How to Create Gromacs Molecules from Scratch

or

My Personal Gromacs Workflow

Michael J. Quevillon

May 24, 2017

Contents

1 Creating the Molecule

In general, you will have a chemical structure of a molecule that you would like to simulate. However, Gromacs needs 3D set of coordinates for the molecule.

The tool that I have been using for this purpose is Avogadro. This allows you to view and edit molecular files relatively easily. To install (on Ubuntu/Debian-like systems), use

```
$ sudo apt-get install avogadro
```

Now you can begin drawing your molecule. Select the Draw Tool (or F8) and begin creating your molecule. It can be a rough sketch of the molecule, as we will optimize its geometry in a later step.

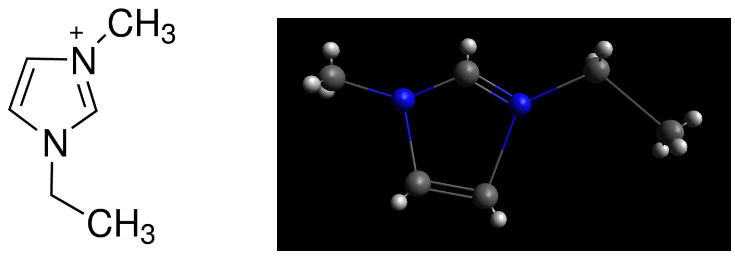

Figure 1: The chemical structure of 1-ethyl-3-methylimidazolium ([EMIM$^+$]), along with a rough sketch in Avogadro. The bonds between carbon and nitrogen atoms are exaggerated.

Once your molecule is properly attached with correct order of bonds and number of hydrogen atoms, we need to optimize its geometry. First, we need to choose a force field to use during the optimization process. This choice does not matter much at this step. The menu for this is found under Extensions > Molecular Mechanics > Setup Force Field... . These are all generalized force fields; I might suggest Generalized Amber Force Field (GAFF) or Universal Force Field (UFF) as a good starting point. Then, perform the optimization by Extensions > Optimize Geometry or Ctrl+Alt+o.

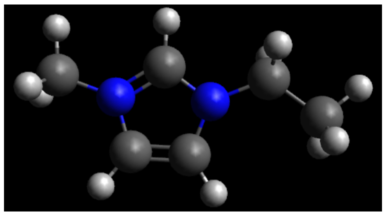

Figure 2: The UFF-optimized structure of [EMIM$^+$].

When saving this file, Avogadro defaults to `.cml`, but a `.pdb` is more useful to Gromacs. The `.pdb` file for this example is listed on page 4. You may find it useful to edit the `.pdb` file directly in order to rename the compound (`COMPND`) or the molecule name (Avogadro defaults to `LIG`). The most important part of this file is the coordinates for each molecule. The `CONECT` records show bonds between atoms, but these will not be used by Gromacs directly. Gromacs will typically accept `.gro` or `.pdb`, depending on what functions you are using.

A sample `.pdb` file for [EMIM$^+$]

```
COMPND    UNNAMED
AUTHOR    GENERATED BY OPEN BABEL 2.3.2
HETATM    1  N   LIG     1      -1.354  1.937 -0.000 1.00  0.00           N
HETATM    2  N   LIG     1       0.889  1.828 -0.004 1.00  0.00           N
HETATM    3  C   LIG     1      -0.201  2.646 -0.005 1.00  0.00           C
HETATM    4  C   LIG     1      -1.000  0.644  0.004 1.00  0.00           C
HETATM    5  C   LIG     1       0.395  0.568  0.003 1.00  0.00           C
HETATM    6  H   LIG     1      -1.710 -0.176  0.008 1.00  0.00           H
HETATM    7  H   LIG     1       0.971 -0.346  0.004 1.00  0.00           H
HETATM    8  C   LIG     1      -2.735  2.453 -0.000 1.00  0.00           C
HETATM    9  H   LIG     1      -2.766  3.564 -0.003 1.00  0.00           H
HETATM   10  H   LIG     1      -3.281  2.093  0.899 1.00  0.00           H
HETATM   11  H   LIG     1      -3.282  2.089 -0.897 1.00  0.00           H
HETATM   12  C   LIG     1       2.279  2.372 -0.012 1.00  0.00           C
HETATM   13  C   LIG     1       3.532  1.455  0.000 1.00  0.00           C
HETATM   14  H   LIG     1       2.383  3.020  0.881 1.00  0.00           H
HETATM   15  H   LIG     1       2.379  2.995 -0.922 1.00  0.00           H
HETATM   16  H   LIG     1       3.663  0.856  0.926 1.00  0.00           H
HETATM   17  H   LIG     1       4.415  2.144 -0.020 1.00  0.00           H
HETATM   18  H   LIG     1       3.654  0.814 -0.898 1.00  0.00           H
HETATM   19  H   LIG     1      -0.149  3.725 -0.010 1.00  0.00           H
CONECT    1    3    4    8
CONECT    2    3    5   12
CONECT    3    1    2   19
CONECT    4    1    5    6
CONECT    5    4    2    7
CONECT    6    4
CONECT    7    5
CONECT    8    1    9   10   11
CONECT    8
CONECT    9    8
CONECT   10    8
CONECT   11    8
CONECT   12    2   13   14   15
CONECT   12
CONECT   13   12   16   17   18
CONECT   13
CONECT   14   12
CONECT   15   12
CONECT   16   13
CONECT   17   13
CONECT   18   13
CONECT   19    3
MASTER        0    0    0    0    0    0    0    0   19    0   19    0
END
```

