Xplor-NIH: Recent Developments

Charles Schwieters and John Kuszewski

November 17, 2003

outline

1. description, history
2. Scripting Languages: XPLOR, Python, TCL
   - Introduction to Python
3. Potential terms available from Python
4. IVM: dynamics and minimization in internal coordinates
5. Parallel determination of multiple structures
   - Using the Biowulf cluster
6. VMD molecular graphics interface

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

What is Xplor-NIH?

Biomolecular structure determination/manipulation

- Determine structure using minimization protocols based on molecular dynamics/ simulated annealing.
- Potential energy terms:
  - terms based on input from NMR (and X-ray) experiments: NOE, dipolar coupling, chemical shift data, etc.
  - other potential terms enforce reasonable covalent geometry (bonds and angles).
  - knowledge-based potential terms incorporate info from structure database.
- includes: program, topology, covalent parameters , potential energy parameters, data for knowledge-based potentials.
  [in the future: protocols.]

Automatic NOE Assignment:

- M. Nilges has said that his group will add full ARIA support to Xplor-NIH.
- John K's MARVIN auto-assignment facility.

What Xplor-NIH is not

Not general purpose molecular dynamics engine. Major deficiency: no Ewald summation for long-range electrostatic potentials. Use CHARMM, Amber, or NAMD.

Crystallography tools are dated. CNS X-ray facilities are more up-to-date.
But, CNS no longer under development, and its NMR facilities are dated.
$\longrightarrow$ use Xplor-NIH for NMR structure determination.

Not an NMR spectrum analysis tool.
[future: tighter integration with tools such as NMRWish.]

Scripting Languages- three choices

scripting language:

- flexible interpreted language
- used to input filenames, parameters, protocols
- relatively user-friendly

XPLOR language:

strong point:

selection language quite powerful.

weaknesses:

String, Math support problematic.

no support for functions/subroutines.

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

NOTE: all old XPLOR scripts should run unchanged in Xplor-NIH.

general purpose scripting languages: Python and TCL

- excellent string support.
- languages have functions: 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.

- Facilitate interaction, tighter coupling with other tools.
  - NMRWish has a TCL interface.
  - pyMol has a Python interface.
  - VMD has TCL and Python interfaces.

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

New development in C++: scripting interfaces (semi-)automatically generated using a tool called SWIG.

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
```

lists and tuples

```
l = [1,2,3]          #create a list
a = l[1]             #indexed from 0 (1 = 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 - whitespace scoping

```
def sum(a,b):
    "return the sum of a and b"    # comment string
    retVal = a+b                   # note indentation
    return retVal

print sum(42,1)                    #un-indented line: not in function
43
```

loops - the for statement

```
for cnt in range(0,3):
    cnt += 10
    print cnt
10
11
12
```

Introduction to Python

Python is modular

most functions live in separate namespaces called modules

The import statement - loading modules

```
import sys          #import module sys
sys.exit(0)         #call the function exit in module sys
```

or:

```
from sys import exit  #import exit function from sys into current scope
exit(0)               #don’t need to prepend sys.
```

Introduction to Python

In Python pretty much everything is an object.

Objects: calling 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 file
['__class__', '__delattr__', '__doc__', '__getattribute__',
  '__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

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 on the built-in function open
```

Accessing Xplor-NIH's Python interpreter

from the command-line: use the -py flag

```
% xplor -py
                       XPLOR-NIH version 2.9.2

 C.D. Schwieters, J.J.  Kuszewski,       based on X-PLOR 3.851 by A.T. Brunger
 N. Tjandra, and G.M. Clore
 J. Magn. Res., 160, 66-74 (2003).       http://nmr.cit.nih.gov/xplor-nih

python>
```

from XPLOR: PYTHon command

```
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

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
```

Atom Selections in Python

use the XPLOR atom selection language.

```
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 a string identifying the atom, and its position.

Python in Xplor-NIH

current status: low-level functionality (similar to that of XPLOR script) implemented.

partially implemented: high-level wrapper functions which will encode default values, and hide complexity.

future: develop repository of still-higher level protocols to further simplify structure determination.

