Detecting overlapping coding sequences with pairwise alignments. (software notes)

Andrew E. Firth
Department of Biochemistry, University of Otago, PO Box 56, Dunedin, New Zealand

March 16, 2005

Contents

1 Introduction

In this document we describe the overlapping coding sequence (CDS) detection software used for the paper *Detecting overlapping coding sequences with pairwise alignments* (Firth & Brown, Bioinformatics, submitted 2004). Given a sequence alignment, this suite of software calculates various statistics that can be used as indicators of whether a region in the sequence is double-coding or single-coding. The software also generates simulated sequences assuming both single-coding and double-coding models, and uses the simulations to interpret the observed statistics.

You will need a computer running `LINUX` or `UNIX` and a `C++` compiler. You will also need to have `R` (`http://www.r-project.org/`) and, optionally, `LaTeX` installed if you want to use the provided scripts to produce plots.

2 Example

We shall demonstrate the programmes with the overlap between the S and P genes in a hepatitis B virus (HBV) sequence alignment. First get the file `double.tar.gz`, unpack it and compile the various programmes (replace '`g++`' by an appropriate alternative (e.g. `c++` or `gcc`) if you're using a different `C++` compiler):

```
> gunzip double.tar.gz
> tar xvf double.tar
> cd DOUBLE
> g++ -o calcprob.out calcprob.cxx
> g++ -o mlogd.out mlogd.cxx
> g++ -o mc_sim.out mc_sim.cxx
> g++ -o n123.out n123.cxx
> g++ -o runmean.out runmean.cxx
```

We will examine the different files in detail later on. First we run a quick example. The basic programmes are all collected together in the C-shell script `dostats`. Before running `dostats` you need to start a C-shell, set the stacksize, and make `dostats` executable:

```
> csh
> limit stacksize unlimited
> chmod 700 dostats
```

Now all you need to type is:

```
> ./dostats seqs.dat 422 1855 test
```

Here `seqs.dat` is a list of aligned HBV P-gene FASTA files. `422` and `1855` are the start and end points of the double-coding region (the S gene) and `test` is a prefix to identify all the output files (don't use '_'s or other 'special' characters). This will take a few minutes to run (about 12 minutes[1] on a LINUX PC with a 2.6 GHz Pentium[R] 4 processor). Ignore the '`rm:`' and '`null device`' error messages. If all goes well, you should end up with a lot of output files, including various tables of statistics and a lot of encapsulated postscript (`*.eps`) files that you can view with `gv` or `ggv`, e.g.

```
> ggv test.stats.eps
```

Included in these is the file `test.stats.eps` which is equivalent to Figure 9 in the paper. Several of the plots and a table of statistics are combined into a summary sheet `test.ps` (Figure 1).

If you don't have `LaTeX` you won't get the summary sheet `test.ps`. Also if you don't have `R` you won't get the `*.eps` files. In this case you may want to remove the last few lines from the `dostats` script so that you don't get error messages saying "Can't find `R`" etc.

[1] Note that most of this time is for the simulations required to interpret the N123 and NsNn statistics. These simulations are particularly costly for high sequence divergences. The MLOGD statistic (`mlogd.cxx`, §3.2) doesn't require simulations, and takes about four seconds to calculate for this data set.

3 In more detail...

3.1 dostats

The script `dostats` combines several different programmes which are described in more detail below. Several of the programmes are controlled by setup files. Normally you won't need to edit these if you're using the `dostats` script. Sometimes you may want to use the individual programmes, in which case you may want to explore some of the other options available in the setup files. In particular, if you want to quickly process a lot of overlaps, then just use `mlogd.cxx` to calculate the $\Delta_{\mathrm{s,d}}$ statistics and forget about the time-consuming simulations.

The basic command is:
```
> ./dostats sequencesfile orfstart orfend prefix
```

The file `sequencefile` is a list of FASTA files. These must be aligned and all of the same length. Accepted characters are a, c, g, t, u, A, C, G, T, U and ., - for gaps. The first sequence in `sequencesfile` is taken as the *reference sequence*. All pairwise comparisons are done between the reference sequence and one of the other sequences. If you want to do other pairwise comparisons then rerun `dostats` with a different reference sequence. The reference sequence must be in the *primary* read-frame, forward read-direction, and its length in nucleotides must be a multiple of three. Typically just feed in a complete known CDS (e.g. the P gene in the HBV example). Gaps should occur in groups of three in the primary frame. The requirements for the non-reference sequences are less strict (e.g. you might have only partial sequences filled out with '.'s, or indel sequencing errors).

The numbers `orfstart` and `orfend` refer to the first and last nucleotide positions of the *secondary* ORF. $|\texttt{orfstart} - \texttt{orfend}| + 1$ must be a multiple of three. If the secondary ORF overlaps the end(s) of the primary ORF, truncate `orfstart`, `orfend` so that they lie within the primary ORF. If the secondary ORF is in the reverse read-direction then make $\texttt{orfstart} > \texttt{orfend}$.

The current array sizes limit the maximum number of input sequences to 1000 (plus the reference sequence), the maximum sequence length to 6000 nucleotides, the maximum number of t steps (see §3.2) to 100, and the maximum number of simulated sequences (see §3.5) per run to 1000.

The basic procedure followed by `dostats` is as follows (see also Figure 2):

- Step through data sequence pairs in `sequencesfile`.
- Calculate the MLOGD, N123, NsNn statistics and divergence λ for each data sequence pair.
- For each data sequence pair, simulate 100 single-coding and 100 double-coding sequence pairs with the same sequence divergence λ. (Note that if $\lambda \sim 0.5$ or larger, then this can take a very long time.) Calculate the MLOGD, N123, NsNn statistics for each simulated pair in order to find the expected single-coding and double-coding distributions for the corresponding data sequence pair.
- Use the expected distributions to calculate error-bars and single-coding:double-coding likelihood ratios for the MLOGD, N123, NsNn statistics.
- Simulate 200 single-coding and 200 double-coding sequence pairs with random sequence divergences λ. Calculate the MLOGD, N123, NsNn statistics for each pair in order to find the typical single-coding and double-coding loci for each statistic as a function of λ.

