DOCUMENTATION

Contents

1. Description

1.1 Purpose

The aim of MODICT is to compare *in silico* generated three-dimensional (3D) models of wildtype and mutated proteins and give a score as well as a graphical output based on how deleterious that specific mutation is. Several different mutations can be compared to other mutations. Generated results include both semi-quantitative (MODICT score) and qualitative graphical output to see the distribution of amino acids with different properties.

We believe that this is a good opportunity for the users to look at their proteins from different perspectives. Observed differences in mutations can guide searching for alternative mutations having the same effect or directing *in vitro/in vivo* studies.

1.2 Methodology

The main idea of MODICT is to superimpose a wildtype and a mutated model of the same protein and generate a score from the RMSD (Root Mean Square Deviation) values. The score generation process is explained in detail in the algorithm section of the MODICT paper.

1.3 Prerequisites

- You will need to have your own wildtype and mutated 3D protein models. You can easily submit your raw protein sequence in fasta format to automated servers like I-TASSER, PHYRE2, MODELLER, ETC.

- You will also need to be able to open these models in a .pdb editor like SWISSPDB, CHIMERA, PYMOL OR YASARA and generate RMSD values. A script for RMSD generation for SWISSPDB is included in the MODICT package.
- You need to have a perl interpreter with IPC::System::Simple and GetOpt::Long modules installed.
- MODICT itself is a very small program for which you need only less than a Mb of space on your drive.

1.4 Dataset

The test dataset includes proteins from distant families like *tubb2b, btd, pah, ren, acadm, tmem, smpd1* as shown in table S1 in the MODICT paper. Several mutations from each protein have been tested with MODICT both with and without the use of conservation scores.

1.5 Tested platforms and known issues

MODICT had been tested on MacOS X 10.9.4, Windows 7 and Linux (Ubuntu). The only drawback is that in versions of Perl newer than 5.16 the smartmatch ability has been classified as experimental. This will lead to warnings during the running of MODICT which however do not interrupt the execution of the program itself.

2. Installation and Directory Structure

2.1 Installation

- Unzip the contents of MODICT.RAR into any drive.
- Microsoft Windows users need to install a perl interpreter (e.g. strawberry perl http://strawberryperl.com)
- Both Linux and Microsoft Windows users have to install the IPC::System::Simple module, preferably from CPAN (*http://search.cpan.org/~pjf/IPC-System-Simple-1.25/lib/IPC/System/Simple.pm*). The IPC::System::Simple module is crucial for ITERATOR but not for MODICT itself.
- Additionally users will have to install the Getopt::Long (http://search.cpan.org/~jv/Getopt-Long-2.42/lib/Getopt/Long.pm) module for both MODICT and iterator.

2.2 Directory Structure

The MODICT folder consists of 4 sub-directories:

"./PROGRAM" : contains core scripts and templates for running MODICT

"./ROC" : contains script and demo files for the generation of ROC plot data

"./SAMPLE" : contains the files for the three examples described in the paper (BTD, PAH and Renin). These examples can be used to get familiar with the program.

"./DOCUMENTATION" : contains the documentation and the printed version of the MODICT paper.

Upon inspection you will see the following directories under PROGRAM:

./Core: contains the main scripts.

./Essentials: contains the automatically generated customized answers to questions asked by MODICT together with a template file for d3.js. Be careful not to modify this template file.

./Input : contains the conservation, .fasta and .rmsd files for your protein. It's from this folder that MODICT takes its input. All these files that you provide must be text files with ".txt" extension.

./Output: This folder will contain the final MODICT results. If ITERATOR was used, it will also contain the graphical output generated by d3.js using the template file located in ./Essentials.

3. Generating RMSD values

RMSD values are core to the algorithm of MODICT. RSMD stands for '*Root Mean Square Deviation*' and is a measure of the distance between two selected groups in a 3D space. In SWISS-PDB it is named as RMS under the "Fit" menu. Imagine 2 groups A and B in 3D space, each with *n* submolecular particles having coordinates (x_i,y_i,z_i). Then the RMS is roughly equal to:

$$\sqrt{\frac{\sum_{i=0}^{n}\left((x_{Ai}-x_{Bi})^2\ +(y_{Ai}-y_{Bi})^2\ +(Z_{Ai}-Z_{Bi})^2\ \right)}{n}}$$

Beware that this definition of RMS is dependent on the fit of the models or more generally the distance between the groups. Therefore rotations and translations to minimize the sum are NOT computed.

Instead models are superimposed with the choice of your program BEFORE the RMS calculation.

You can generate these values by selecting corresponding amino acids in a mutated protein and its wildtype counterpart AFTER superimposing them. For instance you can select Gly10 of mutated and then Gly10 of wildtype and then calculate RMS. Now you should repeat this for every residue of your protein, one by one. This is where the script comes in handy to automate this whole process.

You can calculate RMS with a variety of programs like SWISS-PDB, YASARA, PYMOL, CHIMERA, etc. The only thing you have to make sure is to first superimpose and then calculate the RMS values. Below is an explanation of the script which automates this step when using SWISS-PDB.

3.1 Via Swiss PDB Viewer

In the following section I will demonstrate how to generate a file in the format that MODICT accepts using SWISS-PDB VIEWER. The individual steps for generating similar files with different programs might vary slightly but they will all share the same 2 primary steps:

- superimposition of 2 models (wildtype vs. mutated, mutation 1 vs. mutation 2, etc.).

- extracting residue by residue RMSD values.

First you will have to download and install the latest distribution of swiss pdb viewer from *http://spdbv.vital-it.ch/* according to the instructions provided on the website.

3.1.1. Model superimposition

- Open … .exe
- navigate to the to “MODICT/SAMPLE/BTD/i-tasser/control” folder
- Drag and drop the “wildtype.pdb” located in this folder into the program. As an alternative for ‘drag and drop’ you can also upload the file using the ‘open PDB file’ from the program’s FILE menu. Click OK to ignore warning messages.
- Drag and drop the second .pdb file, “mutated.pdb” located in the same folder and again ignore warning messages.

"Mutated.pdb" is actually the refined wildtype model, which means the wildtype pdb model was submitted to MODREFINER (http://zhanglab.ccmb.med.umich.edu/ModRefiner/) to obtain an enhanced model. You could have renamed the wildtype pdb as mutated and generated RMSD values of 0, but later on you will see that this is not very helpful on later stages. Each time you generate RMSD values, rename the first layer to "wildtype.pdb" and the other layer to "mutated.pdb".

- From the program's FIT menu, choose MAGIC FIT. A window pops up where you can choose which superimposition model you want to apply.

  -Carbon Alpha only

  -Backbone Atom only

  -Sidechain Atom only

  -All atoms

The above super imposition methods differ in the number of subgroups they take into account in implementing the RMSD formula. The 'Carbon alpha' method takes only into account the alpha carbon when calculating RMSD, whereas as you go down the list, the number of atoms to average before calculating RMSD increases. Overall this does not make a big difference. However when you are comparing MODICT scores from two or three RMSD files you generated, you need to stick to one choice to make sure your error margins do not vary much.

So the important point here is to stay consistent for all tested models of your protein. For example, suppose you have:

- 1 wildtype model
- 1 refined wildtype model (from ModRefiner)
- 1 refined test mutation
- 1 refined **known** deleterious/benign mutation

You will normally form 3 pairs which are wildtype/wildtype-refined, wildtype/test-refined, wildtype/deleterious-refined. In all of these pairs you should choose the same superimposition method because you do not want the small changes in RMSD calculation method to be understood as an impact of mutation.

Instead of wildtype-refined you can also use a known benign mutation. However Model refinements have very small overall RMSD differences when compared to the original model whereas a benign mutation would have a larger change due to modeling algorithm. This will make it harder for your mutation to be classified as deleterious. I would like to further explain this with an example. Imagine you have generated a negative control score by super imposing wildtype/wildtype-refined and your score came out to be 0.3. And then imagine you also generated a score from wildtype/known-deleterious-refined mutation and it is 2. This is a condition identical to line 81-89 in ROC plot script located in "./MODICT/ROC/". The threshold for a mutation to be classified as deleterious in this case is ~1.38. Now imagine that instead of using a wildtype-refined model, you used a benign model. Let's assume that the new score is 0.5. In this case, supposing your positive score stays the same, the new threshold would be ~1.45. As you can see although there is around 66 percent increase in negative control score, it is minimally reflected on the threshold which is around 5 percent increase. However, if your test score would happen to fall between 1.38 and 1.45, in the first case it would be classified as deleterious whereas in the second as benign. Supposing your test score can fall anywhere between 0.3 and 2, changing your choice for negative control between wildtype-refined and benign would cause around 4 percent of your test scores to shift classification. This error margin (the 4 percent) is one that you cannot avoid if you want to use a known benign model.

- Below the fit options there are two dialog boxes to define the two layers for the superimposition. The first box is to define the reference model, which is in this case the wildtype model and it will stay as that the rest of the trio. (A trio is a group of *wildtype/wildtype-refined* + *wildtype/test-refined* + *wildtype/known-utation-refined* **OR** *wildtype/known-benign* + *wildtype/test* + *wildtype/known-mutation(benign or deleterious)*)
- Click "OK" and your models will be superimposed. If your models are far from each other you will see them overlap in the screen after clicking "OK". If the models are already close it might be hard to see if superimposition is done. It should normally be done after you click "OK".

3.1.2 calculating and extracting RMSD values

- Go to “./MODICT/PROGRAM/Core/”. Drag the script, named “RMSD.txt” to SWISS PDB VIEWER and a new window opens which shows this script that will generate residue by residue RMSD values. Drag might not work properly in MacOS so your second option is to go to File -> Run Script…
- Click on “please do” on the first line of the script. The program will then start writing output to “./SwissPDB-Viewer/SPDBV_4.10_PC/usrstuff/output.txt”. Congratulations, you now have your RMSD values for the wildtype/mutated model. Copy the output.txt file to “./MODICT/PROGRAM/Input/”. This file will be used as for MODICT.

