Gapped Ancestral Sequence Prediction (for proteins)

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 GASP**, **Input**, and **Output**. Gluttons for punishment can get more details in the **Appendices**. (**NB.** There has been one modification to the algorithm since publication. This can be found in **3.1.4: Changes since Publication**.)

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 GASP version 1.2.

The manual was last edited on July 6th 2005.

1.2: 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 GASP will be marked in text by (dark yellow) Times New Roman.

1.3: Getting Help

Much of the information here is also contained in the GASP paper (Edwards and Shields 2004), the website (http://www.bioinformatics.rcsi.ie/~redwards/gasp/) 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 gasp.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 **Output**) too. Usually, it will be a problem with the input files (possibly formatting) but there are probably still a few bugs in there somewhere too.

1.4: Why use GASP?

The prediction of ancestral protein sequences from multiple sequence alignments is useful for many bioinformatics analyses. Predicting ancestral sequences is not a simple procedure and relies on accurate alignments and phylogenies. Strict consensus methods are quick but suffer from over-representation in dense clades while more sophisticated likelihood-based methods can be computationally expensive.

GASP (Gapped Ancestral Sequence Prediction) predicts ancestral sequences from phylogenetic trees and the corresponding multiple sequence alignments. Alignments may be of any size and contain gaps. GASP first assigns the positions of gaps in the phylogeny before using a probability-based approach centred on amino acid (PAM) substitution matrices to assign ancestral amino acids. Important outgroup information is used by first working down from the tips of the tree to the root, using descendant data only to assign probabilities, and then working back up from the root to the tips using descendant and outgroup data to make predictions.

GASP will predict ancestral sequences from multiple protein alignments of any size. Although unlikely to be as accurate in all cases as some of the more sophisticated maximum likelihood approaches, it can process a wide range of input phylogenies and will predict ancestral sequences for gapped and ungapped residues alike.

1.5: Installation

GASP 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 **Running GASP**, 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 2.3. They have not been tested (yet) with the latest release of Python (2.4) but again I think it should work OK. 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.

2: Fundamentals

2.1: Running GASP

2.1.1: The Basics

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

```
python gasp.py seqin=FILENAME
```

For the example provided in the distribution:

```
python gasp.py seqin=gasp_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 gasp.py seqin=fasta_file nsfin=nsf_file
```

```
python gasp.py seqin=gasp_eg.fas nsfin=gasp_eg.nsf
```

If running in Windows, you can just double-click the gasp.py file and then enter the command line arguments when prompted.

Note: Filenames cannot contain spaces!

Alternatively, you can run the rje_tree.py file (from commandline or double-clicking in Windows) for a menu-driven system. Running this file gives you many more options for editing your tree before you proceed with GASP.

```
python rje_tree.py [seqin=fasta_file nsfin=nsf_file …]
```

2.1.2: Interactivity and Verbosity settings

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

```
python gasp.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 GASP 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 **3.2: Appendix II: Command-line Options**. These may be given after the run command, as above, or loaded from a *.ini file (see for **3.2.3: INI Files** details).

2.2: Input

GASP 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 GASP. Input trees must have branch lengths. Bootstrap values are not used by GASP 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.

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 can be used to tidy up input alignments to the correct format.

2.2.2: Newick (Phylip) format tree

GASP 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 GASP.

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;
```

Rooting. GASP needs a rooted tree for ancestral sequence prediction. However, the GASP 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, GASP will place the root on the first branch that groups all outgroups into a single clade. (A random rooting option is also provided.)

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 GASP program in the file jones.pam. 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.

