

Xplor-NIH: An Introduction

Charles Schwieters

Center for Information Technology

National Institutes of Health

Bethesda, MD USA

outline

1. description, history, installation
2. Scripting Languages: XPLOR, Python, TCL
 - Introduction to Python
3. Overview of an Xplor-NIH Python script
4. Potential terms available from Python
5. IVM: dynamics and minimization in internal coordinates
6. Parallel determination of multiple structures
7. Water Refinement
8. VMD molecular graphics interface
9. Refinement against solution scattering data.
10. Ensemble refinement for a dynamical representation.
11. The PASD facility for automatic NOE assignment

goal of this document:

Xplor-NIH's Python interface will be introduced, described in enough detail such that scripts can be understood, and modified.

Major Contributors at NIH

Guillermo Bermejo John Kuszewski
Yaroslav Ryabov Robin Thottungal
Marius Clore Nico Tjandra

Support:

Andy Byrd, Yun-Xing Wang, Ad Bax

developed in the Imaging Sciences Laboratory, DCB, CIT, NIH

Many contributions from the community

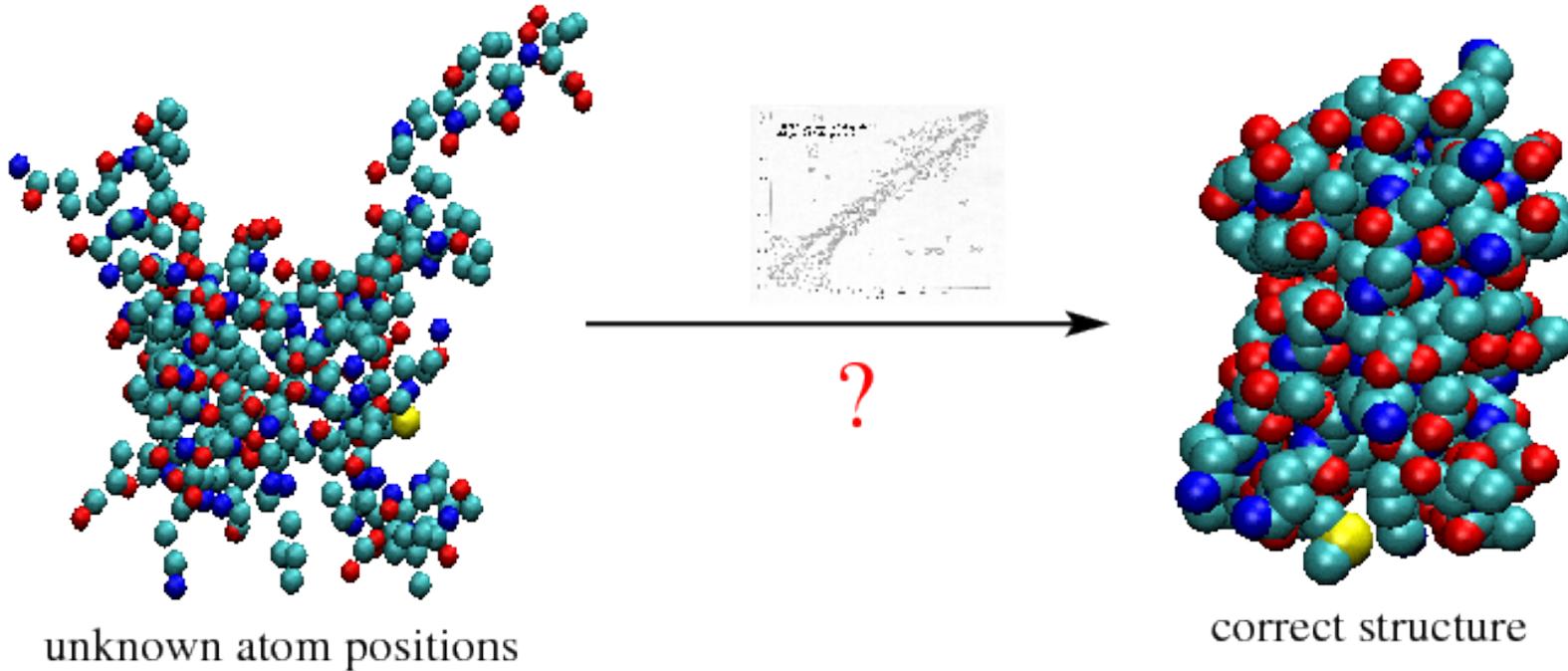


Center for
Information
Technology

Division of Computational Bioscience

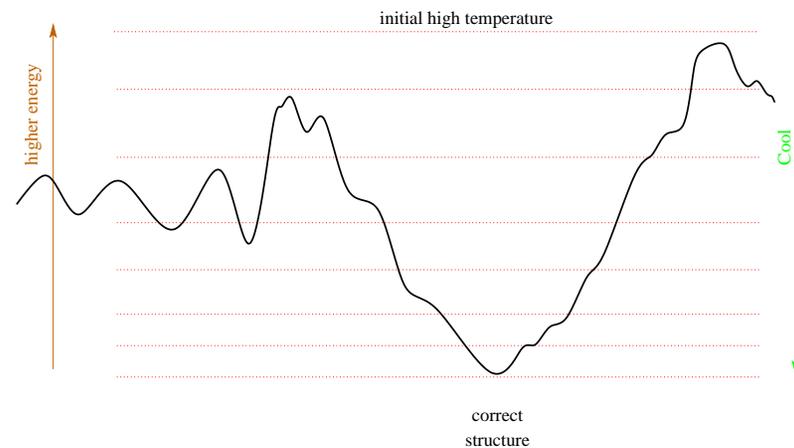


Overview of structure determination



Minimize energy: $V_{\text{tot}} = V_{\text{expt}} + V_{\text{covalent}} + V_{\text{knowledge}} + \dots$

- molecular dynamics to explore the energy surface.
- slowly decrease the temperature to find the global minimum.
- surface smoothed at high temperature



What is Xplor-NIH?

Biomolecular structure determination/manipulation

- Determine structure using minimization protocols based on molecular dynamics/simulated annealing, and other algorithms.
- Potential energy terms:
 - terms based on input from NMR (and other) experiments: NOE, dipolar coupling, chemical shift data, SAXS, SANS, fiber diffraction, etc.
 - other potential terms enforce reasonable covalent geometry (bonds and angles), and to prevent atomic overlap.
 - knowledge-based potential terms incorporate info from structure database - packing, torsion angles, etc.
- **includes:** program, topology, covalent parameters , potential energy parameters, databases for knowledge-based potentials, helper programs, example scripts, and high level protocols for structure helper scripts/programs determination and analysis.
- freely available for non-commercial work. Source code is available.
- For commercial use, please contact <mailto:Charles@Schwieters.org>.

Xplor-NIH Description

New contributions, additions are encouraged.

Source code of Xplor-NIH:

- original XPLOR Fortran source, with contributions from many groups.
- current work uses C++ for compute-intensive work.
- scripts and much code are written in **Python**, TCL scripting languages.
- SWIG used to “glue” scripting languages to C++.
- bazaar (bzd) repository of source code is available online.

Installation

1. download two files from `http://nmr.cit.nih.gov/xplor-nih/`
 - a) a -db file: *e.g.* `xplor-nih-2.42-db.tar.gz`
 - b) a platform-specific file: *e.g.* `xplor-nih-2.42-Linux_x86_64.tar.gz`
2. unpack these files where you wish them to live:

```
zcat xplor-nih-2.42-db.tar.gz | (cd /opt ; tar xf -)
zcat xplor-nih-2.42-Linux_x86_64.tar.gz | (cd /opt ; tar xf -)
```

3. perform initial configuration:

```
cd /opt/xplor-nih-2.42
./configure -symlinks /usr/local/bin
```

The optional `-symlinks` argument creates symbolic links in the specified directory for `xplor` and other commands. It is intended that you specify a directory in your `PATH`. For instance, if installing in your home directory, you might specify `-symlinks ~/bin`.

4. test the new installation:

```
bin/testDist
```

Scripting Languages- three choices

scripting language:

- flexible interpreted language
- used to input filenames, parameters, protocols
- flexible enough to program non compute-intensive logic
- relatively user-friendly

XPLOR language:

strong point:

atom selection language quite powerful.

weaknesses:

String, Math support problematic.

no support for subroutines: difficult to encapsulate functionality.

Parser is hand-coded in Fortran: difficult to update.

XPLOR reference manual:

<http://nmr.cit.nih.gov/xplor-nih/doc/current/xplor/>

NOTE: all old XPLOR 3.851 scripts should run unmodified with Xplor-NIH.

Language Examples - printing 1..10

XPLOR	Python 2	TCL
<pre>eval (\$i=1) while (\$i le 10) loop ploop display \$i eval (\$i=\$i+1) end loop ploop</pre>	<pre>for i in range(1,11): print i</pre>	<pre>for {set i 1} {\$i<11} {incr i} { puts \$i }</pre>
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10

General purpose scripting languages: Python and TCL

- excellent string support.
- languages have functions and modules: can be used to better encapsulate protocols (*e.g.* call a function to perform simulated annealing.)
- well known: these languages are **useful for other computing needs**: replacements for AWK, shell scripting, *etc.*
- contain extensive libraries with additional functionality (*e.g.* file processing, web access, GUI library, *etc.*).
- Facilitate interaction, tighter coupling with other tools.
 - NMRWish has a TCL interface.
 - pyMol has a Python interface.
 - VMD has TCL and Python interfaces.
 - Allow tight integration with CING structure validation suite (currently being implemented).

separate processing of input files (assignment tables) is unnecessary: can all be done using Xplor-NIH.

Introduction to Python

assignment and strings

```
a = 'a string' # <- pound char introduces a comment
a = "a string" # ' and " chars have same functionality
```

multiline strings - use three ' or " characters

```
a = '''a multiline
string'''
```

C-style string formatting - uses the % operator

```
s = "a float: %5.2f   an integer: %d" % (3.14159, 42)
print s
```

```
a float:  3.14   an integer: 42
```

raw strings - special characters are not translated

```
a = r'strange characters: \%~!' # introduced by an r
```

lists and tuples

```
l = [1,2,3]           #create a list
a = l[1]              #indexed from 0 (a = 2)
l[2] = 42             # l is now [1,2,42]
t = (1,2,3)           #create a tuple (read-only list)
a = t[1]              # a = 2
t[2] = 42             # ERROR!
```

Introduction to Python

calling functions

```
bigger = max(4,5) # max is a built-in function
```

defining functions - leading whitespace scoping

```
def sum(item1,item2,item3=0):  
    "return the sum of the arguments" # comment string  
    retVal = item1+item2+item3      # note indentation  
    return retVal  
  
print sum(42,1)                      #un-indented line: not in function
```

43

using keyword arguments - specify arguments using the argument name

```
print sum(item3=2,item1=37,item2=3) # argument order is not important
```

42

loops - the for statement

```
for cnt in range(0,3): # loop over the list [0,1,2]  
    cnt += 10  
    print cnt
```

10

11

12

Introduction to Python

Python is modular

most functions live in separate namespaces called modules

Loading modules - the import statement

```
import sys          #import module sys
sys.version         #return the Python version from the module sys

'2.7.5 (default, Aug 12 2013, 14:03:27) \n[GCC 4.0.2 20051125 (Red Hat
 4.0.2-8)]'
```

or:

```
from sys import version #import version variable into current scope
version                 #don't need to prepend sys.

'2.7.5 (default, Aug 12 2013, 14:03:27) \n[GCC 4.0.2 20051125 (Red Hat
 4.0.2-8)]'
```

Introduction to Python

In Python objects are everywhere.

Objects: associated functions called *methods*

```
file = open("filename")    #open is built-in function returning an object
contents = file.read()    #read is a method of this object
                           # returns a string containing file contents
```

```
dir(file)                  # list all methods of object file
```

```
['_class__', '__delattr__', '__doc__', '__getattr__',
 '__hash__', '__init__', '__iter__', '__new__', '__reduce__',
 '__repr__', '__setattr__', '__str__', 'close', 'closed', 'fileno',
 'flush', 'isatty', 'mode', 'name', 'read', 'readinto', 'readline',
 'readlines', 'seek', 'softspace', 'tell', 'truncate', 'write',
 'writelines', 'xreadlines']
```

Introduction to Python

A mapping type: Dictionaries

```
d={}
d['any'] = 4           #elements indexed like arrays
d['string'] = 5       # but the index can be (almost) any type
print d['string']
```

5

```
d.keys()              #return list of all index keys
d.values()            #return list of all indexed values
```

Tools for List Processing:

List Comprehensions - convert a list to another list

```
stringList=['1','2','3']
[ int(i) for i in stringList ]    # convert list of string to ints
```

[1, 2, 3]

List comprehension:

- expression within square brackets containing for, in and optionally if

```
[ 2*int(c) for c in ['3','2','1'] if c!='2' ]
```

[6, 2]

Introduction to Python

interactive help functionality: `dir()` is your friend!

```
import sys
dir(sys)      #lists names in module sys
dir()         # list names in current (global) namespace
dir(1)        # list of methods of an integer object
```

the help function

```
import ivm
help( ivm )      #help on the ivm module
help(open)       # help about the built-in function open
help(sys.exit)   # help about the exit function in the imported sys module
```

browse the Xplor-NIH python library using your web-browser on your local workstation:

```
% xplor -py -pydoc -g
```

Xplor-NIH Python module reference:

```
http://nmr.cit.nih.gov/xplor-nih/doc/current/python/ref/index.html
```

Linear Algebra Facilities in Python

Direct access to efficient C++ routines for matrix/vector manipulation. Includes Numerical Python-like operations.

```
from cdsMatrix import RMat, transpose, inverse
from cdsMatrix import svd, trace, det, eigen

m=RMat([[1,2],      #create a matrix object
        [3,4]])
print m

print m[0,1]      #element access
m[0,1]=3.14      #element assignment

print trace(m)    #matrix trace
print det(m)      #determinant
print transpose(m)#matrix transpose

print inverse(m)  #matrix inverse

print 0.5*m      # multiplication by scalar
print m+m,m-m    # matrix addition, subtraction
print m*m        # matrix multiplication
```

```
from cdsVector import CDSVector_double as vector
from cdsVector import norm

v=vector([1,2])   # vectors
print norm(v)    # vector norm
print 2*v,v+v,v-v # vector arithmetic
print m*v        # matrix multiplication

# 3-dimensional vectors
from vec3 import Vec3, cross, dot, unitVec
v = Vec3(1,2,3)
cross(Vec3(1,0,0),v) ; dot(Vec3(1,0,0),v)
unitVec(v)

# singular value decomposition
r= svd(m)
print r.u, r.vT, r.sigma

# eigenvalue decomposition
e= eigen(m)
print e[0].value()      #first eigenvalue
print list(e[0].vector()) #first eigenvector
```

Additional Mathematical Facilities

These modules are distributed with Xplor-NIH.

- `cminpack`: nonlinear least squares.
- `fft`: real and complex FFTs.
- `moremath`: special functions and constants.
- `spline`: 1-, 2-, and 3- dimensional cubic splines.
- `numpy`: Numeric Python library is distributed with Xplor-NIH.
- `matplotlib`: powerful plotting package.

Accessing Xplor-NIH's Python interpreter

from the command-line: use the `-py` flag:

```
% xplor -py
```

```

                                XPLOR-NIH version 2.42
C.D. Schwieters, J.J. Kuszewski,   Progr. NMR Spectr. 48, 47-62 (2006).
N. Tjandra, and G.M. Clore        J. Magn. Res., 160, 66-74 (2003).
http://nmr.cit.nih.gov/xplor-nih  based on X-PLOR 3.851 by A.T. Brunger
python>
```

or, the `pyXplor` executable - a bit quieter- and can be used as a complete replacement for the python command:

```
% pyXplor
```

```
python>
```

To run a script:

```
% xplor -py script.py
```

or, as an `extension` to an external Python interpreter:

```
% ( eval 'xplor -csh-env' ; python)
```

or

```
% xplor -sh -c python
```

```
Python 2.7.10 ...
```

```
>>> import xplorNIH
```

```
>>> execfile('script.py')
```

[extension use requires that Python version be consistent between the external interpreter and Xplor-NIH.]

using Python from XPLOR

accessing Python from XPLOR: **PYTHon** command

```
% xplor
```

```
                XPLOR-NIH version 2.42
```

```
C.D. Schwieters, J.J. Kuszewski,          Progr. NMR Spectr. 48, 47-62 (2006).
```

```
N. Tjandra, and G.M. Clore              J. Magn. Res., 160, 66-74 (2003).
```

```
http://nmr.cit.nih.gov/xplor-nih        based on X-PLOR 3.851 by A.T. Brunger
```

```
User: schwitrs      on: khaki      (x86/Linux      ) at: 7-Dec-06 12:37:40
```

```
X-PLOR>python          !NOTE: can't be used inside an XPLOR loop!
```

```
python> print 'hello world!'
```

```
hello world!
```

```
python> python_end()
```

```
X-PLOR>
```

for a single line: **CPYThon** command

```
X-PLOR>cpython "print 'hello world!'"    !can be used in a loop
```

```
hello world!
```

```
X-PLOR>
```

using XPLOR, TCL from Python

to call the XPLOR interpreter from Python

```
xplor.command('''struct @1gb1.psf end  
                coor @1gb1.pdb''')
```

xplor is a built-in module - no need to import it for simple scripts.

to call the TCL interpreter from Python

```
from tclInterp import TCLInterp          #import function  
tcl = TCLInterp()                        #create TCLInterp object  
tcl.command('xplorSim setRandomSeed 778') #initialize random seed
```

Structure Calculation Overview: Script Skeleton

```
import protocol
protocol.loadPDB("model.pdb")      #initialize coordinates

coolParams=[] # a list which specifies potential smoothing
# set up potential terms from NMR experiments, covalent geometry,
# and knowledge-based terms

# initialize coolParams for annealing protocol for each energy term

from ivm import IVM #configure which degrees of freedom to optimize
dyn = IVM()

from simulationTools import AnnealIVM
coolLoop=AnnealIVM(dyn,...)      #create simulated annealing object, specify temperature schedule

def calcOneStructure( structData ):
    """ a function to calculate a single structure """
    # [ randomize velocities ]
    # [ perform high temp dynamics ]
    dyn.run()
    # [ cooling loop ]
    coolLoop.run()
    # [ final minimization ]
    dyn.run()

from simulationTools import StructureLoop
StructureLoop(numStructures=100,          #calculate 100 structures
              structLoopAction=calcOneStructure, #using this function
              doWriteStructures=True,        #then write to pdb file
              pdbTemplate='SCRIPT_STRUCTURE.sa' #using this template
              ).run()                       # a .viols file also written
```

StructureLoop handles parallel structure calculation, and optional structural statistics calculation and regularized mean structure calculation.