****Warning: If you are using Mac, you might have a typo like “beginnin$j” instead of ‘beginning’ near the end of output.txt. It does not make a change in your results.****

3.1.3 RMSD output description

It is important to understand the standard output that MODICT understands:

Some sentence with “RMSD” in it.

……

Values

……

Some other sentence with “RMSD” in it.

Comments or anything with any number of lines.

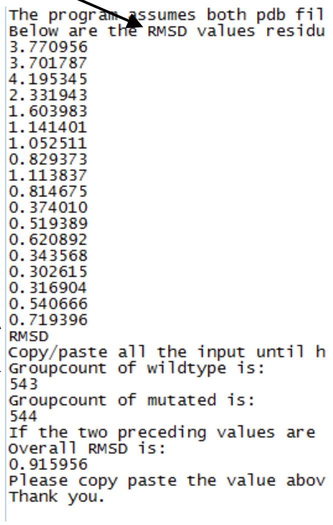
Some sentence with “groupcount” in it.

1 Groupcount Value.

Some other sentence with “groupcount” in it.

1 Groupcount Value.

Comments or anything with any number of lines.



```
The program assumes both pdb fil
Below are the RMSD values residu
3.770956
3.701787
4.195345
2.331943
1.603983
1.141401
1.052511
0.829373
1.113837
0.814675
0.374010
0.519389
0.620892
0.343568
0.302615
0.316904
0.540666
0.719396
RMSD
Copy/paste all the input until h
Groupcount of wildtype is:
543
Groupcount of mutated is:
544
If the two preceding values are
Overall RMSD is:
0.915956
Please copy paste the value abov
Thank you.
```

Some sentence with "overall" in it.

1 Overall RMSD Value.

Comments or anything with any number of lines.

The bold regions above are "blocks" that MODICT reads and extracts data from. The order of these blocks is NOT important. Open "./SwissPDB-Viewer/SPDBV_4.10_PC/usrstuff/output.txt" and it will be clearer what the format looks like.

If you open any model on SWISS PDB VIEWER and go to "Wind -> Control Panel" to visualize the residues, you will see that they are numbered starting from 1 till X (543 in the BTD example). Since most programming languages start counting from 0, they would end up with 1 less than the maximum amino acid count. This is why in the script "RMSD.txt" at line 26 we subtract 1 from "$minresiduecount".

One remark here is that if you carefully look at the models, some of them have end groups like "OXT" (oxygen) or HT (hydrogen). Compare "wildtype.pdb" in "MODICT/SAMPLE/BTD/i-tasser/control/" with "mutated.pdb". You will see that the mutated has an OXT group. This is the reason why two groupcount values exist in the standard input format for MODICT. MODICT takes the smallest of these values as amino acid count. Sometimes it is possible that both models have these "OXT" groups, this causes MODICT to overestimate the amino acid count by 1, which is an insignificant change.

You probably saw the "RMSD_original.txt" in "MODICT/PROGRAM/Core/". This is the original version of the script I worked with. Compare "RMSD.txt" and "RMSD_original.txt". The biggest difference is in line 26 in "RMSD_original.txt" which corresponds to line 45 in "RMSD.txt". "RMSD.txt" automatically calculates the upper limit for the number of amino acids whereas in "RMSD_original.txt" you will have to enter this manually. To exemplify from BTD, line 26 in "RMSD_original.txt" has to change. Look at both "wildtype.pdb" and "mutated.pdb". They both have 543 residues. But as mentioned earlier, the program takes them as an array and the numbering in script should start from 0. Therefore you have to replace the "902" in line 26 of "RMSD_original.txt" by 1084 ((543-1)*2). The rule of thumb is:

(Common maximum number of amino acids between two layers - 1) x 2

It is up to you which script to use. They both do the same thing. I wrote “RMSD.txt” to evade confusion in people. However there might be bugs in this one. If you find one please send an email at: *itanyalc@vub.ac.be*

4. Conservation and fasta files

Other then the RMSD values, you will need conservation .txt and sequence .fasta files. The fasta file is needed to generate a graphical output and the conservation file is needed for generating MODICT scores. MODICT can also generate scores without conservation information but conservation allows attaining different weight scores to aminoacid pairs. If you want to generate a MODICT score based on the physical properties only than you can omit the conservation data.

4.1 Conservation

A conservation .txt file is generated by aligning protein sequences from different organisms. Use Uniprot to select “reviewed” protein sequences from different organisms and align them. Using jalview, import the values corresponding to your sequence of interest (there might be gaps) in a simple .txt file. Under “./MODICT/SAMPLE/BTD/” you will see a conservation .txt file for the BTD protein. The format is simple: 1 value per line per amino acid. It is very important NOT to have extra blank lines at the end of this .txt file, since it might cause check problems with MODICT. MODICT performs a check everytime you provide an input file, if this check is not successful you will not get a result.

4.2 Fasta

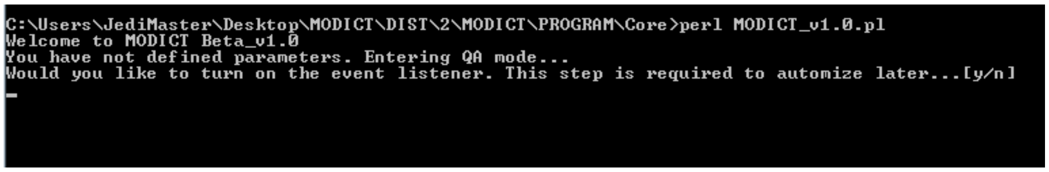
The .fasta file is the standard file format you get from genome browsers (eg. UCSC, Ensembl, NCBI) or Uniprot to represent a DNA/protein sequence. The fasta file used for MODICT is the aminoacid sequence of a protein. Just copy and paste everything in its entirety to a .txt file. Header lines are not a problem, they are discarded by MODICT. If your fasta file has an extension other than “.txt”, you should change it to “.txt”.

5. Using MODICT

5.1 The Basics

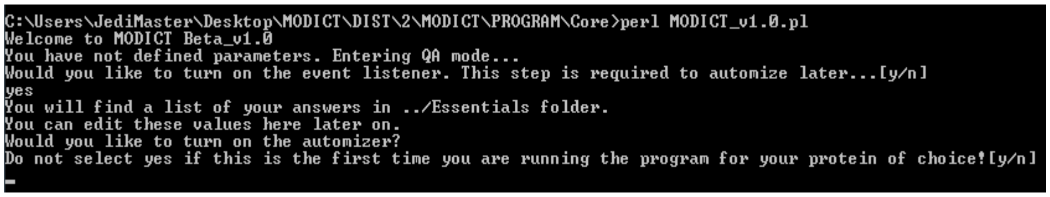
You can run MODICT with parameters that will go into arguments, however we will first look into direct use without arguments.

- First open a terminal window and go to "./MODICT/PROGRAM/Core/".
- Invoke the master script by writing: "perl MODICT_v1.0.pl".



```
C:\Users\JediMaster\Desktop\MODICT\DIST\2\MODICT\PROGRAM\Core>perl MODICT_v1.0.pl
Welcome to MODICT Beta_v1.0
You have not defined parameters. Entering QA mode...
Would you like to turn on the event listener. This step is required to automize later...[y/n]
_
```

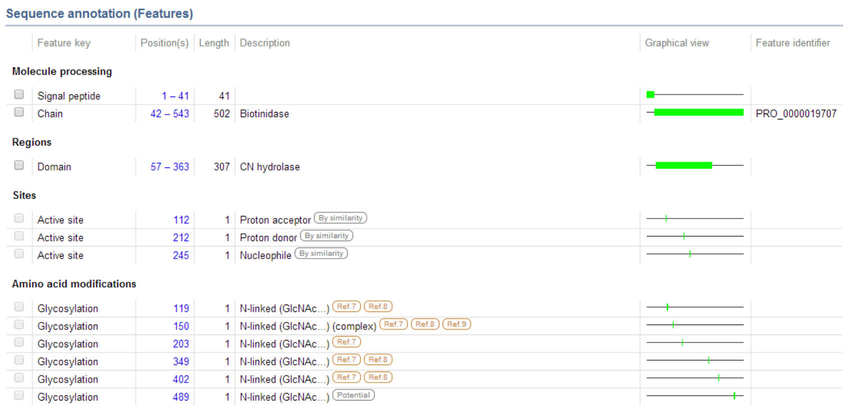
- In the next steps, you'll be asked some questions which you should answer with either 'yes' or 'no'.
  1. The first question is if you want to turn on the eventlistener. Eventlistener stores your answers to the questions asked, so that they can be recalled and used again when you run the program a second time. So write 'yes'. Any answer that starts other than small or capital Y is taken as no.



```
C:\Users\JediMaster\Desktop\MODICT\DIST\2\MODICT\PROGRAM\Core>perl MODICT_v1.0.pl
Welcome to MODICT Beta_v1.0
You have not defined parameters. Entering QA mode...
Would you like to turn on the event listener. This step is required to automize later...[y/n]
yes
You will find a list of your answers in ../Essentials folder.
You can edit these values here later on.
Would you like to turn on the automizer?
Do not select yes if this is the first time you are running the program for your protein of choice![y/n]
_
```

  2. To the second question about turning on the automizer, answer 'no' now because you need to run MODICT at least once for automiser to work. Once turned on, automiser takes the answers stored in eventlistener and provides them without the user having to type them again.
  3. Next the program will ask you the name of the file with RMSD values. For this example this file is called to "BTD_trial_control.txt" which is located in the "input" directory. Alternatively you can use your own file generated earlier. Rename the "output.txt" to "BTD_trial_control.txt".
  4. The next questions all deal with the domains present in your protein. For BTD we know from Uniprot that residues

