Kpax User Manual – Version 5.0.0

https://mbi-ds4h.loria.fr/kpax_software/

Table des matières

Installing *Kpax*

1. Prerequisites

Kpax currently only works on Linux systems. This is mainly because it makes heavy use of the Linux directory structure to store its data and to organise its output. In the future, Windows and Mac versions might also become available. In order to run *Kpax* on Linux, you will need a relatively recent Linux distribution such as Linux Mint 17 or Ubuntu 14 . The *Kpax* binaries were compiled on 64-bit versions of Mint 17.2. The latest binary has been linked statically. This means it should run on other Linux distributions.

2. The Self-Installer

The easiest way to install Kpax is to download and run the self-installer script. Assuming you have a 64-bit system, open a command terminal and enter something like this:

```
sh kpax-5.0.0-x64-mint17.bin
```

The self-installer script will ask some questions about where you wish to install Kpax and whether you will let the script modify one of your shell start-up scripts in order to run the program from the command line. The installer will normally define an environment variable called KPAX_ROOT, and it will add ${KPAX_ROOT}/bin to your command path. It will also run a test script located in the ${KPAX_ROOT}/test subdirectory. If you do not trust the self-installer script, you can extract and inspect the installation tar file as follows:

```
sh kpax-5.0.0-x64-mint17.bin --noexec
cd kpax-dist-5.0.0
gunzip kpax-5.0.0-x64.tgz
tar vtf kpax-5.0.0-x64.tar
```

You can then create your own installation directory, define the KPAX_ROOT variable, and install the installation files yourself. Please see the file ${KPAX_ROOT}/doc/README for further details.

Running *Kpax*

1. Superposing Structures
2. File Naming Conventions
3. The Printed Results Table
4. Sorting the Output
5. Flexible Structure Alignments
6. Multiple Structure Alignments
7. Scoring Pair-wise and Multiple Alignments

1. Superposing Structures

Kpax can compare and superpose multiple protein structures in a single run. Each structure must be given as a separate PDB file. Normally, the first PDB file is treated as the "query" structure, and any subsequent PDB files are taken as the "target" or "database" files which will be compared to the query structure. The following example compares one query against three "target" structures from the Kpax test directory:

```
cd $KPAX_ROOT/test
kpax d1bhga1.ent d1cs6a3.ent 1qb5D00 3ullA00
```

Here is the typical output:

```
Kpax 5.0.0 starting at Mon Sep 21 15:58:11 2015 on host hardy.
Creating RESULTS directory: ./kpax_results/
Using LOG file: ./kpax_results/kpax.log
*Warning* Chain break A:S-51 <--> A:D-61 in file 3ullA00
```

```
Creating RESULTS directory: ./kpax_results/d1bhga1
Done 3 alignments for 3 targets in 0.0066 seconds (458/s).
Done 3 superpositions in 0.0453 seconds (66/s).
Writing QUERY pdb file: ./kpax_results/d1bhga1/d1bhga1_query.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/d1cs6a3_d1bhga1.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/3ullA00_d1bhga1.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/1qb5D00_d1bhga1.pdb

================================================================================================
Top 3 matches for d1bhga1 (length 103) [0.0.0.0] -> [0.0.0.0] (3/3) ROC AUC = 1.000000
================================================================================================
Rank K-Score G-Score J-Score H-Score M-Score T-Score RMSD N/* P/$ I/@ I/% Len Seg TP Match
[Family]
===== ======= ======= ======= ======= ======= ======= ==== === === === === === === == =====
1 39.99 43.27 0.4130 0.4469 0.5538 0.6255 3.07 79 82 10 12.7  91 1 +1 d1cs6a3[0.0.0.0]
2 36.62 25.31 0.3505 0.2422 0.2812 0.3243 3.15 45 81  3  6.7 106 1 +1 3ullA00[0.0.0.0]
3 29.33 10.78 0.2904 0.1068 0.1381 0.1894 4.47 33 68  1  3.0  99 1 +1 1qb5D00[0.0.0.0]
================================================================================================
Mean 35.31 26.45 0.3513 0.2653 0.3244 0.3797 3.56 52 77 4 7.5 98 1 3
================================================================================================

Done 1 queries in 0.11 seconds (0.00 minutes).
Peak memory allocation: 12 Mb.
Total memory on exit: 0 Mb.
Kpax finished in a total of 0.18 seconds (0.00 minutes).
```

2. File Naming Conventions

The above output shows that *Kpax* has created a directory called $KPAX_ROOT/test/kpax_results/d1bhga1/, in which it has written the original query structure (now named d1bhga1_query.pdb) and the three superposed target structures, Except for the first query structure file, the naming convention of the output PDB files is always <target_name>_<query_name>.pdb, which is intended to convey that the file contains the superposed target structure in the coordinate frame of the query.

