Burst After Duplication with Ancestral Sequence Prediction

Richard J. Edwards (2005)

1: Introduction

Software manuals are boring: boring to write and probably even more boring to read. I have therefore tried to keep this one concise. However, given (a) my propensity to waffle, (b) the fact that I am a biologist and not a computer scientist, and (c) my lack of previous experience in writing manuals, there is a good chance that the pleiotropic affect of this is a lack of clarity and/or coherence. For this I apologise, and encourage anyone out there to send in errata and/or suggested improvements. The fundamentals should be covered under the sections **Running BADASP**, **Input**, and **Output**. Gluttons for punishment can get more details in the **Appendices**.

Like the software itself, this manual is a 'work in progress' to some degree. If the version you are now reading does not make sense, then it may be worth checking the website to see if a more recent version is available, as indicated by the **Version** section of the manual. Good luck.

Rich Edwards, 2005.

1.1: Version

This manual is designed to accompany BADASP version 1.1.

The manual was last edited on 16 November 2005.

1.2: Copyright, License and Warranty

1.3: Using this Manual

As much as possible, I shall try to make a clear distinction between explanatory text (this) and text to be typed at the command-prompt etc. Command prompt text will be `written in Courier New` to make the distinction clearer. Program options, also called 'command-line parameters', will be **`written in bold Courier New`** (and coloured **`red`** for fixed portions or **`dark red`** for user-defined portions, such as file names etc.). Command-line examples will be given in (purple) *`italicised Courier New`*. Optional parameters will (where I remember) be `[`in square brackets`]`. Names of files supplied with BADASP will be marked in text by (dark yellow) **Bold Times New Roman**.

1.4: Getting Help

Much of the information here is also contained in the BADASP website (http://www.bioinformatics.rcsi.ie/~redwards/badasp/) and the documentation of the Python modules themselves. A full list of command-line parameters can be printed to screen using the **help** option, with short descriptions for each one.

```
python badasp.py help
```

If none of the above are of help, then please e-mail me (redwards@rcsi.ie) whatever question you have. If it is the results of an error message, then please send me that and/or the log file (see **2.3: Output**) too. Usually, it will be a problem with the input files (possibly formatting) but there are possibly still a few bugs in there somewhere too.

1.5: Why use BADASP?

The divergence of proteins following gene duplication has long been recognized as an important process in the evolution of new or specific protein functions. Functional divergence is proposed to occur through some combination of neofunctionalisation – the evolution of novel gene function – and subfunctionalisation – the partitioning of two or more existing gene functions between paralogues (genes related by duplication) (Zhang 2003). Although no consensus has yet been reached as to which process is more important, the distinction is somewhat irrelevant for the bioinformatic prediction of sites important for differences in gene function between paralogues (though it is vitally important for the interpretation of results). Instead, it is more pertinent to consider the types of substitution that occur at these sites and the phylogenetic signal that they leave.

Sites of functional change following duplication can be broadly classified into two categories, which Gu has named Type I and Type II (Gu 2001). Type I functional divergence shows a change in selective constraint on a site following duplication, either by relaxation of existing purifying selection or by gaining functional importance at a previously unimportant site. In contrast, sites experiencing Type II divergence remain important in both duplicates but a different amino acid is favoured in each duplicate. Both Type I and Type II divergence can occur as the result of either neo- or subfunctionalisation. For example, subfunctionalisation may occur by partitioning domain functions, with different domains maintained in different paralogues (Type I divergence), or by each paralogue specializing for a given set of existing substrates (Type II divergence). Similarly, new gene function may arise at previously unimportant sites (Type I) or by recruiting existing functional sites to the new function (Type II), while the paralogue fulfills the previous roll of the ancestral protein.

Several methods now exist to detect either Type I or Type II divergence (Lichtarge et al. 1996, Caffrey et al. 2000, Hannenhalli and Russell 2000, Johnson and Church 2000, Gu 2001, del Sol Mesa et al. 2003, Kalinina et al. 2004a). Many of these, however, are complex methods that lack simple software implementations and/or rely on additional information, such as structural data, which is not always available. Although there are available tools for state of the art predictions for divergence of either Type I (Gu and Vander Velden 2002) or Type II (Kalinina et al. 2004b) for single protein families, there is still the need for a simple analysis package that can be run from the command line for multiple families and can potentially detect both Type I and Type II divergence. BADASP provides software to implement the previously published Burst After Duplication (BAD) algorithm (Caffrey et al. 2000), plus two variants that have been used successfully in identifying functionally interesting sites in platelet signalling proteins (Edwards et al. submitted) and can identify Type I and Type II divergence.

Because it is open source and written in a readily (and freely) available programming language, there is plenty of scope to add additional methods for detecting specificity and/or

measure conservation, while the simple text output makes comparisons in a number of ways possible using a variety of easily available tools.

1.6: Installation

BADASP is distributed as a number of open source Python modules. It should therefore work on any system with Python installed without any extra setup required – simply copy the relevant files to your computer and run the program (see **2.1: Running BADASP**, below.)

If you do not have Python, you can download it free from www.python.org at http://www.python.org/download/. The modules are written in Python 2.4. The Python website has good information about how to download and install Python but if you have any problems, please get in touch and I will help if I can.

1.6.1: Files Required for BADASP

The following files are required for BADASP to run correctly. All these files should have been provided in the download zip file. The Python Modules are open source and may be changed if desired, although please give me credit for any useful bits you pillage. I cannot accept any responsibility if you make changes and BADASP stops working, however! The additional files may all be replaced with other files in the correct format. These files are described later in this manual and/or in the Appendix.

Python Modules: **badasp.py, rje.py, rje_aaprop.py, rje_ancseq.py, rje_blast.py, rje_conseq.py, rje_pam.py, rje_seq.py, rje_sequence.py, rje_specificity.py, rje_tree.py, rje_tree_group.py, rje_uniprot.py**

Additional Files: **jones.pam, aaprop.txt**

2: Fundamentals

2.1: Running BADASP

2.1.1: The Basics

If you have python installed on your system (see **1.6: Installation**), you should be able to run BADASP directly from the command line in the form:

```
python badasp.py seqin=FILENAME
```

For the example provided in the distribution:

```
python badasp.py seqin=badasp_eg.fas
```

If the tree file is named after the sequence file (with a *.nsf extension) as in the example then this alone should be sufficient. Otherwise, use the **nsfin=TREEFILE** option:

```
python badasp.py seqin=fasta_file nsfin=nsf_file
```

```
python badasp.py seqin=badasp_eg.fas nsfin=badasp_eg.nsf
```

If running in Windows, you can just double-click the badasp.py file and use the menu prompts to navigate through the program.

Note: If filenames contain spaces, they must be enclosed in double quotes, e.g.

```
python badasp.py seqin="badasp example with spaces.fas"
```

2.1.2: Interactivity and Verbosity settings

By default, BADASP will run through to completion without any user-interaction if given all the options it needs. For more interaction with the program as it runs, use the argument '**i=1**'

```
python badasp.py seqin=gasp_eg.fas i=1
```

Both the level of interactivity and the amount printed to screen can be altered, using the interactivity [**i=X**] and verbosity [**v=X**] command-line options, respectively, where **X** is the level from none (-1) to lots (2+). Although in theory **i=-1** and **v=-1** will ask for nothing and show nothing, there is a good chance that some print statements will have escaped in these early versions of the program. Please report any irritations.

2.1.3: Other Options

At first, you will probably want to run BADASP with its default parameters. If you want to change them, there are a number of parameters that can be set by the user and other options. These are described in **4.2: Appendix II: Command-line Options**. These may be given after the run command, as above, or loaded from a ***.ini** file (see for **4.2.3: INI Files** details).

2.2: Input

BADASP uses input from three sources:

1. a multiple sequence alignment (MSA) [**seqin=FILE**]
2. an accompanying rooted phylogenetic tree in Newick (Phylip) format [**nsfin=FILE**]
3. a Point Altered Mutation (PAM) substitution probability matrix [**pamfile=FILE**]

Sequences are read in from the alignment and node relationships established from the tree. The tree may be already rooted or rooted using BADASP. Input trees must have branch lengths. Bootstrap values are not used by BADASP but will be read if present. Sequences in the tree file can be represented by numbers (matching the order of the fasta alignment) or the first word of the sequence name.

