Homology modelling and modelling of missing residues using MODELLER

Eric J. M. Lang
eric.lang@bristol.ac.uk

Mulholland Group
University of Bristol

November 27, 2017

Outline

1. Introduction
   - About Modeller
   - About this tutorial
   - Installing Modeller
2. A Simple example
   - Getting everything ready
   - Loop modelling 1
   - Loop modelling 2
   - Loop modelling 3
3. Advanced topics

Introduction

- Most crystal structures have 'missing' residues
- Corresponds to regions of low electronic density
- Often corresponds to highly flexible regions (e.g. loops)
- For some applications (e.g. MD, SAXS, pKa and electrostatic calculations, molecular docking) all the residues are needed
- Can potentially be modelled approximatively using residual density
- Otherwise several programmes and webserver can be used to efficiently model those missing residues
- Here we are going to use Modeller `https://salilab.org/modeller/`

About Modeller

- Can be used for homology modelling and modelling of missing residues
- Work as a python library, basically the instructions you give to model the missing residues correspond to a small python script.
- However, prior knowledge of Python is not required and several examples of scripts can be found on the Modeller website.
- *Advantages:* very flexible and powerful package. Does not generate a single conformation but a user specified number of conformations which are refined with some MD steps and are given a score
- *Drawbacks:* steep learning curve, no GUI, scripting required

About this tutorial

- Here we are going to go through the various steps that lead to the reconstruction of missing residue in a simple case
- The scripts presented here can then be adapted with minimal effort to suit your needs, providing you with the tools needed to reconstruct missing residue for your own structures
- The same procedure can be used for generating homology models
- Some additional examples and information can be found here:
  `https://salilab.org/modeller/tutorial/`
  `https://salilab.org/modeller/manual/node33.html`
  `https://salilab.org/modeller/wiki/Missingresidues`
  `https://salilab.org/modeller/manual/node81.html`
  `https://salilab.org/modeller/manual/node15.html`

Installing Python

Firstly to install Python:

- On Windows, install Anaconda (pref. with Python 2.7), a free Python environment/package manager and the most easy to use and popular for science `https://docs.continuum.io/anaconda/install`
- On Linux or MacOS, Python is already installed in your Path, however, it might be useful to install Anaconda anyway
- If you do not want Anaconda to become your default python, then you can follow these instructions: `https://giusedroid.blogspot.co.uk/2015/04/dont-let-anaconda-eat-your-python.html`

Installing Modeller

Then you can install modeller by following the instructions for your system (or for Anaconda if installed):
`https://salilab.org/modeller/download_installation.html`

Modeller on Curie

- `ssh username@curie.chm.bris.ac.uk`
- `source /share/apps/AJM-software/anaconda2/anaconda_env.sh`

Files needed for this workshop

All the files needed for this workshop, including the scripts and example of generated models, are available here: `https://github.com/eric-jm-lang/Modeller_tutorial`

Using the terminal

Start a terminal window

- On Windows, open the program Anaconda Prompt
- On MacOS or Linux open a terminal

Let's say your working directory, which include your script and input files is in `C:\Users\eric\tutorial_modeller` on Windows or `/home/eric/tutorial_modeller` on Linux/MacOS, you can go to this directory from the command line by typing:

- `cd C:\Users\eric\tutorial_modeller` on Windows
- `cd /home/eric/tutorial_modeller` on MacOS or Linux

Then to run a python script named `myscript.py` which is present in this directory you can just type

- `python myscript.py`

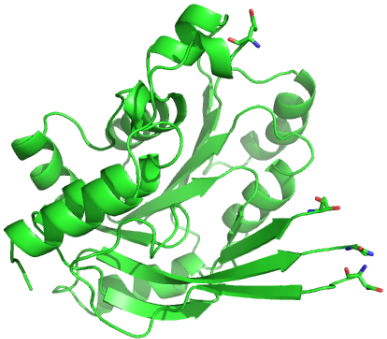
A few comments on Python

A few comments on Python scripts (which are required to run Modeller):

- Lines or sentences that start with a # are comments
- Indentation matters, try to keep the indentation of the scripts provided if you are not sure of what you are doing (otherwise you will run into error messages)
- The first lines (that include an `import` statement) should not be modified as they are needed to import libraries
- If you want to learn very quickly the basics of Python, you can have a look at `http://chryswoods.com/beginning_python/index.html`