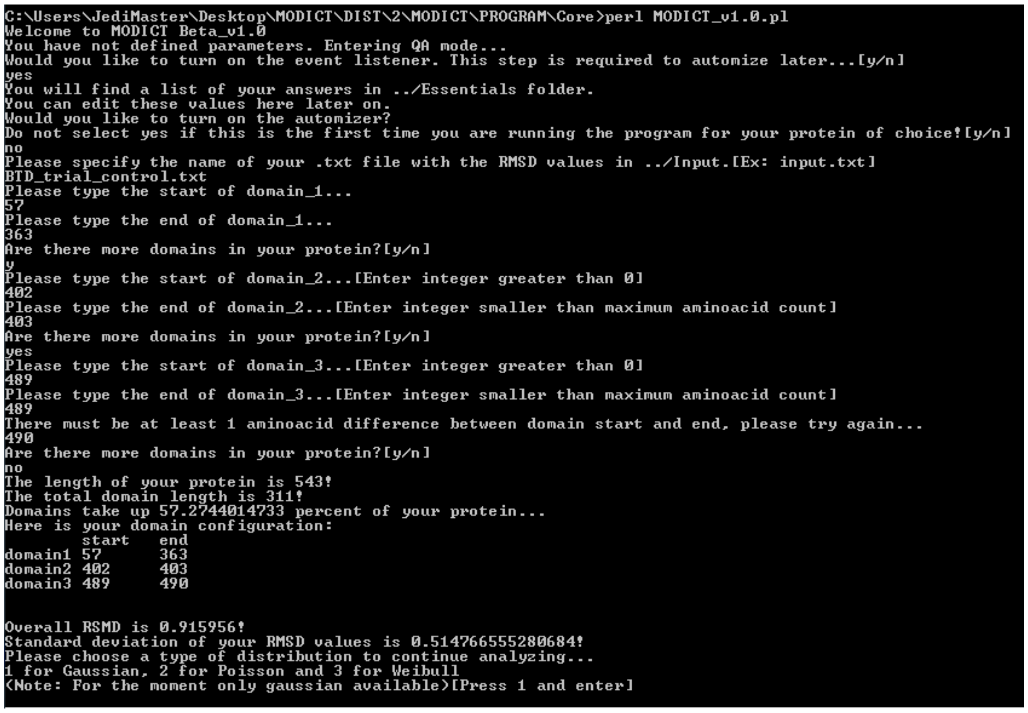
57-363, 402-403 and 489-489 are the domain regions. Below is a recent screenshot of Uniprot showing the aforementioned information (there might be small differences due to the updates).



Sequence annotation (Features)

	Feature key	Position(s)	Length	Description	Graphical view	Feature identifier
Molecule processing						
☐	Signal peptide	1 – 41	41			
☐	Chain	42 – 543	502	Biotinidase		PRO_0000019707
Regions						
☐	Domain	57 – 363	307	CN hydrolase		
Sites						
☐	Active site	112	1	Proton acceptor By similarity		
☐	Active site	212	1	Proton donor By similarity		
☐	Active site	245	1	Nucleophile By similarity		
Amino acid modifications						
☐	Glycosylation	119	1	N-linked (GlcNAc...) Ref.7 Ref.8		
☐	Glycosylation	150	1	N-linked (GlcNAc...) (complex) Ref.7 Ref.8 Ref.9		
☐	Glycosylation	203	1	N-linked (GlcNAc...) Ref.7		
☐	Glycosylation	349	1	N-linked (GlcNAc...) Ref.7 Ref.8		
☐	Glycosylation	402	1	N-linked (GlcNAc...) Ref.7 Ref.8		
☐	Glycosylation	489	1	N-linked (GlcNAc...) Potential		

One small remark here is that at the time the paper was written the user could submit same start and end coordinates such as 489-489. However I find it misleading (MODICT works in residue pairs, see algorithm step 1) so they cannot be the same. The closest you can enter is 489-490. Try to end up with a screen similar to below:



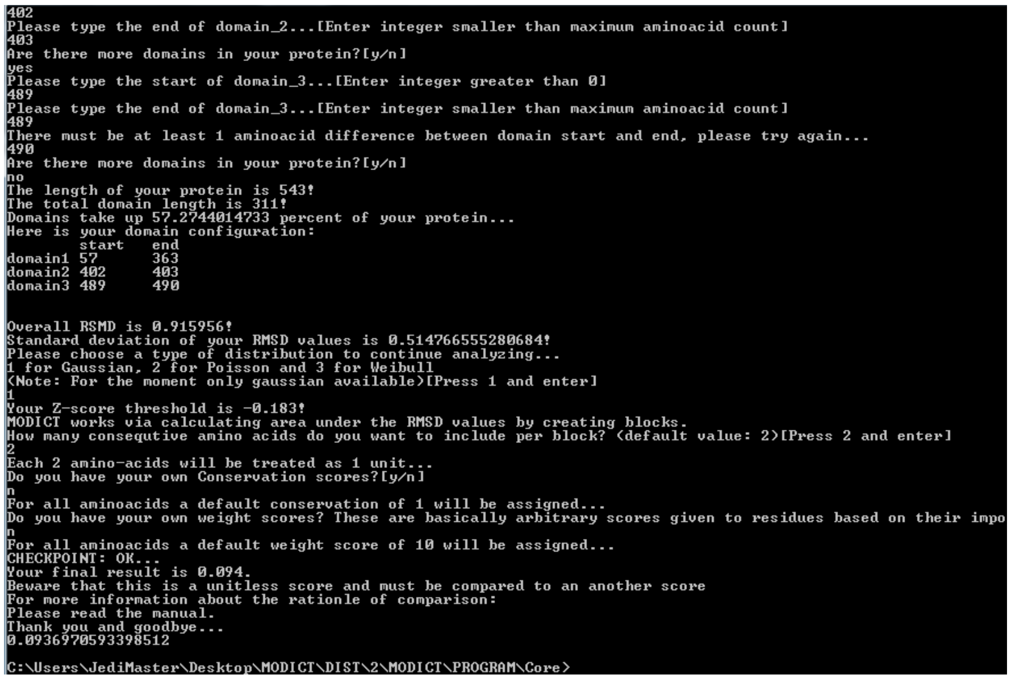
```
C:\Users\JediMaster\Desktop\MODICT\DIST\2\MODICT\PROGRAM\Core>perl MODICT_v1.0.pl
Welcome to MODICT Beta_v1.0
You have not defined parameters. Entering QA mode...
Would you like to turn on the event listener. This step is required to automize later...[y/n]
yes
You will find a list of your answers in ../Essentials folder.
You can edit these values here later on.
Would you like to turn on the automizer?
Do not select yes if this is the first time you are running the program for your protein of choice![y/n]
no
Please specify the name of your .txt file with the RMSD values in ../Input.[Ex: input.txt]
BTD_trial_control.txt
Please type the start of domain_1...
57
Please type the end of domain_1...
363
Are there more domains in your protein?[y/n]
y
Please type the start of domain_2...[Enter integer greater than 0]
402
Please type the end of domain_2...[Enter integer smaller than maximum aminoacid count]
403
Are there more domains in your protein?[y/n]
yes
Please type the start of domain_3...[Enter integer greater than 0]
489
Please type the end of domain_3...[Enter integer smaller than maximum aminoacid count]
489
There must be at least 1 aminoacid difference between domain start and end, please try again...
490
Are there more domains in your protein?[y/n]
no
The length of your protein is 543!
The total domain length is 311!
Domains take up 57.2744014733 percent of your protein...
Here is your domain configuration:
        start   end
domain1 57      363
domain2 402     403
domain3 489     490


Overall RSMD is 0.915956!
Standard deviation of your RMSD values is 0.514766555280684!
Please choose a type of distribution to continue analyzing...
1 for Gaussian, 2 for Poisson and 3 for Weibull
(Note: For the moment only gaussian available)[Press 1 and enter]
```

You cannot enter values smaller or greater than the maximum/minimum residue count. You also cannot enter same start and end positions as well as negative numbers.

The overall RMSD value and a standard deviation are now being calculated and printed on the screen. Since entering identical start and ends were legitimate in previous versions of MODICT (like 489-489), your final answer might slightly differ from the value reported in the paper at table S1.

1. Another question is then asked which handles about the distribution you want to use in the further analysis. There are three options listed, however for now only the Gaussian distribution is embedded, so the answer should be ‘1’. We assume Gaussian distribution (in any case the effect of distribution choice in trios is very very small). The main aim of the distributions is to estimate a threshold RMSD value.
2. Choose the default number of amino acid pairing: 2. (This feature might not be fully functional yet). The default value is 2. It defines the number of consecutive amino acids to average as in step 1 of MODICT algorithm.



```
402
Please type the end of domain_2...[Enter integer smaller than maximum aminoacid count]
403
Are there more domains in your protein?[y/n]
yes
Please type the start of domain_3...[Enter integer greater than 0]
489
Please type the end of domain_3...[Enter integer smaller than maximum aminoacid count]
489
There must be at least 1 aminoacid difference between domain start and end, please try again...
490
Are there more domains in your protein?[y/n]
no
The length of your protein is 543!
The total domain length is 311!
Domains take up 57.2744014733 percent of your protein...
Here is your domain configuration:
        start   end
domain1 57      363
domain2 402     403
domain3 489     490


Overall RSMD is 0.915956!
Standard deviation of your RMSD values is 0.514766555280684!
Please choose a type of distribution to continue analyzing...
1 for Gaussian, 2 for Poisson and 3 for Weibull
(Note: For the moment only gaussian available)[Press 1 and enter]
1
Your Z-score threshold is -0.183!
MODICT works via calculating area under the RMSD values by creating blocks.
How many consequtive amino acids do you want to include per block? (default value: 2)[Press 2 and enter]
2
Each 2 amino-acids will be treated as 1 unit...
Do you have your own Conservation scores?[y/n]
n
For all aminoacids a default conservation of 1 will be assigned...
Do you have your own weight scores? These are basically arbitrary scores given to residues based on their impo
n
For all aminoacids a default weight score of 10 will be assigned...
CHECKPOINT: OK...
Your final result is 0.094.
Beware that this is a unitless score and must be compared to an another score
For more information about the rationle of comparison:
Please read the manual.
Thank you and goodbye...
0.0936970593398512

C:\Users\JediMaster\Desktop\MODICT\DIST\2\MODICT\PROGRAM\Core>
```

Finally one can specify his/her own conservation and weight scores. For now answer ‘no’ to both questions.