In addition, files can be given to bypass certain parts of BADASP:

4. grouping of input sequences [**group=FILE**]
5. ancestral sequence file in GASP fasta format [**useanc=FILE**]

2.2.1: Input Alignment

The input alignment should be in FASTA format with descriptions on one line followed by one or more lines containing the sequence. All sequences should be of the same length. The first word in each description should be unique. *e.g.*

```
>Seq1 And its description
SEQUENCE-ONE-GOES-HERE
>Seq2
---GAPS--ARE--ALLOWED-
>Seq3
---BUT---ALL-SEQUENCES
>Seq4
MUST-BE-EQUAL--LENGTHS
```

Freely available programs such as BioEdit (Hall 2001) can be used to tidy up input alignments to the correct format.

2.2.2: Newick (Phylip) format tree

BADASP can accept a reasonably flexible format of Newick input tree in the general format:

`(Seq1:dis1,Seq2:dis2)Bootstrap:dis3` (formatting added for clarity)

Seq1 and **Seq2** are the sequence names and could be replaced by numbers corresponding to the order of the sequences in the alignment. dis1 and dis2 are the lengths of the branches to **Seq1** and **Seq2**. dis3 is the length of the branch linking the rest of the tree to the ancestral node (in the case of a rooted tree) of **Seq1** and **Seq2**. *Bootstrap* is the bootstrap value associated with the branch of length dis3. Bootstrap values are ignored by BADASP.

Example. For a tree with five sequences: (sequences not shown)

```
>Human
>Mouse
>Chicken
>Zebrafish
>Fugu
```

The following two rooted trees would be read the same:

```
((Zebrafish:0.151,Fugu:0.139)1000:0.162,(Chicken:0.255,(Human:0.092,
Mouse:0.101)1000:0.203)986:0.054):0.1;
```

```
((4:0.151,5:0.139):0.162,(3:0.255,(1:0.092,2:0.101):0.203):0.054):0.
1;
```

NB. To use a *.phb NSF tree generated by ClustalW, try using the [**phbin=FILE**] command.

Rooting. GASP needs a rooted tree for ancestral sequence prediction. However, the BADASP program will root unrooted input trees. This can either be done automatically by mid-point rooting [**root=mid**], by manually selecting the position of the root, or by feeding the program a list of 'outgroup' sequences that are to be in a single clade of the tree [**root=FILE**]. In the latter case, BADASP will place the root on the first branch that groups all outgroups into a single clade. (A random rooting option is also provided.) See **2.5.1: Tree Rooting** for details.)

More information on Newick Format can be found at the PHYLIP homepage and GASP website.

2.2.3: PAM Matrix

The PAM matrix contains amino acid substitution probabilities. A basic PAM matrix (Jones et al. 1992) is available with the BADASP program in the file jones.pam (and is known as JTT, I believe). The important part of this file is the top section, which has the single letter amino acid codes on the first line, followed by the PAM1 matrix, where each subsequent line consists of an amino acid code and the probability of that aa being substituted by each other aa, in the order given in the first line:

```
A   R   N   D   C   Q   E   G   H   I   L   K   M   F   P   S   T   W   Y   V
Ala 0.98754 0.00030 0.00023 0.00042 0.00011 0.00023 0.00065 …
Arg 0.00044 0.98974 0.00019 0.00008 0.00022 0.00125 0.00018 …
Asn 0.00042 0.00023 0.98720 0.00269 0.00007 0.00035 0.00036 …
…
Val 0.00226 0.00009 0.00007 0.00016 0.00012 0.00008 0.00027 …
```

Note that the amino acids must be in the same order in both columns and rows. See http://www.bioinformatics.rcsi.ie/~redwards/gasp/ for more details.

2.2.4: Sequence Group Data

By default, sequence groups are defined by the user. Grouping options can also be given on the command-line (see **4.2.5: Command-line Options**) including the **group=FILE** option, which load groups from a given file. The file is simple text file containing the unique sequence IDs for the defined groups:

```
Group 1: edg7
edg7_HUMAN/AAF91291
EDG7_RAT/Q8K5E0
EDG7_MOUSE/Q9EQ31

Group 2: edg2
EDG2_HUMAN/Q92633
edg2_CAVPO/AAK07693
EDG2_SHEEP/P46628
EDG2_BOVIN/Q28031
EDG2_MOUSE/Q61130
LP11_XENLA/Q9PU17
```

Group names (edg7 and edg2 in this example) are optional.

2.2.5: Ancestral Sequence Data

To save time if GASP (or BADASP) has already been run on the input sequence data, or to use ancestral sequences generated using an alternative method, ancestral sequences in GASP format can be given to BADASP using the **useanc=FILE** option. In this case the file given must be in fasta format with sequences numbered and nodes named X Node (A,B,C), where X is the number of the node itself and A, B and C are the numbers of the nodes directly linked to X. E.g.

```
>06 LP11_XENLA/Q9PU17 Lysophosphatidic acid receptor LPA1…
MASLSEFVSEPISMMSQTSAASESQCYYNETIAFFYNRSGKYLATEWN-AVSKLVMGLGITVC…
>16 Node 16 (6,14,17)
MA--AAFTSSPVISQPQFTAMNEQQCYYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>05 EDG2_MOUSE/Q61130 Lysophosphatidic acid receptor Edg-2…
MA--AASTSSPVISQPQFTAMNEQQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>14 Node 14 (5,12,16)
MA--AASTSSPVISQPQFTAMNEQQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>01 EDG2_HUMAN/Q92633 Lysophosphatidic acid receptor Edg-2…
MA--AISTSIPVISQPQFTAMNEPQCFYNESIAFFYNRSGKHLATEWN-TVSKLVMGLGITVC…
>11 Node 11 (1,2,12)
MA--AISTSIPVISQPQFTAMNEPQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>02 edg2_CAVPO/AAK07693 endothelial differentiation…
----------------------QCYYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>12 Node 12 (11,10,14)
MA--AASTSSPVISQPQFTAMNEPQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
… [truncated]
```

In this example, Node 14 is linked to sequence 5, (descendant) node 12 and (ancestral) node 16 as can be seen with the corresponding portion of the tree:

```
          +---+ 6: LP11_XENLA/Q9PU17 {0.060000}
          |
  #=======+ Node 16 (6,14,17) [1000] {0.130000}
  |       |
  |       | +-+ 5: EDG2_MOUSE/Q61130 {0.010000}
  |       | |
  |       +-+ Node 14 (5,12,16) [1000] {0.020000}
  |         |
  |         |   +-+ 1: EDG2_HUMAN/Q92633 {0.010000}
  |         |   |
  |         | +-+ Node 11 (1,2,12) [235] {0.010000}
  |         | | |
  |         | | +-+ 2: edg2_CAVPO/AAK07693 {0.010000}
  |         | |
  |         +-+ Node 12 (11,10,14) [441] {0.010000}
  |           |
[truncated]
```

Sequence and node numbering is flexible (i.e. need not match the input order of the sequences) as long as the sequence names match and the numbering is consistent with the given topology. Sequences and nodes may be renumbered by BADASP.

2.3: Output

The main output for BADASP is a delimited text file (***.badasp**) containing stats for each residue in the alignment and a log file (see below). In addition this file, there is the output from the Ancestral Sequence Prediction (see GASP Manual for details (Edwards 2004a)) and, if selected, a text file with the sequence groupings (see **2.2.4: Sequence Group Data**).

2.3.1: BADASP Results File

The main BADASP results file is a simple raw text file for easy importing into standard spreadsheet or database applications for analysis. The first two lines contain data on the BADASP run – the time and date, and the command-line parameters used. The rest of the file then contains the results. The statistics themselves are described in **3: BADASP Statistics**. Ways of reducing the output and extracting the key residues are given in **3.3: Reducing BADASP Output using Value/Rank filters** and **3.4: Manipulating and Visualising BADASP results**.

