PATRONUS
User's manual
(build 101, 7/7/2008)

Paolo Ribeca
paolo.ribeca@gmail.com

Table of contents

Overview . . . 3

Structure of the program . . . 3

Algorithm selection . . . 6

Accepted input format . . . 7

Commandline options . . . 8

- General options . . . 8
- Options related to the creation of the Markov model . . . 8
- Options related to finding patterns in the sequence . . . 9
- Options related to evaluating statistics . . . 9
- Options related to the choice of the algorithm . . . 10

Examples of use . . . 11

- Examples of algorithm selection . . . 12

How to read PATRONUS' output . . . 13

- Short output . . . 13
- Definition of the computed indicators . . . 13
- Detailed output . . . 14

Most frequent error messages . . . 15

Known bugs . . . 15

Citing PATRONUS . . . 15

License . . . 16

Changelog . . . 17

Overview

Assessing the statistical relevance of a motif in a stretch of DNA or protein is a task of central importance in biology, since in many cases the over- or under-representation of a motif may be linked to biological activity (although in general additional facts should be taken into account to decide whether or not a given occurrence of a motif has indeed some real biological activity).

PATRONUS is a program designed to compute in a very fast way the exact probability of observing a given number of occurrences of a simple motif (that is, a continuous word without gaps) in a sequence. Its intended scope is the analysis of very long biological sequences, like chromosomes or whole genomes of complex organisms. The probability is computed on the basis of the Markovian statistics of order m for the sequence, that is the recorded number of the occurrences of all the submotifs of length $m+1$ in the sequence. Contrary to what many people believe, computing such a probability for a generic motif is a computationally demanding task, mainly because motifs can overlap in non-trivial ways.

PATRONUS (from "PATtern Recognition by Optimized Numerical Universal Scoring") is able to produce the whole bulk of the probability distribution function, and works best in the range $0 \leqslant m \leqslant 3$, where we believe it to be the fastest in the world in its class (this claim is currently under peer review); for low orders of the Markov model our program is sometimes even faster than approximate methods — and the longer the sequence being analyzed, the better. The principles of our new computational method, which is based on systolic deterministic finite-state automata (systolic DFAs) and the fast Fourier-transform (FFT), are explained in http://arXiv.org/abs/0801.3675 (see Section *Citing PATRONUS* below).

On the other hand, if you need to use Markov models of order larger than 3, you have to evaluate a composite (gapped) motif, or you prefer approximate methods for the analysis of your sequence, then you should still refer to other "cousin" programs, like for instance spatt (http://stat.genopole.cnrs.fr/spatt). Similarly, you should consider that PATRONUS is not by itself a tool for motif finding (although we are currently developing new applications that exploit its algorithm along this direction).

PATRONUS is mostly written in Objective Caml (see http://www.ocaml.org), a very-high-level functional programming language, with a couple of C insets for the sake of optimization. Thanks to this architecture, the source code is remarkably compact if compared to the functionality offered by the program. To give you a feeling about what you can expect from the program a number of different timings for many realistic situations are shown in Table 1 (which again is extracted from http://arXiv.org/abs/0801.3675). In general, the program is very fast but also quite memory-hungry (you should have at least 2 Gigabytes of RAM if you intend to explore sequences which are between 10^8 and 10^9 in length).

At the moment the program is distributed in executable form, but starting from some later point in time the source code will be made freely available. The program is free for academic and non-commercial use, but is *not* free for any commercial use: please check carefully how the conditions of the license (see Section *License* below) apply to your case. Any kind of use of the program implies full acceptance of the license.

Finally, although we tried hard to produce a valuable and high-quality software, we are conscious that many bugs and oddities are still lurking in the depths of the code. Thus, any bug report or suggestion for improvement will be sincerely welcome.

Structure of the program

Although as a user you do not need to understand precisely how PATRONUS works internally, a minimal knowledge about its block structure may be useful to fine-tune its behaviour.

