

seqpp Reference Manual
4.1.5

Generated by Doxygen 1.3.9.1

Thu Aug 4 20:33:12 2005

Contents

1	Welcome to seq++	1
2	seqpp Hierarchical Index	3
2.1	seqpp Class Hierarchy	3
3	seqpp Class Index	5
3.1	seqpp Class List	5
4	seqpp File Index	7
4.1	seqpp File List	7
5	seqpp Page Index	9
5.1	seqpp Related Pages	9
6	seqpp Class Documentation	11
6.1	Markov Class Reference	11
6.2	MTDMarkov Class Reference	18
6.3	Partition Class Reference	20
6.4	PhasedMarkov Class Reference	23
6.5	PhasedMTDMarkov Class Reference	38
6.6	PhasedPMarkov Class Reference	40
6.7	PhasedVLMarkov Class Reference	47
6.8	PMarkov Class Reference	49
6.9	PrimaryCount Class Reference	55
6.10	PrimarySequence Class Reference	60
6.11	PrimarySequenceSet< TSequence > Class Template Reference	63
6.12	Sequence Class Reference	65
6.13	SequenceSet Class Reference	68
6.14	Translator Class Reference	70
6.15	VLMarkov Class Reference	77

7	seqpp File Documentation	79
7.1	Coder.h File Reference	79
7.2	const.h File Reference	80
7.3	DOXYMAINPAGE.h File Reference	82
7.4	Markov.h File Reference	83
7.5	MTDMarkov.h File Reference	84
7.6	Partition.h File Reference	85
7.7	Partition_box.h File Reference	86
7.8	PhasedMarkov.h File Reference	87
7.9	PhasedMTDMarkov.h File Reference	88
7.10	PhasedPMarkov.h File Reference	89
7.11	PhasedVLMarkov.h File Reference	90
7.12	PMarkov.h File Reference	91
7.13	PrimaryCount.h File Reference	92
7.14	PrimarySequence.h File Reference	93
7.15	PrimarySequenceSet.h File Reference	94
7.16	Sequence.h File Reference	95
7.17	SequenceSet.h File Reference	96
7.18	Translator.h File Reference	97
7.19	VLMarkov.h File Reference	98
8	seqpp Page Documentation	99
8.1	Library	99
8.2	Programs	101
8.3	What's new	103
8.4	Bug and Feedback	104
8.5	ToDo	105
8.6	Download and Installation	106
8.7	Licence	107
8.8	References	108

Chapter 1

Welcome to seq++

The seq++ package offers

- a C++ object-oriented **Library**(p. 99) for developers
- a reference set of **Programs**(p. 101) for biologists working on sequences and statistics

seq++ modularity enables the study of genomic data at abstract levels, as well as any biological process which can be seen as a succession of states taken in a finite ensemble:

The genericity of seq++ is ensured by the possibility of storing and working with **any kind of sequence** based on any alphabet (DNA, proteins, codons, others...).

seq++ enables sequences modeling with **Markov, Variable Length Markov and newly developed Parsimonious Markov** models and **Mixture Transition Distributions** models. All these models can be phased.

Simulation modules are implemented so that developers can perform Monte Carlo methods.

- see section **Library**(p. 99) to use the library for your developments.
- see section **Programs**(p. 101) to use estimation and simulation programs.
- see section **References**(p. 108) for information on mathematical background.
- see section **Download and Installation**(p. 106) for operating instructions.
- see section **Bug and Feedback**(p. 104) for your comments on seq++.
- see section **What's new**(p. 103) and **ToDo**(p. 105) for updated releases.
- see section **Licence**(p. 107).

Reference: Miele V, Bourguignon PY, Robelin D, Nuel G, Richard H. (2005) seq++: analyzing biological sequences with a range of Markov-related models. *Bioinformatics* 21 (11): 2783.

Warning: please watch **The correct version of the published Fig.2.**(p. 100)

Contact: Vincent Miele name@genopole.cnrs.fr [anti-spam: please replace "name" by the corresponding name "miele"]

Chapter 2

seqpp Hierarchical Index

2.1 seqpp Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Partition	20
PhasedMarkov	23
Markov	11
MTDMarkov	18
PMarkov	49
VLMarkov	77
PhasedMTDMarkov	38
PhasedPMarkov	40
PhasedVLMarkov	47
PrimaryCount	55
Sequence	65
SequenceSet	68
PrimarySequence	60
Sequence	65
PrimarySequenceSet< TSequence >	63
PrimarySequenceSet< Sequence >	63
SequenceSet	68
Translator	70