A simple example: modelling the missing loops of 1QG8

Structure of nucleotide-diphospho-sugar transferase, SpsA from *B. subtilis*



Resolution 1.5 Å. Missing residues: 134-136 and 218-231

Get the sequence from the PDB file

To retrieve the sequence from the PDB file i.e. without the missing residues we will use the Python script:

`get_sequence.py`

```
from modeller import *
code = '1qg8' # only this need to be adapted
e = environ()
m = model(e, file=code)
aln = alignment(e)
aln.append_model(m, align_codes=code)
aln.write(file=code+'.seq')
```

The script can be run with

`python get_sequence.py`

and will generate the file

`1qg8.seq`

1qg8.seq

```
>P1;1qg8
structureX:1qg8:   2 :A:+238 :A:: :1.50: 0.16
PKVSVIMTSYNKSDYVAKSISSILSQTFSDFELFIMDDNSNEETLNVIRPFLNDNRVRFYQSDISGVKERTEKTR
YAALINQAIEMAEGEYITYATDDNIYMPDRLLKMVRELDTHPEKAVIYSASKTYHLNDIVKETVRPAAQVTWNAP
CAIDHCSVMHRYSVLEKVKEKFGSYWDESPAFYRIGDARFFWRVNHFYPFYPLDEELDLNYITEFVRNLPPQRNC
RELRESLKKLGMG*
```

Aligning the incomplete and the complete sequences

- Now that we have the sequence from the PDB we also need the complete sequence
- These two sequences can then be aligned using your favourite program, e.g `T-coffee`, `Clustalo`, etc.
- The resulting alignment is saved under the following format in `alignment.ali`
- With a text editor create a file alignment.ali that will contain the alignment on the same format as the `.seq` file
- You should obtain something in the following format. Note the header and the line below, the number of ':' matters.

alignment.ali

```
>P1;1qg8
structureX:1qg8:   2 :A:+238 :A:: :1.50: 0.16
PKVSVIMTSYNKSDYVAKSISSILSQTFSDFELFIMDDNSNEETLNVIRPFLNDNRVRFYQSDISGVKERTEKTR
YAALINQAIEMAEGEYITYATDDNIYMPDRLLKMVRELDTHPEKAVIYSASKTYHLN---DIVKETVRPAAQVTW
NAPCAIDHCSVMHRYSVLEKVKEKFGSYWDESPAFYRIGDARFFWRVNHFYPFYPLDEELDLNYIT---------
-----EFVRNLPPQRNCRELRESLKKLGMG*


>P1;1qg8_fill
sequence:::::::::
PKVSVIMTSYNKSDYVAKSISSILSQTFSDFELFIMDDNSNEETLNVIRPFLNDNRVRFYQSDISGVKERTEKTR
YAALINQAIEMAEGEYITYATDDNIYMPDRLLKMVRELDTHPEKAVIYSASKTYHLNENRDIVKETVRPAAQVTW
NAPCAIDHCSVMHRYSVLEKVKEKFGSYWDESPAFYRIGDARFFWRVNHFYPFYPLDEELDLNYITDQSIHFQLF
ELEKNEFVRNLPPQRNCRELRESLKKLGMG*
```

Loop modelling 1 - script loop_modelling_1.py

Now that we have alignment.ali and 1qg8.pdb we can reconstruct the missing loops with the Python script loop_modelling_1.py

```
from modeller import *
from modeller.automodel import *

log.verbose()
env = environ()
env.io.atom_files_directory = ['.', '../atom_files']

# This is where you will need to modify the script
a = loopmodel(env, alnfile = 'alignment.ali',
              knowns = '1qg8', sequence = '1qg8_fill')

# First a single model with the missing residue is created
a.starting_model= 1
a.ending_model  = 1

# Then the shortest loop is refined and 2 models created
a.loop.starting_model = 1
a.loop.ending_model   = 2
a.loop.md_level       = refine.fast
a.make()
```

Loop modelling 1 - output

To run the script:

```
python loop_modelling_1.py
```

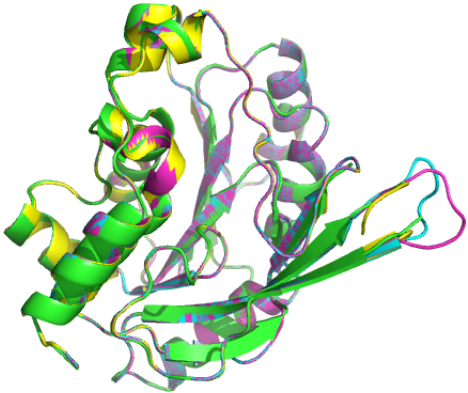
Once the job is completed the following is found in the terminal:

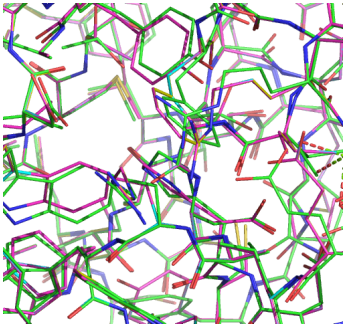
```
>>Summary of successfully produced models:
Filename                              molpdf
----------------------------------------
1qg8_fill.B99990001.pdb       1472.66125

>>Summary of successfully produced loop models:
Filename                              molpdf
----------------------------------------
1qg8_fill.BL00010001.pdb        18.40255
1qg8_fill.BL00020001.pdb      3032.60303
```

A number of files has been created. Only the pdb files listed above are of interest for now.

Loop modelling 1 - results

A number of problems...





What can we do better?

- Fix the position of the known residues
- Specify the loop residues to be refined
- Generate more than 2 models
- Use a slower but more accurate refinement process

Loop modelling 2 - script loop_modelling_2.py

```
from modeller import *
from modeller.automodel import *

log.verbose()
env = environ()
env.io.atom_files_directory = ['.', '../atom_files']

# Create a new class based on 'dopehrloopmodel' (more accurate than 'loopmodel') so
    that we can redefine select_loop_atoms
class MyLoop(dopehr_loopmodel):
    # This routine picks the residues to be refined by loop modelling
    def select_atoms(self):
        # Only the following residues are allowed to move the others are fixed
        return selection(self.residue_range('134:', '136:'), self.residue_range('
    218:', '231:'))
    def select_loop_atoms(self):
        # One loop from residue 134 to 136 inclusive
        return selection(self.residue_range('134:', '136:'))
```

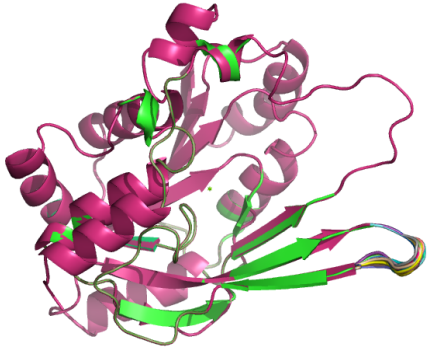
Loop modelling 2 - script loop_modelling_2.py cont'd

```
a = MyLoop(env, alnfile = 'alignment.ali',
                knowns = '1qg8', sequence = '1qg8_fill',
                loop_assess_methods=assess.DOPE) # assess with DOPE

# First a single model with the missing residue is created
a.starting_model= 1
a.ending_model  = 1

# Then the specified loop is refined and 25 models created
a.loop.starting_model = 1
a.loop.ending_model   = 25

# Very thorough optimization:
a.loop.library_schedule = autosched.slow
a.loop.max_var_iterations = 300
# Thorough MD optimization:
a.loop.md_level = refine.slow
# Repeat the whole optimization cycle 2 times
a.loop.repeat_optimization = 2

a.make()
```