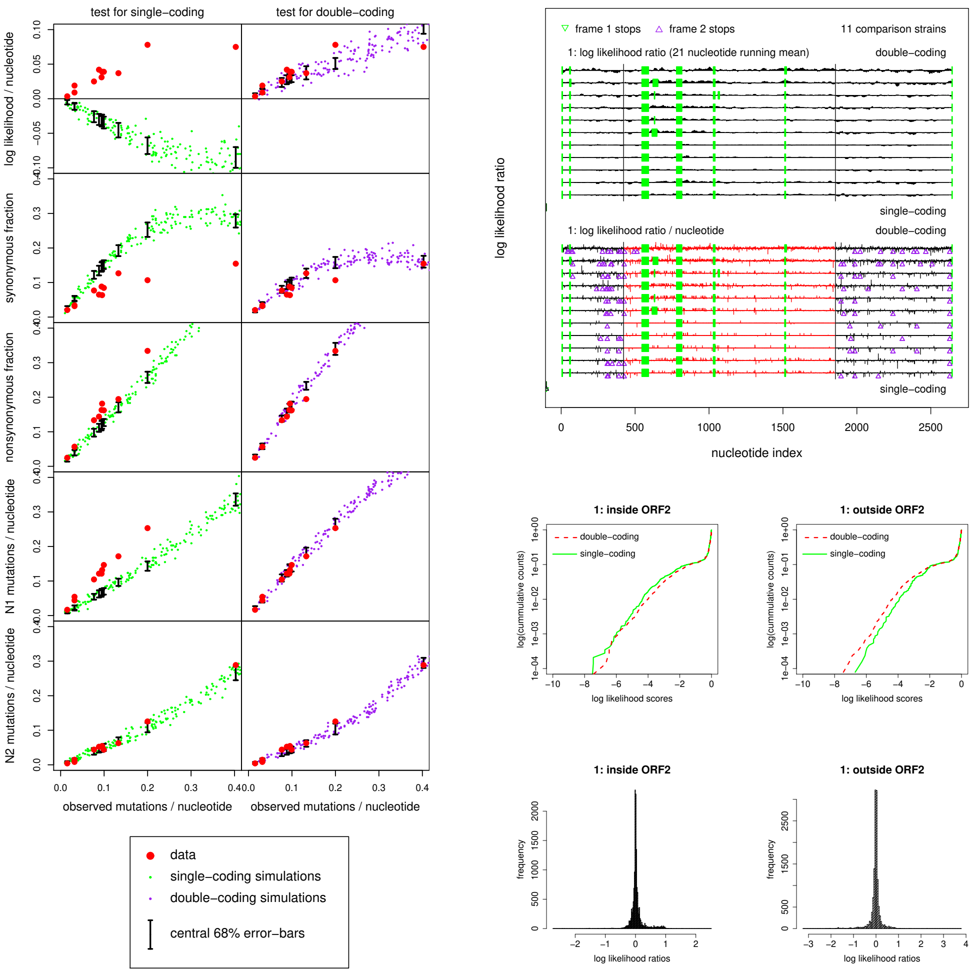

Table 1: Reference sequence = NC_003977. N_{ngap} is the number of non-gap nucleotides in the test region. λ is the *observed* mutation rate per nucleotide. $\Delta_{\mathrm{s,d}}$ is the MLOGD log likelihood ratio *per nucleotide*. Multiply by N_{ngap} to get the MLOGD log likelihood ratio for each sequence pair. f_{syn} and f_{non} are the NsNn synonymous and nonsynonymous mutation rates. f_{N1}, f_{N2} and f_{N3} are the N123 codon position mutation rates. The log likelihood ratios in the right-hand panel of the table are estimated by comparing the observed values with simulations. Positive values indicate double-coding while negative values indicate single-coding.

			raw statistics						log likelihood ratios					
sequence	N_{ngap}	λ	$\Delta_{\mathrm{s,d}}$	f_{syn}	f_{non}	f_{N1}	f_{N2}	f_{N3}	$\Delta_{\mathrm{s,d}}$	f_{syn}	f_{non}	f_{N1}	f_{N2}	f_{N3}
X70185	1317	0.077	0.025	0.08	0.13	0.10	0.04	0.07	21.3	7.7	5.7	14.5	-0.8	12.3
D00329	1317	0.094	0.031	0.09	0.16	0.12	0.05	0.09	33.7	8.7	6.0	12.0	-0.8	13.0
X75665	1317	0.032	0.009	0.03	0.06	0.04	0.01	0.03	9.0	3.8	2.4	5.4	-0.0	3.2
AB074755	1317	0.032	0.019	0.03	0.06	0.05	0.01	0.03	20.8	3.1	2.6	16.1	0.4	5.6
D50520	1317	0.015	0.004	0.02	0.02	0.02	0.00	0.02	3.2	0.6	0.6	-0.0	0.2	-1.0
X02496	1281	0.088	0.042	0.06	0.14	0.12	0.05	0.07	36.6	10.4	2.7	22.5	-0.9	17.4
X75664	1311	0.095	0.039	0.06	0.18	0.13	0.05	0.08	28.4	15.5	11.9	21.3	-0.7	19.7
X75663	1317	0.133	0.037	0.13	0.19	0.17	0.06	0.14	33.2	9.1	-3.2	25.9	0.3	12.1
AB056514	1299	0.099	0.039	0.08	0.16	0.15	0.04	0.10	39.6	12.4	4.8	28.2	0.4	13.1
AF046996	1281	0.200	0.078	0.11	0.33	0.25	0.13	0.17	75.0	19.5	10.2	29.7	-0.8	27.8
J02442	1311	0.403	0.075	0.15	0.52	0.44	0.29	0.40	55.2	19.9	-5.4	21.2	1.0	22.6

Figure 1: The summary sheet for the S and P overlap in HBV. The electronic version is in colour.

