Evan N. Feinberg

Software Specification and/or Tutorial for Lilo: Ligand Parameterization Utility

Note 2: Everything in Step 5 forward will be automated by a program hopefully before the end of the Summer '15 quarter. Subsequently, everything in step 1 forward will be automated by a program, but this is slightly lower priority and will take a bit longer.

Higher Level

The goal of ligand parameterization is to most accurately reflect the bond, angle, and dihedral parameters of your molecule. While the CHARMM force field was originally optimized for biomolecules (e.g., protein and DNA), a generalized force field, called CGenFF, was developed to use CHARMM to accurately model small molecule ligands as well. A software tool called ParamChem automatically fragments uploaded molecules and then assigns CGenFF parameters by analogy. However, ParamChem is imperfect. It often fragments molecules in suboptimal ways, especially when encountering ligands with bridges or other complex topologies like fused rings. Acknowledging this, ParamChem kindly assigns "penalty scores" to roughly estimate how unsure it is of the parameters it assigns. Penalties ≤10 indicate good agreement with known parameters. Penalties where 10≤p≤50 are acceptable, and those where p≥50 have a relatively high probability of being inaccurate. The goal of this tutorial is to attempt two procedures – manual ligand fragmentation, and dihedral fitting with QM methods – to obtain FF parameters as accurately as possible.

Step 1: Prepare and Create a mol2 file of your ligand

Open Maestro. Make sure the following toolbars are selected, "Project", "Edit," "View," "Workspace," "display atoms," "Representation," "Build," "fragments."

If you already have a prepared ligand, import it into Maestro now.

[This section is beta-FOA specific]
As an example system (this will be generally applicable to any ligand), we will create the ligand beta-FOA starting with the M-OR and covalently bonded beta-FNA.

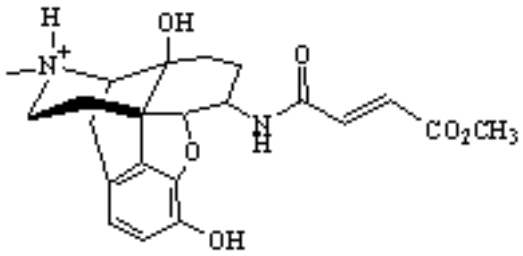


Figure 1: beta-FOA

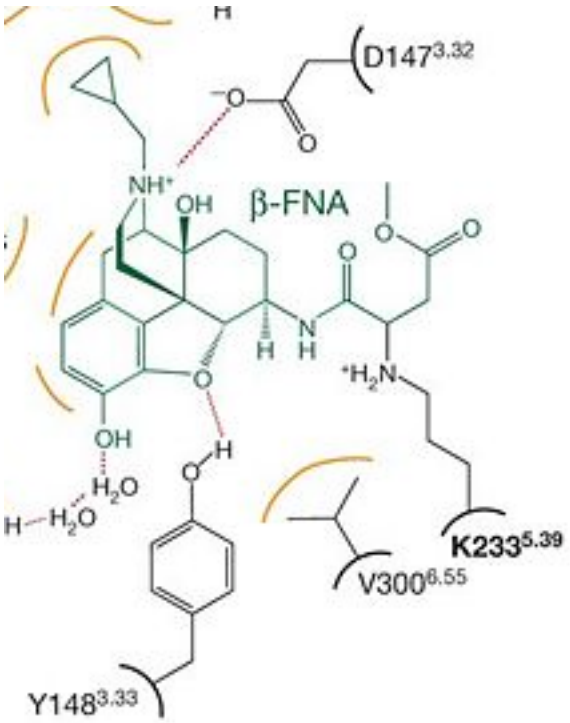


Figure 2: beta-FNA bonded to Lys233 of M-OR

Begin with the inactive mu Opioid Receptor, PDB ID 4DKL. Select the co-crystallized ligand. Right click and go to "invert selection". Delete all the atoms that are then selected. You should be left with only the co-crystallized ligand. The co-crystallized ligand is beta-FNA. You will manipulate beta-FNA to construct beta-FOA. Since beta-FOA is a noncovalent ligand, it is more straightforward to parameterize than beta-FNA. With beta-FNA, one would really have to parameterize a beta-FNA–Lys complex, which adds an order of complexity. So in this tutorial, you will mutate beta-FNA → beta-FOA.

First, select "increase bond order" icon on the far right of your screen. Now, find the atoms named "CAX" and "CAW." You will click on the bond between these two carbon atoms. This should create a double bond between them.

Second, you will delete the cyclopropyl group of the ligand. This is a highly strained three-carbon ring connected to a carbon atom that is in turn connected to the tertiary amine nitrogen of the ligand. Delete the three carbons named CBE CBF and CBG as well as the protons bonded to them.

Third, add a positive charge and a Hydrogen atom to the Nitrogen "NAJ" (close to the removed cyclopropyl group) to make it a "quat" (en.wikipedia.org/wiki/Quaternary_ammonium_cation).

[End beta-FNA specific section]

In many cases, your ligand might be in the format of a PDB file, which has certain deficiencies in the context of ligand preparation. Namely: PDB files only have coordinates, not connectivity (unless there are CONECT records, which are not universally handled) information. In addition, crystal structures will not come with hydrogens, and therefore you will have to add hydrogens to your ligand as well as choose individual protonation states.

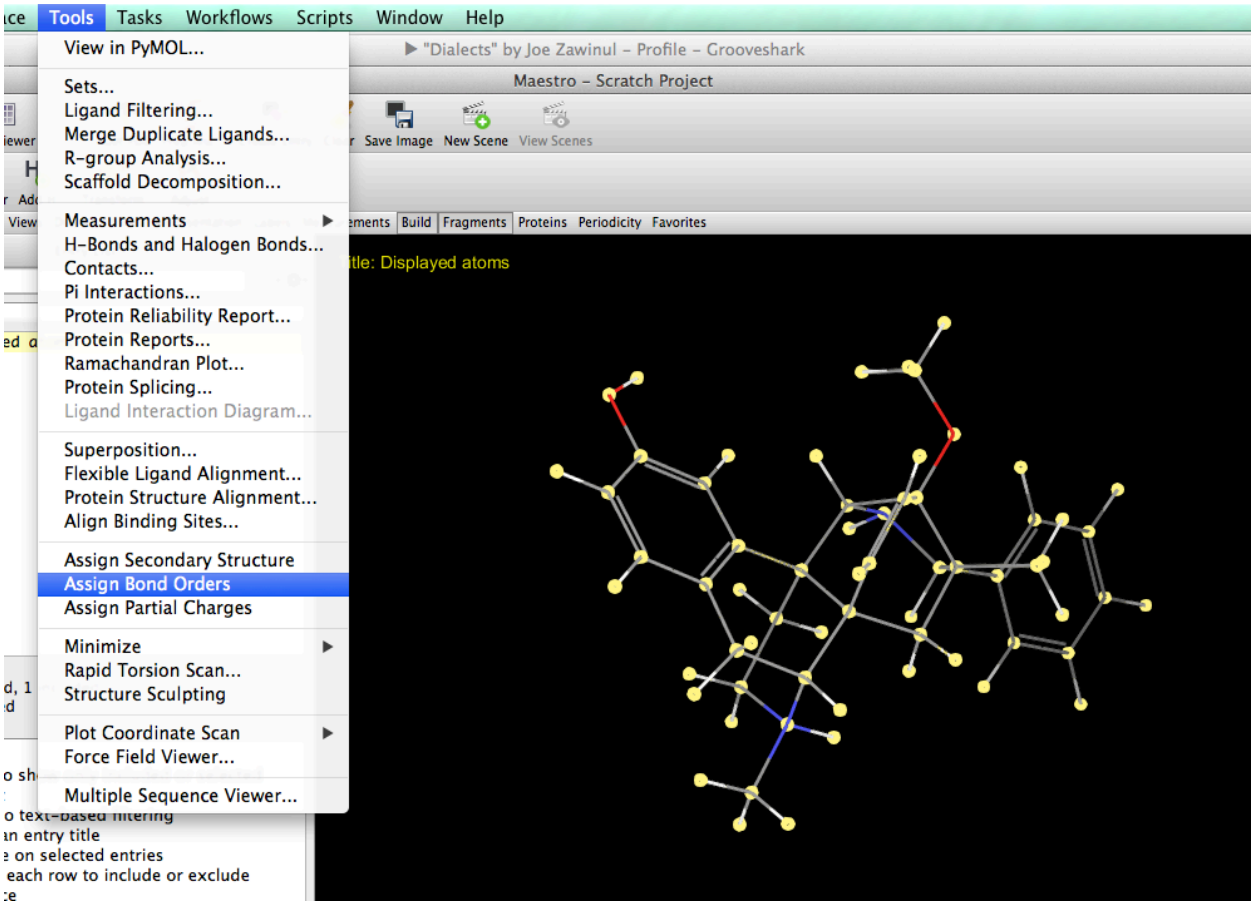
To prepare your ligand, load the structure of your ligand into Schrödinger Maestro, select your ligand, click "assign bond order" in the Tools menu. Now select "residue" from the "Add H" menu, and click on your ligand. Now all bond orders should be properly assigned and protons added.

