ProSa 2003

PROtein Structure Analysis

Version 4.0

CAME

Center of Applied Molecular Engineering

Distributed by

ProCeryon Biosciences GmbH
Jakob–Haringer–Str. 3
A-5020 Salzburg
Austria

August 2003

Contents

LIMITATION OF LIABILITY

IN NO EVENT SHALL CAME BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGE, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL OR OTHER DAMAGES. CAME SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE, RELATED TO DEFECTS IN THE DISTRIBUTION MEDIA.

Notational Conventions

The meanings of the various fonts and symbols used in this manual are described below.

plot	***ProSa 2003*** commands and their arguments in plain text are in *italics*.
/usr/people/myfile	File names in text are in *italics*.
`/usr/local/`	Literal user inputs and examples are in **`bold typewriter`** fonts.
`enter`	Command output in examples is in `typewriter` font.
{*name*}	Strings which have to be supplied by the user are in *italics* surrounded by parenthesis.
{*filename*}	Strings surrounded by braces are mandatory.
[*filename*]	Strings surrounded by brackets are optional.

1 New features

This section gives a brief overview of new features of ***ProSa 2003*** with respect to the former release (***Prosa II***).

1.1 Analysis of sequence mutations

ProSa 2003 includes several new commands and variables that support *in silico* mutagenesis. A detailed description of these commands and the associated variables can be found on page 45. Their usage is demonstrated by the new sample sessions 7 and 8 and in Wiederstein et al. (2003).

1.2 Potentials

A set of new potentials is included with ***ProSa 2003***. They are derived from a set of 3541 protein domains based on the SCOP release 1.61 from September 2002. The generation of these potentials was motivated by the fact that the number of known protein structures has significantly increased since the ***Prosa II*** default potentials have been compiled (1993). It is expected that an update of the knowledge base also yields improved potentials. A study comparing default and updated potentials is in preparation. The table below shows the mean z-scores over the set of 3541 domains for both old and new potentials. In addition, the effects of choosing a different reference state and the usage of virtual C^{β}-atoms for glycine residues is shown (refer to page 43).

Potential	Parameter	$\overline{z}_{comb}$	$\overline{z}_{pair}$	$\overline{z}_{surf}$

[1]`pII3.0.*`	`potential type 1,`
	`skip_gly_cb = 1`
		-8.48	-6.01	-5.72

`prosa2003.*`	`potential type 1,`
	`skip_gly_cb = 1`
		-9.01	-6.87	-6.39

`prosa2003.*`	`potential type 2,`
	`skip_gly_cb = 1`
		-9.48	-7.30	-6.39

`prosa2003.*`	`potential type 2,`
	`skip_gly_cb = 0`
		-9.91	-7.96	-6.39

[1] ***Prosa II*** default setting

1.3 Default settings

All default settings of the current version match the settings of the previous version ***Prosa II***. Exception: the default value of *sorted_depth* has been changed from 10 to 0. This means that no *.sor-file is generated when z-scores are calculated (see page 43).

1.4 New PDB parser

PDB files are now read with a new parser which allows to specify a chain identifier and/or NMR model number. Furthermore, the mapping of PDB residue numbers to corresponding ***ProSa 2003*** residue numbers can now be printed (see page 36).

1.5 Command history

It is now possible to browse through a command history by pressing the CursorUp- and CursorDown-keys in the command line.

1.6 Software design

ProSa 2003 uses several external libraries (Python, Tkinter, Tcl/Tk and BLT). This facilitates the control of ***ProSa 2003*** *via* the scripting language Python (see page 48).

2 Introduction

ProSa 2003 is a powerful tool in protein structure research. ***ProSa 2003*** supports and guides your studies aimed at the determination of a protein's native fold. It is helpful for experimental structure determinations and for modeling studies. Below you will find several examples demonstrating ***Prosa 2003's*** capabilities. But let's set the stage first.

Most biological proteins have characteristic native folds. In physiological conditions the native structure forms spontaneously. The protein's folded structure is a function of its amino acid sequence and its natural environment. In the case of soluble globular proteins the *in vivo* environment is generally an aqueous solution of various ingredients.

The amino acid sequence defines the molecular identity of a protein. The study of the biological role, molecular mechanism, catalysis, molecular interactions, binding of effector molecules, and many more important features of individual proteins require a knowledge of their three dimensional structures.

In general the calculation of native folds of proteins from amino acid sequences is still impossible, although in several cases modeling by homology has turned out to be quite successful. But in general, if the native structure of a protein is needed there is still no escape from X-ray analysis and/or NMR-spectroscopy. Unfortunately, these techniques are time consuming and fail in many cases due to experimental problems (e.g. lack of isomorphous derivatives, size of the protein etc.), and the only remaining possibility is an attempt to build a model.

Now suppose you are confronted with the three-dimensional structure of a protein. The structure may have been determined by X-ray analysis or NMR-spectroscopy, it may be the result of modeling studies or it could be a nice looking structure deliberately misfolded to fool specialists in protein structure theory. Obvious questions are:

Is the structure correct?

Are there faulty parts?

Of course the confidence in structures obtained from modeling studies is generally poor as compared to experimentally determined folds. However, several experimentally determined structures have turned out to be incorrect or have been shown to contain faulty parts.

ProSa 2003 can assist you in protein structure quality control. It calculates a score for your input-structure. Scores of native protein folds are in a characteristic range. If your score is outside this range then your structure may have problems.

ProSa 2003's energy graphs provide diagnostic tools to spot problematic sections. Energy graphs display the energetic architecture of protein folds as a function of amino acid sequence position. High energies correspond to stressed or strained sections of the chain and may point to problematic parts of a fold. As long as scores and energy graphs are non native like chances are that the structure can be improved.

These are ***ProSa 2003***'s basic tools. They provide a strong frame work for protein structure determination and modeling. But there are several additional features you will discover browsing the manual.

The heart of ***ProSa 2003*** is made of energy functions and potentials. These potentials are continuously refined and improved. New ProSa versions will be equipped with the most advanced set of potentials available.

ProSa 2003 is applicable to a particular range of problems. Consult the manual and references on the interpretation of results, on the range of applicability and further details

ProSa 2003's staff is not responsible for your applications. There is no warranty for the consequences of your actions. Proper usage is at your hands and you are responsible for your results and their proper interpretation.

Therefore:

Be suspicious of experimental data.

But also:

Never trust theoretical methods.

3 Z-score plots of native proteins

Z-scores indicate the quality of protein structures. The score of native folds are in a characteristic range and they depend on the sequence length. The file *pII3.0.zscores.txt* contains the scores for all proteins which were used to compile the potentials. However, the scores were obtained by a 'jack-knife-test', i.e. the respective protein was removed from the database and the potentials were recompiled before the score was calculated. The scores obtained from ***ProSa 2003*** will differ (i.e. appear to be more significant) since the protein is now contained in the potentials.

Figures 1, 2 and 3 show the score distribution as a function of sequence length from C^β - C^β pair interactions, C^β surface terms, and combined (pair+surface) energy terms. The regression lines correspond to the scores expected for the native fold of sequence length $n : z_e = an + b$.

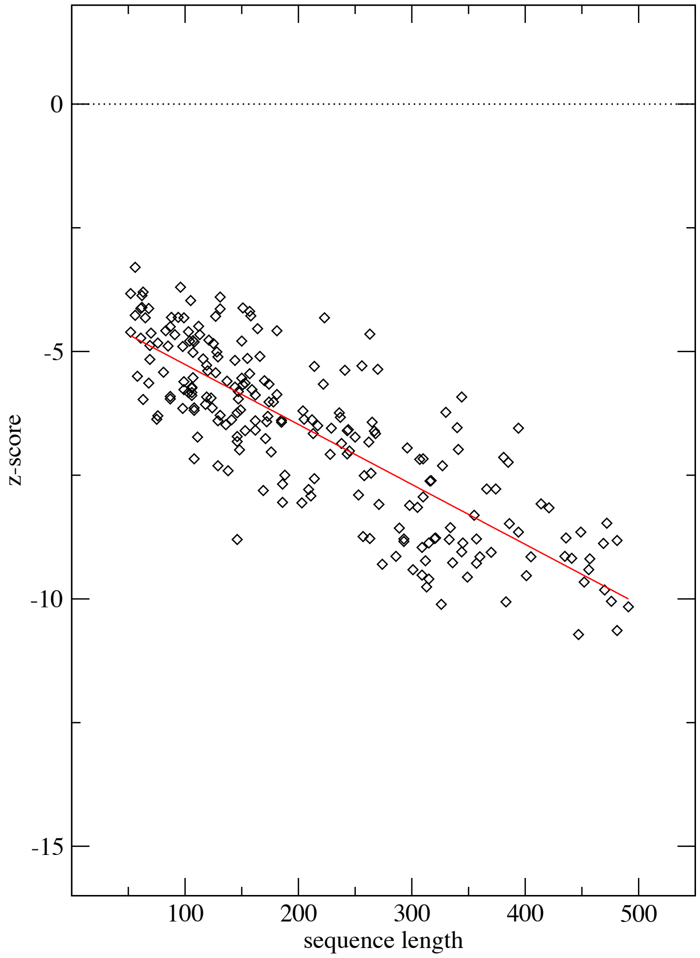


Figure 1: Z-scores of pair energy(C^{β} interactions). Regression equation:$y = -4.05 - 0.0121x$

