How To Make the Most of PyRosetta

Jason Labonte & Michael Pacella
Gray Lab
RosettaCON 2012
Outline

• Custom Rosetta classes in PyRosetta
  – Movers/Protocols
  – Scoring methods

• Tips for better PyRosetta scripts
  – Tricky Python subtleties
    • `import` statements
    • Passing objects to methods in Python
  – Standard Rosetta things that work differently in PyRosetta
    • `PyJobDistributor`
    • Passing & parsing Rosetta option flags
  – Where else to go for help
    • Printing Rosetta objects
    • Demos & test scripts

• How-to
  – RNA
  – NMR constraints
  – Symmetry operations
  – Non-canonical AAs & ligands
CUSTOM ROSETTA CLASSES
Classes in Python: 
__init__() method

class MyShape:
    ""
    This is how one defines a class.
    Within the class statement block go its methods....
    ""
    def __init__(self):
        ""
        This method is called when instantiating a MyShape object.
        (In the method declaration above, note the use of "self". "self" refers to the particular instance of MyShape that is running the code. When one calls a method, the first argument passed to that argument is always the instance of the class calling the method.
        ""
        self.color = 0  # Sets default value for color.  (Again, note # the use of "self".)
Classes in Python: __str__() method

def __str__(self):
    
    """This method determines what string is printed if you print the object.
    """
    return self.__doc__
Classes in Python: Example Methods

```python
def area(self):
    """
    Output the area of the shape.
    """
    return  # Code to calculate the area goes here.

def draw(self, line_width = 1):
    """
    Draw the shape on the screen.
    (Note how "line_width" is given a default value. This is how one overloads a function in Python.)
    """
    pass  # Code to draw the shape goes here.
```
Classes in Python: Inheritance

class MyCircle(MyShape):
    
    This is how one defines a subclass.
    The parent class from which this class inherits all its methods
    goes in the parentheses. We don't need to write the
    __init__, __str__, area, or draw methods, but we can if we
    want to.
    
    radius = 1.0  # Defines a property of MyCircle.

    def area(self):
        
        Output the area of the circle.
        This overrides the "area" method inherited from MyShape.
        
        return math.pi * self.radius**2
Classes in Python: Instantiation & Usage

```python
>>> circle = MyCircle()
>>> print circle
This is how one defines a subclass.
The parent class from which this class inherits all its methods goes in the parentheses. We don't need to write the __init__, __str__, area, or draw methods, but we can if we want to.
>>> print circle.color
0
>>> print circle.area()
3.14159265359
>>> circle.draw()    # Draws a circle of radius 1.0 in color 0 with a line width of 1.
>>> circle.radius, circle.color = 1.5, 3
>>> circle.draw(2)   # Draws a circle of radius 1.5 in color 3 with line width of 2.
```
Movers in PyRosetta: Required Methods

```python
class PhiNByXDegreesMover(rosetta.protocols.moves.Mover):
    """This mover increments the phi angle of residue N by X degrees."
    """
def __init__(self, N_in = 1, X_in = 15):
    # We must run Mover's __init__() method for our custom
    # Mover to act as a true Rosetta Mover.
    rosetta.protocols.moves.Mover.__init__(self)
    self.N = N_in
    self.X = X_in

def __str__(self):
    return self.get_name() + \
    "\nresidue: " + str(self.N) + \
    " phi increment: " + str(self.X) + " degrees"
```
def get_name(self):
    
    """
    Return the name of the class of the object instance, in this case, "PhiNByXDegreesMover".
    All Movers MUST include this method.
    """
    return self.__class__.__name__

def apply(self, pose):
    
    """
    Apply a move to pose.
    All Movers MUST include this method.
    """
Decorators in Python:
Definition

• Decorators are essentially functions that take a class or method as input and return a modified (“decorated”) version of that class or method.

• If we have a class, `MyCircle`, and a decorator function —

```python
def hollow(shape_in):
    """Modifies the draw() method of an input shape class to output a hollow shape.""

@hollow
class MyCircle(MyShape):
circle = MyCircle()
```

— then the “wrapper syntax”...

```python
...results in the same object, `circle`, as...
class MyCircle(MyShape):
circle = hollow(MyCircle())
```
import rosetta.core.scoring.methods as methods  # Alias for the namespace

@rosetta.EnergyMethod()  # An EnergyMethod object is a callable function.
class LengthScoreMethod(methods.ContextIndependentOneBodyEnergy):
    
    A scoring method that favors longer peptides by assigning one Rosetta energy unit per residue.
    
    def __init__(self):
        methods.ContextIndependentOneBodyEnergy.__init__(self, self.creator())
        # (Since the decorator is applied at the definition of the class, 
        # the class method creator(), which is made by the function 
        # EnergyMethod()(), is there at the time when LengthScoreMethod is 
        # instantiated.)