$m=1$							
Genome	L	m	N_{obs}	spatt	PATRONUS	ld/cp-spatt	gpatt
HIV1	9181	CCT	108	0.20s	0.084s	0.008s	0.004s
B.subtilis	4214630		50985	∞	1.2s	0.30s	0.12s
S.cerevisiae	12156678		138318	∞	1.5s	0.89s	0.34s
Human X Chr.	151058754		2512286	∞	6.1s	11s	4.2s
B.subtilis	4214630	ATATTC	1919	∞	0.61s	0.31s	0.12s
S.cerevisiae	12156678		7054	∞	0.84s	0.86s	0.34s
Human X Chr.	151058754		61408	∞	2.8s	11s	4.2s
B.subtilis	4214630	ATATTCATA	52	64s	0.25s	0.46s	0.12s
S.cerevisiae	12156678		173	∞	0.41s	1.3s	0.34s
Human X Chr.	151058754		2146	∞	1.1s	16s	4.2s
B.subtilis	4214630	ATATTCATATTC	2	4.5s	*0.004s*	0.45s	0.12s
S.cerevisiae	12156678		9	41s	*0.036s*	1.3s	0.34s
Human X Chr.	151058754		44	∞	*0.41s*	16s	4.2s
Human X Chr.	151058754	AATATTCATATTC	10	∞	*0.052s*	16s	4.2s
		TAATATTCATATTC	3	260s	*0.012s*	16s	4.2s
		ATAATATTCATATTC	1	130s	*0.008s*	16s	4.2s
$m=2$							
Genome	L	m	N_{obs}	spatt	PATRONUS	ld/cp-spatt	gpatt
B.subtilis	4214630	ATATTCATA	52	110s	1.4s	0.46s	0.12s
S.cerevisiae	12156678		173	∞	1.8s	1.3s	0.34s
Human X Chr.	151058754		2146	∞	5.2s	16s	4.2s
B.subtilis	4214630	ATATTCATATTC	2	7.2s	*0.024s*	0.45s	0.12s
S.cerevisiae	12156678		9	68s	*0.21s*	1.3s	0.34s
Human X Chr.	151058754		44	∞	1.6s	16s	4.2s
Human X Chr.	151058754	AATATTCATATTC	10	∞	*0.26s*	16s	4.2s
		TAATATTCATATTC	3	350s	*0.056s*	16s	4.2s
		ATAATATTCATATTC	1	200s	*0.020s*	16s	4.1s
$m=3$							
Genome	L	m	N_{obs}	spatt	PATRONUS	ld/cp-spatt	gpatt
B.subtilis	4214630	ATATTCATA	52	310s	34s	0.48s	0.12s
S.cerevisiae	12156678		173	∞	42s	1.4s	0.35s
Human X Chr.	151058754		2146	∞	92s	16s	6.3s
Human X Chr.	151058754	AATATTCATATTC	10	∞	*6.6s*	16s	4.1s
		TAATATTCATATTC	3	∞	*1.3s*	16s	4.2s
		ATAATATTCATATTC	1	490s	*0.50s*	16s	4.2s

Table 1. Comparative timings for various biological examples as obtained from **[a]** the exact FMCI method of program `spatt` (`http://stat.genopole.cnrs.fr/spatt`) **[b]** `PATRONUS`, the exact method introduced in `http://arXiv.org/abs/0801.3675`, where the meaning of the symbols appearing in this table is also fully explained **[c]** the FMCI large-deviations and compound-Poisson approximations, as given by the commands `ldspatt` and `cpspatt`, respectively **[d]** the FMCI Gaussian approximation, provided by the command `spatt --gaussian` (in the column labeled `gpatt`). The timings were computed on an Intel Core Duo T2500 processor at 2 GHz with 2 GBytes of RAM excluding all input/output-related operations (like building Markov models and counting pattern occurrences). The gaps present in the current reference assembly of chromosome X have been discarded for these runs. A timing of ∞ means that the corresponding test did not terminate within 10 minutes; in case of orange background `PATRONUS` was faster than both the large-deviations and the compound-Poisson approximations (which always show an almost identical timing), in case of yellow background `PATRONUS` showed a better performance also w.r.t. the Gaussian approximation. An italicized timing in `PATRONUS` column means that the p-value was computed with the direct method implemented since version `94`; in all other cases the `FFT` algorithm was used. The tested versions are `2.0-pre1` for `spatt` and `1.2.2` for `ldspatt`; as for `PATRONUS`, all the examples were run in double precision using build `94` with the default cutoff $\varepsilon = 10^{-14}$. The timings for the case $m=0$ are very similar to those for $m=1$, and thus they have been omitted.

As illustrated in Figure 1, the program can basically be seen as a series of cascaded filters, the

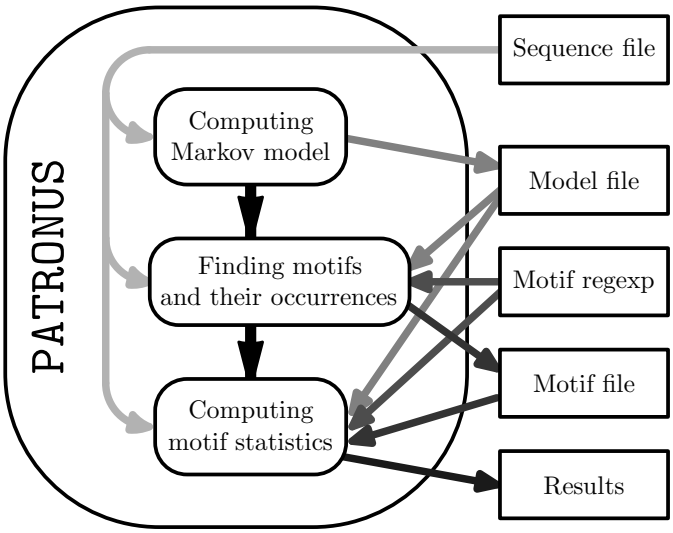

Figure 1. Internal structure of PATRONUS, and how its three stages interact with user-supplied files and parameters.

action of each stage being optional. Once you have chosen a Markovian order m for your analysis, to compute a probability distribution function you need: (1) a Markov model of your sequence (2) a list of motifs together with their occurrences in the sequence. This is why PATRONUS has the following three-staged structure:

1. the first stage produces a Markov model out of a given sequence and optionally writes it to a specified file, or reads an already existing model from a precomputed file.

2. the second stage scans the sequence for a set of motifs described in terms of a regular expression or a IUPAC template, if the user specifies one on the commandline; it then optionally writes the set of motifs, together with the recorded occurrence numbers, to a specified file. Alternatively, the set may be read from a precomputed file.

