Python for molecular modeling

Kathryn Loving
Senior Principal Scientist, Schrödinger, Inc.
kathryn.loving@schrödinger.com
Outline

• Overview and motivational examples
• Example code
• Schrodinger Python modules
• Other Useful Python Modules
• PyMOL movies and Python scripting
Motivational Examples

- PyMOL movie
- Virtual screening
- MD trajectory analysis
Python

- Primarily used to send commands to backend programs, and the heavyweight computation and manipulation of chemical structures is not actually done in Python
  - Docking
  - Pharmacophore modeling
  - Cheminformatics
  - Molecular dynamics
- Easy to construct interesting workflows based on existing Python APIs.
  - E.g. Score pharmacophore features based on docking scores.
  - E.g. Run Molecular dynamics simulation, select a “diverse” set of frames from that simulation, and create a Maya animation based on those structures. (Maybe?)
Useful things to know about Python
Write human-readable code!!

• But nobody but me is going to read this code! I’m just trying out an idea to see if it works.
• I don’t have time right now!
• Hey at least unreadable code will give me job security.
import pymol
from pymol import cmd, movie

frames_per_structure = 6
for i in range(0, 10):
    # load structures from Yale Morph Server and create movie
    pdb_file = '/home/armstron/Desktop/demo/ff'+str(i)+'.pdb'
    cmd.load(pdb_file, "mov")
    mset_command = str(i)+' x'+str(frames_per_structure)
    if (i==1):
        cmd.mset(mset_command)
    else:
        cmd.madd(mset_command)

    cmd.show('cartoon','mov')

# add nutations at the end
    cmd.madd('0 x36')
    start_nutate = (9*frames_per_structure)+1
    movie.nutate(start_nutate, start_nutate+35)
    cmd.madd('9 x36')
    start_nutate = start_nutate+36
    movie.nutate(start_nutate, start_nutate+35)
Useful Python modules

- Software repository for Python modules: Python Package Index [http://pypi.python.org/pypi](http://pypi.python.org/pypi)

Browse ➔ Scientific/Engineering ➔ Bio-informatics
Chemistry
cclib  http://cclib.sourceforge.net

• to facilitate the implementation of QM algorithms that are not specific to a particular computational chemistry package

parses and interprets the results of computational chemistry packages:
• ADF
• GAMESS
• Gaussian
• Jaguar
• Molpro
• ORCA

Also does some calculations:
• Mulliken population analysis
• Overlap population analysis
• Calculation of Mayer's bond orders.
PyChem

http://pychem.sourceforge.net/

• A graphical univariate & multivariate analysis package for WinXP and Linux.
• Features principal components analysis (PCA),
• discriminant function analysis (DFA) (also known as canonical variates or correlation analysis - CVA, CCA)
• cluster analysis, including K-means and hierarchical clustering
• Partial least squares (PLS)
• genetic algorithms for feature selection
• optional standalone GUI
RDKit  http://www.rdkit.org/

- Cheminformatics:
  - Substructure searching with SMARTS
  - Canonical SMILES
  - Chirality support
- 2D depiction
- Generation of 2D -> 3D
- Fingerprinting (Daylight-like, “MACCS keys”, etc.)
- Subgraph/Fragment analysis
- Shape-based similarity
- Molecule-molecule alignment
- Molecular descriptors
- Learning:
  - Clustering
  - Decision trees, naïve Bayes*, kNN*
  - Bagging, random forests
- etc...
The Python Molecular Viewer (PMV) is a Python-based GUI that allows customization by the user with Python. PMV has been developed on top of the following independent and re-usable packages: MolKit, DejaVu, and ViewerFramework. This viewer has most of the features usually expected in a molecule viewer:

- stick and cpk representation
- different coloring schemes (by atom, by residue type, by chain, by molecule, by properties, etc...)
- measuring tools
- atom identification by picking
- support for multiple molecules
- secondary structure representation
- user definable sets of atoms, residues, chains and molecules etc...
matplotlib  http://matplotlib.sourceforge.net/

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
- You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc, with just a few lines of code

NetworkX uses matplotlib:

http://networkx.lanl.gov/gallery.html
NumPy and SciPy

NumPy - N-dimensional Array manipulations
• A very commonly used library for scientific computing with Python
• a powerful N-dimensional array object
• basic linear algebra functions
• basic Fourier transforms
• sophisticated random number capabilities

