PRACTICAL 1

Sequence of a protein → unknown function and structure.

Goal: get a protein with known structure (template) homologous to our target.
Finding templates to model the structure of our target

BLAST
Hits: protein sequences identified similar to our target.

Score: similarity between sequences, +score, +similarity. → computed using substitution matrices (to assign score).
e-value: P of finding a hit by chance. Takes into account size dbs and length of the alignment.

BLAST
PSI-BLAST: iterative version of BLAST, obtains more specific substitution matrices (better finding homologous proteins). Iterates and searches new hits each iteration. Also updates the substitution matrix (PSSM) → hits with higher specificity.

Substitution Matrices
Contain scores associated to the frequency of substitution between 2 aa.
Come from MSA

- Non specific: same scores for all the positions. Compute all substitutions frequencies for all positions in MSA at once. BLOSUM62 (by default BLAST)
- Specific: different score for each position. Compute substitution frequencies for each of the positions of MSA separately.

TUTORIAL

Step 1: using BLAST

Look for proteins of known structure similar to the target:

```
$blastp -query target.fa -db ~/Documents/databases/pdb_seq -out target_pdb.out
```

```
$blastp -query [target.fa] -db [Database] -out [output]
```

Step 2: using PSI-BLAST

```
$psiblast -query target.fa -db ~/Documents/databases/pdb_seq -num_iterations 5 -out target_pdb_5.out
```

```
$psiblast -query [target.fa] -db [Database] -num_iterations [#iter] -out [output]
```

Results per iteration → ROUND
Best iteration is the last one (more refinable)

PROBLEM: PDB only contains sequences of known structures → redundant database
PSSM obtained from biased sequence → biased sequence search.
*searches for higher numbers of known structures.

SOLUTION: use a nonredundant dbs. Uniprot/SwissProt

Store the PSSM generated at the 5th iteration ← -out_pssm

```
$psiblast -query target.fa -num_iterations 5 -out_pssm target_sprot5.pssm -out target_sprot_5.out -db ~/Documents/databases/uniprot_sprot.fasta
```

```
$psiblast -query [target.fa] -num_iterations [#iter] -out_pssm [output.pssm] -out [output] -db [Database]
```

Then, we can apply it to make a search ← -in_pssm
Don't make iterations in PDB, otherwise we'll obtain a biased PSSM again.

NOTE: we are not including the query → the program includes the info of the target sequence in the PSSM to perform the search.

```
$psiblast -db ~/Documents/databases/pdb_seq -in_pssm target_sprot5.pssm -out target_pdb_sprot5.out
```

```
$psiblast -db [Database] -in_pssm [PSSM] -out [output]
```

Step 3: using PSI-BLAST with alignments of ClustalW

We can also obtain the MSA from ClustalW, which enables us to select sequences whose alignment will build the PSSM. Then introduce the MSA generated by ClustalW into PSI-BLAST and run a search.

To do so, concatenate the sequence we want to align in fasta format and type:

```
$clustalw file.fa
```

Get an output of a BLAST (the one obtained for the PSSM in uniprot) and put it in ‘file.list’. Execute FetchFasta.pl

```
$perl ~/Documents/perl_scripts/FetchFasta.pl -i file.list -d ~/Documents/databases/uniprot_sprot.fasta -o file.fasta
```

```
$perl [~/Documents/perl_scripts/FetchFasta.pl] -i [file.list] -d [Database] -o [output.fasta]
```

Create a new document and concatenate the sequences obtained in the previous step with the target.

```
$cat target.fa > pssm.fasta
$cat file.fasta >> pssm.fasta
```

Then, run ClustalW.

```
$clustalw2 pssm.fasta
```

```
$clustalw2 [fasta_file]
```

Change the format of the clustalw2 output.

```
$perl ~/Documents/perl_scripts/aconvertMod2.pl -in c -out f <pssm.aln>pssm.fa
```

```
$perl [aconvertMod2 path] -in c -out f <pssm.aln>pssm.fa
```

Finally, introduce the MSA as input in the PSI-BLAST. ← -in_msa

```
$psiblast -in_msa pssm.fa -out target_pdb_specific.out -db ~/Documents/databases/pdb_seq
```

```
$psiblast -in_msa [pssm.fa] -out [output] -db [Database]
```

BLAST: find homologous sequences to a target
PSI-BLAST: find homologous sequences to a target using iterations
CLUSTALW: make MSAs

PRACTICAL 2

HMM
Model that produces aa one by one. HMMs have different states, each one produces aa with different frequencies. They change from one state to another while producing the protein.

