Andrew Torda
Iryna Bondarenko
Björn Hansen

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Zentrum für Bioinformatik

Übung zur Vorlesung
Angewandte Bioinformatik

Sommersemester 2015

4. Mai 2015

Übung 3: Homology Modelling

1. Introduction

This is about building a model for a protein when you only know its sequence and using the structure of related proteins. We use the program *MODELLER*. During this exercise, you will use many skills and tools from this course such as sequence alignments, structure comparison, *ClustalW* and *Chimera*. A major point of the assignment is to see how the quality of the results depends on the initial alignments.

2. Contents

3. What is *MODELLER*

Homology modelling in general and *MODELLER* in particular is based on the assumption that similar sequences will fold up into similar 3-D structures. Constructing a model for a protein using its homology to other proteins means taking structural features from the parent structure(s) based on sequence alignment and applying them to the sequence in question. These features are then combined with restraints based on the sequence being modelled. *MODELLER* attempts to automate the homology modelling process. It can use multiple sequence alignments. It can also use multiple templates.

The program uses distance geometry and energy minimisation. Given a sequence and template structure(s), it can make an alignment and decide which parts of a template structure are probably conserved in the sequence. This alignment is used to build a set of distance restraints, which is then used by a distance geometry routine. The initial structures are then refined by energy minimization.

MODELLER balances the physical requirements of the modelled sequence against the geometric relations obtained from the template structures. The code and even more information are at: http://salilab.org/modeller/modeller.html.

4. Using *MODELLER*

MODELLER reads a script of instructions including run-time variables and procedural function calls. The variables are lists of flags, or strings, and set like this:

Simplemodel.py

```
from modeller.automodel import *    # Load the automodel class
   log.verbose()                    # request verbose output
   env = environ()                  # create a new MODELLER environment
                                    # directories for input atom files
   env.io.atom_files_directory = './:./atom_files'
   a = automodel( env,
                  alnfile  = 'fdxhomologs.pir',   # alignment filename
                  knowns   = ('1fxd', '1fdn'),    # codes of the templates
                  sequence = 'ferodox')           # code of the target
   a.auto_align()                                 # get an automatic alignment
   a.make()                                       # do the actual homology modeling
```

The `knowns` variable is a list of structure names, and `sequence` is a string containing the name of the sequence to be modelled. These names correspond to the entries for each protein in the alignment file:

fdxhomologs.pir: A PIR file

```
# A Comment
# This is the .pir file entry for 1fdn
>P1;1fdn
structure:1fdn:FIRST:@:55:@:.:.:.:.
*
>P1;1fxd
structure:1fxd:FIRST:@:58:@:.:.:.:.
*
# This entry doesn't get used by simplemodel.py
>P1;5fd1
structure:5fd1:FIRST:@106:@:ferredoxin:Azotobacter vinelandii: 1.90.0.92
*
>P1;ferodox
sequence:1fdx:1: :54: :ferredoxin:Peptococcus aerogenes: 2.00:-1.00
AYVINDSCIACGACKPECPVNIIQGSIYAIDADSCIDCGSCASVCPVGAPNPED*
# The * indicates the end of the sequence entry
```

The alignment file is like a flat-file database. Each '>P1;blah' line is a tag for the protein data called 'blah'. *MODELLER* looks for tags with the same names as those given in the `sequence` and `knowns` variables defined in the *.py* file.

The line after a tag is a ':' separated set of fields defining the protein data. For *5fd1* the first field is 'structure', and the next is the structure's PDB name. *MODELLER* uses this to search for the protein's PDB at each of the paths defined by another runtime variable called `env.io.atom_files_directory`. The next four fields are for the beginning and end residue number and chain ID that define the stretch of coordinates which *MODELLER* should read. In this case, 'FIRST:@' means the first residue in the first chain of the PDB file, and '60:@' means the residue 60 (in PDB file numbering) in the same chain. The rest of the fields are typical sequence database entries (the protein name, and species), and the last two are the resolution and R factor for the X-ray structure. These are defined as a wildcard ('.') in most of the other entries, and the extra information is not important for this example.

The 'ferodox' entry is different, because it is the sequence to be modeled. The first field is 'sequence' - indicating that there is no structure. The second is a filename that will be used if a model is generated, and the rest give the numbering and amino acid chain code that will be used when the model is written.