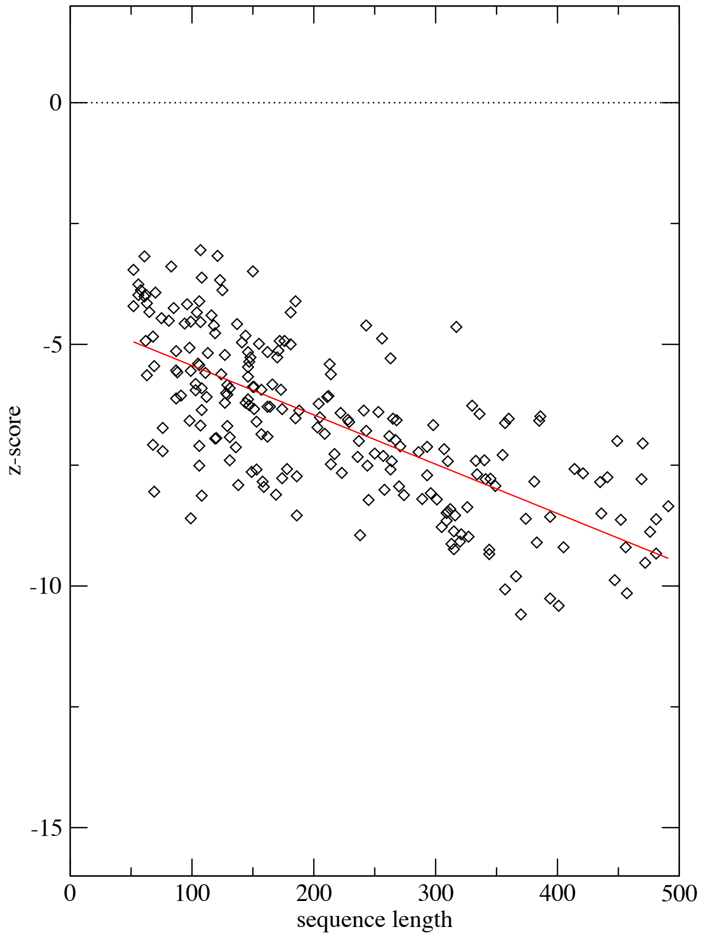


Figure 2: Z-scores of surface energy(C^{β}). Regression equation: $y = -4.43 - 0.0101x$

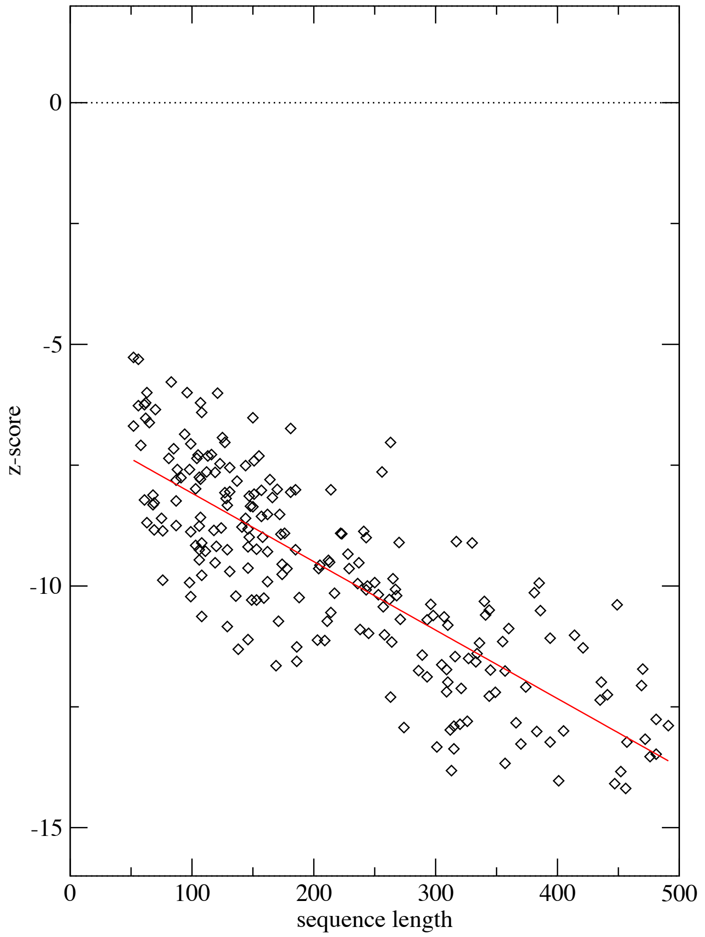


Figure 3: Z-scores of combined energy(C^{β}). Regression equation: $y = -6.67 - 0.0141x$

4 Using ProSa 2003

4.1 Launching of ProSa2003

On Unix/Linux shells or on the Windows DOS prompt type

prosa2003 [*filename*]

If called without argument, ***ProSa 2003*** starts up with a simple graphical user interface. If *filename* is specified, ***ProSa 2003*** reads the ASCII file *filename* and executes the commands contained in this file.

On Windows systems either double click the ***ProSa 2003*** icon or start ***ProSa 2003*** from the programs menu.

If the final command in a command file is *exit*, ***ProSa 2003*** terminates, otherwise the graphical user interface pops up and accepts further input.

4.2 The prosa-startup file

ProSa 2003 needs some initializing commands and variables which are read from the startup file *prosa-startup*. This file is essentially a command file which is automatically loaded each time ***ProSa 2003*** is started. ***ProSa 2003*** searches for this file in your home directory. If it is not found there ***ProSa 2003*** uses the default startup file in the *ProSaData* directory.

Startup performs three main tasks.

- Set Default Parameters

  The program needs several parameters which are important for proper operation and save applications. Such parameters are relative weights for energy terms, distance cutoff parameters and the like.

- Tell ***ProSa 2003*** where to find what

  The commands *bbn_dir* and *pdb_dir* define default pathnames to backbone style files and Brookhaven style files, respectively. These names can be changed during a session.

- Load potentials

  The commands *pair potential* and *surface potential* load the types of potentials specified at the command line.

You can create your own startup file in your home directory. On Unix/Linux copy the default file by

```
% cp ''ProSaData''/prosa-startup $HOME/.prosa-startup
```

On Windows you may use the windows explorer or:

```
C:\ copy ''ProSaData''/prosa-startup
   %HOMEDRIVE%%HOMEPATH%.prosa-startup
```

and edit the `.prosa-startup` file as required.

4.3 Objects

ProSa 2003's data items are called objects. Presently an object consists of protein backbone coordinates, amino acid sequence, associated energy graphs, specification of potentials and a set of plot parameters. A new object is created when a protein is loaded.

The program can handle an arbitrary number of objects at the same time. A particular object is accessed by its name. The name is created when a protein is read or an object is copied.

An [*object*] argument in commands refers to an object name. To set parameters for a particular object specify its name in the [*object*] argument. If you want to access all objects currently loaded, use '*' as the object argument. (Note that the '*' is not a wild card character in the UNIX sense. E.g. 'e*' is invalid.)

Objects can be used to display distinct graphs for a single protein at the same time. Simply copy the object and change the parameters of the new object and plot both (or more) objects.

4.4 Command files

Command files are ASCII text files containing ***ProSa 2003*** commands. A typical example is the prosa-startup file, which contains commands and variables to customize your session. Any sequence of commands can be stored in an ASCII file and executed as a batch job. (See sample sessions below.) A command file can be loaded with the command execute {*filename*} or when you invoke ***ProSa 2003***.

4.5 Sample Sessions

In the following examples we demonstrate the generation of energy graphs and calculation of z-scores.

The used PDB files used in the sample sessions are available for download at **`ftp://ftp.came.sbg.ac.at/pub/Prosa/PDBfiles/`**

4.5.1 Session 1

This session demonstrates how to run ***ProSa 2003*** in interactive mode. We load two proteins, set colors, window sizes and energy types.

1. Start ***ProSa 2003***

`prosa2003`	Start ***ProSa 2003*** without an option. A working screen appears. Activate the command line with a mouse click.

2. Load and analyse a protein

`read pdb pdb2aat.ent obj1`	Load the protein 2aat. The object is named 'obj1'.
`analyse energy obj1`	Calculate energies for obj1.
`plot`	Displays the energy graph of obj1. You should see the combined energy.

3. Change window size

`winsize obj1 50`	Change winsize to 50 residues.
`plot`	

4. Edit the graph

`draw * obj1 1`	Enables display of pair, surface and combined energies for obj1.
`plot`	The new plot appears, this time with pair, surface and combined energy.
`color comb obj1 cyan`	
`color surf obj1 magenta`	Set different colors for the various energy terms.
`plot`	

5. Load and analyse a second protein

`read pdb pdb1spa.ent obj2`	1spa is loaded and named 'obj2'.
`analyse energy obj2`	
`plot`	The energy graphs of 2aat (obj1) are plotted with the set of parameters as defined above. 1spa(obj2) is plotted with the default set of parameters.
`color * obj2 red`	
`winsize * 50`	Set window size for all proteins.
`plot`	

6. Edit the graph

`draw * * 0`	Remove all energy graphs. (This will not become visible until next 'plot' command)
`draw pair * 1`	Draw pair energies of all loaded proteins.
`plot`	Display pair energy graphs.

7. Export a postscript file

`export plot myplot`	A postscript file called myplot.ps is created equivalent to the last plot on your screen. (Figure 4)

8. Exit *ProSa 2003*

`quit`	Exit the program.