Another useful format in which Avogadro can save the file is `.xyz` (example below), which contains just the coordinates and type of each atom. The first line shows the number of atoms in the system. The second line is the "energy" of the system, based on whatever force field you happened to pick for optimization.

A sample `.xyz` file for [EMIM$^+$]

```
19
   Energy:     70.8369912
N       -1.35418      1.93716     -0.00013
N        0.88860      1.82794     -0.00366
C       -0.20062      2.64627     -0.00492
C       -0.99990      0.64390      0.00419
C        0.39475      0.56828      0.00269
H       -1.71027     -0.17593      0.00752
H        0.97129     -0.34556      0.00415
C       -2.73456      2.45292     -0.00034
H       -2.76581      3.56427     -0.00271
H       -3.28111      2.09251      0.89876
H       -3.28223      2.08863     -0.89718
C        2.27877      2.37178     -0.01174
C        3.53220      1.45502      0.00021
H        2.38293      3.02042      0.88064
H        2.37937      2.99544     -0.92213
H        3.66343      0.85647      0.92635
H        4.41515      2.14399     -0.01978
H        3.65407      0.81350     -0.89774
H       -0.14901      3.72494     -0.00953
```

2 Generating the Topology

There are many different methods to get the force field of a certain chemical system (from literature, generalized forces, coarse-graining schemes, etc.). This section will assume that you have a force field from the literature with all bond (2-body), angle (3-body), and dihedral (4-body) parameters. Each type of interaction has many different forms it can take. See Table 5.5 (and Chapter 4) in the Gromacs Manual for each type of interaction. A partial force field for [EMIM$^+$] is given below. However, a force field, by itself, does not define which atoms are connected by bonds!

2.1 Force Field File

If you have literature parameters for a certain force field, you will want to make a file containing these parameters in Gromacs syntax. (I suggest putting the values into a spreadsheet, then copying to a text file.) A sample is given below. It may help to include comments, designated by a semicolon (;), to remind you of the meaning of each column.

A partial `ffnonbonded.itp` file for [EMIM$^+$], from
`https://dx.doi.org/10.1021/ct900009a`

```
[ defaults ]
; nbfunc  comb-rule  gen-pairs  fudgeLJ  fudgeQQ
      1         2         yes      0.5   0.83333

[ atomtypes ]
; name  at_no    mass   charge  ptype   sigma  epsilon
  CR      6    12.0107 -0.090    A      0.355  0.292880
  NA      7    14.0067  0.220    A      0.325  0.711280
  CW      6    12.0107 -0.240    A      0.355  0.292880
  CM      6    12.0107 -0.350    A      0.350  0.276144
  CA      6    12.0107 -0.170    A      0.350  0.276144
  ...

[ bondtypes ]
; atom1  atom2  func     b0      kb
  CR      NA    1    0.1315  199576.8
  HM      CM    1    0.1080  142256.0
  CM      NA    1    0.1465  141000.8
  CR      HR    1    0.1069  153552.8
  CW      NA    1    0.1378  178656.8
  ...
```

```
[ angletypes ]
; i    j    k    func   th0      cth
  HM   CM   HM     1    109.8   138.0720
  HM   CM   NA     1    109.2   156.9000
  CM   NA   CR     1    126.4   292.8800
  CM   NA   CW     1    125.6   292.8800
  NA   CR   HR     1    125.1   146.4400
  ...


[ dihedraltypes ]
;  i    j    k    l   func   C0            C1          C2       C3
  NA   CA   CT   HT   3    00.00000    00.00000    00.00000  00.00000
  CW   NA   CA   CT   3   -31.12896    00.48116    19.14180  11.50600
  CR   NA   CA   CT   3   -08.78640   -06.17140    01.15060  13.80720
  CW   NA   CA   HA   3   -28.54534    07.87638    23.63960 -02.97064
  CW   NA   CA   CS   3   -24.61238    06.57934    20.92000 -02.88696
  ...

[ dihedraltypes ] ; improper torsions
;  i    j    k    l   func   C0            C1          C2       C3
   X   NA    X    X   3    08.36800    00.00000   -08.36800  00.00000
   X   CW    X    X   3    09.20480    00.00000   -09.20480  00.00000
   X   CR    X    X   3    09.20480    00.00000   -09.20480  00.00000
```