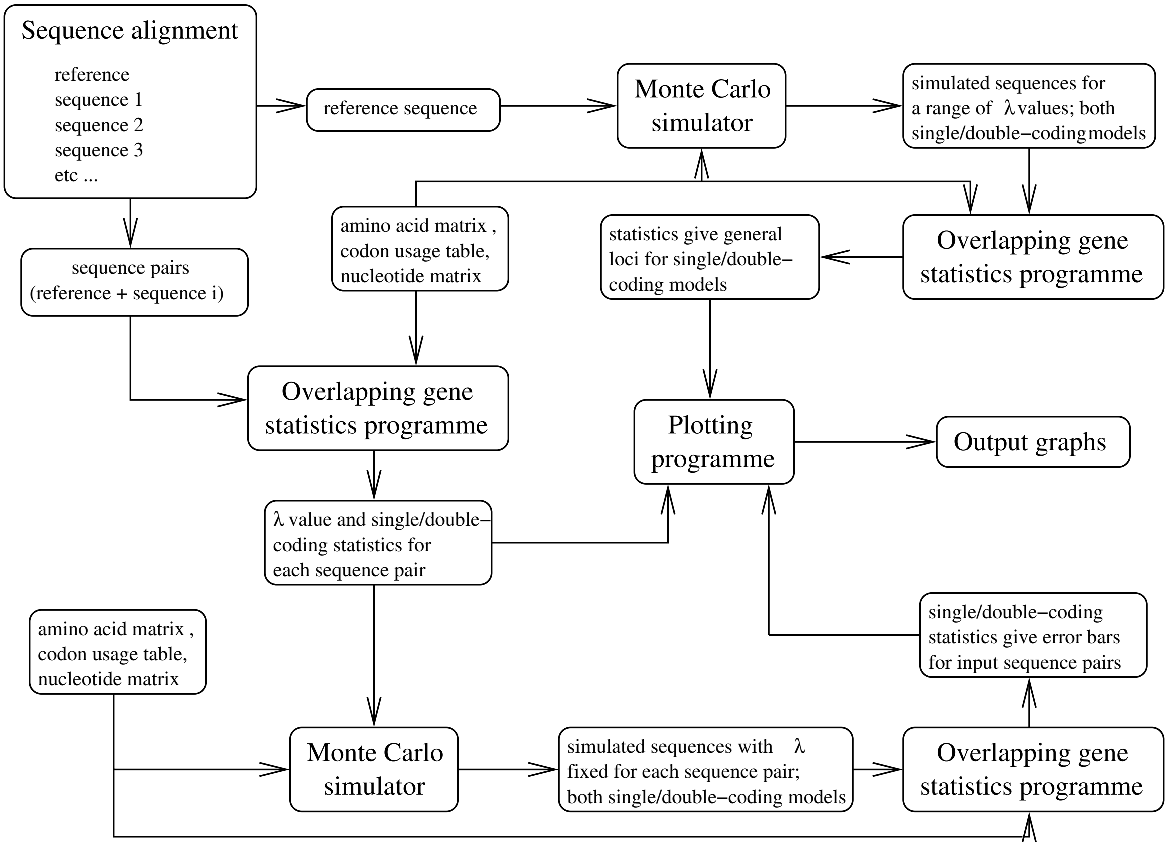

Figure 2: Flowchart for `dostats` script.

Input files	

`mlogd.out`
`n123.out`
`calcprob.out`
`mc_sim.out`
`runmean.out`
	Component programmes (see §3.2, §3.3, §3.4, §3.5, §3.6 respectively).

`setup.mlogd`
`setup.mc_sim`
`setup.mc_sim_eb`
`setup.n123`
	Setup files for `mlogd.out`, `mc_sim.out`, `n123.out`. `setup.mc_sim_eb` controls the simulations for the error-bars and N123, NsNn log likelihood ratios, while `setup.mc_sim` controls the simulations for the general loci.

`aa2codon.dat`
`aa.dat`
`codon.dat`
`nuc.dat`
	Genetic code, amino acid, codon usage and nucleotide matrices. You can replace these with different matrices and/or edit the setup files to use matrices with different names.

sequences file	File containing list of input sequences. Specified as argument to `dostats`.
FASTA files	Files specified in sequences file.
`plots.R` `plots.tex`	Template files (see §3.7 and §3.8 respectively).

Output files	
prefix.pstats	Output file from n123.cxx (see §3.3).
prefix.stats1 *prefix*.stats2 *prefix*.MLElog1 *prefix*.MLElog2 *prefix*.gaplog1 *prefix*.gaplog2 *prefix*.stoplog1 *prefix*.stoplog2 *prefix*.glitches1 *prefix*.glitches2	Output files from mlogd.cxx (see §3.2). 1 ⇒ statistics within the secondary ORF. 2 ⇒ statistics in the primary ORF but outside the secondary ORF (for comparison).
prefix.pstats_s.mc *prefix*.pstats_d.mc *prefix*.stats1_s.mc *prefix*.stats1_d.mc	Basic statistics files from n123.cxx and mlogd.cxx for the simulations. Used for the loci on the *prefix*.stats.eps plot. s ⇒ simulations assuming a single-coding model. d ⇒ simulations assuming a double-coding model.
prefix.pstats_s.mc.* *prefix*.pstats_d.mc.* *prefix*.stats_s.mc.* *prefix*.stats_d.mc.*	Basic statistics files from n123.cxx and mlogd.cxx for the simulations. Used for the error-bars on the *prefix*.stats.eps plot.
prefix.probs	single-coding:double-coding log likelihood ratios for each sequence pair (see §3.4).
prefix.errorbars_sl *prefix*.errorbars_sm *prefix*.errorbars_su *prefix*.errorbars_dl *prefix*.errorbars_dm *prefix*.errorbars_du	Lower, median and upper (68%) percentiles for λ, $\Delta_{\mathrm{s,d}}$, f_{syn}, f_{non}, f_{N1}, f_{N2}, f_{N3}, for single-coding and double-coding simulations with sequence divergence fixed to that for the corresponding data sequence pair.
prefix.cum1d *prefix*.cum1s *prefix*.cum2d *prefix*.cum2s	Cumulative frequencies of nucleotide-by-nucleotide log probabilities. Used for the *prefix*.MLEcum?.eps plots. 1/2 ⇒ inside/outside the secondary ORF; s/d ⇒ single/double-coding model.
prefix.runmean	Running mean in a sliding-window for nucleotide-by-nucleotide $\Delta_{\mathrm{s,d}}$ values. Used for the *prefix*.MLEnuc.eps plot.
prefix.s_stoplog.temp *prefix*.d_stoplog.temp	Positions of STOP codons. Used for the *prefix*.MLEnuc.eps plot.
prefix.R	Script for producing plots (see §3.7).
prefix.stats.eps *prefix*.labels.eps *prefix*.MLEnuc.eps *prefix*.MLEcum1.eps *prefix*.MLEcum2.eps *prefix*.MLEhist1.eps *prefix*.MLEhist2.eps	Plots outputted from *prefix*.R (see §3.7; Figure 1).
prefix.tex	Script for producing LaTeX file combining plots (see §3.8).
prefix.ps	Output of *prefix*.tex (see §3.8; Figure 1).