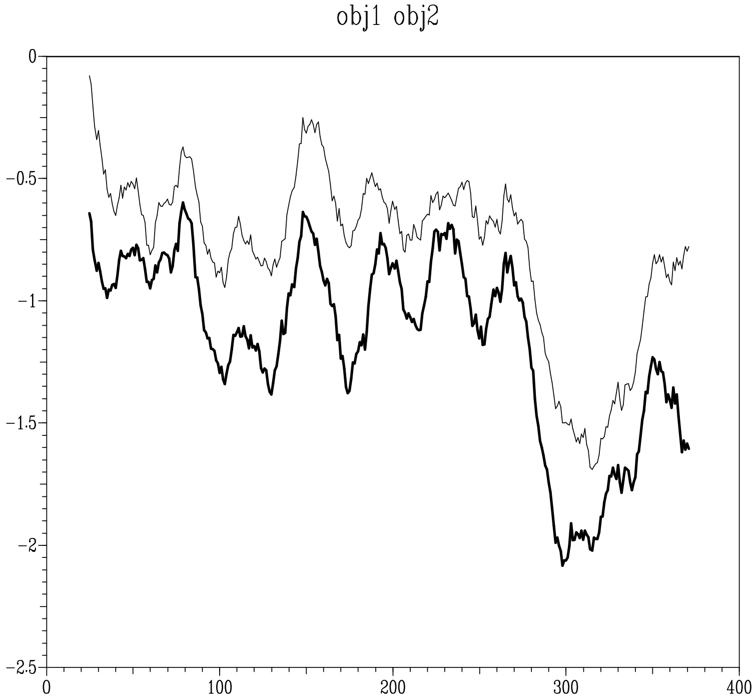


Figure 4: The file myplot.ps resulting from sample session 1. Thin line: Pair energy graph of 2aat. Bold line: Pair energy graph of 1spa. Both graphs are smoothed using a window size of 50 residues.

4.5.2 Session 2

Now we demonstrate the use of command files.

The pair energy graphs of 2aat, 3aat, 1aaw and 1spa are generated. These proteins are identical, but their resolution is different. 2aat has a resolution of 2.8 Å, 3aat has 2.5 Å, 1aaw 2.4 Åand 1spa 2.0 Å.

The commands are summarized in file *session2.cmd*. You can invoke the command file by two different procedures.

1. Either

`prosa2003 session2.cmd`	Execute commands when starting ***ProSa 2003***. A plot as specified in *session2.cmd* appears (Figure 5). You can now continue to use ***ProSa 2003***.

2. or

`prosa2003`	Start ***ProSa 2003***
`execute session2.cmd`	Execute commands. The result will be the same as in (1).

4.5.3 Session 3

The Brookhaven Data Base contains several entries for lactate dehydrogenase. We analyse two of these structures, 3ldh and 6ldh.

Use the command file *session3.cmd* to generate figure 6. The graph of 3ldh is alarming. The structure is problematic due to an incorrect amino acid sequence (Consult the REMARK records of 3ldh).

4.5.4 Session 4

Energy graph and z-score calculation for 2gn5 and 1vqb:

1. Use *session4a.cmd* for energy graphs. (Figure 7)

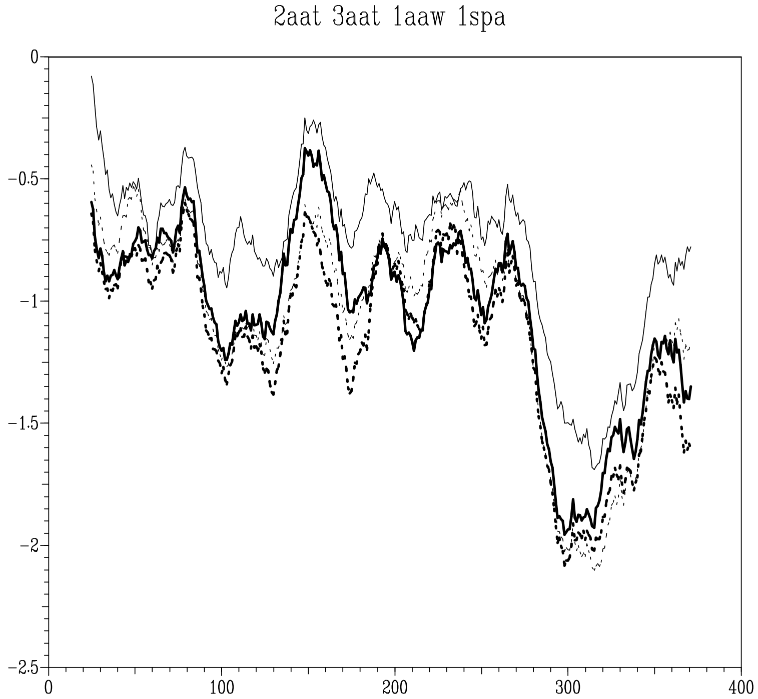


Figure 5: Pair energy graphs of 2aat, 3aat, 1aaw and 1spa, four structure determinations of aspartate amino transferase of different resolution. Thin line: 2aat; thin dotted line: 3aat; bold line: 1aaw; bold dotted line: 1spa.

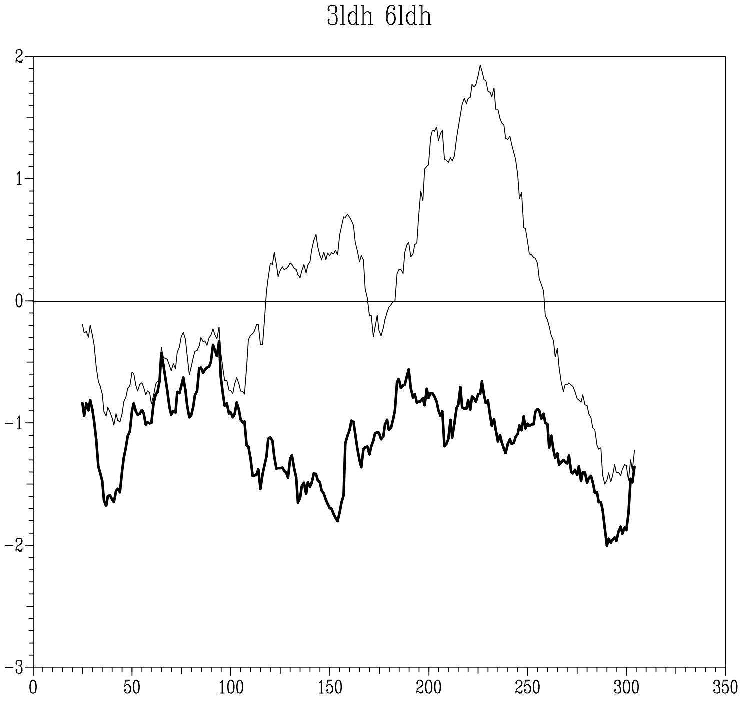


Figure 6: The result of the command file of sample session 3. Combined energies are shown. Window size is 50. Thin line: 3ldh; bold line: 6ldh.

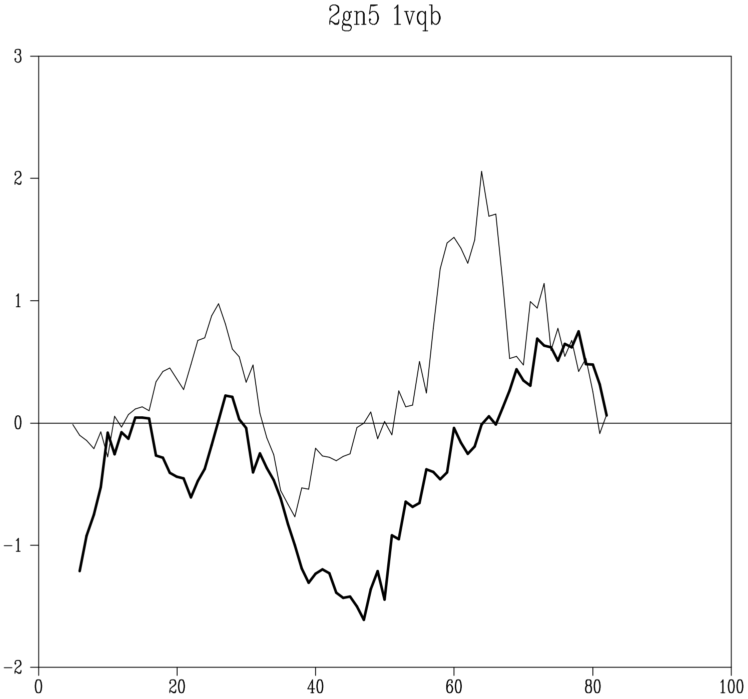


Figure 7: Energy analysis from session 4 (Pair energies). Thin line: 2gn5; bold line: 1vqb.

2. Use *session4b.cmd* for z-score calculation. (Figures 8 and 9)

4.5.5 Session 5

Again we analyse 2aat and 1spa, Aminotransferase at different resolution. This time we generate a graph showing the energy difference of the two.

This example demonstrates the use of the commands *use potential* and *diff*. Both C^α and C^β potentials will be used for the calculation. Thus, the energy graphs resulting from this session will differ from those of previous sessions.