Chapter 3

seqpp Class Index

3.1 seqpp Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Markov (Markov modelling, estimation and simulation)	11
MTDMarkov (Mixture Transition Distribution Markov (p.11) modelling, estimation and simulation)	18
Partition (Dealer of alphabet partitions)	20
PhasedMarkov (Phased Markov (p. 11) modelling, estimation and simulation)	23
PhasedMTDMarkov (Phased Mixture Transition Distribution Markov (p.11) modelling, estimation and simulation)	38
PhasedPMarkov (Phased Parsimonious Markov (p. 11) modelling, estimation and simulation)	40
PhasedVLMarkov (Phased Variable Length Markov (p. 11) modelling, estimation and simulation)	47
PMarkov (Parsimonious Markov (p. 11) modelling, estimation and simulation)	49
PrimaryCount (Abstract class designed for the (coded)word count)	55
PrimarySequence (Virtual class corresponding to a generic sequence)	60
PrimarySequenceSet < TSequence > (Class corresponding to a generic sequence set)	63
Sequence (Has been designed for Markovian analysis of a single sequence)	65
SequenceSet (Set of Sequence (p. 65) objects)	68
Translator (Alphabet designer tool)	70
VLMarkov (Variable Markov (p. 11) modelling, estimation and simulation)	77

Chapter 4

seqpp File Index

4.1 seqpp File List

Here is a list of all documented files with brief descriptions:

arnoldi.h	??
arnoldi_const.h	??
Coder.h (Coder)	79
const.h (Configuration file)	80
DOXYMAINPAGE.h	82
Markov.h (Markov(p.11) model)	83
mtd_core.h	??
MTDMarkov.h (Mixture Transition Distribution Markov (p.11) model)	84
node_base.h	??
Partition.h (Partition (p.20))	85
Partition_box.h (Partition_box)	86
PhasedMarkov.h (Phased Markov (p.11) model)	87
PhasedMTDMarkov.h (Phased Mixture Transition Distribution Markov (p.11) model)	88
PhasedPMarkov.h (Phased Parcimonious Markov (p.11) model)	89
PhasedVLMarkov.h (Phased Variable Length Markov (p.11) model)	90
PMarkov.h (Parcimonious Markov (p.11) model)	91
pmm_forest.h	??
pmm_leaf.h	??
pmm_node.h	??
pmm_tree.h	??
PrimaryCount.h (Abstract class designed for the (coded)word count)	92
PrimarySequence.h (Abstract generic sequence)	93
PrimarySequenceSet.h (Generic set of sequence objects)	94
Sequence.h (Sequence (p.65) class has been designed for Markovian analysis)	95
SequenceSet.h (Set of Sequence (p.65) objects)	96
Translator.h (General Alphabet class)	97
tree_t.h	??
vlm_tree.h	??
VLMarkov.h (Variable Length Markov (p.11) model 01/2004)	98

Chapter 5

seqpp Page Index

5.1 seqpp Related Pages

Here is a list of all related documentation pages:

Library	99
Programs	101
What's new	103
Bug and Feedback	104
ToDo	105
Download and Installation	106
Licence	107
References	108

Chapter 6

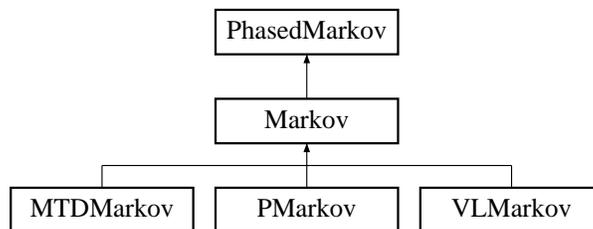
seqpp Class Documentation

6.1 Markov Class Reference

Markov modelling, estimation and simulation.

```
#include <seqpp/Markov.h>
```

Inheritance diagram for Markov::



Public Member Functions

- **Markov** (const char *ConfFile, bool calc_rank=false)
Constructor 1 : read a configuration file.
- **Markov** (const **SequenceSet** &seqset, bool calc_rank=false, const string &prior_alpha_file=string())
Constructor 2 : Estimate the transition matrices on the sequences of seqset.
- **Markov** (const **Sequence** &seq, bool calc_rank=false, const string &prior_alpha_file=string())
Constructor 3 : Estimate the transition matrices on the sequence seq.
- **Markov** (const **Markov** &)
Constructor 4 : Copy constructor.
- **Markov** ()
Constructor 5 : Default constructor.

- **Markov** (short size, short order, bool alloc=true, const string &prior_alpha_file=string())
Constructor 6 : Minimal Constructor.
- **Markov** (const **Markov** &M1, const **Markov** &M2, const float p)
*Constructor 7 : Creation of a "mixed" Markov chain $M = p*M1 + (1-p)*M2$ */.*
- **Markov** (const gsl_rng *r, short size, short order, bool calc_rank=false)
Constructor 8 : random markov matrices.
- **Markov** (unsigned long *count, short size, short order, const string &prior_alpha_file=string(), bool calc_rank=false)
Constructor 9 : Estimate the transition matrices on a word-count.
- virtual ~**Markov** ()
Destructor.
- template<class TSeq> void **estimate** (const TSeq &tseq, unsigned long beg, unsigned long end, bool calc_rank)
Estimate the transition matrices on the sequence/sequenceset tseq.
- void **estimate** (unsigned long *count, bool decal_required, bool calc_rank=false)
Estimate the transition matrices from a word count.
- void **estimate** (const string &count_file, bool calc_rank=false)
Estimate the transition matrices from a file containing the count.
- const double * **markov_matrix** () const
access to the markov matrix
- void **draw_markov_matrix** (const gsl_rng *r)
draw at random the markov matrix
- void **free_markov_matrix** ()
free the memory allocated for markov matrix
- void **compute_stat_law** (bool force=false)
Compute the stationnary laws.
- void **free_stat_law** ()
free the memory allocated for stationnary law
- const double * **stat_law** () const
access to the stationnary distrib
- virtual int **compute_rank** ()
Computes the rank of convergence of the Markov Chain.
- void **compute_power** ()
Initialisation the _rank powers of the markov matrix.