    def residue_energy(self, res, pose, emap):
        emap.get().set(self.scoreType, 1.0)  # 1 energy unit per residue
Scoring Methods in PyRosetta: Usage

```python
>>> from rosetta import *
>>> from scoring_methods import *  # Assumes your LengthScoreMethod class is in the module scoring_methods.py.
>>> init()

>>> pose = pose_from_sequence("ACDEFGHI")

>>> sf = ScoreFunction()
>>> print sf(pose)
0.0

>>> len_score = LengthScoreMethod.scoreType  # Extracts the ScoreType from your custom scoring method.
>>> sf.set_weight(len_score, 1)  # Sets the weight of your custom len_score component to 1.
>>> print sf(pose)
8.0
```
Scoring Methods in PyRosetta: Context-Independent, 2-Body

```python
@rosetta.EnergyMethod()
class CI2BScoreMethod(methods.ContextIndependentTwoBodyEnergy):
    def __init__(self):
        methods.ContextIndependentTwoBodyEnergy.__init__(self, self.creator())

    def residue_pair_energy(self, res1, res2, pose, sf, emap):
        score = 1.0  # A real method would calculate a value from res1 and res2
        emap.get().set(self.scoreType, score)

    def atomic_interaction_cutoff(self):
        return 0.0

    def defines_intrares_energy(self, weights):
        return True

    def eval_intrares_energy(self, res, pose, sf, emap):
        pass
```
Scoring Methods in PyRosetta: Context-Dependent, 2-Body

@rosetta.EnergyMethod()
class CD2BScoreMethod(methods.ContextDependentTwoBodyEnergy):
    def __init__(self):
        methods.ContextDependentTwoBodyEnergy.__init__(self, self.creator())

    def residue_pair_energy(self, res1, res2, pose, sf, emap):
        score = 1.0  # A real method would calculate a value from res1 and res2
        emap.get().set(self.scoreType, score)

    def atomic_interaction_cutoff(self):
        return 0.0

    def defines_intrares_energy(self, weights):
        return True

    def eval_intrares_energy(self, res, pose, sf, emap):
        pass
Import Statements in Python

- **import** module
  - imports the namespace `module` from `module.py`
  - runs `module.py`
  - allows one to call `MyClass` & `my_method` using `module.MyClass()` & `module.my_method()`
- **from** module **import** MyClass, my_method
  - does *not* import the namespace `module`
  - does *not* run `module.py`
  - allows one to call `MyClass` & `my_method` using `MyClass()` & `my_method()`

- When you use **from** rosetta **import** *, it does *not* import all classes and methods from Rosetta.
Argument Passing in Python: By Value

def my_method(argument):
    argument += 1
    return argument

number = 1  # number is a "primitive type".
my_method(number)
print number  # This will print "1".
print my_method(number)  # This will print "2".
Argument Passing in Python: By Reference

def my_method(argument):
    argument.set_phi(1, 180)
    return argument.phi(1)

pose = pose_from_sequence("AAAAA")  # pose is an object.
my_method(pose)
print pose.phi(1)  # This will print "180".
print my_method(pose)  # This will also print "180".
Job Distribution in PyRosetta:
PyJobDistributor

jd = PyJobDistributor("filename", nstruct, sf)
# The above constructs a job distributor that will create nstruct decoys
# named filename_1.pdb to filename_N.pdb and a score file, filename.fasc.
# The PyJobDistributor will not overwrite a file already in existence.
# When initialized, the next available output file is started as an in-
# progress file.

jd.native_pose = native_pose
# If a native pose is provided, a column of RMSDs will be included in the
# score file.

while not jd.job_complete:
    pose.assign(start_pose)
    my_protocol.apply(pose)
    jd.output_decoy(pose)
    # Outputs the next decoy, deletes the in-progress file, and creates the
    # next available in-progress file.
Job Distribution in PyRosetta: Example with PBS

The portable batch system (pbs) script:

```
#!/bin/bash -f
#PBS -M my_name@gmail.com
#PBS -m e
#PBS -l nodes=1:ppn=1
#PBS -l mem=1024mb
#PBS -l walltime=1:00:00
#PBS -l cput=1:00:00
#PBS -j oe
#PBS -q batch
source ~/Applications/PyRosetta/SetPyRosettaEnvironment.sh
cd $PBS_O_WORKDIR
python2.6 relax.py
```

How to submit:

```
$ qsub relax.pbs
```
Option Flags in PyRosetta

Recommended Route: Defining “extra_options”

- This will add-on to a default list of options:
  - -database
  - -ex1
  - -ex2aro

```python
init(extra_options = "-mute basic -mute core -mute protocols")
```
Option Flags in PyRosetta
Alternate Route: Creating a Custom “args” List

• This allows you to fully customize the command line options passed to PyRosetta.

• `app` and `-database /path/to/database` must be included.

```python
opts = ['"app", "-database /path/to/database", "-ex1", "-ex2aro", "-symmetry:symmetry_definition symm_def.dat"]
args = utility.vector1_string()
args.extend(opts)
init(args)
```
Printing Objects in PyRosetta

• The Gray Lab has methodically been going through classes in the Rosetta library and adding print functionality.