1. Load C^α and C^β potentials.

`pair potential $PROSA_BASE/pII3.0.pair-cb pcb` Load pair C^β potential and call it 'pcb'.

```
Hide & Seek for 2gn5 (combined energies) on pII3.0.short.ply
rank molecule      offset overlay z-score    energy     rms  seq-id
   1 1aai-b           148      87   -4.02 -2.00e+00   14.30    0.07
   2 1gpr              32      87   -3.99 -1.62e+00   14.27    0.08
   3 1hyp             -22      65   -3.93 -9.58e-01   15.48    0.09
   4 1pgd              77      87   -3.93 -9.26e-01   14.83    0.07
   5 1ofv              57      87   -3.88 -3.07e-01   12.72    0.07
   6 1lld-b           133      87   -3.86 -1.03e-01   13.48    0.08
   7 1mfa               4      87   -3.80  6.29e-01   15.07    0.08
   8 1fba-b            89      87   -3.70  1.83e+00   15.79    0.07
   9 1paz              42      78   -3.68  2.12e+00   15.34    0.05
  10 1ppn             130      82   -3.64  2.58e+00   15.83    0.10

Hide & Seek for 2gn5 (pair energies) on pII3.0.short.ply
rank molecule      offset overlay z-score    energy     rms  seq-id
   1 1npx              55      87   -3.60 -1.69e+01   16.50    0.02
   2 1gky             -12      75   -3.43 -1.54e+01   14.52    0.07
   3 1btc              37      87   -3.38 -1.49e+01   14.72    0.10
   4 1atr             -19      68   -3.38 -1.49e+01   17.97    0.05
   5 1fdd              54      52   -3.32 -1.43e+01   17.41    0.15
   6 1lfi              34      87   -3.28 -1.40e+01   17.50    0.09
   7 1aak              53      87   -3.27 -1.39e+01   15.51    0.03
   8 1bet             -36      51   -3.25 -1.37e+01   28.70    0.06
   9 1fba-b            59      87   -3.14 -1.27e+01   15.41    0.09
  10 1pgd             -57      30   -3.12 -1.25e+01   19.55    0.07

Hide & Seek for 2gn5 (surface energies) on pII3.0.short.ply
rank molecule      offset overlay z-score    energy     rms  seq-id
   1 1gca             144      87   -4.08 -3.54e+00   12.68    0.11
   2 1bnh              66      87   -3.83 -2.26e+00   15.49    0.13
   3 1aai-b           148      87   -3.75 -1.84e+00   14.30    0.07
   4 1bnh             351      87   -3.69 -1.55e+00   15.25    0.09
   5 1bnh             237      87   -3.66 -1.40e+00   15.29    0.09
   6 1bnh             123      87   -3.65 -1.37e+00   15.39    0.09
   7 1ala             238      78   -3.58 -9.85e-01   12.26    0.03
   8 1bnh               9      87   -3.57 -9.43e-01   16.13    0.09
   9 1bnh             294      87   -3.57 -9.20e-01   15.20    0.05
  10 1pii              52      87   -3.52 -6.80e-01   12.59    0.07
```

Figure 8: The energy sorted list in the file z-result.sor as result of sample session 4. For description see section 5.9.

```
Hide & Seek on polyprotein pII3.0.short.ply - selection of parameters
molecule     seq-l   zp-comb  zp-pair  zp-surf  rk-comb  rk-pair  rk-surf  z1-comb  z1-pair  z1-surf
2gn5            87     -2.18    -0.14    -2.76      589    14023       96    -4.02    -3.60    -4.08

               ep-comb  ep-pair  ep-surf  em-comb  em-pair  em-surf  es-comb  es-pair  es-surf
                 20.13    14.43     3.19    46.36    15.72    17.16    12.04     9.05     5.07


Hide & Seek on polyprotein pII3.0.short.ply - selection of parameters
molecule     seq-l   zp-comb  zp-pair  zp-surf  rk-comb  rk-pair  rk-surf  z1-comb  z1-pair  z1-surf
1vqb            86     -4.65    -3.66    -2.54        1        1      216    -4.08    -3.53    -4.15

               ep-comb  ep-pair  ep-surf  em-comb  em-pair  em-surf  es-comb  es-pair  es-surf
                 -8.41   -17.31     4.91    47.36    15.63    17.53    11.98     9.00     4.97
```

Figure 9: The file z-result.slp (lines are wrapped). Z-scores and ranks show clearly that structure 2gn5 is incorrect.

surface potential $PROSA_BASE/pII3.0.surf-cb scb Load surface C^β potential and call it 'scb'.

pair potential $PROSA_BASE/pII3.0.pair-ca pca Load pair C^α potential and call it 'pca'.

surface potential $PROSA_BASE/pII3.0.surf-ca sca Load surface C^α potential and call it 'sca'.

2. Load proteins

read pdb pdb2aat.ent aat Read protein '2aat' and call object 'aat'.

read pdb pdb1spa.ent spa Read protein '1spa'and call object 'spa'.

3. Specify potentials and analyse energy

use potential aat pca sca pcb scb Use potentials 'pcb','scb', 'pca' and 'sca' for object 'aat'.

use potential spa pca sca pcb scb Use potentials 'pcb','scb', 'pca' and 'sca' for object 'spa'. NOTE: The latter two commands could be replaced by *use potential * **.

analyse energy * Analyse energy with specified potentials.

4. Set parameters

color * aat yellow

color * spa red

winsize * 50

5. Plot graphs of combined energy

plot

6. Show difference of C^α and C^β graphs

diff aat spa diff Calculate difference and write result to object 'diff'.

plot Show difference graph. The difference graph will appear green. You see graphs for combined energy.

7. Show pair energy

draw * * 0

draw pair * 1 Show only pair energy.

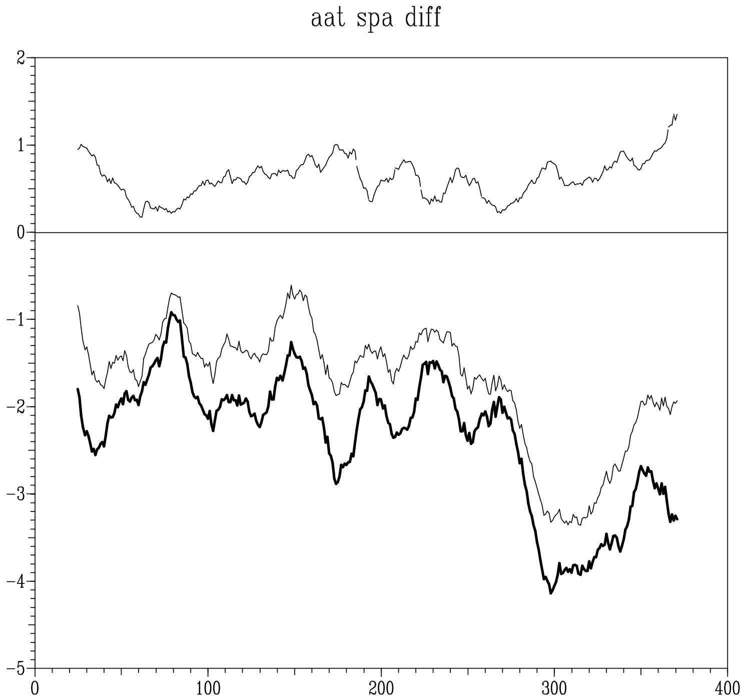


Figure 10: Pair energy graphs of 2aat (thin line) and 1spa (bold line). Potentials with both C^{α} and C^{β} interactions were used. The dashed line shows the difference. (Session 5)

`plot`	The pair energy graphs appear, the difference graph shows difference of pair energies. (Figure 10)

4.5.6 Session 6

Two proteins containing C^{α} coordinates only are analysed. One of them is 2lzh, whose energy graphs resemble a native protein. On the contrary, the energy graphs of 1phy look very suspicious.

1. Load C^{α} potentials.

`pair potential $PROSA_BASE/pII3.0.pair-ca pca`	Load pair C^α potential and call it ’pca’.
`surface potential $PROSA_BASE/pII3.0.surf-ca sca`	Load surface C^α potential and call it ’sca’.

2. Load Proteins

`read pdb pdb1phy.ent phy`	Read protein ’1phy’ and call object ’phy’.
`read pdb pdb2lzh.ent lzh`	Read protein ’2lzh’ and call object ’lzh’.

3. Specify potentials and analyse energy

`use potential phy pca sca`	Use potentials ’pca’ and ’sca’ for object ’phy’.
`use potential lzh pca sca`	Use potentials ’pca’ and ’sca’ for object ’lzh’.
`analyse energy *`	Analyse energy with specified potentials.

4. Set parameters

```
color * phy yellow
color * lzh red
winsize * 10
```

5. Plot graphs of combined energy

`plot`	(Figure 11)

4.5.7 Session 7

This session demonstrates how to substitute amino acids in ***ProSa 2003*** objects. Energy profiles and z-scores are generated and used to analyse the effects of these mutations.

1. Load protein

`read pdb pdb1ubi.ent wt`	Load the pdb file ’pdb1ubi.ent’ and call the resulting object ’wt’.