- int **free_power** ()
free the memory allocated for the power
- double **proba_step** (long w1, long w2, int step)
transition from word w1 to word w2 in step steps
- bool **isPi** () const
- bool **isPow** () const
- bool **isMu** () const
- double & **operator**() (int index)
() operator for Markov matrix Pi elements
- double **Mu** (int index) const
Access to stationnary vector Mu elements.

Protected Attributes

- double * **_Pi**
Pointer to "Matrix" (in a vector format) of transition probabilities (= _Pis[0]).
- double * **_container**
Container of "Matrix" (in a vector format) of transition probabilities (= _containers[0]).
- double * **_Mu**
Vector of stationnary probabilities (= _Mus[0]).
- double *** **_PowPi**
Power of the _Pi matrix.

6.1.1 Detailed Description

Markov modelling, estimation and simulation.

This is a special case of a phased Markov [**PhasedMarkov**(p. 23)] model when only one phase is considered.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Markov::Markov (const SequenceSet & seqset, bool calc_rank = false, const string & prior_alpha_file = string())

Constructor 2 : Estimate the transition matrices on the sequences of seqset.

Parameters:

seqset set of sequences for estimation

calc_rank calculus of the convergence rank if true

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

6.1.2.2 Markov::Markov (const Sequence & seq, bool calc_rank = false, const string & prior_alpha_file = string())

Constructor 3 : Estimate the transition matrices on the sequence seq.

Parameters:

seq sequence for estimation

calc_rank calculus of the convergence rank if true

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

6.1.2.3 Markov::Markov (short size, short order, bool alloc = true, const string & prior_alpha_file = string()) [inline]

Constructor 6 : Minimal Constructor.

Initialises the constants of the model but not the matrix nor the stat law

Parameters:

size alphabet size

order markovian order

alloc true for matrices memory allocation

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

6.1.2.4 Markov::Markov (const Markov & M1, const Markov & M2, const float p)

Constructor 7 : Creation of a "mixed" Markov chain $M = p*M1 + (1-p)*M2$ */.

Parameters:

M1 first Markov chain object

M2 second Markov chain object

p weight of M1 in the resulting M(with $0 \leq p \leq 1$)

6.1.2.5 Markov::Markov (const gsl_rng * r, short size, short order, bool calc_rank = false) [inline]

Constructor 8 : random markov matrices.

Parameters:

r gsl random generator

size alphabet size

order markovian order

calc_rank calculus of the convergence rank if true

GSL use exple:

```

const gsl_rng_type * T;
// Choice a default generator and seed
// from environment variables
gsl_rng_env_setup();
// New created instance of the generator
T = gsl_rng_default;
gsl_rng * r = gsl_rng_alloc (T);
// Initialize/Seeds the random number generator
gsl_rng_set( r, (long)getpid() );

```

6.1.2.6 Markov::Markov (unsigned long * *count*, short *size*, short *order*, const string & *prior_alpha_file* = string(), bool *calc_rank* = false)

Constructor 9 : Estimate the transition matrices on a word-count.

Parameters:

count count of all the coded word(base size) of size order+1 for estimation
size alphabet size
order markovian order
calc_rank calculus of the convergence rank if true
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

6.1.3 Member Function Documentation

6.1.3.1 void Markov::draw_markov_matrix (const gsl_rng * *r*) [inline]

draw at random the markov matrix

Parameters:

r gsl random generator

GSL use exple:

```

const gsl_rng_type * T;
// Choice a default generator and seed
// from environment variables
gsl_rng_env_setup();
// New created instance of the generator
T = gsl_rng_default;
gsl_rng * r = gsl_rng_alloc (T);
// Initialize/Seeds the random number generator
gsl_rng_set( r, (long)(time( NULL )) );
...
...
gsl_rng_free( r );

```

6.1.3.2 void Markov::estimate (const string & *count_file*, bool *calc_rank* = false) [inline]

Estimate the transition matrices from a file containing the count.

Parameters:

count_file file containing the count in the adapted format
calc_rank calculus of the convergence rank if true

Reimplemented from **PhasedMarkov** (p. 32).

6.1.3.3 `void Markov::estimate (unsigned long * count, bool decal_required, bool calc_rank = false) [inline]`

Estimate the transition matrices from a word count.

