu/CHARMM_ff_params.html
[30] http://www.psc.edu/general/software/packages/charmm/tutorial/index.php

Basic CHARMM Scripting

How to run CHARMM

While you can run CHARMM interactively, one usually tells the program what to do by means of a script. Under Unix (at least for non-parallel versions of the program), this means that in order to execute a (short) CHARMM calculation, one runs from the command line (Unix Shell prompt)

```
charmm_executable < charmm_input_script.inp
```

exploiting input redirection available under all Unix shells. Since as we shall see shortly CHARMM output tends to be verbose, one normally also redirects the output to a file, thus ending up with

```
charmm_executable < charmm_script.inp > charmm_output.out
```

Of course, instead of `charmm_executable` use the path to the CHARMM executablr you have installed on your computer and replace `charmm_input_script.inp` and `charmm_output_file.out` by the names of the actual script which you want to run and the file to which you want to save your output.

Data Structures

- *Residue Topology File (RTF)* This file defines groups by including the atoms, the properties of the group, and bond and charge information. CHARMM has standard Residue Topology Files for nucleic acids, lipids, proteins and carbohydrates. An example of a simple RTF, which describes a single residue (TIP3P water) is given below.

```
* Residue topology file for TIP3 water
*
31 1

MASS     4 HT     1.00800 H ! TIPS3P WATER HYDROGEN
MASS    75 OT    15.99940 O ! TIPS3P WATER OXYGEN

RESI TIP3         0.000 ! tip3p, generate using noangle nodihedral
GROUP
ATOM OH2  OT     -0.834
ATOM H1   HT      0.417
ATOM H2   HT      0.417
BOND OH2 H1 OH2 H2 H1 H2    ! the last bond is needed for shake
ANGLE H1 OH2 H2             ! required
ACCEPTOR OH2
PATCHING FIRS NONE LAST NONE

END
```

As you can see, this file containes a title and immediately following it the rather crypting string "31 1". This is the version number of the topology file, which is tied to the CHARMM version it was released with. Next comes two MASS statements, each of which define an atom type. Atom numbers 4 and 75 are assigned to TIP3P hydrogen and oxygen, respectively. Next comes the actually definition of the residue, which should be fairly self-explanatory, and then the file ends with the END keyword.

- *Parameter File (PARA or PARM)* This file determines the energy associated with the structure by defining bond, angle and torsion force constants and van der Waals parameters. CHARMM has standard parameter files for nucleic acids, lipids, proteins carbohydrates, and water. An example of a parameter file with all ofthe parameters needed to simulate a TIP3 water molecule as defined above is given here. Note that the atom naming convention

in the parameter file matches that in the topology file. Failure to uphold the atom naming and numbering conventions will yield incorrect results, which is why topology and parameter files are released together and it is generally not a good idea to mix yopologies and parameters (however, it is possible to append one set of topologies and parameters to another).

```
* parameter file needed to simulate TIP3 water
*

BONDS
!atom type Kb          b0
HT   HT      0.000     1.5139 ! ALLOW WAT
               ! FROM TIPS3P GEOMETRY (FOR SHAKE/W PARA
OT   HT    450.000     0.9572 ! ALLOW   WAT
               ! FROM TIPS3P GEO

ANGLES
!atom types     Ktheta    Theta0   Kub     S0
HT   OT   HT     55.000   104.5200 ! ALLOW WAT
               ! TIP3P GEOMETRY, ADM JR.

DIHEDRALS

IMPROPER

NONBONDED nbxmod  5 atom cdiel shift vatom vdistance vswitch -
  cutnb 14.0 ctofnb 12.0 ctonnb 10.0 eps 1.0 e14fac 1.0 wmin 1.5

!atom  ignored    epsilon    Rmin/2 ignored   eps,1-4  Rmin/2,1
OT     0.000000  -0.152100     1.768200 ! ALLOW   WAT
             !TIP3P OXYGEN PARAMETERS, adm jr., NBFIX obsolete
HT     0.000000  -0.046000     0.224500 ! ALLOW WAT
             !TIP3P HYDROGEN PARAMETERS, adm jr., NBFIX obsolete

END
```

Note that there are no dihedral or improper dihedrals parameters necessary for TIP3 water as there are only 3 atoms in the residue. Some parameter files also contain CMAP parameters, which are 2-dimensional grid corrections for dihedral angles (see MacKerell, A.D., Jr,. Feig, M., Brooks, C.L., III, Extending the treatment of backbone energetics in protein force fields: limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations, *Journal of Computational Chemistry*, **25**: 1400-1415, 2004 for further details).

- *Coordinates (COOR)* These are the standard Cartesian coordinates of the atoms in the system. These are typically read in or written out in PDB or CHARMM card (CRD -- the default file format used throughout CHARMM) file format. The card format keeps track of additional molecule information that can be useful for structure manipulation (i.e. residue name, segment name, segment id, resdiue id, etc.). Below is an example of a .crd file and the information in contains:

```
title = * WATER
title = *  DATE:     4/10/07      4:25:51      CREATED BY USER: USER
title = *
Number of atoms (NATOM)       = 6
Atom number (ATOMNO)          = 1 (just an exmaple)
Residue number (RESNO)        = 1
Residue name (RESName)        = TIP3
Atom type (TYPE)              = OH2
Coordinate (X)                = -1.30910
Coordinate (Y)                = -0.25601
Coordinate (Z)                = -0.24045
Segment ID (SEGID)            = W
Residue ID (RESID)            = 1
Atom weight (Weighting)       = 0.00000

now what the CHARMM crd file containing that information looks like...

* WATER
*  DATE:     4/10/07      4:25:51      CREATED BY USER: USER
*
    6
   1    1 TIP3 OH2   -1.30910  -0.25601  -0.24045 W    1      0.00000
   2    1 TIP3 H1    -1.85344   0.07163   0.52275 W    1      0.00000
   3    1 TIP3 H2    -1.70410   0.16529  -1.04499 W    1      0.00000
   4    2 TIP3 OH2    1.37293   0.05498   0.10603 W    2      0.00000
   5    2 TIP3 H1     1.65858  -0.85643   0.10318 W    2      0.00000
   6    2 TIP3 H2     0.40780  -0.02508  -0.02820 W    2      0.00000
```

- *Protein Structure File (PSF)* The PSF holds lists of every bond, bond angle, torsion angle, and improper torsion angle as well as information needed to generate the hydrogen bonds and the non-bonded list. It is essential for the calculation of the energy of the system.
- *Internal Coordinates (IC)* This data structure defines the internal coordinates for atoms and can be used for analysis. Internal coordinates represent the position of atoms relative to one another rather than relative to Cartesian axes. In many cases, it is not necessary to deal directly with the internal coordinate data structure, however it is possible to manipulate it within a CHARMM script.
- *Non-Bonded list (NBONds)* This is a atoms which are not bound to each other. It is used in calculating the non=bonded energy terms and electrostatic properties. The non-bonded list does contain atoms that are in atom-to-atom contact and engaging in van der Waals interactions.
- *Constraints (CONS)* Constraints fix atoms in exactly one position during the simulation. This information is stored internally in the IMOVe array.
- *Images Data Structures (IMAGe)* This data structure is used to help create symmetrical structures and contains bond information. This is a general image support system that allows the simulation of almost any crystal and also finite point groups. There is also a facility to introduce bond linkages between the primary atoms and image atoms. This allows infinite polymers, such as DNA to be studied. For infinite systems, an asymmetric unit may be studied because rotations and reflections are allowed transformations.
- *Crystal Data Structures (CRYStal)* The crystal module is an extension of the image facility within CHARMM that allows calculations on crystals to be performed. It is possible to build a crystal with any space group symmetry, to optimize its lattice parameters and molecular coordinates and to carry out analysis of the vibration spectrum of the entire crystal similar to normal mode analysis. All crystal commands are invoked by the keyword CRYStal.

Basic CHARMM script elements

Titles

First, let's do something really silly and start up charmm reading from an empty file; which can be easily accomplished by executing

```
charmm_executable < /dev/null
```

CHARMM prints a header telling you copyright info, version and some more stuff, followed by a warning

```
            Chemistry at HARvard Macromolecular Mechanics
              (CHARMM) - Developmental Version 35b3     August 15, 2008
  Copyright(c) 1984-2001  President and Fellows of Harvard College
                          All Rights Reserved
 Current operating system: Linux-2.6.18-128.7.1.el5(x86_64)@n138.lobos.[+ 15]
            Created on 12/ 2/ 9 at  2:23:40 by user: tim

       Maximum number of ATOMS:    360720, and RESidues:      120240
       Current HEAP size:        10240000, and STACK size:  10000000

 RDTITL> No title read.

     ***** LEVEL  1 WARNING FROM <RDTITL> *****
     ***** Title expected.
     ****************************************
     BOMLEV (  0) IS NOT REACHED. WRNLEV IS  5
```

The job finishes by printing some status info. The interesting part is the warning from which we learn that CHARMM expected a "title". Indeed, each CHARMM script should start with a title, and if the main script tells CHARMM to read from another file, the program also expects to find a title at the beginning of that file.

A title should not be confused with comments. E.g., it can only occur at the beginning of a file (we'll explain the apparent exceptions when we encounter them). Title lines start with a star or asterisk (*); to indicate the end of the title give a line containing only a star. (A title can consist of up to 32 consecutive lines) Thus,

```
* This would be a short title
*
```

If you start CHARMM with a short file containing the above snippet (=title), you get the title echoed in uppercase letters

```
RDTITL> * THIS WOULD BE A SHORT TITLE
RDTITL> *
```

instead of the warning when using the empty file.

Comments

Having blabbered so much about titles, what are comments: A comment in a CHARMM script is everything following an exclamation mark (!_ i.e.,

```
! this is a comment on a line by itself
```

and this would be a line containing a CHARMM command, followed by a comment

```
ENERgy ! as you might expect, this command calculates an energy
```

Ending a CHARMM script

So far, CHARMM finished when it reached the end of the script file (as the line

```
                    NORMAL TERMINATION BY END OF FILE
```

in the output informs you. It's OK to end a CHARMM script in this manner, but the preferred way of stopping CHARMM is by issuing the

```
stop
```

command. We, thus, can create a first, completely useless CHARMM script, looking like

```
* A CHARMM script doing nothing
*

! we really should be doing something, but in the meantime
! all we know is

stop
```

In addition to the title, the comment is echoed as well. Note that CHARMM now prints upon finishing

```
 NORMAL TERMINATION BY NORMAL STOP
```

This indicates that the CHARMM script has finished successfully; an **abnormal** termination message will print if CHARMM exits with an error.

Essential CHARMM Features

This section covers several essential features of CHARMM that are used in most every CHARMM script.

File I/O

CHARMM provides a consistent set of commands for reading and writing files. In general, a file must be `OPENed`, read from or written to, and then, in some cases, `CLOSEd`. To provide a basic example, here is how to read an already-created protein structure file (PSF) into CHARMM:

```
open read card unit 10 name myfile.psf
read psf card unit 10
close unit 10
```

Already, from this simple example, there are several features that must be noted. First and foremost, it is necessary to associate each open file with a unit number (FORTRAN programmers will recognize this immediately). Unit numbers must be in the range from 1-99 for CHARMM versions c35 and older. It is generally good practice to use numbers greater than 10, as the lower number units may be in use by the system (e.g. the standard output stream is usually written to unit 6 on modern systems). Another interesting point is the use of the `CARD` subcommand. CHARMM can read and write two types of files, those containing ASCII text are designated as `CARD` or `FORMatted`, and those containing binary data are designated `FILE` or `UNFOrmatted`. By specifying `CARD` in the `OPEN` and `READ` commands, we are telling CHARMM that we expect the data to be ASCII text (as, indeed, PSF files are). Also in the `READ` command we must tell CHARMM which unit we want to read from, since we can have a number of files open at a given time.

Once we are done with the file it is good practice to `CLOSe` it. If you attempt to re-open the unit number before closing the file, CHARMM will close the file for you, but it's cleaner and less confusing to explicitly close it yourself. Note that when writing files, they are generally closed automatically after writing is complete.

In this particular simple example, we can compress the syntax to

```
read psf card name myfile.psf
```

In this case, CHARMM will assign a unit number to myfile.psf, open it, read it, and close it all with one command.

Writing works similarly to reading, the file is opened written to, and closed. For example, one may wish to write out the current coordinate set. This can be done as follows:

```
open unit 11 write card name new-coordinates.crd
write coor card unit 11
* title of the coordinate file
*
```

Note that in this case, we did not close the file as CHARMM closes it automatically once the writing is done.

To give an example of opening an unformatted/binary file, consider the case of opening a file to write a coordinate trajectory for dynamics (trajectories are discussed further in the dynamics and analysis sections):

```
open unit 20 write unfo name my-trajectory.trj
dyna ... iuncrd 20 ...
```

In this case no title is written (after all the file contains binary data) and the unit number is passed to the `IUNCRD` option to the `DYNAmics` command, which specifies the unit number on which to write out the coordinate trajectory.

A complete treatment of how CHARMM handles file I/O can be found in io.doc [1]. Full-fledged examples are given on the complete example page, and an example of opening trajectory files is given on the analysis page.

Main and Comparison Coordinate Sets

During a run, CHARMM stores two sets of coordinates; the coordinates being acted upon are called the main set (`MAIN`), but there is another set available called the comparison set (`COMP`). This is extremely useful for measuring how much atoms have moved around during a particular action, for example...

```
! copy the contents of the main set into the comp set
coor copy comp

! do something to move the coordinates such as
! minimization
.
.
.

! get the difference between the main and the
! comparison set and save it in the
! comparison set
coor diff comp

! print out the differences
print coor comp
```

You can also use

```
coor copy
```

to copy the `COMP` set into the `MAIN` set and

```
coor swap
```

to swap the main and comparison sets.

One important caveat is that some calculations such as molecular dynamics may overwrite the `COMP` coordinate set. If in doubt, you should save the coordinates to file and then read them back into the `COMP` set, e.g.:

```
write coor card name temp.crd ! write set out to a file

! do something that might overwrite the comp set, e.g. dynamics
.
.
.

! re-fill the COMP set from the file we wrote before
read coor comp card name temp.crd
```

This way, you will be sure you have the coordinates that you're expecting in `COMP`!

Atom Selection

One of the most useful features of CHARMM is its powerful atom selection functionality. This can be used to modify the effects of certain commands, making them apply to a subset of the atoms or the whole system. For example, if you want to write only some of the coordinates out, there's a command for that.

All atom selections are done with the `SELEct` subcommand. `SELEct` is not a command itself, but is often used as an integral part of other commands to determine which atom(s) will be operated upon (there will be plenty of examples of this as we move through the tutorial). The basic syntax for `SELEct` is:

```
sele <criteria> end
```

Where <criteria> is the specification of which atoms you want selected (this tutorial will give plenty of examples of which criteria you can use). When experimenting with the SELEct command, it is helpful to put the subcommand SHOW directly before the END statement. This will tell CHARMM to list which atoms were actually selected, which is very helpful for debugging SELEct statements!

A full description of atom selection is contained in select.doc [2].

Controlling which atoms are selected: BYREsidue and BYGRoup

Usually CHARMM will only mark those atoms which match the selection criteria. However, sometimes you want to select all atoms in a group or residue in which one atom matches the criteria. The `.byres.` and `.bygroup.` key words (note the leading and trailing dots) allow you to do this. For example:

```
sele .byres. bynum 12 end
```

will select all atoms in the same residue (`.byres.`) as atom number 12 (the `bynum` factor is described later on, but its use should be self-evident here).

Basic operators: .AND., .OR., and .NOT.

It is possible to use basic boolean logical operators [3] (.AND., .OR., and .NOT.: the periods are optional) with atom selections, and they behave exactly as one would expect. For example, if you need to select atom numbers 12 and 15, this can be done by:

```
sele bynum 12 .or. bynum 15 end
```

A commonly seen mistake is to do:

```
sele bynum 12 .and. bynum 15 end
```

This selection will return no results because it is impossible for an atom to be both number 12 and number 15 at the same time!

Likewise, you can select all atoms except for atom number 12 by doing:

```
sele .not. bynum 12 end
```

Basic factors: atom, segment, and residue types

There are a number of keywords that let you select an atom based on a particular characteristic. We've already seen one of these, the `bynum` command which select an atom based on (surprise, surprise) its number. An advanced feature of the `bynum` token is that you can select a range of numbers by seperating them with a colon, for example:

```
sele bynum 12 : 20 end
```

selects atoms 12-20. One of the most important factors for day to day use is the ATOM token. The syntax is:

```
sele atom <segment ID> <residue ID> <type> end
```

Any of the segment ID, residue ID, or type can use wildcards. A "*" matches any string of characters, a "#" matches any string of numbers, and "%" and "+" match a single character or number, respectively. So for example:

```
sele atom A * C* end
```

will select all carbon atoms in all residues of segment A (note that C* matches C, CA, CT, etc.).

It is possible to select by the segment ID, residue ID and type individually with the `SEGId`, `RESId`, and `TYPE` factors, so for example:

```
sele segid A .and. type C* end
```

will provide equivalent results to the previous atom selection.

Another example of what you can do is select a range of residues. Suppose, for example, residues 22 through 48 of a protein make up an alpha helix and you want to select all of them to perform some action, you can do:

```
sele resid 22:48 end
```

The final basic factor that is commonly used is the `RESName` criteria, which selects atoms based on their residue name. For example:

```
 sele resn GLY end
```

will select all atoms in all Glycine residues.

Advanced operators: AROUnd and BONDed

It is also possible to select atoms based on their spatial or bonded relationship with other atoms. To select atoms within a given distance of a group of atoms, one can use the .AROUND. followed by a real number immediately after another factor. For example:

```
sele resid 10 .around. 5.0 end
```

will select all atoms within 5 angstroms of residue number 10.

Likewise, you can select the atoms that are bonded to a particular atom with the `.BONDED.` modifier:

```
sele atom * * H* .and. .bonded. type C* end
```

will select all hydrogens bonded to a carbon atom.

Additional factors

There are several other keywords that you can use to select atoms

- INITial: This keyword selects all atoms that do not have coordinates initialized (CHARMM sets uninitialized coordinates to 9999.9999).
- HYDRogen: selects all of the hydrogen atoms in the system
- CARBon: selects all of the carbon atoms in the system

Properties

The PROPerty key word allows for atom selection based on CHARMM's SCALar properties, a full list of which may be found in scalar.doc [4]. Both the main and comparison coordinate and weighting arrays are permitted. For example:

```
sele property wcomp .eq. 1 end
```

would select those atoms having a weight of 1 in the comparison weighting array. Other scalar values that may be selected include the X, Y, and Z coordinates (called XCOMP, YCOMP, and ZCOMP for the comparison coordinate set), the mass of the atom (MASS), and several others. The complete list is in select.doc [2].

One interesting note is that you can use this to print atoms that move more than a particular amount during a simulation, e.g.

```
! copy current coordinates to the comparison set
coor copy comp

! minimize, or do dynamics or whatever

! compute the differences between the new coordinates and the ones saved previously,
! storing these differences in the comp set
coor diff comp

! print out those atoms that are displaced more than 10 angstroms
! in any directions
define bigmove sele prop xcomp .gt. 10 .or. prop ycomp .gt. 10 -
 .or. prop zcomp .gt. 10 show end
```

There is a new command in the example above -- DEFIne. DEFIne allows you to associate a name for an atom selection and then use it later, for example:

```
define sel1
.
.
.
print coor select sel1 end
```

In the above example the <big nasty long atom selection> can be used multiple times without having to retype it. To give a concrete example, if you are going to do operations repeatedly on all atoms withing 5 angstroms of residues 10 through 20, you can do::

```
define critreg select resid 10:20 .around. 5 end

... ! do some stuff
```

```
coor stat select critreg end ! get coordinate statistics
                             ! of atoms in critreg only
```

Note that after defining `critreg` it is necessary to encapsulate it within a `SELEct` statement, i.e. `SELEct critreg END`

Variables in CHARMM

User-settable Variables

So far, the CHARMM scripting language seems to be a concatenation of individual commands. It does contain, however, most (if not all) elements of a (imperative) programming language (even though it is not an extremely comfortable one). One key element of a programming language is the ability to set and manipulate variables. In the context of running CHARMM, it is extremely useful to pass certain variables into the program from the outside. Run the following miniscript (we name it title2.inp)

```
* Example of passing a variable from the command line
* The variable myvalue was initialized to @myvalue
*
stop
```

by executing

```
charmm_executable myvalue=500 < title2.inp
```

and study the output (you could also state `myvalue:500`): Before the title is echoed, you see that an argument was passed

```
Processing passed argument "myvalue:500"
Parameter: MYVALUE <- "500"
```

and in the echoing of the title, @myvalue is replaced by the value assigned to the variable *myvalue*, i.e., 500 in our case. The little example highlights something to keep in mind. The first thing that is done when a line in a script (containing a title or a command) is parsed, is to scan for @variables, which are then replaced by their value (no variable replacement is done in comments!).

Substitution Variables

For completeness sake, a preliminary comment on a second type of variable is needed. Many CHARMM commands, aside from producing more or less direct output, place some key values in "internal" variables. E.g., the energy of a system calculated with the most recent `ENERgy` command is put into a variable named `ENER`. To avoid name clashes with variables set by users, the values of these variables can be accessed by preceding the variable name by a question mark "?". For example, if CHARMM encounters (in a title or in a command) the string `?ENER`, it will attept to replace it with the energy value from the most recent `ENERgy` command. This is quite handy, and we'll see many examples later on, once we have reached the stage where we can use

CHARMM to work with biomolecular systems.

Loops and flow control

The beginning of loops in CHARMM is marked by a `label`

```
label someplace
```

basically anywhere in a CHARMM script. If (before or after) that `label` CHARMM encounters a

```
goto someplace
```

command, the script continues from "`label someplace`"; i.e., control is transferred to that line of the script. The `label/goto` tandem, combined with `if`s, make it possible to construct almost arbitrary control structures, in particular loops (see below). First, however, two simpler examples.

Example 1: We just showed how to provide a default value to a variable that is expected to be passed from the command line. Obviously, this is not possible or sensible in all cases. The script `simplemini.inp` expects two parameters, `@nstep` and `@system`. While it makes sense to set a default for the former (if the user forgets to specify a value), the second variable has to be set to a sensible value. Thus, we may want to check whether `@system` is initialized, and if not, write a meaningful error message and exit the script. This is done by querying `@?system` -- the `@?variable` operator is 1 if `@variable` is set and 0 if it is not. The following script fragment shows how to use this

```
* The title
* ...
*
if @?nstep eq 0 set nstep 100 ! provide default for nstep
if @?system eq 0 goto systemerror

... ! continue with normal script up to normal

stop

label systemerror

echo You need to pass a value for variable system
echo
echo Aborting execution

stop
```

Example 2: It was pointed out that many CHARMM scripts do identical things (read rtf, params, psf, coords, some more stuff) whereas the "genuine" work part (minimization, molecular dynamics, etc.) consists of just a few lines. Thus, the first twenty to forty lines of many CHARMM scripts contain essentially the same commands. Using `label/goto` statements one

can reorganize such scripts, so that the boring stuff is placed after a label at the end of the scripts. Thus, the more unique parts of the script can be seen earlier in the file (note that *stream files provide a better way of accomplishing the same goal*, but the technique may prove useful in other cases). Take a look at simple_mini2.inp [5]. After checking whether `@nstep` and `@system` were passed from the command line, command is transferred to `label setupsystem`. The corresponding code is at the end of the script; from there, control is transferred back to the beginning of the script (`label fromsetupsystem`). The first interesting line (`mini sd`) moves up about 15 lines; one can understand a bit more quickly what the script does.

Example 3: As mentioned above, `label` statements provide for a simple way to make a loop. The following example shows a loop that repeats an action ten times over

```
set i = 0

label beginloop
incr i by 1
! ... this is the body of the loop ...

if @i .lt. 10 then goto beginloop

! commands below this point are executed
! after the loop finishes
```

Pay attention to how the `INCRement` command is used to add 1 to the value of `@i` for each iteration of the loop.

Constraints and Restraints

Basic overview

CHARMM has the ability to constrain or restrain atoms using the CONStraint command. This can be used to ensure that atoms stay close to a given point or another atom during a simulation. As an example, it is often desirable to constrain or restrain protein backbone atoms during minimization and only optimize the positions of the various side chains while the basic trace of the backbone remains fixed. Constraints and restraints can also be used to reduce high-frequency motions in the system as in the case of SHAKE, which constrains bond lengths.

The basic difference between a constraint and a restraint is that a constraint completely removes the given degree(s) of freedom from the system while a restraint retains them but applies a penalty function to the energy if the atom(s) involved move away from the desired configuration. More information can be found in the cons.doc [6] CHARMM documentation file. Below we describe three of the most commonly seen restraints.

Harmonic restraints

Harmonic restraints are used to hold atoms near a given location by applying a harmonic penalty function as they move away from the desired location. There are three types of harmonic restraints: absolute, relative, and bestfit.