Alignment/Position Statistics. The first few columns of the output contain position and residue information:

- aln_pos = position in alignment
- anc_aa = predicted residue (aa) at root (ancestral) node
- qry_pos = position in query (if given)
- qry_aa = residue (aa) in query (if given)
- famX_pos = position in family (group) X ancestral node
- famX_aa = residue (aa) in family (group) X ancestral node

Specificity Statistics. These will depend on the options chosen. By default, all statistics will be calculated. The statistics themselves are described in 3: BADASP Statistics.

- bad = Burst After Duplication (BAD)
- badn = BADN variant of BAD
- badx = BADX variant of BAD
- ssc = Livingstone & Barton method (SSC)
- pdad = Property Difference After Duplication (PDAD) method
- eta = Basic Evolutionary Trace Analysis (ETA)
- etaq = Quantitative variant of ETA
- STAT_alnwin = Same as STAT, averaged over a window (**winsize=X**) as defined by the alignment, irrespective of column composition (gap or residue)
- STAT_qrywin = Same as STAT, averaged over a window (**winsize=X**) as defined by the query sequence, *i.e.* skipping gaps in the query
- STAT_famX_win= Same as STAT, averaged over a window (**winsize=X**) as defined by the predicted sequence for the ancestral node of the subfamily X, *i.e.* skipping gaps in the ancestral sequence
- STAT_rank = Same as STAT but converted into a rank number from 0 (lowest score) to 1 (highest score)
- STAT_rankwin = Ranks averaged over windows (**winsize=X**) as above

Conservation Statistics

- info = Information Content
- pcon_abs = Absolute number of properties conserved across all sequences
- qpcon_abs = Absolute number of properties conserved across query subfamily
- pcon_mean = Mean pairwise property difference for all sequences in the alignment
- qpcon_mean = Mean pairwise property difference for query subfamily
- STAT_alnwin, STAT_qrywin, STAT_rank, STAT_rankwin = as above

2.3.2: Sequence Group File

This is an optional output of BADASP and contains the sequence group data. It can be read back in using the **group=FILE** option to perform future analysis on the same dataset with the same grouping and is the same format as given in **2.2.4: Sequence Group Data**.

2.3.3: Log Files

The badasp log file records information that may help subsequent interpretation of results or identify problems. Probably it's most useful content is any error messages generated. By default the log file is badasp.log but this can be changed with the **log=FILE** option. Logs will be appended unless the **newlog** option is used. (See **4.2: Appendix II: Command-line Options** and Error! Reference source not found.Error! Reference source not found. for details.)

2.4: Pre-Treating Datasets to optimise for BADASP

It is assumed that in most cases, the user will have already generated an alignment and tree that they are happy with. Sometimes, however, additional manipulations of the dataset are necessary. BADASP facilitates some basic manipulations for trees and datasets including

removal of redundant sequences (sequences above a certain global % identity threshold), pruning of unwanted sequences/clades and tree re-rooting. Some of these are covered below. For discussion of all the options available with the Python modules included in BADASP, see the documentation for the associated application, HAQESAC (Edwards 2004b). HAQESAC (Homologue Alignment Quality, Establishment of Subfamilies and Ancestor Construction) is a tool designed for assembling datasets for the same kind as BADASP as is a good place to start if having trouble making a suitable dataset.

2.5: Sequence Grouping and Phylogeny Editing with BADASP

A fundamental need of functional specificity predictions is to establish subfamily relationships for the sequences. An important part of this procedure is the correct determination of representative sequences from species that have several possible sequences for a given paralogue. This is particularly common where multiple databases have been used to collect sequences and a given species may have, for example, a SwissProt entry and an EnsEMBL peptide, or two different EnsEMBL gene predictions.

Establishment of Subfamilies in BADASP is semi-automated in a three step process:

1. A (preferably bootstrapped) tree is loaded and rooted as determined by the user.
2. Sequences are grouped according to user parameters and/or manual decisions
3. Sequence groups are manually reviewed to remove unwanted sequence variants

NB. The following has been edited from the HAQESAC manual and some errors may remain where default settings etc. differ between the two programs. Please report any errors to the author.

2.5.1: Tree Rooting

BADASP provides five rooting options. By default, the existing rooting is used. Other options are to midpoint root the tree (**root=mid**), where the root is placed halfway between the most distant tips, manually placing the root using the Root Menu described below (**root=man**), randomly rooting with (**root=ranwt**) or without (**root=ran**) weighting for branch lengths (not recommended), or rooting from a file of outgroup sequences (**root=FILE**). If a file is given, BADASP will place the root on the branch that includes all sequences in the file in one half of the tree, while the other half contains as many sequences not in the file as possible.

Manual rooting gives a menu that, in addition to the above automatic rootings, has three manual rooting options:

```
<0> Keep current rooting.
---
<1> Midpoint Root.
<2> Root from Outgroup File.
<3> Make Outgroup File.
<4> Root on Branch.
<5> Manual rooting using text Tree (with editing options).
<6> Random Root.
<7> Random Root (branch-length weighted).
---
<8> Unroot.
```

These options are as follows:

0. Keep Current Rooting. Proceed onto next stage of ES with current rooting. (If tree is rooted.)
1. Midpoint Rooting. As **root=mid**, above.
2. Root From Outgroup. As **root=FILE**, above.
3. Make Outgroup File. Selects sequences to form outgroup file for future rooting.
4. Root on branch. This option goes through each branch, showing the two clades that the tree would be divided into if the branch was used for the root. Select <**R**> to root on that branch, <**N**> to move to the next branch, <**P**> to return to the previous branch, <**I**> to skip to internal branches, and <**Q**> to quit back to the Rooting Menu.
5. Manual rooting using Text Tree. This will show a text tree on screen with options to edit and/or root the tree (see **2.5.2: Editing the Tree**).
6. Random root. Places the root on a random branch with equal rooting probabilities for each branch (**root=ran**).
7. Random root (branch length weighted). Places the root on a random branch with rooting probabilities proportional to branch length (**root=ranwt**).
8. Unroot. Clears current rooting.

2.5.2: Editing the Tree

This shows a text representation of the tree and some editing options. Note that this can be unwieldy for large trees. When rooting the tree, the following options are given:

```
Display: <C>ollapse, <E>xpand, <F>lip, <N>ame Clade, <Z>oom,
<O>ptions, <D>escriptions, <S>ave to file, <Q>uit

Edit: <B>ranch, <T>erminal node, <P>rune, <R>oot on Branch,
<U>nroot, Root <M>enu
```

In addition to these options, you can enter a node number. This will Collapse or Expand that node as appropriate (see below).

Display Options:

Collapse. Collapse a clade to a single node on the displayed tree.

Expand. Reverses Collapse.

Flip. Vertically flips a clade around the chosen node.

Name Clade. Gives a name to a collapsed node.

Zoom. Zooms the displayed tree in on a particular clade. The chosen node becomes the 'root' for on screen display only. This is useful for large trees. Giving node '0' will zoom back out to the whole tree.

Options. Gives options for changing the way the tree is displayed.

Descriptions. Toggles whether description part of sequence name is displayed.

Save to file. Saves the current text tree to a text file.

Quit. Returns to Root Menu.

Edit Options:

`Branch`. Allows you to edit branch attriubutes (length, bootstrap etc.)

`Terminal node`. Allows you to edit sequence details (name, species etc.)

`Prune`. Permanently removes and node and its descendant clade.

`Root on Branch`. Roots the tree on the selected branch.

`Unroot`. Unroots the tree.

`Root Menu`. Return to the Root Menu.

2.5.3: Sequence Grouping Overview

The Sequence Grouping phase of the program has two goals:

1. Removal of sequence variants (if unwanted). These may be redundancies left by using multiple databases that have not been cleaned up, or unwanted splice variants or 'in-paralogs' (the results of genome-specific duplications).
2. Subdividing the sequences into groups for BADASP specificity analysis.