Using potential terms in Python

available potential terms in the following modules:

1. rdcPot - dipolar coupling
2. noePot - NOE distance restraints
3. jCoupPot - ^{3}J-coupling
4. xplorPot - use XPLOR potential terms
5. potList - a collection of potential terms

all potential objects have the following methods:

instanceName()	-	name given by user
potName()	-	name of potential term, e.g. "RDCPot"
scale()	-	scale factor or weight
setScale(val)	-	set this weight
calcEnergy()	-	calculate and return term's energy

residual dipolar coupling potential

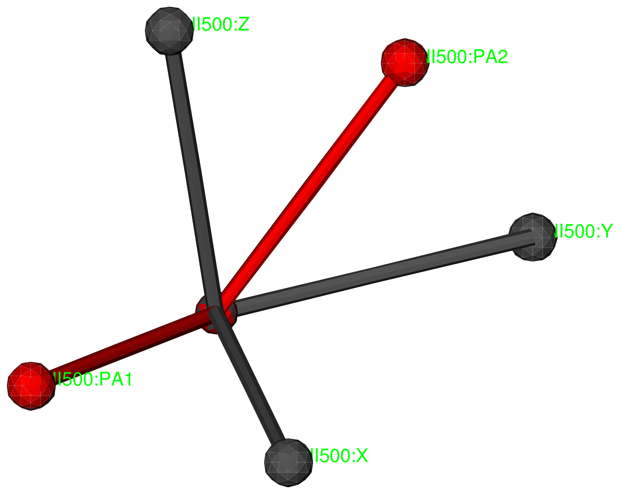
Provides orientational information relative to axis fixed in molecule frame.

$$\delta_{\text{calc}} = D_a[(3u_z^2 - 1) + \frac{3}{2}R(u_x^2 - u_y^2)] \; ,$$

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

rdcPot (in Python)

- tensor orientation encoded in four axis atoms
- allows Da, 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: $D_a \propto 1/r^3$.



How to use the rdcPot potential

```
from rdcPotTools import *         #import all symbols from this module

RDC_addAxisAtoms()
rdcNH = create_RDCPot("NH",file='NH.tbl')

rdcNH.setDa(7.8)          #set initial tensor properties
rdcNH.setRhombicity(0.3)
calcTensor(rdcNH)         #use if the structure is approximately correct
```

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

analysis, accessing potential values:

```
print rdcNH.instanceName()              # prints 'NH'
print rdcNH.potName()                   # prints 'RDCPot'
print rdcNH.rms(), rdcNH.violations()   # calculates and prints rms, violations
print rdcHN.Da(), rdcHN.rhombicity()    # prints these tensor quantities
rdcNH.setThreshold(0)                   # violation threshold
print rdcNH.showViolations()            # print out list of violated terms
print Rfactor(rdcNH)                    # calculate and print a quality factor
```

RDCPot: additional details

using multiple media:

```
RDC_addAxisAtoms()
rdcNH_2 = create_RDCPot("NH_2",file='NH_2.tbl')
#[ set initial tensor parameters ]
```

multiple expts. single medium:

```
rdcCAHA = create_RDCPot("CAHA",resid=rdcNH.oAtom().residueNum(),
                       file='CAHA.tbl')
```

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

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

```
scale_toNH(rdcCAHA,'CAHA')   #rescales relative to NH
scale = (5/2)**2
     #  ^ inverse error in expt. measurement relative to that for NH
rdcCAHA.setScale( scale )
```

NOE potential term

effective NOE distance (sum averaging):

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

Python potential in module noePot

- reads XPLOR-style NOE tables.
- potential object has methods to set averaging type, potential type, etc.

creating an NOEPot object:

```
from noePot import NOEPot
noe = NOEPot('noe', open('noe_all.tbl').read() )
```

analysis:

```
print noe.rms()
noe.setThreshold( 0.1 )          # violation threshold
print noe.violations()           # number of violations
print noe.showViolations()
```

J-coupling potential

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

θ is appropriate torsion angle.
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 jCoupPot import JCoupPot
Jhnha = JCoupPot('hnha',open('jna_coup.tbl').read())
```

analysis:

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

using XPLOR potentials

Example using a Radius of Gyration (COLLapse) potential