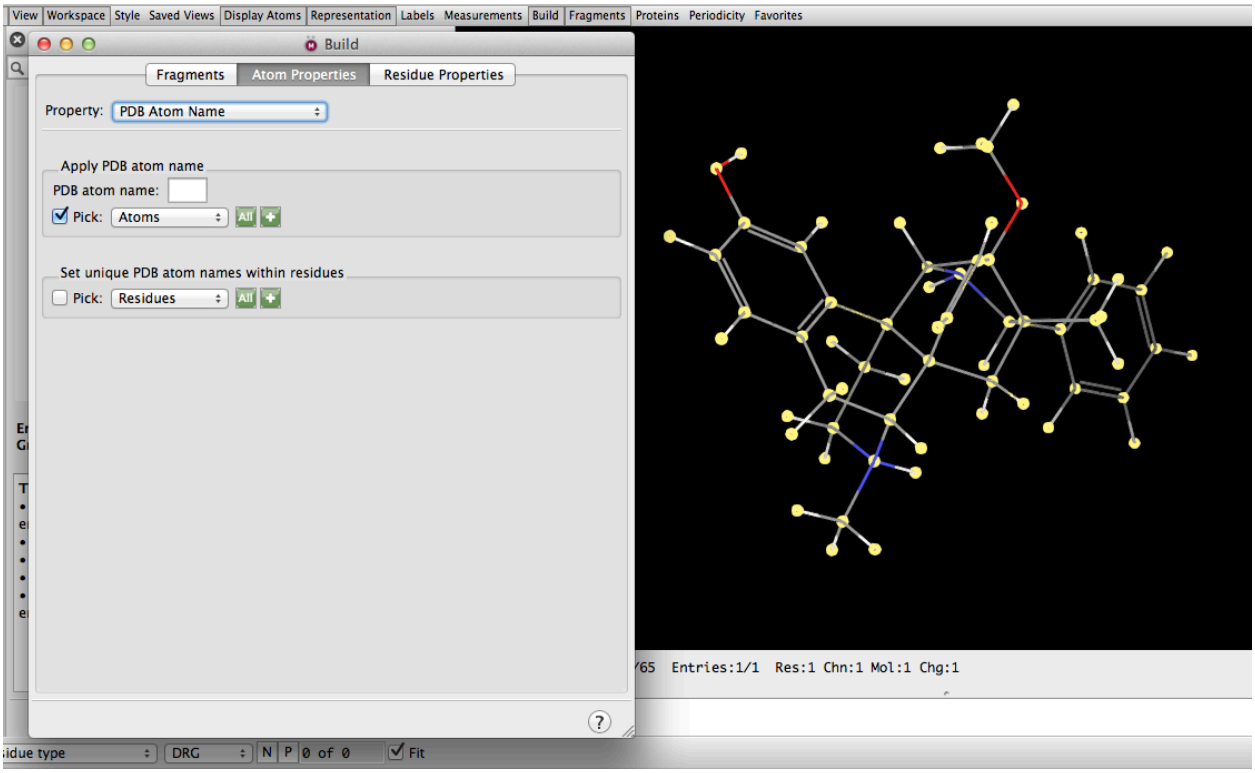
However, if you believe your ligand is charged *in-vivo* or in some assay, you might want to further protonate one of its heavy atoms. Such an example is described above in the tertiary amine nitrogen of beta-FNA and beta-FOA. In this example, the tertiary amine nitrogen is proximal to a negatively charged carboxylic acid on Asp147 in the mu Opioid Receptor. Since it has a high pKa, the tertiary amine nitrogen is protonated and thus becomes positively charged. The resulting ammonium cation forms a salt bridge with Asp147, stabilizing opiate drugs in the mu Opioid Receptor.

To add a proton, click on the "lightning bolt" (+) icon on the far right toolbar in Maestro. Now, click on the atom for which you would like to increase the formal charge. When you mouse over that atom, you should now see it has an increased formal charge in the readout on the far bottom of the screen. Now, add an additional hydrogen to that atom.

Do the following only if absolutely necessary:

If your atom names are not unique, you will want to rename them now, or else things can get pretty confusing later. If your atom names are not unique, ParamChem will automatically make them so, and it could be a major headache to keep track of which atoms are which. Avoid headaches and migraines at all costs, is a good life lesson. So go to Edit → Build → Atom Properties. Change "Property" to "PDB Atom Name." Now, select all atoms in your ligand. Under "Set unique PDB atom names within residues," click the "plus" sign, click Workspace Selection, and now your atom names should be unique. Note that Maestro doesn't confirm this action.


Tools
Tasks
Workflows
Scripts
Window
Help
View in PyMOL...
Sets...
Ligand Filtering...
Merge Duplicate Ligands...
R-group Analysis...
Scaffold Decomposition...
Measurements
H-Bonds and Halogen Bonds...
Contacts...
Pi Interactions...
Protein Reliability Report...
Protein Reports...
Ramachandran Plot...
Protein Splicing...
Ligand Interaction Diagram...
Superposition...
Flexible Ligand Alignment...
Protein Structure Alignment...
Align Binding Sites...
Assign Secondary Structure
Assign Bond Orders
Assign Partial Charges
Minimize
Rapid Torsion Scan...
Structure Sculpting
Plot Coordinate Scan
Force Field Viewer...
Multiple Sequence Viewer...
"Dialects" by Joe Zawinul – Profile – Grooveshark
Maestro – Scratch Project
Save Image
New Scene
View Scenes
Build
Fragments
Proteins
Periodicity
Favorites
Displayed atoms



Export your uniquely named ligand as a mol2 file with a name of choice. The default options should work.

Step 2: Submit mol2 file of ligand to paramchem

Go to paramchem website, and upload the mol2 file. There are a few mistakes that can give you error messages. Make sure: (1) all hydrogens are added to the molecule, (2) any formally charged atoms (e.g., quarternary nitrogen, i.e. ammonium, should be assigned a partial charge of +1 in Maestro) have been assigned as such in Maesto. Make sure all atoms have the proper number of bonds. For example, you should never see an sp3 carbon with only 3 bonds in your mol2 file structure. You should only in very rare circumstances have to check the options boxes.

Take the output from ParamChem if there were no errors, paste it in a TextWrangler or Sublime document, and save the output as [your_mol2_file_name].str. The "str" stands for "stream" file. It is effectively a concatenation of .rtf and .prm files, containing molecule topology, partial charges, and bond/angle/dihedral parameters all in one.

Step 3: Molecule fragmentation, if need be.

ParamChem will give you charge and parameter penalties at the top of the file. These are equal to the maximum partial charge and parameter penalty, respectively, for all the atoms and bonds. It will be located at the top of the .str file it outputs. If these are relatively high,

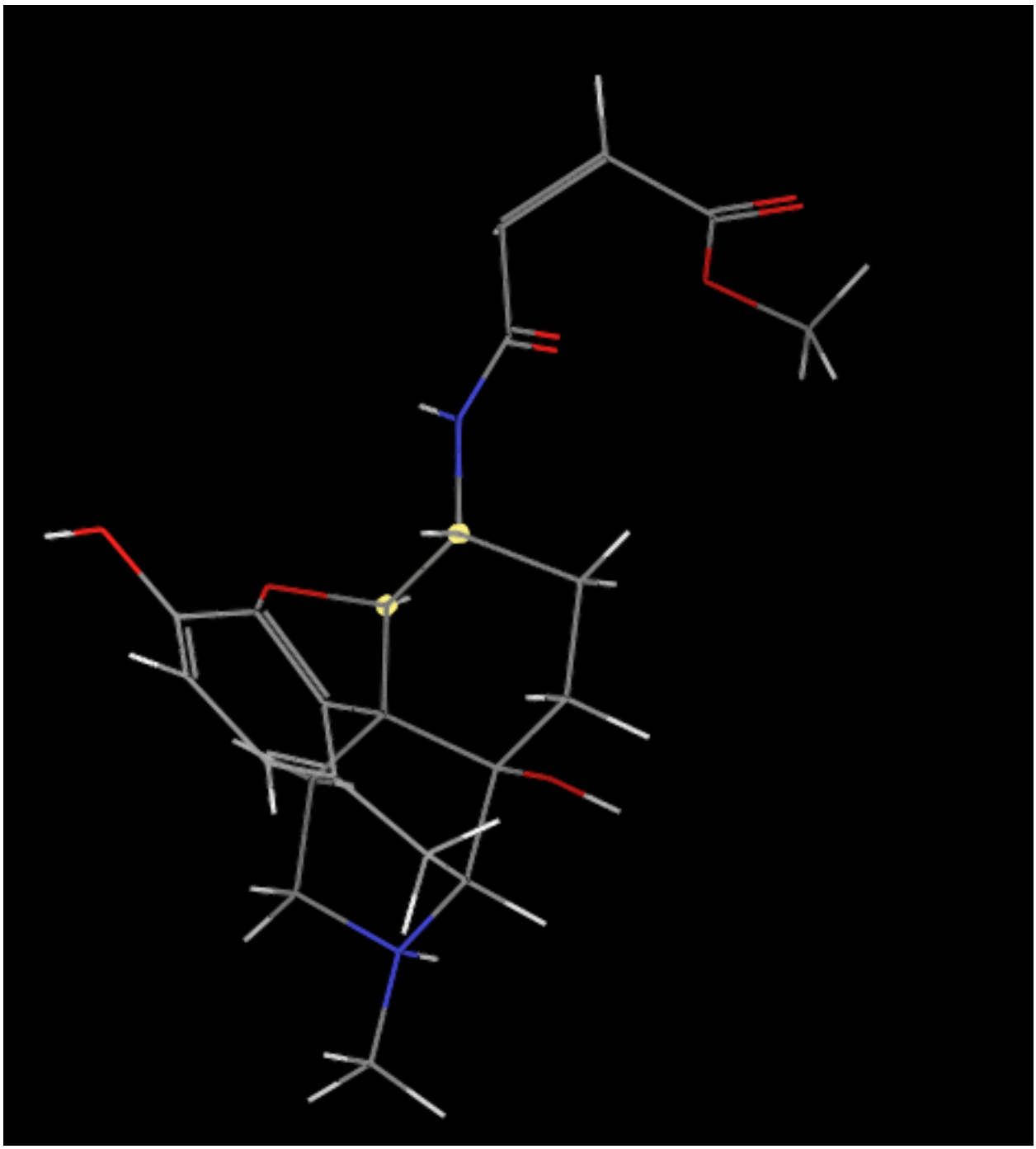
and certainly if they are ≥50, then it is advisable to manually fragment the ligand into smaller components that ParamChem can then more readily and accurately parameterize.

Here are some general rules to follow in fragmenting your ligand into smaller parts:

(1) Do not break up aromatic rings at all costs.
(2) Functional groups like carbonyl C=O should be at least one bond, ideally two, away from the carbon at which you fragment the molecule.
(3) Try to fragment the molecule at sp3 carbons. Ideally, these sp3 carbons should be bonded to only carbons and hydrogens if possible.
(4) Electron donating or withdrawing groups should be at least one, ideally two bonds away from the point of fragmentation.

It is not always possible to follow all these rules for a given molecule and also to fragment it sufficiently. Let's look at an example to illustrate.