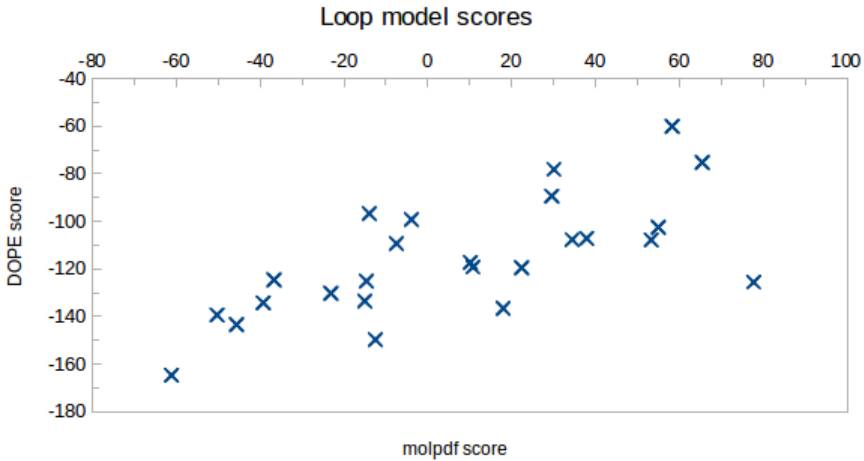
Loop modelling 2 - output

```
python loop_modelling_2.py
```

```
>> Summary of successfully produced loop models:
Filename                          molpdf      DOPE score
----------------------------------------------------------------------
1qg8_fill.BL00010001.pdb       -36.67811      -124.63010
1qg8_fill.BL00020001.pdb        22.40097      -119.55441
1qg8_fill.BL00030001.pdb        -3.87882       -99.31304
1qg8_fill.BL00040001.pdb       -23.11016      -130.26935
1qg8_fill.BL00050001.pdb       -13.94719       -96.79842
1qg8_fill.BL00060001.pdb       -12.47222      -149.74731
1qg8_fill.BL00070001.pdb        53.30920      -107.89207
1qg8_fill.BL00080001.pdb        -7.42850      -109.40715
1qg8_fill.BL00090001.pdb       -61.10769      -164.70064
1qg8_fill.BL00100001.pdb        10.79705      -119.35873
1qg8_fill.BL00110001.pdb        65.48814       -75.31876
1qg8_fill.BL00120001.pdb        77.74438      -125.65179
1qg8_fill.BL00130001.pdb        34.48872      -107.69442
1qg8_fill.BL00140001.pdb        37.91122      -107.23292
1qg8_fill.BL00150001.pdb       -50.25227      -139.36330
1qg8_fill.BL00160001.pdb       -45.62112      -143.40579
1qg8_fill.BL00170001.pdb        58.31322       -60.16826
1qg8_fill.BL00180001.pdb        18.03246      -136.59779
1qg8_fill.BL00190001.pdb       -39.23501      -134.23422
1qg8_fill.BL00200001.pdb       -14.59691      -125.18148
1qg8_fill.BL00210001.pdb        30.09506       -78.29447
1qg8_fill.BL00220001.pdb       -15.04100      -133.51920
1qg8_fill.BL00230001.pdb        29.59528       -89.40944
1qg8_fill.BL00240001.pdb        10.20290      -117.29503
1qg8_fill.BL00250001.pdb        55.00820      -102.54991
```



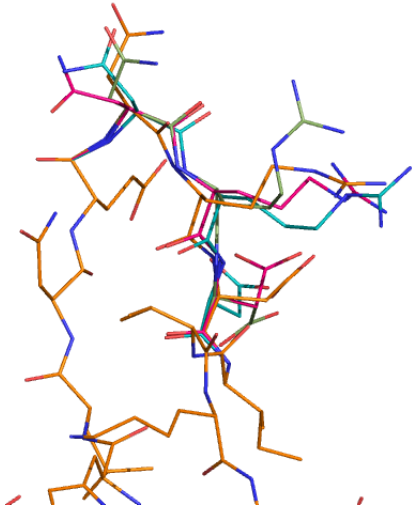
Loop modelling 2 - results



The lower the molpdf and DOPE scores, the better the model

Loop modelling 2 - results

- Best molpdf and DOPE: model 9
- 2nd best DOPE: model 6
- 2nd best mopdf: model 16
- 2nd best olpdf and DOPE combo: model 15