This file and directory naming convention might seem verbose at first, but it becomes very useful when dealing with multiple target or database structures, and even multiple query structures because it avoids filling up your current working directory with large numbers of results files. The only file that is written to the current directory is a file called kpax.log, which contains a copy of all of the text that is printed to the terminal.

Of course, if you want to compare only two structures, you would just supply two PDB file names on the command line. If you want to suppress writing results to sub-sub-directories, you can give the "-nosubdirs" option. For example,

```
kpax -nosubdirs bhga1.ent d1cs6a3.ent
```

will run a single pair-wise comparison which will write all results files to the ./kpax_results directory. If you really want the output files to appear in your current working directory (not recommended) you can set the KPAX_RESULTS environment variable to "."

3. The Printed Results Table

In addition to the various output files (described in more detail below), *Kpax* writes a line of information for each compared target structure, on order of similarity to the query structure. From left to right, this information includes: the alignment "K-score" (calculated before any superposition); the "G-score" (calculated after the superposition); the "J-score" (normalised K-score); the "H-score" (normalised G-score); the "M-score" (the Kpax multiple alignment quality score) the "T-score" (the TM-score of the alignment, as defined by the TM-Align

program); When two identical structures are aligned, the K and G scores will be numerically equal to the number of residues in each structure, and the J, H and M scores will be unity (1.0 = a perfect match).

The remaining values have the following meaning. M is the number of matched residues calculated after the Gaussian superposition (this corresponds to the number of aligned residues of other alignment programs). N is the number of initial residue equivalences calculate in the first alignment (no superposition) using the K-score. RMSD is the root means squared deviation of the superposition after Gaussian refinement (this corresponds to the superposition RMSD of other alignment programs). I is the number of residue identities found in the final Gaussian alignment (by default, residue type is not used in the alignment scoring function). Len is the length (number of residues) of the target structure.

The final two values are useful when searching CATH or SCOP databases. The last column shows the CATH or SCOP classification code of the matching database structure. If the calculation does not involve a CATH or SCOP database, zeros are shown for the classification code, as show above. On the other hand, assuming that the query structure exists in CATH or SCOP, the TP value shows whether the retrieved structure belongs to the same CATH or SCOP family (+1 means yes, the match is a "true positive"; 0 means no, the match is a "false positive")

4. Sorting the Output

By default, the results table is sorted according to the H-score (normalised 3D Gaussian superposition score). However, you can specify that the results table should be sorted on any of the numerical columns using the "-sort" option. For example, to order the results by RMSD, you would write

```
kpax -sort=R
```

You could equally specify K, J, G, H, T, N, M, A, or I, (all case-insensitive) as the sort codes.

5. Flexible Structure Alignments

By default, Kpax calculates rigid structure alignments (i.e. "-rigid" is the default). To calculate flexible alignments, just add the "-flex" command-line option:

```
kpax -flex d1bhga1.ent d1cs6a3.ent 3ullA00
```

This will cause Kpax to treat the first structure (d1bhga1.ent) as the rigid "query" onto which it will flexibly superpose the two following target structures using two structural superposition runs.

6. Multiple Structure Alignments

By default, Kpax calculates pair-wise alignments. To calculate multiple alignments, just specify "-multi" on the command line. For example, the following calculates a multiple alignment of three structures:

```
kpax -multi d1bhga1.ent d1cs6a3.ent 3ullA00
```

By default, Kpax will treat each structure in turn as the pivot structure, and it will return the alignment that gives the best overall M-Score (see below). To force Kpax to use the first structure as the pivot, use the "-nopivot" option:

```
kpax -multi -nopivot d1bhga1.ent d1cs6a3.ent 3ullA00
```

Kpax also supports multiple alignments of flexible structures, where the pivot structure is kept rigid and the other structures are flexibly aligned onto it before building the final multiple alignment. For example:

```
kpax -flex -multi -nopivot d1bhga1.ent d1cs6a3.ent 3ullA00
```