- Absolute restraints require the atom(s) to stay near a given cartesian position. A set of reference coordinates must be given. By default, the current position in the main coordinate set is used, but the comparison set may also be used. It is not possible to specify the reference coordinates directly in the command; they must be in either the main or comparison coordinate set.
- Bestfit restraints are similar to absolute restraints, but rotation and translation are allowed to minimize the restraints energy.This makes it useful for holding inter-atom geometry intact while allowing for block motion. Second derivatives are not supported for bestfit restraints.
- Relative positional restraints are similar to bestfit restraints, but there are no reference coordinates. It is used to force two sets of atoms to have the same shape

One important caveat is that no atom may participate in more than one restraint set. It is possible in the case of `ABSOLUTE` restraints to specify the exponent for the harmonic restraint (i.e. where the penalty function is linear, quadratic, cubic, quartic, etc.) and to scale the restraint separately in the x, y, and z directions.

Some examples are:

```
define backbone sele type N .or. type CA .or. type C end
cons harm absolute sele backbone end
```

restrains the protein backbone based on its current coordinates. This can be handy in minimization where you don't want to change the basic shape of the atom from that in the crystal structure.

```
cons harm absolute expo 4 xscale 1.0 yscale 0.5 zscale 0.0 sele
backbone end
```

restrains the backbone using a quartic penalty function in the x direction. The restraint energy in the y direction is halved, and the restraint is not applied in the z direction. Such an unusual restraint would probably not be used much in normal circumstances.

```
cons harm bestfit sele resid 20 end
```

restrains residue #20 to its current geometry. For example, if you have a ligand whose internal conformation should remain rigid but that is allowed to rotate and translate, a restraint like this could be used.

```
cons harm relative sele segid 1 end sele segid 2 end
```

restrains segment 1 to maintain the same geometry as segment 2.

Note that the `MASS` keyword may be used to mass weight the restraints.

All harmonic constraints can be cleared with the command

```
cons harm clear
```

Distance Restraints

Distance restraints require two atoms to remain within a given proximity of each other. The desired distance, force constant, and exponent must be given to the RESDistance command. For example:

```
resd kval 10.0 rval 5.0 ival 2 sele atom 1 1 CA end sele atom 1 2 CA
end
```

restrains the alpha carbons of residues 1 and 2 of segment 1 to be 5 angstoms apart, with a quadratic penalty function. The functional form of this energy term is:

$E_{resd} = Kval(r - rval)^{ival}$

This energy term is very similar to the bonded energy term!

For individual pairs of atoms, NOE restraints can also be used (this is very good for distances which depend on one another). Discussion of this functionality is beyond the scope of this tutorial, but you can find more information in the NOE sections of cons.doc [6].

SHAKE

SHAKe is a constraint that fixes bond lengths. It can be used to fix angles as well, but this is not recommended. It is most widely used to fix the lengths of bonds involving hydrogens, since these tend to vibrate at very high frequencies, which can lead to numerical imprecision during simulations. The use of SHAKe on bonds involving hydrogens allows for numerical integration at a 1-2 fs step size during dynamics.

To apply shake to all bonds involving hydrogens do

```
shake bonh param sele all end
```

Note that the `sele all end` is the default atom selection when none is given (i.e. it's a bit redundant here). The `PARAm` keyword uses the values in the parameter file as the optimal bond lengths. If it is not used, the current distance from the main coordinate set is used (or the current distance from the comparison set if the `COMP` keyword is given in place of `PARAm`).

Debugging CHARMM scripts

Debugging via prnlev

The `PRNLev` command controls how verbose CHARMM is in its output. The default value is 5, the minimum level is 0 (which will cause CHARMM to be virtually silent), and the maximum level is 9. Higher values of `PRNLev` produce more output. This command also controls which node prints output in a parallel job. The default is that all nodes produce output, which is usually not desirable. The `NODE` subcommand to `PRNLev` restricts print out to a single node, e.g.:

```
prnlev 6 node 0
```

Sets the `PRNLev` to 6 and states that only the first node (index 0) should produce output.

At higher values of PRNLev, CHARMM will detail exactly which energy terms are being computed and give details about which subroutines are being called. This can be very helpful for debugging incorrect results.

The `BOMBlev` and `WARNlev` commands are similar, except that they control the severity of errors which will cause CHARMM to abort execution or issue a warning. Errors and warnings range in severity from +5 (least severe) to -5 (most severe). One common mistake made by novices is to set `BOMBlev` too low. Level 0 errors and below are serious problems that may affect the validity of numerical results from the runs. Level 0 and -1 errors should only be accepted when the user understands what they mean and has determined that they will not affect the results. Errors at the -2 level or below are very severe and generally mean that the results of the run are invalid. In no cases should the BOMBlev ever be set below -2. In general, it is always safe and recommended to set `BOMBlev` to 0 (although it may need to be temporarily lowered for some operations).

Printing data structures

CHARMM provides the functionality to print out data structures. For example, the commands:

```
print psf
print coor
```

would print out the current protein structure file and coordinate set. Parameters can be printed in the same way, however in many cases most of the parameters that are read in do not apply to the structure being studied. The command

```
print param used
```

only prints out those parameters that are currently referenced by the PSF.

It is also possible to display the current forces acting on each atom, for example:

```
coor force comp
print coor comp
```

Puts the forces into the COMParison coordinate set and then prints them out. It is possible to combine these with a `PROPerty` based atom selection (described above) to print all forces over a certain magnitude, e.g.:

```
coor force comp
print coor comp sele prop abs x .gt. 500.0 .or. prop abs y .gt. 500.0
.or. prop abs z .gt. 500.0 end
```

will print all atoms who have at least one of their force components greater than 500 in absolute value.

If a more detailed analysis is needed (for example, if the force on one of the atoms is blowing up for an unknown reason), it is possible to print out all of the individual terms of the first field, this is done by the `ANAL TERM` command. By default, `ANAL TERM` only prints out the bonded interaction terms, `ANAL TERM NONBonded` will print all of the terms. It is possible to see in

which order the energy functions are called by setting the `PRNLev` to 9.

References

[1] http://www.charmm.org/documentation/current/io.html
[2] http://www.charmm.org/documentation/current/select.html
[3] http://en.wikipedia.org/Boolean_logic
[4] http://www.charmm.org/documentation/current/scalar.html
[5] http://www.mdy.univie.ac.at/charmmtutorial/simple_mini2.inp
[6] http://www.charmm.org/documentation/current/cons.html

The Energy Function

A brief introduction to force fields

This section of the tutorial deals with how CHARMM calculates the potential energy of a molecular system and its first derivative (the force or gradient). CHARMM supports methods for evaluating molecular systems at varying levels of theory; this section introduces the purely classical CHARMM force field (which is distinct from the program itself). CHARMM the program includes support for several other force fields as well as mechanisms to calculate *ab initio* and semiempirical quantum mechanical energies and for coarse-grained modeling. These topics are beyond the scope of this tutorial.

In computational chemistry, a force field consists of two things: (1) a functional form defining how to compute the energies and forces on each particle of the system, and (2) a set of parameters that define how positional relationships between atoms determine their energy (e.g. how the bonded potential energy changes as a bond stretches or contracts, that is, as the distance between two bonded atoms increases or decreases).

A full treatment of force fields and their development is beyond the scope of this tutorial. Interested readers are encouraged to consult the Wikipedia entry on force fields [1] and the molecular modeling references given in the introduction (particularly chapter 2 of Becker *et al.*).

Extended background on calculating energies and forces with CHARMM

Energy calculation

The force field terms

The molecular mechanics force field used by CHARMM and similar computer programs is a simplified, parametrized representation of reality. Interactions between chemically bonded nearest neighbors are handled by special terms, the so-called bonded energy terms: e.g., bond stretching, angle bending, dihedral and improper dihedral energy terms. Interactions beyond (chemically bonded) nearest neighbors are represented by the two so-called non-bonded energy terms (Lennard-Jones (LJ) and Coulomb interactions). Very early versions of the force-field contained a special term for hydrogen bonds; the code is still in CHARMM, but it is, in general, not used anymore.

Thus, the energy function of the latest CHARMM "param22"/"param27" force field has the following form (J. Phys. Chem. B, 1998, 102, 3586; J. Comput. Chem., 2004, 25, 1400; J. Comput. Chem., 2000, 21, 86, ibid. 105ff):

$$U_{CHARMM} = U_{bonded} + U_{non-bonded}$$

where U_{bonded}consists of the following terms,

$$U_{bonded} = U_{bond} + U_{angle} + U_{UB} + U_{dihedral} + U_{improper} + U_{CMAP}$$

with

$$U_{bond} = \sum_{bonds} K_b(b - b^0)^2,$$

$$U_{angle} = \sum_{angles} K_\theta(\theta - \theta^0)^2,$$

$$U_{UB} = \sum_{Urey-Bradley} K_{UB}(b^{1-3} - b^{1-3,0})^2,$$

$$U_{dihedral} = \sum_{dihedrals} K_\varphi((1 + \cos(n\varphi - \delta)),$$

$$U_{improper} = \sum_{impropers} K_\omega(\omega - \omega^0)^2,$$ and

$$U_{CMAP} = \sum_{residues} u_{CMAP}(\Phi, \Psi)$$

$U_{non-bonded}$ consists of two terms,

$$U_{non-bonded} = U_{LJ} + U_{elec}$$

with

$$U_{LJ} = \sum_{nonb.pairs} \varepsilon_{ij} \left[\left(\frac{r_{ij}^{min}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{ij}^{min}}{r_{ij}} \right)^{6} \right],$$

$$U_{elec} = \sum_{nonb.pairs} \frac{q_i q_j}{\epsilon r_{ij}}.$$

The values of the naught terms above (e.g. b^0 in U_{bond}, θ^0 in U_{angle} etc.), the various force constants (K_b, K_θ etc.), as well as partial charges q_i and LJ parameters (ϵ, r^{min}) are taken from the force field parameters. The atomic LJ parameters ϵ, r^{min} are combined with the appropriate mixing rules for a pair of atoms i, j. In the current CHARMM force fields, these are $\epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j}$ (geometric mean) and $r_{ij}^{min} = \left(r_i^{min} + r_j^{min}\right)/2$ (arithmetic mean). Note that other force fields may use other mixing rules, such as the geometric instead of the arithmetic mean for r_{ij}^{min}

Both non-bonded terms are normally modulated by a shifting or switching function (however, periodic boundary conditions may be used in place of this modulation, particularly for electrostatics), see below.

The above terms are the standard terms of molecular mechanics force fields just described, with two exceptions that require some explanation. The first is the so-called "Urey-Bradley" (U_{UB}) term in addition to the standard bond stretching U_{bond} and angle bending terms U_{angle}. The Urey-Bradley term is a harmonic term in the distance between atoms 1 and 3 of (some) of the angle terms and was introduced on a case by case basis during the final optimization of vibrational spectra. This term turned out to be important "for the in-plane deformations as well as separating symmetric and asymmetric bond stretching modes (e.g., in aliphatic molecules)" (J. Phys. Chem. B 1998, 102, 3586).

A more recent addition to the CHARMM force field is the CMAP procedure to treat conformational properties of protein backbones (J. Comput. Chem. 2004, 25, 1400). The so-call CMAP term is a cross-term for the ϕ, ψ (backbone dihedral angle) values, realized by grid based energy correction maps. It can, in principle, be applied to any pair of dihedral angles, but is used in the current CHARMM force field to improve the conformational properties of protein backbones. Several programs claiming to be able to use the CHARMM force field(s) lack support for this term, although it is being incorporated into an increasing number of programs.

We note in passing that CHARMM can compute energies and forces with several non-CHARMM forcefields, including converted versions of the AMBER and OPLS-AA forcefield, but also more generic force fields, such as the Merck force field. [**TODO: add refs**] Use of these force fields is beyond the scope of this tutorial.

Calculation of non-bonded energies: cut-offs, shifting, switching etc.

The most time consuming part of an energy or force computation is the evaluation of the LJ and electrostatic interactions between (almost) all pairs of particles.

While straightforward in principle (and in practice energy/force calculation in CHARMM is as simple as

```
ENERgy
```

issuing a single command), numerous options influence the calculation of non-bonded interactions, and so it is very important that you understand their meaning and have the necessary background to understand the rationale for them in the first place.

To compute the interaction of N atoms with each other, one needs in principle N ×N steps. Bonded interactions only involve next neighbors, so they are cheap in terms of computer time. For the non-bonded energy (LJ, electrostatic) it is customary to introduce a cut-off beyond which interactions are ignored. This is a reasonable approximation for the van der Waals (= LJ) interactions, which decay rapidly for large distances, but a bad one for electrostatic interactions which go to zero as $1/r$. For periodic systems (the typical situation found when simulating solvated proteins), the so-called Ewald summation method (usually employed in its fast incarnation, particle-mesh-Ewald (PME)) circumvents this particular difficulty; nevertheless, cut-offs are ever present in molecular mechanics energy/force calculations and their implications need to be understood.

Non-bonded exclusions: Before continuing on the subject of cut-offs, we need to introduce the concept of *non-bonded exclusions*. To avoid computing interactions between chemically bonded atoms (1-2, 1-3 and, possibly, 1-4 neighbors) with multiple different force field terms (specific bonded terms, as well as the "unspecific" LJ and electrostatic term), these pairs are excluded from the non-bonded energy calculation. When using the CHARMM force field, one never calculates non-bonded interactions between 1-2 and 1-3 neighbors. LJ and electrostatic interactions between 1-4 neighbors are calculated, but can be modified compared to more distant pairs (different LJ interactions, scaling of electrostatic interactions). The details depend on the specific force field used. E.g., in the older polar hydrogen force field ("param19"), electrostatic interactions and forces between 1-4 neighbors were scaled by 0.4, whereas in the current all-atom force field(s) ("param22", "param27"), electrostatic energies and forces are used as is. This scaling factor is controlled by one of the many options to the `ENERgy` command, `E14Fac`. Thus, for calculations with the old, polar-hydrogen "param19" force field, `E14Fac` should be set to 0.4; for calculations with the current "param22/27" force field it should be set to 1.0. **IMPORTANT HINT:** *E14Fac is an example of a user settable option which should* ***never*** *be set by a normal user*. When you read in the respective parameter file, the correct default value is automatically set. In fact, many of the options to the `ENERgy` command are of this type --- you should understand these, but, again, you should never change them (unless you are an expert and develop a new force field etc. in which case you won't need this tutorial!). We'll return to the issue of specifying all possible energy options as opposed to relying on defaults later on; unfortunately, this is less clear-cut than it ought to be.

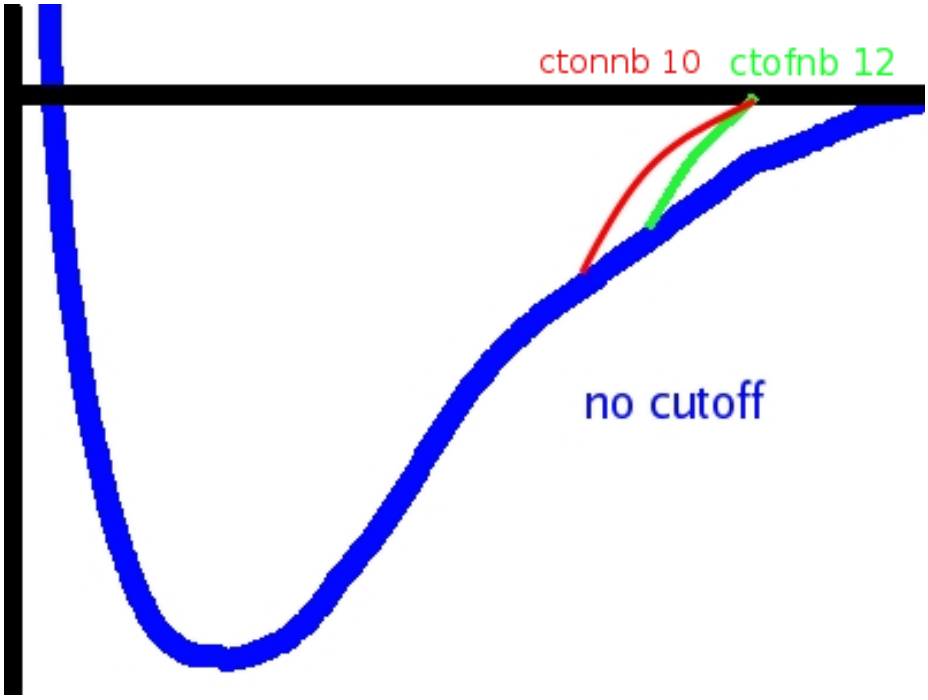
Back to cut-offs: The CHARMM keyword to set the cut-off for electrostatic and LJ interactions is `CTOFnb`. Thus, the statement

```
ENERgy CTOFnb 12.0 ! it's not recommended to have
                   ! a blanket cut-off, switching
                   ! or shifting should be used
```

discards LJ and electrostatic interactions between all pairs of particles that are greater than 12.0 angstroms apart (of course any non-bonded exclusions are always honored!). The straightforward application of a cut-off criterion as in the above (bad) example introduces a discontinuity that can adversely affect the stability of a molecular dynamics calculation. Consider a pair of atoms that are separated by approximately the cut-off distance. At one time-step they interact and contribute to the energy and forces in the system. Then they move relative to each other by a tiny amount and suddenly they are more than the cut-off distance apart. Although very little has changed, they now do

not contribute to the energy and forces at all.

To avoid these abrupt changes, the LJ and electrostatic energy functions in CHARMM are (always) modulated by *switching* or *shifting* functions (it actually takes some tricks to coax CHARMM into doing, effectively, a plain cut-off for you!). For the full details, in particular the detailed functional forms of switching and shifting functions, see J. Comput. Chem. 1983, 4, 187; Proteins 1989, 6, 32; J. Comput. Chem. 1994, 15, 667. The effect of a switching function on a LJ interaction is illustrated by the following figure.



At `CTOFnb` 12., the interaction (and its derivative, i.e., the forces) has to be zero as shown. Up to an *inner* cut-off (`CTONnb`), interactions between the two particles are normal. Between `CTONnb` and `CTOFnb`, the normal interaction is switched off (hence the name) by a sigmoidal function. At distances greater than `CTOFnb`, interactions and forces are zero. The effect of a switching function is shown for two different values of the inner cut-off, 8 angstroms (red) and 10 angstroms (green).

A *shifting function* also modulates the normal interaction between two particles. The design goal is that the original interaction is modified so that it approaches zero continuously at the cut-off distance (`CTOFnb`); in addition, forces are required to vanish continuously as well at the cut-off distance. Visually, it looks as if the original interaction were "shifted" so that it became zero at the cut-off distance. *The actual function is more complicated because of the requirement of continuity of potential and forces at the cut-off distance!!* With shifting functions, only `CTOFnb` is meaningful (as the cut-off radius), whereas an inner cut-off (`CTONnb`) is ignored. Switching and shifting can be activated separately for LJ and electrostatic interactions; i.e., the keywords `VSHIft` and `VSWItch` control whether a shifting or switching function is applied to LJ interactions; similarly, `SHIFt` and `SWITch` choose between shifting and switching function for electrostatic interactions. To illustrate this by a realistic example, let's take a look at the options that correspond to the default force field choices for the current CHARMM all-atom force field (param22/27)

```
ENERgy SHIFt VSWItch CTOFnb 12. CTONnb 10.
```

The line indicates that electrostatic options are shifted to zero using a cut-off value of 12 angstroms (`CTOFnb`) and that LJ interactions are modulated by a switching function between 10 (`CTONnb`) and 12 angstroms (`CTOFnb`). Any interactions beyond 12 angstroms are discarded. Note that the `CTONnb` parameter is ignored for the shifted electrostatic interactions.

Beyond SHIFt/SWITch: In molecular dynamics, the determining factor are the forces, not the potential energy. Thus, it has been suggested to apply shifting/switching functions to the forces, rather than the potential. Such methods are available in CHARMM as well, driven by keywords `VFSWitch` (for LJ) and `FSWItch`/`FSHIFt` (for electrostatics). The `FSHIFt` option is usually the default in c27 and later (non-protein) force fields. For the gory details see nbonds.doc [2] of the CHARMM documentation. The recently introduced, so-called Isotropic periodic Sum (IPS) method provides yet another approach to handle cut-offs in a smooth, continuous fashion. However, all these options go beyond the scope of this tutorial.

GROUp vs. ATOM: A cut-off criterion can actually be applied in two ways. The default is to apply it on an atom by atom basis (keywords `ATOM`, `VATOm`, the option with V in front pertains to the van der Waals (=LJ) energy computation, that without to the electrostatic energy computation). In addition, there is the possibility of a group based cut-off (keywords GROUp, VGROup). In the CHARMM force field topologies, the partial charges of molecules / residues are grouped into small clusters that are usually neutral, or, for charged residues, carry a net charge of +/-1, +/-2 etc. When GROUp/VGROup is set CHARMM first computes the center of geometry for each of these charge groups. If the distance between the centers of geometry for a pair of groups are separated by more than the cut-off distance (`CTOFnb`), then no interaction between all atoms of the two groups is computed. By contrast, in the atom by atom case, some pairs of atoms in different groups might be inside and others might be outside the cut-off distance. Thus, in general the two options give different results. (Note: you cannot mix group and atom based approaches; i.e., the combinations ATOM/VGRO and GROU/VATO are illegal!) There are some theoretical arguments that would indicate that the group by group based cut-off criterion ought to be preferred. This is particularly the case if all groups have an overall charge of zero, which is the case, e.g., for water. Contrary to this, **it has been shown for the CHARMM parameters that the group by group based cut-off criterion gives inferior results**; in addition, it is much slower, explaining that ATOM/VATOm is the CHARMM default.

Some remaining options: Having said this, we can take a look at the full default options of the current all-atom force field

```
ENERgy NBXMod  5 ATOM CDIEl SHIFt VATOm VDIStance VSWItch -
CUTNb 14.0 CTOFnb 12.0 CTONnb 10.0 EPS 1.0 E14Fac 1.0 WMIN 1.5
```

The strings which you don't know yet are `NBXMod 5`, `CDIEl`, `CUTNB 14.0`, `EPS 1.0`, and `WMIN 1.5`. Of these, the only important one is `CUTNB` since this is a value you may indeed want to change. It is the cut-off radius used to generate the non-bonded pair list and is discussed in detail in the next section. `NBXMod 5` controls the handling of non-bonded exclusions (cf. above), and setting it to 5 means skipping 1-2, 1-3 interactions and treating 1-4 interactions specially. The `WMIN` distance of 1.5 is used to print out interaction pairs that are closer than 1.5 Å. At shorter distances, very high LJ repulsion would result, and it's useful to be warned since such pairs could cause problems in MD simulations. `CDIEl` and `EPS` are two related options, which nowadays are a left-over from a distant past and should be left alone. A brief explanation: The solvent (water) is a very important determinant when studying the properties of biomolecules. When computers were slower, one often tried to avoid simulations in explicit solvent, and the two parameters can be used to mimic the influence of solvent. Water has a high dielectric constant (DC), which screens electrostatic interactions. To mimic the presence of water in gas phase simulations, CHARMM allows the user to change the value of the DC (ϵ) by the `EPS` keyword (default value is 1), i.e., the electrostatic energy is computed according to $q_1 q_2/(\epsilon r)$. The `CDIEl` / `RDIEl` option is another attempt to mimic water in gas phase simulations. `CDIE` stands for constant (uniform) dielectric, `RDIE` means a distance-dependent dielectric, $\epsilon = \epsilon(r) = \epsilon \cdot r$, where ϵon the right hand side means the number set by `EPS`. Effectively, when you use `RDIE`, you compute a modified electrostatic energy according to $q_1 q_2/(\epsilon r^2)$. *Nowadays when using explicit solvent, you should always use `CDIE` and leave `EPS` set to 1 (i.e., leave them at the default values!).* Should the use of explicit solvent be too expensive computationally, CHARMM nowadays offers several *implicit solvent* models.

A final remark on ENERgy options: So, what non-bonded options should you set? In fact, if you read in the default param22/27 parameter file (e.g, par_all27_prot_na.prm), the options just discussed get set for you automatically. At first glance, there is no need to set any of these options explicitly; unfortunately, however, this turns out not to be true. This does not mean that you should modify the default options "just for fun". Remember, changing some values (e.g. CDIE or EPS, or replacing (V)ATOM by (V)GROUP , or changing E14Fac) is outright dangerous/false. You may want, e.g., to use some of the force based shifting/switching methods, but you should be experienced enough to understand thoroughly what you are doing and why you are doing it since the parameters were developed with the options shown above. In practice, what you'll want to change/add is to request the use of Ewald summation (PME) for solvated systems (see below); some PME related options need to be adapted depending on your system size. Similarly, for performance reasons you may want to choose a different non-bonded list cutoff `CUTNB` (but we need to understand more about non-bonded lists first). Normally, you would not want to change the cut-off radii `CTOFnb`, `CTONnb`, since they are part of the parameterization. *Unfortunately, this is where it becomes tricky:* Suppose you work with the param22/27 parameters, and, thus, correct default non-bonded options have been set. You decide (for reasons that will become clear in the next subsection) to increase CUTNB from 14 to 16 Å. Thus, you specify

```
ENERgy CUTNB 16. ! WARNING: may not do what you want!
```

and assume that everything else is left alone. Unfortunately, for the above CHARMM command another default mechanism kicks in. If you change `CUTNB`, but do not set `CTOFnb` / `CTONnb` explicitly, the latter get changed according to CTOFnb = CUTNB - 2. and CTONnb = CTOFnb -2., hence you would suddenly be calculating with CUTNB 16. CTOFnb 14. CTONnb 12., which likely is not what you had in mind. It is, therefore, a bit dangerous to rely on defaults set by the parameter file. Although unsatisfactory, we therefore recommend to set the non-bonded options *explicitly* before doing any real work. The respective energy line should be identical to the defaults set in the parameter file, with the exception of individual parameters you might want to change, such as a modified `CUTNB` or replacing `SHIFt` by `FSHIft`.