2.2 Interaction File

The topology file is what defines the connections between atoms of a molecule. If the molecule is small enough, you could create this file by hand. To do this, you would draw the molecule and label the atoms *in the same order* as your coordinate file. Then, you would enumerate all of the bonds, angles, and dihedrals of the molecule in another `.itp` file. These files are relatively simple compared to the force field files.

Note: There are tools to automatically generate these from a molecular file, mentioned in section 2.3.

`[ atoms ]` defines the different atom types in the molecule. `[ bonds ]` contains two-body interactions, only using the index of the atoms. Similarly, `[ angles ]` contains three-body interactions and `[ dihedrals ]` contains four-body interactions. [Whatever function form of the dihedral you use (3 in this example), also needs to be included at the end of each dihedral line.]

An example of this type of file is given on the page 8.

A partial `emim.itp` file.

```
[ moleculetype ]
; molname     nrexcl
CMI           3

[ atoms ]
; id  at type  res nr  residu name  at name  cg nr  charge
   1    NA       1        CMI      N01      1
   2    NA       1        CMI      N02      1
   3    CW       1        CMI      C03      1
   4    CW       1        CMI      C04      1
   5    CR       1        CMI      C05      1
   6    CM       1        CMI      C06      1
   7    CA       1        CMI      C07      1
   8    CT       1        CMI      C08      1
   9    HW       1        CMI      H09      1
  10    HW       1        CMI      H10      1
  11    HR       1        CMI      H11      1
  12    HM       1        CMI      H12      1
  13    HM       1        CMI      H13      1
  14    HM       1        CMI      H14      1
  15    HA       1        CMI      H15      1
  16    HA       1        CMI      H16      1
  17    HT       1        CMI      H17      1
  18    HT       1        CMI      H18      1
  19    HT       1        CMI      H19      1

[ bonds ]
1   3   ;N  C
1   5   ;N  C
1   6   ;N  C
2   4   ;N  C
2   5   ;N  C
2   7   ;N  C
3   4   ;C  C
3   9   ;C  H
...

[ constraints ]

[ angles ]
3   1   5   ;C  N  C
3   1   6   ;C  N  C
5   1   6   ;C  N  C
4   2   5   ;C  N  C
4   2   7   ;C  N  C
5   2   7   ;C  N  C
...

[ dihedrals ]
5   1   3   4   ;C  N  C  C
5   1   3   9   ;C  N  C  H
6   1   3   4   ;C  N  C  C
6   1   3   9   ;C  N  C  H
```

```
3   1   5   2   ;C  N  C  N
3   1   5   11  ;C  N  C  H
6   1   5   2   ;C  N  C  N
6   1   5   11  ;C  N  C  H
3   1   6   12  ;C  N  C  H
3   1   6   13  ;C  N  C  H
3   1   6   14  ;C  N  C  H
5   1   6   12  ;C  N  C  H
5   1   6   13  ;C  N  C  H
5   1   6   14  ;C  N  C  H
...
```

2.3 Automated Tool for Identifying Interactions

Several tools exist for creating the Interaction File (`emim.itp`) from a given `.xyz` file. I have had the best success with a Python script from verahill/linqvist's blog. In case the link goes down, a copy of this code is included in section A.