Here is text the output from Kpax 5.0.0 for the above example:

```
Kpax 5.0.0 starting at Wed Sep 16 16:28:56 2015 on host hardy.
Creating RESULTS directory: ./kpax_results/
Using LOG file: ./kpax_results/kpax.log
*Warning* Chain break A:S-51 <--> A:D-61 in file 3ullA00
Setting CORE threshold = 3 rows (including pivot)
Induced alignment: rows=3, cols=133, T=300, L=106, N=103, C=65, M=0.6618, P=1.8864
Final alignment: rows=3, cols=136, T=300, L=106, N=103, C=65, M=0.6647, P=1.8946
MSA pivot d1bhga1 : M=0.6647, C=65 (keeping)
Creating RESULTS directory: ./kpax_results/d1bhga1
Writing MSA results with pivot = d1bhga1 to FOLDER = ./kpax_results/d1bhga1/
Writing Kpax multiple alignment file:
./kpax_results/d1bhga1/d1bhga1_multi_flex.kmsa
Writing FASTA multiple alignment file:
./kpax_results/d1bhga1/d1bhga1_multi_flex.fasta
Writing PIR multiple alignment file: ./kpax_results/d1bhga1/d1bhga1_multi_flex.pir
Done single pivot MSA for 3 targets in 0.0880 seconds.
Best MSA has pivot = d1bhga1, M=0.6647, C=65, P=1.8946

================================================================================

d1bhga1 A( 226) ------TYIDDI--T---VTTSVEQ-----DSGLVNYQISVKGS-NLFKLEVRLLD-A-ENKVVANGTGTQGQLKVP---
3ullA00 A( 10)  LERSLNRVHLLGRVG---QDPVLRQVEGKNPVTIFSLA-----------TNEMWRS------DVSQKT-TWHRISVFRPG
d1cs6a3 A( 209) --RQYAPSIKAK--FPADTYALTG--------QMVTLECFAFGNPV-PQIKWRKLDGSQTSKWLS-S-EPLLH-I-Q---

--------------------------------------------------------------------------------
Consensus-AA  -(------) r i q v l g r ld k vs t t v
Consensus-SSE -(------) CBBBBBCBBCCCCCCCCCBBBBBCBCCCCCCBCBBBBBBBBCCCCCCBBBBBBBCCCCCCCBCCCBBBCCCBBBBBCCCA
Colour-Codes  -(------) ROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVRO
--------------------------------------------------------------------------------

d1bhga1 A( 284) ------GVSLWWPYLMHERPAYLYSLEV---QLTAQTSLGPVSDFY-TL-PVGIRT
3ullA00 A( 78)  LRDVAYQYVKK---------GSRIYLEGKIDYGEYMDKNNVRRQATTIIADNIIFL
d1cs6a3 A( 269) --------NVDF-------EDE-GTYEC---EAENI----KGRDTY-QG-RIIIHA

--------------------------------------------------------------------------------
Consensus-AA  -(------) lE e rd y iI
Consensus-SSE -(------) AAAAAAABCCCCCCCCCCCCCCBBBBBBCCBBBBBBCCCCCBBBBBCBBCBBBBBB
Colour-Codes  -(------) YGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYGBVROYG
--------------------------------------------------------------------------------

================================================================================

Peak memory allocation: 13 Mb.
Total memory on exit: 0 Mb.
Kpax finished in a total of 0.15 seconds (0.00 minutes).
```

7. Scoring Pair-wise and Multiple Alignments

As shown above, Kpax automatically scores its own pair-wise and multiple alignments using its "M-score" scoring function. Additionally, Kpax can score pair-wise or multiple alignments

that have been calculated by other structure alignment programs, provided that they follow some simple file format conventions. In particular, if an alignment is written in a FASTA-like or PIR-like sequence alignment format, in which the name of the PDB coordinate file is written as the first item in the sequence header line, then Kpax should be able to score the alignment using the "-score" command line option. For example, the above multiple alignment calculation produced a FASTA output file called "1bhga1_multi_flex.fasta" along with three PDB files corresponding to the pivot and the two flexibly superposed structures. Here is is the content of the file "1bhga1_multi_flex.fasta":

```
>d1bhga1_flex A
------TYIDDI--T---VTTSVEQ-----DSGLVNYQISVKGS-NLFKLEVRLLD-A-ENKVVANGTGTQGQLKVP---
------GVSLWWPYLMHERPAYLYSLEV---QLTAQTSLGPVSDFY-TL-PVGIRT

>3ullA00_flex A
LERSLNRVHLLGRVG---QDPVLRQVEGKNPVTIFSLA-----------TNEMWRS------DVSQKT-TWHRISVFRPG
LRDVAYQYVKK---------GSRIYLEGKIDYGEYMDKNNVRRQATTIIADNIIFL

>d1cs6a3_flex A
--RQYAPSIKAK--FPADTYALTG--------QMVTLECFAFGNPV-PQIKWRKLDGSQTSKWLS-S-EPLLH-I-Q---
--------NVDF-------EDE-GTYEC---EAENI----KGRDTY-QG-RIIIHA
```