In addition, there is one case were you *have* to change `CTOFnb` (and hence `CTONnb`): If you simulate small periodic systems, the minimum image criterion dictates that the cut-off radius must be smaller or equal to half the box-length (cubic PBC). For `CTOFnb 12.`, this means that your cubic periodic box should have a side length of at least 24 Ang. Thus, for smaller systems, `CTOFnb` (and, hence, `CTONnb`) have to be reduced. Interestingly, some known workers in the field (notably W. van Gunsteren and his group) find this such a bad choice (viz. reducing the cut-off radius below the force field default), so that the smallest boxes they use always have side lengths of twice the default cut-off radius of their force field. (Again, for the CHARMM all atom force field this would mean no boxes smaller than 24 Å!)

Non-bonded lists

Background on non-bonded lists

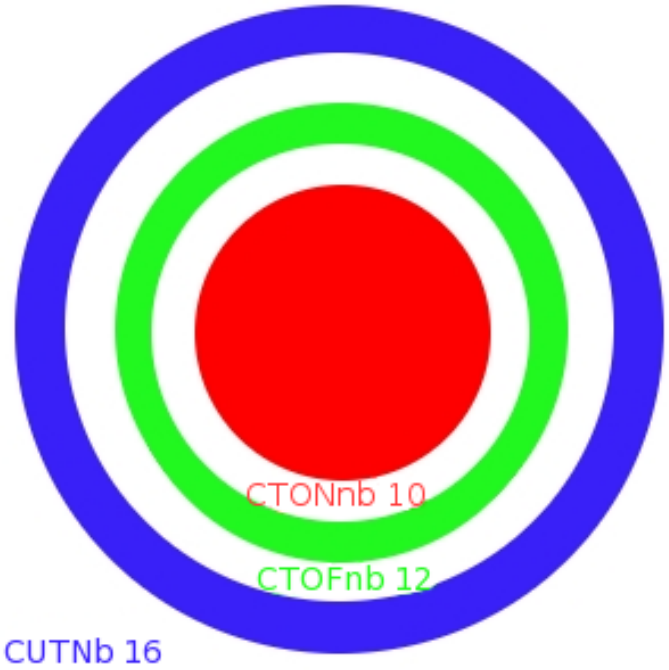
In molecular mechanical energy/force calculations a cut-off criterion is normally used to limit the number of non-bonded interactions. Energy minimizations and especially molecular dynamics simulations require hundreds and thousands (if not millions) of energy/force calculations. To avoid checking for excluded pairs (1-2, 1-3 and, possibly, 1-4 neighbors) and, much more importantly, the cut-off criterion during every single energy calculation, lists are generated that contain all pairs for which the non-bonded interactions really have to be computed. This is an unnecessary overhead for a single calculation, but saves a lot of time during repeated computations.

In fact, any energy calculation in CHARMM consists really of two steps: (1) Calculation of the non-bonded list, (2) calculation of energy and forces using this non-bonded list. The time saving in repeated energy/force calculations (e.g., minimization, MD) comes from the fact that the non-bonded list is not generated for every energy calculation, but instead the list is kept for several consecutive steps (energy calculations). Now suppose the non-bonded list were

generated with the cut-off radius used in the energy calculation (CTOFnb). The forces calculated in the first energy calculation are used to generate new particle positions (regardless whether we are doing minimization or MD). Because of the movement of the atoms, the original non-bonded list may not be correct anymore, i.e., it may contain pairs that are separated by more than CTOFnb and, worse, there may now be pairs less than CTOFnb apart for which no interactions are computed since they are not part of the list. For this reason, the non-bonded list should always be generated with a cut-off criterion/radius larger than CTOFnb, and for this reason CHARMM provides the separate CUTNB parameter used exclusively in list generation. With CUTNB > CTOFnb (by at least 1-2 Å), we can be certain that the non-bonded list is valid at least for a certain number of steps. This now explains the hierarchy of cut-off options in the param22/27 ENERgy example above, where we had

```
ENERgy ... CUTNB 14. CTOFnb 12. CTONnb 10.
```

These are the recommended cut-off values for the standard CHARMM force field! CUTNB controls the list generation, whereas CTOFnb and CTONnb are used for the energy calculation as described above (CTONnb only for those interactions modulated by a switching function). Bear in mind that when periodic boundary conditions (PBC) are used for electrostatics, CTONnb and CTOFnb only affected the van der Waals energy calculation. Graphically, the relationship between the different cut-offs can be illustrated as in the following scheme (where CUTNB 16. instead of 14. was chosen):



The difference between CUTNB and CTOFnb, sometimes referred to as "skin", governs the frequency with which the non-bonded list needs to be updated. Provided the update frequency is appropriate, the choice of CUTNB is a true choice for the user, which should not affect the results, but which may affect performance. A larger skin means more work for a single non-bond list generation, but the possibility of a larger update frequency. The respective performance from smaller / larger skins depend on the computer hardware, the system size and whether the computation is carried out in parallel or not. No general rule can be given, but frequently larger skins (e.g., 4 Å) perform somewhat better than the default skin size of 2 Å; for large scale production calculations it definitely is worth the time to experiment and benchmark to find an optimal value for CUTNB.

A natural question that arises from this discussion is how often the non-bonded list is updated. CHARMM allows the user to specify an interval at which the list is updated, or the program can update the list automatically as needed. If the former option is chosen, the keyword INBFrq is used, and INBFrq n, with n a positive integer tells CHARMM to update the non-bonded list every n energy calculations. The correct choice of n is left to the user and depends on the size of the skin (cf. above). Alternatively, INBFrq -1 (or any negative integer) tells CHARMM to

watch the particle positions and update the non-bonded list automatically. The algorithm is guaranteed to err on the side of safety, so with `INBFrq -1` one should always have a correct non-bonded list. For quite some time, `INBFrq -1` (= heuristic update) has been the default (the old default of +50 was horribly inadequate and almost guaranteed wrong results!); nevertheless, this is one parameter that there is no harm in setting explicitly.

Finally, as is so often the case, CHARMM offers more than one algorithm to construct non-bonded lists; all of them, however, deliver a list in the same format (so for the energy routines it is irrelevant how the non-bonded list was generated!). Only two will be mentioned here. The old, default method is labeled `BYGRoup`, and `BYGR` is the (optional) keyword that chooses this method (again, it's the default). When performance is needed (large systems, running on a parallel machine), then one should use `BYCB`. It supports normal MD of solvated systems under periodic boundary conditions, but cannot be used for some more advanced usage scenarios. Also, BYGR can generate non-bonded lists suitable for group and atom based cut-offs; BYCB only supports atom based cutoffs.

What the ENERgy command really does -- or how to break ENERgy into its individual steps

From the above section, it should be clear that the `ENERgy` command is actually doing more than computing the energy since it also takes care of computing the non-bonded list. In fact, it is relatively smart and computes a non-bonded list if none exists (or if it is deemed outdated), but uses an up to date non-bonded list, should one be available. Thus, the `ENERgy` command is actually encapsulates a lot of the core functionality of CHARMM. Not only does invoking it cause CHARMM to calculate the total energy of the system and the forces acting on each atom, but it also makes CHARMM take care of any necessary bookkeeping work such as regenerating the non-bond and image atom lists (we'll talk more about image atoms below).

If you look again at the options we have discussed so far, then they fall into two groups, (1) those that control details of the energy calculation (e.g., `CTOFnb`), and (2) those thar control the non-bonded list generation (`CUTNB`, `INBFRQ`, `BYGR`/`BYCB`). If you go beyond a single point energy calculation (minimization, MD), then you have a third class of options controlling details of the minimization or MD.

The various steps hiding underneath the ENERgy command can actually be broken up; we show this for pedagogical purposes; also, the availability of the `NBONds`, `UPDAte` and `GETE` commands is occasionally useful in practice as well. `NBONds` parses and stores the energy related options, as well as list generation options. The `UPDAte` command generates the actual list (optionally, one could also here specify list generation and energy options). Finally, `GETEnergy` just computes the energy, the existence of a *valid* non-bonded list is assumed (otherwise your CHARMM job will crash). First, the standard way using ENERgy

```
! read RTF file
! read param file
! read PSF
! read coordinates

! compute energy of system, explicitly specifying the (default)
! options
ENERgy NBXMod  5 ATOM CDIEl SHIFt VATOm VSWItch -
CUTNb 14.0 CTOFnb 12.0 CTONnb 10.0 EPS 1.0 E14Fac 1.0 WMIN 1.5
```

Instead, the same can also be accomplished in steps:

```
! read RTF file
! read param file
! read PSF
! read coordinates
```

```
! parse non-bonded options
NBONds NBXMod  5 ATOM CDIEl SHIFt VATOm VDIStance VSWItch -
CTOFnb 12.0 CTONnb 10.0 EPS 1.0 E14Fac 1.0 WMIN 1.5 -
INBFrq -1 CUTNb 14. ! list options

! do generate list
UPDAte

! calculate "just" the energy
GETE
```

As mentioned, the `MINImiziation` and (molecular) `DYNAmics` commands understand all options controlling non-bonded list generation and non-bonded energy / force calculation, as well as options specific to the individual task (minimization, MD). In particular for DYNA, the cumulative number of options (list generation, energy calculation, control of MD) can easily amount to 30-50. Such a command line can be become quickly very confusing. Given that list generation and energy options are remembered, it is therefore recommended to split the setting of options as shown below:

```
...
! PSF and full set of coordinates have been read
! optionally, periodic boundary conditions (see below) have been set up
ENERgy <list options> <energy options>
DYNA <only dynamics specific options>
```

Instead of ENERgy, one could also use NBONds/UPDAte.

Periodic boundary conditions (PBC): IMAGe/CRYStal

The textbook view of PBCs

We trust that you have read some background material on molecular mechanics and molecular dynamics simulations, and, thus, expect you to have some familiarity with periodic boundary conditions (PBC). The following is intended as a refresher with emphasis on pointing out where things may not be so obvious for macromolecular systems.

Even with today's computers, the system sizes we can handle are microscopically small. Thus, without any special precautions we would be simulating "infinitesimal droplets", the properties of which would be dominated by surface effects. It is standard practice to work around this by employing periodic boundary conditions (PBC), i.e., by assuming that the central simulation system is surrounded in all spatial directions by exact copies (this assumes, of course, that you have a space-filling geometry; obviously, you can't have PBC for a spherical system. Anticipating CHARMM terminology, we refer to these surrounding boxes as images (image boxes). Unless otherwise stated, we'll assume a cubic simulation box (note, though, that CHARMM can handle any valid crystallographic space group for PBC!)

The role of the surrounding boxes (images) is to replace particles that leave the central simulation box. Consider a minimization or MD simulation, when the positions were just updated. You know the basic picture: as a particle leaves the central (cubic) box, say, to the right, it is replaced by the corresponding particle from the image of the central box on the left. Thus, working with PBCs entails making sure that your particle coordinates always fall within the central simulation box. This can be accomplished with the following pseudo-code (which only handles periodic boundaries in the x direction) computer code along the lines (the center of the box is supposed to be at the

origin, x is the x coordinate of the particle of interest, and xsize is the size of the periodic box!)

```
if (periodicx) then
  if (x <  -xsize/2.0) x=x+xsize
  if (x >=  xsize/2.0) x=x-xsize
endif
```

So, for example, if we have a particle with a x position of 4 and xsize is 10, the particle would be within the periodic box, however, if the particle's x coordinate was 6, it would be outside of the box and it's x coordinate would be changed to -4 (effectively, when a particle leaves a box on one edge, it reappears on the opposite edge). Analogous code snippets apply to the y and z coordinate, and for a cube `xsize = ysize = zsize=L`

There is, however, a second, more important component to PBC. Consider the interaction (LJ and/or electrostatic) between two particles i and j. If the replicas of the central box were real, we would have interactions between i and j in the central box, but also of i with j in all of the (infinite) number of images of the central box. This is, of course, not what we want to have; instead, we choose the interaction between i and j having the shortest distance. This may be the interaction between i and j in the central box, but it may also be the interaction between i and a particular image atom of j. In pseudo-code this so-called *minimum image convention* can be expressed as

```
if (periodicx) then
  dx = x(j) - x(i)
  if (dx >   xsize/2.0) dx = dx - xsize
  if (dx <= -xsize/2.0) dx = dx + xsize
endif
```

where x(i) and x(j) are the x coordinates of i and j, respectively, and analogous statements for y and z. If you consider the above realization of the minimum image convention carefully, you'll note that the longest possible inter-particle distance is half the box-length, L/2. Thus, if you used a cut-off radius greater than L/2, your effective cut-off radius would still be L/2. As we shall see, CHARMM handles PBC rather differently, and for the minimum image convention to be obeyed the cutoff radius must not be larger than L/2, i.e., it is the user's responsibility to shorten CTOFnb if necessary!

Again, we assume that you have heard all this before. The problem with the above illustrative examples of PBC/minimum image convention is that the code assumes a collection of atoms. Consider, e.g., a box of waters or a box containing a protein surrounded by water. Suppose one water edges out of the box, i.e., the oxygen and one hydrogen are still within the central box, but the second hydrogen is out of the box. Then naively applying periodic boundary conditions will tear the molecule apart, i.e., one has to make sure that a water is shifted periodically only as a complete molecule. Similar considerations apply of course to the protein, or any molecule present in the system. Also, the PBC/minimum image checks have to be carried out for each energy calculation. For cubic boxes, this is not too bad, and by optimizing the above code snippets, the overhead is small. However, for more complex periodic boxes (truncated octahedron, rhombic dodecahedron), this may quickly become difficult to calculate, however CHARMM will handle this for you as long as you choose one of the periodic boxes that it supports (although you can define your own space group, this is not necessary for setting up basic dynamics simulations).

How CHARMM does PBCs

Thus, CHARMM does things differently than most textbooks (the notable exception being Rapaports' *Art of Molecular Dynamics Simulation*, where the general approach taken by CHARMM is outlined), and, actually, differently than most comparable programs. Here we describe the underlying algorithmic approach, whereas in the next subsection we describe how things work from the user perspective and what pitfalls one has to be aware of.

CHARMM takes periodic images "seriously", i.e., (a subset of) the atoms of the periodic images surrounding the central box are generated. The name of the routine that does this is MKIMAT (that's a subroutine name, not a CHARMM command!); keep it in mind, we'll have to comment on this routine somewhat more later on. First consequence: the actual number of atoms in your system with PBC is (much) larger than the number of atoms in your central box. This apparent "waste of memory" is put to good use for the sake of generality and also performance. Contrast this with the textbook approach, the image boxes are not really present, as an image atom "enters", a "real" atom "leaves/vanishes". The number of image atoms is kept as low as possible by two factors. First, one does not have to generate the atoms from all image cells. In 3D a central cubic box is surrounded by 26 images. Of these, only, those to the right, top, and the back of the central cubic system are needed along with those at the vertices and edges, reducing this number to 14. Second, for large boxes (box lengths of solvated protein systems are easily 60 - 100 Angstroms!), one does not need to replicate the full box, instead, one needs only those image atoms which are within the cutoff radius, or more correctly, the cutoff radius used to generate the non-bonded list(s).

A periodic simulation system in CHARMM consists of the central box (e.g., a cube), and a (partial) layer of image atoms with width `CUTNB` (or actually `CUTNB` + some safety distance). Now, one generates *two* non-bonded list, one for atoms in the central box ("primary atoms"), and a second one between primary atoms and image atoms. Non-bonded interactions are computed twice, once with the primary-primary list, and once with the primary-image list. In the CHARMM output you'll find the energies listed separately, i.e., the total LJ and electrostatic energy of a system is now the sum of two terms each. You'll probably have to draw some diagrams (preferably in 2D) to convince yourself that this works. There is one border line case, and that is the case of small boxes. Here you have to ensure that the cutoff radius for the energy calculation (CTOFnb) is less than half the box length. As long as this is true, even if entries for two particles (call them i and j) exist in *both* ***the primary-primary and primary secondary lists, only one of the two distances can be lower than half of the box size. If, on the other hand,*** `CTOFNB` ***is greater than half the box length then both distances could be lower than half the box length and the interaction energy will be double-counted*** **'(BTM: check this with SB or Rick!!)**. By choosing a short enough cut-off, you ensure that the minimum image convention is implicitly obeyed. *Unfortunately*, CHARMM provides no warning if you overlook such a case, and this is one of the pitfalls lurking when using PBC in CHARMM. More in the next subsection. In general, it is simply best to avoid small boxes, since reducing `CUTNB` brings its own set of problems.

Use of CRYStal / IMAGe in its simplest form

Before dwelling on pitfalls, let's look at the practical aspects of setting up PBC. The user interface for setting up PBC in CHARMM is provided by two modules, `IMAGe` and `CRYStal`. `IMAGE` and `CRYSTAL` provide similar capabilities and complement each other. One may also view `CRYSTAL` as an interface to make `IMAGE` more user-friendly, and this is the way it is usually employed nowadays.

Before trying the following snippets, You should have read in RTF, parameters, PSF and coordinates for your system. Also, assuming a protein / water system, we assume that you have DEFIned two atom selections, *protein* containing all your protein atoms, and *water*, containing all your water molecules (which are often simulated in CHARMM via the TIP3 water model). Finally, let's assume that you have a box length of 60 Ang., and you are using / planning to use the default cutoff parameters (CUTNb 14. CTOFnb 12. CTONnb 10., which are our recommended cut-off parameters, although it does no harm to increase CUTNB to 16.0 as we do in some of our scripts, so long as the other parameters are given explicitly) Then, the following four lines set up PBC for a cubic box:

```
CRYSTal DEFIne CUBIc 60. 60. 60. 90. 90. 90.
CRYSTal BUILd CUTOff @XO NOPE 0
IMAGE BYREsidue XCEN 0.0 YCEN 0.0 ZCEN 0.0 SELE water END
IMAGE BYSEgment XCEN 0.0 YCEN 0.0 ZCEN 0.0 SELE protein END
```

The first line defines the crystal symmetry (CUBIc in our example), and gives the necessary information, side lengths A, B, C and the three angles α, β, γ, which in our case are A = B = C = 60 Å, and $\alpha = \beta = \gamma = 90°$, respectively. The generic form of the command would be

```
CRYS BUILd DEFIme <type> A B C α β γ
```

(It is particularly easy to build rhombic dodecahedrons or truncated octahedrons, which are often preferred over cubic simulation boxes. We generally prefer to use rhombic dodecahedrons for globular systems and hexagonal prisms for long, thin macromolecules.)

The second line initiates the building of image atoms. Since CHARMM knows about cubic boxes, no further information about the crystal is needed, and `NOPEr` (the number of crystal operations) is set to 0. More important is the `CUTOff` parameter, which actually indicates how deep to construct the layer of image atoms. In order to work with the non-bonded list approach of CHARMM, the variable @XO (as we call it here, any variable name is fine, alternatively, you can give the number directly) has to be as large as $L/2$where $L$is the length of the unit cell. This is particularly important if there is a significant amount of vacuum space within the unit cell.

The meaning of the third and fourth line ("raw" `IMAGe` commands for a change) become clear once you understand the CHARMM way of handling PBC during a minimization or MD simulation when the coordinates of all particles change continuously. Eventually, particles in the primary box will drift out of the box, and atoms in the image layer will enter the primary region (or "diffuse" further away). This is no need for immediate concern, since the "skin" in our non-bonded lists gives us a safety net (just as in the absence of PBC). Eventually, however, the central box and the image layer will have to be rebuilt (by the MKIMAT routine). The obvious time to do so is when the non-bonded lists are recomputed. At this point all atoms are essentially subjected to a PBC like test, but as outlined above, one has to avoid breaking molecules into two. The two IMAGe lines tell CHARMM to apply periodic shifts to water on a residue by residue (= molecule by molecule) basis (option BYREsidue, line 3), whereas the protein is shifted as a whole (BYSEgment, line 4) -- in the case of proteins, shifting by residues could pull off individual amino acids or small groups of them!

This is it, or rather, this should be it. One would assume that once PBC is set up (via `CRYStal` / `IMAGe` as shown), any non-bonded list update would update both the primary-primary and primary-image list, and that `CUTNB` would be understood as being applicable to the generation of both lists. Alas, not so ... For (lets assume) historical reasons, the update frequency for the two lists (primary-primary, primary-image) are controlled by two different parameters, `INBFrq` and `IMGFrq`; similarly, there are two cut-off radii, `CUTNB` for the primary-primary and `CUTIM` for the primary-image non-bonded lists. Obviously, `INBFrq` should always equal `IMGFrq`, and `CUTNb`. `CUTIM` is often set to be equal to `CUTNB`, but it may be set larger if computer memory permits to reduce the frequency of list building. To add insult to injury, there is no mechanism to ensure that reasonable values have been chosen for these parameters. For example, while `INBFrq` defaults to -1 (heuristic update, which is good), `IMGFrq` has a default of 50, which is nonsensical! Thus, it is a sad fact of life that ***it is the user's responsibility to ensure meaningful values for INBFrq, IMGFrq, CUTNb and CUTIm***, otherwise rubbish can and will be produced.

So, for the sake of good practice, let us show the necessary steps to set up PBC for a cubic system and ensure correct energy / force calculations. For good measure, we use a large than default `CUTNB`/`CUTIM`, and we request `BYCB`:

```
! read RTF

! read parameters (we assume param22/27, which in principle sets up
```

```
     ! most defaults correctly, i.e., what we get at this point
     ! corresponds to
     !   NONB nbxmod  5 atom cdiel shift vatom vdistance vswitch -
     !        cutnb 14.0 ctofnb 12.0 ctonnb 10.0 eps 1.0 e14fac 1.0 -
     !         wmin 1.5
     ! as discussed

! read psf

! read coordinates

set rc 16.         ! value we want for CUTNB/CUTIM

CRYSTal DEFI CUBI 60. 60. 60. 90. 90. 90.
CRYSTal BUILd CUTOff 30 NOPE 0 ! cutoff is half the unit cell length
IMAGE BYRE XCEN 0.0 YCEN 0.0 ZCEN 0.0 SELE water END
IMAGE BYSE XCEN 0.0 YCEN 0.0 ZCEN 0.0 SELE protein END

! now issue an ENERgy command to (re)set all defaults!
ENER INBFrq -1 IMGFrq -1 CUTNb @RC CUTIm @RC BYCB -
     NBXMod  5 ATOM CDIEl SHIFt VATOm VDIStance VSWItch -
     CTOFnb 12.0 CTONnb 10.0 EPS 1.0 E14Fac 1.0 WMIN 1.5

! Note: in practice we would usually replace SHIFte
!       electrostatics by PME Ewald, see below
```

The above should guarantee that all subsequent minimization and or MD calculations generate both non-bonded lists at correct intervals with sane cut-off radii. As noted in the comment, the one unrealistic aspect of the above code snippet is that PME is not set up. Normally, the ENERgy command above would also be used to replace SHIFted electrostatics by PME; however, we first need to introduce Ewald summation and PME.

Ewald summation / Particle-Mesh-Ewald (PME)

Ewald summation and PME for dummies

Some more rigorous background can be found on Wikipedia [3]. In the following I comment in (over)simplified form on some aspects of Ewald / PME, which I find is surprisingly often misunderstood in practice.

Ewald summation (ES) is a method to sum up (= calculate) electrostatic interactions in an infinite lattice, or, alternatively to compute electrostatic interactions under periodic boundary conditions. If you want to look at it that way: with ES you take these periodic images "seriously". In symbols, the starting point of ES is to compute the (infinite) sum (the lattice sum)

$$U_{elec}^{periodic} = \frac{1}{2} \sum_{\mathbf{n}}{}' \left(\sum_{i=1}^{N} \sum_{j=1}^{N} q_i q_j \left| \mathbf{r}_{ij} + \mathbf{n} \right|^{-1} \right).$$

Here q_i, $q_j$are the charges of particles i, j, $\mathbf{r}_{ij}$is the vector between particle positions i, j, and $\mathbf{n}$is the lattice vector pointing into all periodic images of the primary box. The prime in the sum over the $\mathbf{n}$is used to indicate that for $\mathbf{n} = 0$(primary box), the case i=j is excluded.

$U_{elec}^{periodic}$has the unpleasant mathematical property of being conditionally convergent, i.e., its value depends on the order of summation.

In addition to being conditionally convergent, direct computation of the above lattice sum would be extremely tedious since the convergence is also *slow*. ES is a trick to accelerate the convergence; as a by-product the singularity of the Coulomb interactions responsible for the conditional convergence is avoided. (Thus, ES may be viewed as a modified lattice sum).