2.4 Combining Topology Files

The previous files have been "include topology" files, which cannot stand on their own. These must be combined into a topology file (`.top`), in order to run simulations in Gromacs. The order in which these files are included can make a difference; I suggest putting the force field parameters first, and then the interaction files. Then, you can name the system as well as specifying how many molecules are in the system. (See below for an example.)

A sample `topol.top` file for 100 ion pairs of [EMIM$^+$][NO$_3$$^-$].

```
; If using built-in OPLS parameters, uncomment the following line
; #include "oplsaa.ff/forcefield.itp"

; Include individual .itp files
#include "ffnonbonded.itp"
#include "cmi.itp"
#include "no3.itp"

[ system ]
EMIM/NO3 Ionic Liquid

[ molecules ]
; Compound #mols
CMI        100
NO3        100
```

3 Generating Initial Configuration

Like with many other steps in molecular simulations, there are many ways to create initial configurations of a system. Gromacs contains a built-in routine to randomly place molecules in a system, called `insert-molecules`. Continuing with the example, to insert 100 [EMIM$^+$] molecules into a cube of side length 5 nm, one method is to perform

```
$ gmx insert-molecules -ci emim.pdb -o box.gro -box 5 5 5 -nmol 100
```

This routine may not place all of the molecules that are defined by the `-nmol` flag. If this is the case, either increase the box size or decrease the number of inserted molecules.

If you are adding 100 [EMIM$^+$] molecules to a previous configuration (`prev.gro`), use the `-f` flag.

```
$ gmx insert-molecules -f prev.gro -ci emim.pdb -o box.gro -nmol 100
```

As long as the box is defined in the previous configuration (and it does not change), it does not need to be redefined when adding more molecules.

Other methods that I have used:

1. PACKMOL, which can randomly pack molecules in more complex geometries
2. My own (MATLAB) script(s) to place molecules into a unitcell of a crystalline lattice, then using `gmx genconf` to replicate to a full box
3. A previously created configuration file

4 Running Simulations with Gromacs

Once you have your topology file (`topol.top`) and your starting configuration (`box.gro`), you can begin running simulations in Gromacs!