Loading and Generating Coordinates

PSF file - contains atomic connectivity, mass and covalent geometry information.

This information must be present before coordinates can be loaded.

generate via external helper scripts

1. seq2psf - generate a psf file from primary sequence
 % seq2psf file.seq
2. pdb2psf - generate a psf file from a pdb file
 % pdb2psf file.pdb
3. More involved: most modified and nonstandard residues and small molecules.

within the Python scripting interface (in the `protocol` module)

- `protocol.initStruct` - load pregenerated .psf file. Not necessary for standard residues.
- `protocol.initCoords` - read pdb file using the current PSF. It also reads mmCIF files.
- `protocol.loadPDB` - read pdb or mmCIF and generate psf info on the fly. Also fixes-up input coordinates (naming, symmetric sidechains, disulfide bonds, BIOMT records). It can also delete atoms whose coordinates are not known.

To write out a PDB file use `protocol.writePDB("file.pdb")`.

Loading and Generating Coordinates - details

A **Simulation** object contains atom name, position, mass, *etc* and bonding information.

The default Simulation is `xplor.simulation`

A completely separate PSF can be loaded by creating a new XplorSimulation:

```
from xplorSimulation import XplorSimulation
new_xsim = XplorSimulation()
import protocol
protocol.initState('other.psf', simulation=new_xsim)
```

Each XplorSimulation has a separate XPLOR process associated with it.

Initial atomic coordinate values: $(x,y,z) = (9999.999, 9999.999, 9999.999)$ these are the values if coordinates are not initialized.

To delete these atoms:

```
xplor.simulation.deleteAtoms("not known")
```

To add atomic coordinates if some are not defined:

```
from protocol import addUnknownAtoms
addUnknownAtoms()
```

These coordinates will have proper covalent geometry.

To correct covalent geometry (bonds, angles and impropers):

```
from protocol import fixupCovalentGeom
fixupCovalentGeom('resid 30:50') # this may cause significant changes in
                                # the selected atomic positions
```

Topology and Parameters

Topology specifies how residues and (small) molecules are connected.

Parameters specify force constants, bond lengths, atomic radii, *etc.*

For standard proteins and nucleic acids, no special action required.

For modified or artificial residues or small molecule ligands, may need to generate new topology and parameters:

- PRODRG <http://davapc1.bioch.dundee.ac.uk/cgi-bin/prodrng>
- ACPYPE <http://webapps.ccpn.ac.uk/acpype/>

For water refinement (see Water Refinement below), alternate topology and parameters are required. Please see the examples.

```
Topology Entry for Alanine
residue ALA
group
  atom N  type=NH1 charge=-0.36 end
  atom HN type=H   charge= 0.26 end
group
  atom CA type=CT  charge= 0.00 end
  atom HA type=HA  charge= 0.10 end
group
  atom CB type=CT  charge=-0.30 end
  atom HB1 type=HA charge= 0.10 end
  atom HB2 type=HA charge= 0.10 end
  atom HB3 type=HA charge= 0.10 end
group
  atom C  type=C   charge= 0.48 end
  atom O  type=O   charge=-0.48 end

bond N  HN
bond N  CA  bond CA HA
bond CA CB  bond CB HB1  bond CB HB2  bond CB HB3
bond CA C
bond C  O

improper HA  N  C  CB  !stereo CA
improper HB1 HB2 CA HB3  !stereo CB

end
```

Atom Selections in Python

We use a subset of the XPLOR atom selection language, described here.

```
from atomSel import AtomSel
sel = AtomSel('''resid 22:30 and
                (name CA or name C or name N)''')
print sel.string()           #AtomSel objs remember their selection string
```

```
resid 22:30 and
                (name CA or name C or name N)
```

AtomSel objects can be used as lists of Atom objects

```
print len(sel)              # prints number of atoms in sel
for atom in sel:           # iterate through atoms in sel
    print atom.string(), atom.pos() # prints atom string, and its position.
```

AtomSel objects can be *reevaluated*:

```
xplor.simulation.deleteAtoms("resid 1:2")
sel.reevaluate()
print len(sel)              # prints the correct number of atoms
```

Atomwise AtomSel operations:

```
from atomSel import intersection, union, notSelection
sel2 = AtomSel('name C')
intersection(sel,sel2); union (sel,sel2); notSelection(sel)
```

Named atom selections:

```
import atomSelLang
atomSelLang.setNamedSelection(xplor.simulation,"nTerminus",
                              AtomSel("resid 1:140").indices())
atoms = AtomSel('recall nTerminus')
```

Order of atoms in an AtomSel is the order in the PSF.

File Formats: mmCIF and NEF

PDB replacement format for atomic coordinates: mmCIF

```
loop_
_atom_site.group_PDB
_atom_site.id
_atom_site.type_symbol
_atom_site.label_atom_id
_atom_site.label_alt_id
_atom_site.label_comp_id
_atom_site.label_asym_id
_atom_site.label_entity_id
_atom_site.label_seq_id
_atom_site.pdbx_PDB_ins_code
_atom_site.Cartn_x
_atom_site.Cartn_y
_atom_site.Cartn_z
_atom_site.occupancy
_atom_site.B_iso_or_equiv
_atom_site.Cartn_x_esd
_atom_site.Cartn_y_esd
_atom_site.Cartn_z_esd
_atom_site.occupancy_esd
_atom_site.B_iso_or_equiv_esd
_atom_site.pdbx_formal_charge
_atom_site.auth_seq_id
_atom_site.auth_comp_id
_atom_site.auth_asym_id
_atom_site.auth_atom_id
_atom_site.pdbx_PDB_model_num
ATOM 1      N  N      . MET A 1 1  ? -14.136 1.321   3.616   1.00 0.93 ? ? ? ? ? ? 1  MET A N      1
ATOM 2      C  CA     . MET A 1 1  ? -13.451 0.063   4.032   1.00 0.36 ? ? ? ? ? ? 1  MET A CA     1
ATOM 3      C  C      . MET A 1 1  ? -11.981 0.360   4.336   1.00 0.36 ? ? ? ? ? ? 1  MET A C      1
ATOM 4      O  O      . MET A 1 1  ? -11.557 1.499   4.315   1.00 0.64 ? ? ? ? ? ? 1  MET A O      1
ATOM 5      C  CB     . MET A 1 1  ? -13.532 -0.987   2.924   1.00 1.26 ? ? ? ? ? ? 1  MET A CB     1
ATOM 6      C  CG     . MET A 1 1  ? -14.934 -0.967   2.313   1.00 1.15 ? ? ? ? ? ? 1  MET A CG     1
ATOM 7      S  SD     . MET A 1 1  ? -15.215 0.142   0.911   1.00 1.64 ? ? ? ? ? ? 1  MET A SD     1
```

NMR Exchange Format (NEF)

Contains

- sequence, molecular identity
- chemical shifts
- NOE peak lists
- Derived Restraints
 - distance
 - dihedral
 - RDC

```
save_nef_chemical_shift_list_bmr21.str
  _nef_chemical_shift_list.sf_category          nef_chemical_shift_list
  _nef_chemical_shift_list.sf_framecode        nef_chemical_shift_list_bmr21.str
  _nef_chemical_shift_list.atom_chemical_shift_units ppm
```

```
loop_
  _nef_chemical_shift.chain_code
  _nef_chemical_shift.sequence_code
  _nef_chemical_shift.residue_type
  _nef_chemical_shift.atom_name
  _nef_chemical_shift.value
  _nef_chemical_shift.value_uncertainty
```

```
A  10  HIS  C      175.19  0.4
A  10  HIS  CA     56.002  0.4
A  10  HIS  CB     30.634  0.4
A  10  HIS  CD2    119.578  0.4
A  10  HIS  HA      4.687  0.02
A  10  HIS  HBX     3.106  0.02
A  10  HIS  HBY     3.201  0.02
A  10  HIS  HD2     7.067  0.02
```

Using potential terms in Python

Available potential terms in the following modules:

- noePot - NOE distance restraints
- rdcPot - dipolar coupling
- sardcPot - RDCs in steric alignment media - J.-r. Huang and S. Grzesiek
- rdcCorrPot - fit RDCs without alignment tensor - C. Camilloni and M. Vendruscolo
- csaPot - Chemical Shift Anisotropy
- cstMagPot - refine against chemical shift tensor magnitudes and orientations
- jCoupPot - 3J -coupling
- prePot - Paramagnetic relaxation enhancement
- diffPot - refine against rotational diffusion tensor
- relaxRatioPot - refine directly against NMR relaxation data
- solnScatPot - potential for solution X-ray and neutron scattering
- planeDistPot - distance between atoms and plane
- gyrPot - pseudopotential enforcing correct protein density
- residueAffPot - contact potential for hydrophobic attraction/repulsion
- xplorPot - use XPLOR potential terms
- posSymmPot - restrain atomic positions relative to those in a similar structure
- potList - a collection of potential terms in a list-like object.

All potential objects have the following methods:

- instanceName() - name given when created
- potName() - name associated w/ potential type, e.g. "RDCPot"
- scale() - scale factor or weight (force constant). Set with setScale(val).
- calcEnergy() - calculate and return term's (scaled) energy

Distance Restraints - the NOE potential term

Most commonly used effective NOE distance R is computed:

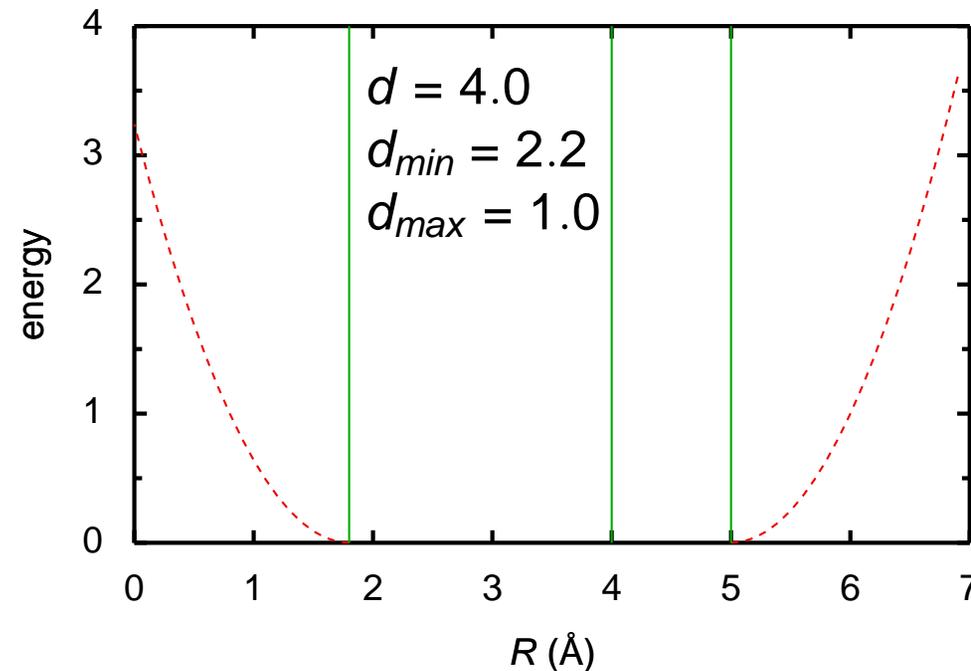
$$R = \left(\sum_{ij} |q_i - q_j|^{-6} \right)^{-1/6}$$

“sum averaging” - usually effectively picks out the shortest interatomic distance.

Potential Energy: piecewise quadratic

“Square potential”

$$V(R) = \begin{cases} (R - d - d_{max})^2 & \text{for } R > d + d_{max} \\ (R - d + d_{min})^2 & \text{for } R < d - d_{min} \\ 0 & \text{in between} \end{cases}$$



XPLOR assignment statement

```
assign (resid 2 and name HA ) (resid 19 and name HB# ) 4.0 2.2 1.0 !#  
or (resid 6 and name HA ) (resid 27 and name HB# ) !ambiguity
```

NOE potential term

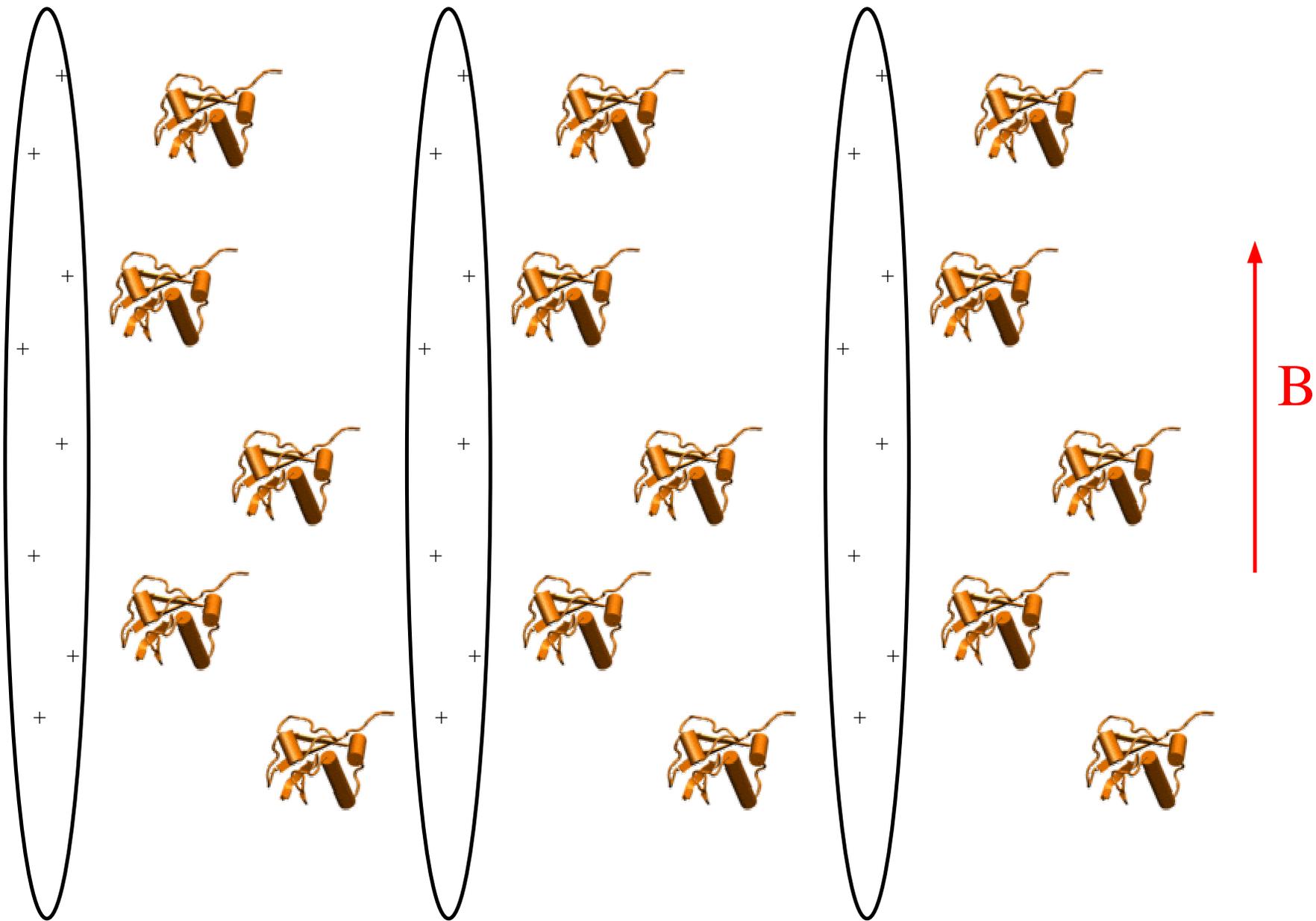
creating an NOEPot object:

```
from noePotTools import create_NOEPot
noe = create_NOEPot("noe", "noe_all.tbl")
#noe.setPotType("soft")    #uncomment if bad NOE restraints may be present
```

use:

```
print noe.instanceName()    # prints 'noe'
print noe.potName()         # prints 'NOEPot'
noe.setAveExp(5)            # change exponent for 1/r6 sum
                             # a reduced value reduces barriers
print noe.rms()             # the rmsd from the allowed distance range
noe.setThreshold( 0.1 )     # violation threshold
print noe.violations()      # number of violations
print noe.showViolations()
```

Residual Dipolar Couplings



partial alignment in aligning medium

Dipolar Coupling potential

Provides orientational information relative to axis fixed in molecule frame.

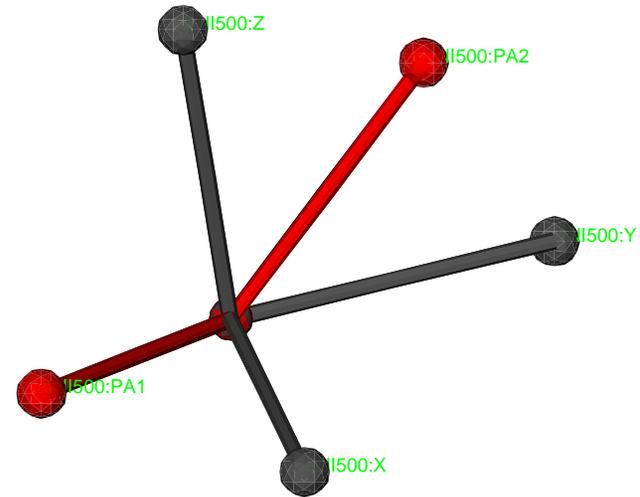
$$D^{AB} = D_a[(3u_z^2 - 1) + \frac{3}{2}R(u_x^2 - u_y^2)] ,$$

u_x, u_y, u_z - projection of bond vector onto axes of an alignment tensor. D_a, R - measure of axial and rhombic tensor components.

rdcPot - used for RDCs in solution and ssNMR dipolar couplings

- tensor orientation encoded in four axis atoms
- allows D_a, R to vary: values encoded using extra atoms.
- reads both SANI and DIPO XPLOR assignment tables.
- allows multiple assignments for bond-vector atoms - for averaging.
- allows ignoring sign of D_a (optional)
- can (optionally) include distance dependence:
 $D_a \propto 1/r^3$.
- tensor values can be computed using SVD.

→can also be used for paramagnetic pseudocontact shifts.