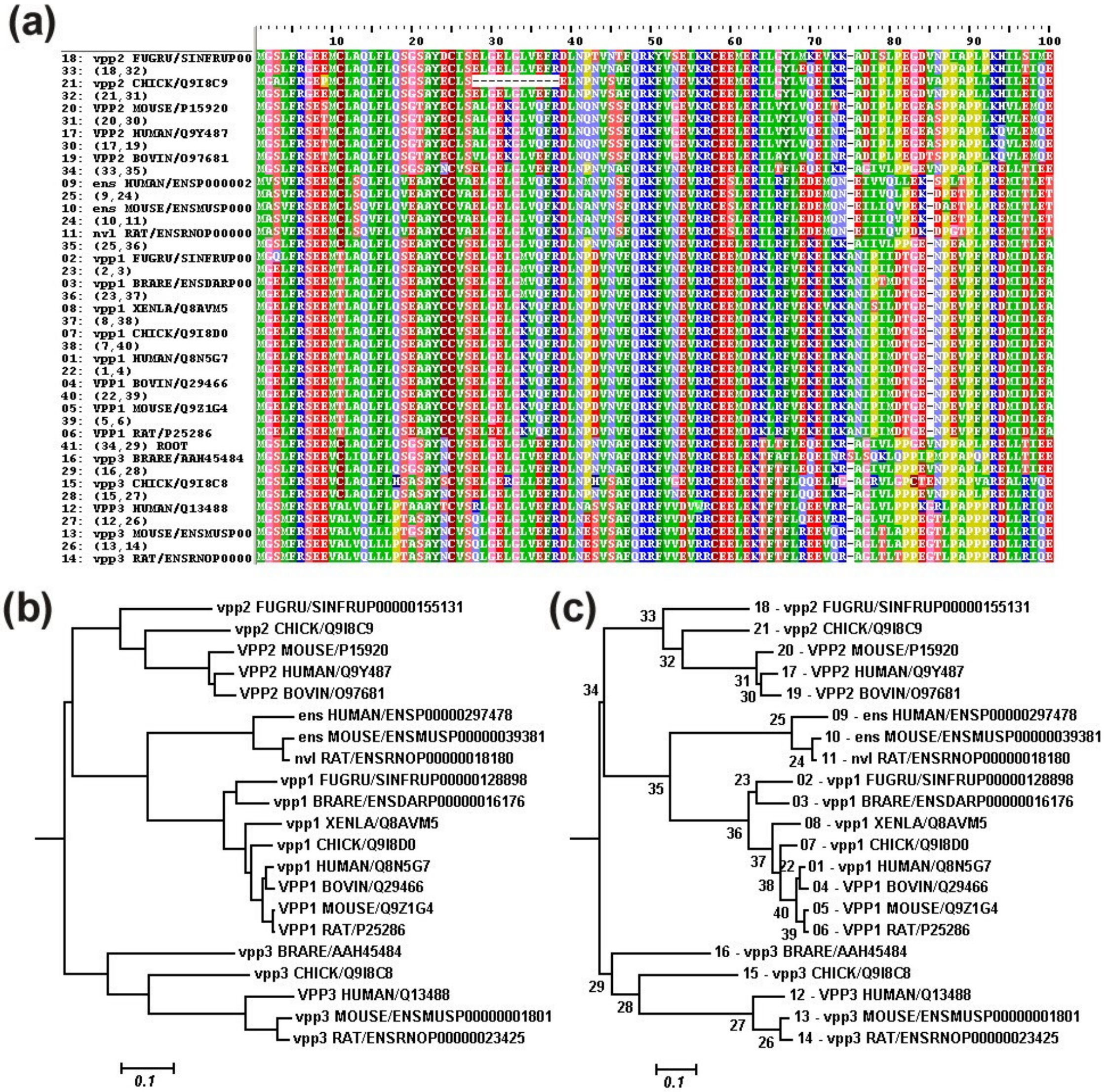

Figure 1. Example GASP Output. (a) Output alignment with ancestral nodes interspersed among input sequences (see below). **(b)** Rooted Input tree with branches scaled according to input tree. **(c)** Rooted Tree with nodes labelled to match alignment (a) and branches rescaled based on ancestral sequences.

2.3: Output

GASP outputs three types of files (five total):

1. A fasta formatted alignment
2. Three output tree files:
    a. Newick format of the original input tree, rooted
    b. Newick format tree adjusted for ancestral predictions
    c. A raw text version of the tree
3. A log file. (See **3.4: Appendix IV: Log Files** for details.)

Sequences are read in from the alignment and node relationships established from the tree. The tree may be already rooted or rooted using GASP. Input trees must have branch lengths. Bootstrap values are not used by GASP 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.

Example output in shown in **Figure 1**. (Alignment viewed in BioEdit (Hall 2001), Trees viewed with TreeExplorer (Tamura 2001)).

2.3.1: Output Alignment

The output alignment is similar in format to the **Input**. Two options are possible:

1. All input sequences followed by all ancestral sequences
2. Ancestral sequences interspersed between input (terminal) sequences in an order to match the output trees. (See Figure 1)

In each case, the node is labelled is the manner:

`X Node X (Y,Z,A)`, where **`X`** is the number of the node (matching the raw text tree and the output tree (c)), **`Y`** and **`Z`** are its two descendant nodes, and **`A`** its ancestor.