To use Kpax to score this alignment directly from the provided data, use:

```
kpax -score 1bhga1_multi_flex.fasta
```

This will produce the following output:

```
Kpax 5.0.0 starting at Wed Sep 16 16:31:04 2015 on host hardy.
Creating RESULTS directory: ./kpax_results/
Using LOG file: ./kpax_results/kpax.log
Loaded structure d1bhga1_flex from PDB file d1bhga1_flex.pdb
Loaded structure 3ullA00_flex from PDB file 3ullA00_flex.pdb
Loaded structure d1cs6a3_flex from PDB file d1cs6a3_flex.pdb

Making MSA data structure for 3 structures and 136 columns...
Assuming PDB structures are already superposed...

Cross-check MSA with PDB structures.

d1bhga1_flex ZERO differences
3ullA00_flex ZERO differences
d1cs6a3_flex ZERO differences
Total differences = ZERO

Setting CORE criterion = 2 rows (including pivot)

Analysing MSA (3 structures, 136 columns)...

No. of EMPTY columns (N==0) is 0 (0/136 = 0.0%)
No. of SINGLE columns (N==1) is 37 (37/136 = 27.2%)
No. of PARTIAL columns (N>=2) is 99 (99/136 = 72.8%); RMSD = 2.17
No. of FULL columns (N==3) is 65 (65/136 = 47.8%); RMSD = 1.98

No. of CORE columns (N>=2) is 99 (99/136 = 72.8%); RMSD = 2.17

No. of IDENTITIES with PIVOT structure: 17 (17/103 = 16.5%)

Total Residues (T) = 300
Longest Chain (L) = 106

M-score (ALL columns) = 0.66472

Writing Kpax multiple alignment file: ./kpax_results/d1bhga1_flex_msa.kmsa
Writing FASTA multiple alignment file: ./kpax_results/d1bhga1_flex_msa.fasta
```

```
Writing PIR multiple alignment file: ./kpax_results/d1bhga1_flex_msa.pir
Writing PIVOT d1bhga1_flex to PDB file: ./kpax_results/d1bhga1_flex_pivot.pdb
Writing TARGET 3ullA00_flex to PDB file:
./kpax_results/3ullA00_flex_d1bhga1_flex.pdb
Writing TARGET d1cs6a3_flex to PDB file:
./kpax_results/d1cs6a3_flex_d1bhga1_flex.pdb
Writing MSA summary file: ./kpax_results/d1bhga1_flex.ksum
Peak memory allocation: 1 Mb.
Total memory on exit: 0 Mb.
Kpax finished in a total of 0.08 seconds (0.00 minutes).
```

Note: the "-score" option causes Kpax to write out additional PDB files and command scripts for visualisation in Hex and VMD. Thus, Kpax may also be used to visualise the alignments produced by other alignment programs.

Kpax Databases

1. Creating and Searching a User Database
2. Deleting a Database
3. Creating a CATH Database
4. Creating a SCOP Database
5. Database ROC Performance

1. Creating and Searching a User Database

While it is convenient to compare a few structures directly on the command line, this becomes impractical if you want to search a database of hundreds or even thousands of structures. Hence, *Kpax* provides some simple but powerful options to create and search structural databases. For example, you can build a database called "test" and add the four example structures to it using this command:

```
kpax -build=test d1bhga1.ent d1cs6a3.ent 1qb5D00 3ullA00
```

You can then search the database using a command such as:

```
kpax -db=test d1bhga1.ent
```