How to use the rdcPot potential

```
from varTensorTools import create_VarTensor, calcTensor, calcTensorOrientation
ptensor = create_VarTensor('phage') #create a tensor object

ptensor.setDa(7.8) #set initial tensor Da, rhombicity
ptensor.setRh(0.3)
ptensor.setFreedom('varyDa, varyRh') #allow Da, Rh to vary

from rdcPotTools import create_RDCPot
rdcNH = create_RDCPot("NH",oTensor=ptensor,file='NH.tbl')

calcTensor(ptensor) #calc tensor parameters from current structure
#using SVD

calcTensorOrientation(ptensor) #calc tensor orientation (with fixed Da, Rh)
#from current structure using SVD
```

NOTE: no need to introduce psf files or coordinates for axis/parameter atoms- this is automatic.

analysis, accessing potential values:

```
print rdcNH.rms(), rdcNH.violations() # calculates and prints rms, violations
print ptensor.Da(), ptensor.Rh() # prints these tensor quantities
rdcNH.setThreshold(0) # violation threshold
print rdcNH.showViolations() # print out list of violated terms
from rdcPotTools import Rfactor
print Rfactor(rdcNH) # calculate and print a quality factor
```

RDCPot: additional details

using multiple media:

```
btensor=create_VarTensor('bicelle')
rdcNH_2 = create_RDCPot("NH_2",tensor=btensor,file='NH_2.tbl')
#[ set initial tensor parameters ]
btensor.setFreedom('fixAxisTo phage') #orientation same as phage
                                         #Da, Rh vary
```

multiple expts. single medium:

```
rdcCAHA = create_RDCPot("CAHA",oTensor=ptensor,file='CAHA.tbl')
```

rdcCAHA is a new potential term using the same alignment tensor as rdcNH.

Normally, experiments are normalized to NH Da values.

```
from rdcPotTools import scale_toNH
scale_toNH(rdcCAHA,'CAHA')    #rescales RDC prefactor relative to NH
                              # includes gyromagnetic ratios and
                              # bond lengths
```

Sign convention: the default is to consider the ^{15}N gyromagnetic ratio to be positive, *i.e.* the sign of NH experiments is flipped in the input tables. If you do not follow this convention, place the following at the *beginning* of your script:

```
from rdcPotTools import correctGyromagneticSigns
correctGyromagneticSigns()
```

Scaling convention: scale factor of non-NH terms is determined using the experimental error relative to the NH term:

```
rdcCAHA.setScale( (5/2)**2 )
# inverse error      ^^^ in expt. measurement relative to that for NH
```

Note: the square well potential is only used for nonbonded (e.g. H-H) experiments.

RDCs in Steric Alignment Media

When alignment is due solely to molecular shape.

```
from sardcPotTools import create_SARDCPot
sardc = create_SARDCPot("saRDC", "NH.tbl")
```

J.-R. Huang and S. Grzesiek, "Ensemble calculations of unstructured proteins constrained by RDC and PRE data: a case study of urea-denatured ubiquitin," *J. Am. Chem. Soc.* **132**, 694-705 (2010).

- Important for ensemble calculations where RDCPot leads to underdetermined alignment tensors.
- Input tables are the same format used in RDCPot.

One can extract a traditional Xplor-NIH alignment tensor:

```
from varTensorTools import saupeToVarTensor
from sardcPotTools import saupeMatrix
```

```
#generate a VarTensor representation of the SARDC alignment tensor
medium = saupeToVarTensor( saupeMatrix(sardc), dmax )
```

```
print medium.Rh() #print rhombicity
```

Chemical Shift Anisotropy potential

Provides additional orientational information from the full chemical shift tensor from measurements in an aligning medium.

$$\Delta\delta = \sum_{i,j} A_i \sigma_j \cos^2(\theta_{i,j})$$

A_i - a principal moment of the alignment tensor

σ_j - a principal moment of the CSA tensor

$\theta_{i,j}$ - angle between the i^{th} orientation tensor principal axis and the j^{th} CSA tensor principal axis.

How to use the csaPot potential

```
from csaPotTools import create_CSAPot
```

```
csaP = create_CSAPot(name,oTensor=tensor,file='csaP.tbl')
```

```
csaP.setDaScale( val )      # s.t. can be used with RDC alignment tensor
```

```
csaP.setScale( forceConstant )
```

```
calcTensor(tensor)         #use if the structure is approximately correct
```

NOTE: `create_CSAPot` uses built-in values for the chemical shift tensor. Alternate values can be specified by modifying `csaPotTools.csaData`.

→can be used with ssNMR CSA or chemical shift tensor data.

J-coupling potential

Karplus relationship

$${}^3J = A \cos^2(\theta + \theta^*) + B \cos(\theta + \theta^*) + C,$$

θ is a torsion angle, defined by four atoms.

A , B , C and θ^* are set using the COEF statement in the j-coupling assignment table (or using object methods).

Use in Python

```
from jCoupPotTools import create_JCoupPot
# set Karplus parameters while creating the potential term.
jCoup = create_JCoupPot("hnha", "jna_coup.tbl",
                        A=15.3, B=-6.1, C=1.6, phase=0)
```

analysis:

```
print Jhnha.rms()
print Jhnha.violations()
print Jhnha.showViolations()
```

Paramagnetic Relaxation Enhancement

$$\Gamma = S_{AB}(\tau_c)r_{AB}^{-6},$$

r_{AB} - distance between paramagnetic center and amide proton.

$S_{AB}(\tau_c)$ - function of correlation time τ_c .

```
from prePotTools import create_PREPot
pre = create_PREPot("pre", "file.tbl",
                    eSpinQuantumNumber=2.5,
                    freq=500,             # Larmor frequency in MHz
                    tauc=3.0,            # correlation time in ns
                    fixTau=True)
potList.append(pre)
```

- uses modified Solomon-Bloembergen Eq. which can account for tag motion and multiple tag conformations.
- can simultaneously determine correlation time.

Example in `eginput/pre/refine/newRefine.py`

Iwahara, *et. al.* JACS 126, 5879 (2004).

[in old XPLOR interface: PMAG term - Donaldson, *et. al.* JACS 123, 9843 (2001).]

Solvent Paramagnetic Relaxation Enhancement data

Empirical relationship:

$$\Gamma_{\text{sPRE}} \approx AS_{\text{Acc}} + B$$

with effective surface area:

$$S_{\text{Acc}} \approx \left(\sum_i r_i^{-2} \right)^{-1}$$

r_i is distance of amide proton in question to a heavy atom, and the sum is over all heavy atoms.

```
from nbTargetPotTools import create_NBTargetPot, calibrate
psol = create_NBTargetPot('psol', 'file.tbl', restraintFormat='xplor')
#psol.setPotType("correlation")
calibrate(psol)      # determine A and B by fit to experiment
potList.append(psol)
```

also used to describe solvent NOE.

Wang, *et. al.* J. Magn. Res. 221, 76 (2012).

Another, more rigorous term available, not yet published.

Use of Relaxation Data in Structure Calculation

Yaroslav Ryabov

Ratio of transverse to longitudinal relaxation rates: $\rho = R_2/R_1$ contains information on bond vector orientation relative to a diffusion tensor.

The diffusion tensor can be computed from atomic coordinates.

Thus: relaxation data can be used to obtain bond vector and overall shape information.

```
# read in relaxation data from file relax.dat
# data format specified by tablePattern
from diffPotTools import readInRelaxData, mergeRelaxData, make_ratio
relax_data = readInRelaxData("relax.dat",
                             pattern=tablePattern,
                             verbose=True)

from diffPotTools import make_ratio
for item in relax_data: make_ratio(item)

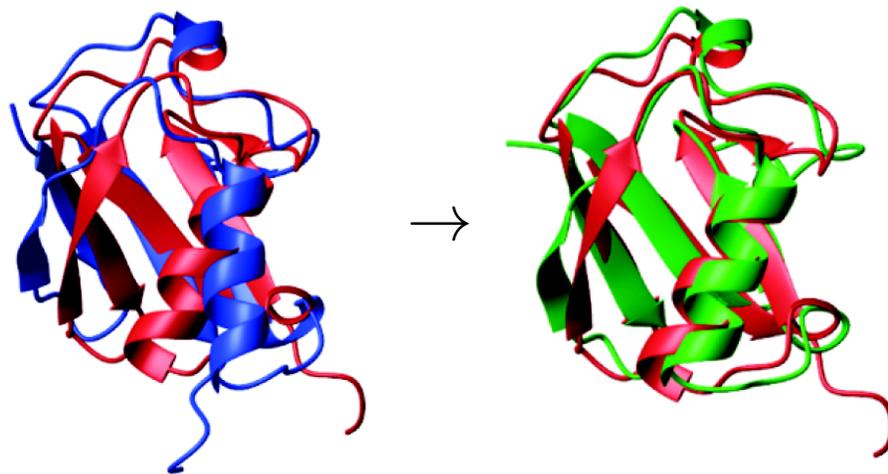
from relaxRatioPotTools import create_RelaxRatioPot
pot =create_RelaxRatioPot('relax',
                           data_in = relax_data,
                           freq = freq,           #spectrometer freq
                           temperature =temperature, # expt. temp.
                           )
```

Use of Relaxation Data in Structure Calculation

Used for docking



Improves structures with sparse restraints:



- temperature is an approximate fit parameter and should usually be optimized (facilities included).
- outliers are determined automatically, and updated regularly during a calculation

References

docking:

Y. Ryabov, G.M. Clore, C.D. Schwieters, *J. Am. Chem. Soc.* **132**, 5987-5989 (2010).

single domain:

Y. Ryabov, C.D. Schwieters, and G.M. Clore, *J. Am. Chem. Soc.* **133**, 6154–6157 (2011).

Gyration Volume potential - a pseudopotential

NOE distance restraints: approximate, loose.

Result: determined structures are too loosely packed.

But: Proteins pack to a constant density of $1.43 \pm 0.03 \text{ g cm}^{-3}$

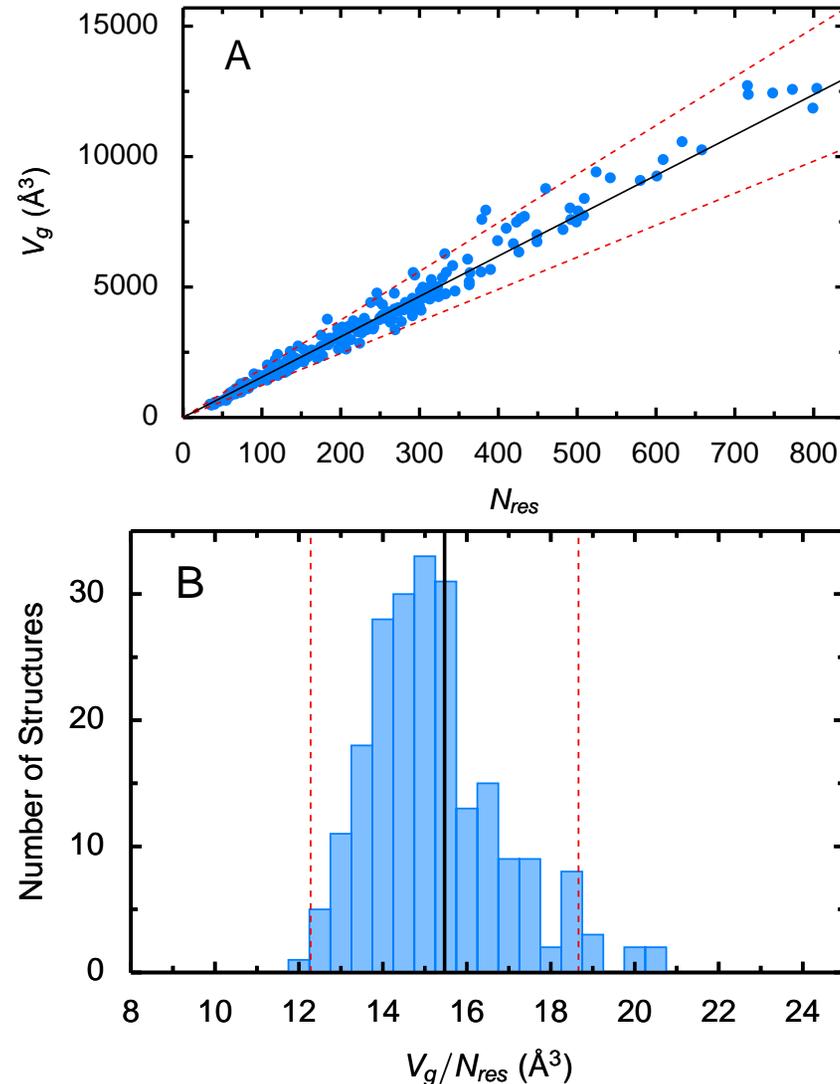
Approximate protein shape as ellipsoidal: gyration tensor:

$$G = \frac{1}{N} \sum_{i=1}^N \Delta q_i \otimes \Delta q_i,$$

gyration volume $V_g \equiv 4/3\pi \sqrt{|G|}$

Predict

$$V_g \approx V_g^{res} N_{res},$$



The v_g potential

$$E_{gyr} = w_{gyr} \left(w_{gyr}^{(1)} E_p(V_g - V_g^{res}; 0) + w_{gyr}^{(2)} E_p(V_g - V_g^{res}; \Delta V_g) \right) \quad E_p(x, \Delta x) = \begin{cases} (x - \Delta x)^2 & \text{for } x > \Delta x \\ (x + \Delta x)^2 & \text{for } x < -\Delta x \\ 0 & \text{otherwise} \end{cases}$$

Example of use of this term:

```
from gyrPotTools import create_GyrPot
gyr = create_GyrPot('Vgyr', 'not rename ANI')
potList.append(gyr)
```

Reference: C.D. Schwieters and G.M. Clore, *J. Phys. Chem. B* **112**, 6070-6073 (2008).

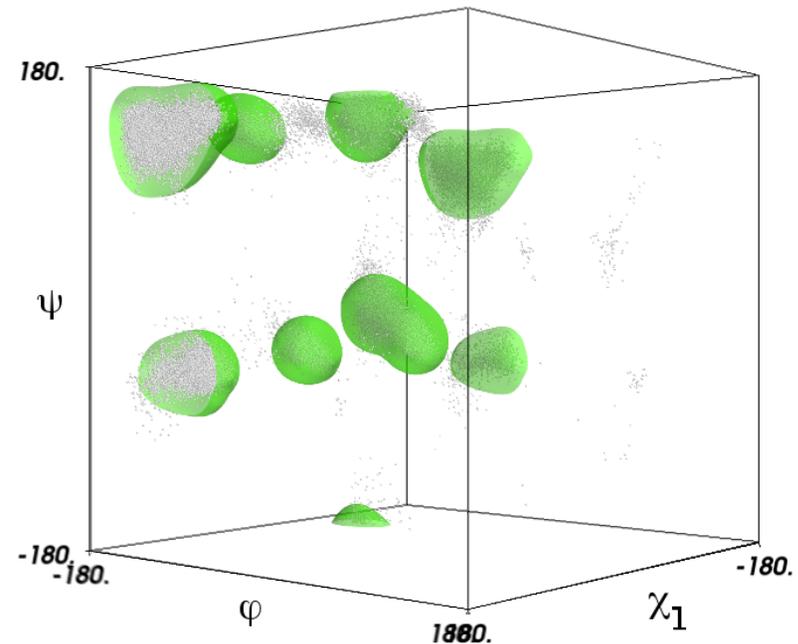
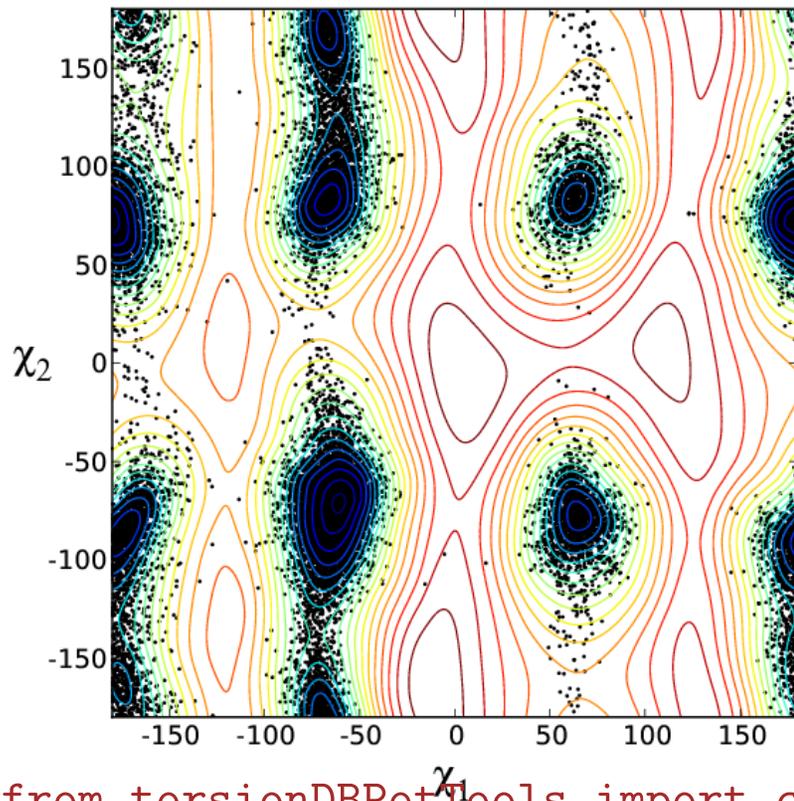
The TorsionDB potential

G. Bermejo *et al* - Protein Sci. **21**, 1824, (2012); Structure, 24, 806 (2016).

Improved dihedral angle potential of mean force based on observed protein structures.

Histidine

Valine



```
from torsionDBPotTools import create_TorsionDBPot
```

```
torsionDB=create_TorsionDBPot('torsionDB',  
                             system="protein", #rna and dna also valid  
                             selection='not recall nTerminus')  
potList.append( torsionDB )  
rampedParams.append( MultRamp(.002,2,"torsionDB.setScale(VALUE)") )
```

using XPLOR potentials

The XPLOR non-bonded potential

```
import protocol
from xplorPot import XplorPot
protocol.initNBond(repel=1.2)      #specify nonbonded parameters
vdw = XplorPot('VDW')

print vdw.violations()           #print number of overlapping atom pairs
print vdw.calcEnergy()           #term's energy
print vdw.potName()              # 'XplorPot'
print vdw.instanceName()        # 'VDW'
```

all other access/analysis done from XPLOR interface.

All parameters for the nonbonded term are listed in the XPLOR manual.