In ES the above sum is split into two sums, according to

$$\frac{1}{r} = \underbrace{\frac{\mathrm{erfc}(\kappa r)}{r}}_{real} + \underbrace{\frac{\mathrm{erf}(\kappa r)}{r}}_{reciprocal}$$

Here, `erf` and `erfc` are the error and complementary error functions, respectively. For a suitable choice of κ, the first term on the right hand side is short-ranged, whereas the second term contains all the long-range interactions. More specifically, if $\kappa$is chosen sufficiently large, for the first term only interactions in the primary simulation box need to be taken into account (think of the erfc term acting as a "shifting" function, damping the long range 1/r potential rapidly to zero. Thus, the kernel labelled *real* leads to the *real space sum of electrostatic interactions* which is computed normally (using the complementary error function erfc(x) as a "shifting" function),

$$U_{elec}^{real} = \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1\neq i}^{N} q_i q_j \frac{\mathrm{erfc}(\kappa r)}{r_{ij}}\right).$$

The kernel labelled *reciprocal*, on the other hand, is long-ranged, and summation here would have to run over all lattice vectors **n**. Long-ranged interactions, however, become short-ranged in reciprocal space (hence the name), and essentially by Fourier transformation the sum resulting from the *reciprocal* kernel becomes a rapidly converging sum in reciprocal space, referred to as a "reciprocal sum" or "k-sum".

The exact mathematics adds a few additional terms and corrections, which CHARMM computes correctly, and hence we can afford to ignore. The computationally intensive parts are the real space and reciprocal space sums just discussed. The former, as pointed out, can be computed within the normal framework of non-bonded interactions, whereas the second is a somewhat funny term, entailing a double sum over atom positions and reciprocal space vectors. We spare you the details since the k-sum is nowadays usually computed by the so-called *particle-mesh-Ewald* (PME) technique. (*Note: one sometimes gets the impression that ES and PME are considered different methods. This is not so: PME is just a fast, efficient way of approximating one term of the ewald sum, namely the k-sum!*)

The basic trick of PME consists of the fact that the k-sum is not computed for atom positions, but for fixed positions on a grid. This allows CHARMM to pre-compute a lot of stuff *and* to use fast fourier transforms (FFT) to speed things up further. The atomic charges are smeared out on a grid in each step, potential (and forces) computed on the grid and are back-transformed to their atomoc positions. By using so-called B-splines for the "smearing", it suffices to compute the potential; the forces are obtained by differentiation of the splines (this avoids separate FFTs for potential and force calculations). Traditional ES scales approximately as $N^{3/2}$for *N* atoms; PME scales as $N \log N$, which makes a big difference for large systems.

CHARMM implements both traditional ES as well as PME to compute the k-sum. Historically, the two methods were coded independently, and the older implementation of ES has a number of limitations. Since it is also much slower than PME, only PME is discussed subsequently.

Setting up PME

Ewald summation is requested by the `EWALd` option to `ENERgy`, and PME is requested by adding `PMEWald` as an option. While the original ES method had numerous options to fine-tune the calculation of k-vectors, for PME only 4 additional options are relevant. The first is the choice of κ, the others specify the number of grid points in each of the three spatial directions. A full energy call requesting PME looks like

```
! set up CRYStal / IMAGes (= PBC), otherwise the following will fail
ENERgy ... EWALd PMEWald KAPPa 0.43 FFTX 64 FFTY 64 FFTZ 64
```

The above sets $\kappa$to 0.43, and requests a $64 \times 64 \times 64$grid. As a rule of the thumb, the grid spacing in each spatial direction should be approximately 1 Å. One has to keep in mind, however, that the FFT implementation of CHARMM has certain limitations concerning the choice of prime factors; essentially, only powers of 2, 3 and 5 are allowed, e.g., FFTX 27 (= 3x3x3) is OK, but FFTX 28 (= 2x2x7) is not. Further, the more powers of 2 relative to powers of 3 and 5, the better (i.e. faster). CHARMMing uses an external Perl script to calculate optimal grid dimensions for a given system size.

The trickier parameter is κ. Since the real space sum is calculated by the normal non-bonded routines, the cut-off radius `CTOFNB` is applied. It, therefore, has to be sufficiently large as to ensure proper damping so that at distances $r \rightarrow r_{cut}$interactions and forces are effectively zero. If the damping is too low (κtoo small), then you miss interactions and introduce a severe error in your calculations. There are several rules of the thumb to check the correct choice of κ; below is the exact criterion to choose κFor your value of r_{cut}(`CTOFNB`) and κevaluate (e.g., Mathematica!)

$$\lambda_{EW}(\kappa, r_{cut}) = \int_0^{r_{cut}} 4\pi r^2 dr \left(\frac{\kappa}{\sqrt{\pi}}\right)^3 e^{-\kappa^2 r^2}$$

To be on the safe side, if your $\lambda_{EW}(\kappa, r_{cut}) < 0.999$, increase κ(ideally, $\lambda_{EW}(\kappa, r_{cut}) = 1$).
Note that as long you use the the default `CTOFNB` (12 Å), you can simply use the default $\kappa$of 0.43 as it has been chosen for this value of `CTOFNB`. In this case it is unnecessary to check the value via the above equation. A less precise rule of thumb (from ewald.doc [4]) is to set $\kappa$to $5/CUTNB$.

Final note for running MD with PME: The implementation of the PME in CHARMM (as well as in most other comparable programs) does not conserve center of mass translation (sum of all momenta is not exactly zero). If no provisions are taken, your system might start to pick up a net translational velocity component. To compensate / avoid this artifact, CHARMM enforces that during MD you take out center of mass translation by specifying a non-zero `NTRFrq` (reasonable values are from 500 to 5000, although it does no harm to set it smaller).

A theoretical PS: ES is mostly discussed in the context of summing up electrostatic interactions in an infinite lattice (see, e.g., the presentation in Allen/Tildesley). A different point of view can be found in the presumably first publication in which ES was used in the context of a computer simulation (Brush, Sahlin and Teller, JCP 45, 2102 (1966), yes *the* H-bomb Teller!). The Ewald potential is simply the solution to the Poisson equation under periodic boundary conditions.

A demonstration of how to use particle mesh ewald is given on the complete example page.

References

[1] http://en.wikipedia.org/wiki/Force_field_(chemistry)|the
[2] http://www.charmm.org/documentation/current/nbonds.html
[3] http://en.wikipedia.org/wiki/Ewald_summation
[4] http://www.charmm.org/documentation/current/ewald.html

Minimization

Description of Minimization

Energy minimization, as the name implies, is a procedure that attempts to minimize the potential energy of the system to the lowest possible point. This can be a challenging problem, as there is only one global minimum of the energy surface, but many local minima. If steps are not taken to avoid it, a minimization algorithm might get stuck in a local minimum without ever finding the global minimum. In most cases, this is what will actually happen in CHARMM, but as explained below, finding a global minimum is often not necessary. CHARMM's minimization algorithms examine the first (and in some cases second) derivatives to determine whether they are at a minimum. More complex methods of exploring the energy surface (e.g. conformational space annealing, in which case the system is repeatedly heated and cooled) are beyond the scope of this tutorial.

Motivation

One might reasonably ask why you would want to perform energy minimization in the first place. There are a number of reasons, but they mostly center around removing nonphysical contacts / interactions. For example, with a structure that has been solved via X-ray crystallography, contacts with neighbors in the crystal can cause changes from the *in vitro* structure. Missing coordinates obtained from the internal coordinate facility or the HBUIld module of CHARMM may be far from optimal. Additionally, when two sets of coordinates are merged (e.g., when a protein is put inside a water box or sphere) it is possible that there are steric clashes present in the resulting coordinate set. Such *bad contacts* may cause large, unphysical changes in potential energy in subsequent dynamics simulations, or possibly other artifacts. A brief minimization can remove such potential problems. As a result, minimization is generally not done as an end unto itself, but to prepare for another type of calculation and must be done under the same conditions as the later calculation. For example, when planning a QM/MM calculation (i.e. when one part of your system is modeled with quantum mechanics while the rest uses the standard CHARMM force field)), it is important to re-minimize the system with the QM/MM conditions in place.

How much should a structure be minimized?

A minimization is considered converged if the root mean squared deviation of the gradient (GRMS) is very close to zero (recall that the necessary condition for a function to have a local minimum is that all its first derivatives are zero -- of course, it could also be an extreme, saddle point etc.). In addition, the energy changes from step to step should be very small when the minimization is close to convergence.

How much minimization is necessary depends upon the intended use of the final structure. In general, if the output is to be used for solvation or dynamics, it is not necessary for the minimization to be fully converged. In fact, when preparing a structure for dynamics, it is not advised to let it move too much from its original conformation. Exploration of conformational changes should be done during dynamics itself, not minimization. Over-minimization can lead to unphysical "freezing" of the structure. It may be desirable to use thermal B-factors to restrain the structure (as these are hints from the people who solved the structure about the certainty of the atomic positions). However, if the structure is to be used for a normal modes calculation, then it should be as close to a local minimum as possible. This is because a normal mode calculation treats the potential as a harmonic well. If the structure is not

at, or very close to, a minimum, it will not be at the bottom of that well. Consequently there will be elements of the matrix made up of second derivatives of the energy (i.e. the Hessian) that will be negative. This this matrix will not be positive definite and there will be negative eigenvalues and, hence, imaginary normal modes.

When both minimizing and solvating a structure, it is often best to minimize the structure in vacuum before performing solvation.It is often difficult to resolve bad protein-protein contacts and bad protein-solvent contacts within the same minimization, so it is often desirable to do these minimizations separately.

What algorithm should be used?

A full description of the minimization options available in CHARMM can be found in minimiz.doc [1], which also documents the exact syntax for the MINImize command (described in the next section). Most people use just two of the available minimizers, steepest descent (SD) and the adopted basis Newton Raphson method (ABNR). SD is the simplest algorithm; it simply moves the coordinates in the negative direction of the gradient. The only adjustable parameter is the step size, which determines how much the coordinates are shifted at each step. The step size is adjusted dynamically to achieve rapid convergence. The main problem with steepest descent is that it does not generally converge to a local minimum; however, it will rapidly improve the conformation when the GRMS is high (i.e. when the system is far from a minimum). Therefore, it is often used briefly on a new structure to quickly remove bad contacts and clashes. Many practitioners ponly use SD for the initial 25-50 steps of minimization to remove bad van der Waals contact, however it can be run for longer. Once this is done, it is advisable to switch to a more precise minimizer, usually ABNR, to finish the calculation. The ABNR minimizer will be able to detect when it has converged and will exit automatically, therefore there is no danger in running it for a long time (i.e., specifying a large number of steps; see below). However, as stated previously, except for when computing normal nodes or preparing for QM/MM dynamics, there is generally no need to arrive at an exact minimum. When the energy change from step to step is less than 0.001 kcal/mol, the structure is sufficiently minimized for most purposes.

In certain special cases, it may be necessary to use another minimizer such as the `CONJugate-gradient` method or the `POWEll` minimizer. Consulting the documentation carefully is recommended in these cases. In particular, note that the `POWEll` minimizer does not support the `INBFrq` and `IMGFrq` keywords (which determine how often the non-bond and image atom lists are regenerated). Therefore, it will certainly fail if the structure moves too much during the calculation. It is suggested to use the `POWEll` method in a loop; however it is probably best avoided altogether.

How to perform minimization

Energy minimization can be performed once a valid PSF and coordinate set have been loaded into CHARMM. The `MINImize` command is used to initiate the procedure. Immediately after the command itself the method (e.g. `SD`, `ABNR`) must be specified, followed by any minimizer specific options. In general, the only options that need to be given are `NSTEp`, TOLGradient, and `TOLEnergy`. The `NSTEp` option specifies the maximum number of steps that the minimizer will run. Likewise, the `TOLGradient` option tells the minimizer to exit once the specified GRMS is achieved and the `TOLEnergy` option tells the routine to exit after the energy changes by less than the given amount. If these are not specified, minimization will run until its own convergence criteria is met. For most systems, it is sufficient to do a few hundred steps of minimization with no `TOLG` or `TOLE` specified to get the system away from sharp peaks in the energy curve, followed by a sufficient number of ABNR steps to meet the desired `TOLG`. For example:

```
! perform a basic energy minimization
mini sd nstep 50
mini abnr nstep 1000000 tolg 0.01
```

will be sufficient in most instances. As mentioned above, there is no harm in setting the large value of NSTEp, even without the TOLG or TOLE options; the minimizer will exit once it has converged (note, however, that the defaults for TOLE and TOLG are 0.00 so unless these are increased it the minimization will run until the system is either right on a local minimum or NSTEp is reached.

There is some debate as to whether a structure should ever be minimized in vacuum, although as mentioned above light vacuum minimization is often very useful when preparing a structure for solvation. This is unavoidable when a gas phase normal mode calculation is to be performed. General hints for both gas phase and solvent minimization are given below.

If minimizing directly before a dynamics run, it is necessary to make sure that the conditions under which the minimization takes place are the same as those intended to be used for dynamics. For example, if particle-mesh ewald is to be used in dynamics, it should be used for minimization as well. Consideration should be made of the system in question; below, general hints for minimizations in gas phase and in explicit solvent are given.

Gas phase minimization

Gas phase minimization may be complex as, without solvent buffering, the structure can minimize into non-physical conformations. Some find it helpful to put harmonic restraints on the backbone atoms of the protein, letting only side chains move. It is possible to go even farther and fix (or, at least, restrain) all atoms except for hydrogens, allowing just their positions to be minimized. As mentioned above, thermal B-factors can serve as guides for how strongly to restrain individual atoms. Minimizing only hydrogens can be desirable since CHARMM's hydrogen position builder (HBUIld) often does not put hydrogens in the most physically advantageous positions. It is generally possible to gradually heat a minimized structure to obtain a higher temperature configuration that is reasonably energetically stable.

Minimizing a solvated structure

As mentioned above, the conditions under which minimization takes place should be the same under which the dynamics run will be performed. Therefore, it is necessary to set up periodic boundary conditions and nonbonded parameters as they will be used later. As alluded to above, it is wise to remember that the goal of the minimization is to reduce bad contacts and relax the structure. Using too many steps can result in over-minimization, which can cause issues with subsequent MD simulations. As in the gas phase, it may be desirable to constrain backbone atoms, allowing the solvent and sidechains to orient optimally while the basic shape of the protein stays intact. Remember, the structure can be heated to and equilibrated at the proper temperature before starting production dynamics, but over minimization will cause these processes to take a lot longer or possibly fail entirely. One hundred steps of SD followed by a few hundred steps of ABNR are often sufficient to prepare a solvated structure for dynamics.

Conclusion

The general difference between vacuum and gas phase minimization is that one must be even more careful to prevent radical changes to the structure during gas phase. Ideally, a structure should only be minimized in the presence of solvent, but there may be issues that need to be resolved before initiating solvation where a short gas phase minimization would be desirable. The main thing that must be considered when setting up the minimization of the solvated system is making sure that, if periodic boundary conditions are to be used in dynamics then they should be configured identically.

References

[1] http://www.charmm.org/documentation/current/minimiz.html

Solvation

Overview

Solvation is surrounding the solute, generally a protein, nucleic acid strand, or other macromolecule with a solvent, typically water. This is very useful in biomolecular simulations since biochemical reactions generally take place in a viscous environment. Though CHARMM supports methods of estimating solvent effects implicitly, in many simulations it is useful to be able to explicitly place the system being studied into a solvated environment. There is no specific command for solvation, but it can be done through a series of commands. The basic procedure for solvation, as implemented by CHARMMing, is:

1. Read in the system to be solvated and center it
2. Read in a large box of pre-equilibrated waters (CHARMMing uses TIP3, which is the residue name for the default TIP3 water model)
3. Delete the waters that overlap the solute (any water whose Oxygen is within 2.5 Å of the solute) and are more than a certain distance from the center (the exact distance depends on the size of the solvent box chosen by the user), leaving a spherical water structure
4. Unless the shape of the water structure is to be a sphere, delete any remaining waters that are outside of the final unit cell

This procedure will be discussed below. A full-fledged example showing the exact CHARMM commands is given in the full example.

How much solvent is needed

As a general rule, biochemists are not particularly fascinated in the behavior of bulk solvent, however, they are very interested with the way that various macromolecules behave in its presence. This creates something of a conundrum: there needs to be enough solvent present to allow the system of interest to interact with it naturally, but not so much that the system becomes too big to simulate on the computing power available (the solvent-solvent interactions add to the computing power needed to simulate the system). Using periodic boundary conditions (PBC) does not relieve the user of the need to deal with this problem: if there is not enough solvent to fill the unit cell then unphysical "vacuum pockets" will be created; if the unit cell is too small then the macromolecule may interact with images of itself.

There is no hard and fast rule on how much solvent is needed. The decision should based on the size and shape of structure and how much it is expected to move during the simulation. If the system of interest is expected to move significantly and PBC is not being used, then it is necessary to provide a buffer of solvent sufficient to accommodate its movement in any direction plus a buffer zone of at least 12 Å. It is important to note that there can be substantial artifacts in the first couple of layers of solvent, and therefore a sufficiently large layer is needed to reproduce bulk solvent conditions. If PBC is being used, then in general the solvent structure and unit cell size should be at least the length of the longest axis of the molecule plus twice the cutoff distance from nonbond interactions so that no solvent water molecule interacts with both the solute and its image. Bear in mind when using "long and narrow" unit cells that some portions of the macromolecule may rotate "outside" of the solvation shell during simulation.

Consideration on shapes

As mentioned in the tutorial section on CHARMM's energy function, CHARMM supports PBC using any valid crystal unit cell (in practice, however, only a few different shapes of unit cells are actually used). It is convenient to set up the crystal structure (if one is needed) while solvating the system. Bear in mind that regular shapes must be used as unit cells, i.e. you cannot have a spherically shaped unit cells because spheres cannot be packed to completely fill a given volume of space,

CHARMMing, our Web based interface to CHARMM, contains functionality to automatically determine an efficient shape. To do so, CHARMMing examines the longest and shortest axis of the structure. If the longest axis is more than 30% longer or the shortest axis is more than 30% shorter when compared to the middle axis, then a hexagonal prism is used. Otherwise, a rhombic dodecahedron is chosen. The rhombic dodecahedron (RHDO) is a good choice for globular proteins because it most closely approximates a sphere and thus is the most efficient crystal shape (not a lot of excess water). The hexagonal prism works well for long, thin structures for a similar reason, however it is necessary to be careful to make sure that the macromolecule does not rotate outside of the prism during the simulation. It is generally not a good idea to put restraints on molecules undergoing in MD, however to keep long and narrow structures from rotating MMFP cylinder restraints (see mmfp.doc [1] might be better than simply restraining the end residues. Ideally, however, the molecule should be able to rotate freely without moving outside the unit cell.

Performing the solvation

Orienting the structure

Once you have the structure read into CHARMM, it is desirable to orient it with the "COOR ORIEnt" command. This will rotate the molecule so that the x axis is the longest axis, the y axis is the second longest, and the z axis is the shortest. You can then figure out the lengths of each of the three axes via:

```
coor stat
calc xdist = abs( ?xmax - ?xmin )
calc ydist = abs( ?ymax - ?ymin )
calc zdist = abs( ?zmax - ?zmin )
```

Reading in and cutting down the water

Once you have the macromolecule read in and oriented properly, the next thing that is needed is a water box large enough to accommodate the desired crystal structure. You can download the one used by CHARMMing as a CHARMM coordinate file here [2]. Once you have the file, you can read it in with your existing structure. To do so, you must do two things: (1) append the waters to the PSF as a new segment and (2) read their coordinates in. Since there are 46656 waters in the box you can do this via the following commands:

```
! append to the PSF -- the generate command
! automatically appends the BWAT segment to
! your existing structure
read sequence tip3 46656
generate bwat noangle nodihedral
```

```
! now read the coordinates in using APPEND
! to add them to the current set
read coor card append name water.crd
```

When this is done, it is necessary to delete the water molecules that overlap with the solute. In CHARMMing, we accomplish this by deleting all waters whose oxygen atom is within 2.5 angstroms of the solute. The command to do this (assuming the segment of bulk waters is name BWAT) is:

```
delete atom sort sele .byres. ( segid BWAT .and. type oh2 .and. -
  (( .not. (segid BWAT .or. hydrogen)) .around. 2.5 )) end
```

After deleting the atoms that overlap, we recommend cutting the water structure down to a sphere just large enough to contain the final crystal structure. A reasonable way to do this is to set the diameter of this sphere just large enough to circumscribe a cube with an edge length of the longest exis of the macromolecule (@xdist) plus two times the padding distance. All waters outside this diameter are then deleted. This is done for efficiency reasons; it will be much faster to remove extraneous water surrounding the unit cell if most of the extraneous waters have already been removed, The CHARMM commands to do so are:

```
calc caxislen = ?xdim + ( 2 * @padding )
calc caxislsq = @caxislen * @caxislen
calc spherer = ( sqrt( 3 * @caxislsq ) ) / 2
delete atom sort sele .byres. ( .not. ( point 0.0 0.0 0.0 cut @spherer ) .and. ( segid bwat ) ) end
```

In the above commands, `@caxislen` is set to the longest dimension of the macromolecule plus two times the desired padding and `@caxislsq` is the square of this value. The `@padding` is often set between 5 and 15 Å, depending on how much of a solvation shield is needed around the molecule. The `@spherer` variable is the radius of a sphere circumscribing the cube of edge length `@caxislsq` (the requisite diameter is calculated via the Pythagorean Theorem and divided by 2 to get the radius). The complex part is the selection inside the delete command. It select all residues that are not within a radius `@spherer` of point (0,0,0) (this is another reason why it's important to do a `COOR ORIEnt` before adding the water, as it will center the molecule at the origin), and that are part of the BWAT segment (in the file from CHARMMing, the waters have their segment ID set as BWAT, adjust as necessary for your own structures). It is important to select by residue (`.BYRES.`) because otherwise you might wind up deleting only part of a water molecule.

Building the crystal

When we have the water structure cut down to size, we can go ahead and create the unit cell. This is done in two steps: the crystal structure is defined and then built. This tutorial assumes that you are using one of the standard shapes built into CHARMM (cube, hexagon, tetragonal, rhombic dodecahedron, etc). To define a crystal shape use the CRYStal DEFIne command. The basic syntax is:

```
cryst defi <type> <a> <b> <c> <α> <β> <γ>
```

where a. b. and c are the edge length and aplha, beta, and gamma are the angles. For example, to define a cube with sides of 20 angstroms you would write:

```
cryst defi cubic 20. 20. 20. 90. 90. 90.
```

to define a rhombic dodecahedron with edge length 30 you would write

```
cryst defi rhdo 30. 30. 30. 60. 90. 60.
```

The correct alpha, beta, and gamma values for each supported crystal type are given in crystl.doc [3]. Note that a high degree of precision for the angles is necessary to construct the truncated octahedron structure (i.e. if you try to round off the angles, you will not get a proper octahedron).

Once the crystal is defined, you can go ahead and build it with:

```
cryst build noper 0
```

The NOPERations option specifies how many crystal operations need to be performed. A regular shape centered at the origin with only translational symmetry does not need any crystal operations (see the discussion on crystal structure from the energy page of this tutorial.

Removing waters outside the crystal structure

It is now necessary to remove the waters that lie outside of the unit cell. This can be done by setting up images and forcing an image update (via the UPDAte command). When the image update is run, all atoms that are outside of the unit cell boundaries will be moved back inside them. By copying the coordinates to the comparison set before doing the update and finding the difference between the main set (the coordinates after the update) and comparison set (the coordinates after the update), we can detect which atoms moved and are, therefore extraneous. We can then delete these so as to preserve the correct density of the equilibrated water structure. This procedure is shown in detail in the worked-out example.

The solvation procedure is now complete.

After solvation

It is desirable to do a quick steepest-descent minimization (only a few tens of steps) to remove bad van der Waals contacts.

When using the solvated structure with PBC in a different script, you will need to recreate the crystal structure. To assist with this task, CHARMM can write out the crystal transform file as follows:

```
open unit 50 write card name crystal.xtl
cryst write card unit 50
* crystal structure -- it might be a good idea to put a, b, c,
* alpha, beta, gamma in the header
*
```

This file can then be read back in the CRYStal BUILd command. Alternatively, you can just re-run CRYStal DEFIne with the exact same a, b, c, α, β, γ, and then re-run CRYStal BUILd. It is also desirable to put the lattice type and dimensions into the PSF and coordinate files that are written out after solvation.