The result is written the last 6 lines. The final result for this test is 0.094 which is close to the 0.092 value that was reported in table S1 from the paper. As I stated earlier the values differ because of a change in the script, which is not to accept equal start and end, which causes a slight shift in the Z score. The Z score is not important for the user, it is used to set threshold RMSD values for significance. The line after the checkpoint is the most important. The same result is also in the last line of STDOUT but unrounded. The reason for this is that the last line in STDOUT is used by the IPC::System and iterator to capture results.

Now, if you were to navigate to "MODICT/PROGRAM/Essentials/", you would see a file with the name "essentials_MODICT.txt". If you open it you will see that all of your answers to the questions are stored in this file. This is the effect of turning the eventlistener 'on' at the start of the program. This will also allow the user to work now with arguments (see next paragraph).

Generate RMSD files as shown previously (under point …) for mutations H447R and R209C and rename the output with RMSD values to "H447R.txt" and "R209C.txt" respectively. Copy/paste them in the "./MODICT/PROGRAM/Input". To run MODICT with arguments: write in the terminal:

perl modict_v1.0.pl --eventlistener off --automiser on --input H447R.txt

This is telling the program to turn off the eventlistener and turn on the automiser. H447R.txt is used as input file. Now you should immediately get the result, without having to answer any questions.

- For H447R.txt the final result should be a value around 0.632. For R209C it will be around 0.273.

5.2 Using conversation scores in MODICT

Now let's try to see the effect of conservation values on the MODICT score. If you haven't done this yet, go to "./SAMPLE/BTD/" and copy/paste "conservation.txt" and "fasta.txt" in your "./PROGRAM/Input" directory. To run MODICT, write in the terminal:

```
perl modict_v1.0.pl  --eventlistener off --automiser on --input BTD_trial_control.txt --conservation conservation.txt
```

The final MODICT score should be a value around 0.102. Try using conservation scores also for H447R and R209C. The final values should be around 0.613 and 0.278 respectively. The first step of the algorithm generates rectangles with width equal to aminoacid number and height equal to RMSD values. With the help of conservation scores certain rectangles receive higher weight or vice versa for their "height"(which is the RMSD value). Without the conservation, all rectangles receive the single conservation of 1. Conservation scores are only significant if there are differences between different rectangles, meaning that the absolute value of a conservation score is very little important. Test this by opening the "conservation.txt" file in excel and multiplying **every** residue by 1000. If you use this file for score calculation you will see that the rounded score is still the same 0.278.

There are two other examples described in the paper for which you can also find examples files in the SAMPLES folder, TUBB2B and Renin. The final MODICT scores that you should obtain can be found in table S1 of the MODICT paper.

5.3 Help Command and Possible Parameters

If you ever need to get a full list of parameters or the version, write in the terminal:

```
perl modict_v1.0.pl --need help
```

```
perl modict_v1.0.pl --need version
```

As a rule of thumb if you ever need to pass down a parameter you can use one of the following:

```
perl modict_v1.0.pl --parameter X

perl modict_v1.0.pl -parameter X

perl modict_v1.0.pl -parameter -X

perl modict_v1.0.pl --parameter –X

perl modict_v1.0.pl --parameter -------X
```

perl modict_v1.0.pl --parameter=X

perl modict_v1.0.pl -parameter=----X

All of the above are equivalent.

6. Using the iterator

6.1 Using the iterator from command line:

You have seen how to generate a MODICT score for a whole model as shown previously. It is also very important to know which residues in a given model are contributing to this score. It is even more interesting to know which combination of weight scores per residue generates the highest difference for a given set of models compared to another set of models. Iterator is designed to come up with an answer to questions like these. Previously you have generated 3 models: BTD control, R209C and H447R. Let's first find the regions that differ the most between a wildtype BTD protein and one with a mutated residue R209C.

First, keep in mind that iterator is a separate script that initiates multiple instances of MODICT. So in principle you can use a newer version of iterator with an old version of MODICT (or vice versa) if updates are not released synchronously. Right-click on "./PROGRAM/Core/iterator_v1.0.pl" and navigate to line 34. You will see a variable with the name "which_MODICT". Make sure that the value of this variable is set to the correct file name of the MODICT you intend to use.

- Open a terminal window and write: perl iterator_v1.0.pl
- Press enter to skip otherwise enter a path.
- You will be asked to point to a file with conservation scores. You have one in the input folder. Type: conservation.txt
- You will be asked to point to the .fasta file. Type: fasta.txt
- Here is the important part. ITERATOR will ask you to define stringency parameters. Stringency parameters tell MODICT how hard to look for better alternatives. This is done in two steps:
    - In the **first step** a reference group of models is specified and then a test group of models for score maximization is

specified. MODICT will run around 2000 times within the reference and test groups and will construct a standard deviation and a mean value. If it encounters a combination that is X standard deviations away from the reference mean, the first phase stops. This X is the first stringency parameter. So if you write 2, it will look for a combination that yields 2 standard deviations of difference from the reference mean. The procedure starts after at least 2000 repetitions of the program. Assuming RMSD scores follow Gaussian distribution, a one-tailed 2 standard deviations (SD) to the right is already 97.5th percentile of all possible weight score combinations. So it is a good value to start however for models that exhibit close MODICT scores to each other 2SD might be cumbersome to achieve. No matter what value you enter as threshold, the program will slowly lower the threshold limit until it is achieved.

- The second step builds its logic on this hypothesis:

"If a given combination of weight scores yields a higher difference between the test group and the reference group than the threshold X, than there must be a reason for it!"

The **second step** tries to refine it further by using a random number approach (not exactly in sense of randomness like in the first step, extreme values tends towards their extremities with a small chance of moving the other direction as well) with competing weight scores. There are 2 separate instances of weight score files: "iterate_MODICT.txt" and "iterate_MODICT_trial.txt". If one is better than the other, the program keeps it aside and continues changing or "mutating" the other one for **Y** repetitions. Once the other one is better, the cycle resets and switches. This creates a "less slanted" gradient to a "highly improbable to obtain by randomness" weight score combination. It is really reminiscent of natural selection, that's why I named it "-iterate_evolve" in the parameter list.

Just like the first step, the **second step** has a user set threshold too. For instance if you were to write 100, that means the program will look better alternatives until there is 100 standard deviations difference between the test group mean (or the test score if there is only one model) and the reference group mean. Of course we do not know how much we can maximize this difference so that's why I have introduced a loop limit. If you set the loop limit to 1500 for example, the program will iterate 1500 times to look for a better alternative between "iterate_MODICT.txt" and "iterate_MODICT_trial.txt". If it cannot, than it will automatically terminate and give you the results.

- There is one more thing, the refine mode. Since both first and second steps use a random number approach, results converge on certain regions of a given protein. For instance high weight scores will be given always between residues X and Y of a protein at each different run with a slight change in bordering residues. If you enter a refine limit like 5, it means the entire set of program flow will cycle 5 times and generate 5 different pairs of "iterate_MODICT.txt" and "iterate_MODICT_trial.txt". The residues that show a variation of more than 25% will be brought down to 0 or 1. The consistent regions will persist and their mean will be taken. Often the result of this is islets of residue pairs next to each other with high scores with low variability.

So there are 3 stringency parameters: 1st, 2nd and then the loop limit. These parameters will be asked from you if you run iterator without parameters. With parameters, you should write something like one of these:

```
perl iterator_v1.0.pl --conservation conservation.txt --fasta fasta.txt --
stringency 2/100/1500 --refine on
```

```
perl iterator_v1.0.pl --conservation conservation.txt --fasta fasta.txt --
stringency 2/100/1500 --refine off
```

```
perl iterator_v1.0.pl --conservation conservation.txt --fasta fasta.txt --
stringency 2/100/1500 --refine 3
```

In the examples shown above the first stringency parameter is 2 standard deviations, the second is 100 and the loop limit is 1500. In the first example the refine mod is on and by default the whole program will initiate 11 times (the minimum amount of runs for the possibility of a residue to receive all possible different weight scores, 0-10). In the second one the refine mod is off (it will display "parameter is not clear or refine mod off") and the third one the refine limit is 3.

Stringency parameters can also be specified as h (high), m (medium), l (low), vh (very high), vvh (very very high)… To have an idea what these tables refer to please see line 129 of iterator.pl. For example vvl refers to 0.5 SD difference for the first phase, 2.5 SD difference for the second phase and 100 loop limit. A “-m/m/m” is a good start to have an idea of your protein.