In the reordered output (1), all sequences are ordered according to the node number (**`X`**). By default, however, ancestral sequences are interspersed so that node **`X`** would be between **`Y`** and **`Z`**. When opened in an alignment viewer, this keeps ancestral sequences in blocks with their descendant clades and makes differences between clades more obvious.

2.3.2: Output Trees

The Newick Output trees are designed to be viewed with K Tamura's TreeExplorer program (Tamura 2001). The second tree (Figure 1c) has node numbers instead of bootstrap values, allowing ancestral sequences to be matched to tree nodes. Branch lengths in this last file are replaced with the most likely PAM distance for a given branch:

$$p_X = \Sigma\, p_{ijX}$$

where p_X is the likelihood for a PAM distance of X, i is the ancestral amino acid at a given position, j is the descendant amino acid, p_{ijX} is the substitution probability of i to j in a PAMX matrix.

This allows a visual comparison between the branch lengths of the inputted phylogeny and the predicted branch lengths given the ancestral sequence predictions. This way, ancestral sequence bias towards the root or tips may be detected.

A raw text version of the input tree, with internal nodes numbered as for the output sequence file (and bootstrap values if provided), is also produced for viewing with a simple text editor, e.g. on a UNIX system (**Figure 2**).

```
           +---+ 1: ens_HUMAN/ENSP00000297478 {0.068990}
           |
    +------+ 20 [1000] {0.129250}
    |      |
    |      |   +-+ 2: ens_MOUSE/ENSMUSP00000039381 {0.017520}
    |      |   |
    |      +---+ 18 [1000] {0.051900}
    |          |
    |          +-+ 3: nvl_RAT/ENSRNOP00000018180 {0.013840}
    |
+---+ 21
    |
    |      +---+ 4: vpp1_FUGRU/SINFRUP00000128898 {0.068760}
    |      |
    |    +-+ 17 [999] {0.019720}
    |    | |
    |    | +---+ 5: vpp1_BRARE/ENSDARP00000016176 {0.058600}
    |    |
    +----+ 19 [1000] {0.098470}
         |
         |  +---+ 6: vpp1_XENLA/Q8AVM5 {0.061730}
         |  |
         +--+ 16 [1000] {0.032980}
            |
            | +--+ 7: vpp1_CHICK/Q9I8D0 {0.037660}
            | |
            +-+ 15 [958] {0.010150}
              |
              |   +-+ 8: vpp1_HUMAN/Q8N5G7 {0.012050}
              |   |
              | +-+ 13 [959] {0.007840}
              | | |
              | | +-+ 9: VPP1_BOVIN/Q29466 {0.014230}
              | |
              +-+ 14 [1000] {0.019250}
                |
                | +-+ 10: VPP1_MOUSE/Q9Z1G4 {0.002750}
                | |
                +-+ 12 [1000] {0.019160}
                  |
                  +-+ 11: VPP1_RAT/P25286 {0.002060}
```

Figure 2. Example Text tree output. Nodes are numbered in correspondence with the fasta file output. Bootstrap values are given in square brackets, branch lengths in curly brackets.

2.3.3: Log Files

The gasp 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 gasp.log but this can be changed with the `log=FILE` option. Logs will be appended unless the `newlog` option is used. (See **3.2: Appendix II: Command-line Options** and **3.4: Appendix IV: Log Files** for details.)

3: Appendices

3.1: Appendix I: The GASP Algorithm

3.1.1: Gap Placement

If the MSA has gaps, GASP will first assign gap status to every residue at every node. Insertions and deletions are assumed to be equally likely, although a gap is assigned in the case of a tied probability (below). For each residue *r*, GASP starts at the tips and works deeper into the tree, assigning a probability of a gap at each node *n*, which is equal to the mean probability of a gap at the descendant nodes:

$$p = \frac{p_1 + p_2}{2},$$

where p is the gap probability for residue r at node n. p_1 and p_2 are the gap probabilities for r at the two descendant nodes.