```
import protocol
from xplorPot import XplorPot
protocol.initCollapse('resid 3:72')      #specify globular portion
rGyr = XplorPot('COLL')
xplor.command('collapse scale 0.1 end') #manipulate in XPLOR interface
```

accessing associated values

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

all other access/analysis done from XPLOR interface.

Commonly used XPLOR terms: VDW, BOND, ANGL, IMPR, RAMA, HBDA, CDHI

collections of potentials - PotList

collection potential terms together:

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

nested PotLists:

```
rdcs = PotList('rdcs')                 #convenient to collect like terms
rdcs.add( rdcNH ); rdcs.add( rdcNH_2 )
pots.add( rdcs )
for pot in pots:                       #pots looks like a list
    print pot.instanceName()
noe
hnha
COLL
rdcs
```

The IVM (internal variable module)

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

- bond lengths, angles.
- aromatic amino acid sidechains
- nucleic acid base regions
- 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: $N_{\text{torsion angles}} \sim 1/3 N_{\text{Cartesian coordinates}}$
- don't have to worry about messing up known coordinates.

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:

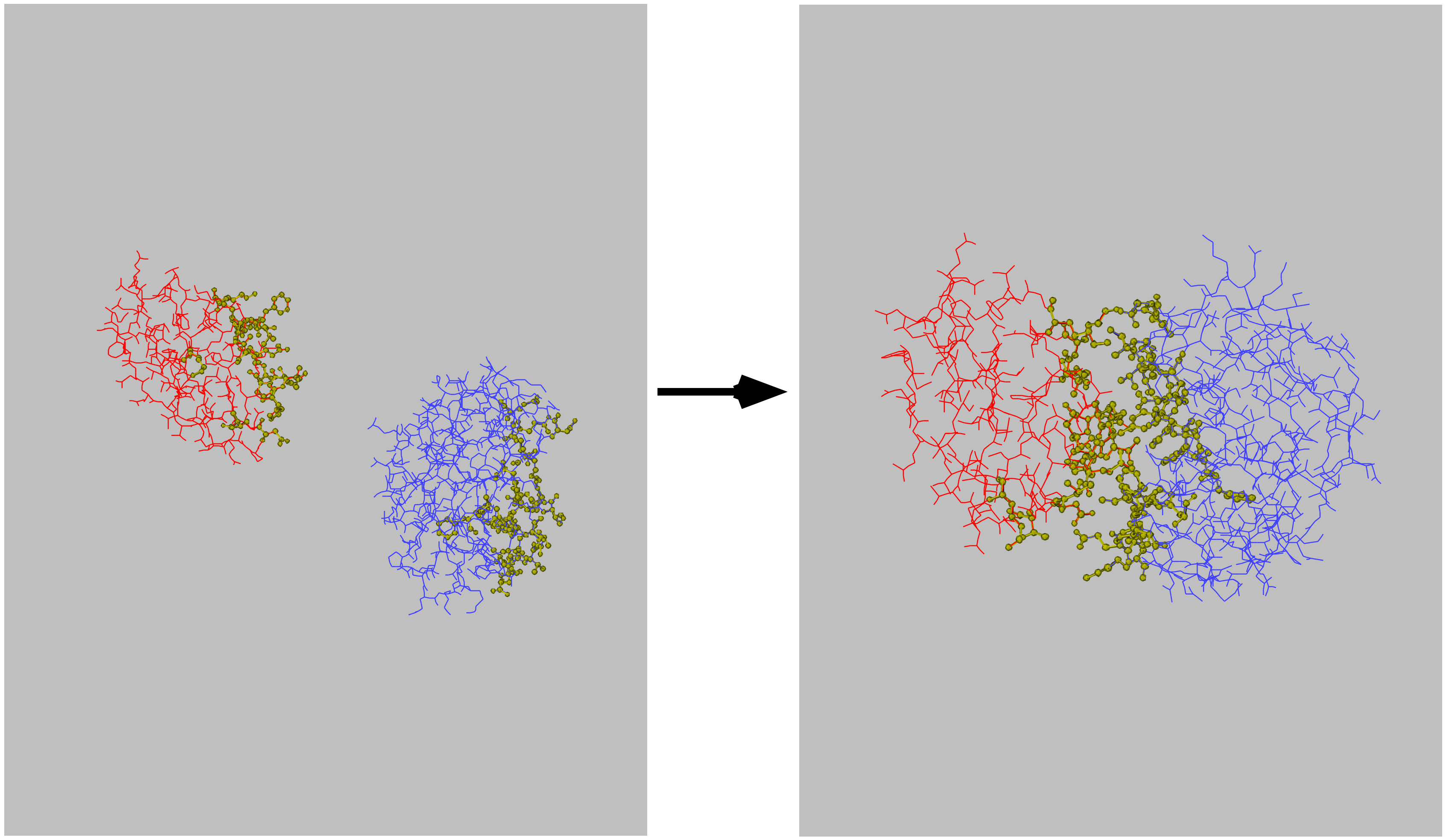
- express forces in internal coordinates
- solve the equation for a.

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 $N^3_{\text{internal coordinates}}$.

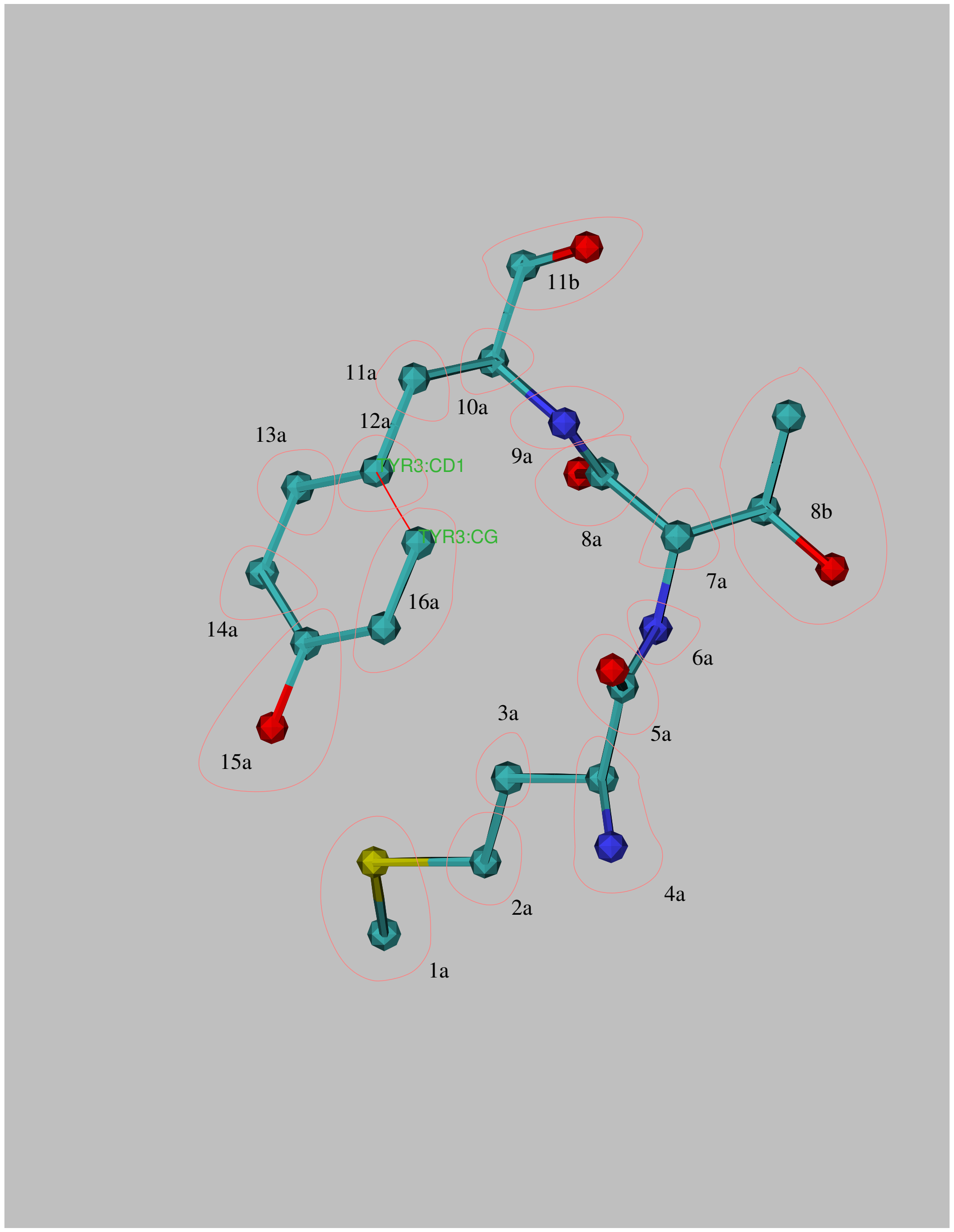
Solution: comes to us from the robotics community. Involves clever solution of Newton's equation: The molecule is decomposed into a tree structure, a is solved for by iterating from trunk to branches, and backwards.

Hierarchical Refinement of the Enzyme II/ HPr complex



active degrees of freedom are displayed in yellow.

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
from selectTools import groupRigidSideChains, IVM_breakProlines

integrator = IVM()                             #create an IVM object
integrator.group( groupRidigSideChains() )     # keep aromatic regions rigid
IVM_breakProlines(integrator)                  # break proline rings appropriately
integrator.fix( AtomSel("resid 100:120") )     # these atoms are fixed
                                               # wrt each other
integrator.autoTorsion()                       # all other regions have torsion
                                               # angles active
```