3. the third stage obtains the probability distribution function by running our algorithm for all the couples (motif, occurrence number for the motif) which have been found in the sequence during the previous stage.

In case of typical usage (see again Table 1), the most time-consuming operations are those involving input/output, like generating Markov models and scanning the sequence to find motif occurrences.

PATRONUS is designed to leave you complete freedom about how the computation should be set up. Of course, the key usage point is that the final stage *always* needs both a Markov model and a list of patterns with their occurrences to be able to produce statistics; however, you can supply the needed information in many possible ways.

For example, you can: (1) give PATRONUS a pregenerated file containing a Markov model for your sequence and let it compute the motif/occurrence list from the sequence, and save some time (2) give PATRONUS two pregenerated files, one containing the Markov model and one the list of motifs/occurrences, and save even more time (3) just give it the sequence and the set of patterns you want to find in terms of a regular expression, and let the program do everything for you. Conversely, you can use the program just to compute a Markov model, or a list of motifs/occurrences, or both, and quit without any computation of statistical indicators. An extensive list of usage examples is provided in Section *Examples of use*.

A few words more should be spent about how the user can select the motifs observed in the sequence. Strictly speaking, the second stage of the program is a search engine which scans the sequence in one or more steps and looks for all the substrings matching a regular expression specified by the user; when this task is completed, the program collects all the occurrences which have been found at least one time, and arranges them as a list of couples (motif, observed occurrence number for that motif). As explained in more detail in the Section *Commandline options*, the accepted syntax for regular expressions is that of the `PCRE` (`P`erl-`C`ompatible `R`egular `E`xpressions) `C` library (`http://www.pcre.org`); in addition, `PATRONUS` implements an extension of the syntax, allowing the user to specify IUPAC patterns if the regular expression is preceded by a suitable filtering directive. For example, the syntax `:iupac_dna:<pattern>` indicates that the `PCRE` regular expression `<pattern>` should be expanded in terms of the standard IUPAC DNA pattern syntax before being applied to the genome under analysis (see Section *Commandline options* below). From this point of view, the main usage difference between `PATRONUS` and other similar programs is that the user has to specify by hand the cases when he/she wants to compute the probability of a motif to occur exactly zero times (however, this case should not occur too frequently in the case of biological applications).

Finally, as in `spatt` and other similar programs, `PATRONUS` works in a way which is completely independent of the symbols actually appearing in the sequence being analyzed; that is, it can be used without any modification to analyze DNA, proteins, or arbitrary strings as well.

Algorithm selection

The new formula presented in `http://arXiv.org/abs/0801.3675` to deduce Markovian probability distribution functions may be evaluated in many different ways. At least two main algorithms (which show different and somewhat complementary strengths/weaknesses) are worthwhile considering:

1. The `FFT`-based algorithm (implemented since build 1). It is in general very fast and robust, and capable of producing a snapshot of the probability distribution function, excluding the far tails when they are too small (see `http://arXiv.org/abs/0801.3675` for more details). In fact, as a rough rule of thumb the `FFT` algorithm is able to sample the probability function for all the occurrence numbers N such that

$$10^{-d} \max p \leqslant p(N) \leqslant \max p,$$

being d a number of precision digits which depends on the floating type used for the computation, and usually falls in the range between 10 and 13 for double precision. From that snapshot, all the moments of the statistical distribution can always[1] be computed with a reasonable precision; from the moments, any indicator which only depends on them (like for instance the z-value) can in turn be obtained. On the other hand, a non-zero value for the p-value can be obtained with this technique only if $p(N_{\mathrm{obs}})$ is greater than the threshold $10^{-d} \max p$; for example, if the maximum of the distribution is close to 1, then the algorithm in double precision will only be able to estimate p-values greater than 10^{-12} or so.

   The typical use for this algorithm might be as a general-purpose robust evaluator when the knowledge of the p-value is not always essential, and some other statistical estimator like the z-value can be used instead — for instance, as the basic block for a motif finder.

1. In fact, in this discussion, the word "always" should be interpreted as "for statistically reasonable datasets". It is always possible (and we mean "always" literally this time) to generate a dataset which is so fantastically improbable that the p-values obtained from it will be *really* small —for instance 1^{-1000}—, therefore falling outside of any reasonable computational precision and underflowing any implementation of a given algorithm. In such cases the p-value for $N_{\mathrm{obs}} > 0$ will be evaluated as 0, and/or the z-value to ∞.

2. Since version 94 on, a new direct evaluation method has been implemented which outperforms the `FFT` one for relatively small values of the occurrence number N_{obs} (empirical tests show that the execution times for the two methods become more or less equivalent when $N_{\text{obs}} \approx 25$). This algorithm is able to evaluate the probability function in the truncated range from 0 to N_{obs} with some 10 digits of *relative* precision at each point of the distribution —being closer in spirit to the behavior of `spatt`—; as a consequence, it is able to estimate the p-value but not the general behavior of the distribution (in particular, it is in general *not* able to deduce the values for the moments of the distribution nor the z-value).

   Thus, the typical situation for which this method might be preferred is when the user needs to compute only the p-value of (very) long patterns.

Since there is a number of applications where `PATRONUS` could be used, each one with different requirements and typical parameter ranges, we concluded that the decision about what algorithm is better suited for the task at hand should be left to the user; on the other hand, we felt it was important to free the user as much as possible from the gory details which would be implied by a low-level manual choice of the algorithm to run.