Parameters:

count count of all the coded word(base size) of size order+1, for estimation
decal_required necessary when using a count of word from 1-word to (_order+1)-word
calc_rank calculus of the convergence rank if true

6.1.3.4 `template<class TSeq> void Markov::estimate (const TSeq & tseq, unsigned long beg, unsigned long end, bool calc_rank) [inline]`

Estimate the transition matrices on the sequence/sequenceset *tseq*.

Parameters:

tseq sequence/sequenceset for estimation
beg begin position in sequence(s) if subsequences
end end position in sequence(s) if subsequences
calc_rank calculus of the convergence rank if true

6.1.3.5 `bool Markov::isMu () const [inline]`

`_Mu == NULL ?`

6.1.3.6 `bool Markov::isPi () const [inline]`

`_Pi == NULL ?`

6.1.3.7 `bool Markov::isPow () const [inline]`

`_PowPi == NULL ?`

6.1.3.8 `double Markov::Mu (int index) const [inline]`

Access to stationnary vector Mu elements.

Parameters:

index index of the word

6.1.3.9 `double& Markov::operator() (int index)` [inline]

() operator for Markov matrix Pi elements

Parameters:

index index of the word

6.1.3.10 `double Markov::proba_step (long w1, long w2, int step)` [inline]

transition from word w1 to word w2 in step steps

if step > _rank, give stat_law(w2). You must have w1-jump, w2-jump < _nMu

Parameters:

w1 first word (of length order) as a Sequence-coded-like integer (see **Sequence**(p. 65))

w2 second word (of length order) as a Sequence-coded-like integer (see **Sequence**(p. 65))

step number of steps between w1 and w2

The documentation for this class was generated from the following files:

- **Markov.h**
- **Markov.cc**

6.2 MTDMarkov Class Reference

Mixture Transition Distribution **Markov**(p. 11) modelling, estimation and simulation.

```
#include <seqpp/MTDMarkov.h>
```

Inheritance diagram for MTDMarkov::



Public Member Functions

- `template<class TSeq> MTDMarkov (const TSeq &tseq, short mkv_order, short nbseed=NBSEED, int nbiter_max=NBITERMAX, double eps=EPS, bool log=false)`
*Constructor 1 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65).*
- `MTDMarkov (unsigned long **count, short size, short mtd_order, short mkv_order, short nbseed=NBSEED, int nbiter_max=NBITERMAX, double eps=EPS, bool log=false)`
Constructor 2 from a coded-word count.
- `void estimate (unsigned long **count, bool decal_required, short mkv_order, short nbseed, int nbiter_max, double eps, bool log)`
performs the estimation [used by each constructor]
- `~MTDMarkov ()`
Destructor.

Protected Attributes

- `mtd_core * _mtdcore`
Parcimonious Context Trees.

6.2.1 Detailed Description

Mixture Transition Distribution **Markov**(p. 11) modelling, estimation and simulation.

MTDMarkov is a **Markov**(p. 11) object with a different estimation step. This object performs the estimation with the Mixture Transition Distribution algorithm and then transforms, once per phase, the MTD model in a markovian matrix.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 `template<class TSeq> MTDMarkov::MTDMarkov (const TSeq & tseq, short mkv_order, short nbseed = NBSEED, int nbiter_max = NBITERMAX, double eps = EPS, bool log = false) [inline]`

Constructor 1 from a `SequenceSet`(p. 68) or a `Sequence`(p. 65).

Parameters:

tseq a set of sequences or a sequence for estimation
mkv_order markovian order on the markovian matrix of the MTD
nbseed number of seeds for the EM algorithm
nbiter_max maximum iterations number of the EM algorithm
eps value of the epsilon of the EM algorithm
log true to log the successive likelihood values

6.2.2.2 `MTDMarkov::MTDMarkov (unsigned long ** count, short size, short mtd_order, short mkv_order, short nbseed = NBSEED, int nbiter_max = NBITERMAX, double eps = EPS, bool log = false) [inline]`

Constructor 2 from a coded-word count.

Parameters:

count count of all the coded word(base size) of size order+1 for each phase, for estimation
size alphabet size
mtd_order markovian order for the resulting model
mkv_order markovian order on the markovian matrices used in the MTD
nbseed number of seeds for the EM algorithm
nbiter_max maximum iterations number of the EM algorithm
eps value of the epsilon of the EM algorithm
log true to log the successive likelihood values

The documentation for this class was generated from the following file:

- `MTDMarkov.h`

6.3 Partition Class Reference

Dealer of alphabet partitions.

```
#include <seqpp/Partition.h>
```

Public Member Functions

- **Partition** ()
empty constructor
- **Partition** (short alphabet_size)
constructor with integrated creation of the set of partitions
- **Partition** (const **Translator** &alphabet, const string &pfile)
Constructor from a file containing the partitions or the list of synonymous tokens.
- **Partition** (const **Partition** &part)
copy constructor
- **~Partition** ()
destructor
- **Partition** & **operator=** (**Partition** &part)
operator =
- const vector< vector< short > > & **list_possible_elements** ()
initialize the list_possible_elements of possible tokens'subset in a partition and return it
- const vector< vector< short > > & **synonymous** () const
returns the synonymous list
- const vector< int > & **nbpartition_with_nbelement** () const
return the number of partition per number of elements
- const vector< int > & **compute_nbpartition_with_nbelement** ()
compute and return the number of partition per number of elements, if no cutoff method previously called
- long **tell_nbposs** () const
returns the number of possible tokens'subset in a partition
- short **tell_size** () const
returns the size of the space to partition
- bool **create_default** ()
create the default motif if it not exists
- int **motifdefault** () const
return the index of the default motif

- bool **motifdefault_skip** () const
true if the default motif is added to `_list_possib` at the end, false if it already exists
- long **tell_cardinal** () const
cardinal of the partition
- const_iterator **begin** () const
first possibility of the partition
- const_iterator **end** () const
end of list of possibilities (not a possibility)

6.3.1 Detailed Description

Dealer of alphabet partitions.

Let a partition of an alphabet be a set of tokens'subset, i.e. a division of the alphabet into subset. The Partition object gives 2 possibilities:

- to compute the overall set of possible partitions (automatically generated) given the alphabet
- to compute the overall set of possible partitions (automatically generated) given a synonymous pseud-alphabet: by declaring synonymous tokens, it is possible to group tokens as a single predictor so that the number of partitions is lower.

In this case, a configuration file with the top key word "`#Synonymous`", containing the lists of synonymous tokens, is required.

Exple:

```
#Synonymous
a t
g c
```

- to input a selected set of partitions. In this case, in a configuration file after a "`#Partition`" on the first line, each partition is represented as a list of tokens'subset delimited by a "|", each subset being composed with tokens of the alphabet separated by space

Exple:

```
#Partition
a | g | c | t
a g | c | t
a c t | g
```

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Partition::Partition (short *alphabet_size*) [inline]

constructor with integrated creation of the set of partitions

Parameters:

alphabet_size size of the alphabet

6.3.2.2 Partition::Partition (const Translator & *alphabet*, const string & *pfile*) [inline]

Constructor from a file containing the partitions or the list of synonymous tokens.

Parameters:

alphabet a link to a **Translator**(p. 70) object required for the file reading

pfile partitions file

-> 1st possibility: after a "#Partition" on the first line, each partition is represented as a list of tokens'subset delimited by a "|", each subset being composed with tokens of the alphabet separated by space

Example:

```
#Partition  
a g | c t
```

-> 2nd possibility: after a "#Synonymous" on the first line, tokens are grouped (separated by spaces) into synonymous classes on each line

Example:

```
#Synonymous  
a t  
g c
```

The documentation for this class was generated from the following file:

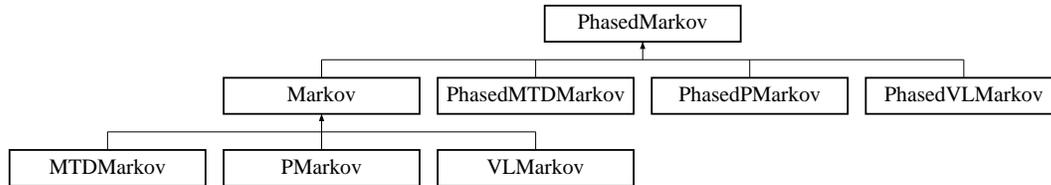
- **Partition.h**

6.4 PhasedMarkov Class Reference

Phased **Markov**(p.11) modelling, estimation and simulation.

```
#include <seqpp/PhasedMarkov.h>
```

Inheritance diagram for PhasedMarkov::



Public Member Functions

- **PhasedMarkov** (const string &markov_file, bool calc_rank=false)
Constructor 1 : read a configuration file.
- **PhasedMarkov** (const **SequenceSet** &seqset, short phase, short initial_phase=0, bool calc_rank=false, const string &prior_alpha_file=string())
Constructor 2 : Estimate the transition matrices on the sequences of seqset.
- **PhasedMarkov** (const **Sequence** &seq, short phase, short initial_phase=0, bool calc_rank=false, const string &prior_alpha_file=string())
Constructor 3 : Estimate the transition matrices on the sequence seq.
- **PhasedMarkov** (const **PhasedMarkov** &phm)
Constructor 4 : Copy constructor.
- **PhasedMarkov** ()
Constructor 5 : Default constructor.
- **PhasedMarkov** (short size, short order, short phase, bool alloc=true, const string &prior_alpha_file=string())
Constructor 6 : Minimal Constructor.
- **PhasedMarkov** (const **PhasedMarkov** &M1, const **PhasedMarkov** &M2, const float p)
*Constructor 7 : Creation of a "mixed" **Markov**(p.11) chain $M = p*M1 + (1-p)*M2$ */.*
- **PhasedMarkov** (const **SequenceSet** &seqset, const vector< int > &Indseq, short phase, short initial_phase=0, bool calc_rank=false, const string &prior_alpha_file=string())
Constructor 8 : Estimation of the transition matrix based on the sequences of seqset given in Indseq.
- **PhasedMarkov** (const gsl_rng *r, short size, short order, short phase, bool calc_rank=false)
Constructor 9 : random markov matrices.