Emission: conserved position of the sequence in reference with the ancestral sequences.
Insertion: insertion of an aa in the sequence in reference with the ancestral sequences.
Deletion: elimination of one aa in the sequence in reference with the ancestral sequences. Consider P of a gap.

Change of states is determined by the probabilities contained in the HMM, and are specific for each position.

Frequencies contained in a HMM are calculated from a Multiple Sequence Alignment (MSA)

TUTORIAL

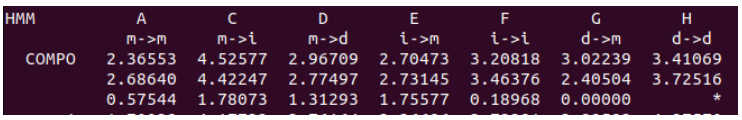
Step 1: creating a HMM using hmmbuild

hmmbuild takes a file in STOCKHOLM and builds a HMM

```
$hmmbuild globins4.hmm globins4.sto
```

```
$hmmbuild [model_HMM] [alignment]
```

Profiles: HMM informative for specific protein domains.



HMM	A	C	D	E	F	G	H
	m->m	m->i	m->d	i->m	i->i	d->m	d->d
COMPO	2.36553	4.52577	2.96709	2.70473	3.20818	3.02239	3.41069
	2.68640	4.42247	2.77497	2.73145	3.46376	2.40504	3.72516
	0.57544	1.78073	1.31293	1.75577	0.18968	0.00000	*

NSEQ = number of sequences used to create the HMM
LENG = number of positions our hmm has. indicate how big/small is the protein domain.

Emission (main) state (1 row)
Insertion state (2 row)
Transition state (3 row)

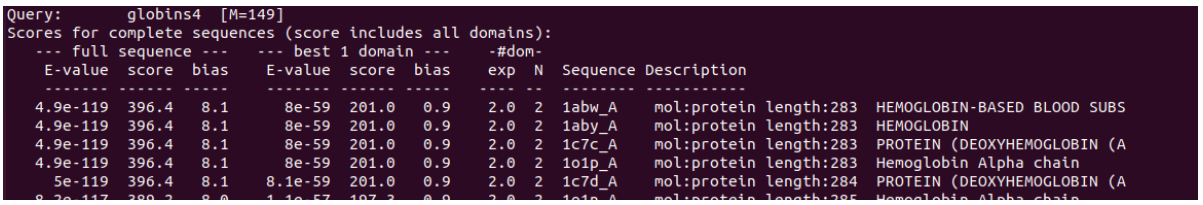
Step 2: searching for similar sequences using hmmsearch

Finding sequences that fit the model → sequences that will be generated by the HMM with higher probability. Search for templates.

```
$hmmsearch globins4.hmm ~/Documents/databases/pdb_seq > globins_pdb.out
```

```
$hmmsearch (options) [model_HMM] [database] > [output]
```

Result are sequences sorted by e-value (like in BLAST)



```
Query:       globins4  [M=149]
Scores for complete sequences (score includes all domains):
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence Description
    ------- ------ -----    ------- ------ -----   ---- --  -------- -----------
   4.9e-119  396.4   8.1      8e-59  201.0   0.9    2.0  2  1abw_A    mol:protein length:283  HEMOGLOBIN-BASED BLOOD SUBS
   4.9e-119  396.4   8.1      8e-59  201.0   0.9    2.0  2  1aby_A    mol:protein length:283  HEMOGLOBIN
   4.9e-119  396.4   8.1      8e-59  201.0   0.9    2.0  2  1c7c_A    mol:protein length:283  PROTEIN (DEOXYHEMOGLOBIN (A
   4.9e-119  396.4   8.1      8e-59  201.0   0.9    2.0  2  1o1p_A    Hemoglobin Alpha chain
     5e-119  396.4   8.1    8.1e-59  201.0   0.9    2.0  2  1c7d_A    mol:protein length:284  PROTEIN (DEOXYHEMOGLOBIN (A
```