3.2 mlogd.cxx

This is the core programme – it calculates the MLOGD statistic $\Delta_{\mathrm{s,d}}$ and the NsNn statistics f_{syn}, f_{non} for a given sequence alignment and secondary ORF. In `dostats` the programme is called twice – once to calculate statistics inside the secondary ORF and again to calculate statistics outside the secondary ORF (for comparison). It is also called to calculate statistics for all the simulated sequence pairs. `mlogd.cxx` reads setup parameters from the file `setup.mlogd`. If you are using the `dostats` script, it will automatically edit the variables `yyy` (file listing input sequences), `xxx` (sequence region to use for calculating the pairwise sequence divergence values λ), and `orf1`, `orf2` (secondary ORF boundaries) in the setup file, run the programme, and tidy up all the output files. However you may want to change the default amino acid, codon and nucleotide matrices or some of the other parameters. Alternatively you can run `mlogd.cxx` as a stand-alone programme. This is especially useful if you have a lot of overlaps to process and are only interested in calculating the $\Delta_{\mathrm{s,d}}$ statistic. In this case you will need to fill in these details yourself and then type:

```
> ./mlogd.out setup.mlogd
```

Notes:

- `aa.dat`, `nuc.dat`, `codon.dat` and `aa2codon.dat` are respectively the amino acid matrix, nucleotide matrix, codon usage table and genetic code table (see §5).
- The input sequences should be in the FASTA format (see examples for HBV). These must be aligned and all of the same length. Accepted characters are a, c, g, t, u, A, C, G, T, U and ., - for gaps.
- The first sequence in the input sequences file is taken as the reference sequence. All pairwise comparisons are done between the reference sequence and one of the other sequences. If you want to do other pairwise comparisons then rerun with a different reference sequence.
- The reference sequence must be in the primary frame, forward read-direction, and its length in nucleotides must be a multiple of three. Typically just feed in a complete known CDS (e.g. the P gene in the HBV example). Gaps should occur in groups of three in the primary frame. The requirements for the non-reference sequences are less strict (e.g. you might have only partial sequences filled out with '.'s).
- The boundaries of the secondary ORF must be contained within the input sequence and $|$`orfstart`$-$`orfend`$|+1$ must be a multiple of three. If the secondary ORF is in the reverse read-direction then make `orfstart` $>$ `orfend`. If the secondary ORF overlaps the end(s) of the primary ORF, truncate `orfstart`, `orfend` so that they lie within the primary ORF. If the secondary ORF is in the same frame as the primary sequence then only the single-coding model is used.
- The first and last codons (in general a START and a STOP) are automatically excluded.
- For each sequence pair, the likelihood for each of the single-coding and double-coding models is maximized with respect to the parameter t, where $\mathbf{P}(t) = \exp(\mathbf{Q}t)$ (t is monotonically related to λ). You can vary the range of t values to test (between 0 and ∞). You can also vary the number of logarithmic steps in t. In general, the default range should be fine.
- There are four options for calculating t.
  1. Calculate separate t values for the single-coding and double-coding models. Calculate t inside the secondary ORF assuming a single-coding model or a double-coding model, as appropriate.
  2. Calculate t outside the secondary ORF assuming a single-coding model. Use this t value within the secondary ORF for both the single-coding and double-coding models.
  3. Specify a range (generally outside the secondary ORF) within which to calculate t, assuming a single-coding model. Use this t value within the secondary ORF for both the single-coding and double-coding models.
  4. As for 1., except for the region outside the secondary ORF. (You can use this mode to calculate background statistics outside the secondary ORF for comparison with the statistics calculated inside the secondary ORF.)

- There is an option to output `glitches` files. These files (one for the single-coding and one for the double-coding model) are logs of all nucleotide mutations whose probability is less than a given value ϵ (in the setup file). They may be of some use in tracking alignment or sequencing errors. However the probabilities are for the best-fitting t value, which would change if any such glitches were removed. So if you identify a glitch, you will then need to rerun `mlogd.cxx` with the glitch 'corrected'. I haven't used this option much – it seems to pick up many more false positives than real sequencing errors.

- In the amino acid matrix, the acceptabilities of mutations between STOPs and non-STOPs are set to zero. If such a mutation turns up (or any other mutation that results in a zero probability score) then the log probability is undefined. Instead of adding this to the sum-over-nucleotides log probability, such nucleotides are just logged in the `stoplog` files. You should check that these files are empty or, equivalently, check the *prefix*`.MLEnuc.eps` plot. Any nucleotide discarded because of this is omitted equally from the single-coding and double-coding log probability sums (so as not to skew results) and the final $\Delta_{\mathrm{s,d}}$ *per nucleotide* and f_{syn}, f_{non} values take into account the reduced number of nucleotides used.

- Gaps are similarly discarded and logged in the `gaplog` files. Not only are gap nucleotide positions discarded, but so are any nucleotides where either the primary read-frame or secondary read-frame codons containing that nucleotide include gap characters. The same nucleotides are omitted equally from the single-coding and double-coding log probability sums.