This will show something like (truncated output):

```
Kpax 5.0.0 starting at Mon Sep 21 16:05:45 2015 on host hardy.
Using RESULTS directory: ./kpax_results/
Using LOG file: ./kpax_results/kpax.log
Using DATABASE directory: /home/ritchied/programs/kpax_database/test_kpax/
Using RESULTS directory: ./kpax_results/d1bhga1
Done 4 alignments for 4 targets in 0.0066 seconds (611/s).
Done 4 superpositions in 0.0543 seconds (74/s).
Writing QUERY pdb file: ./kpax_results/d1bhga1/d1bhga1_query.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/d1bhga1_d1bhga1.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/d1cs6a3_d1bhga1.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/3ullA00_d1bhga1.pdb
Writing TARGET pdb file: ./kpax_results/d1bhga1/1qb5D00_d1bhga1.pdb

================================================================================================
Top 4 matches for d1bhga1 (length 103) [0.0.0.0] -> [0.0.0.0] (2/4) ROC AUC = 1.000000
================================================================================================
Rank K-Score G-Score J-Score H-Score M-Score T-Score RMSD N/* P/$ I/@ I/% Len Seg TP Match
[Family]
===== ======= ======= ======= ======= ======= ======= ==== === === === === === === == ========
1 103.00 103.00 1.0000 1.0000 1.0000 1.0000 0.00 103 103 103 100.0 103 1 +1 d1bhga1[0.0.0.0]
2  39.99  43.27 0.4130 0.4469 0.5538 0.6255 3.07  79  82  10  12.7  91 1 +1 d1cs6a3[0.0.0.0]
```

```
3 36.62 25.31 0.3505 0.2422 0.2812 0.3243 3.15 45 81 3 6.7 106 1 0 3ullA00[2.40.50.140]
4 29.33 10.78 0.2904 0.1068 0.1381 0.1894 4.47 33 68 1 3.0  99 1 0 1qb5D00[2.40.50.50]
=============================================================================================
Mean 52.23 45.59 0.5135 0.4490 0.4933 0.5348 2.67 65 83 29 30.6 99 1 2
=============================================================================================
```

Note that because the database includes a copy of the query structure, this structure is ranked first as a perfect match in the results table. Here, the 4-digit CATH codes are all zero because we did not give sufficient information to be able to assign proper code numbers when creating the database.

By default, all databases are created under a directory called $KPAX_ROOT/kpax_database/. You can change this behaviour by setting the KPAX_DATABASE variable to the name of a different directory. You can list the contents of the database using:

```
kpax -list -db=test
```

This will show something like:

```
Selecting [*] from /home/ritchied/programs/kpax/kpax_database/test_kpax/
-------------------------------------------------------------------
d1bhga1 [0.0.0.0] S=1, CA=103, R=30.60
d1cs6a3 [0.0.0.0] S=1, CA=91, R=29.88
1qb5D00 [0.0.0.0] S=1, CA=99, R=24.48
3ullA00 [0.0.0.0] S=1, CA=106, R=31.25

Found 4 structures in /home/ritchied/programs/kpax/kpax_database/test_kpax/
-------------------------------------------------------------------
```

This shows the physical location of the database, and it confirms that the database contains the four structures, as expected. The output also shows the number of alpha carbons and the average radius of each domain. Note that because *Kpax* excludes residues with missing backbone atoms, the number of CA atoms may be less than the number in the original PDB file.

2. Deleting a Database

Currently, there is no built-in way to delete a database. Instead, you will need to explicitly delete the directory and its contents from the shell command line using the "rm" command (use with care!). For example, to delete the above test database, use something like:

```
cd $KPAX_ROOT
rm -rf ./kpax_database/test_kpax/
```

3. Creating a CATH Database

In order to import a set of CATH domains into a *Kpax* database and to assign the correct CATH code to each structure, it is necessary to provide the name of the folder containing the chopped CATH PDB files, and an index file with gives the mapping between each structure file and its 4-digit CATH code. For CATH, the index file is called "cath-domain-list-<S35%|S60|S95|S100|all>-<version>.txt ", and is available among the files that can be downloaded from CATH FTPsite here. To work with *Kpax*, this file should be copied to the KPAX_DATABASE directory. You can obtain a set of CATH domain structures as a file named " cath-dataset-nonredundant-S[20|40].pdb.tgz" that can be downloaded from CATH

FTP site here. Assuming you have extracted the CATH structure files into a directory called /cath/dompdb, you could then import them into *Kpax* using the following command:

```
kpax -build=cath -source=/cath/dompdb
```

A shell script containing the commands to build a fresh CATH database is provided in $KPAX_ROOT/build/build_cath.

4. Creating a SCOP Database

A SCOP database may be built in a similar way to a CATH database, as described above. However, in order to take into account the different directory structure used by SCOP, a shell script called $KPAX_ROOT/build/build_scop has been provided to simplify the procedure. Please view that script for further details.

