ROSETTACARBOHYDRATE
Tutorial/Demo 1
Residues & Poses

Linear Oligosaccharides & IUPAC Sequences

1. Start up IPython and import PyRosetta.

```
$ ipython
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
Type "copyright", "credits" or "license" for more information.

IPython 1.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from rosetta import *
```

2. The `-include_sugars` flag must be used when modeling carbohydrates with Rosetta.

```
In [2]: init('-include_sugars')
PYROSETTA_DATABASE environment variable was set to:
      /usr/local/lib/PyRosetta/database; using it....
PyRosetta 2014 [Rosetta 2014 unknown:0cae3092782a6dfc70e10d190cc779187406545f]
      retrieved from:
(C) Copyright Rosetta Commons Member Institutions.
Created in JHU by Sergey Lyskov and PyRosetta Team.

core.init: Rosetta version  from
core.init: command: PyRosetta -include_sugars -database
      /usr/local/lib/PyRosetta/database
core.init: 'RNG device' seed mode, using '/dev/urandom', seed=-1985974028
      seed_offset=0 real_seed=-1985974028
core.init.random: RandomGenerator:init: Normal mode, seed=-1985974028
      RG_type=mt19937
```

3. Set up the `PyMOL_Mover` for viewing structures.

```
In [3]: pm = PyMOL_Mover()
```

4. Rosetta can create both linear and branched oligosaccharides from an IUPAC sequence. Use the function, `pose_from_saccharide_sequence()`, which must be imported from the `core.pose` namespace.

```
In [4]: from rosetta.core.pose import pose_from_saccharide_sequence
```

To properly build a linear oligosaccharide, Rosetta must know the following details about each sugar residue being created in the following order:

- Main-chain connectivity — →2) (`->2)`), →4) (`->4)`), →6) (`->6)`), *etc.*; default value is `->4)-`
- Anomeric form — α (`a` or `alpha`) or β (`b` or `beta`); default value is `alpha`
- Enantiomeric form — L (`L`) or D (`D`); default value is `D`
- 3-Letter code — required; uses sentence case
- Ring form code — *f* (for a furanose/5-membered ring), *p* (for a pyranose/6-membered ring); required

Residues must be separated by hyphens. Glycosidic linkages can be specified with full IUPAC notation, *e.g.*, `-(1->4)-` for "-(1→4)-". Rosetta will assume `-(1->` for aldoses and `-(2->` for ketoses. Note that the standard is to write the IUPAC sequence of a saccharide chain in reverse order from how they are numbered.

```
In [5]: glucose = pose_from_saccharide_sequence('alpha-D-Glcp')
core.chemical.ResidueTypeSet: Finished initializing fa_standard residue type
      set.  Created 414 residue types
core.chemical.ResidueTypeSet: Total time to initialize 0.658687 seconds.

In [6]: galactose = pose_from_saccharide_sequence('Galp')

In [7]: mannose = pose_from_saccharide_sequence('->3)-a-D-Manp')

In [8]: maltotriose = pose_from_saccharide_sequence('a-D-Glcp-' * 3)

In [9]: isomaltose = pose_from_saccharide_sequence('->6)-Glcp-' * 2)

In [10]: lactose = pose_from_saccharide_sequence('b-D-Galp-(1->4)-a-D-Glcp')

In [11]: pm.apply(isomaltose)

In [12]: pm.apply(glucose)

In [13]: pm.apply(galactose)
```

5. When you print a `Pose` containing carbohydrate residues, the sugar residues will be listed as `Z` in the sequence.

```
In [14]: print maltotriose
PDB file name: alpha-D-Glcp-(1->4)-alpha-D-Glcp-(1->4)-D-Glcp
Total residues:3
Sequence: ZZZ
Fold tree:
FOLD_TREE  EDGE 1 3 -1

In [15]: print isomaltose
PDB file name: alpha-D-Glcp-(1->6)- D-Glcp
Total residues:2
Sequence: ZZ
Fold tree:
FOLD_TREE  EDGE 1 2 -1
```

```
In [16]: print lactose
PDB file name: beta-D-Galp-(1->4)- D-Glcp
Total residues:2
Sequence: ZZ
Fold tree:
FOLD_TREE  EDGE 1 2 -1
```

However, you can have Rosetta print out the sequences for individual chains, using the `chain_sequence()` method. If you do this, Rosetta is smart enough to give you a distinct sequence format for saccharide chains. (You may have noticed that the default file name for a `.pdb` file created from this `Pose` will be the same sequence.)

```
In [17]: print maltotriose.chain_sequence(1)
alpha-D-Glcp-(1->4)-alpha-D-Glcp-(1->4)-D-Glcp

In [18]: print isomaltose.chain_sequence(1)
alpha-D-Glcp-(1->6)-D-Glcp

In [19]: print lactose.chain_sequence(1)
beta-D-Galp-(1->4)-D-Glcp
```