- You can run `mlogd.cxx` in verbose mode by editing the line '`int verbose = 0;`' to '`int verbose = 1;`' and recompiling.

Output files	
`mlogd_out.dat` (*prefix*.`stats*`)	The main statistics file. One row for each pairwise sequence comparison. Columns are 1. Best-fitting t value used for single-coding model. 2. Best-fitting t value used for double-coding model. 3. Mean log likelihood per nucleotide for single-coding model. 4. Mean log likelihood per nucleotide for double-coding model. 5. Number of nucleotides used (i.e. not in `gaplog` or `stoplog` files). 6. Number of nucleotides discarded due to zero probability in primary read-frame. 7. Number of additional nucleotides discarded due to zero probability in secondary read-frame. 8. Number of nucleotides discarded as gaps or adjacent to gaps. 9. Number of null codon mutations in primary read-frame. 10. Number of synonymous non-null codon mutations in primary read-frame. 11. Number of nonsynonymous codon mutations in primary read-frame. 12. Total number of non-gap nucleotides (columns $5+6+7$). 13. Total number of point nucleotide differences between the two sequences (column 13 / column 12 gives the sequence divergence λ). Note that $3\times$ (columns $9+10+11$) and (columns $5+6+7$) are similar but may differ slightly depending on the positioning of gaps.
`mlogd_out.MLElog`	A record of all the nucleotide-by-nucleotide log probabilities. Columns are sequence 1 ID, sequence 2 ID, nucleotide position, log probability (single-coding model), log probability (double-coding model).
`mlogd_out.nseq0`	A little file – just the total sequence length. Used by `dostats`.
`mlogd_out.stoplog`	A record of discarded zero probability mutations (see above). Columns are sequence 1 ID, sequence 2 ID, nucleotide position, read-frame (0 = primary, 1 = secondary).
`mlogd_out.gaplog`	A record of discarded gap or adjacent-to-gap nucleotides (see above). Columns are sequence 1 ID, sequence 2 ID, nucleotide position.
`mlogd_out.glitches`	A record of mutations with probability less than some value ϵ (see above). These nucleotides are not, however, discarded. Columns are sequence 1 ID, sequence 2 ID, nucleotide position, read-frame (0 = primary, 1 = secondary), log probability.

3.3 n123.cxx

This simple programme calculates the N123 statistics f_{N1}, f_{N2}, f_{N3} both inside and outside the secondary ORF, for a given sequence alignment and secondary ORF. `n123.cxx` reads setup parameters from the file `setup.n123`. If you are using the `dostats` script, it will automatically edit the variables `yyy` (file listing input sequences) and `orf1`, `orf2` (secondary ORF boundaries) in the setup file, run the programme, and tidy up all the output files. If you want to run `n123.cxx` as a stand-alone programme, you will need to fill in these details yourself and then type:

```
> ./n123.out setup.n123
```

Notes:

- The input sequences should be in the FASTA format (see examples for HBV). These must be aligned and all of the same length. Accepted characters are a, c, g, t, u, A, C, G, T, U and ., - for gaps.
- The first sequence in the input sequences file is taken as the reference sequence. All pairwise comparisons are done between the reference sequence and one of the other sequences. If you want to do other pairwise comparisons then rerun with a different reference sequence.
- The reference sequence must be in the primary frame, forward read-direction, and its length in nucleotides must be a multiple of three. Gaps should occur in groups of three in the primary frame. Typically just feed in a complete known CDS (e.g. the P gene in the HBV example). The requirements for the non-reference sequences are less strict (e.g. you might have only partial sequences filled out with '.'s).
- The boundaries of the secondary ORF must be contained within the input sequence and $|$`orfstart`$-$`orfend`$|+1$ must be a multiple of three. If the secondary ORF is in the reverse read-direction then make `orfstart` $>$ `orfend`. If the secondary ORF overlaps the end(s) of the primary ORF, truncate `orfstart`, `orfend` so that they lie within the primary ORF.
- Nucleotides corresponding to gaps in either sequence are discarded, and the f_{N1}, f_{N2}, f_{N3} values take into account the reduced number of nucleotides used.
- You can run `n123.cxx` in verbose mode by editing the line '`int verbose = 0;`' to '`int verbose = 1;`' and recompiling.

Output file	
`n123.dat` (*prefix*`.pstats*`)	Output statistics. One row for each pairwise sequence comparison. Columns are 1. sequence 1 ID 2. sequence 2 ID 3. f_{N1} outside secondary ORF 4. f_{N2} outside secondary ORF 5. f_{N3} outside secondary ORF 6. f_{N1} inside secondary ORF 7. f_{N2} inside secondary ORF 8. f_{N3} inside secondary ORF

3.4 calcprob.cxx

A little programme for calculating the likelihood ratio

$$\log\left[\frac{P(\text{observations}|\text{double}-\text{coding})}{P(\text{observations}|\text{single}-\text{coding})}\right] \tag{1}$$

for the statistics $\Delta_{\rm s,d}$, $f_{\rm N1}$, $f_{\rm N2}$, $f_{\rm N3}$, $f_{\rm syn}$, $f_{\rm non}$ for each pairwise sequence comparison. It assumes that the statistics $\Delta_{\rm s,d}$, $f_{\rm N1}$, $f_{\rm N2}$, $f_{\rm N3}$, $f_{\rm syn}$, $f_{\rm non}$ are normally distributed and estimates μ, σ for each statistic, for each of the single-coding and double-coding models, from simulations. Edit this programme if you want to transform the data first (e.g. $p' = \arcsin\sqrt{p}$ transform). Note that the calculation for $\Delta_{\rm s,d}$ is rather redundant since $\Delta_{\rm s,d}$ is itself, more-or-less, a log likelihood ratio.