Dihedral Restraints:

given a restraint table generated by *e.g.* TALOS-N:

```
import protocol
from xplorPot import XplorPot
protocol.initDihedrals("dihedrals.tbl")    #dihedral restraint table
dihe = XplorPot('CDIH')
```

The XPLOR RAMA (torsion angle database) potential

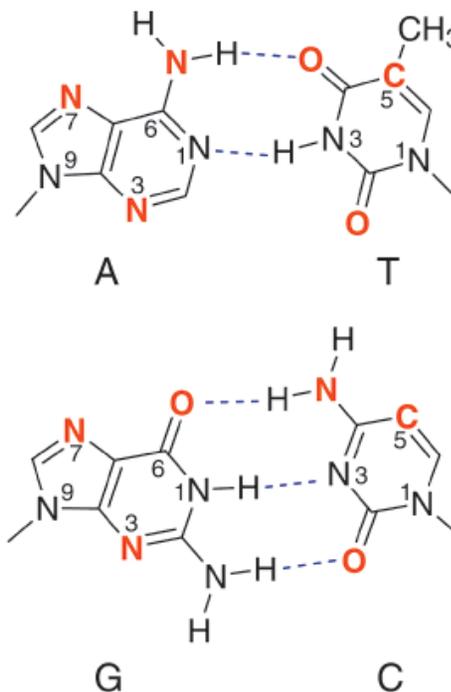
deprecated protein and nucleic acid dihedral angle potential of mean force

```
import protocol
from xplorPot import XplorPot
protocol.initRamaDatabase()
potList.append( XplorPot('RAMA') )
```

For nucleic acids, the XPLOR
ORIE potential: database-derived
basepair packing (translation and
orientation)

References: J. Kuszewski *et al* , J. Am. Chem.
Soc. **123**, 3903-3918 (2001); Clore &
Kuszewski, J. Am. Chem. Soc. **125**,
1518-1525 (2003).

→Important for proper
base-stacking separation.



```
xplor.command("@dna_positional.setup") #some external setup necessary
potList.append( XplorPot("ORIE") )
rampedParams.append( StaticRamp("potList['ORIE'].setScale(0.2)") )
```

Other commonly used XPLOR terms: BOND, ANGL, IMPR, HBDB, COLLapse.

using XPLOR potentials

Example using a Radius of Gyration (COLLapse) potential

```
import protocol
from xplorPot import XplorPot

protocol.initCollapse('resid 3:72') #helper setup function
                                     #specify globular portion
                                     # GyrPot (gyration volume) can be
                                     # used for non-globular domains

rGyr = XplorPot('COLL')
xplor.command('collapse scale 0.1 end') #manipulate in XPLOR interface
```

default target= $2.2 * N_{res}^{0.38} - 1$
(empirical relationship for compact globular proteins)

```
accessing associated values
print rGyr.calcEnergy()           #term's energy
print rGyr.potName()              # 'XplorPot'
print rGyr.instanceName()         # 'COLL'
```

all other access/analysis done from XPLOR interface.

using XPLOR potentials

other terms have no Python helpers, and must be configured via the XPLOR interface.

XRay Diffraction

```
xplor.command(r'''  
xref  
.  
.  
end  
''')  
potList.append( XplorPot("xref") )
```

Fiber XRay Diffraction

```
xplor.command(r'''  
fiber  
.  
.  
end  
''')  
potList.append( XplorPot("xref") )
```

References: R.C. Denny *et al* , Fibre Diffr. Rev **6**, 30-33 (1997) ; H. Wang and G. Stubbs, Acta Cryst **A49**, 504-513 (1993).

Collections of potentials - PotList

potential term which is a collection of potentials:

```
from potList import PotList
pots = PotList()
pots.append(noe); pots.append(Jhnha); pots.append(gyr)
pots.calcEnergy() # total energy
```

nested PotLists:

```
rdcs = PotList('rdcs') #convenient to collect like terms
rdcs.append( rdcNH ); rdcs.append( rdcNH_2 )
rdcs.setScale( 0.5) #set overall scale factor
pots.append( rdcs )
for pot in pots: #pots looks like a Python list
    print pot.instanceName()
```

```
noe
hnha
COLL
rdcs
```

Implementing a new potential term - in Python

```
from pyPot import PyPot ; from vec3 import norm, Vec3
class BondPot(PyPot):
    ''' example class to evaluate energy, derivs of a single bond'''
    def __init__(self,name,atom1,atom2,length,forcec=1):
        ''' constructor - force constant is optional.'''
        PyPot.__init__(self,name) #first call base class constructor
        self.a1 = atom1 ; self.a2 = atom2
        self.length = length; self.forcec = forcec
        return
    def calcEnergy(self):
        self.q1 = self.a1.pos() ; self.q2 = self.a2.pos()
        self.dist = norm(self.q1-self.q2)
        return 0.5 * self.scale() * self.forcec * (self.dist-self.length)**2
    def calcEnergyAndDerivList(self,derivs):
        energy = self.calcEnergy()
        deriv1 = Vec3(map(lambda x,y:x*y,
            [self.forcec * (self.dist-self.length) / self.dist]*3 ,
            ( self.q1[0]-self.q2[0],
              self.q1[1]-self.q2[1],
              self.q1[2]-self.q2[2] )))
        derivs[self.a1] += self.scale() *deriv1
        derivs[self.a2] -= self.scale() * deriv1
        return energy
    pass
```

to use:

```
p = BondPot('bond',AtomSel('resid 1 and name C')[0],
            AtomSel('resid 1 and name O')[0], length=1.5)
```

The IVM (internal variable module)

Used for dynamics and minimization

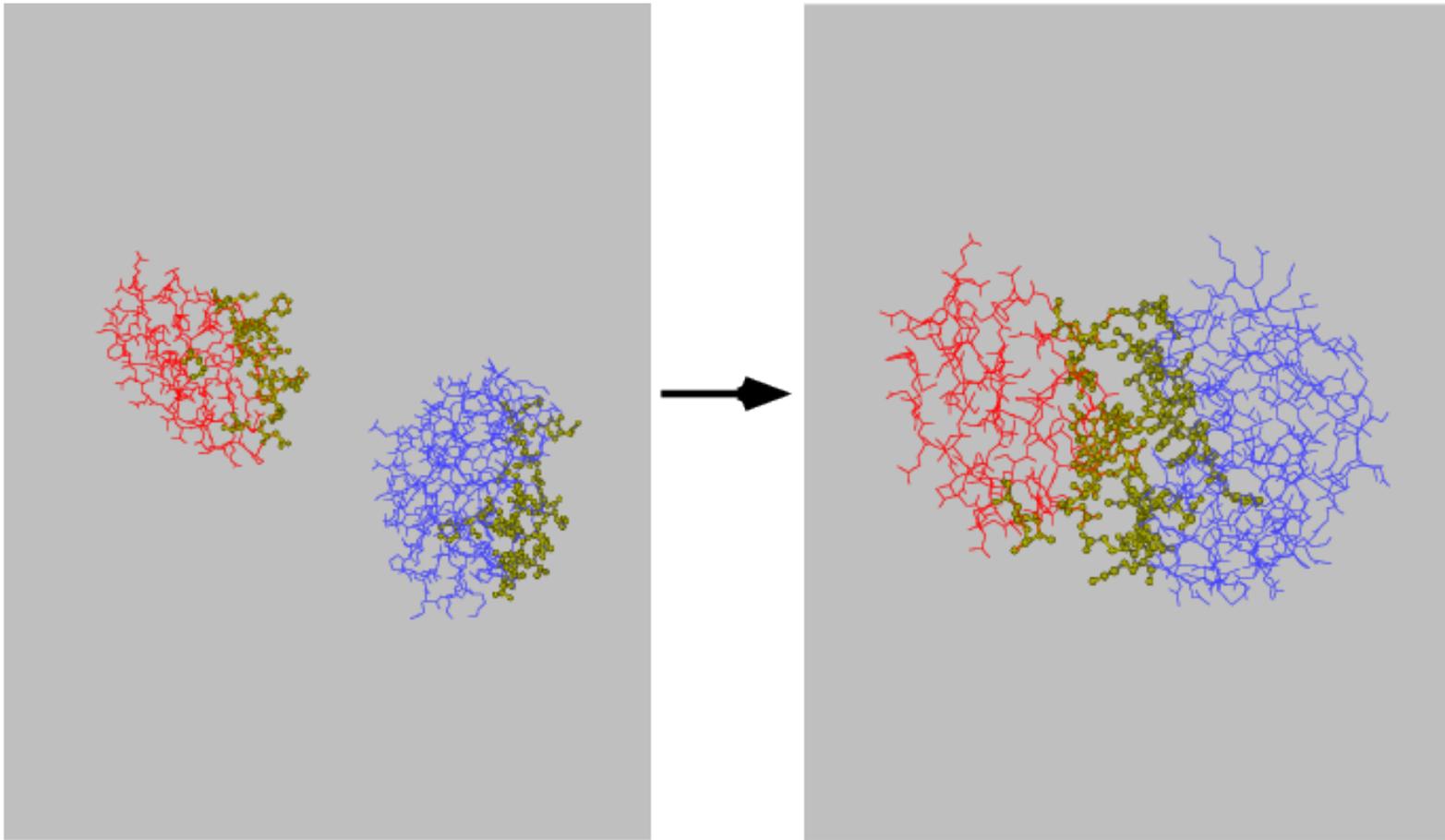
in biomolecular NMR structure determination, many internal coordinates are known or presumed to take usual values:

- bond lengths, angles- take values from high-resolution crystal structures.
- aromatic amino acid side chain regions - assumed rigid.
- nucleic acid base regions - assumed rigid.
- refinement against RDC data can't distort covalent geometry.
- non-interfacial regions of protein and nucleic acid complexes (component structures may be known- only interface needs to be determined)

Can we take advantage of this knowledge (find the minima more efficiently)?

- can take larger MD timesteps (without high freq bond stretching)
- configuration space to search is smaller:
 $N_{\text{torsion angles}} \sim 1/9 N_{\text{Cartesian coordinates}}$
- don't have to worry about messing up known coordinates.

Hierarchical Refinement of the Enzyme II/ HPr complex



active degrees of freedom are displayed in yellow.

MD in internal coordinates is nontrivial

Consider Newton's equation:

$$F = Ma$$

for MD, we need a , the acceleration in internal coordinates, given forces F .

Problems:

- express forces in internal coordinates
- solve the equation for a .

In Cartesian coordinates a is (vector of) atomic accelerations. M is diagonal.

In internal coordinates M is full and varies as a function of time: solving for a scales as $N_{\text{internal coordinates}}^3$.

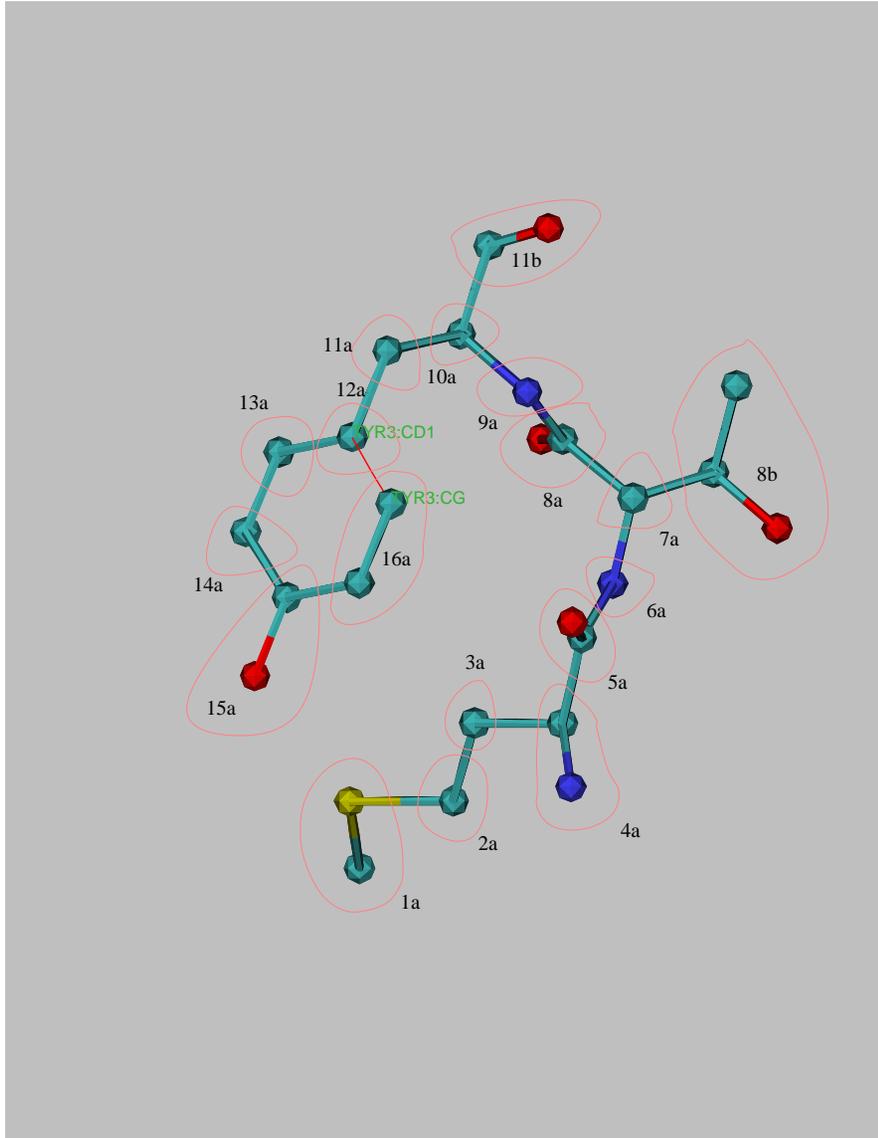
Solution: comes to us from the robotics community. Involves clever solution of Newton's equation: The molecule is decomposed into a tree structure, a is solved for by iterating from trunk to branches, and backwards.

Xplor-NIH implementation: C.D. Schwieters and G.M. Clore; J. Magn. Reson. 152, 288-302 (2001).

A copy of the IVM paper with some corrections is available at

<http://nmr.cit.nih.gov/xplor-nih/doc/intVar.pdf>

Tree Structure of a Molecule



atoms are placed in rigid bodies, fixed with respect to each other.

between the rigid bodies are “hinges” which allow appropriate motion

rings and other closed loops are broken- replaced with a bond.

Topology Setup

torsion angle dynamics with fixed region:

```
from ivm import IVM
integrator = IVM()
integrator.fix( AtomSel("resid 100:120") )
integrator.group( AtomSel("resid 130:140") )

from protocol import torsionTopology
torsionTopology(integrator)
```

#create an IVM object
these atoms are fixed in space
fix relative to each other,
but translate, rotate in space

group rigid side chain regions
break proline rings
group and setup all remaining
degrees of freedom for
torsion angle dynamics

topology setup of pseudoatoms
e.g. alignment tensor atoms:
- tensor axis should rotate
only - not translate.
- only single dof of Da and Rh
parameter atoms is significant.

IVM Implementation details:

other coordinates also possible: e.g. mixing Cartesian, rigid body and torsion angle motions.

convenient features:

- variable-size timestep algorithm
- will also perform minimization
- facility to constrain bonds which cause loops in tree.

full example script in `eginput/gb1_rdc/refine.py` of the Xplor-NIH distribution.

Dynamics with variable timestep

```
import protocol
bathTemp=2000
protocol.initDynamics(ivm=integrator,          #note: keyword arguments
                    bathTemp=bathTemp,
                    finalTime=1,              # use variable timestep
                    printInterval=10,        # print info every ten steps
                    potList=pots)
integrator.run()                             #perform dynamics
```

High-Level Helper Classes

AnnealIVM: perform simulated annealing

```
from simulationTools import AnnealIVM
anneal= AnnealIVM(initTemp =3000,          #high initial temperature
                  finalTemp=25,          #final temperature
                  tempStep =25,          # temperature increment
                  ivm=integrator,        # ivm object used for molecular dynamics
                  rampedParams = coolParams) #list of energy parameters to scale

anneal.run() # actually perform simulated annealing
```

Force constants of some terms are geometrically scaled during refinement: during simulated annealing step n of N , the force constant is

$$k^{(n)} = \gamma^n k^{(0)}$$

- $k^{(0)}$ and $k^{(N)}$ - initial and final force constants
- $\gamma^N = k^{(N)} / k^{(0)}$

```
from simulationTools import MultiRamp      #multiplicatively ramped parameter
coolParams=[]
coolParams.append( MultiRamp(2,30,        #change NOE scale factor
                           "noe.setScale( VALUE )" ) )
```

StructureLoop: calculate multiple structures

```
from simulationTools import StructureLoop
StructureLoop(structureNums=range(10),          # calculate 10 structures
              structLoopAction=calcStructure,  # calcStructure is function
              pdbTemplate=pdbTemplate)        # template for output structures

pdbTemplate = 'SCRIPT_STRUCTURE.sa'
#SCRIPT -> replaced with the name of the input script (e.g. 'anneal.py')
#STRUCTURE -> replaced with the number of the current structure
```

StructureLoop also helps with analysis:

```
from simulationTools import StructureLoop, FinalParams
StructureLoop(structureNums=range(10),
              structLoopAction=calcStructure,
              pdbTemplate=outFilename,
              pdbFilesIn="file_*.pdb"         # specify input files
              doWriteStructures=True,         # after calcStructure, write structure, viol
              averageTopFraction=0.5,        # fraction of structures to use
              averageFitSel="not hydro",     #atoms used for fitting structures
              averagePotList=potList,        #terms to use to compute of ave. struct
              averageContext=FinalParams(rampedParams), #force constants used
              averageFilename="ave.pdb",     #output filename
              genViolationStats=True,        # generate a .stats file with
                                              # energy/violation/structure stats
              ).run()
```

StructureLoop transparently takes care of parallel structure calculation.