- For the BTD case, enter a stringency parameter of 1.75 for the first phase. For the second phase enter 100 or whatever number you think is sufficiently large. The reason why this number does not make a difference too much is because if the loop limit is reached, say after 1500 iterations still no better alternatives were found, the program will automatically terminate. For the loop limit enter 1500.
- Next, the program will ask for the refine mod. You can either set the refine parameter to off, on or any number greater than 2. For this instance, set refine parameter to be 3.
- Now comes the important part. The program will ask you to define the reference group. Enter first your control sample:

BTD_trial_control.txt

Then it will ask you if there is another reference sample. Enter “yes”, and enter:

R209C.txt

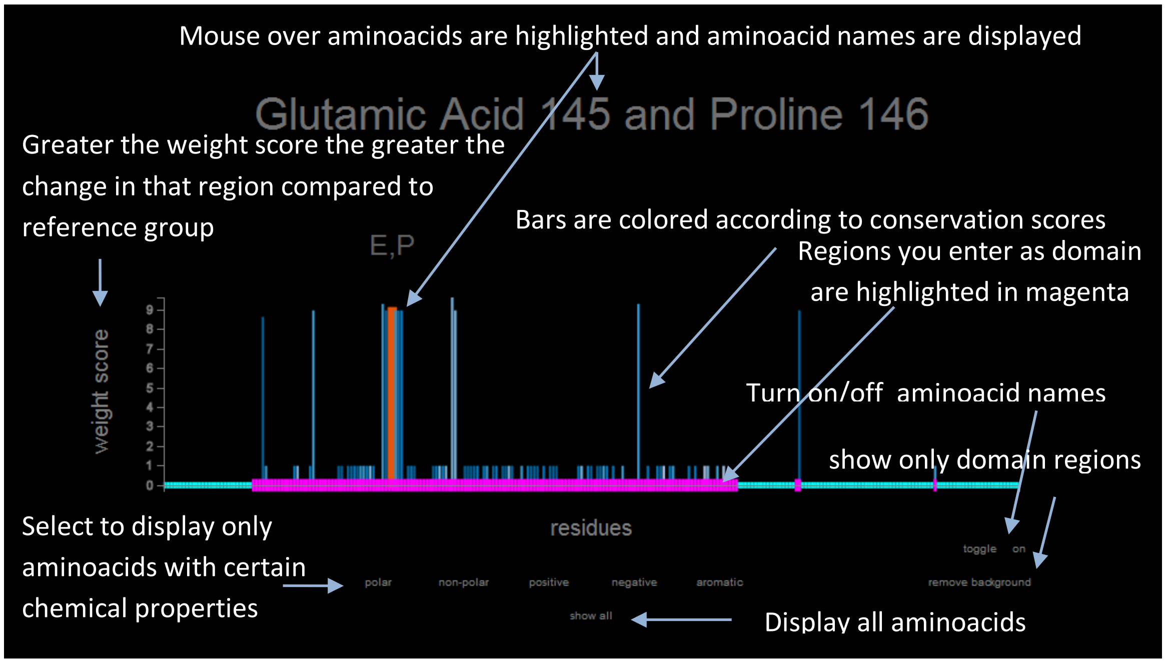
The reason why you enter R209C.txt as reference is that a mean and standard deviation can’t be defined with only 1 model. The algorithm will fix the standard deviation after the first phase and maximizes difference also in means of ratio of test/reference so the fact that R209C.txt is specified as reference does not hinder the algorithms ability. But as a matter of fact there must be at least 2 models in the reference group.

Once you enter R209C.txt as reference the program will ask you if there are more reference models. Answer “no” and then specify the test file:

R209C.txt

- Again the program will ask you if there are other test files. Answer ‘no’ and then the program will start iterating. The whole procedure will take well over 2 hours.

➢ At the end you will have 2 files in the "./Output" folder, an "iterator_results.txt" with the score combination and the "Graphical_Output.html" to visualize the results. Other weight score combinations used during the iteration process is placed in "./Essentials/Dump" folder.


Mouse over aminoacids are highlighted and aminoacid names are displayed
Glutamic Acid 145 and Proline 146
Greater the weight score the greater the change in that region compared to reference group
E,P
Bars are colored according to conservation scores
Regions you enter as domain are highlighted in magenta
weight score
Turn on/off aminoacid names
show only domain regions
Select to display only aminoacids with certain chemical properties
residues
toggle
on
polar
non-polar
positive
negative
aromatic
remove background
show all
Display all aminoacids

The example you see on MODICT paper figure 6 is generated without using the refine mod. Compare once you have the results for R209C. The region you see with consecutive blocks of high scores between residues 138-150 should reappear because there is a significant change in this region that creates a high MODICT score difference between reference and test models.

Run the whole procedure a second time, rename the "Graphical_Output.html" and compare 2 independent runs, the region 138-150 mentioned previously will come back every time due to the significance of difference in these regions. I have included such a sample run in the "./Output/Documentation" folder. There is the first run and the second run. Compare these with your results.

Lastly, run the iterator with the same configuration without the refine mod and compare with MODICT paper figure 6. This will clarify further what the mode does. To run, use the following parameters this time:

```
perl iterator_v1.0.pl --conservation conservation.txt --fasta fasta.txt --stringency 1.8/100/1500 --refine off
```

Specify the reference and test models as mentioned previously and compare the results. You will realize that this, with all the background resembles more of the results in figure 6D, top layer. The rule of thumb is: the higher the amount of repetitions, the less background there will be. However, using a high amount of repetitions might be very time consuming. If you want to have an idea of differences in your reference and test models, run iterator with refine mod off. If you want to have clear cut boundaries of where the most significant changes occur than you can turn refine mod on with the number of repetitions you specify. Do not forget to rename or move the result files in "./Output/" to another folder otherwise they will be overwritten.

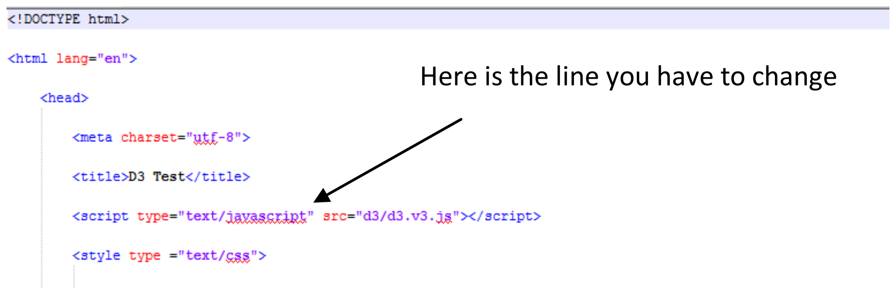
One important thing is that the "Graphical_Output.html" generated by iterator can only be opened if it is located in "./Output" folder. This is because the relative path of the script "d3.js" is also located in this folder. However if you want to be able to open these ".html" files from any location than right-click on the file and edit the line as shown below:

Change this at line 11:

```
<script type="text/javascript" src="d3/d3.v3.js"></script>
```

As:

```
<script src="http://d3js.org/d3.v3.min.js" charset="utf-8"></script>
```



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Test</title>
        <script type="text/javascript" src="d3/d3.v3.js"></script>
        <style type ="text/css">
```

If you want all future generated files to be opened regardless of their location than open "./Essentials/Template.html" and make the above changes at line 6. Don't forget that in order for the above changes to work you need an internet connection.

6.2 Understanding the Iterator

Previous example showed how to generate iterator results for 2 models (negative control and positive control) only. It is important how to proceed for larger models. To better illustrate, lines 226 of iterator.pl should be discussed:

```
until (($mean_test >= ($mean+$threshold))&&($iterations>2000)&&((grep
  {$reference_values[$_] > $mean_test} 0..$#reference_values) == 0)) {
```

….

This "until" loop makes sure of 3 things:

1. The mean of the test samples are bigger than the mean of the reference plus the threshold you set (1st stringency parameter)
2. The number of iterations must be bigger than 2000 to make sure that the mean is not changing too much.
3. All the individual scores in reference group should be smaller than the individual test scores in the test group.

Therefore it is important that you do not put different competing models in both reference and test groups. To better illustrate, imagine there are 3 models, wildtype, A and B. Let's say with uniform weight scores of 1, A is much larger than B. If you were to place A, B and wildtype in reference group and than A and B in the test group, since for most generated random weight score combinations A will be bigger than B, the condition in until loop will never be met. Therefore it is better to compare A and B individually like in the previous example.

If you have another mutation C, you can put wildtype and C in the reference group and than A and B in the test, this is totally ok unless you know that C is much larger than both A and B.

Another example is given from *PAH* in the article at figures 9 and 10 where *PAH*^{Y414C} is compared to less severe mutations that lead to preservation of 50 percent or more of enzyme activity. This comparison was made by placing in the reference group the *PAH*^{E390G}, *PAH*^{v245A}, *PAH*^{D415N}, *PAH*^{R408Q} and *PAH*^{Y414C} in the test group (*PAH*^{P211T} was filtered due to being 2D away from the mean of other mutations listed in MODICT article table 1.). The 1st stringency parameter was set to 0.5SD, the 2nd stringency parameter was set to 50SD and the loop limit was set to 1500. The refine limit was also set to 3. If you run iterator with these parameters you will realize that the 1st phase of the algorithm will reduce the SD threshold all the way down to 0SD because *PAH*^{v245A} will most of the time have higher scores than *PAH*^{Y414C}. Although the 1st phase fails the 2nd phase will take over and in each run it will bring the score of *PAH*^{Y414C} 3SD above the mean of the reference group. You might even come across results that yield higher score in *PAH*^{Y414C} compared to *PAH*^{v245A} . I have included 2 separate runs with the same configuration, please compare them. Click on the aromatic residues only for instance, each time you will see that the middle islet of C,Y 203-204, I,F 209-210 and K,Y 215-216 will come back. You will also have peaks near the C-terminal region.

Iterator is designed to give you an idea of where the most distortion with respect to wildtype and the known model takes place in your mutated protein. As you increase the refine limit, the reproducibility may increase up to the level of aminoacid pairs, however this is not what iterator algorithm has initially designed for. To achieve such accuracy, the refine limit has to be set relatively high (6 or above). A typical run of iterator with refine limit 3 such as in the *PAH* example takes around 2.5 hours.

6.3 Why random number approach?

I want to speak a bit about the approach used in iterator. Suppose a case where we have 50 reference models and 50 test models to compare. Imagine that we do not use a random number approach but some sort of linear computation algorithm where a residue is given all possible scores 1 by 1 and local maxima is computed. There are 2 problems related with this approach:

1. The computation time: As the protein length grows the number of points to compute for a single model becomes Protein Length x 11 (Number of aminoacids times the number of all possible attainable weight scores).
2. Linear computation requires a starting point. For instance you can start from an initial .txt file of all zeros or ones. Then work your way up from there. The main problem is that the weight score combinations can act as a network of connected entities within reference and test groups. For instance, giving a score of 7 to residue number X can lead to a local maximum. However attaining score 9 to residues Y and Z further upstream may lead to a higher score provided that X is 1. Even than a higher score might be attained if residue P further upstream is given score 9 given that Z is set to 1. There is no way a linear maximum algorithm would find the

configuration <1,9,1,9> for X, Y, Z, P if it encounters <9, N, N, N> in the first place. A random number approach that uses a reversible cycle of competition between 2 files foregoes this problem of related numbers.

7. Using the ROC plot.

7.1 The purpose of ROC plot

Suppose that you produce your negative control, test and known mutation scores from MODICT. Let's call these values S_C, S_T and S_K respectively. There are 3 possibilities for your known mutation, it is either benign, partially deleterious or deleterious. IF,

Your **known mutation is deleterious:**

Calculate an imaginary benign (S_I) such that:

$$S_I = \frac{2 * S_K + 3.24 * S_c}{5.24}$$

The equation above is equivalent to *ROC.pl line 102*. Now we define 3 thresholds T_1,T_2 and T_3:

$$T_1 = \frac{(S_C + S_I)}{2} + 3 * \kappa/100 * stdev(S_I, S_C)$$

$$and$$

$$T_2 = \frac{(S_C + S_I)}{2} + \frac{3}{2} * \kappa/100 * stdev(S_I, S_C)$$

$$and$$

$$T_3 = \frac{(S_C + S_I)}{2} + 3 * (100 - \kappa)/100 * stdev(S_I, S_C)$$

Now, if your test score S_T:

$S_T > T_1$ than your test score is classified as deleterious. Else if

$S_T < T_1$ and $S_T > T_2$ than your test score is partially deleterious. Else if

$S_T < T_2$ than your test score is classified as benign. At values of kappa < 66, T_3 is always larger than T_2, which divides partially deleterious mutations into 2 classes: partially deleterious or possibly benign OR partially deleterious only. The ROC plot script allows you to calculate the value of kappa by simulating through trio pairs of known test scores. The larger the simulation dataset, the closer kappa moves to a value to yield highest accuracy. The above equations are written assuming your known mutation is deleterious. IF your **known mutation would be partially deleterious** than you would need to only change the method you compute the imaginary benign as:

$$S_I = \frac{4 * S_K + 2.24 * S_c}{6.24}$$

IF your **known mutation would be benign** than you would simply assume:

$$S_I = S_K$$

This script is responsible for generating the ROC plot as seen in the MODICT paper figure 7. The ROC plot script accepts a list of tab-separated trios. These trios are composed of a negative control (wildtype in table S1), the score of a known sample (given in table S1) and the test score. Your negative control can be RMSD between wildtype–refined (second layer in SWISSPDB) and wildtype (reference). This means that you also have to generate your test and known sample scores by superimposing them on wildtype (again, this layer should be selected as reference in SWISSPDB). Alternatively you can have your negative control score as a benign model superimposed on wildtype. Than you would need another known mutation to be used as a known sample. I just want to clarify the rationale of generating trios here, you either need:

*** *1 wildtype, 1 wildtype refined, 1 test (refined preferred), 1 known model (refined preferred)*

THEN you superimpose wildtype-refined on wildtype and generate RMSD

Next you superimpose test on wildtype and genere RMSD and so on…

OR

**** 1 wildtype, 1 benign, 1 test, 1 known model.*

THEN you superimpose benign on wildtype and generate RMSD and

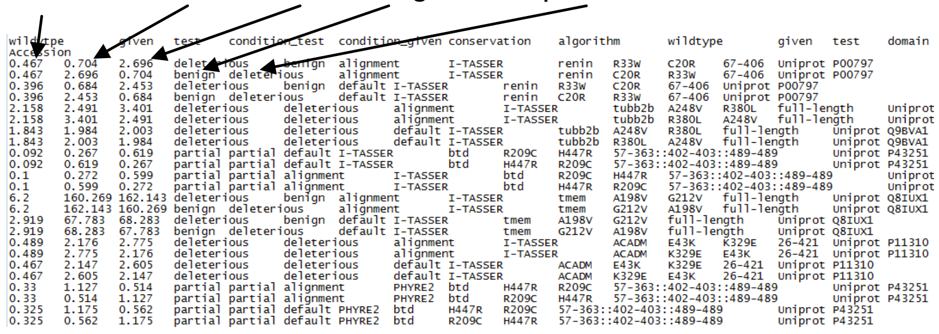
Next you superimpose known mutation on wildtype and generate another RMSD and so on…

I have explained the affects of choosing a benign model to generate a negative control instead of a refined wildtype in section 3.1. Up in the second proposal, you do not need refined models because you do not use a refined wildtype in the first place (the idea is that if an error is integrated into the negative control coming from the refinement, than a similar error should be incorporated into all other test and known samples to avoid misinterpretation).

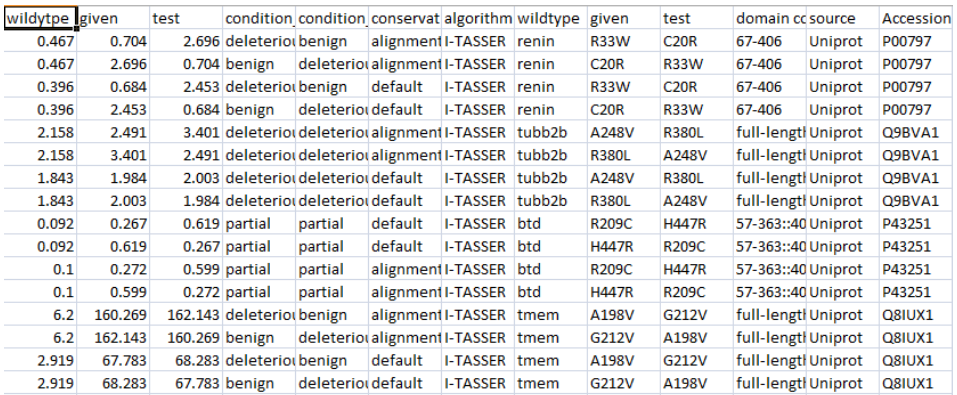
Model refinements are possible through refinement algorithms. You can do this yourself if you know how to do energy minimization or alternatively you can use the MOD REFINER (http://zhanglab.ccmb.med.umich.edu/ModRefiner/) or other similar programs.

Now, I have stated that you need tab separated trios per line in a text file as input. It should look like something like below:

value	value	value	deleterious	benign	X	Y	Z
value	vaue	value	benign	partial	D	G	T