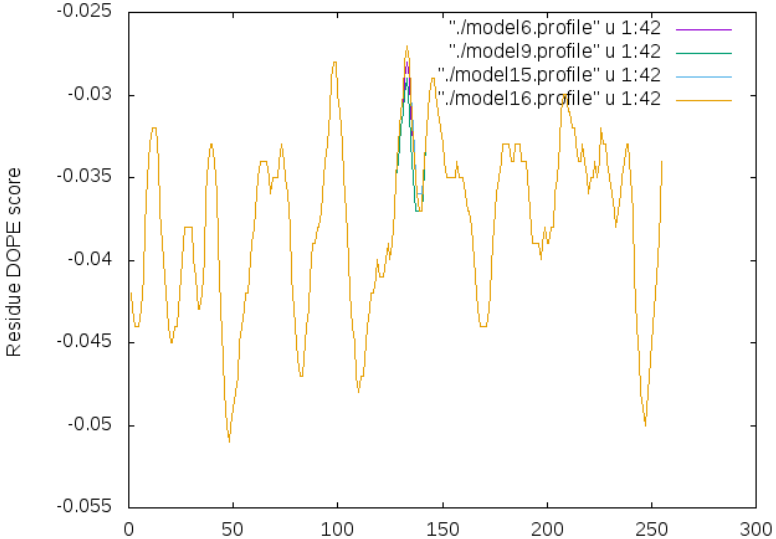
DOPE score per residue

It is also possible to calculate a DOPE score per residue using the following script `evaluate_model.py`

```
from modeller import *
from modeller.scripts import complete_pdb

log.verbose()    # request verbose output
env = environ()
env.libs.topology.read(file='$(LIB)/top_heav.lib') # read topology
env.libs.parameters.read(file='$(LIB)/par.lib') # read parameters

# read model file, modify according to your file name
mdl = complete_pdb(env, '1qg8_fill.BL00120001.pdb')

# Assess all atoms with DOPE:
s = selection(mdl)
s.assess_dope(output='ENERGY_PROFILE NO_REPORT', file='model12.profile', # modify
    the name of the output based on your needs
              normalize_profile=True, smoothing_window=15)
```

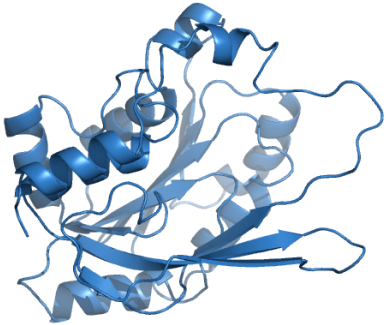
Run it with `python evaluate_model.py`

Loop modelling 2 - results



Loop modelling 2 - results

- To judge the quality of a model use the global molpdf and DOPE scores, the residue DOPE score and visual inspection
- Here model 9 seems to be the best model
- It can be used as an input for the refinement of the second loop following a similar procedure



Loop modelling 3 - script loop_modelling_3.py

```
from modeller import *
from modeller.automodel import *

log.verbose()
env = environ()
env.io.atom_files_directory = ['.', '../atom_files']

class MyLoop(dopehr_loopmodel):
    def select_atoms(self):
        return selection(self.residue_range('218:', '231:'))
    def select_loop_atoms(self):
        return selection(self.residue_range('218:', '231:'))
```

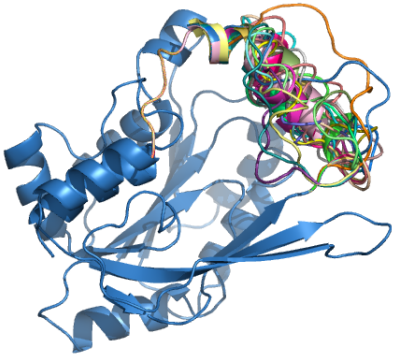
Loop modelling 3 - script loop_modelling_3.py cont'd

```
# no need to refer to an alignment, we just want to optimize the loop
a = MyLoop(env,
           inimodel='model9.pdb', # Best model form previous step
           sequence='model9', # code (same name but without '.pdb')
           loop_assess_methods=assess.DOPE) # assess with DOPE

# Then the specified loop is refined and 25 models created
a.loop.starting_model = 1
a.loop.ending_model   = 25

# Very thorough optimization:
a.loop.library_schedule = autosched.slow
a.loop.max_var_iterations = 300
# Thorough MD optimization:
a.loop.md_level = refine.slow
# Repeat the whole optimization cycle 2 times
a.loop.repeat_optimization = 2

a.make()
```