Input files	
`calcprob.obs.dat`	$\Delta_{\rm s,d}$, $f_{\rm syn}$, $f_{\rm non}$, $f_{\rm N1}$, $f_{\rm N2}$, $f_{\rm N3}$ statistics for a particular pairwise comparison.
`calcprob.mcs.dat` `calcprob.mcd.dat`	$\Delta_{\rm s,d}$, $f_{\rm syn}$, $f_{\rm non}$, $f_{\rm N1}$, $f_{\rm N2}$, $f_{\rm N3}$ statistics for single-coding and double-coding simulations with the same sequence divergence and starting sequence as the data.
Output file	
`calcprob.out.dat` (*prefix*.`probs`)	Single-coding:double-coding log likelihood ratios for $\Delta_{\rm s,d}$, $f_{\rm syn}$, $f_{\rm non}$, $f_{\rm N1}$, $f_{\rm N2}$, $f_{\rm N3}$.

3.5 mc_sim.cxx

This is the Monte Carlo simulation programme. It uses the setup file `setup.mc_sim` or `setup.mc_sim_eb`. If you are using the `dostats` script, it will automatically edit the variables `yyy` (reference sequence name), `zzz` (other sequence name), `orf1` and `orf2` (secondary ORF boundaries), `sss` (force single-coding model option), `ttt` (mutation rate or divergence λ) and `iii` (random seed) in the setup files, run the simulations, calculate the statistics, and delete all the useless files at the end. However you may want to change the number of simulations, the range of λ values or the default amino acid, codon and nucleotide matrices. Alternatively you can run `mc_sim.cxx` as a stand-alone programme in which case you will need to fill in these details yourself and then type:

```
> ./mc_sim.out setup.mc_sim
```

Notes:

- The programme starts with an initial sequence (sequence 1) – generally the reference sequence. Random mutations are added sequentially using the nucleotide matrix, and tested for acceptability using the codon and amino acid matrices (in both frames for the double-coding model).
- If sequence 1 has gaps then these are stripped out (so that codon/amino acid weighting can be applied correctly at sites adjacent to gaps). The gaps are added back in to all output simulated sequences.
- Another sequence (sequence 2) may be entered and is used only to match the treatment of gaps in the simulations to the treatment of gaps in the data (see below). The sequence 2 gaps are also included in the output sequences.
- Gaps in sequences 1 and 2 should occur in multiples of three in the primary frame. Otherwise, when gaps are stripped out, the sequence will be put out of frame.
- Sequences 1 and 2 must be the same length (including gaps), in the primary read-frame, forward read-direction, and their length in nucleotides must be a multiple of three.
- The boundaries of the secondary ORF must be contained within the input sequence and |`orfstart`−`orfend`| + 1 must be a multiple of three. If the secondary ORF is in the reverse read-direction then make `orfstart` > `orfend`. If the secondary ORF is in the same frame as the primary sequence, then a single-coding model is used in the simulations.
- `aa.dat`, `nuc.dat`, `codon.dat` and `aa2codon.dat` are respectively the amino acid matrix, nucleotide matrix, codon usage table and genetic code table (see §5).

- Each simulation is terminated when the number of nucleotide mutations reaches some predetermined value x. For each simulation, x is chosen randomly from the given range. If you want to fix x then set the upper range boundary equal to the lower range boundary.

- There are several options for counting the number of mutations. First, the input range can be a proportion (i.e. in the range 0 to 1), or an exact number of nucleotides (i.e. a whole number). Second, the mutations may be counted on the whole input sequence, or just within the secondary ORF. Third, the programme may count all accepted mutations, or just the observed ones (e.g. A $\rightarrow$ C $\rightarrow$ A counts as two accepted mutations but zero observed mutations).

- If the required number of observed mutations is too high, then the programme can take a very long time to run (since, as the number of mutations increases, further mutations tend to just 'undo' previous ones).

- In order to exactly match the simulations to a data sequence pair (as viewed by the `mlogd.cxx` programme), observed mutations are not counted within or adjacent to the gaps of sequence 2, or adjacent to the stripped-out gaps of sequence 1. Such mutations are applied (since they may affect the acceptabilities of mutations at adjacent sites that are not within or adjacent to gaps), but are not counted towards the total number of mutations.

- In the `dostats` output plot *prefix*`.stats.eps`, the errorbars are derived from simulations that are exactly matched to the input sequences in terms of the number of observed nucleotide differences between sequence pairs. STOP codons and codons containing gaps, in either the primary or secondary frames, are omitted from the count. To accomplish this, the single-coding simulations need to be aware of the secondary frame, even though only single-coding transition probabilities are used. The 'force single-coding model' option allows you to do this. Enter the boundaries of the secondary ORF for `orfstart` and `orfend`.

- The simulated sequences are written to the files *prefix*`.1`, ..., *prefix*`.`*nsims*, where *nsims* is the total number of simulations. The original sequence is also rewritten to the file *prefix*`.org`. Edit the prefix for the output simulated sequences as you like. However don't edit the prefix if you are using the `dostats` script.

- Edit the total number of simulations as you like.

- The random seed is used in the random number generator and should be a positive integer.

- You can run `mc_sim.cxx` in verbose mode by editing the line '`int verbose = 0;`' to '`int verbose = 1;`' and recompiling.

3.6 runmean.cxx

A little programme to calculate running means for the sliding-window nucleotide-by-nucleotide plots of various statistics. There's probably something better in `R` itself, but I haven't found it yet. Reads input file and window half-width from command line and writes to standard output.

3.7 plots.R

This is just a script for producing some useful plots in `R`. The script contains the variables `xxx`, `yyy`, `sss`, `orf1` and `orf2` which `dostats` replaces with the prefix, number of sequences, sequence length and secondary ORF boundaries, to produce the plotting script *prefix*`.R` that you actually use. The plots are in colour but are designed to be legible (more-or-less) in black and white also.