```
wildtpe		given	test	condition_test	condition_given	conservation	algorithm	wildtype	given	test	domain
Accession
0.467	0.704	2.696	deleterious	benign	alignment	I-TASSER	renin	R33W	C20R	67-406	Uniprot	P00797
0.467	2.696	0.704	benign	deleterious	alignment	I-TASSER	renin	C20R	R33W	67-406	Uniprot	P00797
0.396	0.684	2.453	deleterious	benign	default	I-TASSER	renin	R33W	C20R	67-406	Uniprot	P00797
0.396	2.453	0.684	benign	deleterious	default	I-TASSER	renin	C20R	R33W	67-406	Uniprot	P00797
2.158	2.491	3.401	deleterious	deleterious	alignment	I-TASSER	tubb2b	A248V	R380L	full-length	Uniprot
2.158	3.401	2.491	deleterious	deleterious	alignment	I-TASSER	tubb2b	R380L	A248V	full-length	Uniprot
1.843	1.984	2.003	deleterious	deleterious	default	I-TASSER	tubb2b	A248V	R380L	full-length	Uniprot	Q9BVA1
1.843	2.003	1.984	deleterious	deleterious	default	I-TASSER	tubb2b	R380L	A248V	full-length	Uniprot	Q9BVA1
0.092	0.267	0.619	partial	partial	default	I-TASSER	btd	R209C	H447R	57-363::402-403::489-489	Uniprot	P43251
0.092	0.619	0.267	partial	partial	default	I-TASSER	btd	H447R	R209C	57-363::402-403::489-489	Uniprot	P43251
0.1	0.272	0.599	partial	partial	alignment	I-TASSER	btd	R209C	H447R	57-363::402-403::489-489	Uniprot
0.1	0.599	0.272	partial	partial	alignment	I-TASSER	btd	H447R	R209C	57-363::402-403::489-489	Uniprot
6.2	160.269	162.143	deleterious	benign	alignment	I-TASSER	tmem	A198V	G212V	full-length	Uniprot	Q8IUX1
6.2	162.143	160.269	benign	deleterious	alignment	I-TASSER	tmem	G212V	A198V	full-length	Uniprot	Q8IUX1
2.919	67.783	68.283	deleterious	benign	default	I-TASSER	tmem	A198V	G212V	full-length	Uniprot	Q8IUX1
2.919	68.283	67.783	benign	deleterious	default	I-TASSER	tmem	G212V	A198V	full-length	Uniprot	Q8IUX1
0.489	2.176	2.775	deleterious	deleterious	alignment	I-TASSER	ACADM	E43K	K329E	26-421	Uniprot	P11310
0.489	2.775	2.176	deleterious	deleterious	alignment	I-TASSER	ACADM	K329E	E43K	26-421	Uniprot	P11310
0.467	2.147	2.605	deleterious	deleterious	default	I-TASSER	ACADM	E43K	K329E	26-421	Uniprot	P11310
0.467	2.605	2.147	deleterious	deleterious	default	I-TASSER	ACADM	K329E	E43K	26-421	Uniprot	P11310
0.33	1.127	0.514	partial	partial	alignment	PHYRE2	btd	H447R	R209C	57-363::402-403::489-489	Uniprot	P43251
0.33	0.514	1.127	partial	partial	alignment	PHYRE2	btd	R209C	H447R	57-363::402-403::489-489	Uniprot	P43251
0.325	1.175	0.562	partial	partial	default	PHYRE2	btd	H447R	R209C	57-363::402-403::489-489	Uniprot	P43251
0.325	0.562	1.175	partial	partial	default	PHYRE2	btd	R209C	H447R	57-363::402-403::489-489	Uniprot	P43251
```



wildytpe	given	test	condition_	condition_	conservat	algorithm	wildtype	given	test	domain cc	source	Accession
0.467	0.704	2.696	deleteriou	benign	alignment	I-TASSER	renin	R33W	C20R	67-406	Uniprot	P00797
0.467	2.696	0.704	benign	deleteriou	alignment	I-TASSER	renin	C20R	R33W	67-406	Uniprot	P00797
0.396	0.684	2.453	deleteriou	benign	default	I-TASSER	renin	R33W	C20R	67-406	Uniprot	P00797
0.396	2.453	0.684	benign	deleteriou	default	I-TASSER	renin	C20R	R33W	67-406	Uniprot	P00797
2.158	2.491	3.401	deleteriou	deleteriou	alignment	I-TASSER	tubb2b	A248V	R380L	full-lengtl	Uniprot	Q9BVA1
2.158	3.401	2.491	deleteriou	deleteriou	alignment	I-TASSER	tubb2b	R380L	A248V	full-lengtl	Uniprot	Q9BVA1
1.843	1.984	2.003	deleteriou	deleteriou	default	I-TASSER	tubb2b	A248V	R380L	full-lengtl	Uniprot	Q9BVA1
1.843	2.003	1.984	deleteriou	deleteriou	default	I-TASSER	tubb2b	R380L	A248V	full-lengtl	Uniprot	Q9BVA1
0.092	0.267	0.619	partial	partial	default	I-TASSER	btd	R209C	H447R	57-363::40	Uniprot	P43251
0.092	0.619	0.267	partial	partial	default	I-TASSER	btd	H447R	R209C	57-363::40	Uniprot	P43251
0.1	0.272	0.599	partial	partial	alignment	I-TASSER	btd	R209C	H447R	57-363::40	Uniprot	P43251
0.1	0.599	0.272	partial	partial	alignment	I-TASSER	btd	H447R	R209C	57-363::40	Uniprot	P43251
6.2	160.269	162.143	deleteriou	benign	alignment	I-TASSER	tmem	A198V	G212V	full-lengtl	Uniprot	Q8IUX1
6.2	162.143	160.269	benign	deleteriou	alignment	I-TASSER	tmem	G212V	A198V	full-lengtl	Uniprot	Q8IUX1
2.919	67.783	68.283	deleteriou	benign	default	I-TASSER	tmem	A198V	G212V	full-lengtl	Uniprot	Q8IUX1
2.919	68.283	67.783	benign	deleteriou	default	I-TASSER	tmem	G212V	A198V	full-lengtl	Uniprot	Q8IUX1

Above screenshots of the same input file

The first value is the negative score, the “wildtype” value. The second value is your **KNOWN sample**. This sample can be either a known deleterious, benign or partially deleterious mutation. The last value is your test score. This test mutation can also be partially deleterious, benign or deleterious. This outcome is already known for both for the test mutation and the known sample. They are indicated in column 5 and 4 respectively. You write the outcome of your test mutation in column 4 and the known sample to 5 as shown in the MODICT paper table S1. The entries other than **the first 5** are not important (3 value and 2 benign/partial/deleterious as shown above), it is for you. They are not taken by the script. In fact table S1 as a whole is an input file for the roc script “roc_v1.0.pl” and it is located in “./MODICT/ROC/”.

You might have a header in your input file, which will be asked to be skipped once you run the script. The most important part of the script is its rationale which is stated between lines 71-157:

For your positive control (known sample) there are 3 possibilities: deleterious, partially deleterious and benign.

For your test sample there are also 3 possibilities like above. In total there can be 9 different pairs. (deleterious-benign, partially deleterious-deleterious…etc).

You will see a sample input file “myinput.txt” and the script “roc_v1.0.pl” in “./ROC”. Run the script from command line:

```
perl ROC_v1.0.pl
```

You will be asked to give the name of your input file, enter “myinput.txt”. You will be asked whether to skip header. The sample input has 1 headerline, say ‘yes’ and then enter ‘1’. You will have your results with the name “./ROC/output_ROC.txt”. You can also run the script like this:

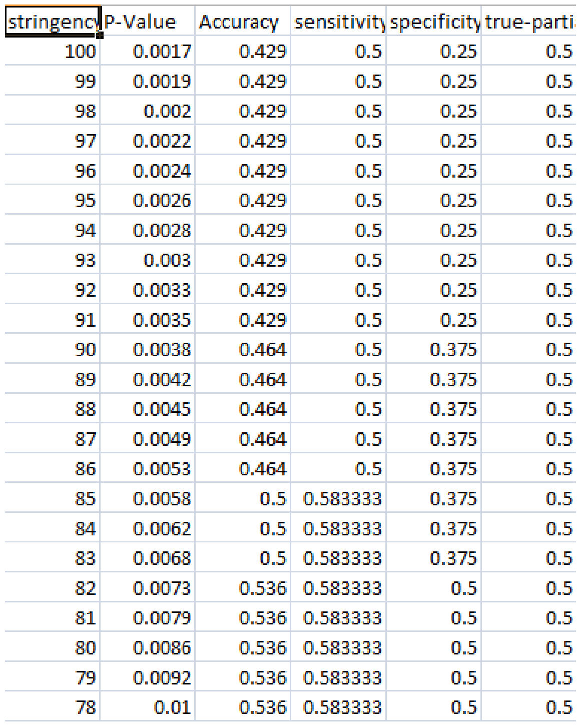
```
perl roc_v1.0.pl –C:/../../MODICT
```

After the last “..” you don’t enter any forward-slash. The last “..” is the parent folder of the ROC. The extra parameter –population changes the way the standard deviation is calculated. If the population parameter is used, than 1 is not subtracted from the denominator of sum of squares. You can run it like this:

```
perl roc_v1.0.pl –C:/../../MODICT –population
```

OR

```
perl roc_v1.0.pl –population
```



stringenc	P-Value	Accuracy	sensitivity	specificity	true-parti
100	0.0017	0.429	0.5	0.25	0.5
99	0.0019	0.429	0.5	0.25	0.5
98	0.002	0.429	0.5	0.25	0.5
97	0.0022	0.429	0.5	0.25	0.5
96	0.0024	0.429	0.5	0.25	0.5
95	0.0026	0.429	0.5	0.25	0.5
94	0.0028	0.429	0.5	0.25	0.5
93	0.003	0.429	0.5	0.25	0.5
92	0.0033	0.429	0.5	0.25	0.5
91	0.0035	0.429	0.5	0.25	0.5
90	0.0038	0.464	0.5	0.375	0.5
89	0.0042	0.464	0.5	0.375	0.5
88	0.0045	0.464	0.5	0.375	0.5
87	0.0049	0.464	0.5	0.375	0.5
86	0.0053	0.464	0.5	0.375	0.5
85	0.0058	0.5	0.583333	0.375	0.5
84	0.0062	0.5	0.583333	0.375	0.5
83	0.0068	0.5	0.583333	0.375	0.5
82	0.0073	0.536	0.583333	0.5	0.5
81	0.0079	0.536	0.583333	0.5	0.5
80	0.0086	0.536	0.583333	0.5	0.5
79	0.0092	0.536	0.583333	0.5	0.5
78	0.01	0.536	0.583333	0.5	0.5

Once you have your output, examine it carefully. You have 6 columns as shown above. The first column stores the stringency parameter. It is a good thing of the stringency parameter to be high. However as the stringency parameter goes high the accuracy of the program drops, but the remaining accuracy can be explained more by the results of the algorithm. Inversely, if stringency goes too low, the accuracy will be very high but the rise in accuracy can progressively be less explained by the measurements of the algorithm. **YOUR AIM** is to find from the list the highest accuracy **WHILE** still keeping stringency as high as possible **AND** staying below a good threshold p-value (I prefer p-value to be **NO GREATER** than 0.05). In columns 3, 4, 5 and 6 you will find accuracy, sensitivity, specificity and true-partial rates respectively. True partial rates are just like sensitivity and specificity, they are the ratio of true-partials and true-partials plus false-partials.

To understand how the stringency parameter works I advise you to take a look at lines 56-171 in “ROC_v1.0.pl”.

A typical visualization of ROC plot is displayed in MODICT paper figure 7. Once you found the highest accuracy point (without losing too much stringency and gaining p-value), the associated stringency value is your κ. For instance the current output file of MODICT indicates a stringency value of 55, which is your κ. Based on this value, calculate your thresholds and proceed as described earlier in this section.

I will soon release a module to automatically fit a curve for the given input data and calculate optimal κ value and pre-compute thresholds based on your data.

8. Tools

“batch.pl”

In this section I will demonstrate two tools that are in the “./Core” folder. One of them is the “batch.pl”. You provide this script a list of file names in a txt file such as:

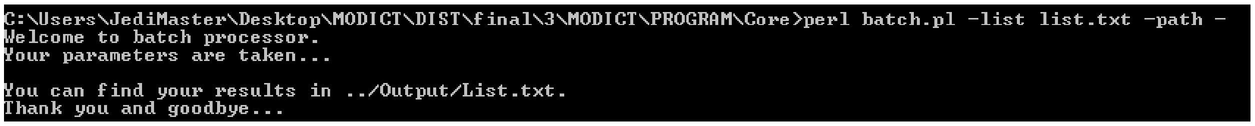
File1.txt

File2.txt

…

This “list.txt” file has to be located in “./Input” folder. The text files listed in the “list.txt” also has to be located in the “./Input” folder. When you run the script it will generate an output in the “./Output” folder with the name of each text file and its corresponding MODICT score. However to be able run this script your “essentials_MODICT.txt” needs to be preconfigured. This means you need to run MODICT once before with one of the files in the “list.txt”. You run “batch.pl” like this:

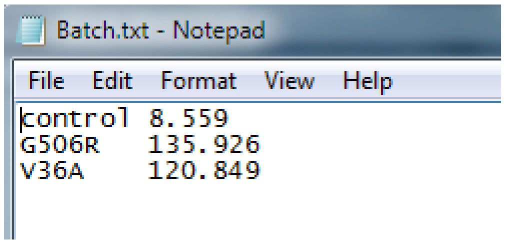
perl batch.pl –list list.txt



```
C:\Users\JediMaster\Desktop\MODICT\DIST\final\3\MODICT\PROGRAM\Core>perl batch.pl -list list.txt -path -
Welcome to batch processor.
Your parameters are taken...