SciPy - Scientific tools for Python
• Open Source library of scientific tools for Python. It depends on the NumPy library, and it gathers a variety of high level science and engineering modules together as a single package

• statistics
• optimization
• numerical integration
• linear algebra
• Fourier transforms
• signal processing
• image processing
• genetic algorithms
• ODE solvers
• special functions
A selection of Schrodinger modules

• Desmond trajectory analysis
  – Molecular dynamics software free for academics: http://www.deshawresearch.com/resources.html
  – Example code on next slide

• Structure manipulation/analysis
  – Make protein residue mutations build.mutate()
  – Search rotamers
  – RMSD

• Set up jobs for docking, QM, Macromodel, etc.
  – Job control interface handles queue submission job.wait()

• Energy analysis/minimization
  – Minimize module requires a license to at least one of: Impact, MacroModel, or Prime

$SCHRODINGER/docs/python/overview.html
http://www.schrodingering.com/ScriptCenter
Desmond trajectory analysis (scroll to see all code)

```python
from schrodinger import structureutil
from schrodinger.trajectory.desmondsimulation import ChorusSimulation

ligand_asl = "mol.num 1" #definition of the ligand; could also detect automatically
protein_asl = "not (mol.num 1)"

csim = ChorusSimulation(input_desmond_cms, trj_directory)
    #get from input

#define atoms based on first frame
frame0 = csim.getFrame(0)
frame0_st = frame0.getStructure()
ligand_atoms = structureutil.evaluate_asl(frame0_st, ligand_asl)
protein_atoms = structureutil.evaluate_asl(frame0_st, protein_asl)

#write out each frame as PDB if it has a new set of protein-ligand hbonds
frame_idx = 0
```
Workflow: similarity-based virtual screen

- Read the query structure and the screening database structures
- Use unique SMILES to remove duplicates from the screening database
- Generate chemical fingerprints
- Compute fingerprint similarity (Tanimoto-similarity criteria for virtual screen)
- Calculate enrichment (if there are known actives in your database)

→ code on next slide
Workflow: similarity-based virtual screen

```python
from schrodinger import structure, structureutil
from schrodinger.application.canvas import fingerprint
from schrodinger.application.canvas import similarity
from schrodinger.utils import log
import sys
import schrodinger.application.canvas.base as canvas

logger = log.get_output_logger("FPExample:")
fp_gen = fingerprint.CanvasFingerprintGenerator( logger=logger )
fp_sim = similarity.CanvasFingerprintSimilarity( logger=logger )

smiles_list = [] #store smiles patterns to detect duplicate structures
for idx, st in enumerate(structure.StructureReader(sys.argv[1])):
    pattern = structureutil.generate_smiles(st)
    if (idx==0):
        # First structure from input file is the query compound
        fp_query = fp_gen.generate(st)
        smiles_list.append(pattern)
    else:
        if not pattern in smiles_list:
            smiles_list.append(pattern)
            fp = fp_gen.generate(st)
            #Calculate and print similarity:
            fp_sim.setMetric("Tanimoto")
            print "Tanimoto Similarity = %5.2f" % fp_sim.calculateSimilarity(fp_query,fp)
            fp_sim.setMetric("Dice")
            print "Dice Similarity = %5.2f" % fp_sim.calculateSimilarity(fp_query,fp)

#Store similarities and take top 1% of database and compute screening enrichment..
```
Wrap your non-Python code as a Python module:

http://www.swig.org/Doc1.3/Python.html#Python
Another useful tool: KNIME  
http://www.knime.org/
KNIME

- Can run a Python script within a KNIME command-line node
- Integrated (free) third-party tools:
  - R
  - Weka
  - Chemistry Development Kit (CDK)
- Several companies develop for KNIME
  - Schrodinger
  - ChemAxon
  - The New Tripos
  - Symyx Technologies
  - Molecular Discovery
  - BioSolveIT
  - CCG
PyMOL

- Viewing 3D Molecular Structures
- Rendering Figures Artistically
- Animating Molecules Dynamically
- Giving Live 3D Presentations

http://www.pymol.org/

- Sharing Interactive Visualizations
- Exporting Geometry in different formats
- Windows, Mac, Linux/UNIX
- Open source: BSDL OSI-approved license