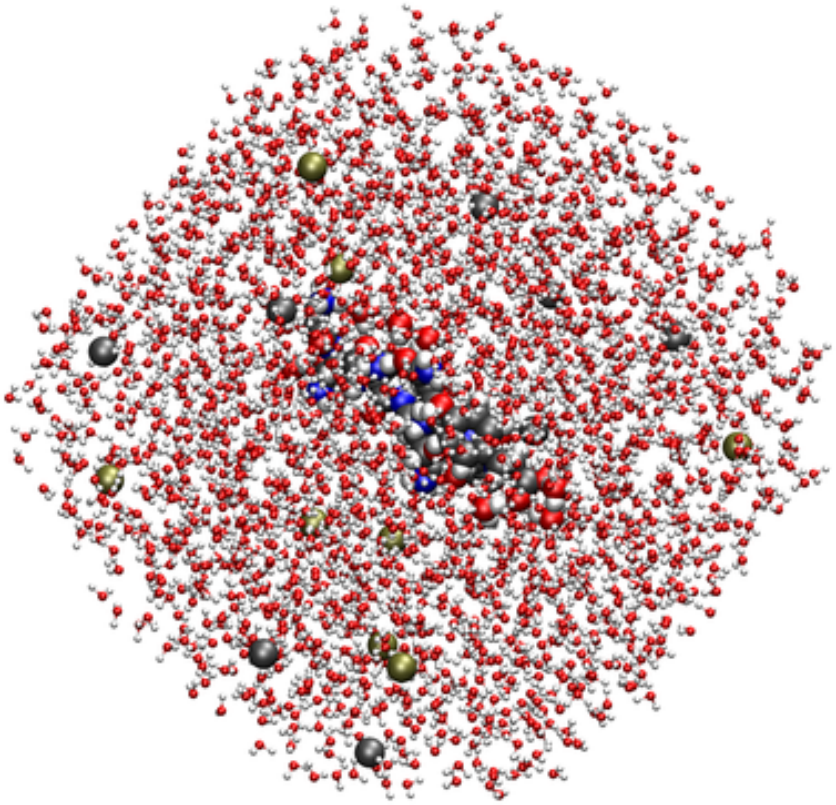
Remember CRYStal by itself sets up the image transformations and everything needed for the energy calculation. You need to use the IMAGe command to specify which atoms in your system are subject to image centering.

References

[1] http://www.charmm.org/html/documentation/current/mmfp.html
[2] http://www.charmmtutorial.org/static/files/water.crd
[3] http://www.charmm.org/documentation/current/crystl.html

Neutralization

Description of Neutralization



Some macromolecules may have an overall positive or negative charge instead of being neutral. The implementation of the Particle Mesh Ewald method (PME) for computing electrostatics that is present in CHARMM works most reliably when there is no net charge on the system. Therefore, when a molecule has a charge, it is generally thought best to cancel it out by adding counter ions to the solution. It is easy to detect when this needs to be done since when CHARMM reads in or modifies a PSF, it will output the total charge of the system as shown in the following sample output.

```
PSFSUM> PSF modified: NONBOND lists and IMAGE atoms cleared.
PSFSUM> Summary of the structure file counters :
        Number of segments      =        1   Number of residues   =      748
        Number of atoms         =     2305   Number of groups     =      770
        Number of bonds         =     2305   Number of angles     =      873
        Number of dihedrals     =      186   Number of impropers  =       21
        Number of cross-terms   =        3
        Number of HB acceptors  =      753   Number of HB donors  =       14
        Number of NB exclusions =        0   Total charge =    0.00000
```

The charge can also be calculated by printing out the `?STOT` variable after running the `SCALar CHARge STAT` command.

Even if the system is neutral overall, it is often desirable to have an ion concentration since this more realistically mimics the interior of a cell or conditions in the blood stream. For example, it is known that cellular NaCL or KCL concentration is on the order of 0.15M and therefore, enough sodium or potassium and chloride ions will be added to both neutralize the system and set the correct concentration. However, adding charged particles close to a protein may influence its behavior in undesirable ways. Therefore, unless it is known that ions are found in close proximity

to the macromolecule, they should be placed at some distance from it so that they are screened by the solvent.

Points to Consider before Neutralizing

There are a few things that need to be considered before you write a script to perform neutralization.

- How many total counter-ions will be needed? Is there enough volume to accommodate them without getting too close to the solute?
- If you want to change the overall charge on the system (e.g. from +3 to 0) you will need different numbers of positive and negative ions (in this case, you would need 3 more negative ions than positive ones).
- Which ions should be used? CHARMM has parameters for a number of different salts.
- Are there any charged residues on the surface of the macromolecule that might be particularly disturbed in the presence of counter-ions?
- Conversely, are there any charged residues that might not behave correctly **without** a nearby counter-ion?

The example given in this tutorial describes a general procedure for neutralizing a system, however it does not consider too carefully the points mentioned above. You will need to understand the electrostatic and physical behavior of your system to know what concentration of ions is most appropriate.

General Neutralization Procedure

The neutralization procedure implemented in CHARMMing is described below and uses a monte-carlo like method for picking a "good" configuration of ions in the solvent. The basic idea is that solvent waters are randomly replaced (with a few caveats) with ions. The system is then minimized briefly to remove bad contacts and the final energy is stored. This is repeated a certain number of times, and the minimized structure with the lowest total energy is chosen as the final neutralized system.

A more detailed outline of the procedure is as follows:

1. User inputs the desired salt and concentration (it is assumed that the final charge will be 0) as well as the desired number of trials.
2. Compute number of positive and negative counter-ions required to achieve the desired concentration and to make the system neutral.
3. Create the positive and negative ion segments in CHARMM.
4. Define all possible water molecules that could be replaced by this ion. These are all bulk waters that are not too close to the solvent or to another ion (CHARMMing uses a 5.5 Å cut-off).
5. For each ion to be placed:
    1. Delete a water molecule at random. and add the ion in its place
    2. Repeat until all ions required to reach the desired concentration have been added
6. Perform short minimization (100 steps of steepest descent)
7. Check the energy versus the previous lowest energy conformation
8. If current energy is lower then previous then save the structure and return to step 4
9. Repeat until the desired number of trials have been run

Needless to say, some of these steps require some advanced magic with the `SELEct` command. A complete example of this is given later on in the tutorial.

MD

Description of Molecular Dynamics

Molecular dynamics simulates the motion of a system by numerically integrating Newton's second law of motion. The objective of a MD simulation is usually to determine the time correlation between events or to get a statistically-valid collection of structures that satisfies ergodicity (this is called sampling from the ensemble). Currently, MD simulations tend to run from several nanoseconds to around a microsecond (although millisecond scale MD simulations are possible on advanced hardware). This is often too short to simulate biological processes such as the folding of complex polypeptides. Biophysicists often therefore look to use the results of MD simulations to make statistical arguments about the behavior of a structure instead of directly observing long scale behavior. This tutorial will focus on the practical issues of setting up a molecular dynamics simulation in CHARMM. We will begin with a discussion of the prerequisites for undertaking a successful simulation.

Prerequisites

In order to begin an MD simulation, you should have a structure that has been reasonably well minimized under the **exact same** conditions as you are planning to use for the dynamics. This means that the non-bonded set-up (including Ewald) and periodic boundary conditions should be exactly the same. However, in many cases it is undesirable to minimize the structure too much, as it may deform in an undesirable manner. Please see the Minimization page for more details.

If there are problems with the way that the model was built, they will likely manifest themselves in the dynamics run. In many cases, the potential energy of the system will explode to unrealistic levels. In other cases, the structure will twist into physically impossible conformations. You should keep an eye on the energy and position of your system during dynamics to make sure that this does not happen. Visualizing the MD trajectory is highly recommended as a check for possible problems. Also, pay careful attention to the warnings CHARMM prints out and resist the temptation to disable ECHECK (or set it to an unreasonably high value). Exceeding the energy change tolerance (ECHECK) and deviations in SHAKE can indicate model building errors or an incorrect dynamics set-up.

It is usually necessary to heat and equilibrate a structure at the desired temperature after minimization. This topic is discussed further below. For finicky structures, it might be necessary to begin the equilibration at a shorter time step (e.g. 0.5 fs instead of 1 fs) to keep within the energy change tolerance. It may also be desirable in certain circumstances to restrain the structure during heating in order to prevent it from deviating too far from the minimized structure. However, it is generally **not** a good idea to fix atoms in place (using cons fix) during any type of dynamics. CHARMMing fixes atoms whose parameters were estimated via GENRTF, but this is not a good idea for real production dynamics (atoms without "real" parameters should either be deleted or real parameters generated for them using, for example, the CHARMM General Force Field).

Choosing an ensemble

As mentioned above, one of the purposes of running molecular dynamics is to sample a collection of structures so as to be able to make a valid statistical argument about the system of interest. CHARMM supports several different types of ensembles which you can sample from. A comparison of these ensembles is beyond the scope of this tutorial; consult your favorite statistical mechanics textbook for more details!

The main types of ensembles used are:

- Canonical (NVT): In this ensemble, number of particles (N), volume (V), and temperature (T) are held constant.
- Microcanonical (NVE): This ensemble holds number of particles, volume, and total energy (E) constant.
- Isothermic-isobaric (NPT): For this ensemble number of particles, pressure (P), and temperature are constant.

CHARMM has several temperature and pressure control mechanisms which are discussed below. After initial thermal equilibration, an NVE ensemble can be simulated by disabling any further heating (setting `IHTFRQ` and `IEQFRQ` to 0). This is discussed in further detail below.

A note about the DYNAmics command

Molecular dynamics simulations in CHARMM are run via the `DYNAmics` command. This command takes a lot of options, and for that reason the first six characters of each subcommand name are significant (as opposed to the first four elsewhere in CHARMM). All of the key words listed on this page of the tutorial are options to the dynamics command. Some of the more important options that will be used in almost every MD run are:

- NSTEP <integer>: specifies the number of steps to be run
- TIMESTep <real>: The time step in picoseconds, 0.001 is 1 fs.
- NPRINT <integer>: specifies the frequency at which the energy should be printed out, e.g. NPRINT 100 prints the energy every one hundred steps.
- NSAVC <integer>: frequency to write the coordinates to a trajectory file
- IUNCRD <integer>: Unit number of the coordinate trajectory file (must by opened before DYNAmics are invoked)
- NTRFRQ <integer>: Number of steps to check for and cancel external translation and rotation forces

A complete dynamics set-up might look like:

```
! open the trajectory file
open unit 50 write unform name trajectory.trj

! IHTFRQ & IEQFRQ are, by default 0, so since
! we don't set them here, this runs with the
! NVE ensemble
dyna leap start nstep 1000000 timestep 0.001 -
     nprint 1000 nsavc 50 iuncrd 50 ntrfrq 5000
```

Note: it is not usually correct to run NVE dynamics without prior heating and equilibration (discussed below); this example merely shows the correct way to invoke the dynamics command,

This will run 1 ns of molecular dynamics, saving a coordinate trajectory frame to a file called trajectory.trj every 50 steps (0.05 picoseconds). It is also possible to save the energy, temperature, and velocities to a trajectory, but this is less used in practice. We will talk much more about trajectory files and what to do with them once you have them as the tutorial progresses. The only option that you have not seen here is `START`, which tells CHARMM to start a new run (as opposed to continuing an old run). We'll talk about restarting molecular dynamics simulations below.

DYNAmics, like other CHARMM commands, will use the previously established non-bond configuration unless these are over-ridden in the `DYNAmics` command itself. The recommended practice is to set up your non-bond options in advanced so as not to clutter up the DYNA command (it has enough options already!).

What actually happens during molecular dynamics

Choosing an integrator

The integrator is the method used to numerically integrate Newton's second law of motion during molecular dynamics. CHARMM supports five integrators:

- The leapfrog verlet integrator (keyword `LEAP`): this integrator is similar to the standard 3-step Verlet integrator, but provides additional accuracy.
- The original verlet integrator (keyword `ORIG`): this is the standard verlet integrator. In most cases, LEAP is preferred for its higher accuracy
- The velocity verlet integrator (keyword `VVER`): implements the Verlet algorithm differently. However, it does not print the Hamiltonian during dynamics so results validation is more difficult.
- The 4D verlet (`VER4`) integrator: this algorithm is primarily used with 4-D molecular dynamics, which is beyond the scope of this tutorial
- The velocity verlet 2 (`VV2`) integrator: this integrator is required for use with polarizable force fields using drude particles due to the fact that the drude particles must be integrated independently.

Most molecular dynamics simulations use the LEAP integrator (which supports Langevin dynamics as well).

How the leapfrog Verlet algorithm works

Given the atomic positions (X) at timestep $t$and the velocities (V) at $t - \frac{1}{2}\Delta t$, the leapfrog verlet integrator computes the positions at $t + \Delta t$and the velocity at $t + \frac{1}{2}\Delta t$using the following procedure:

1. Calculate the acceleration on each atom i(a_i), using the formula $a_i = \frac{F_i}{m_i}$where $F_i$is the force on atom i, which is the negative gradient (first derivative) of the energy function and $m_i$is the mass of atom i.
2. Compute the velocity at $t + \frac{1}{2}\Delta t$via the formula $V_{t+\frac{1}{2}\Delta t,i} = V_{t-\frac{1}{2}\Delta t,i} + a_i$
3. Compute the positions at $t + \Delta t$via the equation $X_{t+\Delta t,i} = X_{t,i} + \Delta t V_{t+\frac{1}{2}\Delta t,i}$

In this case Δtrepresents the time step specified by the users; in general, the larger the time step, the less accurate the numerical integration will be. We can see why the algorithm is called a leap-frog since the velocities are computed at midpoints between the time steps.

Further discussion of the leapfrog Verlet and other molecular dynamics algorithms with formulas may be found on the embnet Theory of Molecular Dynamics tutorial [1].

Velocity assignment

If you study the procedure for leapfrog Verlet outlined above, you will notice a potential problem: for the first time step in the simulation, we need to have velocities for all of the atoms. Fortunately, CHARMM has the ability to generate initial velocities for the system or to read them from the COMP coordinate set. In most cases, CHARMM generates initial velocities from a distribution. The distribution is determined by the value of `IASVEL`; if this is greater than 0, a gaussian distribution is used, if it is less than 0 a uniform distribution is used, and if it is equal to 0, initial velocities are read from the COMP coordinate set. Subsequent velocity assignment and rescaling (during heating and equilibration) are controlled by the `IASORS` and `ISCVEL` options to `DYNAmics`. When `IASORS` is 0, velocities are rescaled, otherwise they are reassigned. In the latter case, the method of assignment is determined by the `IASVEL` option, which works the same way as it does for initial velocity assignment. When velocities are rescaled, the rescaling mechanism is controlled by the `ISCVEL` option; when it is 0, a single scale factor is used for all atoms otherwise the scale factor for each atom is dependent on the ratio of the average kinetic energy along each degree of freedom of the atom. Note that velocities will only be assigned or rescaled at the start of a dynamics run so

long as IHTFRQ, IEQFRQ, and ICHECW are all set to 0 (these options are described below).

Temperature and pressure control

Primitive temperature control

CHARMM employs several methods of controlling temperature and pressure, however not all of these are equally good! A primitive method of temperature control can be obtained by using the basic temperature control subcommands. These are IHTFRQ, IEQFRQ, FIRSTT, FINALT, TBATH, TEMINC, and ICHECW. The FIRSTT and FINALT option set the initial and ending temperatures of the simulation (if the temperature of the simulation is expected to remain constant then these should be the same. The TBATH sets the temperature of the external heat bath that is coupled to the simulation. In general it is a good idea to set this to FINALT. TEMINC and IHTFRQ control heating and should be omitted in constant temperature simulations. During a heating run, the temperatures are increased by TEMINC degrees every IHTFRQ steps, which is accomplished by either reassigning or rescaling (depending on the value of IASORS -- see above) the velocities accordingly. IEQFRQ behaves similarly when equilibrating a simulation at constant temperature; velocities are reassigned or rescaled to match the desired temperature every IEQFRQ steps. The primary difference with IEQFRQ is that velocity adjustments are not tied to the heating frequency (IHTFRQ) or the temperature increment (TEMINC), It is possible to further control this rescaling using ICHECW; if ICHECW is 0, the velocities will always be rescaled, but if if it is 1 they will only be rescaled if they are outside a range around the desired temperature. The upper and lower bounds of this range are set by the TWINDH and TWINDL options.

CPT dynamics and the Hoover Thermostat

It is important to note that the primitive temperature control above is **not** sufficient to ensure a NVT statistical ensemble, however it is often good enough for thermal heating and equilibration prior to an NVE dynamics run. For proper NVT dynamics, however, some sort of thermostat must be used. The best thermostat available in CHARMM is the Hoover thermostat, which is part of the constant pressure and temperature (CPT) method, which is enabled by the CPT option to the dynamics command. The hoover thermostat often is used with constant pressure (the PCONS keyword), but this is not a requirement. NVT, as opposed to NPT, can be run even when PCONS is used (see the next paragraph). The reason to do this is to make CHARMM print out pressure statistics (these will not be displayed if PCONS is not used). The weak-coupling Berendsen thermostat (the TCONS option) has a number of known problems and its use is not recommended **under any circumstances**. Both the constant pressure and temperature methods in CPT use the Langevin piston method.

The options to use when applying the hoover thermostat and constant pressure are:

```
dyna cpt pcons pref AAA pgamma BBB pmass CCC hoover reft XXX tmass YYY ...
```

These options can be divided into two parts, those controlling temperature, and those controlling pressure:

Temperature Options

- HOOVER: tells CHARMM to use the Hoover thermostat
- REFT: the temperature at which the thermostat is set, i.e. this is the temperature at which the thermostat will keep the system.
- TMASS: the mass of the temperature piston. The TMASS option is not in atomic mass units but $\frac{kcal * ps^2}{mol}$. We have found that a good value for the piston mass is generally about 20% of the system's mass (which can be found in the ?STOT variable after performing the SCALar MASS STAT command), and this is the value that CHARMMing uses.

Pressure Options

- PCONS: Turns on pressure reporting. The pressure of the system will be held constant unless PMASS is set to 0 (in this case, pressure statistics are still displayed, but constant pressure is not enforced and an NVT ensemble is obtained).
- PREF: The reference pressure in atmospheres. It is usually set to 1 to mimic biological conditions.
- PMASS: The mass of the pressure piston, which **is** measured in atomic mass units (amu). This value can be set to 0 which will disable the pressure control (but pressure will still be reported in the output). In cases when PMASS is non-zero, a good value is generally 2% of the system mass.
- PGAMMA: The Langevin collision frequency for the pressure piston. This is usually set to 0 except during heating to allow the pressure piston to move freely. During heating it is desirable to damp the piston (to prevent the volume from changing too rapidly) by setting a positive PGAMMA.

Writing out trajectories

CHARMM provides a method to save coordinates and velocities of all atoms at fixed intervals during a simulation. In most cases it is not necessary to store velocities, so most users only save coordinate trajectories. The energies and forces for a given time step can be derived from the coordinate trajectory frame. These so-called trajectory files can be used for later analysis, a topic which is discussed further in the analysis section of this tutorial. In order to write out trajectory files, it is necessary to open a unit for writing unformatted (binary) data. This can be done with:

```
open unit 20 write unform name velocities.trj
open unit 21 write unform name coordinates.trj
```

The IUNCRD and IUNVEL options then specify which units are used for coordinate and velocities trajectories. The NSAVC and NSAVV options determine the frequency at which coordinates and velocities are saved, respectively. Therefore, to write velocities and coordinates out every 100 steps, you would do:

```
dyna ... iuncrd 21 iunvel 20 nsavc 100 nsavv 100 ...
```

Some people like to give their trajectory files .dcd extensions because certain third-party software automatically recognizes these as CHARMM-format trejectories, however CHARMM itself does not care what you name the files. One thing to be concerned about is trajectory portability between 32 and 64 bit machines. This is not a problem with newer versions of CHARMM (generally defined as c34 and later), but trajectories generated by older versions compiled with certain versions of gcc might only be readable on the same type of computer as the one that generated them. Trajectory I/O issues caused by size mismatches can often be dealt with using the DYNA FORMAT command. Please refer to dynamc.doc [2] for further details.

Re-starting MD runs

Because molecular dynamics runs can take a long time (potentially months of wall clock time), CHARMM provides the functionality to stop and restart runs via the use of a restart file. The restart file contains the coordinates, velocities, and other state information. If you have a restart file, you can restart a dynamics run by using "DYNA RESTart ..." rather than "DYNA START ...". Please note that dynamics parameters (number of steps, trajectory units, etc.) are not saved in the restart file, and therefore you need to write out the entire DYNAmics command again (or better yet, paste it from your previous script and modify as necessary). If the RESTart options is used than the IUNREAd option must be given to tell CHARMM which unit to read the restart file from. To write out restart file, it is necessary to open the unit (card format) and use the IUNWRIte option to tell CHARMM which unit to use. You can use the ISVFRQ option to determine how often the re-start file will be saved. However, CHARMM will use the same file name for each restart file so if the program crashes while writing the restart file, the file will be corrupted (you can write a script to take snapshots of the restart file, however). Many users prefer to

write one restart file per run and utilize many short runs rather than a single long one.

Here is an example of starting a run and writing out a restart file:

```
open unit 20 write card name restart.res
open unit 50 write unform name run1.dcd

dyna start leap cpt -
  nstep 1000000 timestep 0.001 -       ! 1 ns in 1 fs timesteps
  pcons pmass 0.0 pgamma 0 pref 1.0 -  ! report pressure, but do NVT
  hoover reft 300.0 tmass 500.0 -      ! constant temperature
  iuncrd 50 iunwri 20 nsavc 1000 -     ! trajectory and restart
  iasors 1 iasvel 1 ihtfrq 0 -         ! assign velocities, but
  ieqfrq 0 ichecw 0                    ! do no rescaling (NVT)
```

This command will produce a restart file after one nanosecond of dynamics. To continue the run, you would write:

```
open unit 20 read card name restart.res
open unit 21 write card name restart2.res
open unit 50 write unform name run2.dcd

dyna restart leap cpt -
  iunrea 20 -                         ! unit to read restart file
  nstep 1000000 timestep 0.001 -      ! 1 ns in 1 fs timesteps
  pcons pmass 0.0 pgamma 0 pref 1.0 - ! constant pressure
  hoover reft 300.0 tmass 500.0 -     ! constant temperature
  iuncrd 50 iunwri 21 nsavc 1000 -    ! trajectory and restart
  iasors 1 iasvel 1 ihtfrq 0 -        ! assign velocities,
  ieqfrq 0 ichecw 0                   ! but do no rescaling
```

Note that it is not possible to append to the trajectory file given by the first dynamics command. Instead, you must create a second trajectory and then merge it with the first one (alternatively, CHARMM's analysis tools have very good support for reading a sequence of trajectory files -- this process is described in the analysis section of the tutorial). Also, although `IASVEL` is set to 1, no velocity re-assignment will be done because the velocities are read from the restart file and `IHTFRQ`, `IEQFRQ`, and `ICHECW` are all 0.

Summary and recommendations

Although the `DYNAmics` command is complex, most of its options can be broken down into simple groups controlling the number of steps and the size of each step, temperature and pressure control, setting up of initial velocities, and and reading/writing various control and trajectory files. We recommend setting up non-bond parameters before the main `DYNA` commands. CHARMM's dynamc.doc [2] provides recommendations of what options to use for various purposes. These are reproduced below with some minor fixes to represent currently suggested best practices (in particular #3 has been changed to use the Hoover thermostat, as we do not recommend using the weak-coupling Berendsen thermometer):

1) For heating and early equilibration:

```
DYNAMICS LEAP VERLET RESTART(*)  NSTEP 20000 TIMESTEP 0.001 -
    IPRFRQ 1000 IHTFRQ 100 IEQFRQ 5000 NTRFRQ 5000  -
    IUNREA 30 IUNWRI 31 IUNCRD 50 IUNVEL -1 KUNIT 70 -
    NPRINT 100 NSAVC 100 NSAVV 0 -
```

```
    FIRSTT 100.0 FINALT 300.0 TEMINC 10.0   -
    IASORS 1 IASVEL 1 ISCVEL 0 ICHECW 0 TWINDH 10.0 TWINDL -10.0
```

(*) Except for first run, the use STRT in place of RESTART

2) For late equilibration and analysis runs:

```
DYNAMICS LEAP VERLET RESTART  NSTEP 20000 TIMESTEP 0.001 -
    IPRFRQ 1000 IHTFRQ 0 IEQFRQ 0(*) NTRFRQ 5000  -
    IUNREA 30 IUNWRI 31 IUNCRD 50 IUNVEL -1 KUNIT 70 -
    NPRINT 100 NSAVC 100 NSAVV 0 -
    FIRSTT 100.0 FINALT 300.0 -
    IASORS 0 IASVEL 1 ISCVEL 0 ICHECW 1 TWINDH 10.0 TWINDL -10.0
```

(*) This should probably be positive for an equilibration run.

3) For constant temperature and/or pressure dynamics

```
DYNA LEAP VERLET STRT(*) CPT NSTEP 20000 TIMESTEP 0.001 -
    IPRFRQ 1000 IHTFRQ 0 IEQFRQ 0 NTRFRQ 0  -
    IUNREA 30 IUNWRI 31 IUNCRD 50 IUNVEL -1 KUNIT 70 -
    NPRINT 100 NSAVC 100 NSAVV 0 IHBFRQ 0 INBFRQ 25  -
    PCONS PMASS(+) 50.0 PGAMMA 0 PREF 1.0 -
    HOOVER REFT 300.0 TMASS(+) 500.0 -
    IASORS 0 IASVEL 1 ISCVEL 0 ICHECW 0 TWINDH 0.0 TWINDL 0.0
```

(*) For first run, use RESTART otherwise (+) PMASS and TMASS must be adjusted to match your system size. PMASS should be set to 0 if NVT, instead of NPT, is desired. See above for more information about this.