As a result, we have implemented three different "strategies", which give the user the freedom to specify what the preferred goal of the computation should be, at the same time allowing the program to select the most efficient algorithmic strategy for the problem at hand:

1. the "***p*-value strategy**": `PATRONUS` uses the non-`FFT` algorithm for relatively small values of N_{obs} (or, more precisely, when $N_{\text{obs}} \leqslant$ `turning_point`, the hardwired default value of `turning_point` being 25), and the `FFT` algorithm otherwise. This strategy usually allows a faster and more precise computation of the p-value, the drawback being that in some cases the z-value, the shape of the probability function and the estimates of its moments will remain `unknown`

2. the "***z*-value strategy**": `PATRONUS` always uses the `FFT` algorithm; this choice used to be the default in `PATRONUS` prior to version 94. This strategy always allows the computation of the z-value, the shape of the probability function and the estimates of its moments, but in some cases the obtained p-value could be 0

3. the "`both` **strategy**": `PATRONUS` first computes a snapshot of the distribution function by using the `FFT` algorithm, and then, in case the obtained estimate for the p-value is 0 and $N_{\text{obs}} \leqslant$ `turning_point`, it recomputes the p-value by using the non-`FFT` algorithm. This strategy allows the best determination of all the indicators, at the price of a sometimes increased computational time (in fact, it could be necessary to run both the `FFT` and the non-`FFT` algorithm one after the other).

Finally, should the user not be satisfied with any of the aforementioned strategies, (s)he will still be able to force by hand the choice of a particular algorithm by modifying the values of a few control parameters via the commandline switches described in Section *Options related to the choice of the algorithm* (some practical examples of use may be found in Section *Examples of algorithm selection*).

Accepted input format

`PATRONUS` accepts file in simplified multi-FASTA format, where each sequence *must* be preceded by a comment line beginning with character '`>`' (in the original standard such lines are optional in case one single sequence is provided) and the sequence itself can be spread among an arbitrary number of lines. Since build 100, empty lines are legal too. The optional comment lines present in the original definition of the format (those beginning with character '`;`') are *not* allowed in `PATRONUS` input.

As for the construction of the Markov model, if multiple sequences are provided PATRONUS ensures that the statistics are still correctly computed; in particular, the sequences are *not* merged into one larger sequence before being processed (in fact, this would introduce artificial patterns across the junctions which are not present in the original sequences).

Commandline options

The program understands the following options (all the options made of more than two characters can be given both with one and two trailing hyphens; if the option takes one, the type of the argument is shown in parentheses after the list of accepted option forms):

General options

-h, -help. Prints a concise summary of all options and exit. *This functionality is not yet implemented in current build.*

-m, -markov-order, -markovian-order (non-negative integer). Specifies the order of the Markov model which has been selected for the analysis.

-i, -input, -if, -input-fasta (string). Specifies as input a sequence file in FASTA format.

-dont-skip-blanks. When reading the sequence from a FASTA file, blank characters (that is, the whitespace ' ', the tab '\t' and '\f') appearing in the sequence are usually skipped. You should turn on this option if such characters belong to your alphabet instead, and you would like them to be taken into account when PATRONUS builds the statistics for the sequence.

-dont-skip-newlines. As above, but for the newline character '\n'.

-a, -alphabet (string). Specifies the alphabet for your sequence, that is the set of symbols it is composed by. PATRONUS computes the alphabet by itself, so this option is normally not necessary; however, it can be useful in case you want to add to the alphabet some characters which the sequence does not include.

-o, -or, -output, -output-results (string). Specifies a filename to which the output of the program should be redirected. If this option is not given, the results of the program are printed on the standard output.

-O, -os, -output-stats, -output-statistics (string). Indicates that a detailed report file containing all the obtained statistical information (complete probability distribution function, moments and indicators) should be written for each analyzed pattern. The resulting files will be given the name <prefix>.<pattern>.res. It should be emphasized that the use of this option does *not* imply any penalty on the execution time of the program: PATRONUS computes all the probability distribution function in any case, and simply discards it when this option is turned off.

-license. After the program header and before starting program execution prints the text of the license for PATRONUS (see Section *License*).

-citation. After the program header and before starting program execution prints the text of the citation for PATRONUS (see Section *Citing PATRONUS*).

Options related to the creation of the Markov model

-om, -output-model (string). Outputs the Markov model obtained after running the first stage of the program to the specified file.

`-im, -input-model` **(string).** Turns off the first stage of the program, and reads a Markov model from the specified file instead.

Options related to finding patterns in the sequence

`-p, -pattern` **(string).** Specifies the motifs to be matched in the sequence by the second stage of the program in terms of a `PCRE` regular expression (see `http://www.pcre.org`). At the end of the second stage, all the substrings which match the regular expression and occur in the sequence at least one time will be kept for possible subsequent analysis by the third stage.