Parallel computation of multiple structures

Computation of multiple structures with different initial velocities and/or coordinates: gives idea of structure precision, convergence of calculation.

on a multi-processor computer:

```
xplor -smp <number of CPUs> -py ...
```

on a cluster running PBS: [Supported implementations: PBSPro and Torque]

```
pbsxplor -l nodes=<number of nodes> -py ...
```

on a cluster running SLURM:

```
slurmXplor -py -nodes < num > [options] script.py
```

on a Scyld cluster

```
xplor -scyld <number of CPUs> -py ...
```

or manual node specification

```
xplor -parallel -machines <machine file> -py ...
```

convenient Xplor-NIH parallelization

- spawns multiple versions of xplor on multiple machines via ssh or rsh.
- structure and log files collected in the current local directory.
- robust to crashing compute nodes, crashing XPLOR runs, and the presence of dead nodes

requirements:

- ability to login to remote nodes via ssh or rsh, without password
- shared filesystem which looks the same to each node
- fully populated /bin and /usr/bin directories.

following environment variables set: XPLOR_NUM_PROCESSES, XPLOR_PROCESS

Integrative Approaches to Structure Calculation

Combine multiple sources of data

Combine NMR data with

- Solution Scattering - SAXS, SANS
- Cryo-EM
- X-ray crystallography
- Fiber Diffraction
- EPR

Solution Scattering Intensity

types of experiments:

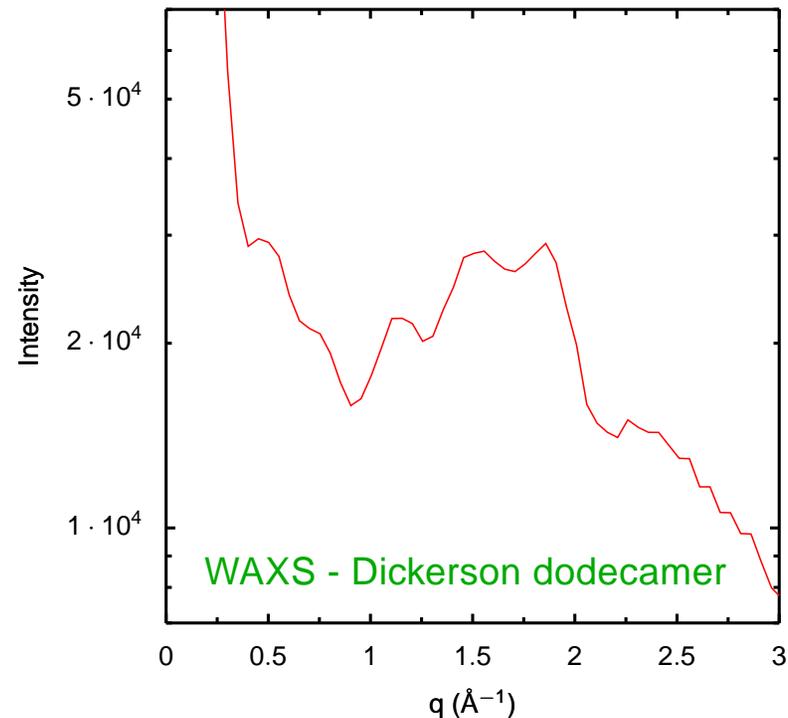
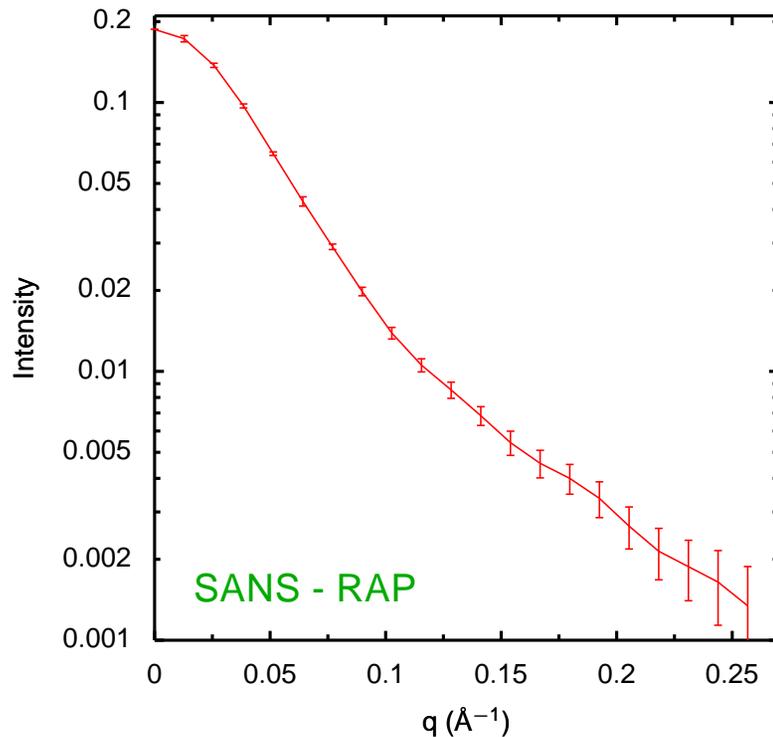
- small-angle X-ray scattering (SAXS)
- wide (or large) angle X-ray scattering (WAXS)
- Neutron scattering (SANS)

Provides information on overall molecular shape, size

→ complementary to solution NMR

→ very similar sample conditions

Example Spectra:



Calculating Scattering Intensity

Sum over all atoms: point-source scatterers

$$A(\mathbf{q}) = \sum_j f_j^{\text{eff}}(q) e^{i\mathbf{q} \cdot \mathbf{r}_j},$$

scattering vector amplitude: $q = 4\pi \sin(\theta)/\lambda$

$\theta = 0$ is the forward scattering direction

effective atomic scattering amplitude: $f_j^{\text{eff}}(q) = f_j(q) - \rho_s g_j(q)$

$f_j(q)$: vacuum atomic scattering amplitude

$\rho_s g_j(q)$: contribution from excluded solvent

-> boundary layer contribution can be optionally included

Difference between neutron and X-ray calculation: different $f_i^{\text{eff}}(q)$

Measured intensity

$$I(q) = \langle |A(\mathbf{q})|^2 \rangle_{\Omega}$$

$\langle \cdot \rangle_{\Omega}$: average over solid angle

Closed form solution: the Debye formula:

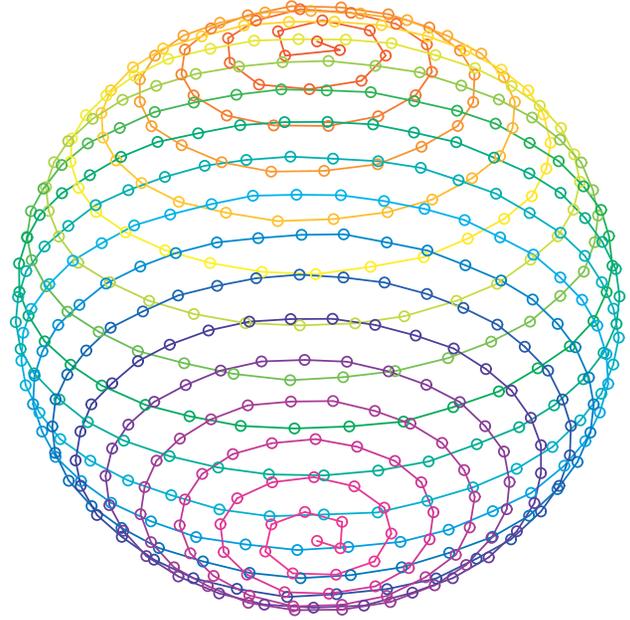
$$I(q) = \sum_{i,j} f_i^{\text{eff}}(q) f_j^{\text{eff}}(q) \text{sinc}(qr_{ij}),$$

sum is over all pairs of atoms. Expensive!

Scattering Intensity Approximations

Instead, compute $A(\mathbf{q})$ on a sphere and integrate over solid angle numerically.

Points are selected quasi-uniformly on the sphere using the Spiral algorithm:



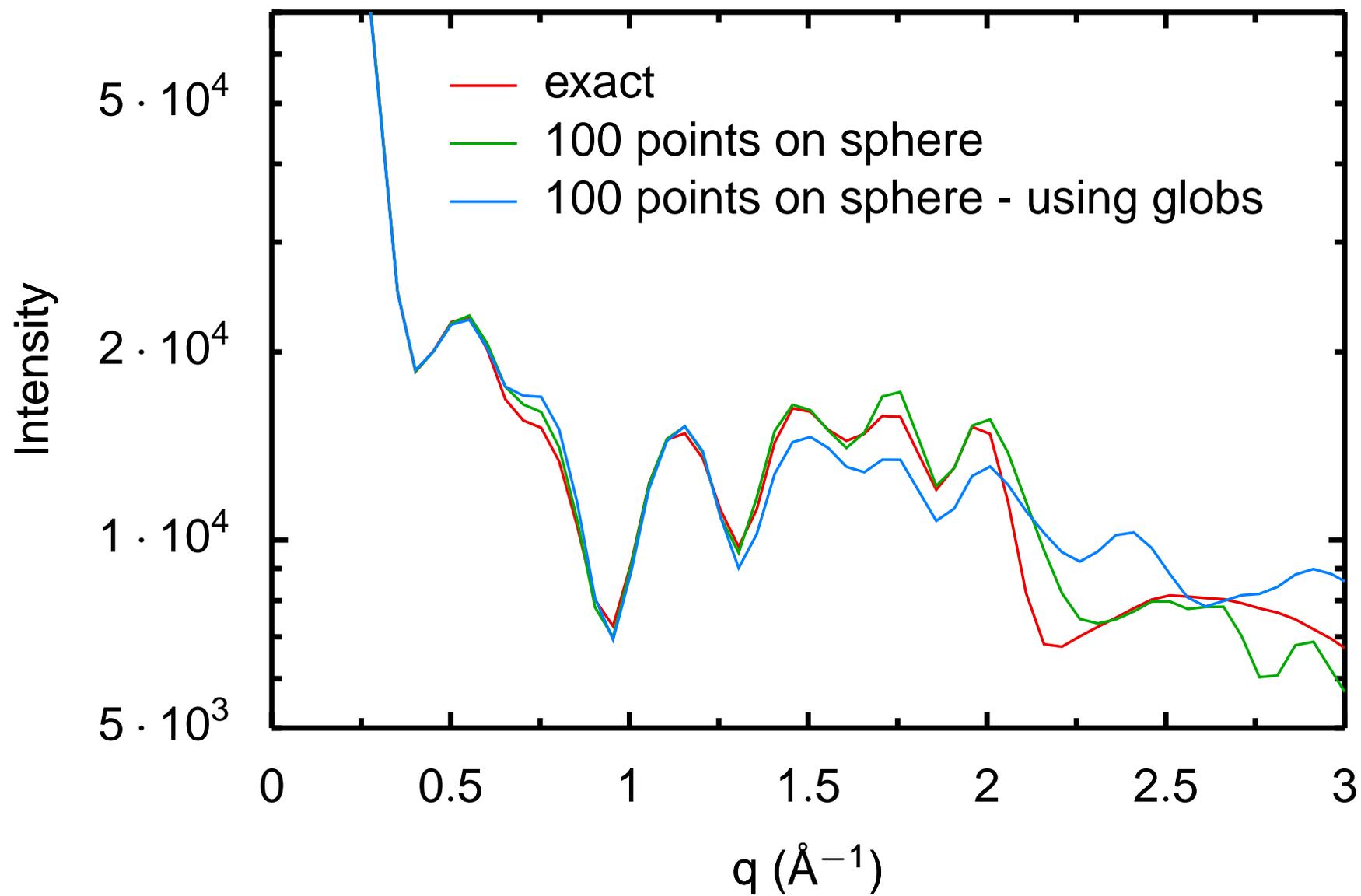
Additionally, combine atoms in “globs”:

$$f_{\text{glob}}(q) = \left[\sum_{i,j} f_i^{\text{eff}}(q) f_j^{\text{eff}}(q) \text{sinc}(qr_{ij}) \right]^{1/2},$$

Correct globbing, numerical integration errors with a multiplicative q -dependent correction factor c_{correct} :

$$I(q) = c_{\text{correct}}(q) I_{\text{approx}}(q),$$

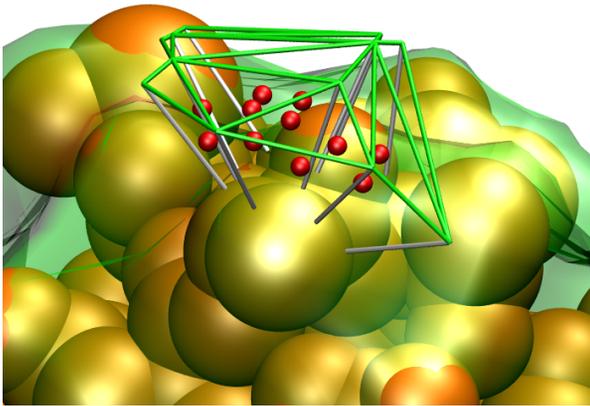
Calculated intensity for DNA scattering: numerical and globbing approximations:



Boundary layer contribution

Bound water contributes to the scattering amplitude.

Model as a layer of uniform thickness around the molecular structure with density ρ_b .



- Use the Varshney^a algorithm to efficiently generate an outer surface: roll solvent molecule over atoms whose radii are increased by r_b .
- Inner surface is generated using the points and surface normals.
- Each voxel defined by the tessellation procedure contributes to the scattering amplitude:

$$\sum_k f^{\text{sph}}(q; r_k) e^{i\mathbf{q} \cdot \mathbf{y}_k},$$

with

$$f^{\text{sph}}(q; r_k) = \rho_b 4\pi / q^2 [\sin(qr_k) / q - r_k \cos(qr_k)]$$

^aA. Varshney, F.P. Brooks, W.V. Wright, *IEEE Comp. Graphics App.* **14**, 19-25 (1994)

Alternate fit procedure available if buffer subtraction is significant source of error.

Determining Solvent Scattering Parameters

as in Crysol^a three parameters are fit

Effective atomic scattering amplitude:

$$f_j^{\text{eff}}(q) = f_j(q) - \rho_s g_j(q),$$

$f_j(q)$: vacuum atomic scattering amplitude

ρ_s : bulk solvent electron density amplitude due to excluded solvent:

$$g_j(q) = s_V V_j \exp(-\pi q^2 V_j^{2/3}) \times \exp[-\pi (q r_m)^2 (4\pi/3)^{2/3} (s_r^2 - 1)]$$

V_j : atomic volume

r_m : is the radius corresponding to the average atomic volume

s_V, s_r : scale factors to be fit.

Bound solvent scattering amplitude

$$f^{\text{sph}}(q; r_k) = \rho_b 4\pi/q^2 [\sin(qr_k)/q - r_k \cos(qr_k)]$$

ρ_b : boundary layer electron density

r_k : radius corresponding to voxel volume.

three parameters are fit using a grid search.

For SANS: one additional parameter: isotropic background added to calculated $I(q)$.

^aD. Svergun, C. Barberato and M.H.J. Koch, *J. Appl. Cryst.* **28**, 768-773 (1995).

Solution Scattering of Rigid Bodies

For atoms within a rigid body, the relative atom positions do not change, so after an initial calculation the corresponding contribution to the scattering amplitude can be computed without referring to atomic positions. If \mathbf{r}'_j is the atomic position of atom j after displacement of the rigid body with initial position given by \mathbf{r}_j , then

$$\mathbf{r}'_j = R\mathbf{r}_j + \Delta\mathbf{r},$$

where R and $\Delta\mathbf{r}$, respectively, describe the rotation and translation of the rigid body, the corresponding rigid body scattering amplitude is:

$$A_{\text{rigid}}(\mathbf{q}; \{\mathbf{r}\}) = e^{i\Delta\mathbf{r}\cdot\mathbf{q}} A_{\text{rigid}}^0(\mathbf{q}'; \{\mathbf{r}\}),$$

where $\{\mathbf{r}\}$ denotes the dependence on the set of initial atomic coordinates and

$$\mathbf{q}' = R^T \mathbf{q}.$$

In practice, $A_{\text{rigid}}(\mathbf{q}; \{\mathbf{r}\})$ is computed using a spline over a spherical surface of constant q to evaluate $A_{\text{rigid}}^0(\mathbf{q}'; \{\mathbf{r}\})$, the scattering amplitude at initial atomic position, but rotated scattering vector amplitude, \mathbf{q}' .

The use of this expression yields vast speedups when a calculation can be decomposed into a small number of rigid bodies, as it becomes independent of the number of atoms.

Refinement against solution scattering data

Refinement target function

$$E_{\text{scat}} = w_{\text{scat}} \sum_j \omega_j (I(q_j) - I^{\text{obs}}(q_j))^2,$$

$w_{\text{scat}}, \omega_j$: weight factors

Typically set $\omega_j = 1/\Delta I^{\text{obs}}(q_j)^2$ - inverse square of error. \rightarrow so $E_{\text{scat}} \sim \chi^2$.

- Correction factor c_{correct} periodically recomputed.
- Non-zero scattering contribution from surface-bound solvent- periodically computed, effect included in c_{correct}
- Rigid subunits' scattering contribution computed very efficiently during dynamics, minimization.
- Buffer background subtraction can be included using `solnScatPotTools.fitSolventBuffer`.
- SANS data is also supported

Example Xplor-NIH SAXS setup

```
from solnXRayPotTools import create_solnXRayPot
import solnXRayPotTools
xray=create_solnXRayPot('xray',
                        experiment='saxs.dat',
                        numPoints=26,
                        normalizeIndex=-3,preweighted=False)

xrayCorrect=create_solnXRayPot('xray-c',
                               experiment='saxs.dat',
                               numPoints=26,
                               normalizeIndex=-3,preweighted=False)

solnXRayPotTools.useGlobs(xray)
xray.setNumAngles(50)
xrayCorrect.setNumAngles(500)
potList.append(xray)
crossTerms.append(xrayCorrect)

#corrects I(q) for globbing, small angular grid and
# includes solvent contribution corrections
from solnScatPotTools import fitParams
rampedParams.append( StaticRamp("fitParams(xrayCorrect)") )
rampedParams.append(StaticRamp("xray.calcGlobCorrect(xrayCorrect.calcd())"))
```

Example Xplor-NIH SANS setup

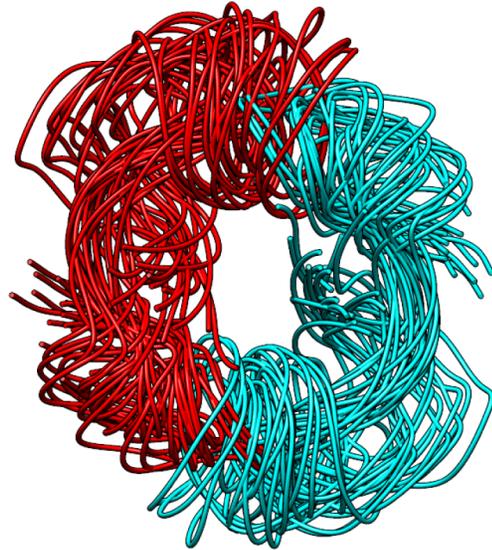
bound-solvent contribution frequently much less important

```
from sansPotTools import create_SANSPot
import sansPotTools
sans=create_SANSPot('sans',
                    experiment='sans.dat',
                    numPoints=20,
                    fractionD2O=0.41,
                    fractionDeuterated=1.,
                    altDeuteratedSels=[("resid 601:685",0.)],
                    cmpType="plain",
                    normalizeIndex=-3,preweighted=False)

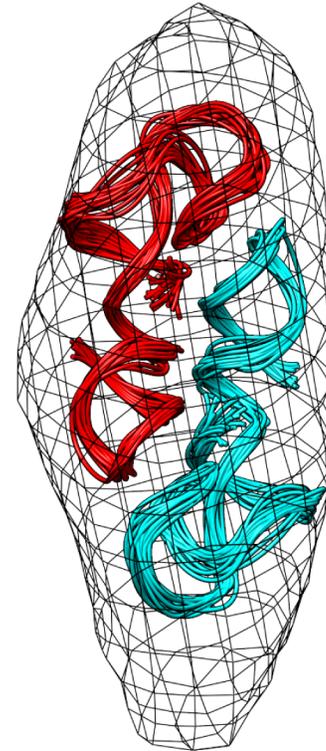
sansPotTools.useGlobs(sans)
sans.setNumAngles(80)
sans.setScale(40)
potList.append(sans)
#correct using the Debye equation
rampedParams.append( StaticRamp("sans.calcGlobCorrect('n2')") )
```

Cryo Electron Microscopy Data

Detailed molecular size and shape information!