By default, the program aims to retain related subfamilies that have at least three members and a single representative sequence from each species. If this is not what you want, then you will need to change the settings (`mfs=X`, see **2.5.4: Important Grouping Options**).

2.5.4: Important Grouping Options

There are four main options that affect how grouping is directed:

1. **FAMILY SIZE.** (`mfs=X`) This is the minimum number of sequences there should be in each family. For specificity analysis, or solid evolutionary inference, this is probably 3 (the default). For a more relaxed conservation analysis, set `mfs=2`. `mfs=1` allows groups of '1' but this is really the same as the using the Orphans setting (below) and is therefore not recommended.
2. **FAMILY NUMBER.** (`fam=X`) This is the minimum number of families that BADASP will accept as OK. For specificity analysis, this is 2. By default, this is 0 and so no grouping is necessary (though it can still be useful to get rid of variants etc.).
3. **ORPHANS.** (`orphans=T/F`) Orphans are sequences not in any groups. For specificity and conservation within orthologues, these sequences are useless and can be dumped. For more general evolutionary studies, they may be important and can be retained (the default). If you trust the input sequences, then I would suggest keeping Orphans. If, however, there are a lot of potentially spurious sequences (e.g. EnsEMBL genscans) then orphans should probably be dumped as anything only present in one species is more likely to be noise. If in doubt, leave `orphans=T` and manually remove them at the group review stage (see **2.5.9: Reviewing Sequence Groups**).
4. **VARIANTS.** (`allowvar=T/F`) This option dictates whether groups may contain multiple sequences from the same family. If you are confident that the dataset you are searching contains no redundancy and you want to retain splice variants and in-paralogs then set `allowvar=T`. If, however, you are compiling a dataset from several complimentary databases and you only want the best representative sequence for each species, set `allowvar=F`.

2.5.5: Grouping Methods

In addition to the Grouping 'Rules' above, there are several different ways of grouping the sequences using the **group=X** option (or the Grouping Menu):

Method	Command	Description
Manual	**group=man**	This is done interactively with a tree, selecting the node number that is at the base of the group to group or ungroup sequences. See **2.5.7: Manual Group Editing/Selection** for details.
Duplication	**group=dup**	The program tries to automatically group sequences into orthologous groups according to the options that are set. By default, all duplication nodes will be considered. If the **groupspec=X** option is used then only duplications of the relevant species will be used. See **2.5.8: Duplication Grouping** for details.
Query	**group=qry**	Duplication Grouping only considering duplications of the Query sequence species
One Group	**group=one**	All sequences in a single group
No Groups	**group=None**	No sequences are grouped. (case sensitive)
Load Groups	**group=FILE**	Load groups from previously generated group **FILE**.

2.5.6: The Grouping Menu

The Grouping Menu allows interactive control and review of the sequence grouping procedure. The Menu first summarises the grouping 'Rules', which are set by command-line parameters:

```
#*# Grouping Options #*#
[Bootstrap cut-off: 800 (0.8)]          (bootcut=X)
[Min Family Size: 3]                    (mfs=X)
[Min Family Number: 0]                  (fam=X)
[Group by Query Species: False (None)] (groupspec=X or group=qry)
[Allow Sequence Variants: False]        (allowvar=T/F)
[Allow Orphan Sequences: True]          (orphans=T/F)
```

Next, a summary of the current grouping selection is given, with the number of groups and the number of orphan sequences.

```
#*# Grouping Summary #*#
Currently X groups. (Y Orphans)
```

If there are any groups, they will be listed along with their size and the sequences they contain. After these summaries is the menu itself:

```
<K>eep current grouping. (Current Grouping OK)
---
<O>ptions (Change Grouping 'rules').
---
<L>oad Grouping.
<D>uplication Grouping. (All Duplications)
<M>anual Grouping.
<N>o Grouping (MinFamNum=0).
```

```
<A>ll sequences in one group.
<E>dit Grouping (Manual)
---
<R>eview Groups
<G>roup in SeqList (Reorder)
<P>urge Orphans Sequences (not in a group)
<S>ave Groups
---
<Q>uit
```

Choice	Option	Description
K or ENTER	Keep Current Grouping	If the current grouping breaks any of the rules, then *** **WARNING: Current Grouping breaks 1+ rules.** *** will be displayed instead of **(Current Grouping OK)**. Breaking the rules includes: having insufficient groups (**fam=X**), having a group with insufficient sequences (**mfs=X**), having a group without sufficient bootstrap support (**bootcut=X**), having species variants in a group when **allowvar=F**, having orphan sequences when **orphans=F**. Groups may still be accepted when rules are broken. Upon keeping groups, you will be given the option to save the groups to a file (See Error! Reference source not found.**Output**.).
O	Options	This allows you to change the displayed Grouping Options ('rules'). If you change the options using the menu you will have to re-group the sequences for them to have an effect on automated grouping.
L	Load Grouping	Loads grouping from file. (See Error! Reference source not found.**Output**.)
D	Duplication Grouping	If **groupspec=X** is used, **(All Duplications)** will be replaced with **(X Duplications)**, where X is the species code. See **2.5.8: Duplication Grouping** for details.
M	Manual Grouping	Clears all groups and instigates manual grouping. See **2.5.7: Manual Group Editing/Selection** for details.
N	No Grouping	Clears all grouping.
A	All sequences in one group	Exactly.
E	Edit Grouping	This is like Manual Grouping except that the current groups are not cleared. This allows you to keep the ones you like and change the ones you don't. See **2.5.7: Manual Group Editing/Selection** for details.
R	Review Groups	See **2.5.9: Reviewing Sequence Groups**.
G	Group in SeqList	Reorders the sequences in the file to match the grouping, i.e. sequences in the same group will be put next to each other.
P	Purge Orphans Sequences	Deletes any sequences not in a group.

Choice	Option	Description
S	Save Groups	Saves groups to a file (see Error! Reference source not found.**Output**). Group names can be given if desired (see **2.5.7: Manual Group Editing/Selection**).
Q	Quit	Quit the Grouping Menu. Depending on the circumstances, this may behave like **Keep Current Grouping** or, if rules are broken, take you to the **Review Groups**.

2.5.7: Manual Group Editing / Selection

This is the essentially the same as the manual tree editing for rooting (see **2.5.2: Editing the Tree**) with two slight changes:

1. The Root options should not be given
2. `<C>ollapse` and `<E>xpand` have become `<C>ollapse/Group` and `<E>xpand/Ungroup`.

Now, collapsing a node will create a sequence group at that node, which can be named if desired. Expanding a node will remove the grouping at that node. Note that groups cannot overlap, so any groups formed at nodes deep in the tree will automatically remove any descendant groupings.

2.5.8: Duplication Grouping

Duplication grouping is an automated attempt to group sequences into groups of orthologous proteins, i.e. the same protein in different species. Duplication grouping is executed as follows:

1. Duplication nodes are identified using the species information of the input sequences. If both descendant branches from a node have descendant sequences of the same species, then that node is identified as a duplication node. If `groupspec=X` or `group=qry` are used, then only the relevant species are considered. Species themselves are gleaned from the description line of the input sequences. For easier visualisation for grouping, there is a sequence name reformatting option (see **2.5.10: The gnspacc=T/F Option**).
2. Each node is considered and assigned as a grouping node if:
    a. It is a duplication node
    b. The bootstrap value of the ancestral branch is high enough (`bootcut=X`)
    c. The number of descendant sequences is sufficient (`mfs=X`)
3. Each grouping node is reconsidered and removed if any of its descendants are also valid groups. This prevents an ancient duplication node wiping out all the more recent duplication groupings.

Note. The presence of sequence variants and branches with poor bootstrap support can lead to strange results from automated duplication grouping. It is therefore highly recommended to manually edit the groups the first time round.

2.5.9: Reviewing Sequence Groups

The Review Groups stage is broken down into stages:

1. The group is displayed as a fragment of a text tree. This is so that you can use phylogenetic relationships to guide your decisions regarding sequence inclusion or

exclusion. If you decide that the grouping is bad after all, then retain all variants and return to the Grouping Menu (see below).