The regular expression can be preceded by a filtering option, which specifies that the regexp should be filtered in a suitable way before being applied to the sequence. The filtering options currently implemented are:

`:iupac_dna:.` Implements the standard IUPAC DNA motif specification. In particular, the characters in the regexp are mapped as follows (all mappings hold both for uncapitalized and capitalized letters):

M		{A|C}
R		{A|G}
W		{A|T}
S		{C|G}
Y		{C|T}
K	becomes	{G|T}
V		{A|C|G}
H		{A|C|T}
D		{A|G|T}
B		{C|G|T}
X|N		{A|C|G|T}

IMPORTANT:

`PATRONUS` is implemented in such a way to work for a generic string; in particular, no assumption is made about the set of characters used, and for any new input file a new alphabet is deduced. As a result, `PATRONUS` does not have any way to recognize that two DNA/protein sequences are one the capitalized version of the other: *it is the user's responsibility to ensure that the pattern (s)he is interested in is given to the* `-p` *option by honoring the capitalization convention found in the file*. For instance, looking for the pattern `acgt` in a capitalized sequence `ACGT...` will (correctly) give 0 matches.

`-op, -output-patts, -output-patterns` **(string).** Outputs the list of the motifs found in the sequence, together with their occurrences, to the specified file. The format of the generated file is explained in the following entry.

`-ip, -input-patts, -input-patterns` **(string).** Turns off the second stage of the program, and reads a list of motifs with their occurrences from the specified file. Each line of the file should be of the form

```
"<pattern>"  <number of its occurrences>
```

and lines beginning with `#` are considered comments. Please notice that the pattern *must* be delimited by double quotes.

Options related to evaluating statistics

`-l, -length, -sequence-length` **(non-negative integer).** Specifies that the supplied number should be used as the sequence length.

`-n`, `-number`, `-nobs`, `-n-obs`, `-occurrences` **(non-negative integer).** Specifies that the supplied number should be used as the number of motif occurrences.

Options related to the choice of the algorithm

`-s`, `-strategy` **(string).** Selects the strategy to be employed for the computation (see also Section *Algorithm selection*). Possible values are:

`p-value`, `p-val`. Tries to compute the p-value using the non-`FFT` algorithm if $N_{\text{obs}} \leqslant$ `turning-point` (see below), and the `FFT` algorithm otherwise. It is the default.

`z-value`, `z-val`. Computes the z-value and all other statistical indicators (p-value, mean, standard deviation, etc.) using the `FFT` algorithm. It used to be the default in versions prior to 94.

`both`. Tries to compute all the indicators (p-value, z-value, mean, standard deviation, etc.) by first running the `FFT` algorithm and then, if needed, the non-`FFT` algorithm too.

`-Za`, `-z-value-algo`, `-z-value-algorithm`. Uses the specified `FFT` algorithm to evaluate the z-value, the p-value and other statistical indicators. Choices supported as of current build are:

`fft`, `fft-double`. The standard systolic-DFA/`FFT` algorithm which is described in `http://arXiv.org/abs/0801.3675`, in double precision. It is the default.

`fft-float`, `fft-single`. The standard algorithm, but in single precision. It is possibly faster and takes half the memory w.r.t. the double-precision one; however, *it should be used with* lot *of care since it has not been carefully tested and the very few precision digits of the float could be not enough to guarantee meaningful results in the end*.

`fft-mem`, `fft-mem-double`. A variation of the standard algorithm, which is (slightly) slower but takes less memory.

`fft-mem-float`, `fft-mem-single`. As the latter, but in single precision. The same *caveat* which has been pointed out for `fft-float` applies also to this case.

`-Zc`, `-z-value-cutoff`, `-Zt`, `-z-value-threshold` **(float).** Use the given value as the cutoff ε for the tails of the distribution when running one of the `FFT` algorithms above (see `http://arXiv.org/abs/0801.3675` for a full explanation). This option should be used with care, since the built-in thresholds —10^{-14} for the double precision algorithms and 10^{-6} for the simple precision ones — are usually fine (and more often than not a larger cutoff may result in larger execution times).

`-ZL`, `-z-value-logfile` **(string).** Prints some statistics about the underlying `FFT` engine to the specified file. This feature is undocumented.

`-Pa`, `-p-value-algo`, `-p-value-algorithm` **(string).** Computes the truncated distribution from 0 to N_{obs} + `guard_width` (see also option `-Pg` below and Section *Definition of the computed indicators*) and the p-value by using the specified direct (non-`FFT`) algorithm. Choices supported as of current build are:

`double`. A straightforward algorithm which directly evaluates Equation 3 of `http://arXiv.org/abs/0801.3675`, in double precision. It is the default.

`single`, `float`. Same as above, but in single precision. Although potentially faster, *this option should be used with* lot *of care since it has not been carefully tested and the very few precision digits of the float could be not enough to guarantee meaningful results in the end.*

`-Pt`, `-p-value-turning-point` **(non-negative integer).** Controls what algorithm should be employed, and how, when strategies `p-value` or `both` are specified (roughly speaking, when computing the p-value the non-`FFT` algorithm is used if $N_{\text{obs}} \leqslant$`turning_point`, and the `FFT` one otherwise; see Section *Algorithm selection* for more details); it has no effect when the strategy is `z-value`. The default value is 25; this value was empirically selected to guarantee maximal efficiency, since when $N_{\text{obs}} \approx 25$ both the non-`FFT` and the `FFT` algorithm roughly take the same execution time. This value can be changed to force or fine-tune the choice of the desired algorithm.

`-Pg`, `-p-value-guard-width` **(non-negative integer).** Controls how precisely the p-value is computed when the non-`FFT` algorithm is used and the motif is over-represented (see Section *Definition of the computed indicators* for more details). The default value is 2.