The final line(s) give sequence information for the protein, as bounded by the start and end positions given in the fields in the previous line. Sequence entries must be in upper case, single letter amino acid codes, where the '*' is the terminator for the sequence. Any whitespace characters are ignored. When the 'START:@:XX:@' field is used, where XX is some number, no sequence is necessary. However, for 'ferrodox', the amino acid sequence must be defined, and is given as shown. It is important that there are precisely the same number of amino acids as is given in the preceding header by the start and end amino-acid numbering (54 a.a. characters in this case). The PIR format allows the representation of multiple alignments. A '-' character can be used for the sequences to indicate a gap. An example of this will be shown later.

The alignment filename is given to *MODELLER* via the `alnfile` variable, and any extra PDB file paths are defined in `env.io.atom_files_directory` - the filenames and paths are all relative to the current working directory in this example. 'a.auto_align()' makes a multiple alignment of all the protein sequences. This is written out as a new .pir file called fdxhomologs.pir.ali. The final command 'a.make()' runs the automatic modeling function, which results in the following:

1. A log file is created, called *simplemodel.log*.
2. *MODELLER* reads *fdxhomologs.pir*, finds the unknown sequence and also reads in the structures specified in knowns.
3. A multiple alignment of all the sequences is made and written to a new *.pir* file called *fdxhomologs.pir.ali*.
4. The known structures are analysed based on the alignment, and used to make a set of distance restraints.
5. A model is made by distance geometry and refined.
6. The model is written out as *ferodox.B99990001.pdb*.

To run *MODELLER* using this script, make a new directory, change to it, and enter the following shell commands

```
➢ echo $SHELL
➢ export PATH=/home/torda/bin/modeller9.14/bin:$PATH
➢ cp -R /home/hansen/teaching/modelling ./
➢ cd modelling
➢ mod9.14 simplemodel.py
```

to set up the path for *MODELLER*, and run the fully automatic modelling script file. The command `export` does only work for the bash shell, which is enabled by default for all student accounts. However, if you have changed your shell to tcsh, you need to write

```
➢ setenv PATH /home/torda/bin/modeller9.14/bin:$PATH
```

instead. To determine which shell is set as standard for your account, type

```
➢ echo $SHELL
```

If everything has gone well, the *MODELLER* command will finish after a few minutes.
The directory listing should then look something like:

```
ferodox> ls
atom files           ferodox.D00000001   ferodox.rsr
fdxhomologs.pir      ferodox.V99990001   ferodox.sch
fdxhomologs.pir.ali  ferodox.ini         ferodox.B99990001.pdb
family.mat           simplemodel.log     simplemodel.py
```

The *ferodox.** files are related to the modeled sequence. The others are the log file *simplemodel.log* written by *MODELLER*, and the multiple alignment that was actually used to generate the model (*fdxhomologs.pir.ali*).

fdxhomologs.pir.ali : Describing a multiple sequence alignment to match the ferodox sequence to two homologs:

```
>P1;1fxd
structure:1fxd:FIRST:@:58   :@:.:.: 0.00: 0.00
PIEVNDDCMACEACVEIC--PDVFEMNEEGDKAVVINP--DSD-LDCVEEAIDSCPAEAI-VRS*

>P1;1fdn
structure:1fdn:FIRST:@:55   :@:.:.: 0.00: 0.00
AYVINEACISCGACEPECPV-NAISSG-D-DRYVID-ADT-CID---CGACAGVCPVDAP-VQA*

>P1;ferodox
sequence:1fdx:1    : :54   : :ferredoxin:Peptococcus aerogenes: 2.00:-1.00
AYVINDSCIACGACKPECPV-NIIQ----GSIYAIDADS--CID---CGSCASVCPVGAPNPED*
```

The alignment also yields a distance matrix: *family.mat*: This is the 'distance' between each sequence, as defined by *MODELLER*'s own multiple alignment (the multiple sequence alignment is shown in the *.pir.ali* file).

The most important files are the generated coordinates:
ferodox.B99990001.pdb: The PDB coordinates for the model. The *9999* indicates that the coordinates are modelled rather than 'measured', and *0001* indicate that this is model number 1. In addition to 'ATOM' records, there are two informational entries in the file:

```
EXPDTA THEORETICAL MODEL, MODELLER 9v5 2012/01/....
REMARK 6 MODELLER OBJECTIVE FUNCTION: 1380.0945
```

The REMARK line gives the 'energy' value or objective function for the model (the smaller the better). The rest of the files give detailed output from the steps of model construction and minimization.

ferodox.ini:	The initial model generated by distance geometry.
ferodox.rsr:	The restraints applied to the atoms in the model, during construction and minimization. It lists pairs of atoms and the ideal distances.
ferodox.sch:	The energy function schedule used in the optimization.

ferodox.V99990001: Heavy atom violations' for each residue.

ferodox.D00000001: The progress of optimization from model 0 (*ferodox.ini*) to model 1 (*ferodox.B99990001.pdb*).