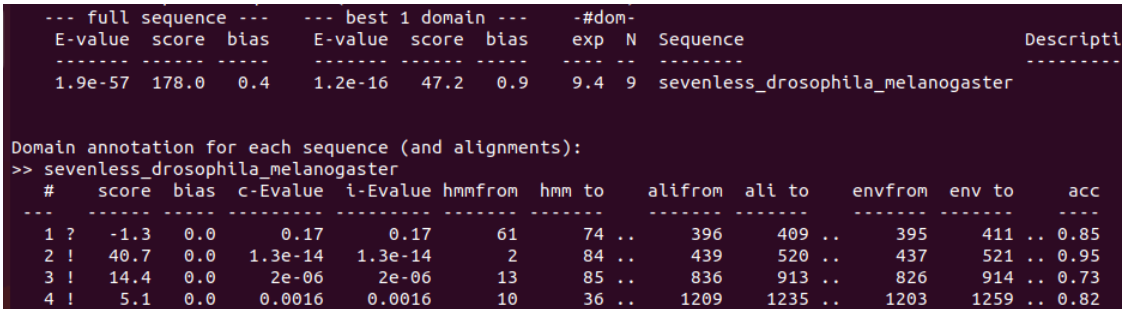
We have different results for the full sequence and the best domain, this happens bc a protein can have more than 1 domain.

Apart from searching similar sequences in a database, we can also search a domain within a single sequence.

```
$hmmbuild fn3.hmm fn3.sto
```

```
$hmmsearch fn3.hmm 7LESS_DROME.fa > fn3.out
```

One e-value is retrieved.



```
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence                          Descripti
    ------- ------ -----    ------- ------ -----   ---- --  --------                          ---------
    1.9e-57  178.0   0.4    1.2e-16   47.2   0.9    9.4  9  sevenless_drosophila_melanogaster


Domain annotation for each sequence (and alignments):
>> sevenless_drosophila_melanogaster
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 ?   -1.3   0.0      0.17      0.17      61      74 ..     396     409 ..     395     411 .. 0.85
   2 !   40.7   0.0   1.3e-14   1.3e-14       2      84 ..     439     520 ..     437     521 .. 0.95
   3 !   14.4   0.0     2e-06     2e-06      13      85 ..     836     913 ..     826     914 .. 0.73
   4 !    5.1   0.0    0.0016    0.0016      10      36 ..    1209    1235 ..    1203    1259 .. 0.82
```

e-value of the full sequence < domain e-value → +1 domain.

Results shown per domain
?: bad e-values, accuracy at we are making the prediction is below the standards of the program (et diu que no es bo, per tel dona per si el vols)
!: good evalues

HMM from-to: part of the HMM that has been aligned.
Ali from-to: where in the protein sequence the domain is located (where the HMM match is)

hmmsearch shows the alignment between the HMM and each of the domains.

```
Alignments for each domain:
== domain 1  score: -1.3 bits;  conditional E-value: 0.17
                                    EES--TT-EEEEEE CS
                             fn3  61 ltgLepgteYefrV 74
                                    l+ L p+t+Y+fr
sevenless_drosophila_melanogaster 396 LEALIPYTQYRFRF 409
                                    67899*******95 PP

== domain 2  score: 40.7 bits;  conditional E-value: 1.3e-14
                                    ---CEEEEEEECTTEEEEEEE--S--SS--SEEEEEEEETTTCCGCEEEEEETTTSEEEEES--TT-EEEEEEEEE CS
                             fn3   2 saPenlsvsevtstsltlsWsppkdgggpitgYeveyqekgegeewqevtvprtttsvtltgLepgteYefrVqav 77
                                    saP   ++ +  ++ l ++W p +  +gpi+gY++++++++++  + e+ vp+   s+ +++L++gt+Y++ +  +
sevenless_drosophila_melanogaster 439 SAPVIEHLMGLDDSHLAVHWHPGRFTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMI 513
                                    789999999999**********************9998.********************************** PP
```

The first alignment (in lowercase) is that of HMM.
The second (in uppercase) is the protein sequence.
Below the sequence, is the alignment score. → % of expected accuracy of the match (7 is 65- 75%, 8 is 75-85%, 9 is 85-95%, * is 95-100%).