Here is betaFOA:



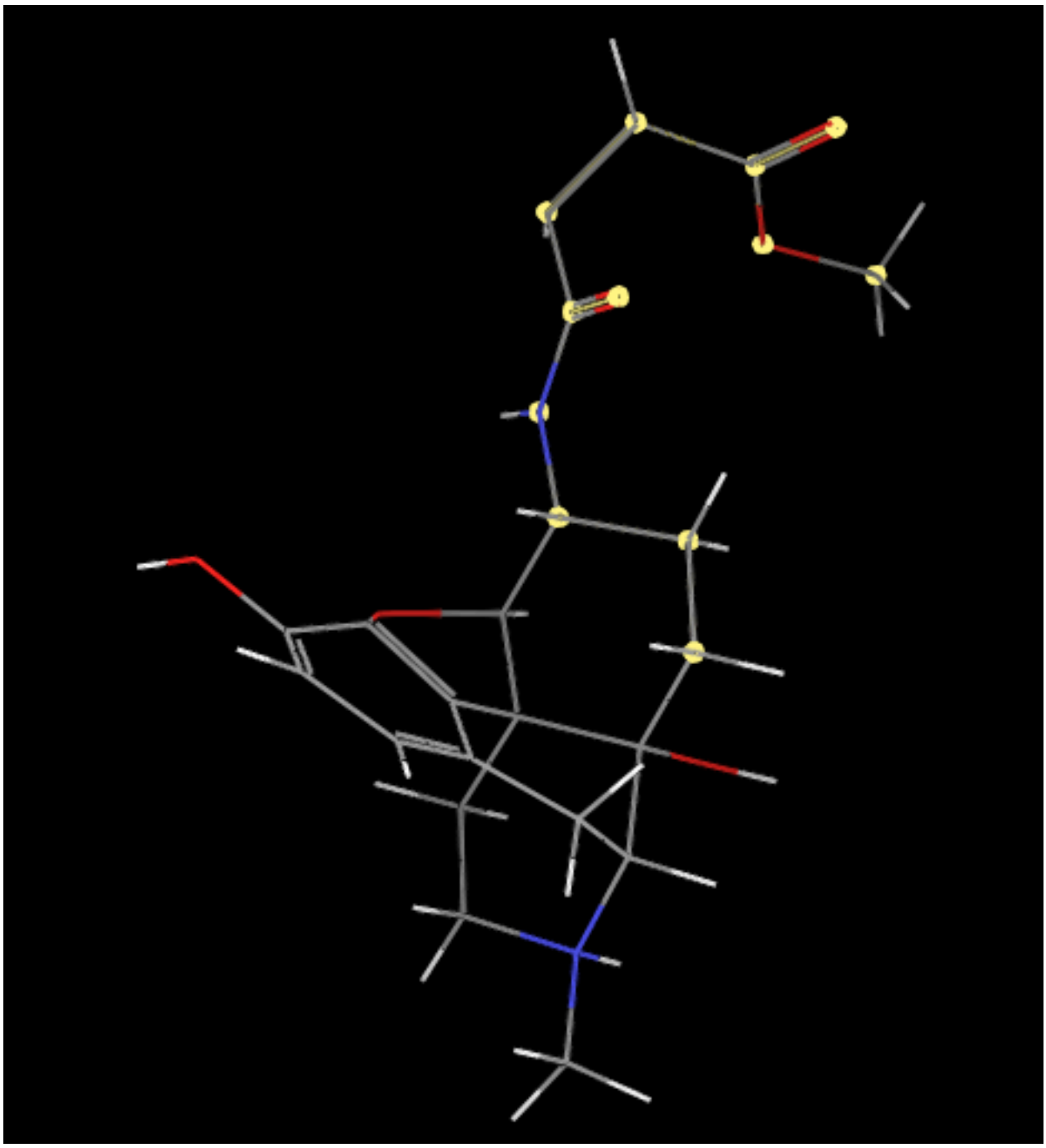
It is natural to split the upper branch of the molecule into its own fragment. However, there are two atoms (highlighted) about which this fragmentation can occur. Unfortunately, both are adjacent to an electronegative (either the ester ring oxygen or the secondary amine nitrogen). In this case, the carbon adjacent to the nitrogen is chosen as the atom about which the fragmentation is made.

To create a molecule fragment:

First, in the “Displayed Atoms” row on the “Entry List” column, rename it to “Original Molecule.” This will be used as the parent molecule. You will make a duplicate of it every time you make a new fragment.

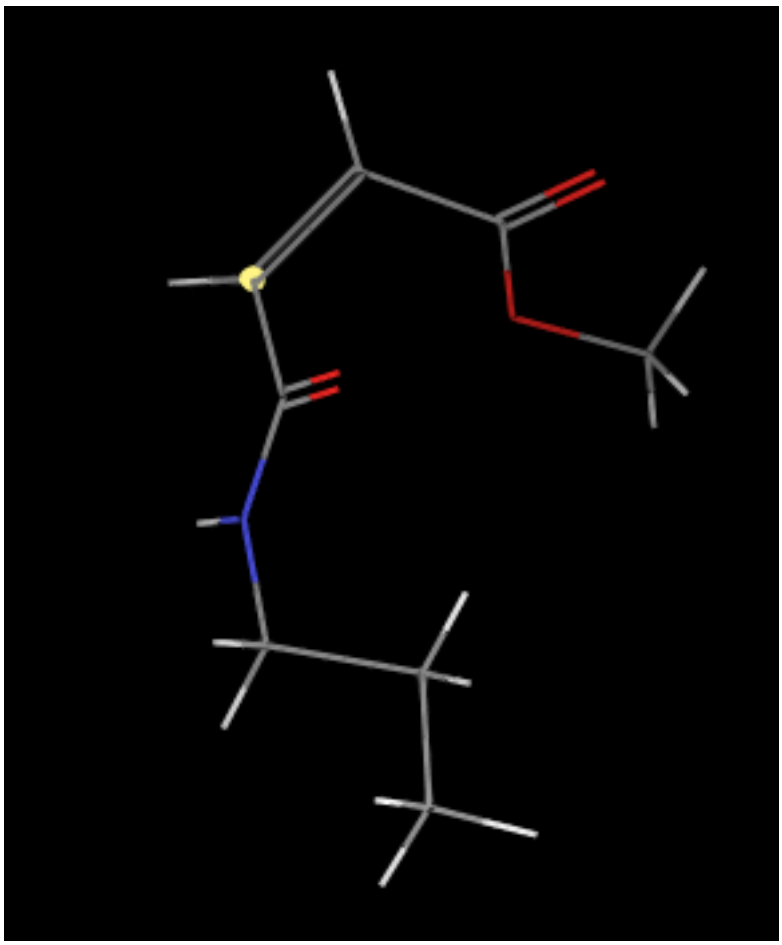
Now right click on it, and duplicate entry in place. Name the new entry, “Fragment 1.”

Now, select all non-hydrogen atoms in the upper fragment:



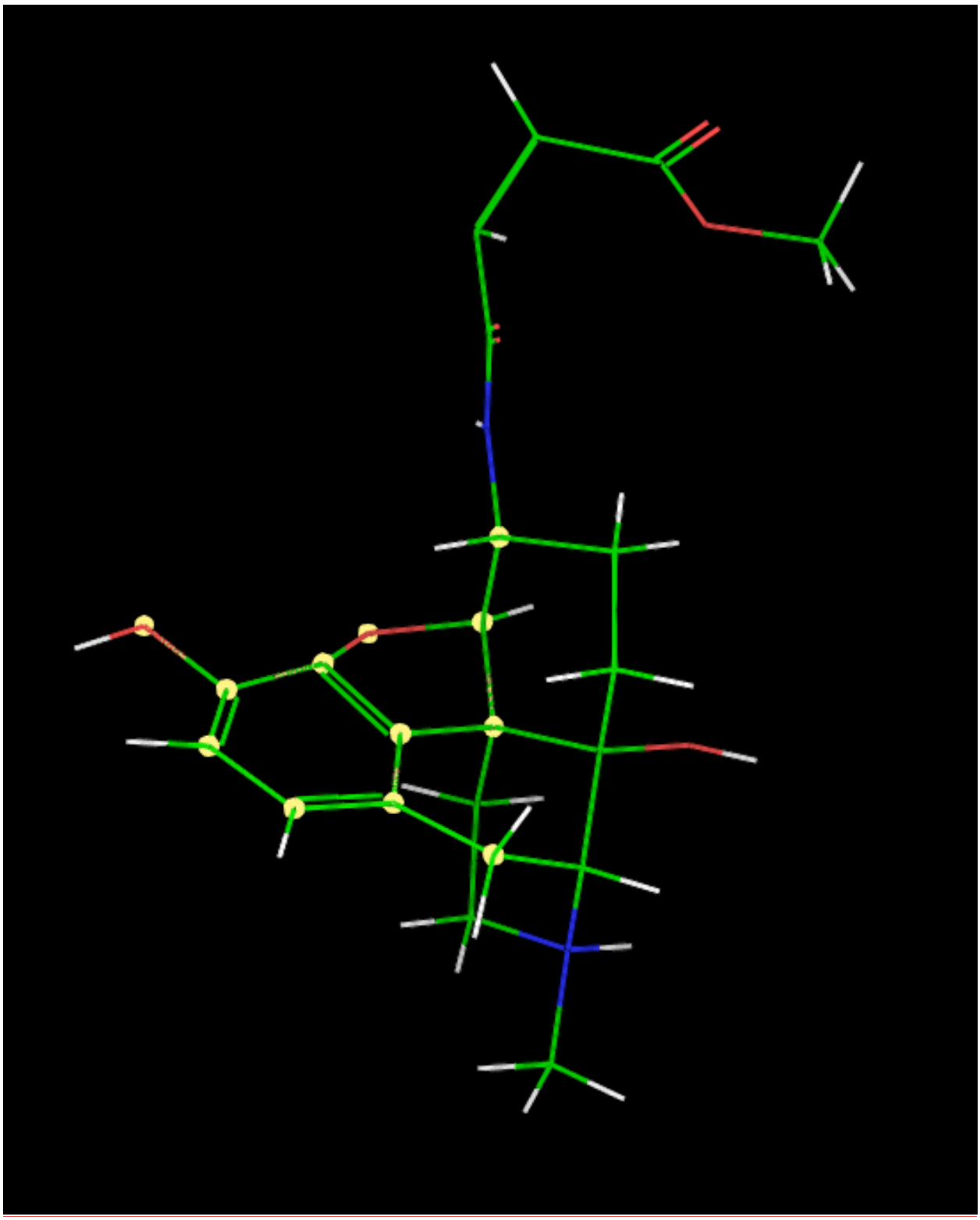
Now, right-click and hold, and choose "Invert Selection." Now right-click and hold, and select "Delete." Use the "Add H" button to add missing hydrogens to your new fragment (deleting and re-adding the hydrogen atoms should ensure that they are named the same as in the complete molecule). Now export the new molecule as a mol2 file.