Again, the standard is to show the sequence of a saccharide chain in reverse order from how they are numbered.

```
In [20]: for res in lactose: print res.seqpos(), res.name()
1 ->4)-alpha-D-Glcp:reducing_end
2 ->4)-beta-D-Galp:non-reducing_end

In [21]: for res in maltotriose: print res.seqpos(), res.name()
1 ->4)-alpha-D-Glcp:reducing_end
2 ->4)-alpha-D-Glcp
3 ->4)-alpha-D-Glcp:non-reducing_end
```

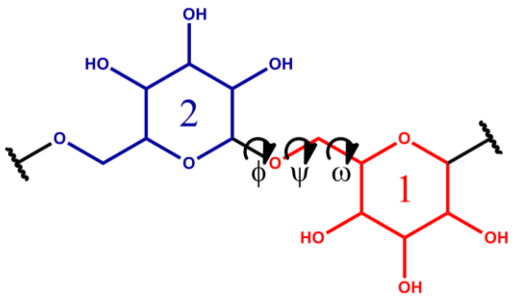
6. Rosetta stores carbohydrate-specific information within `ResidueType`. If you print a residue, this additional information will be displayed.

```
In [22]: print glucose.residue(1)
Residue 1: ->4)-alpha-D-Glcp:reducing_end:non-reducing_end (Glc, Z):
Base: ->4)-alpha-D-Glcp
 Properties: POLYMER CARBOHYDRATE LOWER_TERMINUS UPPER_TERMINUS POLAR CYCLIC
      HEXOSE ALDOSE D_SUGAR PYRANOSE ALPHA_SUGAR
 Variant types: UPPER_TERMINUS_VARIANT LOWER_TERMINUS_VARIANT
 Main-chain atoms:  C1   C2   C3   C4   O4
 Backbone atoms:    C1   C2   C3   C4   O4   C5   O5   VO5  VC1  H1   H2   H3
      H4   HO4  H5
 Side-chain atoms:  O1   O2   O3   C6   O6   HO1  HO2  HO3 1H6  2H6   HO6
Carbohydrate Properties for this Residue:
 Basic Name: glucose
 IUPAC Name: D-glucopyranose
 Abbreviation: D-Glcp
 Classification: aldohexose
 Stereochemistry: D
 Ring Form: pyranose
 Anomeric Form: alpha
 Modifications:
  none
 Polymeric Information:
  Main chain connection: N/A
  Branch connections: none
Ring Conformer: 4C1 (chair): C-P parameters (q, phi, theta): 0.55, 180, 0; nu
       angles (degrees): 60, -60, 60, -60, 60
  O1 : axial
  O2 : equatorial
  O3 : equatorial
  O4 : equatorial
  C6 : equatorial
...

In [23]: print galactose.residue(1)
...

In [24]: print mannose.residue(1)
...
```

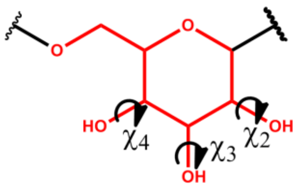
Torsion Angles, PDB File HETNAM Records, & RingConformers

The torsion angles of sugars are as follows:

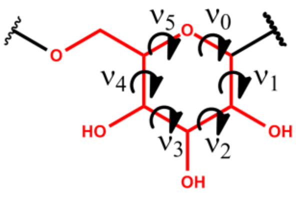
- ϕ — The 1st glycosidic torsion back to the *previous* (*n*-1) residue. The angle is defined by the cyclic oxygen, the two atoms across the bond, and the cyclic carbon numbered one less than the glycosidic linkage position. For aldopyranoses, ϕ(*n*) is thus defined as O5(*n*)–C1(*n*)–O*X*(*n*-1)–C*X*(*n*-1) , where *X* is the position of the glycosidic linkage. For aldofuranoses, ϕ(*n*) is defined as O4(*n*)–C1(*n*)–O*X*(*n*-1)–C*X*(*n*-1) For 2-ketopyranoses, ϕ(*n*) is defined as O6(*n*)–C2(*n*)-O*X*(*n*-1)-C*X*(*n*-1). For 2-ketofuranoses, ϕ(*n*) is defined as O5(*n*)–C2(*n*)-O*X*(*n*-1)–C*X*(*n*-1). *Et cetera*….