rdc topology setup - for tensor atoms

axis should rotate only - not translate.

only single bond angle of D_a and rhombicity parameter atoms is significant.

```
from rdcPotTools import *
configIVM(rdcNH,integrator)        #always call this
```

Initially, when far from correct structure, fix D_a, and rhombicity:

```
configIVM_fixDa(rdcNH,integrator)
configIVM_fixRhombicity(rdcNH,integrator)
```

to refine against these:

```
configIVM_varyDa(rdcNH,integrator)
configIVM_varyRhombicity(rdcNH,integrator)
```

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 eginputs/protG/anneal.py of the Xplor-NIH distribution.

dynamics with variable timestep

```
import protocol
bathTemp=2000
protocol.initDynamics(ivm=integrator,      #note: keyword arguments
                      bathTemp=bathTemp,
                      potList=pots)

integrator.setFinalTime(1)        # will use variable timestep algorithm
integrator.setPrintInterval(10)   # print info every ten steps

integrator.run()         #perform dynamics
```

parallel computation of multiple structures

computation of multiple structures with different initial velocities and/or coordinates: gives idea of precision of NMR structure.

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

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.

requirements:

- ability to login to remote nodes via ssh or rsh, without password
- shared filesystem which looks the same to each node

following environment variables set: `XPLOR_NUM_PROCESSES`, `XPLOR_PROCESS`

example script

```
from simulationTools import StructureLoop
from pdbTool import PDBTool

def calcOneStructure( structData ):
    # [ get initial coordinates, randomize velocities ]
    # [ high temp dynamics ]
    # [ cooling loop ]
    # [ final minimization ]
    # [ analysis ]
    filename = structData.makeFilename( outPDBFilename )
    PDBTool(filename).write()

simWorld.setRandomSeed( 785 )
outPDBFilename = '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(numStructures=100,
              structLoopAction=calcOneStructure).run()
```

Using Biowulf

how to get a Biowulf account:
`http://biowulf.nih.gov/user_guide.html#account`

on Biowulf, compute jobs are managed using the PBS queuing system:
`http://biowulf.nih.gov/user_guide.html#q`