References

[1] http://www.ch.embnet.org/MD_tutorial/pages/MD.Part1.html
[2] http://www.charmm.org/documentation/current/dynamc.html

LD

Description of Langevin Dynamics

Standard Langevin Dynamics

Langevin dynamics (LD) is similar to molecular dynamics (MD), except that instead of numerically integrating over Newton's second law of motion (the familiar $F = ma$, the Langevin equation is used instead. This equation may be written as:

$$F_i = m_i a_i = -\nabla V_i - \gamma_i + R(t)$$

In the above equation, $-\nabla V$ is the negative gradient of the system (i.e. the force), γ represents the frictional drag on the system, and $R(t)$ represents jostling forces at time t. These random forces have an expected value of 0 and are uncorrelated between time steps. In CHARMM, the random force is specified via setting a collision frequency using the `SCALar` command to set the `FBETa` value, e.g.:

```
scalar fbeta set 60.0
```

sets the collision frequency of 60 per picosecond. This is approximately the viscosity of water at room temperature.

The same sorts of integrators that we saw earlier can be used to perform Langevin Dynamics, although in CHARMM only the original Verlet and leap-frog Verlet ones support this feature.

In Langevin dynamics, the system is coupled to an external heat bath, providing fairly constant temperature throughout the simulation. The temperature of the external bath may be specified via the TBATh option to the DYNAmics command.

Self-guided Langevin dynamics

In self-guided Langevin dynamics (SGLD), the standard LD equation is used, but an additional guiding force, which is a momentum-derived force that pushes atoms, is added. Because of this factor, more conformations will be sampled in a given interval and thus it should only be used if determining the timescale in which events occur is not important. However, the current working hypothesis is that the **order** that events occur is unchanged with SGLD (i.e. only the timescale is distorted).

Considerations on using Langevin Dynamics

The DYNAmics command is used to run both Langevin dynamics and molecular dynamics, so once you have learned how to set up a molecular dynamics simulation, it should be fairly straightforward to run LD as well. However, there are a few differences that must be noted.

- The FBETA scalar value must be set (see above). If this is not done, a default value of 0 will be used and you will effectively be doing regular MD. CHARMM did not used to warn you about this, but more recent versions do.
- An `FBETA` value of 1 or less (significantly lower viscosity than water) is often used for explicitly solvated systems since the presence of water molecules will provide all of the desired frictional effects. A value of 2 is often used for vacuum systems which are not otherwise restrained by the presence of solvent. One might reasonably ask why anyone would want to run LD on a solvated system. In many cases, LD is used for its temperature coupling.
- Since Langevin dynamics has its own temperature control, it should not be used with the constant pressure and temperature (CPT) keyword. Likewise, all CPT specific features (such as PCONS and the HOOVER commands) must be removed from the DYNAmics command. Similarly, IHTFRQ and IEQFRQ should be set to 0.
- Usually, Langevin dynamics uses the leap-frog integrator (key word LEAP), but ORIG will work as well.

To initiate Langevin dynamics, use the LANGevin keyword. The command usually looks something like

```
dyna leap langevin tbath XXX ...
```

where XXX is the desired bath temperature.

Analysis

Some generic comments

In a MD simulation, you follow the atomic motions of a system (such as a solvated protein) over time. Normally, you save this information (the coordinates at a specific timestep) in regular intervals, generating *trajectories*. These trajectories are the key for later analysis. In passing, we note that in addition to coordinates one sometimes also saves the velocities for later analysis.

One obvious thing to do with trajectories is to watch a "molecular movie", using programs such as VMD. Occasionally, watching such a movie may provide interesting insight; most of the time, however, one is left with "information overload"; the human mind is not capable of analyzing in raw form the coordinates of several thousands atoms (or more).

Trajectories, therefore, have to be processed before a meaningful analysis is possible. It is at this step that (intimate) knowledge of statistical mechanics is most needed; one might argue that your command of statistical mechanics (and statistics in general) is the limiting factor for your analyses (note: stat. mech. provides the bridge between microscopic quantities you can derive from the trajectories and macroscopic quantities you can deduce from experiments). In as much as there is no limit to the analyses you can carry out, there can be no guide covering all aspects. There is, however, one type of analysis that is carried out after (or during) almost all MD simulations: monitoring how much the structure of your protein changes compared to the starting structure (be that the original pdb structure, or the structure after some minimization / equilibration): In the following, we'll focus on this task and use it to present some of the most important analysis facilities of CHARMM. In particular, we'll compute the root mean square deviation (RMSD) from the starting structure and monitor the solvent accessible surface area (SASA) and the radius of gyration. As a more advanced example, we'll look at the conservation of secondary structure elements (only recent versions of CHARMM can do this with built-in tools, so we can use this example to illustrate how external tools can be used to aid in the analysis of trajectories (if no suitable built in tool exists!).

Trajectories are not the only useful source of information about a run; many properties such as average potential or kinetic energy can be computed from the output file. Rick Venable has written a useful script for doing this, which may be found in this CHARMM forum post [1].

On trajectory organization

In this section, we try to summarize things to know that are common to all (most) analyses in CHARMM.

A production MD simulation of a solvated protein can require weeks and months of computer time. Obviously, one doesn't trust the computer not to crash during this time, so, usually, the calculation is split into shorter pieces by means of restart files. Thus, at the end of the simulation, one doesn't have a single trajectory, but ten or fifty or even more trajectories. (Trajectories contain vast amounts of data, so even if one had an "ultra-stable" computer with a guaranteed uptime of 6 months, one most likely would have to split the computation since a full trajectory might easily exceed the capacity of some widely used file systems!) The analysis commands in CHARMM can handle such series of trajectories and actually check whether pieces are processed in chronological order. For CHARMM version c35 and earlier, the maximum **guaranteed** number of files (including standard input and output) that can be open at any one time is 99. For the newer, Fortran 95 versions (c36 and up, which are as of this writing not available to the

general public) the limit is whatever the maximum number of open files is for given process as imposed by the operating system. Handling a series of trajectories is done as follows. First, as with all other I/O, trajectories need to be opened first

```
OPEN UNIT 11 NAME system.1.dcd READ UNFO
```

with normal syntax; remember, however, that they are binary files, so you need keyword UNFOrmatted or FILE (instead of FORMatted / CARD) for most other files. If you have only one trajectory, the unit number is (mostly) up to you (under Unix/Linux, you may want to avoid unit numbers 0, 1, 5, 6!). Suppose you have 10 trajectories that you want to process by some command later in one sweep (i.e., the 10 trajectories contain chronological coordinate information about your system). You then have to open the ten files with continuous unit numbers. Let's say that your first unit is 11, then your sequence of OPEN statements should look like

```
OPEN UNIT 11 NAME system.1.dcd READ UNFO
OPEN UNIT 12 NAME system.2.dcd READ UNFO
OPEN UNIT 13 NAME system.3.dcd READ UNFO
...
OPEN UNIT 19 NAME system.9.dcd READ UNFO
OPEN UNIT 20 NAME system.10.dcd READ UNFO
```

Let command be a CHARMM command to process trajectories. You then process your trajectories by a statement like

```
command FIRStu 11 NUNit 10 <other options to command>
```

The option FIRStu tells `command` at which unit number to find the first trajectory file; NUNit provides the information about how many trajectories follow. I.e., we tell `command` to read 10 trajectories found at unit numbers 11, 12, ... 20. Command now reads all frames from the 10 trajectories in order. Upon switching to a new trajectory file, CHARMM parses header information in the trajectories to carry out some sanity checks. In particular, CHARMM expects coordinate frames in chronological order, since many properties one is interested in are time-dependent. This traps, e.g., the following mistake: Suppose you confused the order of trajectories, as in

```
! BAD EXAMPLE
OPEN UNIT 11 NAME system.1.dcd READ UNFO
OPEN UNIT 12 NAME system.3.dcd READ UNFO ! ERROR: example of mistake
OPEN UNIT 13 NAME system.2.dcd READ UNFO
...
```

```
command FIRStu 11 NUNit 10 <other options to command>
```

CHARMM will process the first trajectory. When switching to unit 12, connected to system.3.dcd, CHARMM will note that the first coordinate set corresponds not the one it expects following the end of system.1.dcd, ... and you'll crash!

Unit numbers are actually a fairly scarce resource. Once you are done with the trajectories, you should, therefore, CLOSe all trajetories! BTW, OPENing and CLOSing of 10 or 50 trajectories one at a time is not fun. You should quickly consider writing a loop, which is described in the basic CHARMM scripting section.

Going back for one step, we should say a few words about the prerequisites for doing a trajectory analysis. The general rule of the thumb is: Set up your system *exactly* as you set it up during the generation of the trajectories. In particular, read the same PSF and set the same energy options. If you used PBC and you're performing an analysis that uses crystal and image properties, set these up with CRYSTAL/IMAGe as during MD. For good measure you may want to add SHAKe and other restraints you had during the MD. There are exceptions and border line cases

where this doesn't apply, but in 90% of the cases this should work. Thus, an analysis run has typically a structure like the following:

```
! read rtf
! read params
! read PSF
! read some starting coordinates (necessary when using PBC (CRYStal),
       ! otherwise optional

CRYS DEFI <as during MD>
CRYS BUIL
IMAG BYSE ...
IMAG BYRE ...

ENER <all nonbonded options>

! start analysis

! Let there be 10 trajectories
OPEN UNIT 11 NAME system.1.dcd READ UNFO
OPEN UNIT 12 NAME system.2.dcd READ UNFO
OPEN UNIT 13 NAME system.3.dcd READ UNFO
! ... 7 more OPEN statements

command FIRStu 11 NUNit 10 <other options to command>

CLOSe UNIT 11
CLOSe Unit 12
! ... 8 more CLOSe statements
```

Carefully onsulting the documentation of the analysis commands that you want to use is highly recommended. Pay particular attention to whether CRYSTal and IMAGEs need to be set up for that particular analysis and if so be sure to set them up exactly as they were for MD. If these are not needed then you can save some time by not setting them up, particularly if you are running lots of analyses back to back.

Some basic analyses

CORREL

CORREL is a generalized facility for analyzing trajectories. The subsystem is invoked with the CORREL command, which may take several arguments. It is usually necessary to set the maximum number of atoms (MAXAtoms), and it may be necessary to also adjust the maximum number of timesteps (MAXTimesteps) and series (MAXSeries) if the defaults are insufficient. However, settings these values too high will result in lots of memory being allocated, possibly leading to a crash if the system does not have sufficient RAM.

What CORREL can track

One of the main concepts used by CORREL is the concept of the time series. A time series describes the value of a property such as a bond length, the total energy of the system, or the distance between two atoms, over the course of the simulation. Time series are created via the ENTEr command, which takes a number of different options depending on what time series data is needed. The basic syntax for the command is:

```
ENTEr <name> <type of data> -
  <atom selection and subcommands>
```

Each time series must be given a unique name. Then the user must specify the type of data that is to be collected. An exhaustive listing of supported data types is in correl.doc [2], but some commonly used ones are:

- BOND for the length between two bonded atoms
- ANGLe for the angle between three atoms
- DIHEdral and IMPRoper
- RMS, for the root mean squared deviation of the system
- ENERgy, for total system energy
- SDIP for the dipole moment of the solvent shell

The atom selection and subcommands differ based on the type of data that is needed, e.g. DIHEdral requires a selection of four atoms while ENERgy requires no further arguments. Consulting the documentation is recommended as a number of data types have different subcommands (e.g. whether or not to mass weight the RMSD).

Once all desired time series are set up via ENTEr, it is necessary to tell CORREL which trajectory or trajectories will be used to populate the time series data. This is done via the TRAJectory command. The basic syntax is:

```
TRAJectory FIRSTu <x> NUNIt <y> SKIP <skip> -
  BEGIn <start step> STOP <end step>
```

Where <x> is the first unit in the series of trajectories and <y> is the number of units to read; following the previous example x is 11 and y is 10. This is why it is important that trajectory unit numbers be consecutive and in order!

Once the TRAJectory command is issued, the time series data will be populated and can be further used.

Display and Manipulation of Trajectory Data

Once the time series are generated, they can be written out to a file or manipulated further. To write out trajectory data, simply open a unit and tell CHARMM to write the data:

```
correl maxt X
enter psi torsion segid resid atomtype1 segid resid atomtype2 segid resid atomtype3 segid resid atomtype4 geometry
traj firstu 10 nunit 1 begin 100 stop 21000000 skip 10000
write psi card unit 21
* psi of residue 1
*
```

Obviously, you must replace segid, resid, and the atom numbers with the correct segments, residues, and atom types for your system.

The resulting data file may be viewed or plotted in an external program such as GNUPlot or XMGrace.

CHARMM also containes facilities for manipulating the data, via the MANTIME command.

Useful properties that can be calculated

Based only on the limited work with CORREL that has been presented so far, several useful properties may be computed from simulation.

Mean energy

The average energy, denoted <E> or $\bar{E}$is just the average energy value over the course of the simulation.

$$\bar{E} = \frac{1}{n}\sum_{i=1}^{n} E_i$$

This can be calculated via extracting the energy at each time step from the trajectory and averaging them. However, a more precise way of calculating this value is to look in the CHARMM output file for lines beginning with "DYNA AVER>". Usually, CHARMM prints out the average values (including the energy) every 1000 steps (unless you tell CHARMM to behave differently -- see the molecular dynamics page for details). These average values are only for the past 1000 (or however many) steps. Using the "DYNA AVER" values will give you a much greater sample size, unless you are saving out every frame into the trajectory file.

The average energy is of primary interest in NVT simulations to determine how much total energy fluctuates at a given temperature, which is useful for determining various statistical mechanics properties. However, it is also interesting in NVE simulations to see the proportion of the total, constant energy that is tied up as potential energy.

Mean structure

The mean structure is just the average coordinate of each molecule, which may be found in a similar manner as the mean energy. The only difference is that the average structure can be mass-weighted.

Root mean square fluctuation

The root mean squared fluctuation is just the average difference between all particles and their average position for a given time step.

Radius of gyration

The radius of gyration measures the average distance between an atom and its center of mass at a given time step. It is used as another measure of how much atoms move around during a simulation.

Next steps

An example of analyzing a molecular dynamics trajectory is given in the full example. It will show you how to use `CORREL` to graph vvarious properties. We will also introduce more advanced features, such as time correlation functions and manipulating trajectories. However, a detailed discussion of how to analyze a simulation is beyond the scope of this tutorial.

References

[1] http://www.charmm.org/ubbthreads-7-5-5/ubbthreads.php?ubb=showflat&Number=24462
[2] http://www.charmm.org/documentation/current/correl.html

Full example

This section of the tutorial will present an example of a real molecular dynamics simulation using the protein Crambin (PDB ID 1CBN). You will be able to download the input scripts as you read along. These scripts are based off of the inputs generated by CHARMMing with additional comments added and some complexity that is only needed in an automated system removed. You should run these on your own computer to get a feel for the type of output you can expect to see from CHARMM. Suggested exercises are given to encourage you to modify the scripts in various ways. It is highly recommended that you download the tarball containing all of the inputs, outputs, and other files needed before beginning to work through this example. All of the files are available in tar.gz format [1] or zip format [2].

Reading in the Crambin structure

The first step of any simulation is getting your structure read into CHARMM. Unfortunately, it is not always straightforward to just get a PDB file from the Protein Data Bank [4] and load it up. This is mainly because the PDB uses different naming conventions than CHARMM for some atoms and residues. Fortunately, there are a number of tools that can help with this such as the conv_pdb.pl script in MMTSB or the standalone PDB parser from CHARMMing, which you can download from this site [3]. In this case, we will start with a version of 1CBN that has been run through CHARMMing's parser. In order to keep things simple, we will use only the protein atoms and not the bound ligand (the hetero atoms). The parsed PDB with the HETATMs removed can be found here [4].

Reading in the PDB file

Background

Correctly reading in a structure from the PDB can be one of the trickiest parts of using CHARMM because of differences in format and nomenclature between PDB files and what CHARMM expects. CHARMMing [2] uses an automated script to format the PDB into an amenable input source for CHARMM. However, this automated procedure has some drawbacks, namely residue and atom number is not preserved (we intend to fix this in the near future). Also, there will be occasional instances where it will not work quite right for some reason or another. In this case you will need to "roll your own"and the examples in the script archive of the CHARMM forums should be of some use. In particular, see the complex PDB input [5] post by Rick Venable. As a general rule, the script archive is a good place to look for examples of how to do things in CHARMM.

Our script

The first input script to be used is setup.inp [6], which reads the sequence and coordinates from the PDB file and creates the other data structures needed by CHARMM. Most of this script should be reasonably straightforward, however there are a couple of commands that have not been seen previously:

- `READ SEQUence` tells CHARMM to read the sequence for the macromolecule from the file. For example, the sequence "ALA ALA" would refer to an alanine dipeptide.
- The `GENErate` command is used to create a new segment in the protein structure file. In this script, we do not read in a PSF, so the `GENErate` command effectively creates the PSF.
  - The `SETUp` option to `GENErate` tells CHARMM to create the internal coordinate (IC) tables.
  - a-pro is the identifier we give to the segment that we are generating (this is called the SEGID and is limited to 5 characters).
  - `first nter last cter` determine the terminal group patchings for the segments. NTER and CTER are the defaults for proteins and refer to the capping at each end of the peptide chain.

- The REWInd command tells CHARMM to begin reading a file again from the beginning (handy when you want to read two types of data from a file as we do here)!
- The IC commands in this file tell CHARMM to fill in missing data in the internal coordinate table with information from the parameter sets (bonds, angles, etc. being placed at their minimum). This data can then be used to build cartesian coordinates. However, since the PDB file already has cartesian coordinates for all of the atoms except the hydrogens, it is not really needed here (but it does not hurt to have it).
- The `HBUIld` command places the hydrogens relative to the heavy atom positions. Most PDB files do not have hydrogen positions, so this command is used within CHARMM to build them. Note that the hydrogen positions may not be optimal, so it is best to minimize these following the set-up.

Once we have all of the coordinates in place, we go ahead and save out our newly created PSF and coordinate files.

Making the final structure

Now that we have the protein segment read in, there is still a little bit more to do to get a really correct structure that we can work with. In many cases, a protein will have more than one segment and they will all need to be prepared separately, read in, and appended together to make a final structure. This can be done with the APPEnd option to READ PSF and READ COOR. In this case, however, we only have a single segment to read in so this step is somewhat superfluous. It is presented here only because in a lot of cases you will need to do it. However, there is one important thing we must do now! If you look at the original 1CBN PDB from the Protein Data Bank, you will notice some lines in the header that begin with `SSBOND`. These lines indicate that there are disulfide bridges present in the protein, which can have a substantial effect on the behavior of participating residues. These must therefore be added, which is done via the PATCh command in CHARMM. There are other types of patches we could include as well -- these are mostly used to change the protonation state of a residue. For example, the ASPP patch adds an extra proton to an Aspertine residue, giving it a net +1 charge.

In this case, however, we just want to apply the disulfide bridge patches. The input script we have to do all of this is append.inp [7]. When you run it, be careful to note the output from the `PATCh` command to make sure that the patch was applied successfully. CHARMM does not always stop processing when a patch application fails, which can lead to strange problems in later scripts. There are a couple of syntactic points to note as well:

- Note the syntax of the `PATCh` command: `PATCh <patch name> <residue selection(s)> <other options>`. In this case `DISUL` (for disulfide) is the name of the patch, 2 selections are needed (since a disulfide bridge links two residues, and the `SETUp` options is used again because the IC table is modified and thus needs to be re-created.
- We delete the hydrogen atom coordinates and re-place them once the patching is done. In this case this is not really necessary, however when appending multiple segments it is a good idea to re-create the hydrogen atoms coordinates with the whole structure in place. This will reduce bad contacts between the hydrogens and heavy atoms.
- The `IOFOrm EXTEnded` command tells CHARMM to print the coordinate files out in high precision (note: older versions of CHARMM may have trouble reading these in).

Vacuum minimization

Basic notes

Now that the structure has been prepared for use with CHARMM, the next step is to minimize it. The input script (initial-minimization.inp) may be found here [8] and you may also download the output file [9]. As mentioned in the Minimization section of the tutorial, there are several different methods for performing minimization. This script implements a fairly straightforward routine of 100 steps of steepest descent followed by 1000 steps of `ABNR` with a gradient tolerance of 0.01. After the minimization is completed, the COOR RMS command is used to show the

difference between the initial and final structures (the original coordinates are copied to the comparison coordinate set (COMP) set via the COOR COPY COMP command that is issued prior to minimization; by default COOR RMS prints out the root mean square distance between the coordinates in the main and comp sets).

The PSF and initial coordinates are read in from the files created by the append script. We only need to write out a new coordinate file since minimizing the structure does not change the coordinates.

For real production use, this may be too much vacuum minimization. Restraints might need to be applied to keep the structure from twisting too far out of shape. Remember that the examples in this tutorial serve illustrative purposes only. **You** are responsible for examining your outputs and visualizing your structures after every step to make sure that they make sense in the context of the work you are doing. Adding restraints is given as an exercise below.

Exercises

1. Modify the minimization script to do three separate rounds of minimization: (1) minimize only the hydrogens, fixing the heavy atoms, (2) minimize with the backbone C, N, and O atoms fixed and all other atoms free to move, (3) minimize with all atoms free. Which minimization reduces system energy the most?
2. Use COOR DIFF to get the atom by atom difference between the initial coordinates and final coordinates. Which atoms moved the most? The least?

Solvation and neutralization

The first simulation we'll set up for this tutorial will be molecular dynamics in explicit solvent. In order to continue on, we need to place a water structure around the protein. Although Crambin is neutral, we will also place it in a 0.15 molar solution of potassium chloride to illustrate the process of neutralization.

Solvation

The input file for solvation is Solvate.inp [10] (the corresponding output file can be downloaded here [11]). We have chosen to solvate the Crambin in a rhombic dodecahedron (RHDO) because it is the most globular. As seen earlier, the RHDO approaches the efficiency of a sphere, and so it is a good choice for this system. A cubic or rectangular box would have worked well too.

There are a few points of interest in this script:

1. After the structure is read in, the COORdinate ORIEnt command is issued which centers the structure at the origin, making the longest axis of the protein correspond to the Cartesian X-axis and the second longest axis correspond to the Cartesian Y-axis. In our case this is OK because the water box is centered. If the water box is far off-center this can move the protein coordinates in an undesirable way. It is best to make sure that the water is pre-centered, but failing this you can do a COOR STAT on the water and then COOR TRANS to translate the water structure such that it is roughly centered.
2. The dimensions of the three axes are determined via the CALC command. CALC allows for mathematical manipulation within CHARMM scripts
3. The method of creating the water structure is that outlined on the solvation page:
    1. Delete the waters whose oxygen atoms are within 2.5 Å of the protein (note that the waters we read in are generated as segment BWAT for easy identification later). We do this by selecting .BYRES. all OH2 atoms within 2.5 angstroms of any atom that is not in segid BWAT or is a hydrogen. Pay careful attention to the syntax of the select statement -- this can be a bit tricky!
    2. Calculate the radius of a circle that will circumscribe the final RHDO (we call the radius variable @safesphere in the script) and then delete all waters outside it.
    3. Create the crystal structure using CRYStal DEFIne and CRYStal BUILD

4. Use the IMAGe command to force the whole BWAT segment to be updated by residue (BYRES). Recall that this means any water molecules in BWAT that are outside the unit cell will be translated back inside it when images are updated.
5. Save the current coordinates and use the UPDAte command to move the waters that are outside the RHDO back inside it.
6. Detect which waters have moved (via COOR DIFF) and delete them as extraneous.

4. Minimize the final structure slightly to remove bad contacts. An alternate approach is to fix (useing CONS FIX) or restrain (using CONS HARM or similar; these are discussed on the essential CHARMM features page) the Crambin and minimize only the solvent in this step.

This script, while complex, should be easily adoptable to different situations. As an **exercise**, try solvating Crambin in a cubic box rather than an RHDO. You should visualize both the cubic and the RHDO structures to make sure that the protein is entirely surrounded by water with a sufficient buffer to ensure that the Crambin won't interact directly with an image of itself.

Neutralization

The neutralize.inp [12] script is slightly simplified from the version used by CHARMMing, mainly by removing support for triatomic salts such as $MgCL_2$. In this script we limit ourselves to the diatomic salt Potassium Chloride (KCl). Nonetheless, this is still one of the most complex scripts used by CHARMMing. Once you have the input script you may also download its output file [13].

As mentioned on the Neutralization page, the basic idea of this script is to run a certain number of trials, each one of which replaces a number of water molecules with counter-ions in order to achieve the desired concentration. The trial that yields the lowest energy (after a brief minimization) is then used. The first thing that is done is to calculate the volume used by the entire system and the volume taken up by the protein, which is accomplished by the COOR VOLUme command (this is necessary to get the concentration right). As stated in the documentation, we must store the radii of each atom in the first scalar array and the outer probe diameter in the second. We also need to give a maximum number of cubic pixels via the SPACE option. A larger SPACE value gives a more accurate answer at the cost of more CPU time and memory. We use 168 times the number of atoms, which is a reasonable heuristic. The total volume from COOR VOLUme is the sum of the ?VOLUME (space taken up by the atoms themselves) and ?FREEVOL (space in between them) CHARMM variables. When we have the total volume for the whole system and for the protein, their difference is the volume of the solvent -- this is the space we have to insert the ions.