The result should look like this:



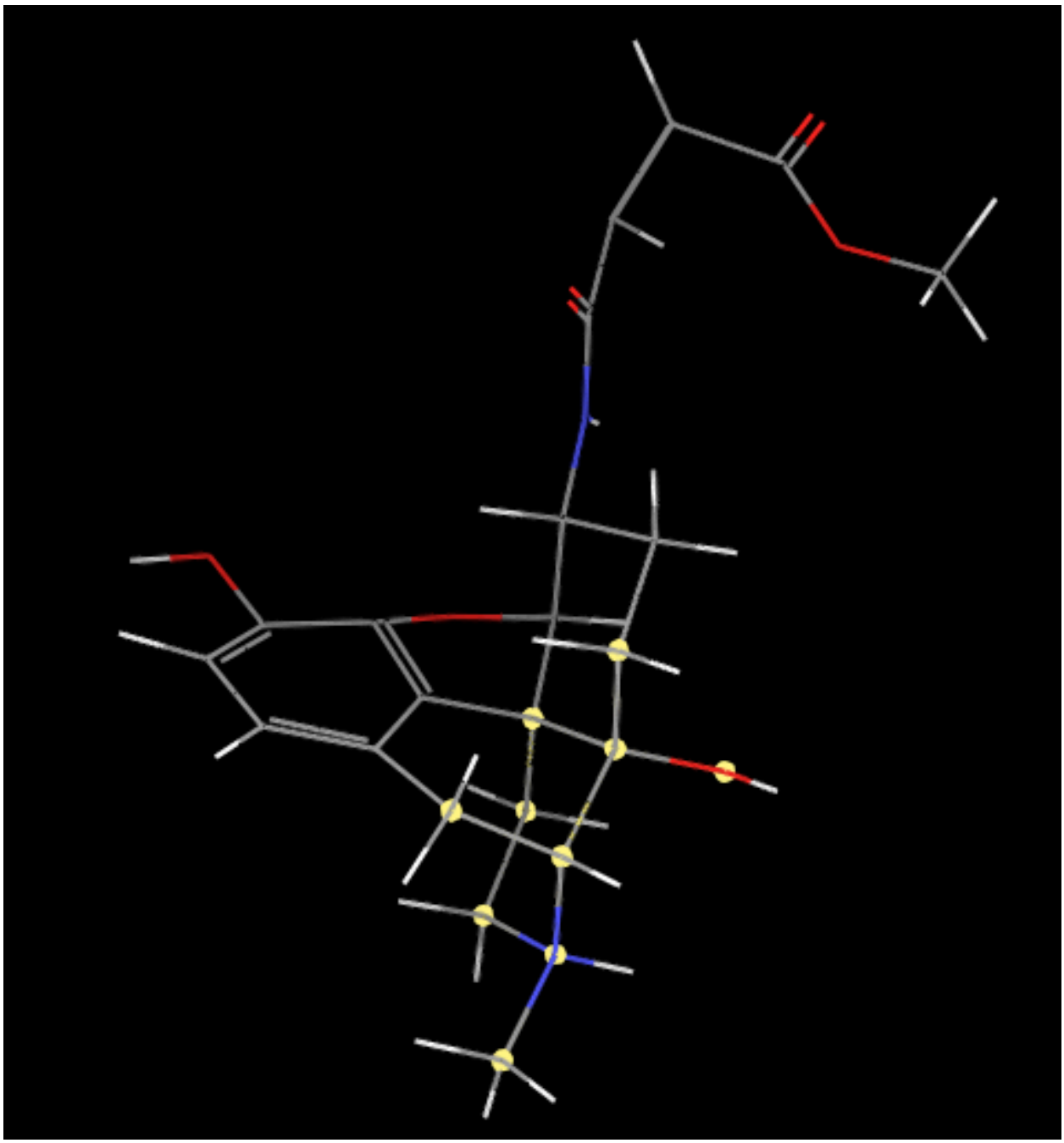
Fragment 1

The other two fragments will be:



Fragment 2

And:



Fragment 3

Note that all fragments have *at least* one overlapping carbon with one or more other fragments. Make sure to write down the names of those overlapping carbons.

Add hydrogens to all new fragments and use paramchem to obtain the respective .str files.

Check your work:

List all the carbons that overlap between your fragments.

If you did this step correctly, it should be:

CAH, CAQ, CAM, CAO

Step 4: Parameterize the fragments

For each molecule fragment mol2 file, upload the result to ParamChem. Save .str files with the output of each.

As a sanity check, you should open up each Mol2 in PyMol, and see that the full molecule is indeed composed of each of the fragments uploaded to ParamChem, and that together they comprise the parent molecule exactly.

If you split the molecule reasonably, avoiding molecular bridges and multiple fused rings, you should receive relatively low penalties. If not, you might want to reconsider the way in which you have fragmented your molecule, and see if a different split would be more optimal.

Note how each fragment overlaps at a carbon atom. This is counterintuitive: instead of splitting bonds, we are splitting over individual atoms. We will see how to handle this in the following section.

(Our penalties for for betaFOA are:
Fragment 1: param penalty= 59.400 ; charge penalty= 26.497
Fragment 2: param penalty= 75.100 ; charge penalty= 17.571
Fragment 3: param penalty= 12.000 ; charge penalty= 26.091
Complete FOA: param penalty= 99.000 ; charge penalty= 64.257)

Step 5: Stitch the ligand back together

This section will review how to use the fragments generated to assign partial charges to the parent molecule. At a higher level, for each *non-overlapping* atom, i.e. atoms unique to a fragment, we will simply take the new partial charge assigned by ParamChem to that fragment and this will constitute the partial charge in the final parameterization of the ligand. For *overlapping* atoms, i.e. atoms common to multiple fragments, we will sum the individual partial charges assigned to the same atom in different fragments to get its total partial charge for the final parameterization. While this is counterintuitive at first, think of yourself as a chemical accountant. Your goal is to make the final charge of the whole molecule add up to an integer corresponding to its total charge. To do this, if you "superimposed" each fragment – each of which has an integer charge – on top of each other, and added all the charges, you would get the final integer you desire. However, if you did this, for overlapping carbons, in the final molecule, those carbons would end up with more than four hydrogens each. So, in each molecular fragment, we will delete hydrogens of overlapping carbons, add the charges of each of those hydrogens (usually 0.090 each) to

their respective carbons, and then add the charges of those carbons together to get the charge of the carbon on the final str file.

First, take the .str file for the parent molecule, make a copy, and name the copy [my_molecule]_stitched.str.

Notice the anatomy of the .str file.

The very first line should contain the proper residue name and partial charge of your ligand, e.g.,:

RESI BF0 Displa 1.000

The first section contains partial charges and atom types for all atoms in your molecule.

AtomName | Atom Type | Partial Charge | ! Comment with penalty score

The next section contains topology information, i.e. how each of the atoms is bonded:

BOND Atom1 | Atom2
Where Atom1 and Atom2 are PDB atom names, *not* atom types.

The next section contains information on bond strength. In CHARMM each bond is modeled as a spring, and this gives the spring constant:

AtomType1 | AtomType2 | Eq-distance | Spring-Constant

Notice how these are atom *types*. When CHARMM (or CHAMBER) assigns parameters to all bonds in your system, it will look for known parameters in the rtf/prm or str files it is given. So for all bonds where one atom is of AtomType1 and the other atom is of AtomType2, it will assign a bond with the given equilibrium distance and spring constant.

The story is similar for *angles* (subsequent section):

AtomType1 | AtomType2 | AtomType3 | “spring”-constant | angle

And the last section for dihedrals:

AtomType1 | AtomType2 | AtomType3 | AtomType4 | Phi_k (barrier height) | periodicity | phase angle

Now, we will use this information for our parameterization of the ligand at hand. First, we will deal with the charges.

Partial Charge Assignment

For all *non-overlapping* atoms – i.e., for atoms unique to each fragment – the task is straightforward. In the [your_molecule]_stitched.str file, for each non-overlapping atom in the partial charges section, make a copy of each "AtomName | Atom Type | Partial Charge | !" line, paste the copy under the original line, and place a "!" exclamation point in front of the pasted line. This will serve as a "save" so that, if anything goes wrong later (which may very well happen when you are doing something this intricate by hand), you will be able to trace back all of your steps exactly. Now, find the partial charge in the respective fragment for this atom, copy the new charge in the fragment's .str file, and paste that into the [your_molecule]_stitched.str file for the same atom.

So now, for each *non-overlapping atom*, in the [your_molecule].str file, you will have lines that look like this:

AtomName | Atom Type | New Partial Charge | ! Comment with penalty score
!AtomName | Atom Type | Original Partial Charge | ! Comment with penalty score

Do this for all hydrogens too *except for hydrogens of overlapping atoms. For hydrogens of overlapping atoms, leave them* ***untouched*** (i.e., use the parameters derived from the unfragmented ligand). If you split over sp3 carbons, the hydrogens will have charge 0.090 and should remain unchanged in the [your_molecule]_stitched.str file.

Now, for *overlapping atoms*, it's a bit more involved.

The algorithm is like so:

For each overlapping atom (i) (in this example, for each atom in the set {CAH, CAQ, CAM, CAO}, where $1 \leq i \leq$ number_overlapping_atoms

> In the [your_molecule]_stitched.str file, for each overlapping atom in the partial charges section, make a copy of each "AtomName | Atom Type | Partial Charge | !" line, paste the copy under the original line, and place a "!" exclamation point in front of the pasted line. In the line that is not commented out, add an exclamation point to the end of it. Following the exclamation point, you will write out the individual charges (partial charges of overlapped carbons) that contribute to its total charge.
>
> **In each** fragment (j) file ([your_molecule]_frag_j.str where that atom occurs, where $1 \leq j \leq$ number_of_fragments
>
> > find the partial charge in the respective fragment for this atom, call it $c_{i,j}$. Add a new exclamation point at the far right of the row. Here we will calculate the partial charge of that carbon plus the partial charges of the hydrogens to which it is bonded in that fragment. Find out how many

hydrogens it has. The hydrogens are named after the carbons to which it is bonded. For example, if the carbon is named "CAW", the hydrogen will be named "HAW1," "HAW2," etc. Add the charges of those hydrogens to the charges of the carbon, and write out the expression for this at the far-right of the row.