2. A group 'master' sequence will be chosen. This is the Query sequence if one is given, or a sequence of the same species. If there is no query, or no sequence in the group from the same species, then the master will be selected as the sequence with the highest mean %ID (based on the multiple sequence alignment (MSA)) to the other sequences in the group. Each sequence is then summarised, relative to the Group Master for three criteria, which are displayed as percentage values with the actual number of amino acids in brackets afterwards:

```
SeqName: X.XX% ID (X aa); X.XX% Gaps (X aa); X.XX% Extra (X aa); vs
GroupMaster
```

   a. **% ID.** This is the percentage of the variant sequence that is identical (in the MSA) in the group 'master'.

   b. **% Gaps.** This is the percentage of the overlap between the two sequences that is a gap in the variant and a residue in the group master. (Gaps in both are ignored.)

   c. **% Extra.** This is the percentage of the variant that overlaps with a gap in the Group Master.

3. If the group master is itself a sequence variant, then the "best" sequence of that species must be selected as the group master. By default, BADASP will suggest the sequence with the most identical residues (X aa), as determined by the MSA, compared to the Query sequence. Selecting `0` or simply hitting `ENTER` will choose this sequence and the other variants will be removed. Alternatively, you can keep all sequence variants but this sequence will remain as group 'master' for the next stage.

4. Once the group 'master' is chosen, any other sequences from species with more than one variant in the group will be compared in a similar fashion. This time, however, BADASP will suggest the sequence with the highest identity to the group master. This is to maximise the chance of getting the same splice variant, for example, across different species. Again, selecting `0` or simply hitting `ENTER` will choose this sequence and the other variants will be removed. Alternatively, you can keep all sequence variants.

BADASP will suggest what it thinks is the best sequence based on certain rules but it is important at this stage to think also about what you are trying to achieve by removing variants of the same species and not to simply accept the suggested sequence without looking. You may choose to keep a SwissProt sequence, for example, even though it has slightly lower %ID to the group 'master' than a trEMBL or ensEMBL one.

Once all the variants are dealt with, there are a number of options presented:

```
Subfamily Options:

<P>revious, <T>ree Edit, <G>ene Edit, <U>ngroup, <D>elete Group,
Group <M>enu, <Q>uit Review, <N>ext, <F>inish Review
```

Note that not all of these options are displayed every time. (The `<P>revious` option for example, will only be there when there *is* a previous group.)

Choice	Option	Description
P	Previous	Return to previous group.
T	Tree Edit	Edit details of the group using the Tree Edit functions. See **2.5.2: Editing the Tree**.
G	Gene Edit	This allows you to give a new 'Gene' annotation to the sequences. See **2.5.10: The gnspacc=T/F Option** for details. This will not change the 'Gene' part of SwissProt IDs.
U	Ungroup	This removes this group, converting the sequences to orphans.
D	Delete Group	This removes the group and deletes all the sequences in it from the dataset.
M	Group Menu	Returns to the Group Menu. (See **2.5.6: The Grouping Menu**, above.)
Q	Quit Review	This will return to the Group Menu.
N or **ENTER**	Next	This will proceed to the next group if there is one.
F or **ENTER**	Finish Review	This will finish the review, if this is the last group.

Upon quitting or finishing the review, you are given the choice to enter a sentence for the log file, which may be useful if you wish to annotate any abnormal decisions made. There is also the option to save a backup of the current alignment and tree.

2.5.10: The gnspacc=T/F Option

The **gnspacc=T** command reformats the names of the input sequences from something like:

```
>ENSP000000223233 blah blah blah
```

to

```
>ens_HUMAN__ENSP000000223233 blah blah blah
```

It is not necessary, and an be switched off with **gnspacc=F**, but does make it much easier to work out what species each sequence is from or, at least, whether they are the same species. The first word of the description always becomes:

>**GN_SP__ACC**, where:

- **GN** is a gene or database identifier. (If you select 'New Gene' while reviewing a group, this is the bit you change. It can be very helpful in trees of multiple subfamilies but keep it short and match SwissProt where possible, e.g. DBOH in DBOH_MOUSE)

- **SP** is a species code. This is generally the Uniprot species code or the best guess the program can make (namely the first three letters of the genus and the first two letters of the species).

- **AC** is the accession number.

This is done because the first word must be unique for each sequence and, for clustalW, the first 30 characters must be unique. It always should be but if you get an error relating to the names, this may be one of the potential causes.

2.5.11: Establishment of Subfamilies Options

Option	Description	Default
es=T/F	Establishment of Subfamilies	[True]
root=X	Rooting of tree, where **X** is: - **mid** = midpoint root tree. - **ran** = random branch. - **ranwt** = random branch, weighted by branch lengths. - **man** = always ask for rooting options (unless i<0). - **FILE** = with seqs in **FILE** as outgroup. (Any option other than above)	[mid]
bootcut=X	cut-off percentage of tree bootstraps for grouping.	[0.8]
mfs=X	minimum family size	[3]
fam=X	minimum number of families (If 0, no subfam grouping)	[2]
orphan=T/F	Whether orphans sequences (not in subfam) allowed.	[True]
allowvar=T/F	Allow variants of same species within a group.	[False]
groupspec=X	Species for duplication grouping	[None]
group=X	Grouping of tree, where **X** is - **man** = manual grouping (unless i<0). - **dup** = duplication (all species unless groupspec specified). - **qry** = duplication with species of Query sequence (or Sequence 1) of treeseq. - **one** = all sequences in one group. - **None** = no group (case sensitive). - **FILE** = load groups from file	[None]

3: BADASP Statistics

BADASP currently incorporates two types of statistics:

1. **Functional Specificity Prediction.** These aim to identify Type I or Type II divergence of sequences in individual amino acid residues, which may therefore be important in the functional differences between the sequence groups/subfamilies.
2. **Conservation.** In addition to residues implicated in functional divergence (positive selection) it is of interest to identify those residues under consistent purifying (negative) selection, which may be important for common functionality between all sequences/subfamilies.

Statistics currently implemented in BADASP are described below. The modular open source nature of BADASP makes it easy to add extra statistics. For additional measures of specificity or conservation, please contact the author… or add them yourself!