Step 3: using HMM databases: hmmpress and hmmscan

hmmscann assigns a sequence the best HMM. Find HMM that fits a sequence.

```
$hmmbuild Pkinase.hmm Pkinase.sto
```

```
$cat globins4.hmm fn3.hmm Pkinase.hmm > minifam
```

Compress and index the database to be able to check sequences and profiles very fast.

```
$hmmpress minifam
```

```
$hmmpress [Database]
```

Get what's the best profile for a given target.

```
$hmmscan minifam 7LESS_DROME.fa > 7LESS_DROME_minifam.out
```

```
$hmmscan (options) [Database_HMM] [sequence] > [output]
```

```
Query:       sevenless_drosophila_melanogaster  [L=2554]
Scores for complete sequence (score includes all domains):
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Model    Description
    ------- ------ -----    ------- ------ -----   ---- --  -------- -----------
    5.6e-57  178.0   0.4    3.5e-16   47.2   0.9    9.4  9  fn3       Fibronectin type III domain
      3e-44  139.0   0.0    4.7e-44  138.3   0.0    1.3  1  Pkinase   Protein kinase domain


Domain annotation for each model (and alignments):
>> fn3  Fibronectin type III domain
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 ?   -1.3   0.0      0.33       0.5      61      74 ..     396     409 ..     395     411 .. 0.85
   2 !   40.7   0.0   2.6e-14   3.8e-14       2      84 ..     439     520 ..     437     521 .. 0.95
   3 !   14.4   0.0   4.1e-06   6.1e-06      13      85 ..     836     913 ..     826     914 .. 0.73
```

Description: we can find the domains matching our sequence.

Step 4: performing multiple sequence alignments using hmmalign

We can also use HMM to perform alignments.

```
$hmmalign globins4.hmm globins45.fa > globins45_hmm.sto
```

```
$hmmalign [model_HMM] [file_with_sequences ] > [output]
```

```
$clustalw globins45.fa
```

```
$clustalw [file_with_sequences]
```

We are working with STOCKHOLM format, we can change the format back to fasta by running using: `aconvertMod2.pl`

```
$perl ~/Documents/perl_scripts/aconvertMod2.pl -in h -out c <globins45_hmm.sto>globins45_hmm.clu
```

```
$perl [path_aconvertMod2.pl] -in h -out c <[file.sto]>[file.clu]
```

Step 5: looking for homolog sequences using phmmer and jackhmmer

Find homologous sequences in a database.
jackhmmer is the iterative version of phmmer.

several iterations = jackhmmer
use of n-flag to change the number of iterations.
phmmer is like jackhmmer at iteration 1.
Use ROUND to find the iterations.

```
$jackhmmer hbb_human globins45.fa > globins_jackhmmer.out
```

```
$jackhmmer [target_sequence] [database_of_sequences] > [output]
```

jackmmer performs several iterations, if the database is small or redundant → profiles are biased

```
$phmmer hbb_human globins45.fa > globins_phmmer.out
```

```
$phmmer [target_sequence] [database_of_sequences] > [output]
```

Step 6: using HMMs from the PFAM database

PFAM is a database that contains profiles of several known families.
Find the best HMMER profile in PFAM for our target sequence.

hmmscan (to scan in PFAM)
hmmfetch (to fetch a profile from PFAM).