5. Database ROC Performance

By construction, the *Kpax* scoring function guarantees to retrieve as the first match any database structure which is identical to the query structure. Therefore, by counting the numbers of true and false positives with respect to the CATH (or SCOP) code of the first match, Kpax can automatically calculate a receiver-operator-characteristic (ROC) performance curve for each query. If multiple query structures are given in a single run, *Kpax* also calculates an aggregate ROC curve by summing the individual query ROC curves, and it calculates the individual and aggregate area under the curve (AUC) in order to give a single numerical measure of overall performance. This is shown as the "AUC" value in the header line of the results table.

The individual ROC curves are written to files named $KPAX_RESULTS/<query>/<query>.kroc. The aggregate ROC curve and AUC result files are written to $KPAX_RESULTS/aggregate.kroc and $KPAX_RESULTS/aggregate.kauc, respectively. If desired, the ROC calculation and output may be suppressed using the "-noroc" option.

Kpax Results Files

1. Controlling the PDB Output

By default, for each superposed structure, *Kpax* writes out a separate PDB file containing all of the atoms from the original input file. If desired, all structures can be written to a single multi-model PDB file using the "-unified" option.

```
kpax -unified ...
```

It is also possible write out the coordinates of only the aligned residues, or in other words to "mask" out the unaligned residues:

```
kpax -mask ...
Equally, Kpax can mask out the aligned residues, in order to show only the
unaligned regions of a structure:
kpax -unmask ...
```

Finally, to save space and for greater speed, it is possible to suppress writing PDB files altogether:

```
kpax -nopdb ...
```

2. Additional Results Files

As well as containing the query and superposed target query structures, the *Kpax* results directory also contains several further results files. Assuming you have superposed two example structures

```
kpax d1bhga1.ent d1cs6a3.ent
```

you will find the following files in the ./kpax_results/d1bhga1/ results directory:

- d1cs6a3_d1bhga1.fasta – the aligned residues in FASTA format

```
>d1bhga1_query.pdb A
----TYIDDITV---TTSVEQDSGLVNYQISVKGSNLFKLEVRLLDAENKV----VANGTGTQGQLK-VPGVSLWWPYLM
HERPAYLYSLEVQLTAQTSLGPVSDFYTLPVGIRT

>d1cs6a3_d1bhga1.pdb A
RQYAPSIKAKFPADTYALTG---QMVTLECFAFGNPVPQIKWRKL---DGSQTSK-WLSSEPLLHIQNV-DFED------
------EGTYECEAENI----KGRDTYQGRIIIHA
```

- d1cs6a3_d1bhga1.kalign – Kpax alignment summary with SSEs

```
#=======================================================================================
#Query In: d1bhga1
#Target In: d1cs6a3
#Query chain: A
#Target chain: A
#Query Out: d1bhga1_query.pdb
#Target Out: d1cs6a3_d1bhga1.pdb
#Kscore: 39.99
#Gscore: 43.27
#Jscore: 0.4130
#Hscore: 0.4469
#Mscore: 0.5538
#Tscore: 0.6255
#Nquery: 103
#Ntarget: 91
#Nsegments: 1
#Ncolumns: 115
#Ncover: 111
#Naligned: 79
#Nmatched: 79
#Nanchor: 50
#Nidentity: 10 (12.66%)
#RMSD-aligned: 3.07
#RMSD-matched: 3.07
#RMSD-anchor: 1.72
#MaxDist: 7.52
#Contact: 8.00
#Alignment (X + Gap): 79 + 36 = 115
#Identities (@ matched + # unmatched): 10 + 0 = 10
#Others (* matched + $ unmatched): 69 + 0 = 69
#SSEs (A = ALPHA, B = BETA, C = COIL/LOOP)
#=======================================================================================
```

```
----TYIDDITV---TTSVEQDSGLVNYQISVKGSNLFKLEVRLLDAENKV----VANGTGTQGQLK-VPGVSLWWPYLM
RQYAPSIKAKFPADTYALTG---QMVTLECFAFGNPVPQIKWRKL---DGSQTSK-WLSSEPLLHIQNV-DFED------
**@***** ***** **@*******@********@*@ *** *********** @ ****

HERPAYLYSLEVQLTAQTSLGPVSDFYTLPVGIRT
------EGTYECEAENI----KGRDTYQGRIIIHA
****@****** ***@*@*****@**


#===================================================================================

----ccccccbbb---bbbbccccccbbbbbbbbbccccccbbbbbbbbbccccccc----ccbbccccccbb-bccbbccccccc
bbbccbbccccccccbbbccc---cbbbbbbbbbccccccbbbbbcc---ccccccb-bccccccbbbccc-cccc------
**@***** ***** **@*******@********@*@ *** *********** @ ****

ccccccbbbbbbbbbbbbccccccbbbbbbbbbbbbbb
------cccbbbbbbbcc----ccbbbbbbbbbbbbb
****@****** ***@*@*****@**


#===================================================================================
```