Terminal branches are given a probability of 1 if a gap is present or 0 if not. Once the root is reached, the gap status is fixed for the root. If the probability of a gap is greater than or equal to 0.5, residue r is fixed as a gap, otherwise r is fixed as a 'non-gap'. GASP then works back up the tree from the root, this time using the new ancestral gap probability and both descendant gap probabilities to recalculate the gap probability:

$$p = \frac{p_0 + p_1 + p_2}{3},$$

where p_0 is the gap probability for r at the ancestral node.

As with the root, r is fixed as a gap if $p \geq 0.5$. This continues until all nodes are assigned as 'gap' or 'non-gap'.

3.1.2: Ancestral Sequence Prediction

Once gaps are assigned, ancestral sequences are predicted in a similar fashion. Each residue r is assigned a probability for each amino acid at each node n. At the tips, r has a probability of 1 for the amino acid that is present in the MSA. GASP then works down the tree assigning probabilities based on the descendant nodes, branch lengths and a substitution matrix. By default, the PAM matrix of Jones *et al.* 1992 (Jones et al. 1992) is used. This is a PAM**1** matrix, which represents the probability that a given amino acid will be substituted by each other amino acid when the mean substitution rate is **1**/100 residues. To make a PAM*X* matrix, which represents a length of evolutionary time where a sequence will have undergone *X* substitutions per 100 residues, the PAM1 matrix is multiplied by itself *X*-1 times:

$$p_{ijX} = \sum_{k=1}^{20} p_{ik(X-1)} p_{kj1},$$

where i is the ancestral amino acid, j is the descendant amino acid, k is the 20 possible transitory amino acids, p_{ijX} is the substitution probability of i to j in a PAM*X* matrix, $p_{ik(X\text{-}1)}$ is the substitution probability of i to k in a PAM(X-1) matrix and p_{kj1} is the substitution probability of j to k in a PAM1 matrix.

Unless the ancestral node has a gap (as calculated above) at position r, the ancestral probabilities for each amino acid are calculated for the two descendant branches individually, using a PAM*X* matrix, where *X* is 100 times the branch length as substitutions per site, *i.e.* a branch of 0.1 substitutions per site would use a PAM10 matrix:

$$p_i = \frac{\sum_{j=1}^{20} p_{ijX1} p_{dj1} + \sum_{j=1}^{20} p_{ijX2} p_{dj2}}{2},$$

where p_i is the probability of amino acid *i* at residue *r* of node *n*, p_{ijX1} and p_{ijX2} are the probabilities of substitution from amino acid *i* to each amino acid *j* in the appropriate PAM matrix for the two descendant branches, p_{dj1} and p_{dj2} are the probabilities of amino acid *j* being at position *r* at the two descendant nodes.

Once the root is reached, the most probable amino acid is fixed as the ancestral sequence. As with gaps, GASP then works back up the tree, using the fixed ancestral node amino acid and the descendant node probabilities to give new probabilities for each amino acid. The most probable amino acid is then fixed and the process continues until all residues and all nodes have a fixed sequence.

GASP is primarily designed for reasonably small trees (6-30 sequences), although there is no limit on input tree size. For larger trees, probabilities for each amino acid get very small near the root, which can lead to a heavy bias towards the fixed ancestral amino acid when GASP works back up the tree. To counter this GASP arbitrarily reduces any probabilities below a certain exclusion threshold (0.05 by default) to zero, thus reducing the slow accumulation of very unlikely amino acids.

3.1.3: Variations

To optimise the PAM matrices used for probability calculations, GASP uses the variable branch lengths read from the input phylogeny. There is also an option to fix the PAM distance used for all branches, which would allow the use of trees without branch lengths.

Assignment of ancestral amino acids with the GASP algorithm is achieved by combining data from the descendants of a given node *n* and its direct ancestor n_0. This ancestor itself is heavily influenced by the descendants of *n* but also by the 'outgroup' to *n*, namely those sequences that are descendant to n_0 but not to *n*. The outgroup information contained by the ancestral node n_0 can be vital in determining the correct sequence for *n* when the descendants of *n* are variable. For this reason, the GASP algorithm will, by default, fix ancestral sequences as it moves back 'up' the tree from the root, increasing the relative weighting of the outgroup to the two descendants. Because there is a chance of the wrong amino acid sweeping back up the tree (especially if rare amino acid probabilities are allowed to accumulate by reducing the exclusion threshold), there is an option to use amino acid probabilities from the ancestral node in the last stage of GASP rather than giving the fixed amino acid an ancestral probability of 1. This option should be used with caution.