Without EM

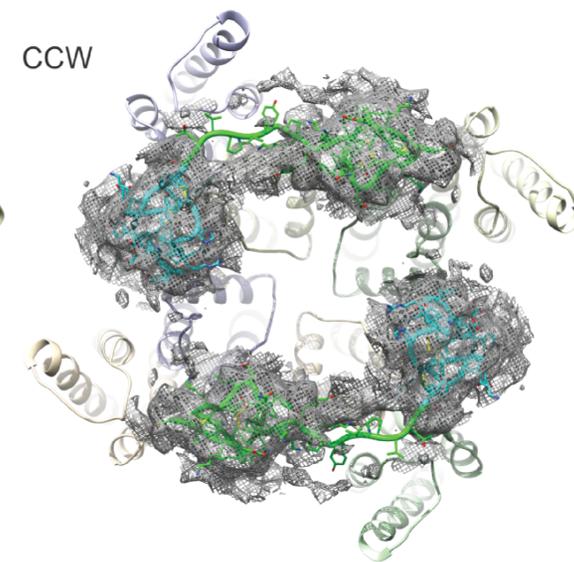
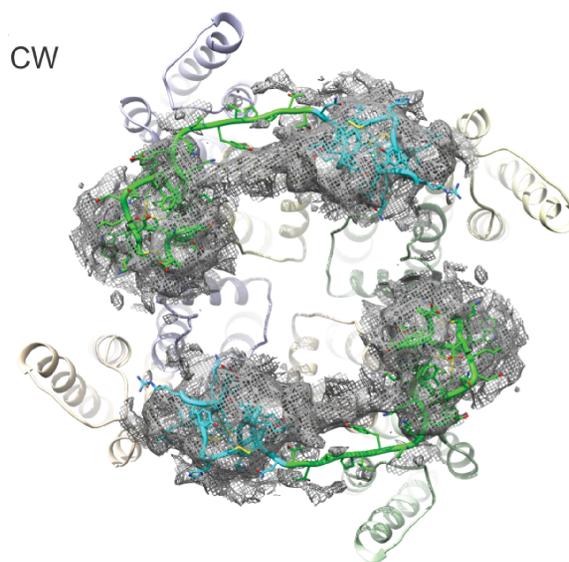
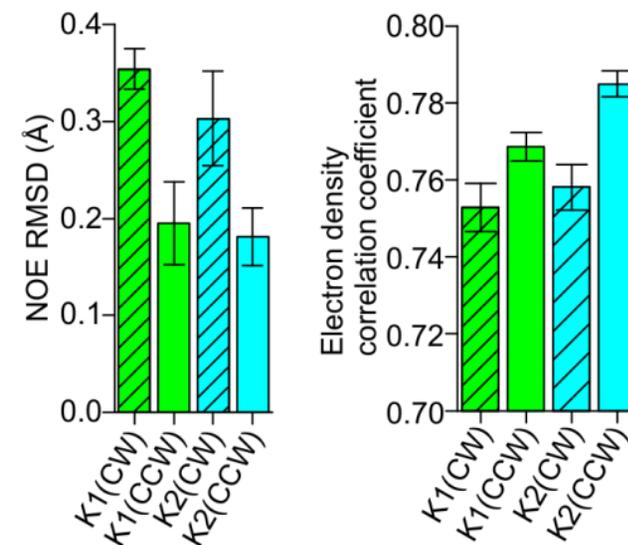
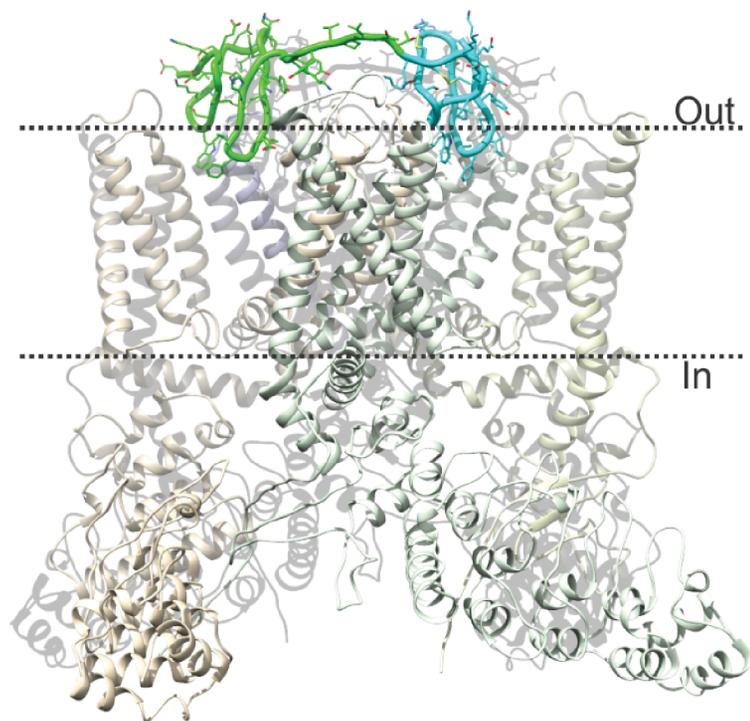


With EM

Gong *et. al.* Plos One **10**, e0120445 (2015)

```
from atomDensity import DensityGrid
Dmap = DensityGrid()
Dmap.readCCP4('map1.ccp4', verbose = True)
from probDistPotTools import create_probDistPot
prob = create_probDistPot("prob", Dmap, "not pseudo and not name H*",
                          potType = "cross_correlation",)
```

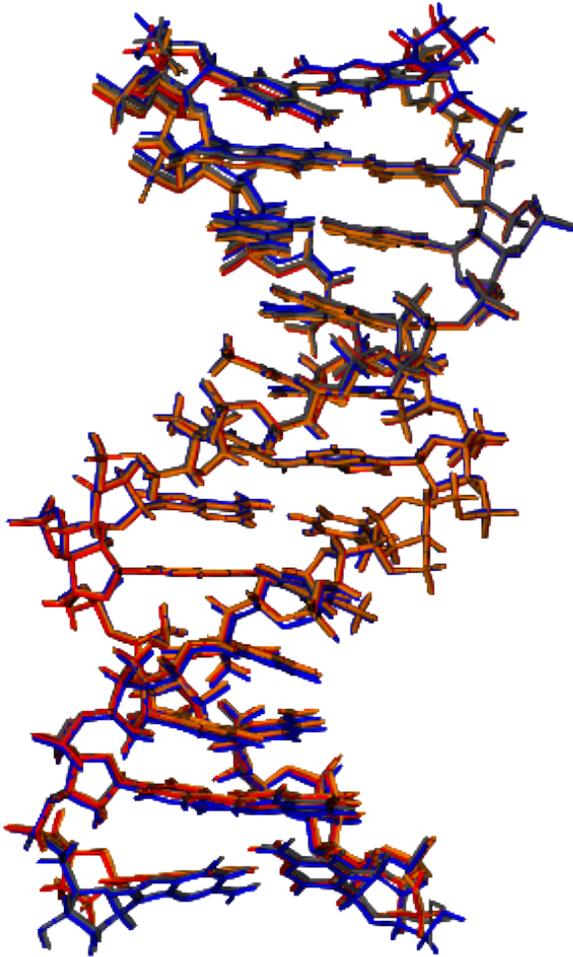
Cryo EM + NMR: TRPV1 with bound double-knot toxin



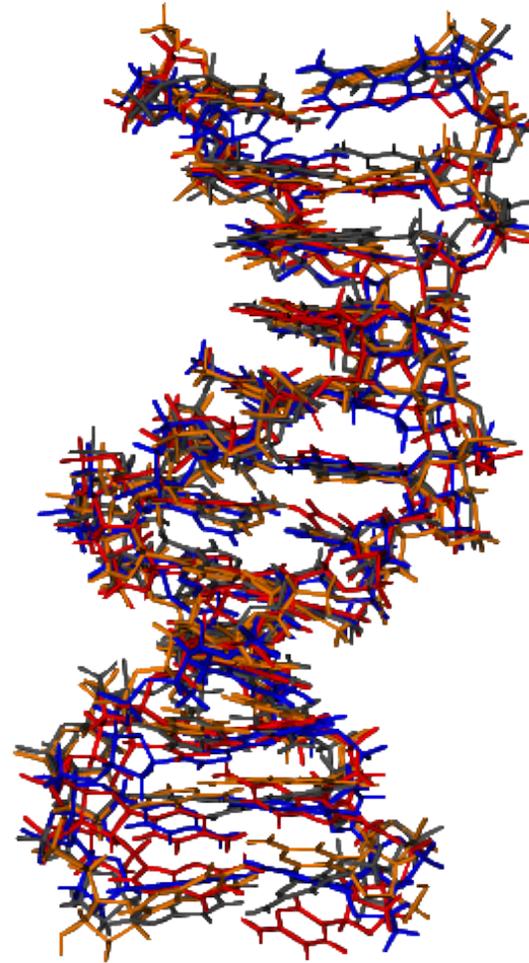
Bae *et. al.* eLife, **5**, e11273 (2016).

Refinement against an ensemble

Refinement of DNA 12-mer using NOE, RDC and X-ray scattering data



four calculated structures



One four-membered ensemble

Refinement against an ensemble

```
esim = EnsembleSimulation('ensemble',3) #creates a 3-membered ensemble
```

creates two extra copies of the current atom positions, velocities, *etc.*

Ensemble members don't interact, except with explicit potential terms.

Ensemble Features:

- Heterogeneous ensembles of mixed species can be treated.
- Ensemble calculations can be parallelized by specifying the `-num_threads` option to the `xplor` command.
- Care is taken to ensure data locality on Linux NUMA hardware.

Energy terms:

AvePot- average over the ensemble with no intra-ensemble interactions.

```
from avePot import AvePot
```

```
aveBond=AvePot(XplorPot,'bond') # ensemble averaged bond energy
```

aveBond's energy is $\langle E_{\text{BOND}} \rangle_e$ averaged over the ensemble.

Refinement against an ensemble

Most NMR observables must be averaged appropriately- AvePot is not appropriate- it only averages ensemble energies.

For example, the appropriate RDC value is $\langle D^{AB} \rangle_e$ averaged over the ensemble. The resulting energy is then $E(\langle D^{AB} \rangle_e)$.

Most Python energy terms will do proper ensemble averaging. XPLOR terms will not.

Additional potential terms: RAPPot, ShapePot - restrain atom positions within an ensemble - so members don't drift too far apart.

Example: restrain the positions of C_α atoms to be the same in all members of the ensemble.

```
from posRMSDPotTools import RAPPot
rap = RAPPot("ncs", "name CA") # create term
rap.setScale( 100.0 )
rap.setPotType( "square" )    # harmonic potential has a flat region
rap.setTol( 0.3 )             # 1/2-width of flat region
```

Can also refine against bond-vector **order parameter** for ensemble of size N_e , with unit vector u_i along the appropriate bond vector in ensemble member i

$$S^2 = \frac{1}{2N_e^2} \sum_{ij} (3 \cos(u_i \cdot u_j)^2 - 1)$$

[can use data from e.g. relaxation experiments.]

```
from orderPot import OrderPot
orderPot = OrderPot("s2_nh", open("nh_s2.tbl").read())
```

and **crystallographic temperature factor** for atom j in terms of q_{ij} , it's position in ensemble i , and it's ensemble-averaged value q_j

$$B_j = 8\pi^2 / N_e \sum_i |q_{ij} - q_j|^2$$

```
from posRMSDPotTools import create_BFactorPot
bFactor = PotList("bFactor")
```

Variable Ensemble Weights

Setting ensemble weights

```
esim = EnsembleSimulation('ensemble',3)
esim.setWeights( [0.2,0.1,0.7] ) # set weights for all ensemble members
noe = NOEPot('noe')
noe.setEnsWeights( [0.2,0.1,0.7] ) # set weight for only this NOE term
```

Ensemble Weights can be optimized during structure calculation.

```
from ensWeightsTools import create_EnsWeights
ensw = create_EnsWeights('ensw')
ensw.setWeights([0.2,0.2,0.6]) # set the initial/target weights
from sardcPotTools import create_SARDCPot
rdc = create_SARDCPot("RDC",restraints=stericRDC)
rdc.addEnsWeights(ensw) # specify that this potential term use this set
                        # of ensemble weights
```

different potential terms can have different ensemble weights

```
potList.append(ensw) #potential term biases the weights towards
                    # initial values
```

Using the EnsWeights energy term avoids situations with zero ensemble weight.

Symmetric Multimers

Maintain C_2 Symmetry

“NCS” term - keep dimer subunits identical

```
from posDiffPotTools import create_PosDiffPot
diNCS = create_PosDiffPot("diNCS", "segid A", "segid B")
potList.append(diNCS)
```

Distance symmetry to enforce C_2 symmetry

```
from distSymmTools import create_DistSymmPot, genDimerRestrains
dSymm = create_DistSymmPot('dSymm')
for r in genDimerRestrains(segids=['A', 'B'],
                           resids=range(10, 150, 10)):
    dSymm.addRestrains(r)
    pass
dSymm.setShowAllRestrains(True)
potList.append(dSymm)
```

Proper NOE distance calculation for SUM averaging subunit ambiguous restraint: the nMono setting

```
noe.setNMono(2)
```

Refinement in Explicit Solvent

Nederveen AJ, *et. al.*, “RECOORD: a recalculated coordinate database of 500+ proteins from the PDB using restraints from the BioMagResBank,” *Proteins*, **59**, 662-672 (2005).

Refinement Protocol:

1. solvate with water
2. heat system while
restraining protein heavy
atoms
3. high temperature dynamics
4. simulated annealing

	No Water	Explicit Water
<i>Violation analysis</i>		
RMS distance restraint violations (Å)	0.04 ± 0.06	0.04 ± 0.05
# Consistent violations > 0.5 Å ^b	0.3 ± 1.5	0.1 ± 0.6
RMS dih. restr. violations (degrees)	0.5 ± 0.7	0.5 ± 0.5
# Bumps per 100 residues ^c	11 ± 9	10 ± 7
<i>WHAT_CHECK Z-scores</i>		
2 nd Generation packing quality	-4.1 ± 1.9	-2.5 ± 2.0
Ramachandran plot appearance	-4.6 ± 1.2	-3.4 ± 1.4
χ ₁ /χ ₂ Rotamer normality	-0.9 ± 1.3	-0.9 ± 1.0
Backbone conformation	-3.4 ± 2.6	-3.8 ± 2.7
<i>DSSP secondary structure analysis</i>		
Helical content	22.3 ± 20.2	25.6 ± 22.2
Sheet content	14.6 ± 13.1	17.8 ± 15.0
Secondary structure content ^d	69.3 ± 10.9	73.7 ± 9.0
<i>PROCHECK results</i>		
Most favored regions	69.0 ± 13.1	76.1 ± 11.3
Allowed regions	26.0 ± 9.9	19.6 ± 8.5
Generously allowed regions	3.7 ± 3.2	2.5 ± 2.1
Disallowed regions	1.3 ± 1.4	1.8 ± 1.8
<i>Precision NMR ensemble^e</i>		
Backbone RMSD (Å)	2.9 ± 3.2	2.9 ± 3.1
Well-ordered RMSD (Å)	1.0 ± 1.4	1.1 ± 1.4
Backbone RMSD ORG (Å) ^f	3.7 ± 3.9	3.7 ± 3.8
Well-ordered RMSD ORG (Å)	1.4 ± 1.7	1.5 ± 1.6
Circular variance	0.05 ± 0.05	0.05 ± 0.03

```
import waterRefineTools
```

```
waterRefineTools.refine(potList=potList, coolingParams=rampedParams)
```

example in `eginput/gb1_rdc/wrefine.py`

Calculations Using Implicit Solvent: EEFx

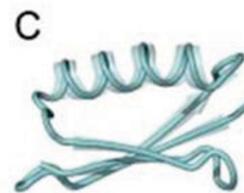
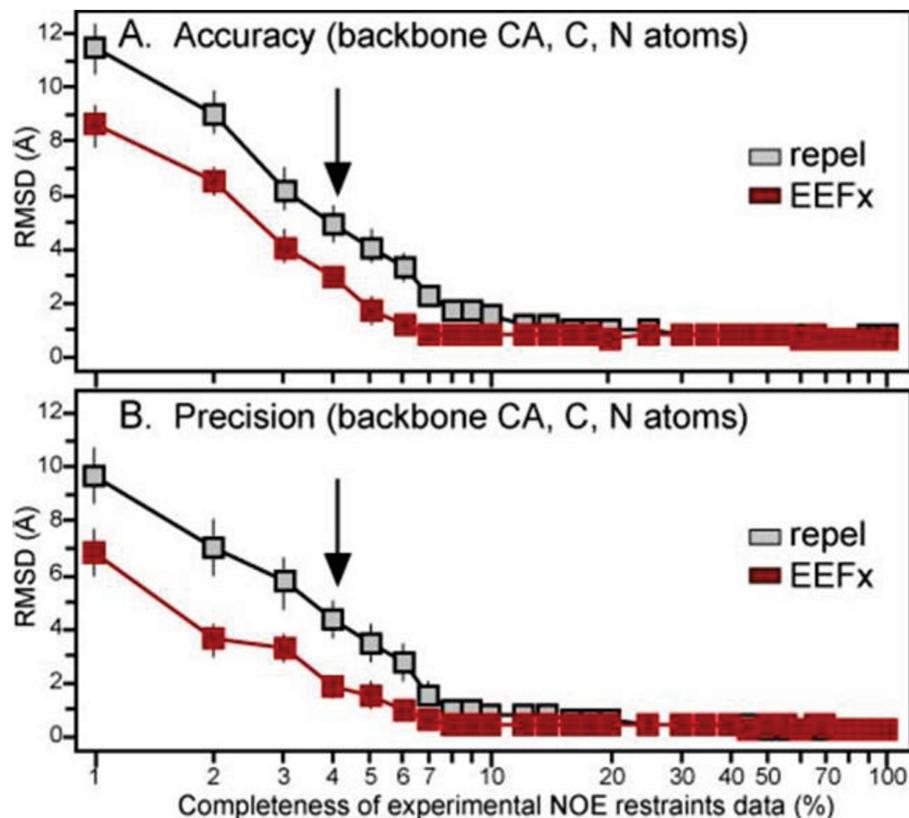
The EEFx Implicit Solvent Model

Y. Tian, *et. al.*, "A Practical Implicit Solvent Potential for NMR Structure Calculation," *J. Magn. Res.* **243**, 54-64 (2014).