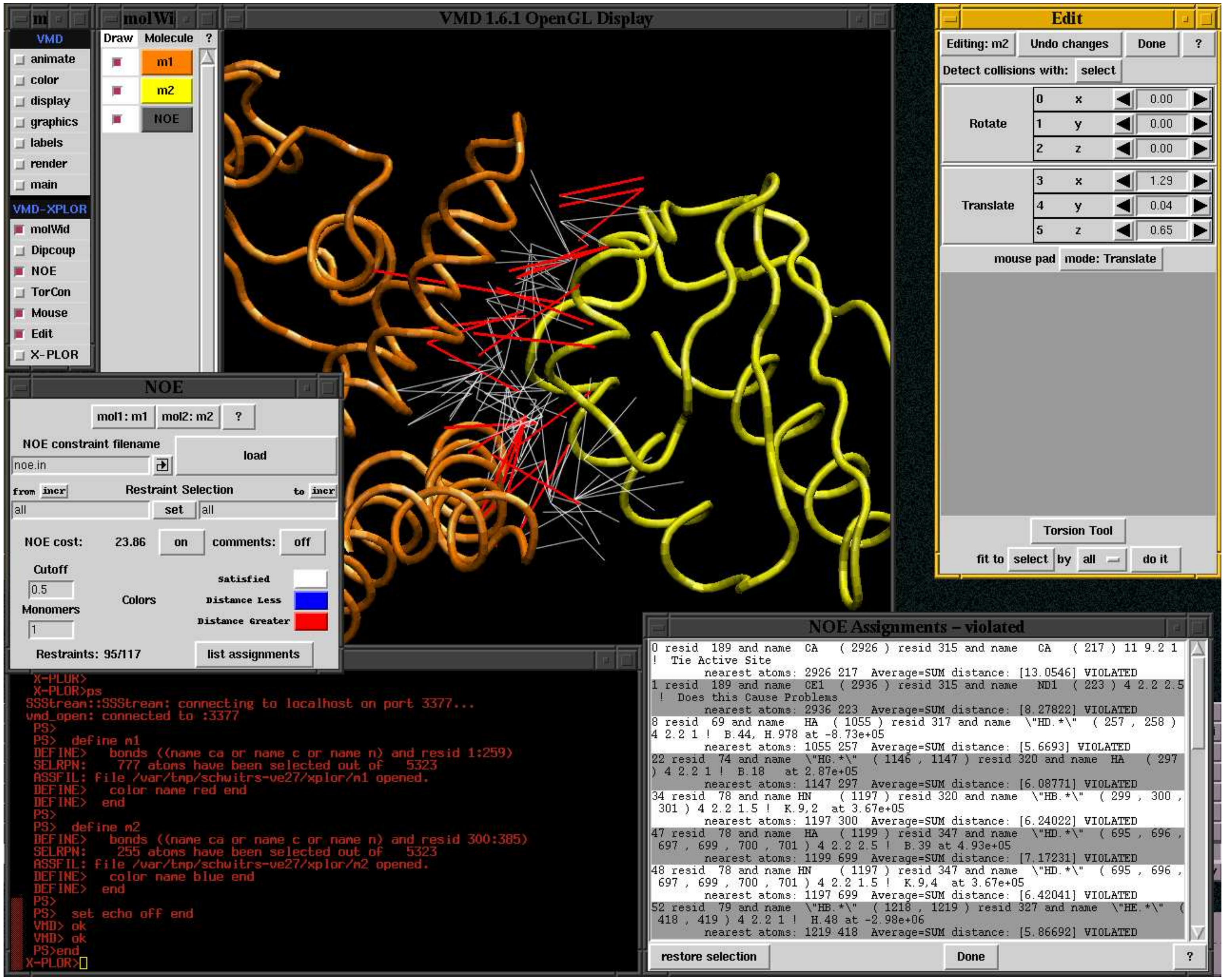
submit jobs using `qsub`:

```
qsub -l nodes=4 xplor.pbs
```

note that each node has two CPUs.