- ψ — The 2nd glycosidic torsion back to the *previous* (*n*-1) residue. The angle is defined by the anomeric carbon, the two atoms across the bond, and the cyclic carbon numbered two less than the glycosidic linkage position. ψ(n) is thus defined as $C_{anomeric}$(*n*)–O*X*(*n*-1)–C*X*(*n*-1)–C*X*-1(*n*-1), where *X* is the position of the glycosidic linkage.
- ω — The 3rd (and any subsequent) glycosidic torsion(s) back to the *previous* residue. ω_1(*n*) is defined as O*X*(*n*-1)–C*X*(*n*-1)–C*X*-1(*n*-1)–C*X*-2(*n*-1), where *X* is the position of the glycosidic linkage. (This only applies to sugars with exocyclic connectivities.)



- ν_1–ν_n — The internal ring torsion angles, where *n* is the number of atoms in the ring. ν_1 defines the torsion across bond C1–C2, *etc.*
- χ_1–χ_n — The side-chain torsion angles, where *n* is the number of carbons in the sugar residue. The angle is defined by the carbon one less than the glycosidic linkage position, the two atoms across the bond, and the polar hydrogen. The cyclic ring counts as carbon 0. For an aldopyranose, χ_1 is thus defined by O5–C1–O1–HO1, and χ_2 is defined by C1–C2–O2–HO2. χ 5 is d+6efined by C4–C5–C6–O6,



because it rotates the exocyclic carbon rather than twist the ring. χ_6 is defined by C5–C6–O6–HO6.

Take special note of how ϕ, ψ, and ω are defined in the reverse order as the same angles for amino acid residues!

1. The `chi()` method of `Pose` works with sugar residues in the same way it works with amino acid residues, where the first argument is the χ subscript and the second is the residue number of the `Pose`.

```
In [25]: galactose.chi(1, 1)
Out[25]: -61.91429425427617

In [26]: galactose.chi(2, 1)
Out[26]: -179.445405

In [27]: galactose.chi(3, 1)
Out[27]: 178.810585

In [28]: galactose.chi(4, 1)
Out[28]: -120.00001100000001

In [29]: galactose.chi(5, 1)
Out[29]: -62.79708300000001

In [30]: galactose.chi(6, 1)
Out[30]: -176.382127
```

2. Likewise, we can use `set_chi()` to change these torsion angles and observe the changes in PyMOL, setting the option to keep history to true.

```
In [31]: observer = PyMOL_Observer(galactose, True)

In [32]: galactose.set_chi(1, 1, 180)

In [33]: galactose.set_chi(2, 1, 60)

In [34]: galactose.set_chi(3, 1, 60)

In [35]: galactose.set_chi(4, 1, 0)

In [36]: galactose.set_chi(5, 1, 60)

In [37]: galactose.set_chi(6, 1, -60)
```

3. The `phi()`, `set_phi()`, `psi()`, `set_psi()`, `omega()`, and `set_omega()` methods of `Pose` also work with sugars. However, since `pose_from_saccharide_sequence()` creates a `Pose` with non-ideal angles, larger oligomers may appear all jumbled together. Instead, let's reload some `Poses` from PDB files.

   Rosetta uses a slightly modified PDB format for importing carbohydrate residues—although the `-alternate_3_letter_codes pdb_sugar` flag can get around this issue in some cases. (See https://www.rosettacommons.org/docs/wiki/rosetta_basics/preparation/Preparing-PDB-files-for-non-peptide-polymers for more information.) The key difference in formats involves the `HETNAM` record of the PDB format. The standard PDB `HETNAM` record line:

   ```
   HETNAM     GLC ALPHA-D-GLUCOSE
   ```

   …means that all `GLC` 3-letter codes in the entire file are α-D-glucose, which is insufficient, as this could mean several different α-D-glucoses, depending on the main chain of the glycan—and many, many more if one includes modified sugars! The modified Rosetta-ready PDB `HETNAM` record line:

   ```
   HETNAM     Glc A   1  ->4)-alpha-D-Glcp
   ```

   … means that the `GLC` residue at position A1 requires the `->4)-alpha-D-Glcp` `ResidueType` or any of its `VariantTypes`. (Note also that Rosetta uses sentence case 3-letter-codes for sugars.)

   The `test/data/carbohydrates/` folder of PyRosetta includes a collection of PDB files for oligosaccharides. Change to that directory and reload maltotriose and isomaltose from PDB files.

   ```
   In [38]: cd test/data/carbohydrates/
   ...

   In [39]: maltotriose = pose_from_file('maltotriose.pdb')

   In [40]: isomaltose = pose_from_file('isomaltose.pdb')
   ```

   Now try out the torsion angle getters and setters.