For example, for CAO, for the upper fragment, it should look like:

ATOM CAO CG321 **0.141** ! = -0.039 + 2 * 0.09 =
! ATOM CAO CG321 -0.039 ! 2.567

Add to the row of the [your_molecule]_stitched.str file the final charge obtained for that atom (0.141 for CAO in that particular fragment).

For each overlapping atom, the number of charges that are added to obtain its final value should equal the number of fragments in which it appears plus the term that , subtracts the charges of the hydrogens that are present in the complete molecule:. For CAO, the lines should look like this in []_stitched.str:

ATOM CAO CG311 0.081 ! (-0.007+2*0.09) + (-0.272+3*0.09) – (1*0.09) = 0.081
!ATOM CAO CG311 0.034 ! 19.778

Note: these charges are just an example and may look different for you depending on how you split up your molecule.

To practice this, try parameterizing **betaFOA**. This is identical to betaFNA but for a seemingly sublet (though it turns out to be major difference!) distinction, namely it has a methyl group instead of a methylcyclopropyl and, instead of a bond to Lys233 of the Mu-Opioid Receptor, it has a double bond between its CAW and CAX carbons.

In this exercise:

1. Download 4DKL from the PDB. Import it into Maestro.
2. Delete all non-4DKL atoms
3. Increase the bond order by 1 of the CAW-CAX bond to make it an alkene. Both CAW and CAX should now be sp2 hybridized carbons.
4. Add a partial charge of +1 to the tertiary amine nitrogen. To do this, use the "build" toolbar and use the + sign lightning-bolt icon.
5. Assign bond orders and then add hydrogens to the molecule.
6. Split the molecule as described in this tutorial, get parameters from ParamChem, and stitch the fragments back together into [your_molecule]_stitched.str. See if your final parameters match ours! Here are the final values we got:

* Topologies generated by
* CHARMM General Force Field (CGenFF) program version 0.9.7.1 beta

```
*
36 1

! "penalty" is the highest penalty score of the associated parameters.
! Penalties lower than 10 indicate the analogy is fair; penalties between 10
! and 50 mean some basic validation is recommended; penalties higher than
! 50 indicate poor analogy and mandate extensive validation/optimization.

!CAH (two hydrogens, frag_1 and frag_2)
!CAM (zero hydrogens) (frag_1, frag_2)
!CAO (one hydrogen)                  (frag_1, frag_3)
!CAQ (two hydrogens)         (frag_2, frag_3)


RESI BF0 Displa      1.000
! stitched together from a file with param penalty=  99.000 ; charge penalty=  64.257
GROUP          ! CHARGE   CH_PENALTY

ATOM CBF    CG3C31 -0.180 !    0.075
!ATOM CBF    CG3C31 -0.180 !    0.075

ATOM CBG    CG3C31 -0.180 !    0.075
!ATOM CBG    CG3C31 -0.180 !    0.075

ATOM CBE    CG3C31 -0.092 !    4.770
!ATOM CBE    CG3C31 -0.092 !    4.770

ATOM CBD    CG324   0.189 !    6.172
!ATOM CBD    CG324   0.178 !    6.172

ATOM NAJ    NG3P1  -0.363 !   21.778
!ATOM NAJ    NG3P1  -0.364 !   21.778

ATOM CAK    CG324   0.178 !   12.414
!ATOM CAK    CG324   0.159 !   12.414

ATOM CAL    CG321  -0.180 !   31.829
!ATOM CAL    CG321  -0.168 !   31.829

ATOM CAM    CG3RC1  0.082 !        (-0.185+0.18) + (-0.093+0.18) - (0)
!ATOM CAM    CG3RC1  0.084 !   53.701

ATOM CBC    CG2RC0  0.130 !   64.257
!ATOM CBC    CG2RC0  0.141 !   64.257

ATOM CAN    CG3RC1  0.183 !   30.066
```

!ATOM CAN CG3RC1 0.119 ! 30.066

ATOM OAA OG3C51 -0.313 ! 16.285
!ATOM OAA OG3C51 -0.244 ! 16.285

ATOM CAB CG2RC0 0.040 ! 32.087
!ATOM CAB CG2RC0 0.051 ! 32.087

ATOM CAC CG2R61 0.010 ! 3.709
!ATOM CAC CG2R61 0.010 ! 3.709

ATOM OAD OG311 -0.529 ! 3.576
!ATOM OAD OG311 -0.529 ! 3.576

ATOM CAE CG2R61 -0.153 ! 0.000
!ATOM CAE CG2R61 -0.149 ! 0.000

ATOM CAF CG2R61 -0.218 ! 8.946
!ATOM CAF CG2R61 -0.219 ! 8.946

ATOM CAG CG2R61 -0.134 ! 51.211
!ATOM CAG CG2R61 -0.136 ! 51.211

ATOM CAH CG321 -0.158 ! (-0.275+0.27) + (-0.243+0.27) – (0.18)
!ATOM CAH CG321 -0.184 ! 38.320

ATOM CAI CG314 0.192 ! 42.658
!ATOM CAI CG314 0.228 ! 42.658

ATOM CAR CG301 0.223 ! 40.366
!ATOM CAR CG301 0.199 ! 40.366

ATOM OAS OG311 -0.639 ! 16.482
!ATOM OAS OG311 -0.638 ! 16.482

ATOM CAQ CG321 -0.197 ! (-0.27+0.27) + (-0.287+0.27) – (0.18)
!ATOM CAQ CG321 -0.187 ! 12.983

ATOM CAP CG321 -0.179 ! 6.322
!ATOM CAP CG321 -0.180 ! 6.322

ATOM CAO CG311 0.081 ! (-0.007+0.18) + (-0.272+0.27) – (0.09)
!ATOM CAO CG311 0.034 ! 19.778

ATOM NAT NG2S1 -0.462 ! 11.253
!ATOM NAT NG2S1 -0.422 ! 11.253

```
ATOM CAU    CG2O1   0.511 !   15.556
!ATOM CAU    CG2O1   0.494 !   15.556

ATOM OAV    OG2D1  -0.500 !    0.000
!ATOM OAV    OG2D1  -0.498 !    0.000
ATOM CAW    CG2DC1 -0.156 !   14.632
ATOM CAX    CG2DC1 -0.051 !   22.660
ATOM CAY    CG2O2   0.604 !   26.497
ATOM OAZ    OG2D1  -0.636 !    9.883
ATOM OBA    OG302  -0.294 !   18.444
ATOM CBB    CG331  -0.036 !   10.542
ATOM HBF1   HGA2    0.090 !    0.000
ATOM HBF2   HGA2    0.090 !    0.000
ATOM HBG1   HGA2    0.090 !    0.000
ATOM HBG2   HGA2    0.090 !    0.000
ATOM HBE    HGA1    0.090 !    0.075
ATOM HBD1   HGA2    0.090 !    0.030
ATOM HBD2   HGA2    0.090 !    0.030
ATOM HAK1   HGA2    0.090 !    0.691
ATOM HAK2   HGA2    0.090 !    0.691
ATOM HAL1   HGA2    0.090 !    2.504
ATOM HAL2   HGA2    0.090 !    2.504
ATOM HAN    HGA1    0.090 !    3.579

!ATOM HOAD   HGP1    0.420 !    0.075
ATOM HOAD   HGP1    0.420 !    0.075

ATOM HAE    HGR61   0.197 !    0.000
ATOM HAF    HGR61   0.197 !    0.000

ATOM HAH1   HGA2    0.090 !    0.121
ATOM HAH2   HGA2    0.090 !    0.121
ATOM HAI    HGA1    0.090 !    3.008

ATOM HOAS   HGP1    0.431 !    8.269
!ATOM HOAS   HGP1    0.442 !    8.269

ATOM HAQ1   HGA2    0.090 !    0.905
ATOM HAQ2   HGA2    0.090 !    0.905
ATOM HAP1   HGA2    0.090 !    0.000
ATOM HAP2   HGA2    0.090 !    0.000
ATOM HAO    HGA1    0.090 !    1.050

ATOM HNAT   HGP1    0.293 !    7.412
!ATOM HNAT   HGP1    0.297 !    7.412
```

```
ATOM HAW    HGA4    0.150 !    1.425
ATOM HBB1   HGA3    0.090 !    0.000
ATOM HBB2   HGA3    0.090 !    0.000
ATOM HBB3   HGA3    0.090 !    0.000

ATOM HNAJ   HGP2    0.319 !    2.473
!ATOM HNAJ   HGP2    0.319 !    2.473
ATOM HAX    HGA4    0.150 !    2.975
```

Angles, torsions, bonds…

Bonded Parameters

For angles, torsions, and bonds, you should likely use those parameters assigned to the fragments, not those that ParamChem initially assigned. However, this is bound to be a painstaking process very easily prone to human error for some very specific, practical reasons. First, CHARMM str/prm files contain bonded parameters in a format predicated on the atom **type**, not the atom **name**. This is compounded by a second problem: the atom types assigned by ParamChem to the individual fragments are frequently not the same as the atom types assigned to the original whole molecule.

The solution is simple for atoms not common to different fragments: use the atom types and parameters of the individual fragments.

The solution is not at all obvious for atoms common to different fragments. The same atom in different fragments may be assigned to different atom types based on its adjacent atoms and general covalent electronic environment.

Therefore, I present the following methodology, which ultimately will be codified in code since it would be laborious, tedious, and error-prone to do by hand. You will use the original atom types assigned for the molecule as a whole by paramchem.

For each parameter in the individual fragment, see if it's in the master fragment already. To do this, you will have to do the following translation: fragment_atom_type → fragment_atom_name → master_atom_type. If

For each atom listed in the rtf file of the original (non-fragmented) ligand:

Create a dictionary with atom names as keys and atom types as values

Create a list, called frag_list, which is going to be a list of atom name → atom type dictionaries for each fragment

For each fragment
    Create a dictionary with atom names as keys and atom types as values, and add that dictionary to frag_list

In the main str/prm file for the stitched together ligand:
    Create dictionary mapping 2-tuples to doubles called master_bonds
    For each bond:
        master_bonds[(atomtype_1,atomtype_2)] = bond

    Create dictionary mapping 3-tuples to 2-tuples called master_angles
    For each angle:
        master_bonds[(atomtype_1,atomtype_2)] = (angle_1, angle_2)

    Create dictionary mapping 4-tuples to 3-tuples called master_dihedrals
    For each dihedral:
        Master_dihedral[(at_1,at_2,at_3,at_4)] = (term1, term2, term3)

Repeat the same as above for each of the fragments, placing each individual fragment dictionary for each parameter type into a list.