Once we know how much volume that we are dealing with, it is necessary to figure out how many ions are needed. This is a fairly straightforward calculation based on Avogadro's number, with the total number of ions needed to achieve the desired concentration is stored in the @totions variable. The number of charges necessary is then added (or subtracted) from @totions depending on whether we need to add or subtract cations (positive charges). The final result is stored in the @nions variable. We then make sure that an even number of charges are added (since we have to pair positive and negative charges except for the any additional ions needed to neutralize the system). Finally we add negative or subtract positive charges as needed to neutralize the system, storing the very final number of positive and negative ions in @ipos and @ineg.

Once we know how many ions that we need to place, we enter into the main Monte-Carlo loop. Right at the beginning, we use a neat trick to redirect the output to another file to keep from cluttering up our "main" output file. To do so, we open up a new unit for writing and then use CHARMM's OUTUnit command to tell the program to send its output to the new unit we opened up. When we're done with the loop (at the end of the script), we issue:

```
outu 6
```

to tell CHARMM to go back to sending its output to unit 6 (which is standard output on Unix-like systems).

In each iteration of the loop, the PSF and coordinates from solvation are re-read so we start fresh each time. We then call a **stream file** (addions.str). Stream files are just regular CHARMM scripts, but they can be called by other CHARMM scripts (like a subroutine in other programming languages). This allows us to modularize CHARMM scripts and re-use different pieces of code without copying and pasting them into different scripts. In this case the addions.str [14] stream file handles the actual placing of the ions. To do so, it first generates two new segments, one for the positive ions and one for the negative ions. All of the coordinates for the atoms in these segments are initially set to zero via the COOR SET command.

To actually place the ions, the script uses two loops, one for the cations and one for the anions. The first step in each loop is to select which water molecules are candidates for being replaced. We select all of the oxygens (type OH2) that are part of water moleculaes that are at least 5.5 Å away from the protein and from another ion (notice how SELEct AROUnd comes in handy). These are the candidates for deletion! We then get a random number via the CHARMM ?RAND variable (which will produce a new random number between 0 and 1 each time that it is used). To convert this random number to an atom number we multiply it by the number of candidates for deletion, stripping out any fractional part (via the `INTeger` function which can be used with `CALC`). This gives us the variable `@in` -- the index of the atom within the selection to replace. The `SUBSET` option is then used in the `SELECT` command to select the `@{in}`th atom in the selection. We save its coordinates, assign them to the ion, and then delete the water molecule by using `BYRES` in the delete command (so that the whole water molecule is deleted, not just the oxygen).

After the ions are placed, we perform a brief minimization (10 steps of SD plus 25 steps of ABNR). Since the PSF has been cleared and reloaded, we must also redo all of the crystal setup and shake (for ABNR). In general, whenever the PSF is modified in any way existing constraints and possibly crystal data is cleared. When in doubt, repeat the crystal set-up! This is also why it's usually a good idea not to muck with the PSF within a script unless you know you need to. We then compare the minimum energy value to the previously realized minimum energy (stored in `@emin`, which is initially set to a ridiculously high energy value). If it is lower, we write out the new structure; if not, we simply move on, discarding it.

Exercises

1. Practice solvating Crambin in various types of structures (box, rectangle, hexagonal prism, octahedron). Make sure the protein can rotate 360 degrees without leaving the solvated box. Which structure is most efficient (highest ratio of protein volume to total volume)? Which is least efficient?
2. Try altering the ion concentration and type of ion used (consult the CHARMM topology file for a list of supported ions). Can you cause an error by adding too many or too few ions? If so, how might this be avoided in the script?

Minimization with PBC

With solvation and neutralization complete, we now have the very final PSF that we will use for dynamics. However, the very brief minimizations done as part of solvation and neutralization are probably not sufficient to completely remove bad contacts, therefore we need to do a more thorough minimization of this structure to place it at a lower potential energy. You can download the input [15] and output files [16].

Comments on setting up PBC

Because we are planning to run molecular dynamics with periodic boundary conditions (PBC), it is necessary to perform our final minimization under the same conditions. There are three basic steps steps for this: (1) defining and building the crystal structure, (2) setting up images, and (3) configuring the nonbond parameters. As in the case of neutralization, we must be careful to set the crystal parameters to their correct values from solvation, so we just copy the crystal dimension we calculate from the solvation script and re-do CRYSTal BUILd. Setting up images requires

us to define which parts of the system for which we want image atoms created. We also need to define how image centering will work, i.e. how atoms that move outside of the unit cell should be treated. As a simulation progresses, some non-image atoms may move far away from the primary structure; recentering allows CHARMM to pick new "primary" atoms from among the images of these based upon their distance from the rest of the primary structure. In order to activate this feature, the user must explicitly state which atoms are allowed to be re-centered (the default is not to re-center anything); atoms can be re-centered two ways: by segment and by residue. In general, connected residues should be recentered by segment (after all, you don't want to break up a polymer chain along a backbone bond!) and non-connected segments (e.g. solvent) recentered by residue. This is the approach we take in our script.

Once the crystal structure and images are set up, we can issue the NBONDs command to set the nonbond parameters.

Exercises

1. Try minimizing only the solvent, keeping the Crambin fixed. How close to the minimum energy do you get simply by optimizing the solvent positions?
2. Find the RMSD between the Crambin minimized in vacuum and the Crambin minimized freely in the presence of solvent. How does the minimum energy structure change in the presence of solvent?
3. Go back to the neutralization step and increase the ion concentration, and then re-minimize. Does Crambin have any charged surface residues? How do these affect the optimal positions of the ions?
4. Try moving one of the ions right next to the protein. What happens to it during the course of the minimization?

Dynamics: heating

Heating is performed via the md-heat.inp [17] script and its output file [18]. The goal of this script is to heat the system from 110K to 310K (since the low energy conformation from minimization represents a structure that would likely be present at a lower temperature). The choice of 110K is somewhat arbitrary; it's mostly based on the fact that it is 200K lower than a reasonable physiological temperature which makes the temperature incrementing scheme use round numbers. For problematic structures, it may be necessary to start at a lower temperature, decrease the heating increments, or reduce the timestep to lower than 1 femtosecond (e.g. 0.5 fs).

Pre-dynamics Setup

The setup for dynamics looks very similar to that used in the final minimization with PBC. We once again need to set up crystals, images, and nonbond parameters. The only new thing here is that we open up two units for writing: unit 31 for the trajectory file and unit 41 for the restart file. Make sure you open these in the correct manner; trajectory files are written as binary data and must be opened `UNFOrmatted` (`FILE` is a synonym for `UNFOrmatted`) while the restart file is plain text and can be opened with the CARD keyword.

The DYNAmics Command

If you paid attention to the molecular dynamics page in this tutorial, this should look fairly straightforward. The options to the `DYNAmics` command may be given in any order, but to make this script more legible I've grouped them by functionality. The first two lines set up basic parameters (we want to run for 100 picoseconds (100,000 steps with a 1 femtosecond timestep) while writing coordinates to the trajectory and printing the status periodically (100 and 1000 steps, respectively). We have to give the unit numbers for the restart and coordinate files to the `IUNWRI` and `IUNCRD` options.

The third line (starts with "firstt") and fourth line determine our heating schedule. The temperature is increased by 5K every 2500 steps for a total increase of 200K over the course of the run. Note that this is a much more gradual heating than that set up by default by the publically accessible CHARMMing install on www.charmming.org [2], as this installation is limited to 1000 steps per dynamics run -- users who install their own CHARMMing server may set

the limit higher). The temperature bath is set to the final temperature, but we do not do any equilibration (`IEQFRQ` is 0), just heating. The fifth line handles initial velocity assignment. In this case velocities are assigned (not rescaled, since `ISCVEL` = 0) using a Gaussian distribution (`IASVEL` = 1). We do not want to impose any external checking of the temperature, so we set `ICHECW` to 0. `ICHECW` can be used to conditionally rescale or reassign velocities if the temperature moves outside of a predetermined range. The last three lines all control important options: `NTRFRQ` determines the frequency at which CHARMM removes net rotation and translation from the system. This is necessary when PME is used since otherwise the system's energy can be diverted to this motion, leading to a "flying ice cube." `NTRFRQ` should be set to somewhere between a few dozen and a few thousand steps; it's not clear what an optimal value is, but it should be set to **something** within this range. The `ISEED` option controls the seeds for the random number generator. Two simulations with identical starting conditions done with the same seed should yield identical results (numerical rounding aside). Finally the `ECHECK` option tells CHARMM to abort the simulation if total energy shifts more than 100 kcal/mol in one step. This is to provide a watchdog against runaway energy changes which may be the result of model building errors. Depending on the size of your system and the type of simulation being run, it will probably be necessary to adjust this value.

Interpreting the Output

Until now, output from the scripts has been relatively short and straightforward to interpret. However, since the output from DYNAmics is quite long and involved, it makes sense to spend a little time figuring out exactly what's what.

Immediately after the DYNAmics command, CHARMM prints out the options it is going to use. Reading these is a good way to check that everything really is set up the way that you want it (and that you didn't make any typos in your DYNAmics command). After this is the output from the dynamics run itself. Every NPRINT steps CHARMM will print out the energy at that particular step as well as the amount of time (in picoseconds) that has been simulated and the current temperature. It will also print the averages of these over all of the time steps since the last print out in lines beginning with "AVER" and their fluctuations in lines beginning with "FLUC". When calculating averages, it is important to use the average values since only these reflect all of the time steps in the simulation. Additionally, every `NTRFRQ` steps (in this case 500) you will see a group of lines beginning with "DETAILS ABOUT THE CENTER OF MASS" -- this data is used by CHARMM to stop translational and rotational motion. Also, periodically you will see lines such as:

```
SELECTED IMAGES ATOMS BEING CENTERED ABOUT 0.00000 0.00000 0.00000
UPDECI: Nonbond update at step       176
UPDECI: Image update at step       482
```

The first line indicates that the system is being recentered in the unit cell (Remember the `IMAGE BYRES` and `IMAGE BYSEG` commands we used in the set-up? They are at work here!). The next two lines indicate that the nonbond and image atom lists are being updated. We set `INBFRQ` and `IMGFRQ` to -1 in our `NBONDs` command, which means that these updates do not occur on any particular schedule, but are done heuristically when the atoms in the system have moved enough.

If you look to the end of the output, you'll see that molecular dynamics can take some time, even for a relatively short simulation like this one. I ran this script in parallel (4 processors) on a fairly modern machine and it still took over 3 hours to complete.

Dynamics: equilibration and production runs

Now that the system is heated up, it is usually wise to let it "settle in" and the desired temperature via an equilibration run. There are several ways of doing this (including using `IEQFRQ` in the NVE ensemble), but the method we use here is to run NVT using the Hoover thermostat. This is the same script that would be used for an NVT production run with Hoover, but it's often wise to let a system equilibrate before collecting statistics from the trajectory. We use the md-equil.inp [19] script to run this simulation, and you can find the output here [20] (note that the equilibration output is large enough that it has been gzipped to save space). Some people like to use the NPT ensemble for equilibration and run with this ensemble until the volume becomes stable. To modify this script to use NPT, use must set a positive value for `PGAMMA`,

The three main differences between this scripts and the heating script are:

1. The Hoover thermostat is used for temperature control; IHTFRQ and IEQFRQ are not used.
2. Pressure statistics are reported (via the `PCONS` option), but constant pressure is not imposed (`PGAMMA` = 0)
3. The Dynamics run is restarted, rather than started fresh
4. We do 200 picoseconds of dynamics instead of 100

The crystal, image, and nonbond setups are the same as for heating, as you would expect.

Restarting a Dynamics run

Restarting a dynamics run is relatively easy. We open the restart file from the heating run and then pass its unit number to the IUNREA option to DYNAmics. CHARMM tells us this worked with the lines:

```
READYN> dynamics restart file was read. Current step=  100000
NSTEP  =100000  JHSTRT =     0
```

The number of steps already run is tracked in the restart file. All of the simulation times in the dynamics output will be global (i.e., they will include the time in previous simulations). We save out a new restart file at the end of this run in case we want to extend it further. One word of advice: develop a naming convention for your restart files and stick to it so you always know which one to use. One method is to number them sequentially, e.g. dyn1.res, dyn2.res, etc. Note that we name our restart file 1cbn-equil-1.dcd for this reason.

Using the Hoover thermostat and constant pressure

The Hoover thermostat applies strict temperature control to the simulation and is generally used for NVT runs. It requires the `CPT` keyword, even though hoover can be run without constant pressure (as, indeed, we do). The line:

```
hoover reft 310.0 tmass @tmass
```

tells CHARMM to use it. Note that the `TCONS` keyword will invoke the weak-coupling Berendsen thermostat and **should not be used**! We choose to let the simulation run at 310K which is the final temperature from the heating run (the temperature will still fluctuate slightly, however, particularly for small systems). Also IHTFRQ and IEQFRQ are both set to 0 so CHARMM will never try to use velocity assignment or rescaling for temperature control (even though IASORS and IASVEL are turned on, they have no effect because IHTFRQ and IEQFRQ are 0 and we are restarting the simulation). Likewise, pressure reporting is invoked with:

```
pcons pint pref 1.0 pmass @pmass pgamma 0.0
```

Since PGAMMA (piston collision frequency) is 0, we aren't actually using constant pressure.

Trajectory Analysis

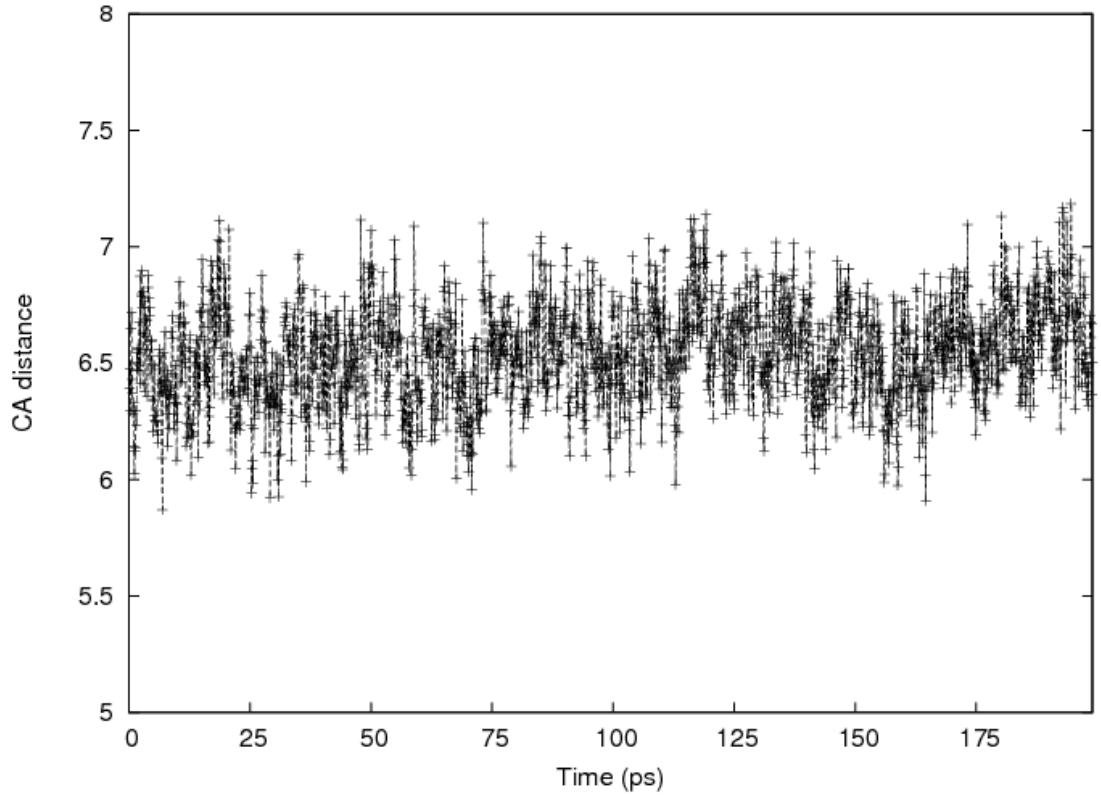
A basic file for doing analysis can be found here [21] (output file is here [22]). This file runs through some basic analysis using the CORREL command. The end result is to get three different properties from the simulation.

- The radius of gyration over time
- The distance between the alpha-carbons of residues 10 and 46
- The time correlation function of this distance

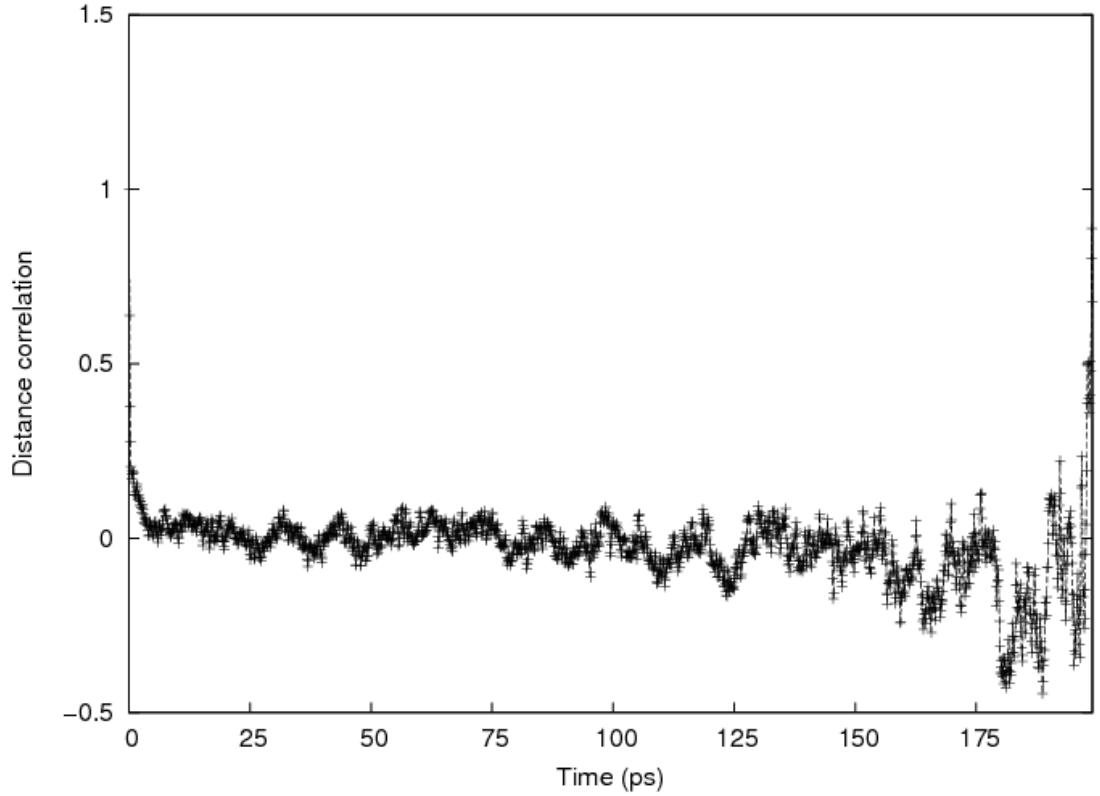
Residues 10 and 46 were chosen because some literature suggests that a disulfide bridge between between these residues has implications for the physical behavior of the macromolecule (see Bang et al, *Mol Biosyst.* 2009 Jul;5(7):750-6. Epub 2009 May 28). Extracting data from the trajectories takes several steps:

- we first must set up the system using the exact same parameters as used with the MD simulation. This is because energies are not stored in trajectory files, just coordinates or velocities; we have to be able to calculate the energy under the same conditions as the MD run.
- the trajectory file from dynamics is opened for reading
- we then open a log file for each of the three properties. The log file will have the format "time: value" which can easily be plotted by GNUPLOT or a similar plotting program.
- correl itself is set up. We could use `MAXSeries 2` as we will only be working with three sets of data: the radius of gyration, the residue distance, and the time correlation of the distance, the last of which has its own special series set up by CHARMM. We will only be looking at a few atoms, so `MAXAtoms` is 50, likewise, there are 2000 time steps in our trajectory file. These numbers will change from run to run depending on what you want to look at.
- We use `ENTEr` to set up two sets of data, one for the radius of gyration (we name this data set "rgyr", `GYRA` is the key word that tells CHARMM this data set is to contain the radius of gyration at each step. Likewise a set for the CA-CA distance is created (the set is named "dist", the `BOND` keyword tells CHARMM that this set will contain the distance between two atoms (even though they are not bonded). The atom selection tells CHARMM which two atoms we want.
- We fill in the sets by reading from the trajectory file opened earlier
- We write both sets of data to their output files (we do this now since we need to modify the "dist" set to compute the time correlation).
- The `MANTime` command is used to subtract the average off from each element of the "dist" set (so each data point becomes a displacement from the mean).
- The `CORFUN` command is used to calculate the correlation of the "dist" data. It is implicitly saved in a set called "corr" (if we call `CORFUN` again before writing the data out, it will be lost!)
- We write the "corr" data set out in the same manner as "dist" and "rgyr"

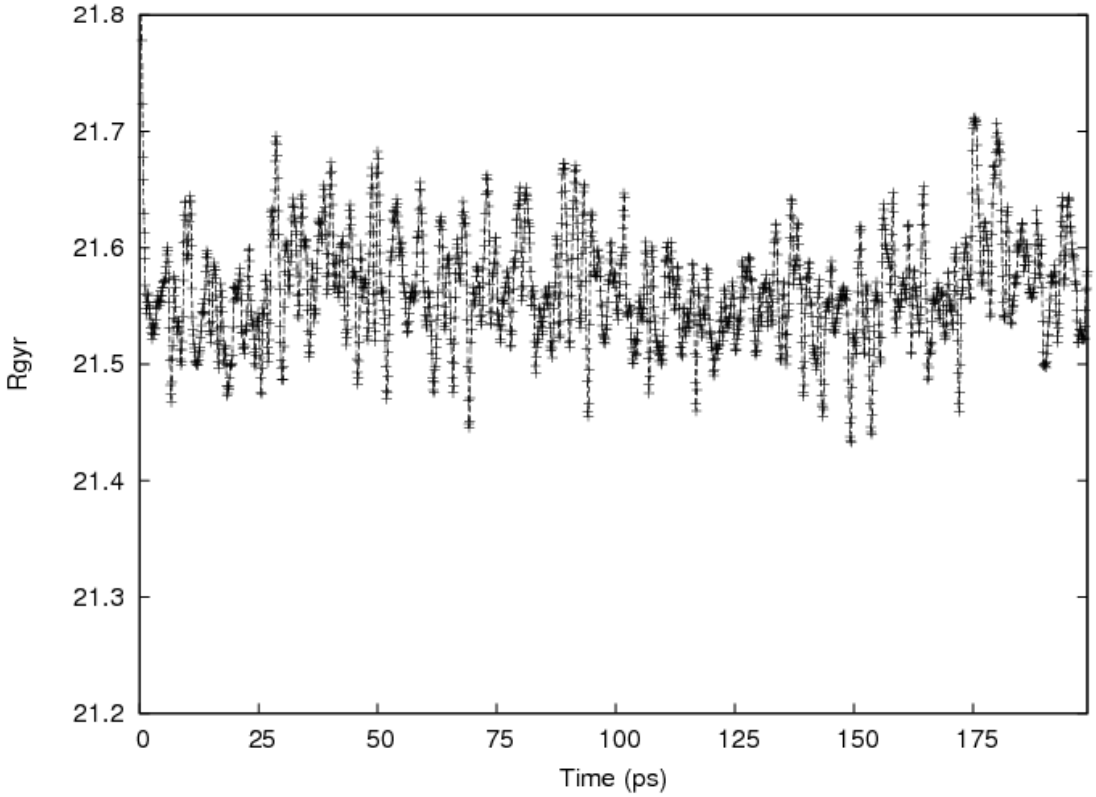
Below are plots of the distance, radius of gyration, and time correlation:



CA-CA distance between residues 10 and 46 of 1CBN from our dynamics trajectory.



Time correlation calculated from the above data



Radius of gyration of 1CBN from our dynamics trajectory.

The GNUPLOT scripts used to generate these images from the raw data files are available in the complete tarball of this example.

These graphs show that the distance between the alpha carbons of the two residues fluctuates around an average value slightly greater than 6.5 Å. Likewise, the time correlation function drops styeeply, indicating that this distance between these two residues quickly become uncorrelated with the starting structure. Likewise, the radius of gyration fluctuates around 21.5 and 21.6 Å.