   ```
   In [41]: pm.apply(maltotriose)

   In [42]: maltotriose.phi(1)
   core.pose.carbohydrates.util: Glycosidic torsions are undefined for the first
         polysaccharide residue of a chain unless part of a branch.
   core.pose.carbohydrates.util: Returning zero.
   Out[42]: 0.0

   In [43]: maltotriose.psi(1)
   core.pose.carbohydrates.util: Glycosidic torsions are undefined for the first
         polysaccharide residue of a chain unless part of a branch.
   core.pose.carbohydrates.util: Returning zero.
   Out[43]: 0.0

   In [44]: maltotriose.phi(2)
   Out[44]: 96.93460655617179

   In [45]: maltotriose.psi(2)
   Out[45]: 109.94421849476635
   ```

```
In [46]: maltotriose.omega(2)
core.pose.carbohydrates.util: Omega is undefined for this residue, because the
       glycosidic linkage is not exocyclic.
core.pose.carbohydrates.util: Returning zero.
Out[46]: 0.0

In [47]: maltotriose.phi(3)
Out[47]: 103.21420435050914

In [48]: maltotriose.psi(3)
Out[48]: 118.64096726060517

In [49]: observer = PyMOL_Observer(maltotriose, True)
~PosePyObserver...

In [50]: for i in (2, 3):
   ....:     maltotriose.set_phi(i, 180)
   ....:     maltotriose.set_psi(i, 180)
   ....:

In [51]: pm.apply(isomaltose)

In [52]: isomaltose.phi(2)
Out[52]: 44.32677030464958

In [53]: isomaltose.psi(2)
Out[53]: -170.86933817075462

In [54]: isomaltose.omega(2)
Out[54]: 49.383018645410004

In [55]: observer = PyMOL_Observer(isomaltose, True)
~PosePyObserver...

In [56]: isomaltose.set_phi(2, 180)

In [57]: isomaltose.set_psi(2, 180)

In [58]: isomaltose.set_omega(2, 180)
```

4. Any cyclic residue also stores its ν angles.

```
In [59]: pm.apply(glucose)

In [60]: Glc1 = glucose.residue(1)

In [61]: for i in range(1, 6): print Glc1.nu(i)
60.755967
-62.556893
-299.976484
-57.917154
56.8
```

5. However, we care more about the ring conformation of a cyclic residue's rings, in this case, its only ring with index of 1. (The output values here are the ideal angles, not the actual angles, which we viewed above.)

```
In [62]: print Glc1.ring_conformer(1)
4C1 (chair): C-P parameters (q, phi, theta): 0.55, 180, 0; nu angles
       (degrees): 60, -60, 60, -60, 60
```

6. `Pose::set_nu()` does not exist, because it would rip a ring apart. Instead, we need to use the `set_ring_conformer()` method, which takes a `RingConformer` object. We can ask a cyclic `ResidueType` for one of its `RingConformerSet`s to give us the `RingConformer` we want. Then, we can set the conformation for our residue through `Pose`. (The arguments of `set_ring_conformer()` are the `Pose`'s sequence position, ring number, and the new conformer, respectively.)

```
In [63]: ring_set = Glc1.type().ring_conformer_set(1)

In [64]: conformer = ring_set.get_ideal_conformer_by_name('1C4')

In [65]: glucose.set_ring_conformation(1, 1, conformer)

In [66]: pm.apply(glucose)
```

Modified Sugars, Branched Oligosaccharides, & PDB File `LINK` Records

1. Modified sugars can also be created in Rosetta, either from sequence or from file. In the former case, simply use the proper abbreviation for the modification after the "ring form code". For example, the abbreviation for an *N*-acetyl group is "NAc". Note the *N*-acetyl group in the PyMOL window.

```
In [67]: LacNAc = pose_from_saccharide_sequence('b-D-Galp-(1->4)-a-D-GlcpNAc')

In [68]: pm.apply(LacNAc)
```

2. Rosetta can handle branched oligosaccharides as well, but when loading from a sequence, this requires the use of brackets, as in the standard IUPAC notation. For example, here is how one would load Lewis^x (Le^x), a common branched glyco-epitope, into Rosetta by sequence.

```
In [69]: Lex = pose_from_saccharide_sequence('b-D-Galp-(1->4)-[a-L-Fucp-(1-
       >3)]-D-GlcpNAc')
core.conformation.Conformation: appending residue by a chemical bond in the
       foldtree: 3 ->4)-alpha-L-Fucp:non-reducing_end:branch_lower_terminus
       anchor:  O3     1 root:  C1

In [70]: pm.apply(Lex)
```