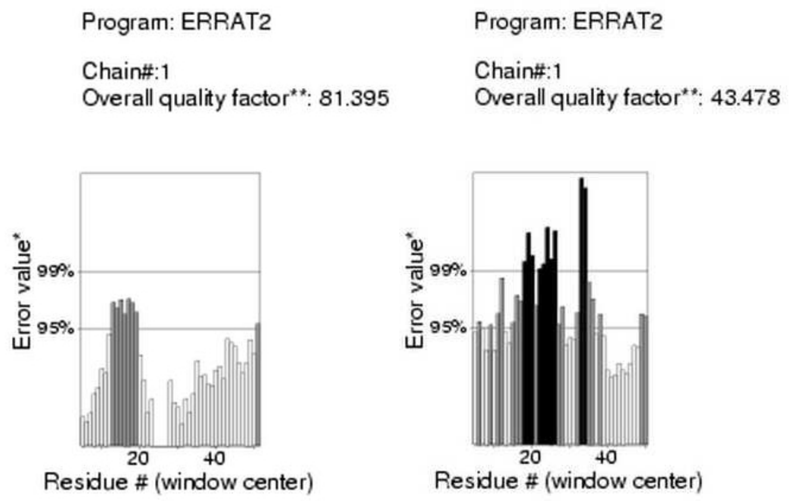
That was a very simple example. In the next section, we see how one can use better alignments from outside the *MODELLER* program to make better models.

5. Measuring Model Quality

The protein sequence used in this exercise is from a known structure (PDB ID *1dur*), so it is possible to determine if a model is good or bad simply by comparing it to the real protein structure. In a real homology modelling experiment, the true structure is not known, so it is necessary to use tools for analyzing the quality of the models. Such a measurement of model quality can be based on physical energy functions - which give a measure of structural quality, but it can be hard to estimate the absolute quality with these. The alternative is to assess the backbone conformations and side-chain packing using statistical observations of the typical geometry of proteins (knowledge based energy functions). One of these protein structure validation programs, called ERRAT, will be used in this exercise. The program examines a PDB file, and generates a score based on the quality of the local structure surrounding each residue, as compared to the typical ranges of dihedral angles and side chain contacts observed in real proteins. You can find the webserver for the program at:

http://nihserver.mbi.ucla.edu/ERRAT/

To use the server, upload your PDB file using the web form. The program generates a plot which gives a measure of the structure error at each residue in the protein. It also calculates an overall score for the structural quality. You can read more information about the method by following the links on the ERRAT server page. *Figure 1* shows two ERRAT assessments - one for the model generated by the *simplemodel.py* script, and along side it is the plot for the real structure for the sequence. The overall measure of quality is given at the top of the plot and each bar in the histogram is shaded according to the significance of the local structural error.



```
Program: ERRAT2

Chain#:1
Overall quality factor**: 81.395
```

```
Program: ERRAT2

Chain#:1
Overall quality factor**: 43.478
```

Figure 1. ERRAT output for the Ferredoxin structure and model.

The validation plot for the real structure (PDB code *1dur*) is on the left, and the model generated by *MODELLER* is on the right. The overall quality factor is given just above the plot of structure error.

6. Exercise: Modelling Ferredoxin

In this exercise, you will reproduce the ferredoxin model and structural quality analysis which were described in the last two sections. You will then try to generate a better model by modifying the alignment. This will involve making a multiple sequence alignment with a variety of sequences and gap penalties, in order to get a better mapping of the sequence on to the template structures. The quality should be described by the ERRAT plot, its overall quality measurement, and the *MODELLER* objective function value.

If you have followed all the steps described, and sent *ferodox.B99990001.pdb* to ERRAT server, you should get a result which is similar to the left one on *figure 1*. Save it as a .jpg or postscript file.

Examine the generated model (*ferodox.B99990001.pdb* with chimera), its ERRAT plot, and the alignment file (*fdxhomologs.pir.ali*). This shows the parts of the other ferredoxin structures which were used to make the model. Comment on the quality, using these aspects as a guide:

Find the residues that ERRAT suggests are badly modelled (grey or black in histogram).
Using the viewer program see if these are α–helical, ß-sheet or not regular structure.

Are the poorly modelled regions the result of gaps or insertions?