To assign the best profile/s to the target sequence. (i.e. hbb_human)

```
$hmmscan ~/Documents/databases/Pfam-A.hmm hbb_human.fa > hb_human_db.out
```

```
$hmmscan [Database] [target.fa] > [output]
```

HMM of the profile(s) assigned to our target sequence. Extract from the PFAM database using the program hmmfetch

```
$hmmfetch ~/Documents/databases/Pfam-A.hmm Globin > domain_hbb.hmm
```

```
$hmmfetch [database_HMM] [name_HMM] > [file_HMM]
```

Search for sequences with known structure that contain the same domain as our target.

```
$hmmsearch domain_hbb.hmm ~/Documents/databases/pdb_seq > hbb_pdb_by_HMM.out
```

```
$hmmsearch [file_HMM] [database_HMM] > [output]
```

Step 7: Some format changes

Transforming a CLUSTALW format of a MSA
ALN → FASTA

```
$perl ~/Documents/perl_scripts/aconvertMod2.pl -in c -out f alignment.fa
```

FASTA → STOCKHOLM

```
$perl ~/Documents/perl_scripts/fasta2sto.pl alignment.fa > alignment.sto
```

FASTA → FASTA MSA

```
$perl ~/Documents/perl_scripts/sto2fasta.pl -g alignment.sto > alignment.fa
```

Transform HMMER3.0 format files in HMMER2.0 format files (old) and ASCII to binary

```
$hmmconvert [options] [name_HMM] > [name2_HMM]
```

Generate a set of sequences derived from a profile

Output:

- generated sequences in FASTA format
- #number is the number of sequences to be generated

```
$hmmemit [options] -N #number [name_HMM] -o [output]
```

hmmbuild: create HMMs from MSAs
hmmsearch: find matches of a HMM in a database of sequences
hmmscan: find matches of a sequence in a database of HMMs
hmmpress: build a database of HMMs
hmmalign Make MSAs using a HMM
phmmer: find homologous sequences to a target
jackhmmer: find homologous sequences to a target using iterations
hmmfetch: extract a HMM from a database
aconvertMod2.pl: change the format of MSAs

PDB:

- Proteins with available structure
- Biased
- Redundant

SCOP:

- Classification of protein structures (from the PDB) into domains.

PFAM:

- HMMs for protein domains

UNIPROT:

- Proteins with available sequence
- Non-biased
- Non-redundant

PRACTICAL 3

Superimposition
To know if two proteins are similar, their structures are superimposed. One of the two proteins will remain still and the other one will be rotated. In the end, we'll get the two proteins fitted in the smallest space possible.

RMSD (Root Mean Square Deviation
Measurement of the average distance between 2 sets of atoms. Tells you how different the structures are. Evaluation of the geometry of the 2 proteins, just with distances.

n = # atoms analyzing

$$\mathrm{RMSD}(\mathbf{v}, \mathbf{w}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}((v_{ix} - w_{ix})^2 + (v_{iy} - w_{iy})^2 + (v_{iz} - w_{iz})^2)}$$

Structural alignment: equivalent residues are close in space (not like in sequence alignment, where equivalent residues are those that fill the same position). Also contain evolutionary information (bc structure is more conserved) → reliable source of evolutionary data (distant homologous proteins).

TUTORIAL

Step 1: superimpose 2 structures

PyMol has 3 different commands to perform protein superimposition:

Align	Program based on sequences. Performs a sequence alignment between the two proteins. Then, matches the two structures by putting close in space the pairs of amino acids aligned. Works for close homologs.
Super	Structure based. Keep one obtained fixed and move the other one. Guide the movement of the protein by the criteria that the RMSD is minimized (get as small as possible). Works with proteins that are distant homologs.
Cealign	Structure based. Superimposes two structures without using sequence information. Tries to minimize the RMSD between the two proteins, but using a different algorithm than super. More flexible. Works well with very distant homologs.