3. One can also load branched carbohydrates from a PDB file. These PDB files must include `LINK` records, which are a standard part of the PDB format. A `LINK` record looks like this:

```
LINK         O3  Glc A   1                   C1  Fuc B   1     1555   1555  1.5
```

It tells us that there is a covalent linkage between O3 of glucose A1 and C1 of fucose B1 with a bond length of 1.5 Å. (The 1555s indicate symmetry and are ignored by Rosetta.)

```
In [71]: Lex = pose_from_file('Lex.pdb')
core.conformation.Conformation: Connecting residues: 1 ( ->4)-beta-D-
      Glcp:reducing_end:->3)-branch:2-AcNH ) and 3 ( ->4)-alpha-L-Fucp:non-
      reducing_end:branch_lower_terminus ) at atoms  O3  and  C1
core.conformation.Conformation:  with mutual distances: 0.18607 and 1.79499

In [72]: pm.apply(Lex)
```

You may notice when viewing the structure in PyMOL that the hybridization of the carbonyl of the amido functionality of the *N*-acetyl group is wrong. This is because of an error in the model deposited in the PDB from which this file was generated. This is, unfortunately, a very common problem with sugar structures found in the PDB.

You may also have noticed the test/data/carbohydrates/Lex.pdb file indicated in its HETNAM records that Glc1 was actually an *N*-acetylglycosamine (Glc*N*Ac) with the indication 2-acetylamino-2-deoxy-. This is optional and is helpful for human-readability, but Rosetta only needs to know the base ResidueType of each sugar residue; specific VariantTypes needed—and most sugar modifications are treated as VariantTypes—are determined automatically from the atom names in the HETATM records for the residue. Anything after the comma is ignored.

4. Print out the Pose to see how the FoldTree is defined. Note the CHEMICAL Edge (-2). This is Rosetta's way of indicating a branch backbone connection. Unlike a standard POLYMER Edge (-1), it tells you which atoms are involved.
   Print out the sequence of each chain.

```
In [73]: print Lex
PDB file name: Lex.pdb
Total residues:3
Sequence: ZZZ
Fold tree:
FOLD_TREE  EDGE 1 2 -1  EDGE 1 3 -2  O3   C1

In [74]: for i in range(2): print Lex.chain_sequence(i + 1)
beta-D-Galp-(1->4)-D-GlcpN
alpha-L-Fucp-
```

5. Print out information about each residue in the Pose to see which VariantTypes and ResiduePropertys are assigned to each.

```
In [75]: for res in Lex: print res
...
```

6. Output the various torsion angles. Now it should be clear why ϕ and ψ are defined the way they are. If they were defined as in AA residues, they would not have unique definitions, since Glc*N*Ac is a branch point.

```
In [76]: Lex.phi(2)
Out[76]: -85.80426357119143

In [77]: Lex.psi(2)
Out[77]: 135.6468768989725

In [78]: Lex.phi(3)
```

```
Out[78]: -76.88533924419255

In [79]: Lex.psi(3)
Out[79]: -97.03727535363538
```

Note that for this oligosaccharide $\chi_3(1)$ is equivalent to $\psi(3)$ and $\chi_4(1)$ is equivalent to $\psi(2)$.

```
In [80]: Lex.chi(3, 1)
Out[80]: -97.03727535363538

In [81]: Lex.chi(4, 1)
Out[81]: 135.6468768989725
```

For modified sugars, χ angles are redefined at the positions where substitution has occurred. For new χs that have come into existence from the addition of new atoms and bonds, new definitions are added to new indices. For example, for Glc*N*^{2}Ac residue 1, $\chi_{C2–N2–C'–C\alpha'}$ is accessed through `chi(7, 1)`.

```
In [82]: Lex.chi(2, 1)
Out[82]: -230.8915297047683

In [83]: Lex.set_chi(2, 1, 180)

In [84]: pm.apply(Lex)

In [85]: Lex.chi(7, 1)
Out[85]: 179.81012671885887

In [86]: Lex.set_chi(7, 1, 0)

In [87]: pm.apply(Lex)
```