2. Substitute amino acids

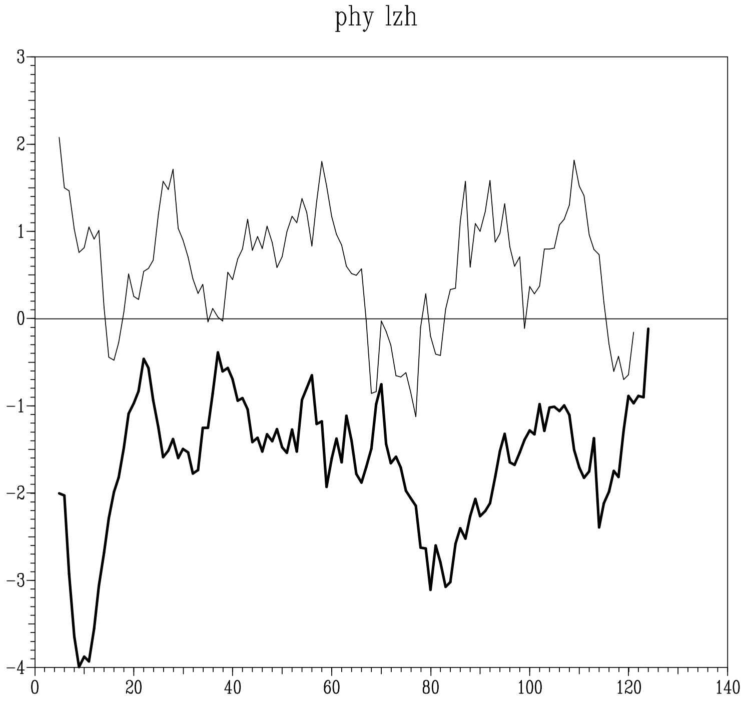


Figure 11: Graphs of combined energy of two C^{α} proteins, 2lzh (bold line) and 1phy (thin line). The energy graph of 1phy is suspicious. (Session 6)

`mutate sequence wt 46 P mutant1` Substitute the amino acid at position 46 of object 'wt' with a proline (P). The resulting object named 'mutant1' inherits all properties from 'wt' except the sequence, its name and all calculated energies.

`mutate sequence wt 5 E mutant2`

`mutate sequence mutant2 32 P mutant3`

`mutate sequence mutant3 47 L mutant4` Substitute three other amino acids. To accumulate mutations, repeatedly apply `mutate sequence` on mutant objects.

`list objects` A list of all currently existing objects is displayed. The mutation is shown as suffix to the protein name.

3. Perform analysis of local effects of mutations

`analyse energy *`

`color * mutant1 red`

`hide *`

`show wt`

`show mutant1`

`plot` Calculate energies for all objects, color energy graphs of 'mutant1' red, show only 'wt' and 'mutant1', display graphs.

A difference in the energy graphs appears around residue 46. To make this more clear, a difference plot can be created by:

`diff wt mutant1 diff1`

`hide *`

`show diff1`

`plot` Calculate difference plot between 'wt' and 'mutant1', hide all objects, plot difference graph. The positive peak at residue 46 shows that the energy of the mutant is lower than that of the wild type, indicating a stabilising mutation.

`diff wt mutant4 diff4`

`color * diff4 blue`

```
show diff4
plot
```

The difference plot for 'mutant4' displays three negative main peaks, indicating that all three mutations in 'mutant4' are destabilising. The smaller peaks show that these mutations also affect long range interactions. (Figure 12)

4. Perform analysis of global effects of mutations

```
init zscore
```

Initialise z-score calculation.

```
delete diff1
delete diff4
```

Delete difference objects since z-score calculations cannot be applied to them.

```
zscore *
```

We use the `*` argument to apply the `zscore` command to all objects we have generated so far. The result (Figure 13) shows that 'mutant1' has a higher z-score than 'wt', whereas 'mutant2' to 'mutant4' have a lower one. Hence, A46P is supposed to be a stabilising mutation, whereas the others are destabilising (also note the ranks of 'mutant4'!).

4.5.8 Session 8

This session demonstrates how to examine the mutability of specified residues in a protein. Two positions in an immunoglobulin structure are compared with respect to the number of stabilising mutations among all possible substitutions. A single-site mutability profile is generated for two regions of the structure, pointing at sites that may be more important for structural stability than others. To examine multiple-site mutability, 2×10^3 mutants for two different sets of solvent-exposed residues are randomly generated. It is examined how many of these mutants are destabilised with respect to the wild type.

1. Initialise z-score calculation and load a protein

```
init zscore
read pdb pdb1aqk.ent,H, Fd
```

Load chain H of 1aqk and name the resulting object 'Fd'. This is the structure of an immunoglobulin Fd fragment.

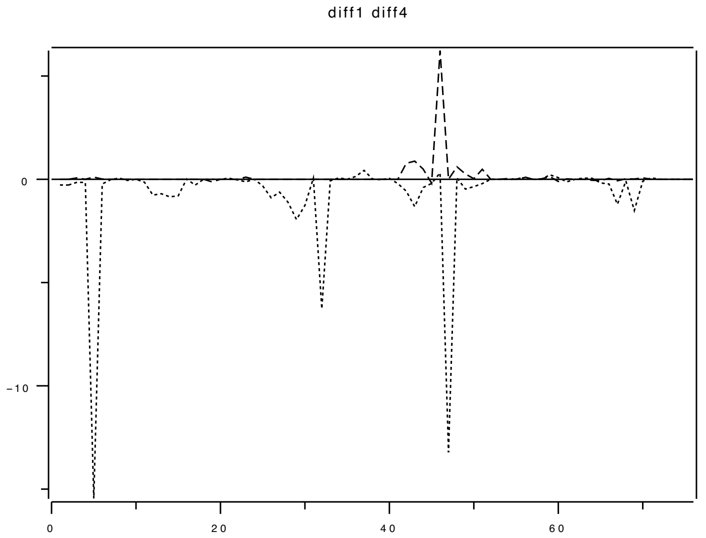


Figure 12: Difference plots for two mutants of 1ubi. (Session 7)

```
Hide & Seek on polyprotein pII3.0.short.ply - selection of parameters
molecule          seq-l zp-comb zp-pair zp-surf rk-comb rk-pair rk-surf

wt                  76   -8.94   -5.74   -6.48       1       1       1
wt_A46P             76   -9.54   -6.64   -6.63       1       1       1
wt_V5E              76   -7.90   -4.86   -5.86       1       1       1
wt_V5E_D32P         76   -7.68   -4.46   -5.84       1       1       1
wt_V5E_D32P_G47L    76   -6.71   -3.92   -5.08       1       3       1
```

Figure 13: Z-scores and ranks for a wild type protein and four mutants. (Session 7)

2. Substitute amino acids exhaustively

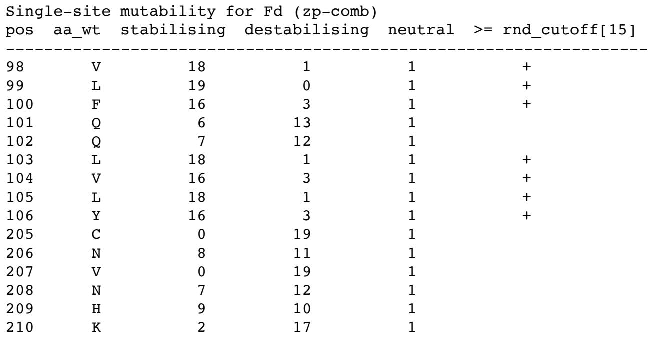
analyse mutability Fd 69 This command generates all possible 20 mutants for residue 69. The mutant objects are called Fd_I69y, where y is the introduced amino acid. Z-scores for all mutant objects are calculated and the portion of stabilising, destabilising and neutral mutations is recorded. These values are finally put to standard output.

3. Calculate mutability profile

analyse mutability Fd 98-106,205-210 Fd_xmpl To check single-site mutability for more than one residue, we specify a list of positions. Given a computing time of roughly 20 seconds per z-score calculation for a protein of this length on a 1.4 GHz CPU, the analysis will approximately take 1.75 hours (15 residues × 20 amino acids × 20 secs.). The output is collected in four files: *Fd_xmpl.slp* holds the z-scores for wild type and mutants and *Fd_xmpl.mut_comb*, *Fd_xmpl.mut_pair* and *Fd_xmpl.mut_surf* hold the mutability profiles for the three energy terms. As seen in Figure 14, all randomisable residues are marked with a '+'. The residues of the first set (98-106) highly tolerate single-site mutations. This result is promising, since the residues lie within the hypervariable region of the Fd fragment. The residues of the second set (205-210) are part of a β-sheet in the C-terminal half of the Fd fragment. Each of them is much more restricted in terms of the allowed amino acid substitutions. Remarkably, only a cysteine is accepted at position 205 (bridging to a cysteine at 149), all other mutations are regarded as destabilising.

4. Create and analyse pool of mutants

randomise sequence Fd 153,180,182,184 Fd_epi1 This command first derives 10^3 mutants (default value) from the object 'Fd' by substituting all amino



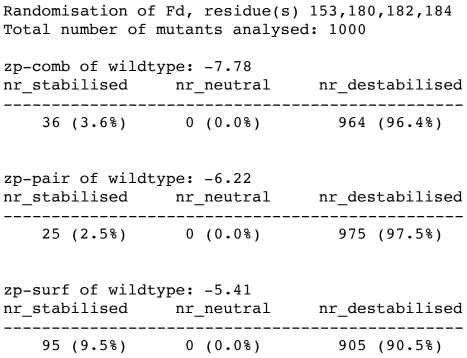
```
Single-site mutability for Fd (zp-comb)
pos  aa_wt  stabilising  destabilising  neutral  >= rnd_cutoff[15]
-----------------------------------------------------------------
98      V         18           1          1          +
99      L         19           0          1          +
100     F         16           3          1          +
101     Q          6          13          1
102     Q          7          12          1
103     L         18           1          1          +
104     V         16           3          1          +
105     L         18           1          1          +
106     Y         16           3          1          +
205     C          0          19          1
206     N          8          11          1
207     V          0          19          1
208     N          7          12          1
209     H          9          10          1
210     K          2          17          1
```

Figure 14: Single-site mutability profile for residues 98-106 and 205-210 of 1aqk, chain H. (Session 8)

acids in the specified positions with randomly chosen ones. For each mutant, z-scores are calculated and compared to the 'Fd' z-scores. Since 1001 z-scores have to be determined, this calculation will take about 5.5 hours on a 1.4GHz CPU. Results are written to two files: *Fd_epi1.slp* contains all the z-scores, and *Fd_epi1.nrm* holds a summary about the number of stabilised, destabilised and neutral mutants within the sample population. Only a very small number of mutants (3.6%) shows a combined z-score that is wild type-like or below, the majority is destabilised (Figure 15).

randomise sequence Fd 100,104-106 Fd_epi2

In this region, which is part of the natural epitope of the immunoglobulin structure, the situation is almost reversed: only a minority of the mutants (1.1%) exhibits a combined z-score that indicates destabilisation (Figure 16). This result is in good agreement with our prior knowledge about immunoglobulin structures: the amino acids of the epitope are supposed to be primarily selected for binding of the antigen, rather than for their contribution to structural stability.