example Biowulf PBS script:
`http://nmr.cit.nih.gov/xplor-nih/nih/xplor.pbs`

VMD interface



Editing: m2
Detect collisions with: select
Rotate: 0 x 0.00; 1 y 0.00; 2 z 0.00
Translate: 3 x 1.29; 4 y 0.04; 5 z 0.65
mouse pad mode: Translate
fit to select by all

mol1: m1 mol2: m2
NOE constraint filename: noe.in
Restraint Selection from incr: all set to incr: all
NOE cost: 23.86 on comments: off
Cutoff: 0.5
Monomers: 1
Colors: Satisfied ⬜ Distance Less 🟦 Distance Greater 🟥
Restraints: 95/117

```
 X-PLOR>
 X-PLOR>ps
SSStream::SSStream: 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>
```

NOE Assignments – violated

```
0 resid  189 and name  CA   ( 2926 ) resid 315 and name   CA   ( 217 ) 11 9.2 1
!  Tie Active Site
        nearest atoms: 2926 217  Average=SUM distance: [13.0546] VIOLATED
1 resid  189 and name  CE1  ( 2936 ) resid 315 and name   ND1  ( 223 ) 4 2.2 2.5
 !  Does this Cause Problems
        nearest atoms: 2936 223  Average=SUM distance: [8.27822] VIOLATED
8 resid  69 and name   HA  ( 1055 ) resid 317 and name  \"HD.*\"  ( 257 , 258 )
4 2.2 1 !  B.44, H.978 at -8.73e+05
        nearest atoms: 1055 257  Average=SUM distance: [5.6693] VIOLATED
22 resid  74 and name  \"HG.*\"  ( 1146 , 1147 ) resid 320 and name  HA   ( 297
) 4 2.2 1 !  B.18   at 2.87e+05
        nearest atoms: 1147 297  Average=SUM distance: [6.08771] VIOLATED
34 resid  78 and name HN    ( 1197 ) resid 320 and name  \"HB.*\"  ( 299 , 300 ,
 301 ) 4 2.2 1.5 !  K.9,2  at 3.67e+05
        nearest atoms: 1197 300  Average=SUM distance: [6.24022] VIOLATED
47 resid  78 and name  HA   ( 1199 ) resid 347 and name  \"HD.*\"  ( 695 , 696 ,
 697 , 699 , 700 , 701 ) 4 2.2 2.5 !  B.39 at 4.93e+05
        nearest atoms: 1199 699  Average=SUM distance: [7.17231] VIOLATED
48 resid  78 and name HN    ( 1197 ) resid 347 and name  \"HD.*\"  ( 695 , 696 ,
 697 , 699 , 700 , 701 ) 4 2.2 1.5 !  K.9,4  at 3.67e+05
        nearest atoms: 1197 699  Average=SUM distance: [6.42041] VIOLATED
52 resid  79 and name  \"HB.*\"  ( 1218 , 1219 ) resid 327 and name  \"HE.*\"  (
 418 , 419 ) 4 2.2 1 !  H.48 at -2.98e+06
        nearest atoms: 1219 418  Average=SUM distance: [5.86692] VIOLATED
```

vmd-xplor screenshot

Use VMD-XPLOR to

- visualize molecular structures
- visualize restraint info
- manually edit structures