Example PyMOL and Python scripts available on the Wiki:
http://www.pymolwiki.org/index.php/Main_Page
PyMOL: interacting with Python

1. Any command that PyMOL doesn’t recognize will be passed to the Python interpreter. This text could be saved as a python script.py or typed at the PyMOL> command prompt:

   ```python
   f = 10
   for x in range(0,100,10):
       cmd.set("spec_direct_power", float(float(x) / f))
   cmd.png("spec_dir_power" + str(x) + ".png", ray=1)
   ```

2. Create a Python MiniShell:

   ```python
   python
   f = 10
   for x in range(0,100,10):
       cmd.set("spec_direct_power", float(float(x) / f))
   cmd.png("spec_dir_power" + str(x) + ".png", ray=1)
   python end
   ```

   #note, Python still knows about variables set within the minishell
   print f
   ```
PyMOL

http://www.pymol.org/

Good set of example scripts:

http://www.pymolwiki.org/index.php/Simple_Scripting

```python
from pymol import cmd, stored

def yourFunction( arg1, arg2 ):
    # # Your code goes here #
    print "Hello, PyMOLers"
    print "You passed in %s and %s" % (arg1, arg2)
    print "I will return them to you in a list. Here you go."
    return (arg1, arg2)

cmd.extend( "yourFunction", yourFunction );
```
PyMOL

Good set of example scripts:

http://www.pymolwiki.org/index.php/Simple_Scripting

from pymol import cmd, stored

def yourFunction( arg1, arg2 ):
    # # Your code goes here #
    print "Hello, PyMOLers"
    print "You passed in %s and %s" % (arg1, arg2)
    print "I will return them to you in a list. Here you go."
    return (arg1, arg2)

cmd.extend( "yourFunction", yourFunction );

http://www.pymol.org/

cmd module:

Call PyMOL commands. There is no real PyMOL Reference Manual that's recent (Google for "pymol refman" for the old one). 95% of the commands are documented on the PyMOLWiki; and, you can find them by just searching for the command name.

Inside of PyMOL try:
PyMOL> help commandName
PyMOL> commandName ?
for more help on a command.
PyMOL

Good set of example scripts:

http://www.pymolwiki.org/index.php/Simple_Scripting

from pymol import cmd, stored

def yourFunction( arg1, arg2 ):
    # Your code goes here #
    print "Hello, PyMOLers"
    print "You passed in %s and %s" % (arg1, arg2)
    print "I will return them to you in a list. Here you go." 
    return (arg1, arg2)

    cmd.extend( "yourFunction", yourFunction );

stored module:

Get data from PyMOL into your script and from your script back into PyMOL.

For example, the alter/iterate commands will apply some string/function to each atom in a selection you specify.

# print out the coordinates for alpha carbons
cmd.iterate( "n. CA", "print [x,y,z]"

Now, if I wanted to save those coordinates, cmd.iterate("n. CA", "myList.append( [x,y,z] )") doesn't work because "myList" is inside a string. How are we to save myList and pass it to PyMOL inside the string? You use pymol.stored:

from pymol import stored
stored.myList = []

    cmd.iterate("n. CA", "stored.myList.append ([x,y,z])")

print stored.myList
from pymol import cmd, stored

def yourFunction( arg1, arg2):
    # Your code goes here
    print "Hello, PyMOLers"
    print "You passed in %s and %s" % (arg1, arg2)
    print "I will return them to you in a list. Here you go."
    return (arg1, arg2)

    cmd.extend( "yourFunction", yourFunction );

extend command:
Typically the last line of a Python script. Use this when you write a function and want PyMOL to recognize the new command.

from pymol import cmd
def foo(moo=2):
    print moo
    cmd.extend('foo', foo)

Run this script foo.py in PyMOL:
PyMOL>foo
2
PyMOL>foo 3
3
PyMOL>foo ?
Usage: foo [ moo ]
PyMOL “movies”  
http://www.pymol.org/

cmd and stored modules will cover 95% of scripting, but there are others, especially useful for movies:

```python
import epymol  # new, for incentive features
from epymol import rigimol
import util
import movie
```

In PyMOL's source go to "pymol/modules/pymol" to see more possibilities.

Free alternative to rigimol:

The Yale Morph Server

http://molmovdb.mbb.yale.edu/molmovdb/morph/
PyMOL “movies”

http://www.pymol.org/

Option 1:
Save “scenes” within PyMOL (can only run within PyMOL).

Option 2:
- Script movie either using Python or PyMOL GUI commands.
- Save images as a movie.
Save depends on the OS:
MacPyMOL – save directly as a QuickTime movie
Linux – save as a .mpg movie
Windows – save .png images and convert into a movie with e.g. ImageMagick

Option 3:
AxFPyMOL (next slide)
AxPyMOL http://pymolwiki.org/index.php/Axpymol
util:
util.mrock(start, finish, angle, phase, loop-flag)
util.mroll(start, finish, loop-flag)

Here a static structure is subject to a gentle rock. The following statements create a sixty frame movie which simply rocks the protein by 10 degrees.

load test/dat/pept.pdb  # load a structure
mset 1 x60  # define the movie
util.mrock(1, 60, 10, 1, 1)
# issues mdo commands to create +/- 10 deg. rock over 60 frames
# e.g.
# mdo 1: turn y, 10;

In this next example, the protein is rotated through a full 360 sweep about the Y-axis over 120 frames:

load test/dat/pept.pdb  # load a structure
mset 1 x120  # define the movie
util.mroll(1, 120, 1)
# issues mdo commands to create full rotation over 120 frames
movie.rock(first, last, angle=30, phase=0, loop=1, axis='y')
movie.roll(first, last, loop=1, axis='y')
movie.zoom(first, last, step=1, loop=1, axis='z')
movie.screw(first, last, step=1, angle=30, phase=0, loop=1, axis='y')
movie.sweep(pause=0, cycles=1)
movie.pause(pause=15, cycles=1)
movie.nutate(first, last, angle=30, phase=0, loop=1, shift=math.pi/2.0, factor=0.01)
movie.tdroll(first, rangex, rangey, rangez, skip=1)
movie.timed_roll(period=12.0, cycles=1, axis='y')
movie.load(*args, **kw)
example_movie.py

```python
import pymol
from pymol import cmd, movie

frames_per_structure = 6
for i in range(0, 10):
    # load structures from Yale Morph Server and create movie
    pdb_file = '/home/armstron/Desktop/demo/ff'+str(i)+'.pdb'
    cmd.load(pdb_file, "mov")
    mset_command = str(i)+' x'+str(frames_per_structure)
    if (i==1):
        cmd.mset(mset_command)
    else:
        cmd.madd(mset_command)

    cmd.show('cartoon','mov')

# add nutations at the end
    cmd.madd('0 x36')
    # start_nutate = (9*frames_per_structure)+1
    # movie.nutate(start_nutate, start_nutate+35)
    cmd.madd('9 x36')
    # start_nutate = start_nutate+36
    # movie.nutate(start_nutate, start_nutate+35)
    movie.nutate(0,132)
```

1. Type this in your favorite text editor
2. Save as example_movie.py
3. In PyMOL: run example_movie.py
Other ways to modify example script:

viewport 320,240  #change size from default 640x480
set ray_trace_frames=1  #ray-trace
set cache_frames=0  #also required for ray-tracing
mclear  #clear already-set mset commands
PyMOL

If time, switch to Linux and demonstrate:

Scenes

PyMOL source e.g. preset.py

http://www.pymol.org/
import sys, subprocess

#generate an image for each frame of your trajectory with:
#maestro.command( "saveimage %s" % filename )
#OR mpng MyMovie in PyMOL
#(this command will create MyMovie0001.png, MyMovie0002.png etc.)
#save all image file names in array “filenames”

output_filename = "my_movie.gif"
args = ["convert", "-delay", "10"]
for image_file in filenames:
    args.append(image_file)
args.append(output_filename)
try:
    return_code = subprocess.call(args)
except:
    print “ImageMagick may not be installed”
sys.exit(1)
Thanks!

- Daniel Panne
- Piotr Sliz
- Ruth Hazlewood

- Comment from the class: check out emovie
  http://www.weizmann.ac.il/ISPC/eMovie.html

Schrodinger is hiring for research positions:
http://www.schrodinger.com -> About -> Careers