Translate all dictionaries from atom types to atom names:

Translate(input, original dictionary, input that fragment's name→type dictionary, input master dictionary)
    Create new dictionary to return (shallow copy the input)
    For each key in dictionary:
        For each atom type in key:
            Look up atom name in that fragment's dict
            Translate that name in the master dictionary to a type
        Pop to change the key in the new dictionary
    Return the dictionary

For each parameter type:
    For each dictionary:
        For each key:
            Check if that key is in the master dictionary
                If so, change its value to the new value
                If not, add a new key → value

Function writeOutput:
    Write a new rtf/prm file

Function translateType(input: atom type of fragment atom):

For each angle:

For each dihedral:

Dihedral Fitting in Paramfit

Paramfit is an excellent system created by Robin Betz to create accurate bonded parameters for small organic molecules. A tutorial for it is available at:

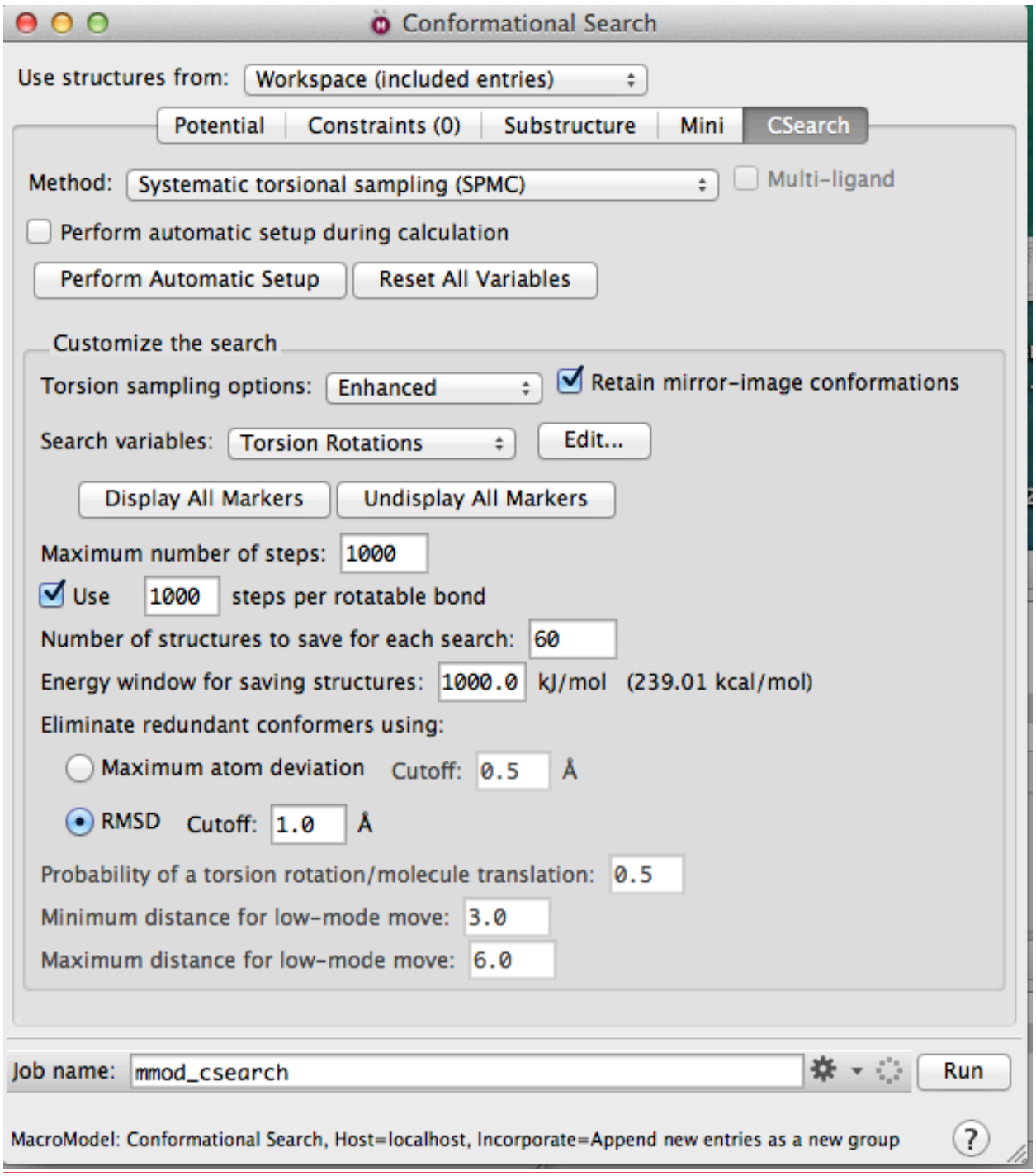
http://ambermd.org/tutorials/advanced/tutorial23/

However, there are several added complications stemming from the fact that you will be using the CHARMM force field instead of AMBER, for which Paramfit was originally designed. This section of the tutorial will guide you through this process for your CHARMM-parameterized molecule. In particular, you will fit the dihedral parameters of your choice for your molecule.

1. Generate conformations representing different torsional states of your molecule.

Import a mol2 file of your ligand into Maestro (properly setting up your molecule is covered earlier in the tutorial). Go to tasks → conformational search → advanced search. This is Schrödinger's built in utility for finding different potential conformations of a given ligand.

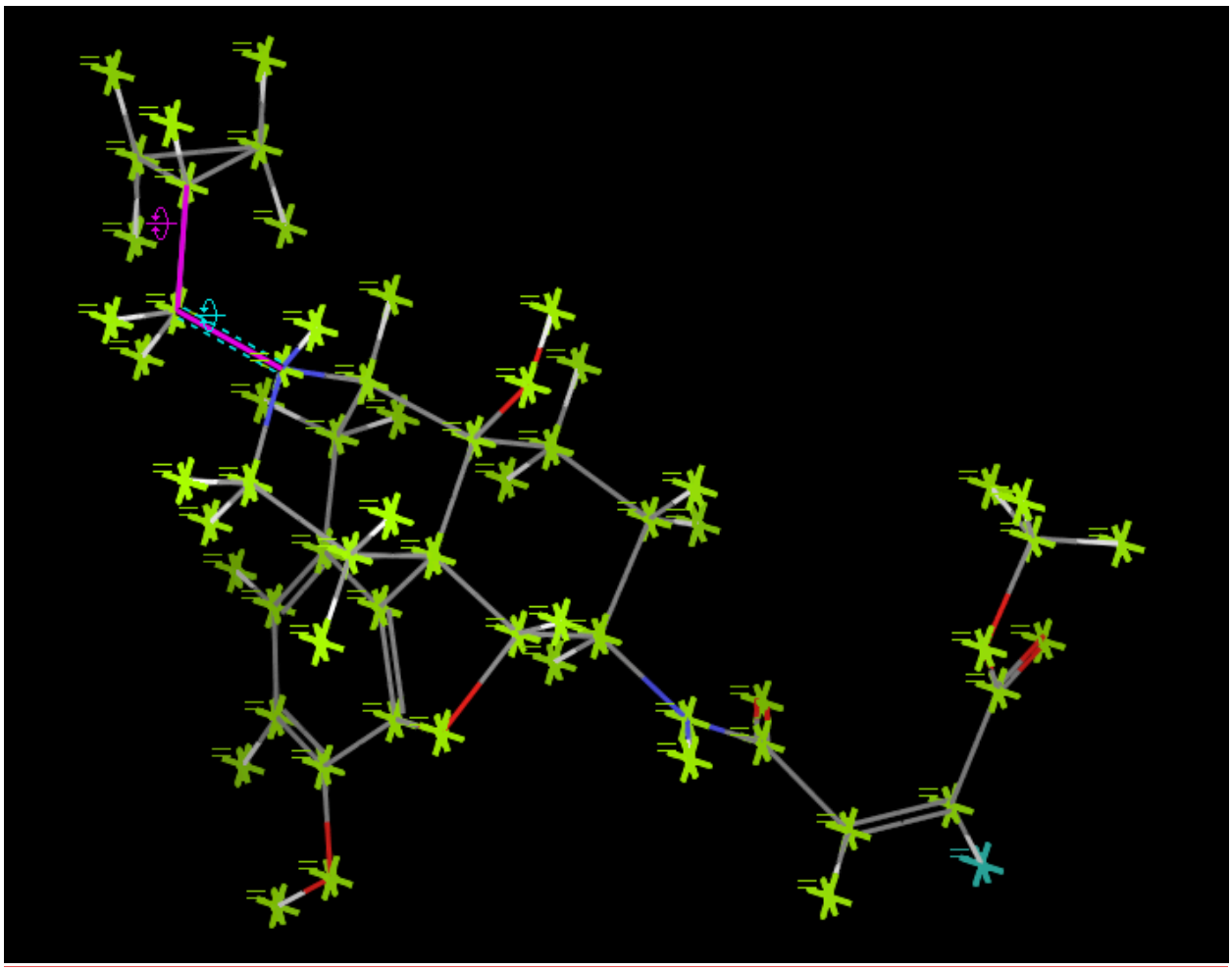
Choose the options displayed below. You may want to modify them based on the specific application.



Under search variables, go to Torsion Rotations, and hit “Edit…”. Change “pick” to “Bonds.” Now click on the bonds for which you would like to sample dihedral angles on your molecule in the workspace.