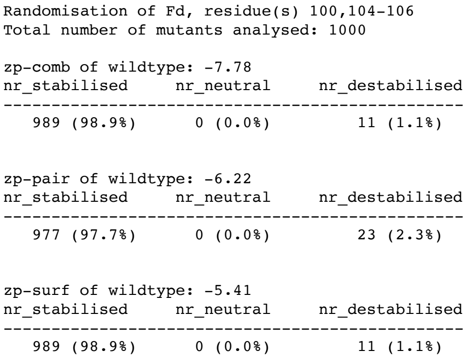
```
Randomisation of Fd, residue(s) 153,180,182,184
Total number of mutants analysed: 1000

zp-comb of wildtype: -7.78
nr_stabilised     nr_neutral     nr_destabilised
------------------------------------------------
    36 (3.6%)      0 (0.0%)       964 (96.4%)


zp-pair of wildtype: -6.22
nr_stabilised     nr_neutral     nr_destabilised
------------------------------------------------
    25 (2.5%)      0 (0.0%)       975 (97.5%)


zp-surf of wildtype: -5.41
nr_stabilised     nr_neutral     nr_destabilised
------------------------------------------------
    95 (9.5%)      0 (0.0%)       905 (90.5%)
```

Figure 15: Result of randomisation on a region of an Fd fragment (PDB file 1aqk, chain H). The region consists of four spatially close residues in the C-terminal half of the chain. (Session 8)



```
Randomisation of Fd, residue(s) 100,104-106
Total number of mutants analysed: 1000

zp-comb of wildtype: -7.78
nr_stabilised     nr_neutral     nr_destabilised
------------------------------------------------
   989 (98.9%)      0 (0.0%)        11 (1.1%)


zp-pair of wildtype: -6.22
nr_stabilised     nr_neutral     nr_destabilised
------------------------------------------------
   977 (97.7%)      0 (0.0%)        23 (2.3%)


zp-surf of wildtype: -5.41
nr_stabilised     nr_neutral     nr_destabilised
------------------------------------------------
   989 (98.9%)      0 (0.0%)        11 (1.1%)
```

Figure 16: Result of randomisation on part of the antigen binding region of an Fd fragment (PDB file 1aqk, chain H). (Session 8)

5 ProSa 2003 Commands and Variables

5.1 Syntax

5.1.1 Commands

A command line has the following form:

`command [subcommand1 [subcommand2] ]` [*arguments ...*]

Examples:

read pdb {*filename*} [*object*]	command subcommand1 argument argument
analyse energy {*object*}	command subcommand1 argument
color {*token*} [*object*] {*color*}	command argument argument argument

Commands and arguments are insensitive to more than one white space character (e.g. "color title green" is the same as "color title green ", but "colortitlegreen" is of course invalid).

The command interpreter can evaluate environment variables. E.g. if you have an environment variable 'PDBDIR=/usr/mydir/pdb/', you can type 'pdb_dir = $PDBDIR' to set *pdb_dir* to the value of $PDBDIR. Environment variables are not handled as real ***ProSa 2003*** variables. They are just substituted by their current value.

5.1.2 Variables

ProSa 2003 needs a set of variables. Variables can be of type float, integer, boolean or string. To set a variable type the name, followed by a space, an '=' and the value.

`variable` = {*value*}

To display the current value of a variable type the variable name followed by a space and a question mark.

`xmax ?`

The answer will look like

```
xmax = 8.000000e+01
```

5.2 Input

read pdb {*filename[,[chain],[model]]*} [object]

Reads the Brookhaven file named {*filename*}. If a chain identifier {*chain*} and/or a model number {*model*} is given, the respective polypeptide will be extracted from the PDB file. Otherwise the first chain/model in the PDB file will be read.

Examples: `read pdb pdb2hhb.ent,B,` reads chain B, whereas `read pdb pdb2hhb.ent` reads chain A, as this is the first chain in the PDB file. `read pdb pdb1ans.ent,,2` reads model 2 of 1ans.

An object name can be specified in [*object*]. If [*object*] is missing, the protein's name will be used as the object name. In case the name already exists for an other object, an index is added to the name automatically. This ensures that object names are unique.

PDB files which contain only C^α coordinates can also be read. ***ProSa 2003*** identifies the file type automatically. Such C^α files can only be analysed with C^α potentials. Make sure that you do not specify any other potential type for C^α files.

print residue mapping {*object*} [filename]

Displays the mapping of PDB residue numbers to ***ProSa 2003*** residue numbers.

read bbn {*filename*} [object]

Reads a file in **b**inary **b**ackbo**n**e format. A bbn-file must have been generated by a *write bbn* command.

pair potential {*filename*} {*idname*}

Loads the specified pair potential file. Pair potential files have the extension '.frq'. Note that {*filename*} has to be a full pathname (i.e. directory/name) and must not carry the file extension.

Several pair potential files can be loaded. Each potential must be given an identification name. If a potential with given idname already exists, it will be replaced by the new potential.

Both pair and surface potential files are loaded in the startup file. By default, C^β potentials are used. To use other potentials than those given in the startup file, you can edit the startup file or load additional potentials with the respective commands either in your command file or interactively.

For each protein (object), it must be specified which potential(s) to use for energy calculations. When a new object is loaded, ***ProSa 2003*** automatically sets the current potentials to be used for this object. If an object is loaded before any potential(s), no potential are specified for the object. In this case potentials have to be specified later by the user. To change the potential(s) which should be used for an object, use the command *use potential*.

All potentials which are set for an object will be used for both the energy analysis and for calculation of Z-scores. The energies are summed up for the results. Thus, e.g. if you specify C^{α}- C^{α} and C^{β}- C^{β} potentials, the energy analysis will include both interactions.

surface potential {*filename*} {*idname*}

Loads surface potential file. Surface potential files have the extension '.sff'. Note that {*filename*} has to be a full path name (i.e. directory/name) and must not carry the file extension.

See also *pair potential*.

5.3 Output

print energy {*object*} [filename]

Prints a file containing the results of an energy analysis for {*object*}. If [*filename*] is omitted or if the command is executed for all objects ({*object*} = *), the output files have the same name(s) as the object(s) with extension '.ana'. The extension '.ana' is added automatically to the filename.

Each file contains four columns. The first column contains the residue numbers. The others contain the values of pair energy, surface energy and combined energy, respectively. The values are smoothed by the window size set in the object. If the data are smoothed, residue numbers refer to the central residue in the window. E.g. if winsize is 20 residues, position of the first window is 10. This facilitates processing of data using other graphic software.

The first lines specify the protein name and window size. This line starts with a '#'.

The file can be used as input to other programs like gnuplot.

export plot {*filename*}

Writes the current plot to a postscript file. The extension '.ps' will be added automatically to the filename. Postscript plots are black and white. Screen colors are translated to different linestyles. See command *color* for mapping of screen colors to line styles.

write bbn {*object*} [filename]

This writes the backbone of the given object in binary format to [*filename*]. If no filename is specified, the file will be named by {*object's*} name. The extension '.bbn' is added.

5.3.1 Output Options

pscolor = {*boolean*}

Selects color output for postscript files instead of black-and- white linestyles. If "pscolor = 1" lines are printed with colors, if "pscolor = 0" lines are translated to different linestyles. Default is 0. Axes and titles are always black.

5.4 Energy Analysis

analyse energy {*object*}

Performs an energy analysis for *object*. The results can be plotted or exported to a data file.

diff {*object1*} {*object2*} [newobject]

The purpose of *diff* is to generate a graph showing the energy difference between *object1* and *object2*. The energy of each residue of *object2* is substracted from the energy of the corresponding residue of *object1*. This is done for each energy term separately. In case of objects with unequal length or if objects are shifted against each other, only the overlapping region is considered.

When using *diff* a new object *newobject* is generated. This object contains the difference data for each energy term. The results can be plotted or exported to a data file as can be done with any object.

By default, difference graphs will appear green. All other plotting parameters of the new object are assigned the same values as in *object1*. The values can be changed as in any object, although it is not recommended to do so.

If the energy graphs of *object1* and *object2* are smoothed by a window size, the difference graphs will be smoothed by the same window size. In this case first the energy differences are calculated for each residue and second the difference data are smoothed. Thus, in margin areas of the graph, the difference plot will not represent the exact difference between the smoothed energy graphs of *object1* and *object2*.

5.5 Object Handling

`hide` {*object*}

Hides an object. Hidden objects are not shown on a plot. The command does not delete or change any data. Hidden objects remain active in the background and can be accessed by the commands as usual. Thus, they are also affected by commands using '*' for the {*object*} argument.

`show` {*object*}

Activates {*object*} for display.