7. Play around with getting and setting the various torsion angles for Le^x.

N- and *O*-Linked Glycans

1. Branching does not have to occur at sugars; a glycan can be attached to the nitrogen of an Asn or the oxygen of a Ser or Thr. *N*-linked glycans themselves tend to be branched structures.

```
In [88]: N_linked = pose_from_file('N-linked_14-mer_glycan.pdb')
...

In [89]: pm.apply(N_linked)

In [90]: pm.send_ss(N_linked)

In [91]: print N_linked
PDB file name: N-linked_14-mer_glycan.pdb
Total residues:19
Sequence: ANASAZZZZZZZZZZZZZZ
Fold tree:
FOLD_TREE  EDGE 1 5 -1  EDGE 2 6 -2  ND2  C1   EDGE 6 14 -1  EDGE 8 15 -2  O6
       C1   EDGE 15 17 -1  EDGE 15 18 -2  O6   C1   EDGE 18 19 -1
```

```
In [92]: for i in range(4): print N_linked.chain_sequence(i + 1)
ANASA
alpha-D-Glcp-(1->3)-alpha-D-Glcp-(1->3)-alpha-D-Glcp-(1->3)-alpha-D-Manp-(1-
      >2)-alpha-D-Manp-(1->2)-alpha-D-Manp-(1->3)-beta-D-Manp-(1->4)-beta-D-
      GlcpNAc-(1->4)-beta-D-GlcpNAc-
alpha-D-Manp-(1->2)-alpha-D-Manp-(1->3)-alpha-D-Manp-
alpha-D-Manp-(1->2)-alpha-D-Manp-

In [93]: O_linked = pose_from_file('O_glycan.pdb')
...

In [94]: pm.apply(O_linked)

In [95]: pm.send_ss(O_linked)

In [96]: print O_linked
PDB file name: O_glycan.pdb
Total residues:4
Sequence: ASAZ
Fold tree:
FOLD_TREE  EDGE 1 3 -1  EDGE 2 4 -2  OG   C1

In [97]: for i in range(2): print O_linked.chain_sequence(i + 1)
ASA
alpha-D-Glcp-
```

2. `set_phi()` and `set_psi()` still work when a glycan is linked to a peptide.

```
In [98]: N_linked.set_phi(N_linked.pdb_info().pdb2pose("B", 1), 180)

In [99]: pm.apply(N_linked)

In [100]: N_linked.set_psi(N_linked.pdb_info().pdb2pose("B", 1), 0)

In [101]: pm.apply(N_linked)

In [102]: N_linked.set_omega(N_linked.pdb_info().pdb2pose("B", 1), 90)

In [103]: pm.apply(N_linked)
```

Notice that in this case ψ and ω affect the side chain torsions (χs) of the asparagine residue.

3. One can also create conjugated glycans from sequences if performed in steps, first creating the peptide portion by loading from a `.pdb` file or from sequence and then using the `glycosylate_pose()` function, (which needs to be imported first.) For example, to glycosylate an ASA peptide with a single glucose at position 2 of the peptide, we perform the following:

```
In [104]: peptide = pose_from_sequence('ASA')

In [105]: pm.apply(peptide)

In [106]: from rosetta.core.pose.carbohydrates import glycosylate_pose,
      glycosylate_pose_by_file

In [107]: glycosylate_pose(peptide, 2, 'Glcp')
```

```
core.conformation.Conformation: appending residue by a chemical bond in the
       foldtree: 4 ->4)-alpha-D-Glcp:non-reducing_end:branch_lower_terminus
       anchor:  OG     2 root:  C1
core.pose.carbohydrates.util: Glycosylated pose with Glcp-OGSER2

In [108]: pm.apply(peptide)
```

(Unfortunately, PyMOL tends to get confused about which atoms are actually forming bonds, but if you examine the structure carefully, you will find that all of the atoms are in the right place.)
It is also possible to glycosylate a pose with common glycans found in the database. These files end in the `.iupac` extension and are simply IUPAC sequences just as we have been using.

```
In [109]: cd ../../../database/chemical/carbohydrates/common_glycans/

In [110]: ls
...

In [111]: peptide = pose_from_sequence('ASA')

In [112]: glycosylate_pose_by_file(peptide, 2, 'core_5_O-glycan.iupac')
core.conformation.Conformation: appending residue by a chemical bond in the
       foldtree: 4 ->3)-alpha-D-Galp:branch_lower_terminus:2-AcNH anchor:  OG
       2 root:  C1
core.pose.carbohydrates.util: Glycosylated pose with a-D-GalpNAc-(1->3)-a-D-
       GalpNAc--OGSER2

In [113]: pm.apply(peptide)
```

ROSETTACARBOHYDRATE Tutorial/Demo 2

Scoring

sugar_bb Scoring Method

Currently, scoring for carbohydrates has simply used the talaris scoring function but with the addition of a sugar_bb scoring term. sugar_bb implements the CHI energy function from Grant & Woods, *Curr. Opin. Struct. Biol.* **2014**, *28C*, 47–55. The CHI energy function for ϕ angles depends on whether the residue is axial or equatorial at its anomeric position—that is, on whether or not it is an α or β carbohydrate. The CHI energy function for ψ depends on the attachment position and axial/equatorial designation of the previous residue. It is intended that further functions for other torsion angles will be added to the sugar_bb term in the future.