Output plots	
prefix.`stats.eps`	Combined plots of $\Delta_{\mathrm{s,d}}$, f_{syn}, f_{non}, f_{N1}, f_{N2} versus sequence divergence λ. Points are plotted for each data sequence pairwise comparison. Error-bars and general single-coding and double-coding loci are added using the simulations.
prefix.`labels.eps`	The key to the plot symbols used in *prefix*.`stats.eps`.
prefix.`MLEnuc.eps`	A nucleotide-by-nucleotide plot of the $\Delta_{\mathrm{s,d}}$ values over the whole input sequence. The single-coding and double-coding models are best-fitted (with respect to t) independently both inside and outside the secondary ORF. Because t has to be fitted for a whole ORF before $\Delta_{\mathrm{s,d}}$ is calculated, it is not straightforward to produce a whole-genome six-frame plot of the $\Delta_{\mathrm{s,d}}$ statistic as a gene-finder in the same way as one might do for various codon-usage statistics in single-coding sequences. Gaps (including gaps in the plotted sequence, nucleotides aligned to gaps in the reference sequence, and nucleotides in primary or secondary frame codons that overlap such gaps) are indicated by blanked-out regions. Transitions between STOP and non-STOP codons in the primary and secondary read-frames are indicated by triangles (typically you should expect non in the overlap region and lots of secondary read-frame STOPs outside the overlap region). Note that transitions between gaps and non-gaps and between STOPs and non-STOPs – between the plotted sequence and the reference sequence – are plotted, rather than gaps and STOPs *per se*, so *prefix*.`MLEnuc.eps` is not an ORF-finder. The upper half of the figure shows plots with a 21-nucleotide sliding-window mean filter applied. The sliding-window skips straight over gaps.
prefix.`MLEcum1.eps` *prefix*.`MLEcum2.eps`	Cumulative distributions of single-coding and double-coding log probabilities (all sequence pairs). 1 $\Rightarrow$ statistics within the secondary ORF. 2 $\Rightarrow$ statistics in the primary ORF but outside the secondary ORF (for comparison).
prefix.`MLEhist1.eps` *prefix*.`MLEhist2.eps`	Histograms of $\Delta_{\mathrm{s,d}}$ values (all sequence pairs). 1 $\Rightarrow$ statistics within the secondary ORF. 2 $\Rightarrow$ statistics in the primary ORF but outside the secondary ORF (for comparison).

3.8 plots.tex

This is just a script for combining some useful plots into a summary sheet (Figure 1 shows an example for the S and P overlap in HBV). The script contains the variables `xxx` and `nnn` which `dostats` replaces with the prefix and the name of the reference sequence. It also adds a table of sequence statistics. These are all put in the `LaTeX` script *prefix*.`tex` that you actually use.

4 Included files

The following is a brief description of the files included with the distribution:

`dostats`	Master script.
`mlogd.cxx`	Calculates MLOGD and NsNn statistics.
`n123.cxx`	Calculates N123 statistics.
`calcprob.cxx`	Calculates single-coding:double-coding likelihood ratios.
`mc_sim.cxx`	Generates Monte Carlo simulated sequences.
`runmean.cxx`	Calculates running mean for nucleotide-by-nucleotide plots.
`plots.R`	`R` plotting script template.
`plots.tex`	`LaTeX` script template.
`setup.mlogd`	Setup parameters for `mlogd.cxx`.
`setup.n123`	Setup parameters for `n123.cxx`.
`setup.mc_sim`	Setup parameters for `mc_sim.cxx` (loci run).
`setup.mc_sim_eb`	Setup parameters for `mc_sim.cxx` (error-bars run).
`aa.dat`	Default amino acid weight matrix.
`codon.dat`	Default codon usage table.
`nuc.dat`	Default nucleotide matrix.
`aa2codon.dat`	Genetic code.
`seqs.dat`	Example sequence list for HBV.
`*.fasta`	Example aligned FASTA files for HBV P gene.
`make_blosum.cxx`	Programme for making new amino acid matrices.

5 Nucleotide, codon, amino acid and genetic code matrices

5.1 Nucleotide matrix

The programmes use a general reversible model (Yang 1994) for the nucleotide matrix $\mathbf{Q}$, i.e.

$$\mathbf{Q} = \begin{bmatrix} - & a.\pi_C & b.\pi_A & c.\pi_G \\ a.\pi_U & - & d.\pi_A & e.\pi_G \\ b.\pi_U & d.\pi_C & - & f.\pi_G \\ c.\pi_U & e.\pi_C & f.\pi_A & - \end{bmatrix}, \tag{2}$$

where a, b, c, d, e, f are rate parameters, the π_j are the equilibrium nucleotide frequencies, the order of the rows and columns is U, C, A, G, and the diagonals are set so that

$$q_{ii} = -\sum_{j \neq i} q_{ij}. \tag{3}$$

In the paper we use a $\kappa = 3$ Kimura (1980) model, i.e. $\pi_U = \pi_C = \pi_A = \pi_G$ and $a = f = 3b = 3c = 3d = 3e$. Before using $\mathbf{Q}$ as input to the programmes `mlogd.cxx` and `mc_sim.cxx`, or `dostats`, we need to diagonalize it. See the file `nuc.dat` for an example. This file contains four matrices. The first is $\mathbf{Q}$ (normalized so that the row sums are zero and $-\sum_i \pi_i.Q_{ii} = 1$). The next three matrices are $\mathbf{U}^{-1}$, $\mathbf{D}$, $\mathbf{U}$, where $\mathbf{D}$ is diagonal and $\mathbf{Q} = \mathbf{U}^{-1}\mathbf{D}\mathbf{U}$. In general, if you have the a, b, c, d, e, f parameters, you can make a $\mathbf{Q}$ matrix that conserves relative nucleotide frequencies for a particular gene or species by multiplying these values by the π_j values for that gene or species.

5.2 Codon usage matrix

The codon usage tables (CUTs) used by the software are relative frequencies, multiplied by the degeneracy of the corresponding amino acid (e.g. the score for AUG (*met*) is multiplied by 1 while the score for CUG (*leu*) is multiplied by 6), and divided by the π_j for the particular species or msa (e.g. the score for CUG is divided by $\pi_C.\pi_U.\pi_G$). The best way to combine codon usage frequencies with amino acid matrices and nucleotide matrices is not very clear: our procedure does not, for example, properly take into account the hypermutability of GC dinucleotides. You may prefer to use a null CUT (i.e. equal codon frequencies), as in fact we did in the paper.