- **PhasedMarkov** (unsigned long **count, short size, short order, short phase, short initial_phase=0, bool calc_rank=false, const string &prior_alpha_file=string())
Constructor 10 Estimate the transition matrices on a word-count.
- virtual ~**PhasedMarkov** ()
Destructor.
- template<class TSeq> void **estimate** (const TSeq &tseq, short phase, short initial_phase, unsigned long beg, unsigned long end, bool calc_rank=false, bool count_again=true)
Estimate the transition matrices on the sequence/sequenceset tseq.
- void **estimate** (const string &count_file, bool calc_rank=false)
Estimate the transition matrices from a file containing the count.
- void **estimate** (unsigned long **count, bool decal_required, bool calc_rank=false)
Estimate the transition matrices from a word count.
- const double ** **markov_matrices** () const
access to the markov matrix(ces)
- const double * **markov_matrix** (short numphase) const
access to the numphase-th markov matrix
- void **draw_markov_matrices** (const gsl_rng *r)
draw at random the markov matrices
- virtual void **new_markov_matrices** ()
allocate the memory for markov matrices
- virtual void **free_markov_matrices** ()
free the memory allocated for markov matrices
- double **total_variation** (const **PhasedMarkov** &M)
*Total variation distance between *this and M.*
- void **compute_stat_laws** (bool force=false)
Compute the stationnary laws.
- const double * **stat_law** (short numphase=0) const
access to the stationnary distrib in phase numphase
- void **free_stat_laws** ()
free the memory allocated for stationnary laws
- void **compute_init_law** (double *MuInit, const **SequenceSet** &seqset) const
Get the empirical relative frequency of the first order+1 letters on the set of sequences "seqset".
- virtual int **compute_rank** ()
*Computes the rank of convergence of the **Markov**(p.11) Chain.*

- virtual long **nb_parameters** () const
return the number of effective parameters
- void **link_to_translator** (const **Translator** &trans)
*link to a **Translator**(p. 70) object to use proba methods with strings*
- double **proba_c** (const string &word, Coder &coder, short numphase=0) const
Stationnary proba of a word(size greater than _order) conditionnaly of its first letters (!use link_to_translator before!).
- double **proba** (const string &word, Coder &coder, short numphase=0) const
Stationnary proba of a word.
- double **proba_c** (const vector< short > &word, Coder &coder, short numphase=0) const
Stationnary proba of a word(size greater than _order) conditionnaly of its first letters.
- double **proba** (const vector< short > &word, Coder &coder, short numphase=0) const
Stationnary proba of a word.
- double **proba_c** (long word, int lw=-1, long jump=-1, short numphase=0) const
Stationnary proba of a word(size greater than _order) conditionnaly of its first letters.
- double **proba** (long word, int lw=-1, long jump=-1, short numphase=0) const
Stationnary proba of a word.
- double **proba_c** (const long *seq, long tbegin, long tend, short numphase=0) const
Stationnary proba of the word seq[tbegin...tend](size greater than _order) conditionnaly of its first letters with seq[tend] in phase numphase.
- double **proba** (const long *seq, long tbegin, long tend, short numphase=0) const
Stationnary proba of the word seq[tbegin...tend] with seq[tend] in phase numphase.
- double **log_likelihood** (const **SequenceSet** &seqset, short initial_phase=0, short numphase=-1) const
loglikelihood of a set of sequence
- double **log_ratio_likelihood** (const **SequenceSet** &seqset, const **PhasedMarkov** &M, short initial_phase1=0, short initial_phase2=0) const
Calculation of the logarithm of the ratio of the probability of observing "seq" under "this" distribution and "M".
- double **log_likelihood** (const **Sequence** &seq, short initial_phase=0, short numphase=-1) const
loglikelihood of a sequence
- double **log_ratio_likelihood** (const **Sequence** &seq, const **PhasedMarkov** &M, short initial_phase1=0, short initial_phase2=0) const
Calculation of the logarithm of the ratio of the probability of observing "seq" under "this" distribution and "M".
- template<class TSeq> double **BIC** (const TSeq &tseq, short initial_phase=0) const

*BIC of sequences ($BIC = -2 * \log \text{likelihood} + \text{nbparam} * \log(\text{length})$).*