References

[1] http://www.charmmtutorial.org/static/files/crambin_example.tar.gz
[2] http://www.charmmtutorial.org/static/files/crambin_example.zip
[3] http://www.charmmtutorial.org/static/files/charmming_parser.tar.gz
[4] http://www.charmmtutorial.org/images/c/c2/1cbn-a.pdb
[5] http://www.charmm.org/ubbthreads-7-5-5/ubbthreads.php?ubb=showflat&Number=4116#Post4116
[6] http://www.charmmtutorial.org/images/7/75/Setup.inp
[7] http://www.charmmtutorial.org/images/2/25/Append.inp
[8] http://www.charmmtutorial.org/images/5/5f/Initial-minimization.inp
[9] http://www.charmmtutorial.org/images/1/1b/Initial-minimization.out
[10] http://www.charmmtutorial.org/images/1/1b/Solvate.inp
[11] http://www.charmmtutorial.org/5/5b/Solvate.out
[12] http://www.charmmtutorial.org/images/5/5c/Neutralize.inp
[13] http://www.charmmtutorial.org/images/d/d8/Neutralize.out
[14] http://www.charmmtutorial.org/images/d/da/Addions.str
[15] http://www.charmmtutorial.org/images/a/a9/Final-minimization.inp
[16] http://www.charmmtutorial.org/images/4/4f/Final-minimization.out
[17] http://www.charmmtutorial.org/images/9/9c/Md-heat.inp
[18] http://www.charmmtutorial.org/images/a/af/Md-heat.out
[19] http://www.charmmtutorial.org/images/a/a4/Md-equil.inp
[20] http://www.charmmtutorial.org/images/0/0f/Md-equil.out.gz
[21] http://www.charmming.org/images/6/68/Md-analysis.inp

[22] http://www.charmming.org/images/c/c6/Md-analysis.out

Installation

Installation of CHARMM

CHARMM runs on a variety of UNIX-like systems, although most installations currently are being done on Linux. With a little bit of work, the program can be installed on Windows machines via the CYGWIN emulation layer.

There are many options for installing CHARMM, many of which the beginning user need not touch. It is highly recommended that the user read install.doc [1] and the installation board on the CHARMM forums [28] before trying to install CHARMM.

CHARMM must be installed through a command line. The general format is:

```
./install.com <host machine type> [ <CHARMM_size> <install switches> ]
```

It is recommended that you type "./install.com" with no arguments to see all of the possible machine types, sizes, and install switches. It is usually a good idea to build CHARMM at the large or xlarge sizes, although very large systems may require the "huge" size. Be careful with huge, though, as several people have reported problems with it. If building a huge version you should run your binary on the test cases (found in the test subdirectory of the CHARMM source) and carefully compare the results with known good runs (sets of known-good results may be found on charmm.org [2]). This is a good idea with any new binary you build! Note that very small numeric differences in test case output are to be expected, but otherwise results should agree. If compiling CHARMM to perform QM/MM calculations in conjunction with a diufferent QM package, consult the output of "./install.com" to determine the correct flag to use (e.g. QC for Q-CHEM). Additional flags must also be given to compile parallel versions using MPI; again consulting "./install.com" is necessary.

Note: Compiling with the gcc/gfortran 4.0 as your compiler may result in installation problems. It is recommended that gcc 4.1.1 or later be used. Currently gcc version 4.2 or greater is recommended due to trajectory file compatibility issues with gcc 4.1. If gcc 4.1 must be used, a workaround is available (consult the Installation board mentioned above for details). CHARMM is being updated to Fortran 95, so versions c36 and later require gcc version 4.3 or later. Relatively recent versions of the Intel, Pathscale, and PGI compilers may also be used.

References

[1] http://www.charmm.org/documentation/current/install.html
[2] http://www.charmm.org/html/develop/testcases.html

CHARMMing's PDB Reader

CHARMMing's PDB reader exists to take a raw PDB file from the Protein Data Bank and convert it into a CHARMM-formatted PDB file. CHARMM has its own syntactic expectations for PDB files, so many PDB files will have problems when loaded into CHARMM (some of these problems are quite minor while others are potentially serious). This section of the tutorial is an explanation of how CHARMMing's PDB reader modifies a file from the PDB. It is not meant to be an exhaustive exploration of the PDB file format.

The PDB reader program itself is a standalone Python script (pdbinfo/parser_v3.py in the CHARMMing tarball, required libraries which are included in the tarball are lib/Atom.py and lib/Etc.py -- these are general purpose structure management libraries that we developed along with the parser).

Basic algorithm

The basic algorithm used by CHARMMing is:

1. Remove all lines that do not start with ATOM or HETATM
2. Handle multi-model atoms. Some PDBs separate different models cleanly into different sections, but others have alternate configurations listed for individual atoms or residues within a single model. These are identifieded via the following procedure:
    1. The line is checked to see if it belongs to an alternate model (this is done by looking at the PDB B-Factor, which is usually < 1 when multiple models are present and by checking to see if an 'A' has been prepended to the residue type). If the line is not part of a multi-model description, the algorithm skips to the last step.
    2. The line is stored in a two state (empty or full) buffer. If the buffer is empty, store the current line in it and move onto the next line (the buffer is flushed, printing the line, if a corresponding multi-model line is not found).
    3. If the buffer is full, compare the current line to the one in the buffer to see that they are multiple configurations of the same atom. If not, flush the buffer and store the current line into it (it is starting a multi-model section for a new atom).
    4. If the current line matches with the line in the buffer, look at the B-factor weight. Whichever line has a lower B-factor (less likely to be the correct structural data) is tagged for removal (it will not be written out when the multimodel buffer is flushed), and go on to the next line
    5. If the buffer is full and the line we are working on does not refer to the atom in the multimodel buffer (if said buffer is full), flush the buffer (i.e. collect the lines in it excepting those tagged for removal).
3. Sort the lines that have been read in by segment ID, residue number, and atom number, in that order (this groups all segments together, orders residues within segments, and orders atoms within residues)
4. Group the segments together (i.e. make a separate data structure for each independent segment) and decide whether each segment is a protein, nucleic acid, composed of "good hetero-atoms", or composed of "bad hetero-atoms" (accomplished by looking at the residues present within the segment)
5. Rename the terminal oxygen atom type in each protein segment from OXT to OT2 (most PDB files use OXT for the terminal Oxygen, but CHARMM expects OT2).
6. Decide if each nucleic acid segment is DNA or RNA (by testing for the presence of Uracil or Thymine residues)
7. Change the residue names and atom types to fit in with CHARMM's naming conventions
    1. HOH residues are renamed TIP3
    2. Adenine, Thymine, Cytosine, Uracil, and Guanine residues are given the correct names
    3. The CD1 atom in residue ILE (isoleucine) is given the atom type CD
    4. ZN, NA, CS, CL, CA, and K residues are renamed ZN2, SOD, CES, CLA, CAL, and POT respectively

8. Re-index atom number and residue IDs, to make atom and residue numbering continuous starting from 1. The mapping from original to new atom and residue numbers is preserved. Soon CHARMMing will allow the user to download PDBs with either the original or new atom numbering.
9. Write all segments out in separate PDB files.

Comments

- The PDB parser removes all lines other than ATOM and HETATM ones. A separate routine is called beforehand to parse out title, author, journal, and disulfide bond information.
- PDBs with multiple MODEL segments are passed to a separate routine to be split into their constituent models.
- We can tell if two atoms are the same by looking at their segment IDs, residue IDs, and atom numbers.
- Nothing is written out until the very end of the processing (last step of the algorithm above). PDB lines are kept in internal data buffers until then. A Python pickle file is written out of the data structure that has a snapshot of the internal data structures (especially those needed to restore the original atom and residue numbering).

Installation of CHARMMing

CHARMMing (www.charmming.org [2]) is an open source web-based front end to CHARMM. It can be used to prepare and run a variety of common tasks such as minimization, solvation, and molecular dynamics. In fact, the scripts used in this tutorial are based off of those created for CHARMMing.

You may download the CHARMMing source code from its Google code repository [1]. If you'd like to live on the somewhat-bleeding edge, A subversion repository is available at http://charmming.googlecode.com/svn.

Anyone is welcome to set up their own CHARMMing server, however a few things are needed:

1. A computer capable of acting as the server with Linux installed (CHARMMing has been tested on CentOS and Ubuntu, but it should work on most any Linux distro). The Apache Web server and mod_python should be configured and working. In theory, any Unix or Unix-like operating system, including Max OS X, that supports Django should be able to run CHARMMing. However, it has been developed and tested exclusively on Linux.
2. MySQL and the Python MySQLdb module should be working.
3. Django version 0.96 installed (version 1.0 and later will NOT work -- we are in the process of making CHARMMing compatible with these versions of Django) -- **Note:** the installer program can fetch and install Django for you. Most recent Linux distributions (e.g. Ubuntu 9.10) include newer versions of Django in their repositories. Therefore, you should install version 0.96 yourself or let the installer do it for you.
4. A CHARMM executable (version c35 preferred). We do NOT distribute CHARMM with CHARMMing; it must be properly licensed through Harvard University.
5. If QM/MM support is desired, a Q-Chem executable must also be present.
6. Knowledge of the Torque batch scheduler is helpful, but not required.
7. Light to moderate Linux system administration experience is also very helpful.

You also need to decide where to store user data (structures users upload and create) and private CHARMMing data (the CHARMM executable, topology files, etc.). We'll refer to these two locations as $USERDIR and $DATADIR, respectively.

The installation of CHARMMing is done via an automated install script. This page will describe what the install script is doing. If you have questions or problems with the install, please post the on the CHARMMing board of the CHARMM forums [28].

Running the installer

This section assumes that you have downloaded a complete CHARMMing release tarball. For instructions on other install methods (e.g. installing from the Subversion repository), please see the INSTALLER documentation file distributed with CHARMMing. To get the install process started, extract the tarball and change into the newly extracted directory. Then run the installer.py script **as root**. We are looking at ways to make installation work as a non superuser, but right now root privileges are required. The installer will run some startup checks, and then present you with a prompt:

```
>
```

The prompt allows you to install any of the components of CHARMMing individually. However, for a new install you can just run through all the components in order which you can do by simply typing "install" (no quotes) at the prompt. The following sections will go through each of the install routines:

Installing Django

Django itself can be downloaded from the Django project web site [2]. Remember to get **only** version 0.96. You are welcome to download and install it yourself, for example if you need to install it to a non-standard location. However, CHARMMing's installer can download and install Django for you. If you would like it to do so, answer "y" when prompted. If you already have Django installed, feel free to say "n" here to proceed to the next step. If you choose "y", the script will download Django 0.96 to /tmp and install it to the default location (generally /usr/lib/python2.X/site-packages). As stated above, Django 0.96 is required to make CHARMMing operate; work is in progress to make CHARMMing compatible with newer Django versions.

Installing third-party software

CHARMMing requires the freely-available OpenBabel [3] software package, however it is not distributed directly with CHARMMing. The third-party software install routine will download and install it into /usr/local for you. Feel free to skip this step (answer "n" when prompted) if OpenBabel is already installed on your system. Some Linux distributions (Ubuntu and Debian, at least) have OpenBabel in their package repositories. These versions should be safe to use. More third-party software may be added as CHARMMing develops.

Installing the CHARMMing files

CHARMMing itself needs to be installed in a directory that your web server (Apache) is configured to serve up. The default install path is /var/www/html, which is the default web root on RHEL and Fedora based systems. On Debian based systems the web root is usually /var/www and other distributions use paths like /srv/httpd, so be sure to check your own system before proceeding. There is limited functionality to detect which operating system you are using, but it generally works best to just specify the installation path that you want the installer to use. As part of the install process, this routine also sets up the database that CHARMMing will use. You will be prompted to enter the MySQL user name, password, and database to use. If these are not configured, the installer will ask you for your MySQL root password and will create the database and charmming database user for you. Additionally, you have the opportunity to create a Django administrative account for your CHARMMing setup, which is required to administer your installation. These steps are described in further detail below:

Extracting the tarball

The first step of this process is the extraction of the charmming.tar.gz tarball in your chosen Web directory. This creates a subdirectory called charmming in this directory, which is the root of the CHARMMing install. CHARMMing assumes that the charmming directory is in the Web root directory so that all CHARMMing URLs look like:

http://www.mysite.org/charmming/<blah>

If this is not the case, you will need to edit charmming/urls.py to change the URLs to match up with the ones used on your server. If CHARMMing is the only site running on your server, it is suggested that you redirect www.mysite.com to www.mysite.com/charmming.

Configuring the database

Once the tarball has been extracted, the routine attempts to set up the database, prompting you for the correct MySQL credentials and database. The database is brought on-line by running:

"python manage.py syncdb"

Note that the database credentials are stored in settings.py. Currently only databases running on localhost are supported by the install script, but you can edit settings.py if you want to run the database on another system. Various tables must be populated before CHARMMing is started; the required data is in the .sql files distributed with CHARMMing, so these are processed automatically. The correct credentials for database access are also automatically loaded into the scheduler/schedd.py script since it needs to access the database independently.

At this point, Django offers the user a chance to create an administrator account to log in to CHARMMing. This task **must be performed**, otherwise you will be locked out and it will be necessary to reinstall.

Set up $USERDIR and $DATADIR

$DATADIR

CHARMMing requires a place to store private data. This store is created once and does not grow as the site is used (unless you add to it!). It is primarily used to hold topology, parameter, and other data files, water box coordinates, and the CHARMM binary itself. Web users will not have access to this area (although they are allowed to download topologies, parameters, and the water box coordinates). You are able to choose what directory is used for this. To get this area set up routine simply extracts the charmming-private.tar.gz tarball into the chosen directory and renames the created "charmming-private" directory into the "charmming" directory.

Please note that the CHARMM binary is **NOT** distributed with CHARMMing. We will explain how to install it into $DATADIR later on.

$USERDIR

Next, the installer will configure the user area and the area where CHARMMing keeps its own files. Each user has their own subdirectory of this main data directory into which the files that they generate are placed. Like with $DATADIR, you can choose the location of $USERDIR Consequently, the space requirements of this directory may grow substantially over time.

There is no tarball to extract to create this directory; the installer creates a data directory, along with an admin subdirectory to store files generated by the administrative user (who gets set up in section 1.3.

Install visualization Package

CHARMMing distributes the GPL licensed Jmol [4] software to use for visualization; however, an interface to ChemAxon's MarvinSpace has also been developed. This interface may be enabled in this installation routine. However, you will have to obtain the MarvinSpace software yourself, as it requires a click-through license agreement. The Jmol software is simply copied from the CHARMMing software. You should be able to use a newer version than the one we distribute.

Install and configure the job scheduler

CHARMMing is designed to act as a front end to a larger computing resource, and as such runs jobs via an external job scheduler. A daemon (schedd) exists to arbitrate between CHARMMing and the target batch queuing system. Currently, CHARMMing integrates with both the TORQUE [5] and Condor [6] batch schedulers. Of these, TORQUE is more thoroughly tested and should be used unless you have some reason not to. There are thus two parts two the job scheduling system:

1. A daemon distributed with CHARMMing (schedd) that interfaces between the web application and the batch queue system.
2. The batch queue system itself which is an external program (currently PBS/TORQUE and Condor are supported)

Configuring each of these two is described here:

Configuration of schedd

The schedd daemon is very simple and only requires two pieces of information:

- what user account it should run under
- what batch queue system it will be interfacing with

The install routine will prompt for this information. Even if you already have a batch scheduler running you have to tell schedd which one it is and where it is installed (you don't actually have to have the installer download and install the batch scheduler for you though -- you can skip that part of the install routine).

You should create a user for schedd to run as. The user account specified should be a member of the apache group (this group might be named differently on different systems; it's the group that the Apache web server runs as) so it can write files into the Apache-created user directories. **UNDER NO CIRCUMSTANCES SHOULD SCHEDD RUN AS ROOT OR ANY USER THAT CAN SU/SUDO TO ROOT!**

Installation/configuration of the chosen batch scheduler

Currently, because CONDOR is distributed primary in binary form, automatic installation of it is not supported. You can download and install it directly from the Wisconsin site. Please make sure that the schedd user has condor_q, condor_submit, and condor_submit_dag in its $PATH.

If you choose to install PBS, a recent version of TORQUE (a free PBS-compliant scheduler) will be downloaded and installed into /usr/local. Please make sure that /usr/local/bin is in the $PATH of the schedd user. Although TORQUE is fully installed, it requires further configuration. Details on how to do this are given in section III.

Configuring Apache

The installer can attempt to configure your Apache set-up to use mod_python for CHARMMing. Note that the configuration method is very naive, and probably will not work or actively break more complex set-ups. Therefore this routine gives you an option to skip it. When in doubt (i.e. working on a production box), it is recommended that you skip this step. Nonetheless, the configurator seems to work OK on stock Fedora/CentOS httpd.conf files. It has not yet been tested on other systems.

If you elect to proceed with this routine, you will be asked the location of your httpd.conf file. The appropriate configuration for CHARMMing will be appended to the end of this file.

Please see the next section for information on how to configure Apache by hand.

Post-install configuration

Manual Apache configuration

It is necessary to configure Apache to use mod_python in order to run the Python files that power the Web site. Since CHARMMing is contained entirely in one directory you can create a directory specification that tells Apache to use mod_python in that directory. Here is an example of such a specification assuming that CHARMMing is installed in /var/www/html/charmming (adjust as necessary for your system):

```
<Directory /var/www/html/charmming>
  SetHandler python-program
  PythonHandler django.core.handlers.modpython
  SetEnv DJANGO_SETTINGS_MODULE settings
  PythonDebug On
  PythonPath  "['/var/www/html/charmming'] + sys.path"
</Directory>
```

Once this section has been added to the Apache configuration, it is necessary to restart the server. On Red Ht/Fedora systems, this is usually done with:

```
/etc/init.d/httpd restart
```

and on Debian based systems (including Ubuntu) with:

```
/etc/init.d/apache2 restart
```

however, startup scripts vary from distribution to distribution. Consult your documentation if you need help.

Installing CHARMM/Q-Chem

CHARMMing expects an XXLARGE CHARMM serial binary to be placed in its private directory. It expects the file to be named gfortran-xxlg.one (even if some non-gfortran compiler is used) although this can be changed by modifying the source. The binary can be either 32 or 64 bits. It can be created with the following install.com commands (assuming gfortran is used):

```
./install.com gnu xxlarge GFORTRAN # 32 bit mode
./install.com gnu xxlarge GFORTRAN X86_64  # 64 bit mode
```

To compile with Q-Chem support add the QC key word to the above commands. Note that you need the Q-Chem software itself in order to use CHARMM's Q-Chem interface.

Once install.com completes successfully copy exec/gnu/charmm into the private data directory with the file name gfortran-xxlg.one, e.g.:

```
 cp exec/gnu/charmm $DATADIR/gfortran-xxlg.one
```

If you would like to use Q-Chem, you will also need to put your qchem directory (containing bin/, exe/, and samples/) and the QCAUX directory into $DATADIR. If you do not have Q-Chem and wish to disable the QM/MM portion of the user interface, you can edit charmming_config.py in your CHARMMing install directory (set haveqchem = 0).

Start scheduling

It is necessary to start the two parts of the scheduling system before you can use CHARMMing. First you must start the batch scheduler that you chose (TORQUE or CONDOR) and then you need to start CHARMMing's own scheduler daemon (schedd), which mediates between CHARMMing itself and the batch scheduler. Since TORQUE is the default batch scheduler, its configuration is described below.

Starting TORQUE

If you installed TORQUE (PBS) from scratch, it is still necessary to configure it. In the extracted charmming-package tarball, there should be a subdirectory called pbs containing two files: (1) a sample node configuration file called node_config and (2) a script that can be passed to TORQUE's qmgr program to configure the pbs_server.

Here are the exact steps that you need to perform:

1. Copy the node_config file to /var/spool/torque/mom_priv/config (cp pbs/node_config /var/spool/torque/mom_priv/config).
2. Edit /var/spool/torque/mom_priv/config and adjust the $ideal_load and $max_load variables to something suitable for your system (note that the defaults are for a 4-core system).
3. In the same file, modify the $clienthost variable to be the host name of your server.
4. In the same file provide an appropriate $usecp line -- which determines whether or not to use rcp/scp to copy files remotely. For set-ups running entirely on one machine "$usecp *:/ /" should be OK. For more complex configurations, consult the TORQUE manual.
5. Start the TORQUE server, scheduler, and mom daemons with:

```
pbs_server -t create
pbs_sched
pbs_mom
```

You will probably want to have a script to start these on boot. Assuming you have the installer script download TORQUE for you, you can find init scripts in /tmp/torque-2.2.1/contrib/init.d. Note, when starting the server after the first time, be sure to use pbs_server -t hot rather than pbs_server -t create.

Once the server, scheduler, and mom are up and running, you can configure the server to reasonable defaults by running qmgr < conf_pbs (where conf_pbs is in the pbs directory of the extracting charmming-package).

Once this is done, the final step is to configure the scheduler so it sees the local host as a worker node. This can be done with the following commands:

```
qmgr -c "create node <your.host.name>"
qmgr -c "set node np = <# of processors/cores in your node>"
qmgr -c "set node properties = exec"
```

The last line isn't strictly necessary, but does help describe your node.

Starting CHARMMing's job scheduler

Once the batch scheduler is started, it is necessary to start CHARMMing's own scheduler daemon (schedd). Before doing so, make sure that the TORQUE or CONDOR commands are in the schedd user's $PATH. TORQUE requires qsub and qstat while CONDOR requires condor_q, condor_submit, and condor_submit_dag. To actually start schedd, su to the user that schedd is supposed to run as, and change to <CHARMMing install path>/scheduler. Then simply run ./schedd.py. You can look at the schedd.log for information about what schedd is doing, and it will also create either a pbs.log or condor.log detailing commands it has issued to the queuing system. Note that these files can grow quickly, so be sure to rotate them occasionally, especially on systems with limited disk space (the logrotate

utility can be used to accomplish this).

Once again, it will probably be desirable to create an initialization script that will start schedd automatically on system boot.

Initial login

Once everything is up and running, you should be able to go to the URL of your CHARMMing install; if not you may need to restart the apache server. Log in using the administrative account you created in step II.b, and begin running jobs. At this point, it is a good idea to create a non-administrative account. Fill out the registration form (which is linked from the front page) and submit it. If you entered your e-mail address correctly in step II.b and outgoing mail is set up properly, you should receive an e-mail stating that a new user has registered.

To approve this user, go to www.mysite.com/charmming/admin (replace /charmming/ with the actual URL path of your install) and log in using your administrative user name and password. In the auth pane, click on the users link and select the user name of the new user. In the groups display at the bottom of the screen, click on "students", which will unselect "preapprove." Then click on "Save" (bottom right of the screen). You can also give the new user staff status (which allows them to log in to the administrative interface) or superuser status (which allows them to do anything they want in the administrative interface).

Log out as your administrative user and test that the newly-created user can now log in and submit jobs.

Everything should now be ready.

References

[1] http://charmming.googlecode.com
[2] http://www.djangoproject.com
[3] http://www.openbabel.org
[4] http://jmol.sourceforge.net
[5] http://www.supercluster.org/torque
[6] http://www.cs.wisc.edu/condor

Article Sources and Contributors

Introduction *Source*: http://www.charmmtutorial.org/index.php?oldid=764 *Contributors*: Hlwoodcock, Tim

Basic CHARMM Scripting *Source*: http://www.charmmtutorial.org/index.php?oldid=755 *Contributors*: Tim

Essential CHARMM Features *Source*: http://www.charmmtutorial.org/index.php?oldid=757 *Contributors*: Tim

The Energy Function *Source*: http://www.charmmtutorial.org/index.php?oldid=766 *Contributors*: Sboresch, Tim

Minimization *Source*: http://www.charmmtutorial.org/index.php?oldid=793 *Contributors*: Sboresch, Tim

Solvation *Source*: http://www.charmmtutorial.org/index.php?oldid=770 *Contributors*: Tim

Neutralization *Source*: http://www.charmmtutorial.org/index.php?oldid=774 *Contributors*: Tim

MD *Source*: http://www.charmmtutorial.org/index.php?oldid=772 *Contributors*: Tim

LD *Source*: http://www.charmmtutorial.org/index.php?oldid=778 *Contributors*: Tim

Analysis *Source*: http://www.charmmtutorial.org/index.php?oldid=780 *Contributors*: Sboresch, Tim

Full example *Source*: http://www.charmmtutorial.org/index.php?oldid=782 *Contributors*: Tim

Installation *Source*: http://www.charmmtutorial.org/index.php?oldid=751 *Contributors*: Tim

CHARMMing's PDB Reader *Source*: http://www.charmmtutorial.org/index.php?oldid=689 *Contributors*: Tim

Installation of CHARMMing *Source*: http://www.charmmtutorial.org/index.php?oldid=700 *Contributors*: Tim

Image Sources, Licenses and Contributors

Image:Modeling_Approaches_White.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Modeling_Approaches_White.png *License*: unknown *Contributors*: -

Image:graph-cutoff.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Graph-cutoff.png *License*: unknown *Contributors*: -

Image:bullseye-cutoff.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Bullseye-cutoff.png *License*: unknown *Contributors*: -

Image:Solvated-system2.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Solvated-system2.png *License*: unknown *Contributors*: -

Image:neut-output.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Neut-output.png *License*: unknown *Contributors*: -

Image:dist.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Dist.png *License*: unknown *Contributors*: -

Image:dcor.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Dcor.png *License*: unknown *Contributors*: -

Image:rgyr.png *Source*: http://www.charmmtutorial.org/index.php?title=Image:Rgyr.png *License*: unknown *Contributors*: -