Examples of use

In all the following examples, we will assume that your sequence is contained in a FASTA file called `sequence.fas`. Following FASTA syntax, the sequence may be spread over more than one chunk.

The following command produces a Markov model of order 1 for the sequence, outputs it to a file called `sequence.1.model` and quits since no motif has been specified:

```
patronus -m 1 -i sequence.fas -om sequence.1.model
```

The following command produces a Markov model of order 1 for the sequence and finds in it all the occurrences of pattern `ATCAT`; after that, if the occurrence number is greater than zero it computes the statistical indicators relative to the pattern, and outputs them to a file called `sequence.1.out`; in addition, the complete probability distribution function is written to the file `sequence.1.ATCAT.res`.

```
patronus -m 1 -i sequence.fas -p ATCAT -o sequence.1.out -os sequence.1
```

Same as before, but for all the patterns of four letters with a '`T`' as the third letter (a statistical report with a suitable name will be produced for each pattern):

```
patronus -m 1 -i sequence.fas -p "..T." -o sequence.1.out -os sequence.1
```

It should be noted that the (double) quotes for the pattern are not strictly needed in this example; however, it is a good idea to use them when the motif is described in terms of a regular expression, to avoid interferences with the shell. The previous example can also be reformulated in terms of IUPAC pattern syntax:

```
patronus -m 1 -i sequence.fas -p ":iupac_dna:NNTN" -o sequence.1.out
         -os sequence.1
```

The list of motifs which have been found in the sequence, together with their occurrences, can be written to a file called `sequence.1.patterns` after the completion of the second stage of the program as follows:

```
patronus -m 1 -i sequence.fas -p ":iupac_dna:NNTN" -op sequence.1.patterns
```

How to avoid the recomputation of the Markov model and the motif occurrences at each run? With the following syntax we can reuse the Markov model file `sequence.1.model` which we produced before:

```
patronus -m 1 -i sequence.fas -im sequence.1.model -p ":iupac_dna:NNTN"
         -o sequence.1.out -os sequence.1
```

and with the following command we can reuse both the Markov model file and the pattern file `sequence.1.patterns` (in this case we can omit the sequence file, since all the needed information is already contained in the Markov model file and in the pattern file):

```
patronus -m 1 -im sequence.1.model -ip sequence.1.patterns -o sequence.1.out
```

If we intend to compute the probability of a pattern occurring exactly zero times, we cannot use the previous machinery, since the patterns occurring zero times are discarded by the second stage of the program. However, we can explicitly specify that the number of occurrences is zero as follows (we are looking for pattern `ATCTTTCTA` in this case, output goes to the screen and the probability distribution function is written to file `sequence.1.ATCTTTCTA.res`):

```
patronus -m 1 -im sequence.1.model -p ATCTTTCTA -n 0 -os sequence.1
```

Alternatively, we could prepare a pattern file containing entries where the number of occurrences is zero, and supply it to the program.

Finally, we can compute indicators and probability distribution functions for a sequence of arbitrary length and an arbitrary number of pattern occurrences even without supplying the actual sequence (but we still need to specify a Markov model file):

```
patronus -m 1 -im sequence.1.model -l 1000000 -p ATCTTTCTA -n 1200 -os sequence.1
```

Examples of algorithm selection

The strategy selected by default (see Sections *Algorithm selection* and *Options related to the choice of the algorithm* above for the precise algorithmic definition of the strategies and the meaning of the various parameters) is `p-value`. This strategy uses the non-`FFT` algorithm if $N_{\mathrm{occ}} \leqslant$ `turning_point` —the default value for `turning_point` being 25—, and the `FFT` one otherwise. However, it is possible to fine-tune the choice of the algorithm by modifying the value of `turning_point`. For example, suppose that we wish to force the use of the `FFT` algorithm even if $N_{\mathrm{occ}} = 40$; we can do so with the syntax

```
patronus -m 1 -im sequence.1.model -p ATCTTT -n 40 -Pt 100
```

The last option, `-Pt 100`, specifies that `turning_point` is 100 for this run, and by consequence the non-`FFT` algorithm will be used even if $N_{\mathrm{occ}} = 40$.

Let us suppose now that we are interested in knowing the p-value with more precision. Such precision depend on how fast the probability distribution function decays for $N_{\mathrm{obs}} \to \infty$, and can sometimes be enhanced by increasing the value of the parameter `guard_width` (see Section *Definition of the computed indicators* below for a precise definition). To specify in the context of our previous example that the number of guard points should be, say, 4 instead of the default value of 2 we can write

```
patronus -m 1 -im sequence.1.model -p ATCTTT -n 40 -Pt 100 -Pg 4
```

The use of the non-`FFT` algorithm allows us to obtain a non-zero value for the p-value in more cases; on the other hand, it does not in general allow us to deduce the full bulk of the probability function, nor an estimate of its moments. If we are not primarily interested in the p-value, but rather satisfied with the knowledge of the z-value and the moments of the distribution, we can then specify that the `FFT` algorithm —that is, the `z-value` strategy— should be employed:

```
patronus -m 1 -im sequence.1.model -p ATCTTT -n 40 -s z-value
```

thus recovering the default behavior of `PATRONUS` prior to version 94.