`delete` {*object*}

Removes {*object*} and all its data.

`copy` {*oldobject*} [newobject]

{*oldobject*} is copied to [*newobject*]. If [*newobject*] is missing or already exists, an index is added to [*newobject's*] name.

Copies can be used to draw graphs generated with different parameters on the same plot.

`list objects`

Prints the names of all objects and the corresponding proteins currently loaded. Visible objects are marked by an asterisk (*).

`info object` {*object*}

Prints information on several parameters of *object*.

`rename` {*oldobject*} {*newobject*}

Renames the object in {*oldobject*} with {*newobject*}.

5.6 Plot

`plot`

Draws energy graphs on the screen. Changes of an object's parameters become visible only after you issue a *plot* command.

5.7 Graph Editing

`draw` {*token*} {*object*} {*boolean*}

Controls the types of energy graphs displayed in a plot. {*token*} specifies the energy type. Valid entries are *pair, surf, comb* and *. '*' affects all three types. Setting them to 1 adds the curve of the corresponding energy type to the plot, setting it to 0 removes it. By default, only the combined energy is plotted.

`winsize` {*object*} {*integer*}

Winsize controls the width of gliding averages. If *winsize* = 1 no averaging is performed.

`graph title` [*string*]

Edits the title of the graph. 'off' as argument removes the title. If [*string*] is omitted the title lists the object names whose graphs are displayed on the plot.

`xaxis title` [*string*]

Edits the title of the x axis. 'off' as argument removes the title. *Xaxis title* with no argument sets the title to 'residues'.

`yaxis title` [*string*]

Edits the title of the y axis. 'off' as argument removes the title. *Yaxis title* with no argument sets the title to 'energy'.

`color` {*token*} [object] {*color*}

With this command you can set colors. Valid entries for token are *axis, title, pair, surf, comb* and *back* (background color). '*' can be used to specify pair, surface and combined energy of an object. If *pair, surf, comb* or * is used for {*token*}, the object must be specified. Available colors are *white, red, yellow, green, blue, cyan, magenta* and *black.* Note that the colors are translated to different linestyles in a postscript plot. This affects graphs only, while axes and fonts are always plotted with solid lines.

Color translation map:

yellow	———————
blue	....
cyan	. . . .
green	- - - - - - - - - - - - - - -
red	━━━━━━━
magenta	....
white	. . . .
black	- - - - - - - - - - -

shift {*object*} {*offset*}

Shifts relative position of {*object's*} graphs along the sequence axis by the amount specified. Positive values cause a shift to the right, negative ones to the left. This feature is useful if the graphs of two proteins of different length are compared (e.g. session 4).

xmin = {*float*}

Sets the minimum value of the x axis. The default is *xmin* = 0. Note that the *xmin* value is only active if autoscaling is off, i.e. if *xmax* has a value different from 0.

xmax = {*float*}

Sets the maximum value of the x axis. Automatically turns x axis autoscaling off. *xmax* = 0 reactivates autoscaling.

ymin = {*float*}
ymax = {*float*}

The same as *xmin, xmax* for the y axis.

xticks = {*integer*}
yticks = {*integer*}

Specifies the number of tickmarks on the x and y axis, respectively. *xticks* is only active, if autoscaling is off. Otherwise tickmarks are set automatically. *ticks* = 0 removes tickmarks.

zeroaxis = {*boolean*}

If *zeroaxis* = 1 the x axis is drawn. By default, *zeroaxis* is on. *zeroaxis* = 0 suppresses drawing of the x axis.

5.8 Energy and Potential Parameters

factor_pair = {*float*}
factor_surf = {*float*}

The total energy is a combination of pair and surface energies. These energy terms are not directly comparable. They must be properly weighted. Recommended values are *factor_pair* = 1, *factor_surf* = 5. For small proteins smaller values for *factor_surf* (∼ 3) are recommended.

lower_k = {*integer*}
upper_k = {*integer*}

Pair interaction energies are calculated for residue pairs whose distance k along the sequence is *lower_k* ≤ k ≤ *upper_k*. Default values are *lower_k* = 1, *upper_k* = 600. For example if you want to see only the short range energy contributions (e.g. sequence separation k ≤ 9) set *lower_k* = 1, *upper_k* = 9. These parameters affect pair interactions only.

pot_lb = {*float*}
pot_ub = {*float*}

Pair energies are calculated in the distance range [*pot_lb, pot_ub*] Å. Outside this range energies are zero. You can set the desired distance range of pair potential calculations. For example if you are not interested in energy contributions of close contacts, set *pot_lb* = 4. The energy of pair interactions in the intervall [0; *pot_lb*] is then zero. Default: *pot_lb* = 0, *pot_ub* = 15. In addition the upper bound depends on the pair potentials used. The current maximum range is 15Å.

combine type {*token*}

Specifies the type of energy combination. For energy combination, surface and pair energies can be weighted by user defined values or by the standard deviations of pair and surface energy distributions.

To use user defined values, set *combine type* to 'free'. Energies are then weighted by the current values of *factor_pair* and *factor_surf*. If *combine type* is 'sdev', energies are weighted automatically. In this case the standard deviation of pair (σ_p) and surface (σ_s) energy distributions are calculated. The surface energies are then weighted by $\frac{\sigma_p}{\sigma_s}$.

Note that *combine type sdev* sets *factor_pair* = 1 and *factor_surf* = $\frac{\sigma_p}{\sigma_s}$. These values are now used in energy graphs. They may be inappropriate for other proteins.

use potential {*object*} {*potential name(s)*}

Specifies, which potentials should be used for given object. The interactions of all given potentials will be summed up for energy calculations.

info potential

Lists all loaded potentials and prints information.

potential type

Specifies the type of reference state for pair potentials. Type '1' uses the frequency of observing *any* two residues at a given distance, whereas type '2' uses the frequency of observing a *particular* amino acid at a given distance to any residue. The selected value applies to all loaded pair potentials. The default potential type is '1'.

skip_gly_cb = *boolean*

If *skip_gly_cb* = 1, interactions for virtual glycine C^β atoms are skipped in energy calculations, if *skip_gly_cb* = 0, they are included. Default value: 1.

5.9 Z-score Commands

init zscore [*polyprotein*]

Set up ***ProSa 2003*** for z-score calculations. The command reads a polyprotein, which by default is the polyprotein 'pII3.0.short.ply'. A different polyprotein can be used by invoking *init zscore* with the name of a polyprotein as argument. Polyproteins are constructed from protein modules which are connected by linker regions. A polyprotein is a device for efficient generation of alternative conformations for a given amino acid sequence. These conformations have good stereochemistry and have many features of native protein folds. The set of conformations derived from the polyprotein represents a sample of the conformation space of a given protein.

zscore {*object*} [filename]

The amino acid sequence of the protein {*object*} is combined with all conformations in the polyprotein and the energies are calculated. The z-score is derived from the resulting energy distribution. The z-score Z_p of {*object*} is obtained from the energy E_p of {*object*} by

$$z_p = \frac{(E_p - \overline{E})}{\sigma}$$

where $\overline{E}$ is the average energy of all fragments derived from the polyprotein and σ the associated standard deviation. The results are stored in two files with extensions '.sor' and '.slp', respectively.

***.sor** contains the energy sorted table of fragments derived from the polyprotein. A fragment is a section of the polyprotein. Its length is equal to the sequence length of {*object*}. The location of a fragment is described by its relative position to a protein module. The reference module (**molecule**) is defined as that protein in the polyprotein which has maximum overlap (**overlay**) with fragment. The position of the N-terminus of fragment with respect to the N-terminus of this protein (**molecule**) is defined as **offset**. The n fragments of lowest energy are appended to this file where n is specified by the variable *sorted_depth*. Output is suppressed by *sorted_depth* $= 0$. The output consists of three tables corresponding to the three energy terms pair, surface and combined, respectively.

The columns are:

rank Fragment's position in energy sorted table.

molecule The fragment is contained in this protein module. (Brookhaven code)

offset **Offset** is the position of the N-terminus of fragment relative to the N-terminus of **molecule**. If the N-terminus of fragment is upstream of the N-terminus of **molecule** then **offset** is negative.

overlay Number of residues of fragment aligned with **molecule** (i.e. number of overlapping residues of fragment and **molecule**).

z-score Z-score of fragment.

energy Energy of fragment (with respect to the amino acid sequence of {*object*}).

rms Root mean square deviation of fragment and {*object*}.

seq-id Relative sequence identity of fragment and {*object*} (number of identical residues in fragment and {*object*} divided by sequence length).

***.slp** contains z-score and energy values of {*object*}.

seq-l Sequence length of {*object*}.

zp Z-Score of {*object*}.

rk Rank (relative position) of {*object*} in the energy sorted table.

zl Z-score of fragment of lowest energy found in the polyprotein.

ep Energy of {*object*}.

em	Average energy of all fragments derived from the polyprotein.
es	Standard deviation of energies obtained from the polyprotein.

If no filename is specified, results are written to standard output.

CPU time for z-score calculations depends on the sequence length of {object} and on the parameters used. For example it takes about 4 seconds CPU time for 5pti (58 residues, C^{β}- C^{β} interactions, polyprotein pII3.0.long.ply) and about 14 seconds for 1mbd (153 residues, same parameters) on an AMD Athlon 1.4 GHz system. You can save time by using the small polyprotein (∼ one third) but the scores can be less reliable.

sorted_depth = {*integer*}

Sorted_depth is the number of fragments saved from the energy sorted table to the *.sor file. Default is 0, hence the output of this file is suppressed.