The EMBOSS (Rice *et al.* 2000) format for CUTs comprises five columns:

1. 3-letter codon code,
2. 1-letter amino acid code,
3. relative frequencies (sum to 1 for each amino acid),
4. absolute frequencies (total for all codons sums to 1000),
5. raw codon counts.

The following simple commands can be used to produce new CUTs. E.g. for the EMBOSS CUT, `Ehum.cut`:

Sort the CUT into the order and code used by `dostats` and its component programmes:
```
awk '{print $1,$3,$4}' Ehum.cut | sed 's/T/a/g' | sed 's/U/a/g' | \
  sed 's/C/b/g' | sed 's/A/c/g' | sed 's/G/d/g' | sort | sed 's/a/U/g' | \
  sed 's/b/C/g' | sed 's/c/A/g' | sed 's/d/G/g' > Ehum2.cut
```

Null CUT:
```
awk '{print $1,1./64.}' Ehum2.cut > null.cut
```

Absolute CUT:
```
awk '{print $1,$3/1000.}' Ehum2.cut > abs.cut
```

Relative CUT:
```
awk '{print $1,$2}' Ehum2.cut > rel.cut
```

Relative CUT scaled by degeneracy for each amino acid:
```
csh
touch rel2.cut
sed 's/\*/Z/' aa2codon.dat > temp0
awk '{print $1}' temp0 > temp2
foreach i (`awk '{print $1}' rel.cut`)
  set aa = `grep $i temp0 | awk '{print $1}'`
  set freq = `grep $i rel.cut | awk '{print $2}'`
  set degen = `grep $aa temp2 | wc | awk '{print $1}'`
  echo `echo $i $freq $degen | awk '{print $1,$2*$3}'` >> rel2.cut
end
exit
```

CUT divided by $\pi_i.\pi_j.\pi_k$ (in this example, $\pi_U = 0.22$, $\pi_C = 0.26$, $\pi_A = 0.26$, $\pi_G = 0.26$):
```
cat rel2.cut | sed 's/U/0.22 /g' | sed 's/C/0.26 /g' | sed 's/A/0.26 /g' | \
  sed 's/G/0.26 /g' | awk '{print $4/$1/$2/$3/64.}' > temp1
paste rel2.cut temp1 | awk '{print $1,$3}' > rel3.cut
```

Then copy the relevant `*.cut` file to `codon.dat`.

Alternatively you can get CUTs from `www.kazusa.or.jp/codon/index.html` or `http://uther.otago.ac.nz/Transterm.html`. The former CUTs are in a rectangular array format. For each amino acid, the CUT gives the 3-letter codon code, the absolute frequencies (total for all codons sums to 1000) and the raw codon counts. You can use the following commands to obtain `abs.cut` and `rel.cut`, and then the above commands to obtain `rel2.cut` and `rel3.cut`.

```
cat kazusa.cut | sed 's/(/ /g' | sed 's/)/ /g' | \
  awk '{printf "%s %s\n%s %s\n%s %s\n%s %s\n",$1,$2,$4,$5,$7,$8,$10,$11}' | \
  grep "[UCAGTucagt]" | \
  sed 's/T/a/g' | sed 's/U/a/g' | sed 's/C/b/g' | sed 's/A/c/g' | \
  sed 's/G/d/g' | sort | \
  sed 's/a/U/g' | sed 's/b/C/g' | sed 's/c/A/g' | sed 's/d/G/g' | \
  awk '{print $1,$2/1000.}' > abs.cut
```

```
csh
touch rel.cut
sed 's/\*/Z/' aa2codon.dat > temp0
paste temp0 abs.cut | awk '{print $1,$5}' > temp2
foreach i (`awk '{print $1}' abs.cut`)
  set aa = `grep $i temp0 | awk '{print $1}'`
  set absfreq = `grep $i abs.cut | awk '{print $2}'`
  set aaabsfreq = `grep $aa temp2 | awk '{sum+=$2}END{print sum}'`
  echo `echo $i $absfreq $aaabsfreq | awk '{print $1,$2/$3}'` >> rel.cut
end
rm temp0 temp2
exit
```

Note that the CUT normalization is unimportant.

5.3 Amino acid matrix

In the paper we use the BLOSUM40 matrix to derive the amino acid matrix. The BLOSUM40 matrix is for very divergent sequences, so it maximizes the amino acid acceptability component relative to the nucleotide mutation probability component. This is best for our software because we deal with the latter separately with the nucleotide matrix. If you want to make amino acid matrices using different BLOSUM matrices (which you can ftp from `ncbi.nlm.nih.gov`) then the programme `make_blosum.cxx` will help.

I used the frequencies matrices in the `blosum??.out` files. Cut off all the headers so that the first line contains the amino acid one-letter codes (A, R, N etc.) and the next 20 lines are a lower triangular matrix of frequency values. `make_blosum.cxx` converts these values to q_{ij}/e_{ij} values (see Henikoff & Henikoff 1992 for more details).

```
g++ -o make_blosum.out make_blosum.cxx
make_blosum.out inputmatrix aa.dat
```

Note that the normalization of the amino acids matrix is unimportant.

5.4 Genetic code table

The table `aa2codon.dat` is used to convert the amino acid one-letter codes in the amino acid matrix `aa.dat` into codons. The programmes only read the first column of `aa2codon.dat`, so they rely on you maintaining the codon order given in the second column. You can edit the first column of `aa2codon.dat` to use a different genetic code.

References

Henikoff S., Henikoff J. G., 1992, Amino acid substitution matrices from protein blocks, *Proc. Natl. Acad. Sci. USA*, 89, 10915–10919

Kimura M., 1980, A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences, *J. Mol. Evol.*, 16, 111–120

Rice P., Longden I., Bleasby A., 2000, EMBOSS: the European molecular biology open software suite, *Trends Genet.*, 16, 276–277

Yang Z., 1994, Estimating the pattern of nucleotide substitution, *J. Mol. Evol.*, 39, 105–111