Finally, in case we are interested in obtaining both the shape of the probability function and a p-value as precise as possible (at the price of a possibly longer execution time) we can specify the strategy `both`:

```
patronus -m 1 -im sequence.1.model -p ATCTTT -n 40 -Pt 100 -s both.
```

How to read PATRONUS' output

Short output

How to read PATRONUS' output? The answer is very simple. First of all, the program produces a header

```
Welcome to PATRONUS build 101 (2008/07/07 12:07:21)
 (c) 2007-2008 Paolo Ribeca <paolo@paoloribeca.net>
 All rights reserved. Unauthorized use is forbidden.
 Non-commercial use is free, commercial use is not.
 Please consult the license (option "--license"),
  and contact the author in case of any doubt
```

then a progress line

```
Evaluating pattern "ATAATATTCATATTC" (1/1)... done.
```

and after that is comes the interesting part containing the results of the run:

```
# "ATAATATTCATATTC"
P_value(1)=0.3486
Z_value(1)=unknown Mean=unknown StdDev=unknown
```

If more than a pattern is being evaluated, only one header is printed, and after it a sequence of blocks composed each by the progress line and the results.

Depending on the strategy selected by the user (see Section *Options related to the choice of the algorithm*) some of the quantities may be `unknown`, as the example above shows. In fact, if we repeat the run specifying the strategy `-s both` —which implies the attempt by PATRONUS to compute both the z-value and the p-value— we obtain for the results the complete information

```
# "ATAATATTCATATTC"
P_value(1)=0.0626195
Z_value(1)=3.67814 Mean=0.064666 StdDev=0.254295
```

Definition of the computed indicators

Aside from standard statistical indicators (mean, variance, skewness, etc.), and depending on the evaluation strategy selected by the user (see Section *Options related to the choice of the algorithm*), PATRONUS attempts to compute from its numerical estimate of the probability function one or both of the following indicators: the p-value and the z-value. They evaluate in different ways how improbable is the number of times N_{occ} the specified pattern is found in the sequence, and are computed from N_{occ} and the snapshot of the distribution as follows:

$$p\text{-value}(N_{\text{obs}}) = \begin{cases} \sum_{N=0}^{N_{\text{obs}}} p(N) & \text{if } N_{\text{obs}} \leqslant \mu \\ \sum_{N=N_{\text{obs}}}^{\infty} p(N) & \text{if } N_{\text{obs}} > \mu \end{cases}$$

$$z\text{-value}(N_{\text{obs}}) = \frac{N_{\text{obs}} - \mu}{\sigma}$$

being μ and σ the estimates of the mean and the standard deviation of the distribution, respectively.

The p-value might be zero when the pure `FFT` algorithm has been used —that is, when specifying either the `-s z-value` strategy or the `-s both` one— and $p(N_{\text{obs}})$ falls in a far tail of the distribution which has been truncated; alternatively, the z-value could be `unknown` when the non-`FFT` algorithm has been used (possible when the default strategy `-s p-value` is selected). In addition, when the non-`FFT` algorithm is used `PATRONUS` is in general unable to give an estimate of μ, so the quantity returned for the p-value is actually

$$p\text{-value}(N_{\text{obs}}) = \min\left(\sum_{N=0}^{N_{\text{obs}}} p(N), \sum_{N=N_{\text{obs}}}^{N_{\text{obs}}+\texttt{guard\_width}} p(N)\right)$$

where the parameter `guard_width` specifies the number of additional points of the distribution to the right of N_{occ} which should be computed and contribute to the evaluation of the p-value if the pattern is over-represented (see Section *Options related to the choice of the algorithm* for more information about how to modify the value of `guard_width` from the commandline).

Detailed output

In this section we briefly explain how to read the output returned by `PATRONUS` when option `-O` is specified, although the format should be pretty much self-explanatory.

First of all, it comes an header with some information about the program and the time it has been run.

```
# Generated by PATRONUS build 89 (2008/04/14 20:52:26)
# (c) 2007-2008 Paolo Ribeca <paolo@paoloribeca.net>
# on Tuesday 15 April 2008 19:56:00.
```

Then, there is a section recording the value of the most relevant quantities during the run (the main features of the sequence, the parameters of the Markov model deduced from it and the pattern being analyzed). In particular, we note that the Markov model is encoded as a list of couples (`"substring"`, `<normalized occurrence probability>`): for instance, in the present example the string `"T"` appears with the 29.94% of probability with respect to all possible one-letter groups.

```
##
# Sequence length: 89997
# Alphabet: "ACGT"
# Order of Markov model: 0
# Statistics:
# "" 1
# "A" 0.3383446114870496
# "C" 0.176828116492772
# "G" 0.185461737613476
# "T" 0.2993655344067024
# Pattern: "ATTAAAAACC"
```

Next, the results of the run: first of all, the probability distribution of the number of occurrences of the given pattern, truncated when they are too small. For instance, in this example the probability of recording exactly 4 occurrences of pattern `"ATTAAAAACC"` will be 0.000584. As a convenience to the user, all the moments of the distribution from order 0 to 4 (that is, integral, mean, standard deviation, skewness and kurtosis) are also listed, although it would be of course perfectly possible to recompute them from the given distribution data. It is worthwhile noting that apart from the lines containing the distribution all the other lines in the file start with the comment character `#`, making it possible to plot the distribution with programs like `gnuplot` without editing/parsing the file in any way.