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

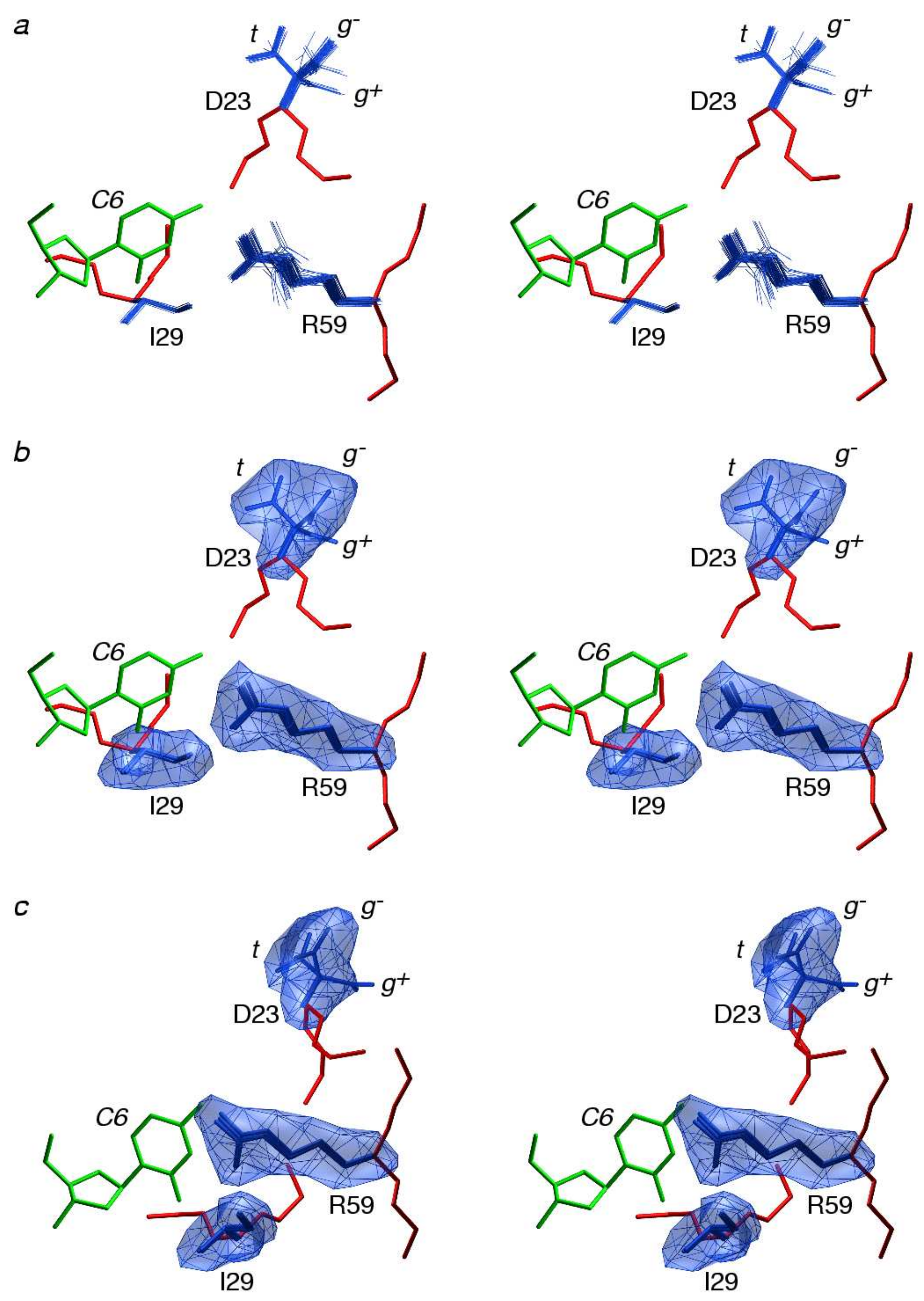
```
% vmd-xplor -port 3359 -noxplor
% xplor -port 3359 -py
```

XPLOR 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



Stereoviews illustrating various representations of the side chains of Asp23, Ile29 and Arg59 of the hnRNPK KH3-ssDNA complex. (a) Superposition of the sidechains of 100 simulated annealing structures (blue). (b) and (c) Isosurface of the reweighted atomic density map for the three side chains drawn at a value of 20% of maximum; within the map, the coordinates of the sidechains of three representative structures are displayed in blue. The view shown in (b) is identical to that in (a). The coordinates of the protein backbone and the C6 nucleotide of the restrained regularized mean structure are shown in red and green, respectively.

Where to go for help

online:

`http://nmr.cit.nih.gov/xplor-nih/`	-	home page
xplor-nih@nmr.cit.nih.gov	-	mailing list
`http://nmr.cit.nih.gov/xplor-nih/faq.html`	-	FAQ
`http://nmr.cit.nih.gov/xplor-nih/xplorMan`	-	XPLOR manual

subdirectories within the xplor distribution:

eginputs	-	newer complete example scripts
tutorial	-	respository of XPLOR scripts
helplib	-	help files
helplib/faq	-	frequently asked questions

Python:

M. Lutz and D. Ascher, "Learning Python," (O'Reilly, 1999).

`http://python.org`

TCL:

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

`http://www.tcl.tk`

Please complain! and suggest!