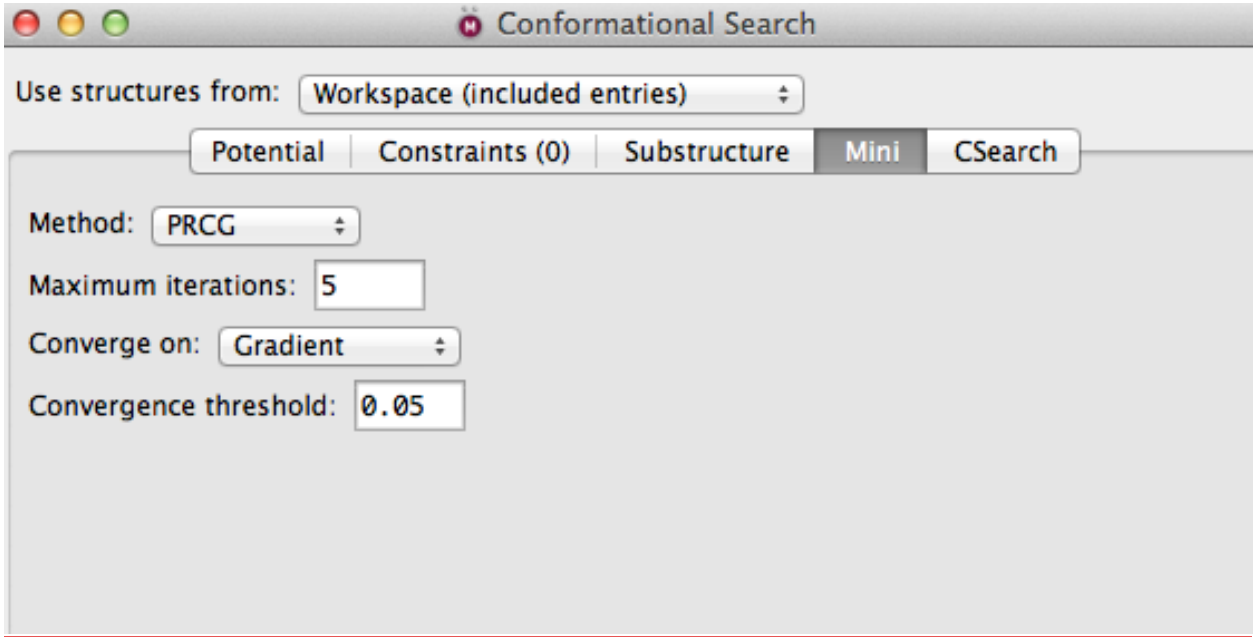
Now under search variables go to Comparison Atoms. Hit “Edit”. Select all atoms.

Your molecule should now look something like this:



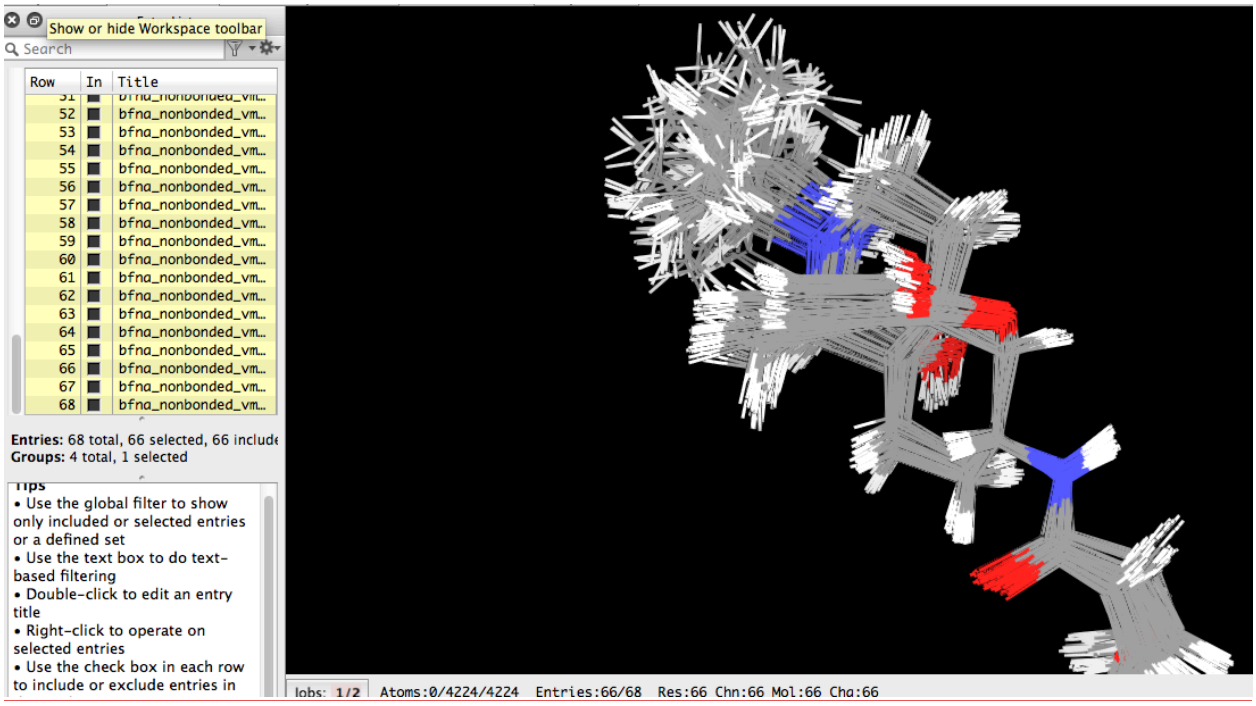
Make sure your “Maximum atom deviation” is large enough such that Schrodinger doesn’t generate too many redundant conformers. I routinely set mine as high

[This step is optional]
Under the “Mini” tab, you can reduce the number of minimization steps done. This is to make sure Maestro doesn’t spend an inordinate amount of time generating conformations, especially for large molecules for which you are sampling several dihedrals, and also to make sure that you will sample a large range of the molecule’s conformational space. If Schrodinger minimizes “too well,” your top 1000 or so conformers will end up all falling under similar torsion angles and you won’t sample the conformational space very well.

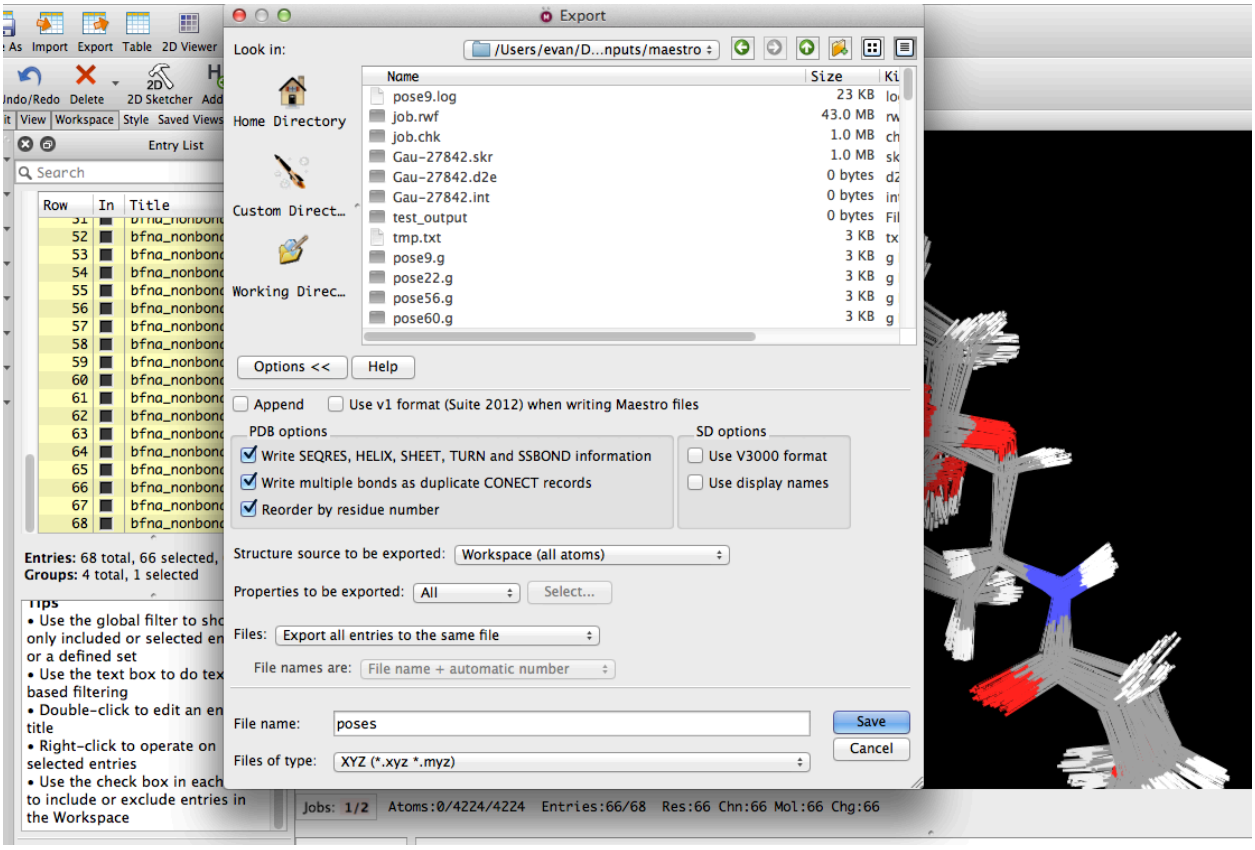


Now hit “Run”.