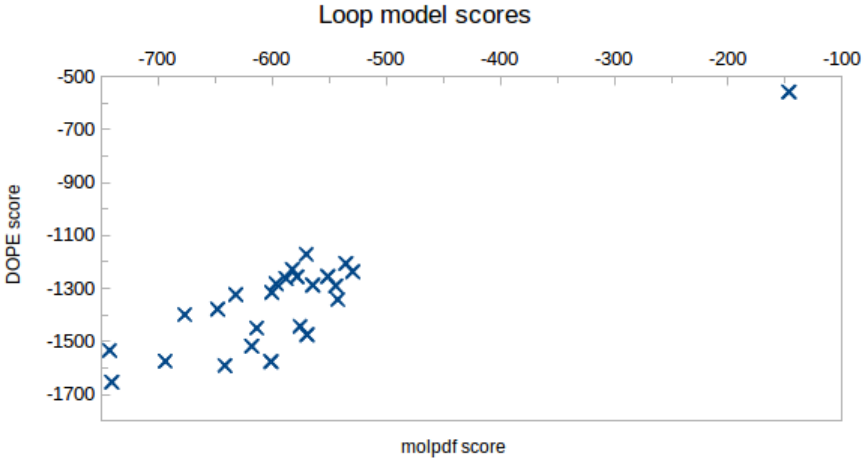
Loop modelling 3 - output

```
python loop_modelling_3.py
```

```
>> Summary of successfully produced loop models:
Filename                          molpdf      DOPE score
----------------------------------------------------------------------
model9.BL00010001.pdb          -648.01672   -1378.28064
model9.BL00020001.pdb          -575.53906   -1442.28687
model9.BL00030001.pdb          -617.99707   -1517.81775
model9.BL00040001.pdb          -613.58093   -1449.34668
model9.BL00050001.pdb          -551.22217   -1254.72021
model9.BL00060001.pdb          -600.32861   -1315.25916
model9.BL00070001.pdb          -569.50171   -1474.44324
model9.BL00080001.pdb          -146.70552    -558.54486
model9.BL00090001.pdb          -582.37024   -1227.64954
model9.BL00100001.pdb          -564.46912   -1287.57715
model9.BL00110001.pdb          -601.11768   -1575.82629
model9.BL00120001.pdb          -693.61298   -1574.89331
model9.BL00130001.pdb          -542.57153   -1341.94666
model9.BL00140001.pdb          -740.71765   -1652.90576
model9.BL00150001.pdb          -569.99017   -1170.70496
model9.BL00160001.pdb          -529.52032   -1236.92212
model9.BL00170001.pdb          -578.33887   -1255.39856
model9.BL00180001.pdb          -631.94574   -1323.02722
model9.BL00190001.pdb          -587.88086   -1262.59045
model9.BL00200001.pdb          -544.29077   -1290.23816
model9.BL00210001.pdb          -535.37756   -1205.70764
model9.BL00220001.pdb          -742.58740   -1533.11499
model9.BL00230001.pdb          -641.59418   -1590.25696
model9.BL00240001.pdb          -596.53149   -1282.42761
model9.BL00250001.pdb          -676.57007   -1399.05774
```



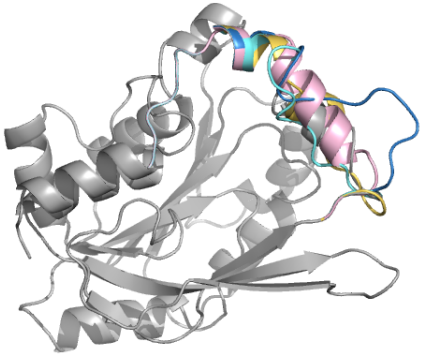
Loop modelling 3 - results



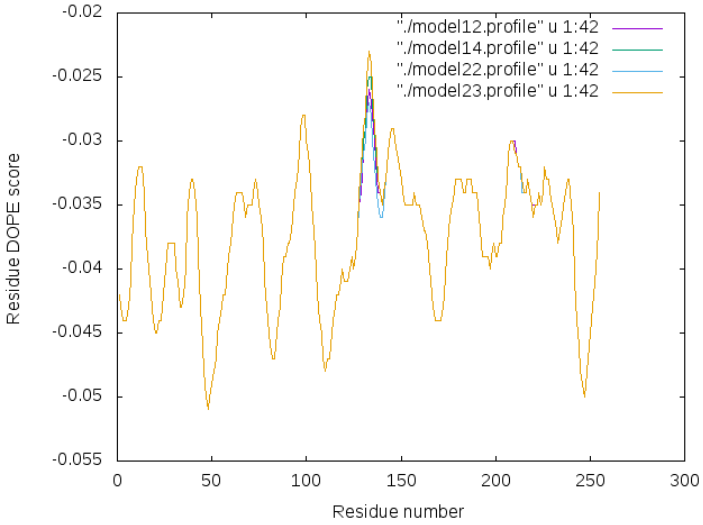
The lower the molpdf and DOPE scores, the better the model