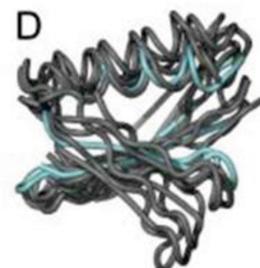
Can be used at all stages of structure determination.

Example in `eginput/gb1_rdc/refine_eefx.py`

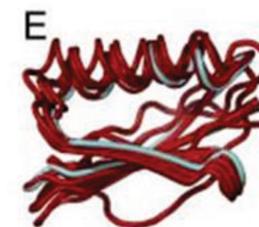
Now also includes implicit membrane potential.



Reference



No Solvent

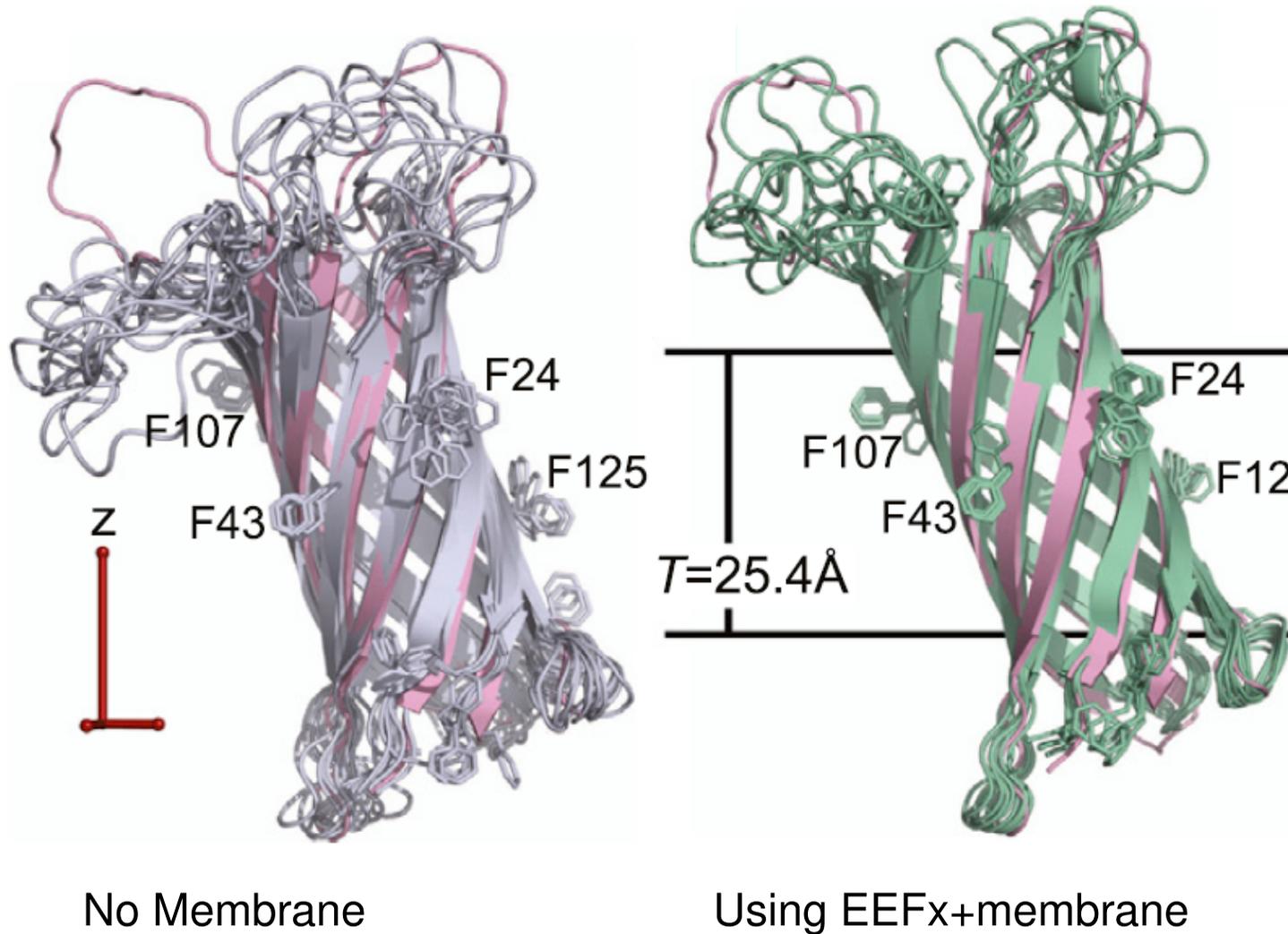


Using EEFx

```
from eefxPotTools import create_EEFxPot, param_LK
eefxpote=create_EEFxPot("eefxpote","not name H*",paramSet=param_LK)
```

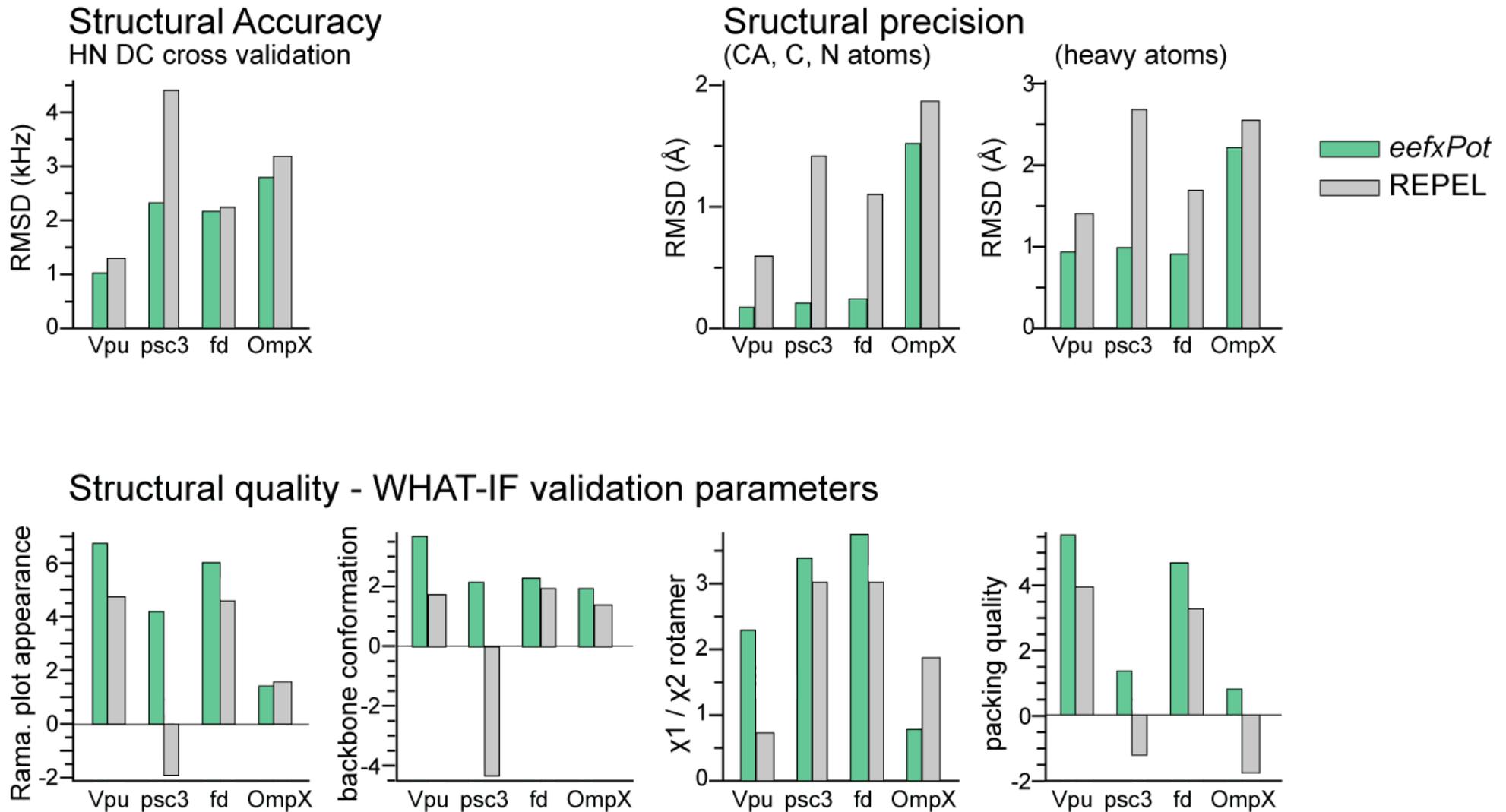
Implicit Solvent with a membrane: EEFx

Ye Tian, C.D. Schwieters, S.J. Opella and F.M. Marassi, "A Practical Implicit Membrane Potential for NMR Structure Calculations of Membrane Proteins," *Biophys J.* 109, 574-585 (2015).



Example in `eginput/eefx/membrane`

EEFXPot improves accuracy, precision and conformation of membrane protein structures



VMD interface: VMD-XPLOR

The screenshot displays the VMD 1.6.1 OpenGL Display window. The main view shows a protein structure with orange and yellow ribbons and red lines representing NOE restraints. The interface includes several panels:

- molWi:** A list of molecules (m1, m2) and NOE.
- VMD-XPLOR:** A sidebar with checkboxes for various tools like animate, color, display, graphics, labels, render, main, molWid, Dipcoup, NOE, TorCon, Mouse, Edit, and X-PLOR.
- NOE:** A panel for NOE constraint management, including a filename field (noe.in), a load button, and a Restraint Selection section with options for from, to, and set. It also shows NOE cost (23.86), Cutoff (0.5), and Monomers (1).
- Edit:** A panel for editing the current molecule (m2), featuring buttons for Undo changes, Done, and a table for Rotate and Translate operations.
- NOE Assignments - violated:** A list of violated NOE assignments with details such as residue numbers, names, and distances.
- Terminal:** A command window at the bottom showing the execution of X-PLOR commands to define molecules and load NOE restraints.

```
X-PLOR>
X-PLOR>ps
SSSStream::SSSStream: connecting to localhost on port 3377...
vmd_open: connected to :3377
PS>
PS> define m1
DEFINE> bonds ((name ca or name c or name n) and resid 1:259)
SELRPN: 777 atoms have been selected out of 5323
ASSFIL: file /var/tmp/schwitrs-ve27/xplor/m1 opened.
DEFINE> color name red end
DEFINE> end
PS>
PS> define m2
DEFINE> bonds ((name ca or name c or name n) and resid 300:385)
SELRPN: 255 atoms have been selected out of 5323
ASSFIL: file /var/tmp/schwitrs-ve27/xplor/m2 opened.
DEFINE> color name blue end
DEFINE> end
PS>
PS> set echo off end
VMD> ok
VMD> ok
PS>end
X-PLOR>
```

Assignment	Distance (Å)	Violation
0 resid 189 and name CA (2926) resid 315 and name CA (217)	11.92	VIOLATED
1 resid 189 and name CE1 (2936) resid 315 and name ND1 (223)	4.22	VIOLATED
8 resid 69 and name HA (1055) resid 317 and name HD (257, 258)	8.27822	VIOLATED
22 resid 74 and name HG (1146, 1147) resid 320 and name HA (297)	5.6693	VIOLATED
34 resid 78 and name HN (1197) resid 320 and name HB (299, 300, 301)	6.08771	VIOLATED
47 resid 78 and name HA (1199) resid 347 and name HD (695, 696, 697, 699, 700, 701)	6.24022	VIOLATED
48 resid 78 and name HN (1197) resid 347 and name HD (695, 696, 697, 699, 700, 701)	7.17231	VIOLATED
52 resid 79 and name HE (1218, 1219) resid 327 and name HE (418, 419)	6.42041	VIOLATED
	5.86692	VIOLATED

vmd-xplor screenshot

- visualize molecular structures
- visualize restraint info
- manually edit structures
- generate publication-quality figures

load multiple files at once

```
% vmd-xplor -noxplor refine*.pdb
```

command-line invocation of separate Xplor-NIH and VMD-XPLOR jobs:

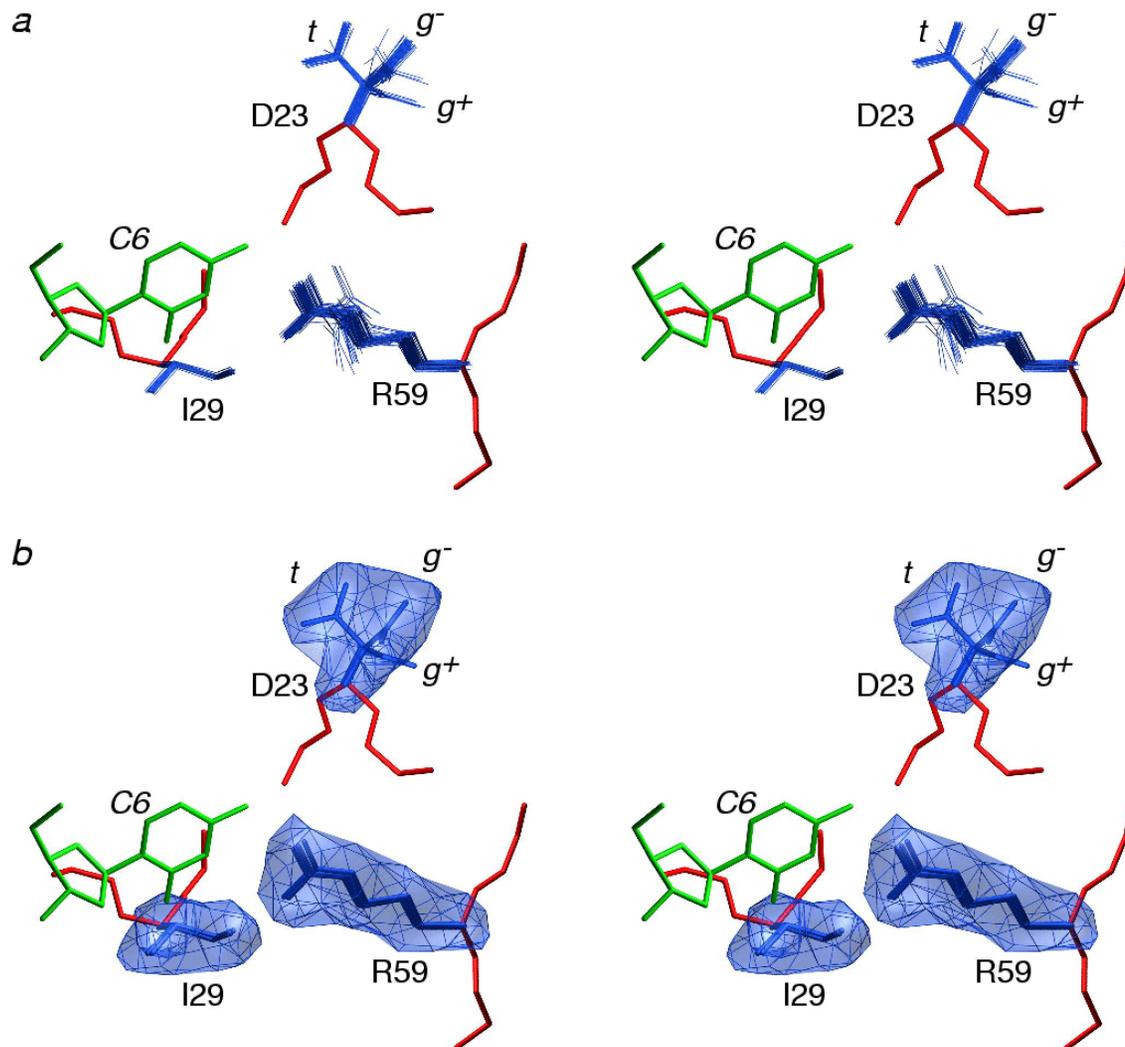
```
% vmd-xplor -port 3359 -noxplor
```

```
% xplor -port 3359 -py
```

Xplor-NIH snippet to draw bonds between backbone atoms, and labels:

```
import vmdInter  
vmd = VMDInter()  
x = vmd.makeObj("x")  
x.bonds( AtomSel("name ca or name c or name n") )  
label = vmd.makeObj("label")  
label.labels( AtomSel("name ca") )
```

Graphical Representation of ensembles



atom Prob: intelligently convert ensemble of structures into a probability distribution.

The PASD facility for automatic NOE assignment

developed by John Kuszewski

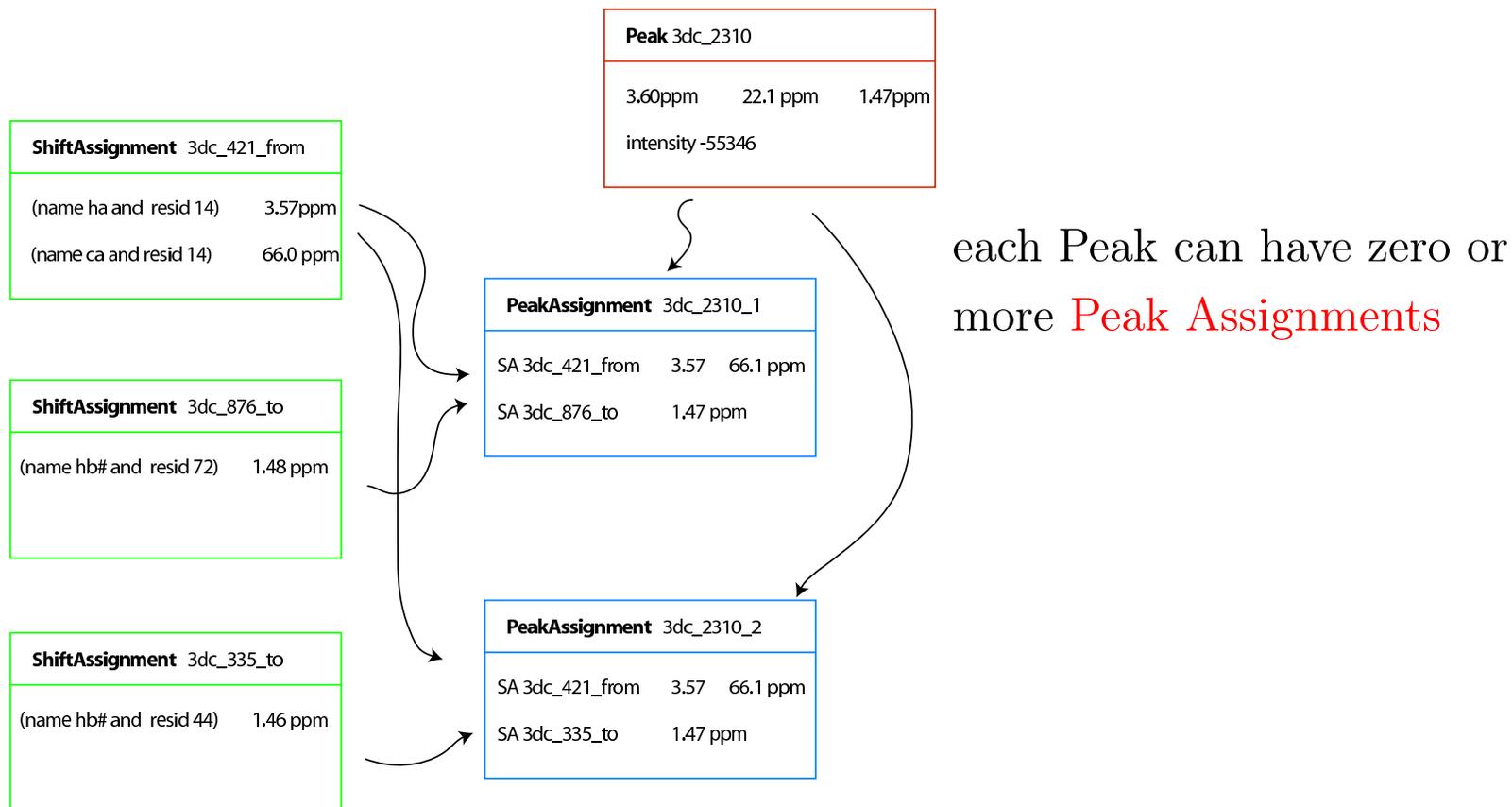
sometimes referred to as **Marvin**

main features:

- implemented in TCL interface.
- initial assignment likelihoods set by topological network of interconnected distance restraints.
- probabilistic selection of good NOE assignments
- for a given NOE peak, multiple possible assignments are simultaneously enabled during initial passes.
- inverse (repulsive) NOE restraints are used, consistent with the current set of active assignments.
- soft linear NOE energy.
- during structure calculation assignment likelihoods slowly change from relying on prior data to reflecting structures.
- successive passes of assignment calculation are not based on previously determined structures.
- in addition to NOE data, TALOS dihedral restraints are used.