3.1.4: Changes since Publication

All of the above algorithm information is taken straight from the GASP paper (Edwards and Shields 2004). Since publication, there has been one additional modification to the algorithm. This was to rectify a tendency in certain circumstances of incorrect amino acids sweeping back up a tree. Correction takes the form of an extra pass down the tree, where amino acid frequencies are first reassigned at each node using the all three neighbouring nodes and then a second pass down where the amino acids are fixed. This works the same way as the fixing of amino acids on the first pass up the tree from the root and works to even out the relative influence of leaves and outgroups on internal nodes. This is controlled by the **`xpass=X`** option, which is set by default to **`xpass=1`**.

3.2: Appendix II: Command-line Options

3.2.1: How to Use this Section

This section lists all the Command-line options than may be of use when using GASP. 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 gasp.__doc__ (in Python)

python gasp.py help (commandline)

Note that the **help** option of GASP 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.

3.2.2: 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

3.2.3: 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 gasp.ini, if it is present.

3.2.4: 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 gasp.ini.) This means that ini file options can be over-ruled, e.g.

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

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

3.2.5: Command-line Options

GASP Options:

- **fixpam=X** : PAM distance fixed to **X** [100]. (rje_ancseq.py)
- **rarecut=X** : Rare aa cut-off [0.05]. (rje_ancseq.py)
- **fixup=T/F** : Fix AAs on way up (keep probabilities) [True]. (rje_ancseq.py)
- **fixdown=T/F** : Fix AAs on initial pass down tree [False]. (rje_ancseq.py)

- **ordered=T/F** : Order ancestral sequence output by node number [False]. (rje_ancseq.py)
- **pamtree=T/F** : Calculate and output ancestral tree with PAM distances [True]. (rje_ancseq.py)
- **desconly=T/F** : Limits ancestral AAs to those found in descendants [True]. (rje_ancseq.py)
- **xpass=X** : How many extra passes to make down tree after initial GASP [1]. (rje_ancseq.py)
- **indeltree=FILE** : output a text tree with indel data to **FILE** (gasp.py)

Tree and Sequence Options:

- **nsfin=FILE** : load NSF tree from **FILE** (rje_tree.py)
- **seqin=FILE** : load sequence list from **FILE** (rje_tree.py and rje_seq.py)
- **useanc=FILE**: load (ancestral) sequence list from GASP format **FILE** (rje_tree.py)
- **root=X** : Rooting of tree (rje_tree.py):
  - **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 **X** other than above)
- **deflen=X**: Default length for branches when no lengths given [0.1] (rje_tree.py)
- **rootbuffer=X**: Min. distance from node for root placement (percentage of branch length) [0.1] (rje_tree.py)

PAM Matrix options

- **pamfile=X** : Sets PAM1 input file [jones.pam] (rje_pam.py)
- **pammax=X** : Initial maximum PAM matrix to generate [100] (rje_pam.py)
- **pamcut=X** : Absolute maximum PAM matrix [1000] (rje_pam.py)

Standard Options (for my programs!):

- **v=X** : Sets verbosity level (rje.py)
- **i=X** : Sets interactivity level (rje.py)
- **log=FILE** : Redirect log to **FILE** [Default = calling_program.log] (rje.py)
- **newlog=T/F** : Create new log file. [Default = False: append log file] (rje.py)
- **win32=T/F** : Run in Win32 Mode [False] (rje.py)
- **help** : Prints module documentation to screen

3.3: Appendix III: Distributed Python Modules

3.3.1: gasp (Gapped Ancestral Sequence Prediction)

This module is purely for running the GASP algorithm as a standalone program. All the main functionality is encoded in the modules listed below. The GASP algorithm itself is now encoded in the rje_ancseq module. GASP may also be run interactively from the rje_tree module.

Uses general modules: os, re, sys, time

Uses RJE modules: rje, rje_ancseq, rje_pam, rje_seq, rje_tree

3.3.2: rje_ancseq (Ancestral Sequence Prediction Module)

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.