Loop modelling 3 - results

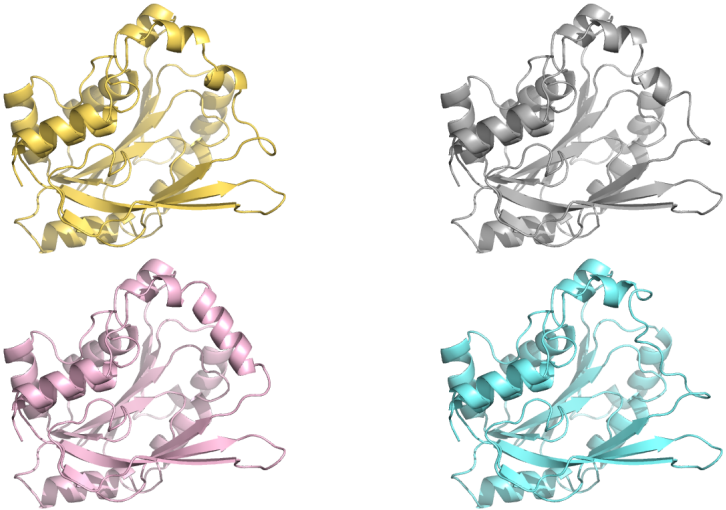
- Best molpdf and DOPE combo: model 14
- 2nd best DOPE: model 23
- best mopdf: model 22
- 2nd best olpdf and DOPE combo: model 12



Loop modelling 3 - results



Loop modelling 3 - output



Advanced topics 1

- If several loops need to be refined, it is better to generate several models using the `automodel` routine (as opposed to the `loopmodel`) using for example `a.ending_model = 100` and then select the best models and refine the loops of interest
- It is possible to work with proteins containing more than one chain `https://salilab.org/modeller/manual/node30.html`
- It is possible to run the modelling process in parallel, which is especially useful for large multimeric proteins and/or long loops `https://salilab.org/modeller/9v2/examples/automodel/model-parallel.py`

Advanced topics 2

- If your protein is a homomultimer, you can set up symmetry restraints so the modelled parts adopt a similar conformation across monomeric units `https://salilab.org/modeller/manual/node29.html`
- It is possible to add secondary structure restraints `https://salilab.org/modeller/manual/node28.html`
- It is possible to include ligands by setting `env.io.hetatm` to `True` `https://salilab.org/modeller/manual/node18.html` It is even possible to add the parameters for your own ligands

Advanced topics 3

- Let's modify the previous script, `loop_modelling_3.py`, so it can run in parallel (i.e. using more than one processor core)
- This is achieved by importing the `modeller.parallel` library and adding a few instructions within the script.
- To be able to run the job in parallel, it is also required to save the `class MyLoop` statement into a separate python script that we are going to call `MyLoop.py`

Advanced topics 3 - script MyLoop.py

```
from modeller import *
from modeller.automodel import *

class MyLoop(dopehr_loopmodel):
    def select_atoms(self):
        return selection(self.residue_range('218:', '231:'))
    def select_loop_atoms(self):
        return selection(self.residue_range('218:', '231:'))
```

Advanced topics 3 - script loop_modelling_3.py

```
from modeller import *
from modeller.automodel import *
from modeller.parallel import * # This enables to work with multiple processor cores
from MyLoop import MyLoop # The MyLoop class is in a different file and thus needs
    to be imported

j = job()
# The following specify the number of processor cores to use, do not specify more
    core than you have available on your machine
j.append(local_slave())
j.append(local_slave())
j.append(local_slave())
j.append(local_slave())
j.append(local_slave())
j.append(local_slave())

log.verbose()
env = environ()
env.io.atom_files_directory = ['.', '../atom_files']
```

Advanced topics 3 - script loop_modelling_3.py cont'd

```
# no need to refer to an alignment anymore are there, we just want to optimize the
    loop
a = MyLoop(env, inimodel='model9.pdb', # Best model form previous step
           sequence='model9', # code (same name but without '.pdb')
           loop_assess_methods=assess.DOPE) # assess with DOPE

# Then the specified loop is refined and 25 models created
a.loop.starting_model = 1
a.loop.ending_model   = 25

# Very thorough optimization:
a.loop.library_schedule = autosched.slow
a.loop.max_var_iterations = 300
# Thorough MD optimization:
a.loop.md_level = refine.slow
# Repeat the whole optimization cycle 2 times
a.loop.repeat_optimization = 2
# Specify that the run should be in parrallel
a.use_parallel_job(j)
a.make()
```