1. Start up IPython and import and initialize PyRosetta.

```
$ ipython
...

In [1]: from rosetta import *

In [2]: init(extra_options='-include_sugars -read_pdb_link_records')
...
```

2. Load in maltotriose.

```
In [3]: cd tests/integration/tests/carbohydrates/input/
...

In [4]: maltose = pose_from_file('maltotriose.pdb')
...
```

3. Create a standard ScoreFunction and score the Pose. Note that when the -include_sugars flag is used, the sugar_bb scoring term weight is set to 1.0 by default.

```
In [5]: sf = get_fa_scorefxn()
...
core.scoring.ScoreFunctionFactory: The -include_sugars flag was used with no
      sugar_bb weight set in the weights file.  Setting sugar_bb weight to 1.0
      by default.

In [6]: sf.show(maltose)
...
------------------------------------------------------------
 Scores                       Weight   Raw Score Wghtd.Score
------------------------------------------------------------
 fa_atr                       0.800     -4.279      -3.423
 fa_rep                       0.440      1.693       0.745
 fa_sol                       0.750      8.367       6.275
 fa_intra_rep                 0.004      4.866       0.019
 fa_elec                      0.700     -0.359     -0.251
 pro_close                    1.000      0.000      0.000
 hbond_sr_bb                  1.170      0.000      0.000
 hbond_lr_bb                  1.170      0.000      0.000
 hbond_bb_sc                  1.170      0.000      0.000
 hbond_sc                     1.100     -0.530     -0.583
 dslf_fa13                    1.000      0.000      0.000
 rama                         0.200      0.000      0.000
 omega                        0.500      0.000      0.000
 fa_dun                       0.560      0.000      0.000
 p_aa_pp                      0.320      0.000      0.000
 ref                          1.000      0.000      0.000
 sugar_bb                     1.000      2.691      2.691
---------------------------------------------------
 Total weighted score:                       5.473
```

4. If we print maltose, we can see that it has 1α→4 connections.

```
In [7]: print maltose.chain_sequence(1)
alpha-D-Glcp-(1->4)-alpha-D-Glcp-(1->4)-D-Glcp
```

α-D-Glucopyranose has all equatorial hydroxyl groups except at the anomeric position where the hydroxyl is axial. So in the series of four energy plots above, for ϕ we care about the upper left plot, which is for α connections, and for ψ we care about the lower right plot, which is for equatorial connections to the 4 position. Currently the values for ϕ and ψ are in the right vicinity but not ideal.

```
In [8]: maltose.phi(2)
Out[8]: 96.93460655617179

In [9]: maltose.psi(2)
Out[9]: 109.94421849476635

In [10]: maltose.phi(3)
Out[10]: 103.21420435050914

In [11]: maltose.psi(3)
Out[11]: 118.64096726060517
```

5. The MinMover should be able to idealize the ϕ and ψ components of the CHI energy function.

```
In [12]: sugar_bb_only_sf = ScoreFunction()

In [13]: sugar_bb_only_sf.set_weight(sugar_bb, 1.0)

In [14]: sugar_bb_only_sf.show(maltose)
...
------------------------------------------------------------
 Scores                       Weight   Raw Score Wghtd.Score
------------------------------------------------------------
 sugar_bb                     1.000      2.691      2.691
---------------------------------------------------
 Total weighted score:                       2.691

In [15]: mm = MoveMap()
In [16]: mm.set_bb(True)

In [17]: minimizer = MinMover(mm, sugar_bb_only_sf, "dfpmin", 0.01, True)

In [18]: minimizer.apply(maltose)
...

In [19]: sugar_bb_only_sf.show(maltose)
------------------------------------------------------------
 Scores                       Weight   Raw Score Wghtd.Score
------------------------------------------------------------
 sugar_bb                     1.000       0.341       0.341
---------------------------------------------------
 Total weighted score:                    0.341

In [20]: maltose.phi(2)
Out[20]: 71.5925163033063

In [21]: maltose.psi(2)
Out[21]: 103.62750223435093

In [22]: maltose.phi(3)
Out[22]: 75.19453791137022

In [23]: maltose.psi(3)
Out[23]: 114.69894025990594
```

We can show the energies for each residue individually. Here, we confirm that ϕ and ψ are undefined for the first residue.

```
In [24]: maltose.energies().show(1)
E              sugar_bb
E(i)   1           0.00

In [25]: maltose.energies().show(2)
E              sugar_bb
E(i)   2           0.09

In [26]: maltose.energies().show(3)
E              sugar_bb
E(i)   3           0.26
```