Better structure based methods for distant homologs.
Sequence based methods for close homologs.
Structure more conserved than sequence!

```
fetch 4mbn
fetch 1ecd
```

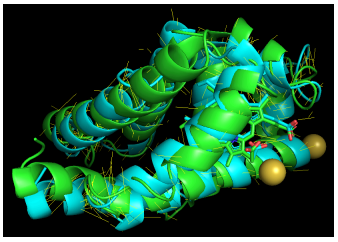
Red crosses floating and surrounding the proteins → H_2O molecules.
They can complicate the superimposition, as the program sometimes wants to superimpose them too. Remove water molecules, since they could interfere with the superimposition.

```
remove resn hoh
```

Use the super command to superimpose both of the structures:

```
super 4mbn, 1ecd, object=aln2
```

```
super [prote_that_moves] [protein_no_move], object = [name_superimposition_object]
```



```
eRMS: 1 atoms rejected during cycle 4 (RMSD
eRMS: 1 atoms rejected during cycle 5 (RMSD
e: RMSD =    2.476 (733 to 733 atoms)
e: object "aln2" created.
ked /4mbn/A/A/GLY`65/CA
```

Yellow lines: representing how we're associated together with the aa we are superimposing. Matching pairs of amino acids is similar to an alignment. We can see the alignment if we click in:

```
display > sequence
```

We can visualize the score:

```
PyMOL>super 4mbn, 1ecd, object=aln2
 MatchAlign: aligning residues (153 vs 136)...
 MatchAlign: score 482.689
 ExecutiveAlign: 775 atoms aligned.
 ExecutiveRMS: 23 atoms rejected during cycle 1 (RMSD=2.77).
 ExecutiveRMS: 12 atoms rejected during cycle 2 (RMSD=2.58).
 ExecutiveRMS: 5 atoms rejected during cycle 3 (RMSD=2.51).
 ExecutiveRMS: 1 atoms rejected during cycle 4 (RMSD=2.49).
 ExecutiveRMS: 1 atoms rejected during cycle 5 (RMSD=2.48).
 Executive: RMSD =    2.476 (733 to 733 atoms)
 Executive: object "aln2" created.
```

We can save the alignment with the following command:

```
save aln2.aln, aln2
```

```
save [name_file] [name_object]
```

*pwd: to know the directory from where you are executing the program, which is the same as where the file is saved.

```
CLUSTAL

4mbn        VLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKK
1ecd        LS----ADQISTVQASFDKVK-GDPVGILYAVFKADPSIMAKFTQFAGK-DLESIKGTAPFET
                  ......*.*.....  *....**...**..*....**..*... .....*.......

4mbn        HGVTVLTALGAILKKKGHHEAELKPLAQSHATKHKIPIKYLEFISEAIIHVLHSRHPGDFGAD
1ecd        HANRIVGFFSKIIGELPNIEADVNTFVASHK-PRGVTHDQLNNFRAGFVSYMKAHTDFAGAEA
            *..........*.......**.......**. ........*......................

4mbn        AQGAMNKALELFRKDIAAKYKELGYQG
1ecd        AWGATLDTFFGMIFSKM----------
            *.**.............
```

Here, the alignment is not really good, but as we saw in PyMol, the structure was conserved.

Step 2: superimpose as many structures as you want

Create a new object, by superimposing 4 different proteins.

Have a reference protein (one that remains still). Since the 4 proteins are homologs the order of executing doesn't matter.

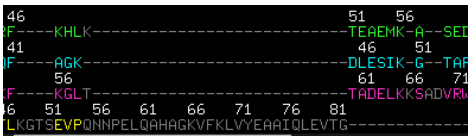
```
remove resn hoh

super 4mbn, 1ecd, object=aln4

super 2lhb, 1ecd, object=aln4

super 1lh1, 1ecd, object=aln4
```

Gap: when PyMol makes the assignment between aa, it is doing it based on a distance threshold. If the aa are in a distance bigger than the threshold it is generating a gap.



The helix doesn't fit with the other helix as the distances between them are too large.

We see that the threshold, which now is in 2Å, is very restrictive.
We can change the distance threshold and repeat the commands.

```
super 4mbn, 1ecd, object=aln4_corrected, cutoff=5.0

super 2lhb, 1ecd, object=aln4_corrected, cutoff=5.0

super 1lh1, 1ecd, object=aln4_corrected, cutoff=5.0
```

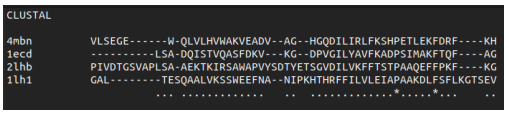
Cutoff is the maximum distance we consider superimposed amino acids.