After a couple minutes, you will get something like this (after selecting all entries on the left):

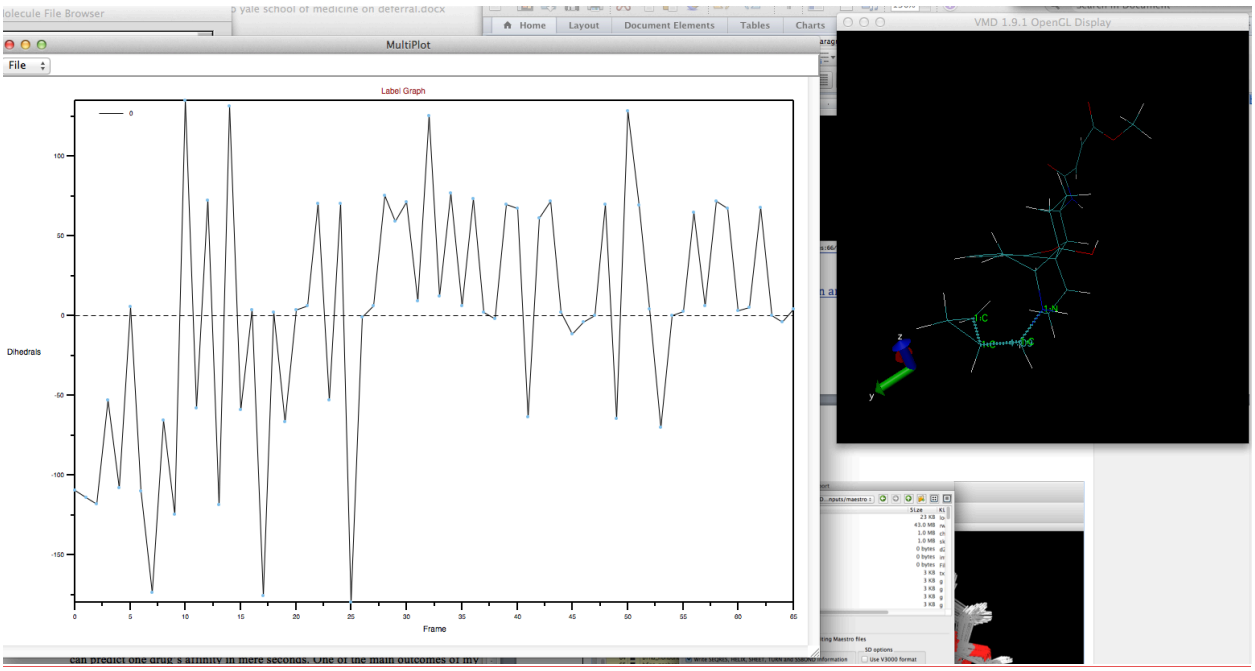


Right click on the entries on the left column and export them in “xyz” format to the same file:



Repeat this, exporting them in Gaussian Z-matrix format as well, but this time choosing to Export all entries to different files, with the file name “pose”, in some new directory of your choice.

Now open the result in VMD. Measure the dihedral of interest, and make sure it is reasonably well sampled. If it is not, go back to the Conformational Search step, and you might have to change some variables. For example, you might have to increase the maximum energy allowed to allow for more sampling; you might have to increase or decrease RMSD to make sure Schrodinger saves less or more structures, you might have to change the number of steps applied for each dihedral angle, etc. This is usually an iterative process. The benefit of using Schrodinger for this is it estimates a single point energy for each structure and therefore automatically filters them so you don’t get any with ridiculously large energies.



Examining the dihedral angles in the different conformations in VMD. Note how it needs to more finely sample certain angles.

Generating an AMBER compatible prmtop (parameter + topology) file for your ligand

In this section, you will use a combination of PSFGEN, VMD, and CHAMBER to generate a .prmtop file of your ligand, a prerequisite for Paramfit. For this section, you will need:

A PDB/MOL2 file of your ligand
RTF/PRM files describing topology and parameters for your ligand

The general arc will be:

Use PSFGEN in VMD to convert a PDB/MOL2 file of your ligand to a PSF file containing molecular topology and parameters.

Then, you will use CHAMBER in ParmEd to convert your coordinate (PDB) and topology (PSF) and parameter (RTF/PRM) info and generate an amber .prmtop file.

Step 1: Open VMD. Import your molecule in PDB or MOL2 format. Go to Extensions → Modeling → Automatic PSF Builder

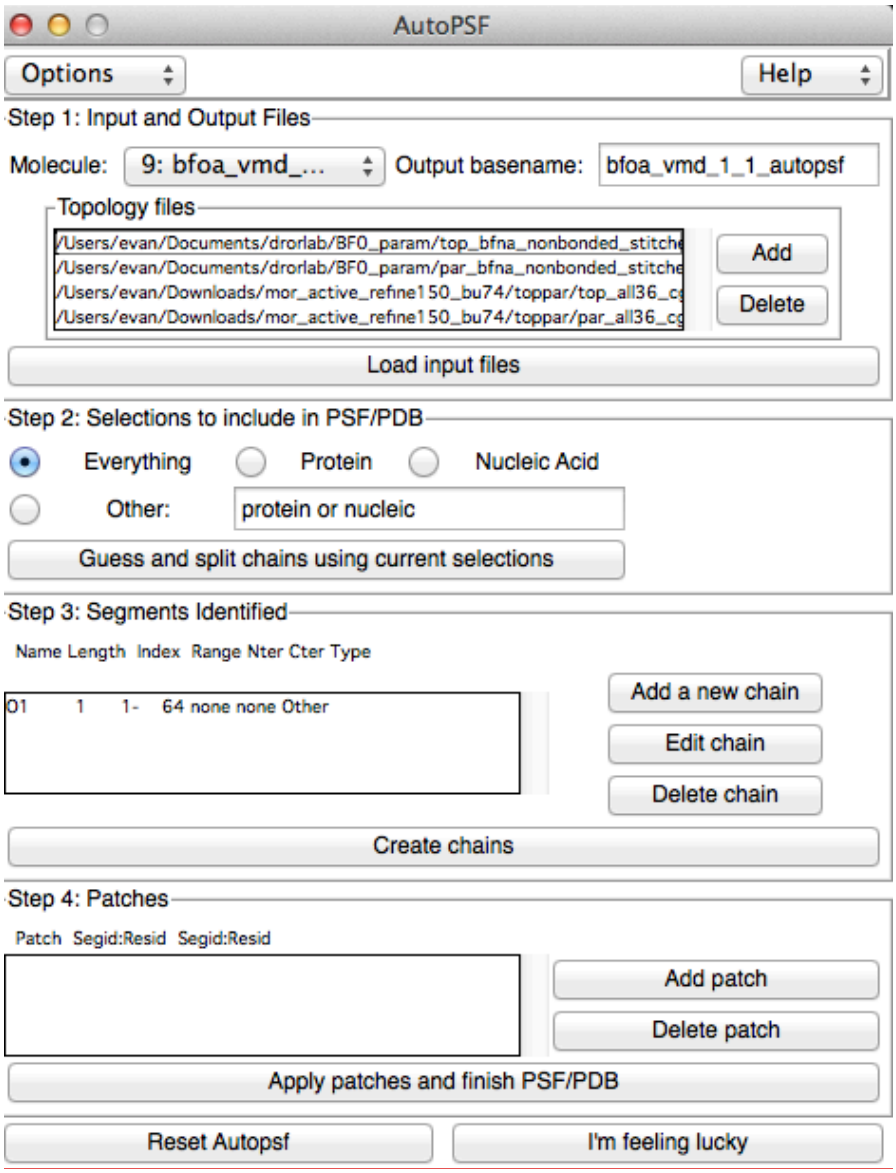
Choose the molecule for which you want to generate a PSF on the top left.

Then click “Add” to add all RTF and PRM files needed to parameterize your molecule. This will likely be the RTF and PRM files you derived from ParamChem, in addition to the “CGenFF” files located in $ambersim/tutorial/toppar. Then, hit “Load Input Files.”

Proceed to click “Guess and split chains…” You probably don’t need to specify the original coordinate file.

Now, hit “Create Chains,” and if all goes well, a new PDB/PSF file combination will be generated.

The window should ultimately look like this (with differences depending on your specific ligand):



Generating .prmtop file in ParmEd Chamber

Now you will use the PSFGEN–generated PDB and PSF files to create a .prmtop file using Chamber command in ParmEd.

Move the pertinent files to a new directory in Sherlock. You should need the following:

par_all36_cgenff.prm
par_all36_prot.prm [if your ligand is covalently bonded to an amino acid]
top_all36_cgenff.rtf
top_all36_prot [[if your ligand is covalently bonded to an amino acid]]
[your_ligand].rtf
[your_ligand].prm
[your_ligand].psf
[your_ligand].pdb

where the last two files should have been generated by PSFGen.

Now open ParmEd in the directory containing these files, and enter a command like so:

chamber –top top_all36_cgenff.rtf –top [your_ligand].rtf –prm par_all36_cgenff.prm –prm [your_ligand].prm –psf [your_ligand].psf –crd [your_ligand].pdb –box bounding

When it finishes, save your prmtop and inpcrd files by using:

outparm [your_ligand].prmtop [your_ligand].inpcrd

Generating mdcrd file

Now you will use this prmtop and the .xyz file you generated earlier in Maestro to obtain a trajectory file in .mdcrd format needed for Paramfit.

Make sure AMBER is loaded on Sherlock (ml load amber/14-intel). Transfer the .xyz file containing all the poses/conformations you generated earlier in Schrodinger to the current directory containing the prmtop file.

Now, in the directory containing all the files, run:

cpptraj [your_ligand].prmtop

If it loads properly, run:

cpptraj [your_poses_file].xyz
If that also works without errors, run:

trajout [your_poses_name].mdcrd

Now run (not kidding):

go

IT should generate a message saying that the mdcrd file has been generated.

Now quit cpptraj, you have completed this phase.

Generating Quantum Energies

Now, you have all the ingredients you need to use Paramfit to parameterize your dihedral of choice. For this, Robin's very well written and documented Paramfit tutorial should do the trick.

From my own experience using it, here are some tips and tricks to smooth the process.

First, to generate quantum input files, it is required to create ".com" or ".g" files in Gaussian format (or whatever quantum code is your favorite). The problem is that it is nontrivial to interpolate bond orders and overall formal charges from the .prmtop file itself. The codes Paramfit calls will frequently get these wrong, and then you will have to manually fix the files. To get around this issue:

1. Open the project file pertaining to the conformational search you did earlier in Maestro, if it's not still open. Now select all the conformers on the left-hand panel, and right click on them to "Export" the structures. Choose as your output format Gaussian Z-matrix, and make sure to choose to export each structure to its own file. Name the file "pose". Therefore, each structure will have format pose1.g, pose2.g, etc.
2. Transfer all these files to a directory on Sherlock called "[paramfit_session_directory]/quantum_inputs"
3. Modify the inputs. Scripts exist in the directory: $ambersim/paramfit_files
    a. Modify the file "gaussian.header" to your liking (the defaults already there should be fine. But if you want to change level of theory or basis set this is where to do it)
    b. Now run "mod_inputs.bash" in the directory immediately parent to "quantum_inputs", the dir containing all of your .g files
4. Now you should be able to run "do_qm.bash" and it will automatically create a directory called "quantum_outputs" and calculate energies for each of your structures. Depending on your structure, this step can take many hours
5. Modify "process_qm.bash" to reflect the total number of structures you have (i.e., change "67" to the number of conformations in your mdcrd file), and run it