- d1cs6a3_d1bhga1.kpairs – Kpax alignment summary showing residue distances

```
#====================================================================================
#Query In: d1bhga1
#Target In: d1cs6a3
#Query chain: A
#Target chain: A
#Query Out: d1bhga1_query.pdb
#Target Out: d1cs6a3_d1bhga1.pdb
#Kscore: 39.99
#Gscore: 43.27
#Jscore: 0.4130
#Hscore: 0.4469
#Mscore: 0.5538
#Tscore: 0.6255
#Nquery: 103
#Ntarget: 91
#Nsegments: 1
#Ncolumns: 115
#Ncover: 111
#Naligned: 79
#Nmatched: 79
#Nanchor: 50
#Nidentity: 10 (12.66%)
#RMSD-aligned: 3.07
#RMSD-matched: 3.07
#RMSD-anchor: 1.72
#MaxDist: 7.52
#Contact: 8.00
#Alignment (X + Gap): 79 + 36 = 115
#Identities (@ matched + # unmatched): 10 + 0 = 10
#Others (* matched + $ unmatched): 69 + 0 = 69
#Tmatrix (Target Out = Tmatrix * Target In):
# -0.912632  0.407884 -0.027094 71.614277
# -0.395763 -0.865025  0.308388 72.981215
#  0.102350  0.292167  0.950875 50.880029
#  0.000000  0.000000  0.000000  1.000000
#====================================================================================
# Posn Dist K-Score G-Score SegID Query Target
#====================================================================================
1 -1.00 0.0000 0.0000 0 - A:209:ARG -
2 -1.00 0.0000 0.0000 0 - A:210:GLN -
3 -1.00 0.0000 0.0000 0 - A:211:TYR -
4 -1.00 0.0000 0.0000 0 - A:212:ALA -
5  2.09 0.1617 0.5716 1 A:226:THR A:213:PRO *
6  0.97 0.5132 0.8877 1 A:227:TYR A:214:SER *
7  1.27 0.6448 0.8138 1 A:228:ILE A:215:ILE @
8  1.26 0.5421 0.8163 1 A:229:ASP A:216:LYS *
9  2.23 0.2957 0.5306 1 A:230:ASP A:217:ALA *
10 1.57 0.4035 0.7315 1 A:231:ILE A:218:LYS *
```

```
11 2.20 0.1691 0.5382 1 A:232:THR A:219:PHE *
12 5.88 0.0778 0.0121 0 A:233:VAL A:220:PRO *
13 -1.00 0.0000 0.0000 0 - A:221:ALA -
14 -1.00 0.0000 0.0000 0 - A:222:ASP -
15 -1.00 0.0000 0.0000 0 - A:223:THR -
16  7.52 0.6211 0.0007 0 A:234:THR A:224:TYR *
17  4.01 0.3814 0.1287 0 A:235:THR A:225:ALA *
18  4.45 0.2192 0.0798 0 A:236:SER A:226:LEU *
19  2.80 0.1589 0.3679 0 A:237:VAL A:227:THR *
20  2.04 0.0154 0.5884 0 A:238:GLU A:228:GLY *
21 -1.00 0.0000 0.0000 0 A:239:GLN - -
22 -1.00 0.0000 0.0000 0 A:240:ASP - -
23 -1.00 0.0000 0.0000 0 A:241:SER - -
24  5.77 0.1646 0.0143 0 A:242:GLY A:229:GLN *
...
(truncated)
```

The results directory will also contain the following files (contents not listed here):

- d1cs6a3_d1bhga1.profit – to align the structures using Profit
- d1cs6a3_d1bhga1.wpairs – list of aligned residue numbers for KBDOCK

When searching a database or a list of target structures, the following query-specific files summarise the results in various ways:

- d1bhga1.ktops – summary of top hits (like the printed terminal output table)
- d1bhga1.khits – ranked list of hits, similarity scores, and CATH codes
- d1bhga1.krank – even simpler ranked list of structures and scores
- d1bhga1.kcath – summary of top hits by frequency of CATH family
- d1bhga1.kroc – x-y coordinates for plotting a ROC curve

Two files which summarise the aggregate ROC plot results are written to the $KPAX_RESULTS directory:

- aggregate.kroc – x-y coordinates for the aggregate ROC curve
- aggregate.kauc – summary list of individual and aggregate ROC curve AUCs

Several files are written to provide easy visualisation of the superposition in Jmol, Hex, and VMD.