RMSD will increase, because we are allowing the distance to be higher.

Most gaps are happening in loops, which are the most variable, and then, flexible region of a protein.

We then save the output file.

```
save                aln4_corrected.aln,
aln4_corrected
```



Step 3: create a structure based HMM

Alignment obtained is based on structural similarity. MSA contains structural information for distant homologs of the protein family. → create HMM that will contain structural information for distant homologs.

With the alignment we want to put together the aa that have the same role.
Alignments don't represent the same as the superimposition.

```
$perl ~/Documents/perl_scripts/aconvertMod2.pl -in c -out f
<aln4_corrected.aln>aln4_corrected.fa
```

```
$perl ~/Documents/perl_scripts/fasta2sto.pl aln4_corrected.fa >
aln4_corrected.sto
```

Execute hmmbuild to create the HMM using as input the MSA in stockholm format.

```
$hmmbuild hmm_structural.hmm aln4_corrected.sto
```

Step 4: Use super imposition to model protein-protein interactions

Homologous proteins tend to share their protein-protein interactions.
Proteins are able to carry their function because of the shape they have.

Example:
A is homologous to C, B is homologous to D
To know if C and D interact, we have to superimpose A with C and B with D.

Work with leghemoglobin, and the tetrameric form of human hemoglobin.
They are homologous proteins, so we expect that leghemoglobin can also create tetramers.

Limitation: we need the structures. We don't know what this leghemoglobin looks like, but we can solve this by using superimposition.

Clash: when we are having 2 structures that are filling the same space. Whenever you find a clash, it won't be possible. Tetramer structure by using leghemoglobin as monomer.

Whenever having a protein complex and its functionality in nature, search for clashes.
Showing surface ! (sometimes they are difficult to see).

Leghemoglobin will account for a different set of coordinates each time we repeat the procedure for each chain.

Create selections for each one of the chains of the tetramer:

```
display > sequence mode > chain identifier
```

To superimpose leghemoglobin to hemoA:

```
super leghemoglobin, hemoA
```

```
save leghemoglobin_A.pdb, leghemoglobin
```

Open a new PyMol window and open the file results.

Step 5: Superimpose specific regions of two proteins

Use 2 PDB structures to reconstruct the almost complete DNA binding domain of CTCF.
5t0u and 5yel are different fragments of the same proteins.

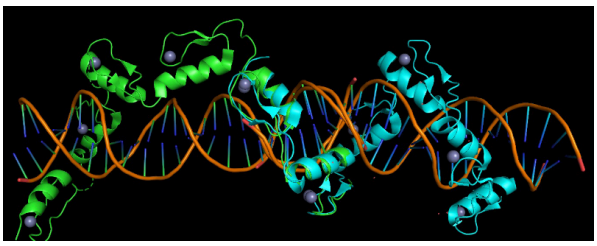
- 5t0u goes from amino acid 293 to amino acid 461.
- 5yel goes from amino acid 405 to amino acid 576.

We want to superimpose these two structures, but we need to ensure that we only superimpose the regions that are overlapping.
We need to specify what residues we want to include in the superimposition.

Align command is used too, because the two structures that we want to superimpose belong to the same protein, and the amino acid sequence will be identical.

```
align 5t0u and resi 405-460, 5yel and resi 405-460
```

Since the two structures belong to the same protein, they match perfectly. We are also superimposing the DNA molecules that are in the structures.



PRACTICAL 4

Protein with known sequence but not the structure → obtain structural models (comparative modeling and threading).

Comparative Modeling (homology modeling)
Modeling the structure of one protein using the structure of homologous proteins.

Threading
Modeling the structure of one protein by assigning to that protein an already known fold → fitting protein sequence an already known structure. Make a lot of models and then differentiate the accurate ones.

Statistical potentials
Scoring functions derived from the analysis of experimental structures. useful to know if our model is correct ← good scores if the model has similar structural features to the protein in the reference set.
Relative measurements, we need something to compare with ← model and template.
Low scores → good, high scores → bad

Structural features that use:

- aa contacts: between 2 closest atoms or between specific aa (beta carbons ← orientation side chain).
- aa exposure: hydrophobic aa in the core, hydrophilic aa in the surface.