You can find your results in ../Output/List.txt.
Thank you and goodbye...
```

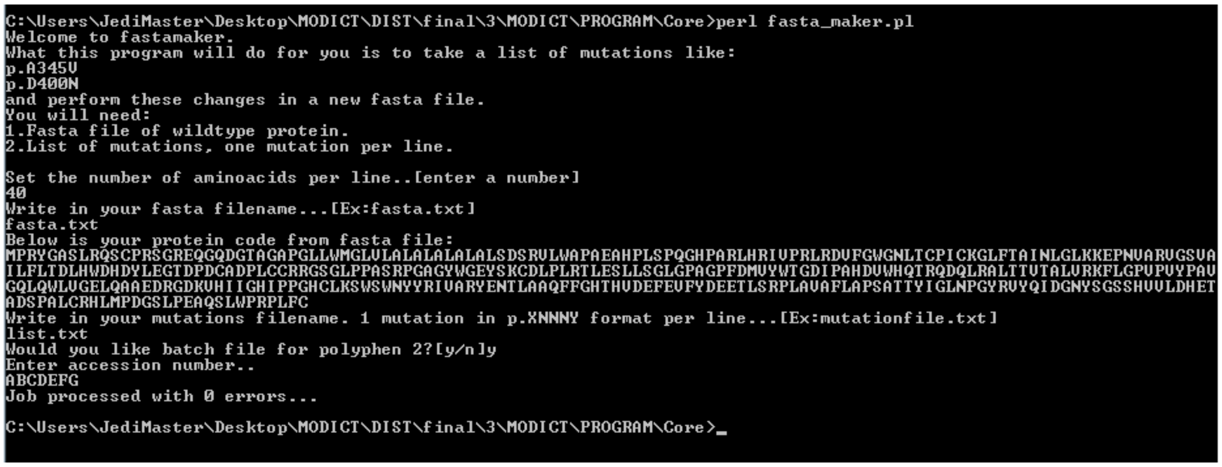
Your output should look like below:



```
control 8.559
G506R   135.926
V36A    120.849
```

“fasta_maker.pl”

This script is to generate multiple copies of a fasta file with selected amino acids mutated to another one. You will need 2 things, a fasta file, and a list file that has mutations listed in the “p.XNNNY” (amino acid X mutated to Y at position NNN) format. The list file is as always a text file, and there can be only 1 mutation per line. Both “list.txt” and the fasta files have to located in “./Input” folder. Below is a screenshot:

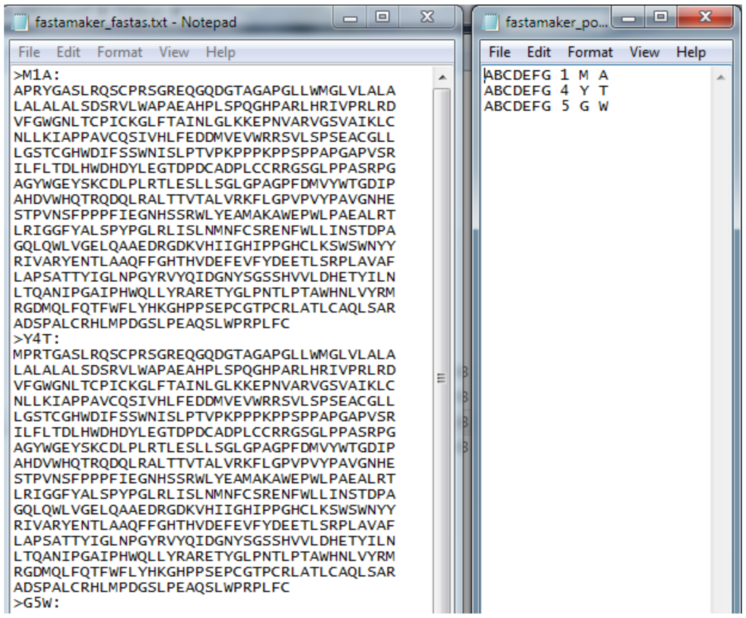


```
C:\Users\JediMaster\Desktop\MODICT\DIST\final\3\MODICT\PROGRAM\Core>perl fasta_maker.pl
Welcome to fastamaker.
What this program will do for you is to take a list of mutations like:
p.A345V
p.D400N
and perform these changes in a new fasta file.
You will need:
1.Fasta file of wildtype protein.
2.List of mutations, one mutation per line.

Set the number of aminoacids per line..[enter a number]
40
Write in your fasta filename...[Ex:fasta.txt]
fasta.txt
Below is your protein code from fasta file:
MPRYGASLRQSCPRSGREQGQDGTAGAPGLLWMGLVLALALALALALSDSRVLWAPAEAHPLSPQGHPARLHRIVPRLRDVFGWGNLTCPICKGLFTAINLGLKKEPNVARVGSVA
ILFLTDLHWDHDYLEGTDPDCADPLCCRRGSGLPPASRPGAGYWGEYSKCDLPLRTLESLLSGLGPAGPFDMVYWTGDIPAHDVWHQTRQDQLRALTTVTALVRKFLGPVPVYPAV
GQLQWLVGELQAAEDRGDKVHIIGHIPPGHCLKSWSWNYYRIVARYENTLAAQFFGHTHVDEFEVFYDEETLSRPLAVAFLAPSATTYIGLNPGYRVYQIDGNYSGSSHVVLDHET
ADSPALCRHLMPDGSLPEAQSLWPRPLFC
Write in your mutations filename. 1 mutation in p.XNNNY format per line...[Ex:mutationfile.txt]
list.txt
Would you like batch file for polyphen 2?[y/n]y
Enter accession number..
ABCDEFG
Job processed with 0 errors...

C:\Users\JediMaster\Desktop\MODICT\DIST\final\3\MODICT\PROGRAM\Core>_
```

With this script you are also able to control how many amino acids you want to be printed per line. It also generates a batch file for polyphen2 with the accession code you provide. The outputs should look like below:


fastamaker_fastas.txt - Notepad
File Edit Format View Help
fastamaker_po...
File Edit Format View Help

```
>M1A:
APRYGASLRQSCPRSGREQGQDGTAGAPGLLWMGLVLALA
LALALALSDSRVLWAPAEAHPLSPQGHPARLHRIVPRLRD
VFGWGNLTCPICKGLFTAINLGLKKEPNVARVGSVAIKLC
NLLKIAPPAVCQSIVHLFEDDMVEVWRRSVLSPSEACGLL
LGSTCGHWDIFSSWNISLPTVPKPPPKPPSPPAPGAPVSR
ILFLTDLHWDHDYLEGTDPDCADPLCCRRGSGLPPASRPG
AGYWGEYSKCDLPLRTLESLLSGLGPAGPFDMVYWTGDIP
AHDVWHQTRQDQLRALTTVTALVRKFLGPVPVYPAVGNHE
STPVNSFPPPFIEGNHSSRWLYEAMAKAWEPWLPAEALRT
LRIGGFYALSPYPGLRLISLNMNFCSRENFWLLINSTDPA
GQLQWLVGELQAAEDRGDKVHIIGHIPPGHCLKSWSWNYY
RIVARYENTLAAQFFGHTHVDEFEVFYDEETLSRPLAVAF
LAPSATTYIGLNPGYRVYQIDGNYSGSSHVVLDHETYILN
LTQANIPGAIPHWQLLYRARETYGLPNTLPTAWHNLVYRM
RGDMQLFQTFWFLYHKGHPPSEPCGTPCRLATLCAQLSAR
ADSPALCRHLMPDGSLPEAQSLWPRPLFC
>Y4T:
MPRTGASLRQSCPRSGREQGQDGTAGAPGLLWMGLVLALA
LALALALSDSRVLWAPAEAHPLSPQGHPARLHRIVPRLRD
VFGWGNLTCPICKGLFTAINLGLKKEPNVARVGSVAIKLC
NLLKIAPPAVCQSIVHLFEDDMVEVWRRSVLSPSEACGLL
LGSTCGHWDIFSSWNISLPTVPKPPPKPPSPPAPGAPVSR
ILFLTDLHWDHDYLEGTDPDCADPLCCRRGSGLPPASRPG
AGYWGEYSKCDLPLRTLESLLSGLGPAGPFDMVYWTGDIP
AHDVWHQTRQDQLRALTTVTALVRKFLGPVPVYPAVGNHE
STPVNSFPPPFIEGNHSSRWLYEAMAKAWEPWLPAEALRT
LRIGGFYALSPYPGLRLISLNMNFCSRENFWLLINSTDPA
GQLQWLVGELQAAEDRGDKVHIIGHIPPGHCLKSWSWNYY
RIVARYENTLAAQFFGHTHVDEFEVFYDEETLSRPLAVAF
LAPSATTYIGLNPGYRVYQIDGNYSGSSHVVLDHETYILN
LTQANIPGAIPHWQLLYRARETYGLPNTLPTAWHNLVYRM
RGDMQLFQTFWFLYHKGHPPSEPCGTPCRLATLCAQLSAR
ADSPALCRHLMPDGSLPEAQSLWPRPLFC
>G5W:
```

```
ABCDEFG 1 M A
ABCDEFG 4 Y T
ABCDEFG 5 G W
```

****There are no options in "fasta_maker.pl". It follows general Q&A mode****

9. Contact

I will update this manual and the scripts in time as I receive feed back and have new ideas to incorporate. If there is anything unclear about this document or any of the scripts mentioned please contact:

itanyalc@vub.ac.be