In the case when no `FFT` algorithm has been used —that is, essentially, if the default strategy given by the option `-s p-value` is employed— the values of the indicators will be set to `unknown` (but all the lines will be emitted anyway to make automatic parsing easier).

```
##
# Support width: 11
# Integral: 0.999999999999
# Mean: 0.378309
# Standard deviation: 0.615044
# Skewness: 1.62564
# Kurtosis: 5.64229
# Distribution:
0 0.6850085952533627
1 0.2591654688805601
2 0.04901585151883907
3 0.006178925953539214
4 0.0005840620128848418
5 4.415736715173342e-05
6 2.781469958875471e-06
7 1.501436639017004e-07
8 7.090157748904565e-09
9 2.975503429314495e-10
10 1.123608364059526e-11
```

And finally, here come the p-value and the z-value (see Section *Definition of the computed indicators* above):

```
##
# P_value(1)=0.314991
# Z_value(1)=1.01081
```

Most frequent error messages

To be completed.

Known bugs

- When using the slower memory-saving algorithm (options `-A fft-mem` and `-A fft-mem-single`) the computed support is wrong.
- Help from the commandline (option `-h`) is not yet implemented.
- Verbose messages for fatal errors are not yet implemented (although the names of the generated exceptions should be pretty much self-explanatory).

These bugs will be fixed in some upcoming version.

Citing `PATRONUS`

Many friends of mine keep on asking me: "Why did you waste so many months of your two-year postdoc contract to make the program publicly available? You could have kept it to yourself... and like that you will only help some Big Guys who will use the program to get even more Big Papers without even thanking you."

Although at the beginning I used to think that questions of the like are simply too cynical, and there are moral reasons to make your programs publicly available, I must say that with time I am getting less and less convinced by my own arguments. And something must really be wrong with the system if this kind of arguments turns out to be true.

So, please help me proving that my friends are wrong: if you use `PATRONUS` to obtain some sizeable scientific result, please be fair and cite the following paper:

> Paolo Ribeca, Emanuele Raineri
> *"Faster exact Markovian probability functions for motif occurrences: a DFA-only approach".*
> `http://arXiv.org/abs/0801.3675`

or any more up-to-date reference which should in the meantime appear in `PATRONUS`' site (`http://www.paoloribeca.net/software/PATRONUS`). You can also obtain the text of the citatation by invoking `PATRONUS` with the `--citation` option (see Section *Commandline Options* above).

License

The use of `PATRONUS` is free for academic no-profit purposes, and for non-commercial purposes in general.

On the other hand, you cannot use `PATRONUS` for commercial purposes without a written separate licensing agreement obtained from its author,

> Paolo Ribeca
> `paolo@paoloribeca.net` or
> `paolo.ribeca@gmail.com`.

In particular, without the explicit written consent of the author you can not —no matter how you obtained the program—:

1. redistribute `PATRONUS` together with other packages/softwares
2. sell `PATRONUS` in any form
3. use `PATRONUS` for any project of commercial nature whatsoever, irrespective of whether such a commercial nature is directly manifest in the way the software is used.

   For instance, the use of `PATRONUS` as a stage of a larger software pipeline which leads to a patent or a commercial advantage shall equally be considered commercial, and subject to a licensing agreement with the author of the software.
4. disassemble `PATRONUS`, reverse engineer it, link or bundle it to other libraries, and so on.

In any case, no commercial use of `PATRONUS` shall be considered approved without the prior written consent of its author.

Finally, in both the cases of commercial and non-commercial use:

The program is provided "as is", without any warranty of any kind either expressed or implied, in the hope that it will be useful; the entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the entire cost of all necessary servicing, repair or correction.

(You can also obtain the text of the license from the file `LICENSE` which is distributed along with the program, or by invoking `PATRONUS` with the `--license` option, see Section *Commandline Options* above).

Changelog

Version 101 (7/7/2008) — Various bugfixes w.r.t. the new features introduced in build 94. Empty lines in FASTA input, which were not legal in previous builds, are now accepted.

Version 94 (not public) — **A new non-`FFT` algorithm to directly evaluate the truncated distribution and the *p*-value has been introduced; this algorithm is much faster for moderate values of the number of pattern occurrences, $N_{\mathbf{obs}} \leqslant 25$**. Different evaluation "strategies" are now possible, allowing the user to specifiy whether the computation should be preferentially directed towards the computation of the p-value, the z-value, or both. The default behaviour (which was 'z-value' in previous versions) has been changed to p-value.

Command-line options `-a`, `-c` and `-L` have been renamed, while all the new options controlling the choice of the strategy (described in Section *Options related to the choice of the algorithm*) have been introduced.

Version 90 (5/5/2008) — It is mostly a bug-fixing release. **Since two major bugs have been fixed, we recommend to all the users of version 84 to upgrade to version 90**.

Bugs fixed: (1) Although the computation of the distribution and z-value were correct, a bug in version 84 was causing incorrect values for the p-value to be deduced in many cases. (2) A bug in the computation of Markov statistics made the program crash whenever the creation of a Markov model of order 0 was attempted.

New features: Introduced command-line options `-dont-skip-blanks` and `-dont-skip-newlines`.

Version 84 (12/2/2008) — The first version publicly available.