The PASD facility

each NOE cross-peak generates a **Peak**



each Peak Assignment contains a from- and a to- **Shift Assignment** - selections of one or more atoms (containing generally indistinguishable atoms such as stereo pairs).

distances calculated between these selections using $1/r^6$ summing.

The PASD facility

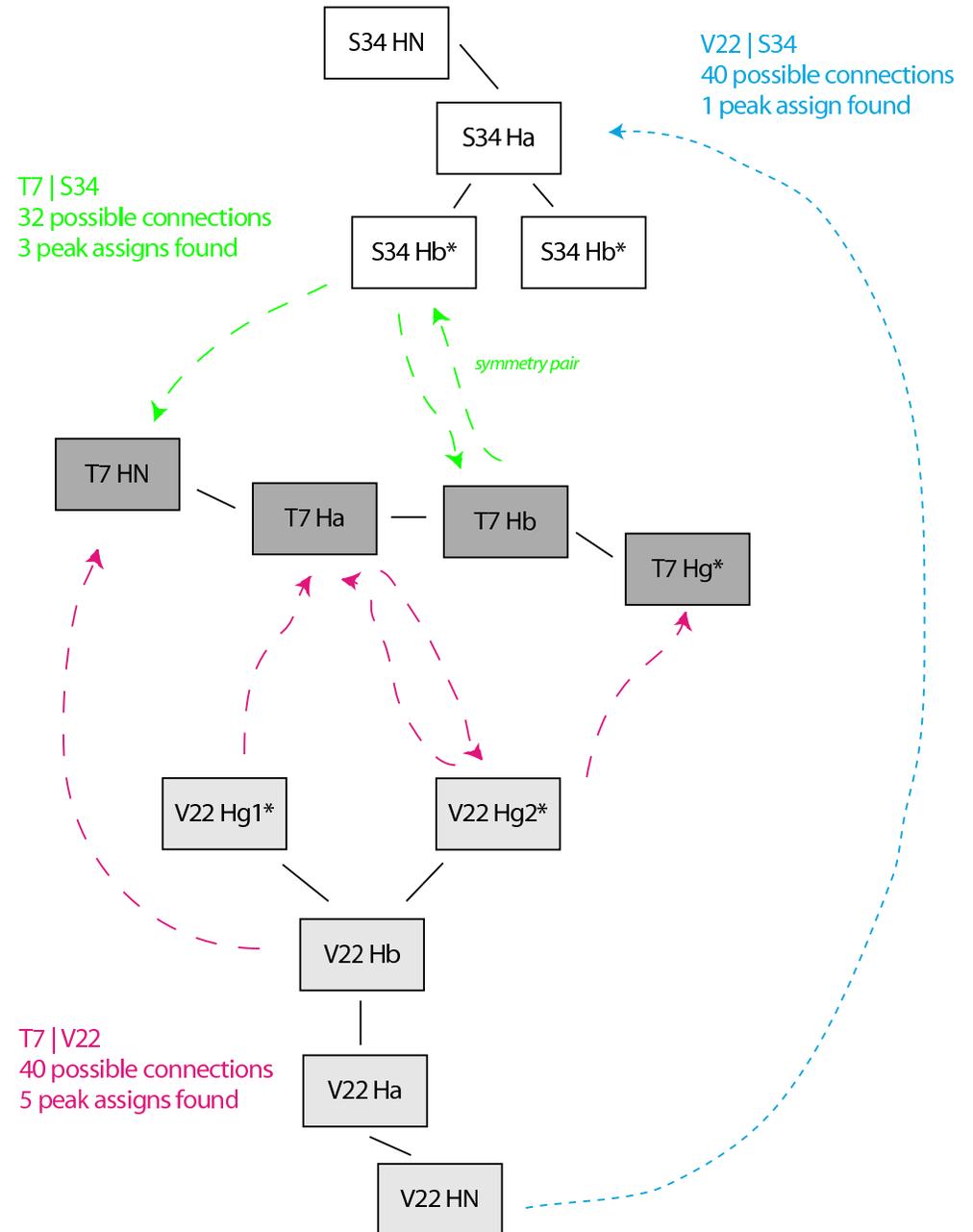
Initial Likelihoods

network analysis:

mark as likely assignments between residues with more interconnecting assignments.

Primary sequence filter:

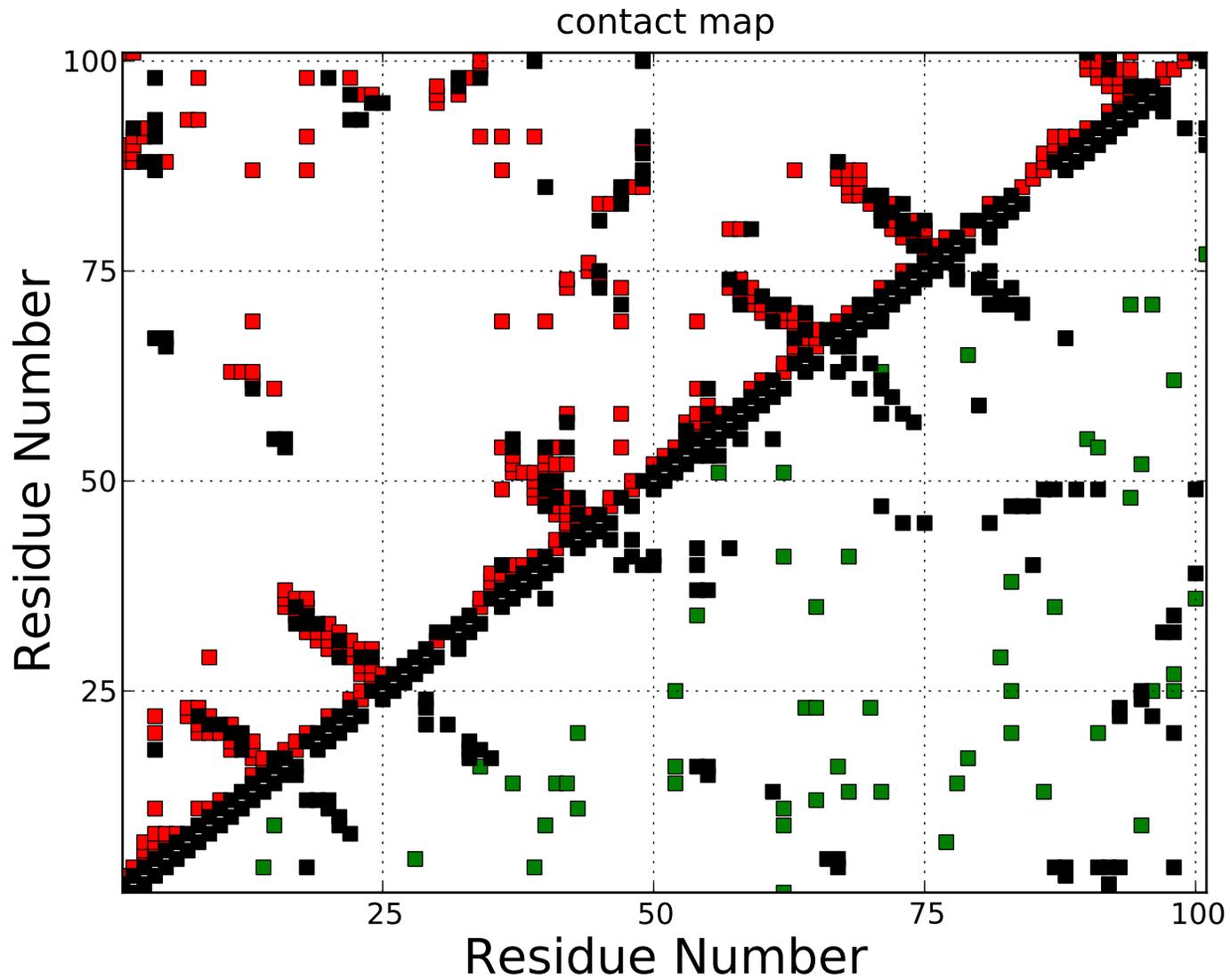
when there are multiple choices, always choose intra-residue assignment. Long range assignments get zero initial likelihood.



The PASD facility

Initial Likelihoods

network analysis: a contact map



The PASD facility

each assignment is activated or deactivated based on combination of current distance violations and prior likelihoods.

λ_i : likelihood of assignment i :

$$\lambda_i = w_0 \lambda_{pi} + (1 - w_0) \lambda_{vi}$$

λ_{pi} - prior likelihood fraction of good structures from previous calculation pass in which assignment i is satisfied.

λ_{vi} - instantaneous likelihood [= $\exp(-\Delta_i^2/D_v^2)$]

Δ_i - violation of assignment i

D_v - tunable parameter

$w_0 = 1 \dots 0$ - relative weight of λ_{pi} to λ_{vi}

assignment i is activated if random num between $0 \dots 1$ is smaller than λ_i

Entire collection of assignments is accepted or rejected using a Monte Carlo criterion, based on the NOE energy. Activation/deactivation of assignments is continued until Monte Carlo acceptance.

PASD Assignment optimization protocol

pass 1:

- start with collapsed structure with random torsion angles
- Linear NOE pot used.
- Inverse NOE potential used.
- high temp 1: 4000K
 - activation/deactivation carried out 10 times using only prior likelihoods.
 - only C_α nonbonded repulsion is enabled.
- high temp 2: 4000K
 - activation/deactivation carried out 10 times using equally weighted prior and instantaneous likelihoods ($w_0 = 0.5$).
- cooling: 4000 \rightarrow 100K
 - 64 assignment activation/deactivation steps, with decreasing D_v
 - w_0 reduced from 0.5 \rightarrow 0.
- prior likelihoods regenerated from top 10% of structures.

pass 2:

- quadratic NOE potential used.
- high temp: 4000K
 - assignment, single activated assignment chosen at 10 intervals, based solely on the pass 2 prior likelihoods.
- cooling: 4000 \rightarrow 100K
 - assignments selected, restraints activated/deactivated 64 times w_0 reduced 0.5 \rightarrow 0. force constants increased.

The PASD facility

Final Assignment:

- final likelihoods are computed for each assignment from top 10% of structures.
- incorrect restraint should have all likelihoods near 0
- correct restraint should have one assignment with a likelihood near 1.

Results:

- Demonstrated successfully on proteins with over 210 residues.
- method can tolerate about 80% bad NOE data.
- failure is clearly indicated by a low value of resulting NOE coverage: the number of long-range high-likelihood assignments per residue. [a value > 2]
- poor structural precision may mean that the algorithm failed, or that only subregions have been determined.
- regardless, high-likelihood assignments are very likely to be correct.

input formats supported: nmrdraw, nmrstar (including combined version 2.1), pipp, xeasy, Sparky, and NEF.

Helper Programs

The following helper programs are distributed with Xplor-NIH, and present in the bin subdirectory of the Xplor-NIH distribution. If the `--symlinks DIR` option to the `configure` script was used during installation, symbolic links to the programs will be present in the directory DIR. These options can be supplied to any Xplor-NIH command.

xplor: Invoke Xplor-NIH. Use the `-py` option for the Python interface.

pyXplor: Invoke the Xplor-NIH Python interpreter.

tclXplor: Invoke the Xplor-NIH TCL interpreter.

getBest: Print out the filenames of the lowest energy structures listed in an Xplor-NIH `.stats` file. By default, all `.stats` files in the current directory are consulted. It can also be used to create symbolic links to these files.

seq2psf: Generate a PSF file from a PDB file. This is only possible for PDBs with standard residues.

pdb2psf: Generates a psf file from the given pdb file. The psf file is written to a filename derived the input filename.

targetRMSD: Compute positional RMSD to a reference structure.

calcTensor: Calculate alignment tensor using SVD given RDC (or PCS) values and one or more molecular structures. Can optionally create plot of observed vs. calculated RDCs.

calcETensor: Calculate an ensemble of SVD alignment tensors from an ensemble of structures and observed RDC values. The tensors are underdetermined.

calcDaRh: Calculate an estimate of alignment tensor Da and rhombicity from input RDC values (no structures) using the assumption of isotropically distributed bond vectors and a maximum likelihood approach. The rdc tables use default Xplor-NIH format.

domainDecompose: Given an ensemble of structures, find regions of structural similarity, using maximum-likelihood fitting.

mleFit: Fit structures and determine unstructured regions using the maximum likelihood methodology of D.L. Theobald and D.S. Wuttke, "THESEUS: Maximum likelihood superpositioning and analysis of macromolecular structures," *Bioinformatics* 22, 2171-2172 (2006).

calcSAXS: Given a molecular structure, compute a SAXS or SANS curve, optionally fitting to experiment. Also compute optimal excluded solvent parameters (including boundary layer contribution).

calcSAXS-bufSub: Given a molecular structure, compute a SAXS curve, fitting to experimental an SAXS curve, while performing background subtraction. This is an adaptation of the AXES algorithm: A. Grishaev, L.A. Guo, T. Irving, and A. Bax, "Improved fitting of solution X-ray scattering data to macromolecular structures and structural ensembles by explicit water modeling," *J. Am. Chem. Soc.* 132, 15484-15486 (2010).

torsionReport: Generate report on all protein torsion angle values for one or more structure files. Reports generated for individual structures and averages.

findClusters: This script clusters structures by their positional RMSD, using the given atom selection.

compareTensors: Given two raw Saupe matrices, compute the normalized scalar tensor product and the dot product between axes. Also, compute angle of rotation between two axes

aveStruct: Given an ensemble of structures, compute an unregularized average structure and report per-atom RMSD.

ens2pdb: Converts an ensemble of structure files to a single PDB, with structures separated by MODEL records. The SEGID field is written to the ChainID

field so it must be, at most, a single character. The result should be acceptable for PDB submission.

calcSARDC: Given one or more structures, compute the Residual Dipolar Couplings resulting from partial alignment in a steric alignment medium and compare with observed values, optionally producing a plot. This will produce garbage for measurements made in charged alignment media.

contactMap: Generate contact map from one or more pdb files.

ramaStrip: Perform Ramachandran analysis on each specified residue. Print contours levels containing 98% and 99.8% of all high-resolution structure. For each structure specified, plots a blue point if the associated phi/psi value lies inside the 99.8% contour, and a red point otherwise. For each structure, also print out all violated phi/psi values on stdout. For a description of the structure database and generation of the density surface, please see G.A. Bermejo, G.M. Clore, and C.D. Schwieters, Protein Science 21, 1824-1836 (2012).

scriptMaker: Generate an Xplor-NIH script using a graphical user interface (written by Alex Maltsev).

calcDimerConc: Given subunit concentration and Kd, return dimer and monomer concentrations

convertTalos: Convert Talos+ and talos-N output to phi/psi torsion-angle restraints for use in Xplor-NIH. The allowed range of torsion angles will be all those from the ten Talos+/TalosN hit values. This script will produce restraints for all residues which for which Talos+/N classifies its predictions as "Good" or "Strong". By default, hits are ignored if Talos+/N give them zero weight.

runSparta: Compute the backbone chemical shifts, and optionally fit to experimental values. This uses a modified version of "SPARTA+": a modest

improvement in empirical NMR chemical shift prediction by means of an artificial neural network Yang Shen and Ad Bax," J. Biomol. NMR, 48, 13-22 (2010).

plotLog: Create a two dimensional plot from X-Y data.

calcPSol: Compute the solvent PRE given a molecule structure and a restraint list.

pbsxplor: Generate a PBS script and submit it as a PBS job using qsub.

slurmXplor: Generate a SLURM script and submit it as a s job using sbatch.

idleXplor: Edit and run Xplor-NIH scripts.

Where to go for help

online:

<http://nmr.cit.nih.gov/xplor-nih/>

<mailto:xplor-nih@nmr.cit.nih.gov>

<http://nmr.cit.nih.gov/xplor-nih/faq.html>

<http://nmr.cit.nih.gov/xplor-nih/doc/current/>

<http://nmr.cit.nih.gov/xplor-nih/doc/current/python/tut.pdf>

<http://nmr.cit.nih.gov/xplor-nih/xplor-nih-tutorial.tgz>

- home page
- mailing list
- FAQ
- current Documentation
- Tutorial
- Hands-on Examples

subdirectories within the xplor distribution:

eginputs - newer complete example scripts

tutorial - repository of older XPLOR scripts

helplib - help files

helplib/faq - frequently asked questions

Python:

M. Lutz, "Learning Python, 4th Edition" (O'Reilly, 2009);

<http://python.org>

TCL:

J.K. Ousterhout "TCL and the TK Toolkit" (Addison Wesley, 1994);

<http://www.tcl.tk>

Please complain! and suggest!