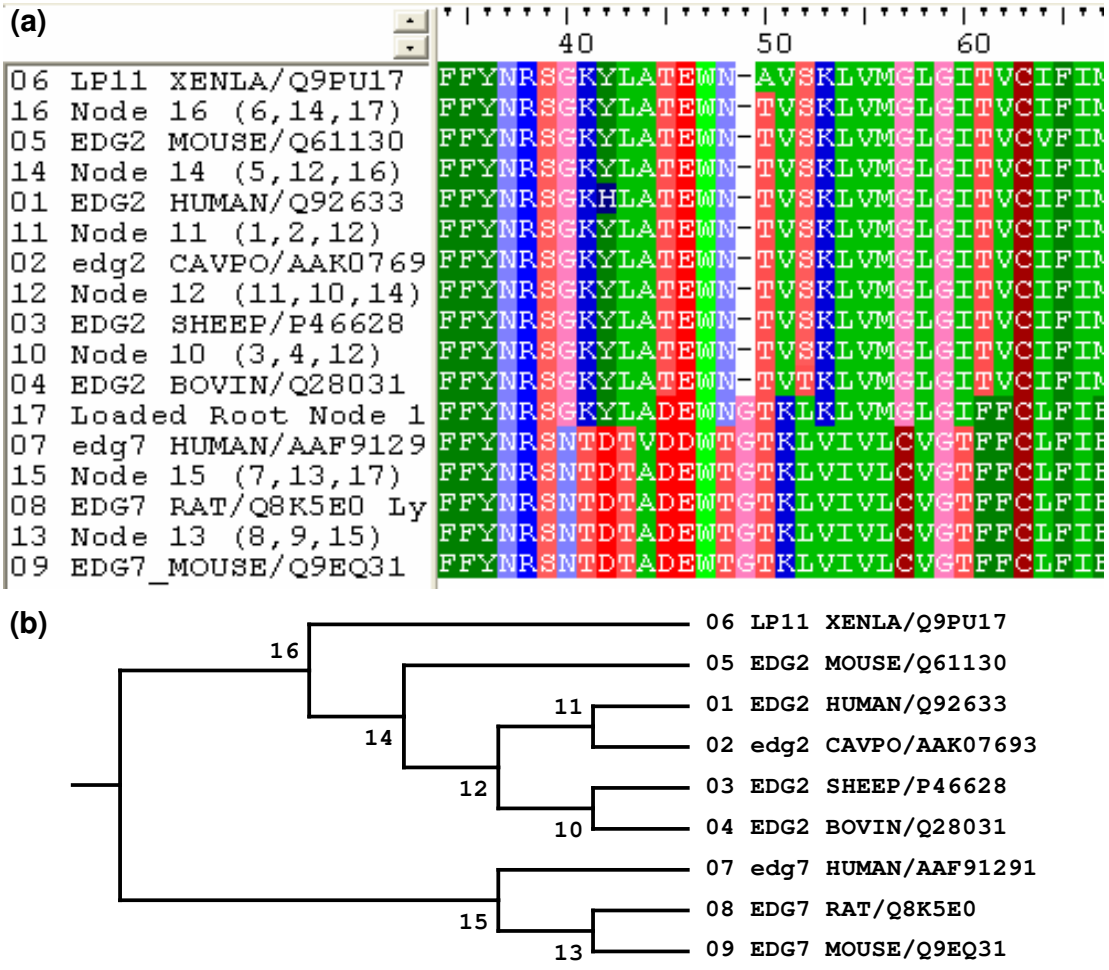

Figure 1. Portion of example alignment with ancestral sequences in GASP Format. (a) Screen shot of part of alignment as viewed with BioEdit (Hall 2001). Ancestral nodes are interspersed among extanct sequences and numbered according to **(b)** phylogenetic tree, without branch lengths and with node numbers labelled.

3.1: Functional Specificity Prediction

3.1.1: Burst After Duplication (BAD) Methods (BAD, BADX, BADN)

The primary functional specificity measure used in BADASP is based on the Burst After Duplication (BAD) score (Caffrey et al. 2000). The BAD method is based on amino acid properties and gives a high specificity score for residues that are conserved within subfamilies but differ between them (e.g. **Figure 1** position 53: K vs V). BAD is built on the underlying assumption that sites critical to changes in gene function between paralogues are marked by a burst of radical amino acid substitutions directly after duplication, which are subsequently conserved within orthologous groups. This is calculated by comparing the changes in physiochemical properties along the relevant branches for each site:

$$BAD = RC - AC \qquad (1),$$

where AC is the 'Ancestral Conservation' score, calculated as the change in physiochemical properties between the duplication node and the ancestral node for the subfamily; RC is the 'Recent Conservation' calculated as the mean change in properties between the ancestor of the subfamily and each orthologous (leaf) sequence.

The standard BAD measure (also called BADT) explicitly analyses two subfamilies, related by a single duplication event, for Type II divergence. BADT is simply the sum of the BAD scores (1) for the two subfamilies.

BADX looks for Type I neofunctionalisation in a specific Query subfamily by comparing it to its duplicate only. The BADX score considers only the Query subfamily and the branch leading to the outgroup clade:

$$BADX = RC - AC_X \qquad (2),$$

where AC_X is a modified AC score, calculated as the change in physiochemical properties between the two post-duplication nodes; RC is for the Query subfamily only. This method is more robust to incorrect ancestral sequence assignment at the duplication node.

BADN looks for Type II divergence across multiple subfamilies. The BADN score considers all the Query subfamily and all outgroup subfamilies:

$$BADN = BAD\Sigma / (N - 1) \qquad (3),$$

where $BAD\Sigma$ is the sum of the BAD scores (1) for each subfamily; N is the number of subfamilies. AC values (1) are calculated using the ancestor of each subfamily and the root of the tree, which should be the most ancient gene duplication.

As with other methods, all three BAD algorithms are obviously sensitive to alignment and tree quality. In addition, incorrect ancestral sequence prediction will give erroneous results. However, the advantage of BAD is that is allows some variability at the site of interest, so long as the property conservation is retained. Position 52 in **Figure 1** therefore gets a fairly high BAD score, even though the consensus S is a T in one sequence of the EDG2 subfamily.

3.1.2: Livingstone and Barton (SSC)

This method (Livingstone and Barton 1993) formed the initial foundation of the BAD method and relies on similar underlying assumptions. Like BAD, SSC subtracts 'Ancestral Conservation' from 'Recent Conservation'. Unlike BAD, however, ancestral sequences are not used. Instead, the conservation scores simple measure the number of properties that are absolutely conserved across all sequences in a subgroup (aRC) or the whole dataset (aAC). aAC for the whole family is subtracted from the mean aRC for the subfamilies:

SSC = MaRC – aAC, where MaRC is mean aRC.

Unlike the BAD measures, rare variants within a subfamily are given equal weighting with common variants.

3.1.3: Property Difference After Duplication (PDAD)

This is an unpublished method of R. Edwards, also based on the method of Livingstone and Barton (1993) and relying on the same assumptions. As with SSC, ancestral sequences are not used. PDAD, however, does weight conservation to allow for rare variants by using mean property conservation rather than absolute property conservation:

$$PDAD = \Sigma(SC - OC)/n,$$

where SC is the mean pairwise property conservation within a subfamily, OC is the mean pairwise property conservation between sequences within a subfamily and all those in a different subfamily, and n is the number of subfamilies.

3.1.4: Evolutionary Trace Analysis (ETA, ETAQ)

BADASP incorporates a limited version of the Evolutionary Trace method (Lichtarge et al. 1996) that does not incorporate 3D information. In the basic method, residues are simply scored 1 if they are entirely conserved within subfamilies but differ between them, or 0 if either condition is not met (*i.e.* there is variation within one or more subfamilies and/or conservation between at least two subfamilies). Unlike BAD or PDAD, residues with a single variant – even if highly similar to the consensus – will be penalised, such as position 52 in **Figure 1**.

With the Quantitative variant (ETAQ) (R. Edwards, unpublished), some effort is made to allow for some variation by changing the all-or-nothingness of ETA to a quantitative trait:

$$ETAQ = \Sigma(ET) / n,$$

where $\Sigma(ET)$ is the number of subfamilies that are 100% conserved at that residue and with a different amino acid to any other ET family, and n is the number of subfamilies.

ETAQ is therefore reduced by each family that is variant and by each family that is invariant but for the same amino acid as another ET family.

Both ETA variants suffer from the problem that they take no account of the type of substitution observed between different subfamilies, grading a highly conservation change (such as leucine to isoleucine) the same as a dramatic change (such as leucine to arginine).

3.2: Conservation Statistics

In addition to the specificity scores listed above, BADASP will also calculate a number of simple conservation measures for the alignment as a whole. This can help assess the relevance of residues with high scores, indicating whether they fall inside otherwise well conserved (and therefore probably well-aligned) regions. **NB.** These have not been rigorously tested. Please report any anomalous results!

3.2.1: Information Content [info]

This is a simple measure of the information content of the residue across the whole alignment.

$$I = \Sigma\ f_i \log(f_i) / \log(n),$$

where f_i is the frequency of amino acid i, and n is the number of different amino acids, including gaps (21).

I ranges from 0 (low information content) where all amino acids are different, to 1 (high information content) where all amino acids are the same.

3.2.2: Absolute Property Conservation (pcon_abs)

This is a count of the number of properties that are absolutely conserved across all sequences of the alignment for a given position. If any one sequence differs to any with respect to a given property then that property is not counted. This gives a number from 0, where no property is conserved, up to the number of properties (10 by default), where all properties are conserved in all sequences. Gapped and X residues are ignored.

3.2.3: Mean Property Conservation (pcon_mean)

This calculates the mean pairwise property difference for all sequences in the alignment. This time, gapped residues are handled using the parameter **`aagapdif=X`**, which assigns all comparisons between a gap and another residue (including another gap) a value of **`X`** property differences (5 by default).