- `template<class TSeq> double AIC (const TSeq &tseq, short initial_phase=0) const`
*AIC of a set of sequences ($AIC = -2 * \log \text{likelihood} + 2 * \text{nbparam}$).*
- `template<class TSeq1, class TSeq2> double post_log_likelihood (const TSeq1 &tseq_train, const TSeq2 &tseq_eval, bool force=false, short initial_phase_train=0, short initial_phase_eval=0)`
compute the mean posterior likelihood over the parameters
- `void print (const string &FileOut)`
Print a summary of the object.
- `void print (ofstream &Out) const`
Print a summary of the object.
- `int tell_size () const`
Returns the alphabet size.
- `int tell_rank () const`
Returns the convergence rank.
- `int tell_order () const`
Returns the order.
- `int tell_phase () const`
Returns the phase.
- `int nMu () const`
size of the stat law vector
- `int nPi () const`
size of the matrix
- `double Pi (int index, int p=0) const`
*Access to **Markov**(p.11) matrix *Pi*.*
- `double & operator() (int index, int p=0)`
*() operator for **Markov**(p.11) matrix *Pi* elements*
- `double Mu (int index, int p=0) const`
*Access to stationnary vector *Mu* elements.*
- `bool isPis () const`
_Pis != NULL ?
- `bool isMus () const`
_Mus != NULL ?
- `short nextPhase (short p) const`

Give the phase following p.

- short **prevPhase** (short p) const
Give the phase preceding p.
- bool **Stochasticity** ()
Verify stochasticity of the `_Pis[]` and eventually rescale it.
- void **file_to_count** (const string &src_file, unsigned long **dest_count)
fill a count from a file

Protected Member Functions

- bool **isNextPhase** () const
`_nextPhase != NULL` ?
- bool **isPrevPhase** () const
`_prevPhase != NULL` ?

Protected Attributes

- short **_phase**
Phase of the model.
- double ** **_Pis**
Pointer to "Matrices" (in vector format) of transition probabilities for each phase.
- double ** **_containers**
Container of "Matrices" (in vector format) of transition probabilities for each phase.
- double ** **_Mus**
Vector of stationnary probabilities for each phase.
- short **_size**
Size of the alphabet.
- short **_order**
Order of the model (the same at each phase).
- long **_nPi**
Dim of Pi : $_{size}^{order+1}$.
- long **_nMu**
Dim of Mu : $_{size}^{order}$.
- long **_nb_param**
number of effective parameters

- `int _rank`
How many steps to converge to μ ?
- `long _jump`
jump to the codes of $_{order}+1$ letters when Sequence-like code
- `short * _nextPhase`
(Optimization) For each phase, give the next phase
- `short * _prevPhase`
(Optimization) For each phase, give the previous phase
- `const Translator * _trans`
link to a translator object for the use of proba methods
- `double _postloglike`
current posterior likelihood from a training [set of] sequence
- `vector< vector< double > > _prior_alpha`
prior on the counts, one value per alphabet element, and for each phase

6.4.1 Detailed Description

Phased **Markov**(p. 11) modelling, estimation and simulation.

This is generalization of a **Markov**(p. 11) chain, using different matrices in function of the considered position in the sequence. The phase is variable.

For example, if we consider 3 phases, and we note respectively P_{i1} , P_{i2} and P_{i3} , the three transition matrices, the **Markov**(p. 11) sequences will be generated by the indices 123123123123... In a DNA modelisation (genomic field), this is useful to take into account the fact that a coding region is read by 3 bases-blocks. The order of the **Markov**(p. 11) Model, i.e. the number of previous states necessary to determine the distribution of the current state, is variable. It is assume here that this order is the same in all the phases.

Methods are implemented for Markovian transition matrix estimation, stationary distribution calculus, word probabilities, total variation distance between two Markovian matrices, and further. The efficiency of eigenproblems computation is ensured by the use of the implicitly restarted Arnoldi algorithm.

Simulations are also possible.

6.4.2 Constructor & Destructor Documentation

- #### 6.4.2.1 PhasedMarkov::PhasedMarkov (const SequenceSet & seqset, short phase, short initial_phase = 0, bool calc_rank = false, const string & prior_alpha_file = string())

Constructor 2 : Estimate the transition matrices on the sequences of seqset.

Parameters:

seqset set of sequences for estimation
phase selected phase
initial_phase phase of the first element of each sequence
calc_rank calculus of the convergence rank if true
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")

6.4.2.2 PhasedMarkov::PhasedMarkov (const Sequence & seq, short phase, short initial_phase = 0, bool calc_rank = false, const string & prior_alpha_file = string())

Constructor 3 : Estimate the transition matrices on the sequence seq.

Parameters:

seq sequence for estimation
phase selected phase
initial_phase phase of the first element of each sequence
calc_rank calculus of the convergence rank if true
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")

6.4.2.3 PhasedMarkov::PhasedMarkov (short size, short order, short phase, bool alloc = true, const string & prior_alpha_file = string())

Constructor 6 : Minimal Constructor.

Initialises the constants of the model but not the matrices nor the stat laws

Parameters:

size alphabet size
order markovian order
phase selected phase
alloc true for matrices memory allocation
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")

6.4.2.4 PhasedMarkov::PhasedMarkov (const PhasedMarkov & M1, const PhasedMarkov & M2, const float p)

Constructor 7 : Creation of a "mixed" Markov(p.11) chain $M = p*M1 + (1-p)*M2$ */.