Classes:

- **Gasp(log=None,cmd_list=[],tree=None,ancfile='gasp'):**
```
- Handles main GASP algorithm.
>> log:Log = rje.Log object
>> cmd_list:List = List of commandline variables
>> tree:Tree = rje_tree.Tree Object
>> ancfile:str = output filename (basefile))
```
- **GaspNode(realnode,alphabet,log):**
```
- Used by Gasp Class to handle specific node data during GASP.
>> realnode:Node Object (rje_tree.py)
>> alphabet:list of amino acids for use in GASP
>> log:Log Object
```

Uses general modules: copy, sys, time

Uses RJE modules: rje, rje_pam

3.3.3: rje_tree (Phylogenetic Tree Module)

Reads in, edits and outputs phylogenetic trees. Executes duplication and subfamily determination. More details available in documentation for GASP and BADASP at http://www.bioinformatics.rcsi.ie/~redwards/

Classes:

- **Tree(rje.RJE_Object):**
```
- Phylogenetic Tree class.
```
- **Node(rje.RJE_Object):**
```
- Individual nodes (internal and leaves) for Tree object.
```
- **Branch(rje.RJE_Object):**
```
- Individual branches for Tree object.
```

Uses general modules: copy, random, re, string, sys, time

Uses RJE modules: rje, rje_ancseq, rje_seq

3.3.4: rje_seq (DNA/Protein sequence module)

Contains Classes and methods for sets of DNA and protein sequences. (Currently only protein sequences supported.)

Classes:

- **`SeqList(rje.RJE_Object):`**
  ```
  - Sequence List Class. Holds a list of Sequence Objects and
  has methods for manipulation etc.
  ```
- **`Sequence(rje.RJE_Object):`**
  ```
  - Individual Sequence Class.
  ```
- **`DisMatrix(rje.RJE_Object):`**
  ```
  - Sequence Distance Matrix Class.
  ```

Uses general modules: copy, os, random, re, sre_constants, string, sys, time

Uses RJE modules: rje, rje_blast, rje_pam

3.3.5: rje_pam (Contains Objects for PAM matrices)

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.)

Classes:

- **`PamCtrl(rje.RJE_Object):`**
  ```
  - Controls a set of PAM matrices.
  ```
- **`PAM(pam,rawpamp,alpha):`**
  ```
  - Individual PAM matrix.
  >> pam:int = PAM distance
  >> rawpamp:dictionary of substitution probabilities
  >> alpha:list of amino acids (alphabet)
  ```

Uses general modules: os, sys, time

Uses RJE modules: rje

3.3.6: rje (Contains General Objects for all my (Rich's) scripts)

General module containing Classes used by all my scripts plus a number of miscellaneous methods.
- Output to Screen, Commandline parameters and Log Files

Classes:

- **`Info(prog='Unknown',vers='X',edit='??/??/??',desc='Python script',author='Unknown',ptime=None):`**
  ```
  - Stores intro information for a program.
  >> prog:str = program name
  >> vers:str = version number
  >> edit:str = last edit date
  >> desc:str = program description
  >> author:str = author name
  >> ptime:float = starting time of program, time.time()
  ```
- **`Out(cmd=[]):`**
  ```
  - Handles basic generic output to screen based on Verbosity
  ```

```
and Interactivity
>> cmd:list = list of command-line arguments
```

- **Log(itime=time.time(),cmd_list=[]):**

  ```
  - Handles log output; printing to log file and error reporting
  >> itime:float = initiation time
  >> cmd_list:list of commandline variables
  ```

- **RJE_Object(log=None,cmd_list=[]):**

  ```
  - Metclass for inheritance by other classes.
  >> log:Log = rje.Log object
  >> cmd_list:List = List of commandline variables
  On intiation, this object:
  - sets the Log object
  - sets verbosity and interactive attributes
  - calls the _setAttributes() method to setup class attributes
  - calls the _cmdList() method to process relevant Commandline
  Parameters
  ```

Uses general modules: os, re, string, sys, time

3.3.7: rje_blast

This module is only included because it is needed to enable rje_tree.py to compile. (All these modules are part of a larger suite of programs.)

3.4: 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.

3.5: 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. (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 GASP**). 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.
- Make sure branch lengths are decimals and not plain integers. (i.e. 0.0 not 0)

3.6: Appendix VI: References

Edwards, R.J. and Shields, D.C. (2004) GASP: Gapped Ancestral Sequence Prediction for proteins, *BMC Bioinformatics*, **5**, 123.
Hall, T. (2001) BioEdit Sequence Alignment Editor for Windows 95/98/NT. *http://www.mbio.ncsu.edu/BioEdit/bioedit.html*
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.
Tamura, K. (2001) TreeExplorer manual. *http://evolgen.biol.metro-u.ac.jp/TE/TE_man.html*