5.10 Mutagenesis

mutate sequence {*wild type object*} {*residues*} {*amino acid*} [*mutant object*]

Generates one or more mutant object(s) by copying {*wild type object*} and substituting some of its amino acids.

The {*residues*} argument is used to specify one to several residues that are to be substituted. It is a comma-separated list of positions or ranges. A position denotes the sequence index of the residue (start: 1, end: length of sequence). A range is specified by two positions (from, to), separated by a dash. Example for valid values: ’5’, ’6-12’, ’1,7-9’. ’*’ can be used to specify all residues.

The {*amino acid*} argument specifies the introduced amino acid (one capital letter code, e.g. ’F’). Alternatively, a ’*’ tells ***ProSa 2003*** to generate one mutant for each amino acid of the current alphabet (see variable *alphabet*).

The optional argument *mutant object* allows to specify a name for the resulting mutant object. If it is omitted, the name of the wild type object is used, followed by an underscore, the wild type amino acid, the residue number and the mutant amino acid, e.g. wildtype_V15F.

analyse mutability {*wild type object*} {*residues*} [*filename*]

Each amino acid at the specified residues in *wild type object* is exhaustively replaced with all amino acids of the currently defined alphabet. The z-scores z_{mut} of the resulting mutant objects are calculated and compared to the wild type z-scores z_{wt}, and the number of stabilising, destabilising and neutral mutations is recorded. A mutation is regarded as stabilising if $z_{wt} - z_{mut} > 0$, as destabilising if $z_{wt} - z_{mut} < 0$ and as neutral if $z_{wt} - z_{mut} = 0$. Results are written to three files called *filename.mut_surf*, *filename.mut_pair* and *filename.mut_comb*, corresponding to the three energy terms, respectively. In these files, all residues that yield more than *rnd_cutoff* non-destabilising mutations are marked with a '+'. The z-scores of wild type and mutants are stored in a file called *filename.slp*. If *filename* is omitted, results go to standard output. Note that an *init zscore* command has to precede this command, otherwise no z-score calculations are possible.

See the *mutate sequence* command for a description of the *residues* argument.

randomise sequence {*wild type object*} {*residues*} [*filename*]

Generates a sample population of mutant objects derived from {*wild type object*} by substituting all amino acids in the specified positions with randomly chosen ones. The size of the population is determined by the variable *nr_mutants* and the amino acids are chosen from the set specified with the variable *alphabet*. For each generated mutant, z-scores (pair,surface, combined) are calculated and compared to the wild type z-scores. The portion of stabilising, destabilising and neutral mutants within the sample population is determined an written to a file called *filename.nrm*. The z-scores of wild type and mutants are stored in a file called *filename.slp*. If *filename* is omitted, results go to the standard output. Note that an *init zscore* command has to precede this command, otherwise no z-score calculations are possible.

See the *mutate sequence* command for a description of the *residues* argument.

alphabet = {*string*}

Specifies the list of amino acids that are used for exhaustive or random amino acid substitutions. The default set consists of all 20 standard amino acids ('ARNDCQEGHILKMFPSTWYV').

rnd_cutoff = {*integer*}

Minimum number of mutations among all possible 20 that have to be non-destabilising in order to characterise a residue as randomiseable. Randomiseable residues are marked with a '+' in the output of the *analyse mutability* command. The default value is 15.

nr_mutants = {*integer* }

Size of the sample population of mutants generated by the *randomise sequence* command. Default: 1000.

5.11 Miscellaneous

pdb_dir = {*directory*}

By default, Brookhaven data files are read from this directory. If a Brookhaven file with the same file name exists in the local directory the latter has priority.

pdb_dir can accept multiple path names. Type the path names, separated by a colon ':' (semicolon ';' on Windows systems). No spaces must occur in the string. E.g.:

pdb_dir = /usr/brookhaven/disk1:/usr/brookhaven/disk2

Brookhaven files can be in compressed form. Compressed PDB files are uncompressed in the directory /usr/tmp/. I.e. they are copied to /usr/tmp/, uncompressed and deleted. Thus no write permission is required for PDB directories.

bbn_dir = {*directory*}

This is the default directory for Binary Backbone files. Like *pdb_dir, bbn_dir* can accept multiple path names (see above).

Bbn-files have several advantages. They can be loaded faster than Brookhaven files and they need less memory than Brookhaven files.

To create a bbn file, use the command *write bbn.* The file is created in your local directory. To read a binary backbone file, use *read bbn.* ***ProSa 2003*** searches the local directory first and then *bbn_dir* to locate *.bbn files.

help [*argument*]

Yields information on all commands and variables. *Help* with no argument lists all commands and variables. The argument can be any command name or variable or any part of a name. All commands containing the given string will be shown. Each command is shown with a brief description. For example,

help pdb

displays

```
Initializing help ...
read pdb:
     {filename} [object]
     read Brookhaven Protein File
-----------------------------------------------------------------
pdb_dir (const string):
     default path to brookhaven style files
-----------------------------------------------------------------
```

`execute` {*filename*}

With this command you can read and execute command files.

`run python` {*filename*}

Run a Python script that may include calls to ***ProSa 2003*** commands. Use the following syntax to call ***ProSa 2003*** commands from Python:

```
Prosa.passToProsa('{prosa command}')
```

`exit`
`quit`

Exit ***ProSa 2003***.

`log =` [*boolean*]

If *log* = 1, all commands executed are stored in the file *ProSa2003.log*. Default *log* = 0.

`ver`

Prints name of current version.

6 Bugs and Suggestions

Please report problems, suggestions, wishes, etc. to:

E-mail

prosa@came.sbg.ac.at

Fax

”ProSa 2003”

(+43) 662 / 8044-176

Address:

Center of Applied Molecular Engineering
Jakob Haringerstr. 5
A-5020 Salzburg
Tel.: (+43) 662 / 8044-5797

APPENDICES

A Related Publications

References

[1] G. Casari and M. J. Sippl. Structure-derived hydrophobic potential. Hydrophobic pot ential derived from X-ray structures of globular prote ins is able to identify native folds. *J.Mol.Biol.*, 224:725–732, 1992.

[2] M. Hendlich, P. Lackner, S. Weitckus, H. Floeckner, R. Froschauer, K. Gottsbacher, G. Casari, and M. J. Sippl. Identification of native protein folds amongst a large n umber of incorrect models. The calculation of low ener gy conformations from potentials of mean force. *J.Mol.Biol.*, 216:167–180, 1990.

[3] W. A. Koppensteiner and M. J. Sippl. Knowledge-based potentials–back to the roots. *Biochemistry (Mosc).*, 63:247–252, 1998.

[4] M. J. Sippl. Calculation of conformational ensembles from potentials of mean force. An approach to the knowledge-based pred iction of local structures in globular proteins. *J.Mol.Biol.*, 213:859–883, 1990.

[5] M. J. Sippl. Boltzmann's principle, knowledge-based mean fields and p rotein folding. An approach to the computational deter mination of protein structures. *J.Comput.Aided Mol.Des.*, 7:473–501, 1993.

[6] M. J. Sippl. Recognition of errors in three-dimensional structures of proteins. *Proteins.*, 17:355–362, 1993.

[7] M. J. Sippl. Knowledge-based potentials for proteins. *Curr.Opin.Struct.Biol.*, 5:229–235, 1995.

[8] M. J. Sippl and M. Jaritz. Predictive power of mean force potentials. In H. Bohr and S. Brunak, editors, *Distance Based Approaches to Protein Structure Determination*, pages 113–134. IOS Press, 1994.

[9] M. J. Sippl, M. Jaritz, M. Hendlich, M. Ortner, and P. Lackner. Applications of Knowledge Based Mean Fields in the Determinatio n of Protein Structures. In S. Doniach, editor, *Statistical Mechanics, Protein Structure and Protein Substr ate Interactions.*, pages 297–315. Plenum Press, New York, 1994.

[10] M. J. Sippl, S. Weitckus, and H. Floeckner. In search of protein folds. In K. H. Merz and S. LeGrand, editors, *The Protein Folding Problem and Tertiary Structure Prediction.*, pages 353–407. Birkhaeuser, Boston, 1994.

[11] M. Wiederstein, P. Lackner, F. Kienberger, and M.J. Sippl. Directed *in silico* mutagenesis. In S. Brakmann and A. Schwienhorst, editors, *Evolutionary Methods in Biotechnology*. Wiley-VCH, 2003.

For a complete list of publications at CAME visit: **`http://www.came.sbg.ac.at`**
For a survey of publications that reference *Prosa II* visit:
`http://www.came.sbg.ac.at:8080/CAME/CAME_EXTERN/ProsaII/references_html`

B Potential Files Distributed with ProSa 2003

pII3.0.pair-ca.frq	Pair potentials (C^α - C^α) as distributed with Prosa II
pII3.0.pair-cb.frq	Pair potentials (C^β - C^β)as distributed with Prosa II
pII3.0.surf-ca.sff	Surface potentials (C^α) as distributed with Prosa II
pII3.0.surf-cb.sff	Surface potentials (C^β) as distributed with Prosa II
prosa2003.pair-ca.frq	Pair potentials (C^α - C^α) compiled from SCOP domains (v. 1.61)
prosa2003.pair-cb.frq	Pair potentials (C^β - C^β) compiled from SCOP domains (v. 1.61)
prosa2003.surf-ca.sff	Surface potentials (C^α) compiled from SCOP domains (v. 1.61)
prosa2003.surf-cb.sff	Surface potentials (C^β) compiled from SCOP domains (v. 1.61)