TUTORIAL

Step 1: find templates

First step in homology modeling is always to find templates → BLAST.

```
$blastp -query P11018.fa -db ~/Documents/databases/pdb_seq -out P11018.out
```

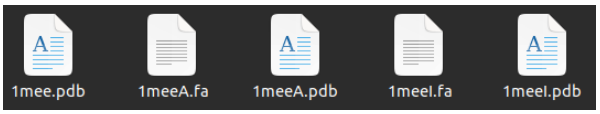
Then, go to PDB and look for the first PDB ID obtained with BLAST. Download the file in pdb format.

Use PDBtoSplitChain to split one PDB file in different chains and correct the PDB format. Also provides us the fasta sequence of each one of the chains.

```
$perl ~/Documents/perl_scripts/PDBtoSplitChain.pl -i 1mee.pdb -o 1mee
```

```
$perl ~/Documents/perl_scripts/PDBtoSplitChain.pl -i [template.pdb] -o [prefix]
```

We will obtain multiple files



Step 2: build a protein model

Obtain models for our target using MODELLER, we need:

- Target file
- Alignment file: alignment between target and template (.pir)
- Script file (modeling.py): execution commands for MODELLER.

```
$cat P11018.fa > target_template.fa

$cat 1meeA.fa >> target_template.fa
```

Execute ClustalW to run the alignment.

```
$clustalw target_template.fa
```

Modify the format of the MSA obtained with ClustalW (from .aln to .pir)

```
$perl ~/Documents/perl_scripts/aconvertMod2.pl -in c -out p <target_template.aln>target_template.pir
```

```
$perl ~/Documents/perl_scripts/aconvertMod2.pl -in c -out p <[file.aln]>[output.pir]
```

Modify the modeling.py file

```
# Homology modeling with multiple templates
from modeller import *              # Load standard Modeller classes
from modeller.automodel import *    # Load the automodel class

log.verbose()    # request verbose output
env = environ()  # create a new MODELLER environment to build this model in

# directories for input atom files
env.io.atom_files_directory = ['.', '../atom_files']

a = automodel(env,
              alnfile  = 'target_template.pir', # alignment filename
              knowns   = ('1meeA'),     # codes of the templates
              sequence = 'P11018')              # code of the target
a.starting_model= 1                 # index of the first model
a.ending_model  = 2                 # index of the last model
                                    # (determines how many models to calculate)
a.make()                            # do the actual homology modeling
```

`$mod10.5 modeling.py`

XXXX.B9999000N.pdb → produced models, where XXXX is the input name and N is the number corresponding to the different models generated by a MODELLER run.

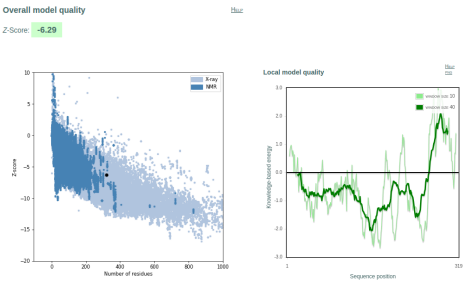
XXXX.log → information of the MODELLER execution. Look here in case of error.

Step 3: evaluating our models with prosa

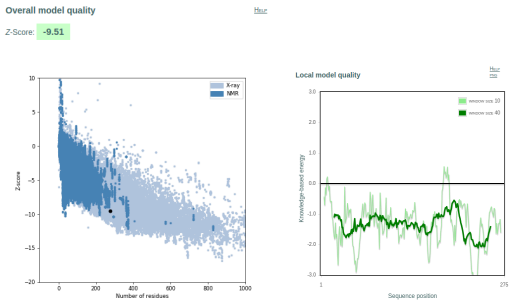
Once we have our model, we should evaluate its quality to check if it's a reliable model. Statistical potentials.

Use the target and the template to be able to compare.

Target:



Template:



Energies are lower in the template model.
Our model is not correct.

Step 4: evaluating a set of threading models

Open the models with prosa and compare between them.