3.2.4: Query Property Conservation (qpcon_mean, qpcon_abs)

These measures are the same as the two above with the exception that, rather than compare all the sequences in the alignment, they compare the query protein (if given) with the other members of its subfamily.

3.3: Reducing BADASP Output using Value/Rank filters

Following the full output of BADASP Statistics, there is an option to output a reduced dataset of selected statistics and/or residues that have above a certain threshold value or rank for one or more statistics. By default, this will save to ***.partial.badasp** but the filename can be changed. If the file already exists then it can be appended allowing, for example, the output of all the residues above a certain score into the same file. (Of course, this will only work if the same output columns are present in both files. For this reason, headers are still output.)

The first option is to reduce the number of columns that are output. This is not generally necessary as the command-line parameters can control a lot of these but additional options are given to exclude query-specific columns or the details for a given subfamily if desired.

A more common desire is to return only those residues with the top 10% of BAD scores, for example. Any statistic can be used to filter residues, either by value or by rank. Only residues with a value equal to or greater than the given value will be returned. Multiple statistics can be filtered on simultaneously, allowing you to return, for example, all residues with a BAD score of 3+ and a BADX score in the top 10% (rank >= 0.9).

3.4: Manipulating and Visualising BADASP results

As an alternative to filtering data at the output stage, an accessory application can be used to manipulate or visualise data. Microsoft Excel is particularly recommended for this task as it has a number of features that make such manipulations easy. The output file can be opened in Excel easily as a delimited text file. The first three rows of the results file list details on the run of BADASP used to generate the results and can be deleted if desired. (Simply select and use **Edit, Delete...(, Entire row)** to delete rows.)

3.4.1: Visualising Data

For easy scanning of results, use the "Freeze panes" function to keep the column headings and key positions/aas on screen at all times. Drag the narrow bar by the vertical and horizontal scroll bars to split the spreadsheet into the desired partitions and use the

Window, Freeze Panes command to fix them in place. There are then two easy ways to pick out the results of interest. One is to select the columns of interest and use **Format, Conditional Formatting...** to reformat cells with certain value ranges. The other is to select the columns of interest along with the desired residue position column and draw a graph (**Insert, Charts..., XY (Scatter)**) of the results. See MS Excel help for more information on these tools. (Other similar packages will have similar functions.)

3.4.2: Extracting Relevant Portions of Output data

In MS Excel, simple limiting of data to ranges of values can be done by using the sort command (**Data, Sort...**) on the relevant columns and then deleting all the rows above or below the desired value (**Edit, Delete...(, Entire row)**). This can then be repeated on other rows. For more complex queries (e.g. BAD > 3 *or* BADX > 3), an extra column can be created and an **IF** function used to give it certain values dependent on other columns (**Insert, Function..., IF**). This new column can then be used to sort and delete unwanted rows (or conditional format as above).

For really complex manipulating and display of data, it may be desirable to import the results into a statistics package, such as STATA. These packages will have their own functions for removing rows which do not conform to certain rules. Alternatively, a custom parsing script can be written in Perl or Python, for example, to extract certain data. It is beyond the scope of this manual to list all these options but I am happy to be contacted if someone wants advice on how to best extract the results they desire.

4: Appendices

4.1: Appendix I: Amino Acid Property Matrix

4.1.1: Default Matrix

By default, a property matrix based on that used by Livingstone and Barton (1993) is used. This is file can be replaced by one in the same format using the **aaprop=FILE** option. This file may have different (numbers of) properties than those used in the default file:

```
# Based on Property Matrix of Livingstone and Barton.
PROPERTY    I L V C A G M F Y W H K R E Q D N S T P
Hydrophobic 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0
Polar       0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0
Small       0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1
Proline     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
Positive    0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
Negative    0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0
Charged     0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0
Tiny        0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0
Aliphatic   1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Aromatic    0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
```

This property matrix is then converted by BADASP into a property difference matrix using the rje_aaprop.py module.

4.1.2: Gap treatment in property difference calculations

Currently gaps are handled using a special parameter **aagapdif=X**, which assigns all comparisons between a gap and another residue (including another gap) a value of **X**. In future, gaps may be incorporated into input property matrices if desired. Please contact the author to have this implemented sooner. By default this difference is 5, which is half of the default number of properties. This parameter is independent of property number, however, and can be given a value in excess of the number of properties if it is desirable to make gapped regions extremely different from ungapped ones.

4.1.3: Wildcard residue treatment in property difference calculations

Wildcard residues such as X are handled using a special parameter **aanulldif=X**, which assigns all a set difference for each property. By default this is 0.5, so a wildcard/unknown residue will share half its properties with any other residue. A value of 0 will give no property differences, while 1 will assume all properties are different.

4.2: Appendix II: Command-line Options

4.2.1: How to Use this Section

This section lists all the Command-line options than may be of use when using BADASP. Note that different options are associated with different modules. These are indicated by the name of the module given (in brackets). Default values are given [in square brackets]. Not all the options for a given module are listed here but can be found by printing the `__doc__` attribute of the module at a Python prompt, or using the **`help`** option:

`print badasp.__doc__` (in Python)

`python badasp.py help` (commandline)

Note that the **`help`** option of BADASP will list all the options for all the relevant modules.

This section has not been completed. For now, the listing provided as part of the module documentation is given. This shall be expanded with time. (Hopefully soon.) In the meantime, please contact me if you want any further details of a specific option and/or advice as to when (not) to use it.

4.2.2: Option Types

There are essentially two types of command-line option:

1. Those that require a value (numerical or text), **`option=X`**. Those that require a filename as the value will be witten: **`option=FILE`**
2. True/False (On/Off) options, **`option=T/F`**. For these options:
    a. **`option=F`** and **`option=F`**alse are the same and turn the option off
    b. **`option`**, **`option=T`** and **`option=True`** are the same and turn the option on

4.2.3: INI Files

As well as feeding commands in on the command-line, any options listed can also be save in a plain text file and called using the option **`ini=FILE`**. Automatically, the program will read in any options from the file **badasp.ini** and **rje.ini**, if present.

4.2.4: Option Precedence

Later options will supersede earlier ones if they are mutually exclusive. Options from an ini file will be inserted into the list at the point the ini file is called. (At the start for badasp.ini.) This means that ini file options can be over-ruled, e.g.

`badasp.py ini=eg.ini i=1` will supersede any interactivity setting in **eg.ini** with **`i=1`**.

`badasp.py i=1 ini=eg.ini` will use any interactivity setting in **eg.ini** and over-rule **`i=1`**.

4.2.5: Command-line Options

Option	Description	Default	Module
	General Dataset Input/Output		
`seqin=FILE`	Loads sequences from FILE	[None]	rje_seq
`query=X`	Selects query sequence by name (or part of name, e.g. Accession Number)	[None]	rje_seq
`basefile=X`	Basic 'root' for all files X.*	['root' of seqin=FILE]	rje_seq
`v=X`	Sets verbosity (-1 for silent)	[0]	rje
`i=X`	Sets interactivity (-1 for full auto)	[0]	rje
`log=FILE`	Redirect log to FILE	[haqesac.log or basefile.log]	rje
`newlog=T/F`	Create new log file.	[False]	rje
`rank=T/F`	Whether to output ranks as well as scores	[True]	badasp
`append=T/F`	Append results to FILE instead of standard output to *.badasp	[False]	badasp
	BADASP Statistics		
`funcspec=X1[,X2,..]`	List of functional specificity methods to apply X1,X2,..,XN - `BAD` = Burst After Duplication (2 Subfamilies) - `BADX` = Burst After Duplication Extra (Query Subfam versus X subfams) - `BADN` = Burst After Duplication vs N Subfams (2+ Subfams) - `SSC` = Livingstone and Barton Score - `PDAD` = Variant of Livingstone and Barton - `ETA` = Evolutionary Trace Analysis (Basic) - `ETAQ` = Evolutionary Trace Analysis (Quantitative) - `all` = All of the above!	[all]	badasp
`seqcon=X1[,X2,..]`	List of sequence conservation measures to apply X1,X2,..,XN - `Info` = Information content - `PCon` = Property Conservation (Absolute) - `MPCon` = Mean Property Conservation - `QPCon` = Mean Property Conservation with Query - `all` = All of the above	[all]	badasp