Making a better alignment

Use the EMBNet ClustalW web form to make an alignment between the ferodox sequence, and the two templates (1fxd and 1fdn):

```
http://www.ch.embnet.org/software/ClustalW.html
```

1. Make a copy of the *fdxhomologs.pir.ali* file, and change it to a FASTA file by removing the ':' separated fields line and the 'P1;' from the tags.
2. Remove the '*' and any '-' symbols from the sequence entries.
3. Change the tag of the ferodox entry to ferodoxM.
4. Save the FASTA file as *clustalM.fasta* as you will use it again later.
5. Paste the file into the form. Change the output format to 'PIR' but leave the alignment parameters at their defaults (BLOSUM matrix, 10 for gap opening and end gap penalties, and 0.05 for extension).
6. Select the 'RunClustalW' button to make the alignment.
7. The PIR file will be presented shortly, as a link ('PIR') amongst a number of links to other alignment file formats. Save the file as *fdxCWM.pir*.
8. Edit the new alignment so it can be used by *MODELLER*:
   - Copy the ':' separated fields line for each protein into the new alignments so that *MODELLER* can match the structure files to the sequence entries.
   - Save it.
9. Make a copy of *simplemodel.py*, called *cwMmodel.py*. Edit this new *.py* file so it will read the ClustalW alignment and make a new model:
   - Change the `alnfile` variable:

     ```
     alnfile = 'fdxCWM.pir'
     ```

     `alnfile` is the variable that defines the filename for an alignment that is to be used by the modelling routine.
   - Change the `sequence` variable assignment so that `sequence` is set to ferodoxM. This is the tag that you used in the FASTA file you submitted to ClustalW.
   - Comment out "`a.auto_align()`" by putting a hash (#) at the start of that line.
   - Save the updated file.
10. Run the *MODELLER* script with *cwMmodel.py*. It should finish in about a minute. The new model will be called 'ferodoxM', after the tag used in the new PIR file.
11. Send *ferodoxM.B9990001.pdb* to ERRAT and save the plot as *ferodoxM.ps* or *ferodoxM.jpg*. Note which parts of the model have been improved due to the more accurate alignment.

Adding more homologs to the multiple alignment

The two template sequences are reasonably good homologs for the sequence that you are modeling, but they are not the only ones that could be used. Open the file *fdxfamily.fasta*. You will see some additional members of the feredoxin family.

1. Add the extra sequences to the sequences as input to ClustalW - and submit them to the server using the same gap penalties as in the last section.
2. Save the new, wider multiple sequence alignment as *fdxCWW.pir*.
3. Examine the file *fdxfamily.pir* to find the tags and extra information fields needed to complete the entries in your new alignment.
4. Modify your *fdxCWM* script again for this new alignment, and include the following lines before the call to the 'model' procedure.

   a.starting_model = 2

   a.ending_model = 2

   These statements set the first and last 'model number' for the generated models. By default, *MODELLER* makes one model. If the variables above are defined, then *MODELLER* enters a loop:

```
i = a.starting_model
do {
      generate a new distance geometry structure
      optimize the structure
      write the i'th model (as name.B(99990000 + i))
      i++
   }  while (i < a.ending_model)
```

5. Generate the new model and assess its quality.

Finding better gap penalties

Repeat the multiple sequence alignment and model generation that you did in the last section, but play with the following alignment parameters:

BLOSUM matrix (unchanged)

Gap Opening and End Gap = 1 or 90

Extension and Separation = 0.05

Remember to save this alignment as a PIR file under a new name, and add the extra tag lines. You also need to modify your *cwMmodel* script so that it uses this new alignment file rather than *fdxCWM.pir*, and assign 3 to the a.starting_model and a.ending_model.

7. Your Conclusions

a) Summarize in a few sentences what you have been doing during this exercise: What have you investigated? Which tools did you use to generate the alignments, the homology models and the measurement of model quality?

b) How can a model of a protein's structure be generated by using its homology to some known structures?

c) Why does ERRAT not need to know any template structures in order to estimate the quality of your model?

d) Look at your results: Are high/low quality regions of your models associated with certain secondary structure elements or high/low sequence similarity? If there is a poor region in the model structure, try to give a reason for it.

e) What is the meaning of the *MODELLER* objective function value?

f) How did the results change with different alignment parameters? Why does the quality of the calculated structures depend on the alignment’s quality? Which alignment was most effective for obtaining a good model? Why?

Please answer these questions. We will discuss them at the end of the exercise on May 11, 2015.