Appendices

A Python Script to Generate Parameters

This is a copy of a Python script to generate parameters for the bonds, angles, constraints, and dihedrals of a molecule. Original source: https://verahill.blogspot.com/2013/10/524-generating-bonds-angles-and.html.

```
#!/usr/bin/python
import sys
from math import sqrt, pi
from itertools import permutations
from math import acos,atan2

# see table 5.5 (p 132, v 4.6.3) in the gromacs manual
# for the different function types


#from
#http://stackoverflow.com/questions/1984799/cross-product-of-2-different-\
#vectors-in-python
def crossproduct(a, b):
   c = [a[1]*b[2] - a[2]*b[1],
   a[2]*b[0] - a[0]*b[2],
   a[0]*b[1] - a[1]*b[0]]
   return c
#end of code snippet

# mostly from
#
    http://www.emoticode.net/python/calculate-angle-between-two-vectors.html
def dotproduct(a,b):
   return sum([a[i]*b[i] for i in range(len(a))])

def veclength(a):
   length=sqrt(a[0]**2+a[1]**2+a[2]**2)
   return length

def ange(a,b,la,lb,angle_unit):
   dp=dotproduct(a,b)
   costheta=dp/(la*lb)
   if costheta > 1:   #MQ
      costheta = 1   #MQ
   elif costheta < -1L: #MQ
      costheta = -1  #MQ
   angle=acos(costheta)
   if angle_unit=='deg':
      angle=angle*180/pi
   return angle
# end of code snippet

def diheder(a,b,c,angle_unit):
   dihedral=atan2(veclength(crossproduct(crossproduct(a,b),\
          crossproduct(b,c))),
               dotproduct(crossproduct(a,b),crossproduct(b,c)))
   if angle_unit=='deg':
      dihedral=dihedral*180/pi
   return dihedral

def readatoms(infile):
   positions=[]
   f=open(infile,'r')
   atomno=-2
   for line in f:
      atomno+=1
      if atomno >=1:
         position=filter(None,line.rstrip('\n').split(' '))
         if len(position)>3:
            positions+=[[position[0],int(atomno),\
            float(position[1]),float(position[2]),\
            float(position[3])]]
   return positions

def makebonds(positions,rcutoff,prevent_hhbonds):

   bonds=[]

   for firstatom in positions:
      for secondatom in positions:
         distance=round(sqrt((firstatom[2]-secondatom[2])**2\
                +(firstatom[3]-secondatom[3])**2\
                +(firstatom[4]-secondatom[4])**2)/10.0,3)
         xyz=[[firstatom[2],firstatom[3],firstatom[4]],\
            [secondatom[2],secondatom[3],secondatom[4]]]

         # print(firstatom)
         # print(secondatom)

         if distance<=rcutoff and firstatom[1]!=secondatom[1]:
            if prevent_hhbonds and (firstatom[0][0:1]=='H' and\
                              secondatom[0][0:1]=='H'):
               pass
            elif firstatom[1]<secondatom[1]:
               bonds+=[[firstatom[1],secondatom[1],\
               distance,firstatom[0],secondatom[0],xyz[0],xyz[1]]]
            else:
               bonds+=[[secondatom[1],firstatom[1],\
               distance,firstatom[0],secondatom[0],xyz[1],xyz[0]]]
   return bonds

def dedupe_bonds(bonds):

   newbonds=[]
   for olditem in bonds:
      dupe=False
      for newitem in newbonds:
         if newitem[0]==olditem[0] and newitem[1]==olditem[1]:
            dupe=True
            break;
      if dupe==False:
         newbonds+=[olditem]
   return(newbonds)

def genvec(a,b):
   vec=[b[0]-a[0],b[1]-a[1],b[2]-a[2]]
   return vec

def findangles(bonds,angle_unit):
   # for atoms 1,2,3 we can have the following situations
   # 1-2, 1-3
   # 1-2, 2-3
   # 1-3, 2-3
   # The indices are sorted so that the lower number is always first

   angles=[]
   for firstbond in bonds:
      for secondbond in bonds:
         if firstbond[0]==secondbond[0] and not
             (firstbond[1]==secondbond[1]): # 1-2, 1-3
            # print(firstbond)
            # print(secondbond)
            vec=[genvec(firstbond[6],firstbond[5])]
            vec+=[genvec(secondbond[6],secondbond[5])]
            # print(vec)
            angle=ange(vec[0],vec[1],firstbond[2]*10,secondbond[2]*10,angle_unit)
            angles+=[[firstbond[1],firstbond[0],\
            secondbond[1],angle,firstbond[4],firstbond[3],secondbond[4],firstbond[6],\
            firstbond[5],secondbond[6]]]

         if firstbond[0]==secondbond[1] and not
            (firstbond[1]==secondbond[1]): # 1-2, 3-1
            #this should never be relevant since we've sorted the atom
                numbers
            pass

         if firstbond[1]==secondbond[0] and not
            (firstbond[0]==secondbond[1]): # 1-2, 2-3
            vec=[genvec(firstbond[5],firstbond[6])]
            vec+=[genvec(secondbond[6],secondbond[5])]
            angle=ange(vec[0],vec[1],firstbond[2]*10,secondbond[2]*10,angle_unit)
            angles+=[[firstbond[0],firstbond[1],\
            secondbond[1],angle,firstbond[3],firstbond[4],secondbond[4],firstbond[5],\
            firstbond[6],secondbond[6]]]

         if firstbond[1]==secondbond[1] and not
            (firstbond[0]==secondbond[0]): # 1-3, 2-3
            vec=[genvec(firstbond[6],firstbond[5])]
            vec+=[genvec(secondbond[6],secondbond[5])]