- d1cs6a3_d1bhga1.jmol – list of aligned residue numbers for Jmol
- d1cs6a3_d1bhga1_aligned.mac – scripts to draw the superposition using *Hex*
- d1cs6a3_d1bhga1_matched.mac
- d1cs6a3_d1bhga1_segments.mac
- d1cs6a3_d1bhga1_rainbow.mac
- d1cs6a3_d1bhga1_aligned.tcl – scripts to draw the superposition using VMD
- d1cs6a3_d1bhga1_matched.tcl
- d1cs6a3_d1bhga1_segments.tcl
- d1cs6a3_d1bhga1_rainbow.tcl

If desired, all of the above files may be suppressed using the "-nowrite" option. You can then enable individual output files using command-line keywords. For example:

```
kpax -nowrite -pdb -hex -kalign ...
```

3. Controlling the Amount of Results

It is worth remembering that *Kpax* initially scores all structures using only the K-score without performing any superpositions, and it ranks them according to their J-score (normalised K-score). It then superposes only the top-scoring 60 matches. When comparing multiple structures, it is usually only of interest to look at the first handful of matches. Hence, by default, *Kpax* shows only the top 40 matches. You can change these parameter using the -show and -top options. For example, the following command will use the J-score to rank the

similarity of all structures in the CATH database, and then superpose and list the top 200 matches also ordered by J-score similarity:

```
kpax -top=200 -show=200 -db=cath -sort=J d1bhga1.ent ...
```

Parameters and Options

1. Alignment and Superposition Parameters

Kpax has several parameters which control its structural alignment and superposition calculations. There is normally no need to change these.

The following command shows the names of the alignment score weight parameters and their default values:

```
kpax -trace=0.5 -spatial=0.5 -blosum=0.0 -window=3 ...
```

Here, "-trace" is the local alignment score over a window of +/- 3 residues from the residue of interest; "-spatial" is the corresponding spatial similarity score calculated using the centre of mass of the protein, and "-blosum" is the Blosum62 amino acid similarity score. Please see the Bioinformatics paper for further details.

The next command shows the secondary structure-specific gap penalty factors:

```
kpax -rho=0.1 -eta=0.0 -alpha=2.0 -beta=1.0 -gamma=0.5 ...
```

Here, "-rho" is the basic penalty unit (leave at 0.1), and "-eta" is the gap extension penalty (leave at zero); "-alpha", "-beta", and "-gamma" are the penalty factors (in units of rho) for opening a gap in an alpha-helix, beta-sheet, or coil region, respectively.

The next command shows the Gaussian superposition parameters:

```
kpax -seeds -cycle=1 -contact=8.0 ...
```

Here, "-seeds" means use contiguous fragments of the initial structural alignment as superposition fitting seeds ("-noseeds" means use the whole alignment as the seed); "-cycle" is the number of least-squares fitting cycles to apply, and "-contact" is the alpha carbon distance threshold to use (pairs of CA atoms with distances greater than this threshold are excluded from the superposition calculation).

2. More Options

To see a list and a brief description of all command line options, please use:

```
kpax -help
```

References

Article on multiple flexible structure alignments using *Kpax: Calculating and scoring high quality multiple flexible protein structure alignments*. D.W. Ritchie (2016) Bioinformatics, 32(17) 2650-2658, and Supplementary Material.

Article describing the original *Kpax* pair-wise structure alignment algorithm: *Fast Protein Structure Alignment using Gaussian Overlap Scoring of Backbone Peptide Fragment Similarity*. D.W. Ritchie, A.W. Ghoorah, L. Mavridis, V. Venkatraman (2012). Bioinformatics, 28(24) 3274-3281, and Supplementary Material.

Contact

Sjoerd de Vries
SISR LORIA
Campus Scientifique
BP239
54500 Vandoeuvre-lès-Nancy
sjoerd.devries AT loria.fr