• E.g.:

```python
>>> min_mover = MinMover()
>>> print min_mover
Mover name: MinMover, Mover type: MinMover, Mover current tag: NoTag
Minimization type: linmin, Score tolerance: 0.01, Nb list: 1, Deriv check: 0
```
Demos & Test Scripts

• A large selection of demos can be found in your PyRosetta install directory in the /test folder.
HOW-TO
RNA in PyRosetta: To Do Beforehand

- pdb files with RNA must be in a special format to be imported into Rosetta.
  - Residue names GUA (G), ADE (A), CYT (C), & URA/URI (U) must be changed to rG, rA, rC, & rU, respectively, so that Rosetta knows they have ribose, not deoxyribose, rings.
  - A handy script, *make_rna_rosetta_ready.py*, has been written to do this for you.
RNA in PyRosetta: Sample Code

# Create residue type set for RNA.
rna_set = ChemicalManager.get_instance().residue_type_set("rna").get()

# Load pose.
pose = pose_from_pdb(rna_set, "filename.pdb")

# RNA has different torsion angles....
print pose.gamma(1)  # 1 is the residue number.
print pose.delta(1)
print pose.epsilon(1)
print pose.chi(1)
print pose.zeta(1)

# Construct an RNA score function.
sf = create_score_function("rna_hires")
# Import RNA movers and protocols.
from rosetta.protocols.rna import *

# Construct an RNA minimization mover.
min_mover = RNA_Minimizer()

# Minimize the pose.
min_mover.apply(pose)
NMR Constraints in PyRosetta: ConstraintSetMover

# Construct constraint set mover.
set_constraints = ConstraintSetMover()
set_constraints.constraint_file("filename.cst")

# Prepare scorefunction.
sf = create_score_function("standard")
sf.set_weight(atom_pair_constraint, 1.0)

# Set constraints into pose.
set_constraints.apply(pose)

# Score the pose.
sf.show(pose)
NMR Constraints in PyRosetta: 
List of Constraint Scoring Components

- atom_pair_constraint
- constant_constraint
- coordinate_constraint
- angle_constraint
- dihedral_constraint
Symmetry in PyRosetta: To Do Beforehand

• prepare a pdb of the “master” subunit
• prepare a symmetry definition file
• include `--symmetry:symmetry_definition name_of_symm_def_file.dat` in your args
Symmetry in PyRosetta: Sample Code

# Extra import statements are necessary.
import rosetta.core.conformation.symmetry
import rosetta.core.pose.symmetry
import rosetta.core.scoring.symmetry
import rosetta.protocols.simple_moves.symmetry

# Create a symmetric pose.
def symmetrize_pose(pose):
    pose_symm_data = core.conformation.symmetry.SymmData(pose.n_residue(), pose.num_jump())
    pose_symm_data.read_symmetry_data_from_file("sym_def_file.dat")
    core.pose.symmetry.make_symmetric_pose(pose, pose_symm_data)

    # Many other useful utility functions are in core.pose.symmetry.
Symmetry in PyRosetta: Sample Code

# Create a symmetric score function.
sym_sfxn = core.scoring.symmetry.SymmetricScoreFunction()

# Create a symmetric pack rotamers mover.
sym_packer = protocols.simple_moves.symmetry.SymPackRotamersMover(sym_sfxn, task)

# Create a symmetric min mover.
sym_min_mover = protocols.simple_moves.symmetry.SymMinMover()

# Create a symmetric move map.
move_map = MoveMap()
core.pose.symmetry.make_symmetric_movemap(pose, move_map)

# Many other useful movers are in protocols.simple_moves.symmetry.
Custom Parameter Files in PyRosetta

To Do Beforehand

• Obtain an .mdl-formatted file of your residue’s geometry. (OpenBabel is great for converting formats on chemical structures.)

• Run molfile_to_params.py to convert to a Rosetta-readable .params file
Custom Parameter Files in PyRosetta: Sample Code

# Create a vector1 of paths to your extra .params files you want loaded.
params_paths = utility.vector1_string(
    "list", "of", "paths", "to", "extra", "params")

# Create a non-standard ResidueTypeSet that includes your extra .params.
nonstandard_residue_set = generate_nonstandard_residue_set(params_paths)

# Use this ResidueTypeSet when loading your pdb w/ non-standard residues.
pose = pose_from_pdb(nonstandard_residue_set, "nonstandard.pdb")
Custom Parameter Files in PyRosetta: Another Option

• A more permanent route (though inappropriate for check-ins) is to add your new .params file to the chemical database.

• You will also need to specify the path in residue_types.txt (also in the database) and ensure it is not commented out.
Special Thanks

• Will Sheffler
• Jeff Gray
• Sid Chaudhury
• Sergey Lyskov
• Evan Baugh
• Boon Uranukul
• Gray Lab
• you, for listening