6. Isomaltose has an exocyclic connection between its tow saccharide residues.

```
In [27]: isomaltose = pose_from_file('isomaltose.pdb')

In [28]: print isomaltose.chain_sequence(1)
alpha-D-Glcp-(1->6)-D-Glcp

In [29]: sugar_bb_only_sf.show(isomaltose)
core.conformation.util: The attachment point for the query atom is not found
      in the ring; an axial/equatorial designation is meaningless.
------------------------------------------------------------
 Scores                       Weight   Raw Score Wghtd.Score
```

```
------------------------------------------------------------
 sugar_bb                     1.000       1.925       1.925
---------------------------------------------------
 Total weighted score:                     1.925

In [30]: isomaltose.phi(2)
Out[30]: 44.32677030464958

In [31]: isomaltose.psi(2)
Out[31]: -170.86933817075462

In [32]: isomaltose.omega(2)
Out[32]: 49.383018645410004

In [33]: minimizer.apply(isomaltose)
...

In [34]: sugar_bb_only_sf.show(isomaltose)
------------------------------------------------------------
 Scores                       Weight   Raw Score Wghtd.Score
------------------------------------------------------------
 sugar_bb                     1.000       0.030       0.030
---------------------------------------------------
 Total weighted score:                     0.030

In [35]: isomaltose.phi(2)
Out[35]: 73.93352994380763

In [36]: isomaltose.psi(2)
Out[36]: -170.86933817075462

In [37]: isomaltose.omega(2)
Out[37]: 49.383018645410004
```

Note that only ϕ changed. This is because for an exocyclic connection, ψ is no longer a connection to a ring, so axial and equatorial are meaningless, and the scoring function shown above does not apply. Likewise, ω torsions do not show the same statistical trends as ψ.

7. The `sugar_bb` scoring term also works across branches and conjugation connections to peptides or other polymers. Again, only ϕ and ψ angles axial or equatorial to a ring will be scored. (Note that we need to set the `MoveMap` to allow angles with `BRANCH` `TorsionID`s to move.)

```
In [1]: from rosetta import *

In [2]: init(extra_options='-include_sugars -read_pdb_link_records')
...

In [3]: sugar_bb_only_sf = ScoreFunction()

In [4]: sugar_bb_only_sf.set_weight(sugar_bb, 1.0)

In [5]: mm = MoveMap()

In [6]: mm.set_bb(True)
In [7]: mm.set_chi(True)

In [8]: mm.set_branches(True)

In [9]: minimizer = MinMover(mm, sugar_bb_only_sf, "dfpmin", 0.01, True)

In [10]: cd tests/integration/tests/carbohydrates/input/
...

In [11]: Lex = pose_from_file('Lex.pdb')
...

In [12]: for chain in range(1, Lex.conformation().num_chains() + 1): print
       Lex.chain_sequence(chain)
beta-D-Galp-(1->4)-D-GlcpN
alpha-L-Fucp-

In [13]: print Lex.fold_tree()
FOLD_TREE  EDGE 1 2 -1  EDGE 1 3 -2  O3   C1

In [14]: Lex.phi(2)
Out[14]: -85.80426357119143

In [15]: Lex.psi(2)
Out[15]: 135.6468768989725

In [16]: Lex.phi(3)
Out[16]: -76.88533924419255

In [17]: Lex.psi(3)
Out[17]: -97.03727535363538

In [18]: mm.set_branches(True)

In [19]: minimizer.apply(Lex)
core.pose.util: WARNING: Unable to find atom_tree atom for this Rosetta branch
       connection angle: residue 3 BRANCH 1

In [20]: Lex.phi(2)
Out[20]: -65.62117800426097

In [21]: Lex.psi(2)
Out[22]: 136.95480413762755

In [23]: Lex.phi(3)
Out[23]: -73.03033840543367

In [24]: Lex.psi(3)
Out[24]: -93.63314466264218

In [25]: O_linked = pose_from_file('O_glycan.pdb')
...

In [26]: for chain in range(1, O_linked.conformation().num_chains() + 1):
       print O_linked.chain_sequence(chain)
```

```
ASA
alpha-D-Glcp-

In [27]: O_linked.phi(4)
Out[27]: 71.4566619001219

In [28]: O_linked.psi(4)
Out[28]: 177.2781916108563

In [29]: minimizer.apply(O_linked)
...

In [30]: O_linked.phi(4)
Out[30]: 73.93352351769039

In [31]: O_linked.psi(4)
Out[31]: 177.2781916108563
```