            angle=ange(vec[0],vec[1],firstbond[2]*10,secondbond[2]*10,angle_unit)
            angles+=[[firstbond[0],firstbond[1],\
            secondbond[0],angle,firstbond[3],firstbond[4],secondbond[3],firstbond[5],\
            firstbond[6],secondbond[5]]]
   return angles
```

```
def dedupe_angles(angles):
   dupe=False
   newangles=[]
   for item in angles:
      dupe=False
      for anotheritem in newangles:
         if item[0]==anotheritem[2] and (item[2]==anotheritem[0]\
                            and item[1]==anotheritem[1]):
            dupe=True
            break
      if dupe==False:
         newangles+=[item]

   newerangles=[]
   dupe=False

   for item in newangles:
      dupe=False
      for anotheritem in newerangles:
         if item[2]==anotheritem[2] and (item[0]==anotheritem[1]\
                            and item[1]==anotheritem[0]):
            dupe_item=anotheritem
            dupe=True
            break

      if dupe==False:
         newerangles+=[item]
      elif dupe==True:
         if dupe_item[3]>item[3]:
            pass;
         else:
            newerangles[len(newerangles)-1]=item

   newestangles=[]
   dupe=False

   for item in newerangles:
      dupe=False
      for anotheritem in newestangles:
         if (sorted(item[0:3]) == sorted (anotheritem[0:3])):
            dupe_item=anotheritem
            dupe=True
            break

      if dupe==False:
         newestangles+=[item]
      elif dupe==True:
         if dupe_item[3]>item[3]:
            pass;
         else:
            newestangles[len(newestangles)-1]=item
   return newestangles

def finddihedrals(angles, bonds, angle_unit):
   dihedrals = []
   for item in angles:
      for anotheritem in bonds:
         if item[2] == anotheritem[0]:
            vec = [genvec(item[7], item[8])]
            vec += [genvec(item[8], item[9])]
            vec += [genvec(item[9], anotheritem[6])]
            dihedral = diheder(vec[0], vec[1], vec[2],angle_unit)
            dihedrals += [[item[0], item[1], item[2],anotheritem[1],
                 dihedral,item[4], item[5], item[6], anotheritem[4]]]

         if item[0] == anotheritem[0] and not item[1] == anotheritem[1]:
            vec = [genvec(anotheritem[6], item[7])]
            vec += [genvec(item[7], item[8])]
            vec += [genvec(item[8], item[9])]
            dihedral = diheder(vec[0], vec[1], vec[2],
                            angle_unit)
            dihedrals += [[anotheritem[1], item[0], item[1],
                        item[2], dihedral, anotheritem[4], item[4],
                             item[5], item[6]]]
   return dihedrals


def dedup_dihedrals(dihedrals):
   newdihedrals = []

   for item in dihedrals:
      dupe = False
      for anotheritem in newdihedrals:
         rev = anotheritem[0:4]
         rev.reverse()
         if item[0:4] == rev:
            dupe = True
      if not dupe:
         newdihedrals += [item]
   return newdihedrals


def print_bonds(bonds, func):
   constraints = ""
   funcconstr = '2'
   print '[ bonds ]'

   for item in bonds:
      if item[2] <= 0.098:
         constraints += (str(item[0])+'\t'+str(item[1])+'\t' +
                     funcconstr+'\t'+str("%1.3f" %
                        item[2])+'\t;'+str(item[3])+'\t'+str(item[4])+'\n')
      else:
         print str(item[0])+'\t'+str(item[1])+'\t'+func+'\t' +\
            str("%1.3f" % item[2])+'\t'+'50000.00' + \
            '\t;'+str(item[3])+'\t'+str(item[4])

   print '[ constraints ]'
   print constraints
   return 0
```

```
def print_angles(angles, func):
   print '[ angles ]'
   for item in angles:
      print str(item[0])+'\t'+str(item[1])+'\t'+str(item[2])+'\t' +\
         func+'\t'+str("%3.3f" % item[3])+'\t'+'50000.00'+'\t;'\
         + str(item[4])+'\t'+str(item[5])+'\t'+str(item[6])
   return 0


def print_dihedrals(dihedrals, func):
   print "[ dihedrals ]"
   force = '50000.00'
   mult = '2'
   for item in dihedrals:
      print str(item[0])+'\t'+str(item[1])+'\t'+str(item[2])+'\t' +\
         str(item[3]) +\
         '\t'+func+'\t'+str("%3.2f" % item[4])+'\t'+force+'\t'+mult +\
         '\t;'+str(item[5])+'\t'+str(item[6]) +\
         '\t'+str(item[7])+'\t'+str(item[8])
   return 0

if __name__ == "__main__":
   infile = sys.argv[1]
   rcutoff = float(sys.argv[2]) # in nm
   itemstoprint = int(sys.argv[3])

   angle_unit = 'deg' # {'rad'|'deg'}
   prevent_hhbonds = True # False is safer -- it prevents bonds between
   # atoms whose names start with H

   positions = readatoms(infile)

   bonds = makebonds(positions, rcutoff, prevent_hhbonds)
   bonds = dedupe_bonds(bonds)

   print_bonds(bonds, '6')

   angles = findangles(bonds, angle_unit)
   angles = dedupe_angles(angles)

   if itemstoprint >= 2:
      print_angles(angles, '1')

   dihedrals = finddihedrals(angles, bonds, angle_unit)
   dihedrals = dedup_dihedrals(dihedrals)
   if itemstoprint >= 3:
      print_dihedrals(dihedrals, '1')
```