Option	Description	Default	Module
	Tree and Grouping Options		
nsfin=FILE	File name for Newick Standard Format tree	[*.nsf]	rje_tree
root=X	Rooting of tree, where X is: - mid = midpoint root tree. - ran = random branch. - ranwt = random branch, weighted by branch lengths. - man = always ask for rooting options (unless i<0). - FILE = with seqs in FILE as outgroup. (Any option other than above)	[mid]	rje_tree
bootcut=X	cut-off percentage of tree bootstraps for grouping.		rje_tree
mfs=X	minimum family size	[3]	rje_tree
fam=X	minimum number of families (If 0, no subfam grouping)	[0]	rje_tree
orphan=T/F	Whether orphans sequences (not in subfam) allowed.	[True]	rje_tree
allowvar=T/F	Allow variants of same species within a group.	[False]	rje_tree
gnspacc=T/F	Convert sequences into gene_SPECIES__AccNum format wherever possible.	[True]	rje_seq
groupspec=X	Species for duplication grouping	[None]	rje_tree
group=X	Grouping of tree, where X is - man = manual grouping (unless i<0). - dup = duplication (all species unless groupspec specified). - qry = duplication with species of Query sequence (or Sequence 1) of treeseq. - one = all sequences in one group. - None = no group (case sensitive). - FILE = load groups from file	[None]	rje_tree
	GASP Ancestral Sequence Prediction		
pamfile=FILE	Sets PAM1 input file	[jones.pam]	rje_pam
pammax=X	Initial maximum PAM matrix to generate	[100]	rje_pam
pamcut=X	Absolute maximum PAM matrix	[1000]	rje_pam
fixpam=X	PAM distance fixed to X	[100].	rje_ancseq
rarecut=X	Rare aa cut-off	[0.05].	rje_ancseq
fixup=T/F	Fix AAs on way up (keep probabilities)	[True].	rje_ancseq

Option	Description	Default	Module
fixdown=T/F	Fix AAs on initial pass down tree	[False].	rje_ancseq
ordered=T/F	Order ancestral sequence output by node number	[False].	rje_ancseq
pamtree=T/F	Calculate and output ancestral tree with PAM distances	[True].	rje_ancseq
desconly=T/F	Limits ancestral AAs to those found in descendants	[True].	rje_ancseq
Xpass=X	How many extra passes to make down & up tree after initial GASP	[1].	rje_ancseq
	System Info		
win32=T/F	Run in Win32 Mode	[False]	rje

4.3: Appendix III: Distributed Python Modules

Module	Description	Classes
badasp	This module contains the main BADASP method, which calls the relevant methods from the other modules and handles results output.	-
rje	General module containing Classes used by all my scripts plus a number of miscellaneous methods.	RJE_Object_Shell, RJE_Object, Info, Out, Log
rje_aaprop	This module Takes an amino acid property matrix file and reads into an AAPropMatrix object. Converts in an all by all property difference matrix. By default, gaps and Xs will be given null properties (None) unless part of input file.	AAPropMatrix
rje_ancseq	This module contains the objects and methods for ancestral sequence prediction. Currently, only GASP (Edwards & Shields 2004) is implemented. Other methods may be incorporated in the future.	Gasp, GaspNode
rje_blast	Performs BLAST searches and loads results into objects.	BLASTRun, BLASTSearch, BLASTHit, PWAln
rje_conseq	Contains objects for calculating conservation stats used in BADASP.	SeqStat
rje_seq	Contains Classes and methods for sets of DNA and protein sequences. (Currently only protein sequences supported.)	SeqList, Sequence, DisMatrix
rje_specificity	Contains objects for calculating specificity stats used in BADASP.	FuncSpec
rje_tree	Reads in, edits and outputs phylogenetic trees. Executes duplication and subfamily determination.	Tree, Node, Branch
rje_pam	This module handles functions associated with PAM matrices. A PAM1 matrix is read from the given input file and multiplied by itself to give PAM matrices corresponding to greater evolutionary distance. (PAM1 equates to one amino acid substitution per 100aa of sequence.)	PamCtrl, PAM
rje_sequence	Contains the Sequence class used by rje_seq.py	Sequence
rje_tree_group	Contains all the Grouping Methods for rje_tree.py	-

4.4: Appendix IV: Log Files

This part of the manual has not yet been written. Sorry! Contact the author if you have questions about the log files.

4.5: Appendix V: Troubleshooting

Currently, this is a small section as I have not had enough feedback to have FAQs, or anything like that. Here is a list of things that I think MAY cause problems to the unwary:

- Giving file names with spaces without enclosing them in "double quotes". (Otherwise only the first word will be taken as the filename)
- Incorrect formatting of input files. Check that all the sequences in the alignments are really the same length and that the tree does not have more that one trifurcation.
- I'm not sure when but there is a possibility of problems if running in Windows without the **`win32`** option.
- Some commands not listed above may be used when running the program interactively using **rje_tree.py** (see **2.1: Running BADASP**). Run *`python rje_tree.py help`* to see a full list of options and check the log file to see which ones are being invoked. Filtering out sequences etc. may cause errors.

4.6: Appendix VI: References

Caffrey, D.R., O'Neill, L.A. and Shields, D.C. (2000) A method to predict residues conferring functional differences between related proteins: application to MAP kinase pathways, *Protein Sci*, **9**, 655-670.

del Sol Mesa, A., Pazos, F. and Valencia, A. (2003) Automatic methods for predicting functionally important residues, *J Mol Biol*, **326**, 1289-1302.

Edwards, R.J. (2004a) GASP: Gapped Ancestral Sequence Prediction. *http://www.bioinformatics.rcsi/~redwards/gasp/*

Edwards, R.J. (2004b) HAQESAC: Homologue Alignment Quality, Establishment of Subfamilies and Ancestor Construction. *http://www.bioinformatics.rcsi/~redwards/haqesac/*

Gu, X. (2001) Maximum-likelihood approach for gene family evolution under functional divergence, *Mol Biol Evol*, **18**, 453-464.

Gu, X. and Vander Velden, K. (2002) DIVERGE: phylogeny-based analysis for functional-structural divergence of a protein family, *Bioinformatics*, **18**, 500-501.

Hall, T. (2001) BioEdit Sequence Alignment Editor for Windows 95/98/NT. *http://www.mbio.ncsu.edu/BioEdit/bioedit.html*

Hannenhalli, S.S. and Russell, R.B. (2000) Analysis and prediction of functional sub-types from protein sequence alignments, *J Mol Biol*, **303**, 61-76.

Johnson, J.M. and Church, G.M. (2000) Predicting ligand-binding function in families of bacterial receptors, *Proc Natl Acad Sci U S A*, **97**, 3965-3970.

Jones, D.T., Taylor, W.R. and Thornton, J.M. (1992) The rapid generation of mutation data matrices from protein sequences, *Comput Appl Biosci*, **8**, 275-282.

Kalinina, O.V., Mironov, A.A., Gelfand, M.S. and Rakhmaninova, A.B. (2004a) Automated selection of positions determining functional specificity of proteins by comparative analysis of orthologous groups in protein families, *Protein Sci*, **13**, 443-456.

Kalinina, O.V., Novichkov, P.S., Mironov, A.A., Gelfand, M.S. and Rakhmaninova, A.B. (2004b) SDPpred: a tool for prediction of amino acid residues that determine differences in functional specificity of homologous proteins, *Nucleic Acids Res*, **32**, W424-428.

Lichtarge, O., Bourne, H.R. and Cohen, F.E. (1996) An evolutionary trace method defines binding surfaces common to protein families, *J Mol Biol*, **257**, 342-358.

Livingstone, C.D. and Barton, G.J. (1993) Protein sequence alignments: a strategy for the hierarchical analysis of residue conservation, *Comput Appl Biosci*, **9**, 745-756.

Zhang, J. (2003) Evolution by gene duplication: an update, *Trends Ecol Evol*, **18**, 292-298.