Parameters:

M1 first Markov(p.11) chain object
M2 second Markov(p.11) chain object
p weight of M1 in the resulting M(with $0 \leq p \leq 1$)

6.4.2.5 PhasedMarkov::PhasedMarkov (const SequenceSet & *seqset*, const vector<int > & *Indseq*, short *phase*, short *initial_phase* = 0, bool *calc_rank* = false, const string & *prior_alpha_file* = string())

Constructor 8 : Estimation of the transition matrix based on the sequences of *seqset* given in *Indseq*.

Parameters:

seqset set of sequences for estimation

Indseq index of selected sequences

phase selected phase

initial_phase phase of the first element of each sequence

calc_rank calculus of the convergence rank if true

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")

6.4.2.6 PhasedMarkov::PhasedMarkov (const gsl_rng * *r*, short *size*, short *order*, short *phase*, bool *calc_rank* = false)

Constructor 9 : random markov matrices.

Parameters:

r gsl random generator

size alphabet size

order markovian order

phase selected phase

calc_rank calculus of the convergence rank if true

GSL use exple:

```
const gsl_rng_type * T;
// Choice a default generator and seed
// from environment variables
gsl_rng_env_setup();
// New created instance of the generator
T = gsl_rng_default;
gsl_rng * r = gsl_rng_alloc (T);
// Initialize/Seeds the random number generator
gsl_rng_set( r, (long)getpid() );
...
...
gsl_rng_free( r );
```

6.4.2.7 PhasedMarkov::PhasedMarkov (unsigned long ** *count*, short *size*, short *order*, short *phase*, short *initial_phase* = 0, bool *calc_rank* = false, const string & *prior_alpha_file* = string())

Constructor 10 Estimate the transition matrices on a word-count.

Parameters:

count count of all the coded word(base size) of size *order*+1 for each phase,for estimation

size alphabet size

order markovian order

phase selected phase

initial_phase phase of the first element of each sequence

calc_rank calculus of the convergence rank if true

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")

6.4.3 Member Function Documentation

6.4.3.1 `template<class TSeq> double PhasedMarkov::AIC (const TSeq & tseq, short initial_phase = 0) const [inline]`

AIC of a set of sequences (AIC = $-2*\log\text{likelihood} + 2*\text{nbparam}$).

Parameters:

tseq a sequence or a set of sequences

initial_phase phase of the first element of each sequence

6.4.3.2 `template<class TSeq> double PhasedMarkov::BIC (const TSeq & tseq, short initial_phase = 0) const [inline]`

BIC of sequences (BIC = $-2*\log\text{likelihood} + \text{nbparam}*\log(\text{length})$).

Parameters:

tseq a sequence or a set of sequences

initial_phase phase of the first element of each sequence

6.4.3.3 `void PhasedMarkov::draw_markov_matrices (const gsl_rng * r)`

draw at random the markov matrices

Parameters:

r gsl random generator

GSL use exple:

```
const gsl_rng_type * T;
// Choice a default generator and seed
// from environment variables
gsl_rng_env_setup();
// New created instance of the generator
T = gsl_rng_default;
gsl_rng * r = gsl_rng_alloc (T);
// Initialize/Seeds the random number generator
gsl_rng_set( r, (long)getpid() );
...
...
gsl_rng_free( r );
```

6.4.3.4 void PhasedMarkov::estimate (unsigned long ** *count*, bool *decal_required*, bool *calc_rank* = false)

Estimate the transition matrices from a word count.

Parameters:

count count of all the coded word(base size) of size order+1 for each phase, for estimation
decal_required necessary when using a count of word from 1-word to (_order+1)-word
calc_rank calculus of the convergence rank if true

6.4.3.5 void PhasedMarkov::estimate (const string & *count_file*, bool *calc_rank* = false) [inline]

Estimate the transition matrices from a file containing the count.

Parameters:

count_file file containing the count in the adapted format
calc_rank calculus of the convergence rank if true

Reimplemented in **Markov** (p. 15).

6.4.3.6 template<class TSeq> void PhasedMarkov::estimate (const TSeq & *tseq*, short *phase*, short *initial_phase*, unsigned long *beg*, unsigned long *end*, bool *calc_rank* = false, bool *count_again* = true) [inline]

Estimate the transition matrices on the sequence/sequenceset *tseq*.

Parameters:

tseq sequence/sequenceset for estimation
phase selected phase
initial_phase phase of the first element of each sequence
beg begin position in sequence(s) if subsequences
end end position in sequence(s) if subsequences
calc_rank calculus of the convergence rank if true
count_again false if the word-count already performed before the use of this method

6.4.3.7 void PhasedMarkov::file_to_count (const string & *src_file*, unsigned long ** *dest_count*)

fill a count from a file

Parameters:

src_file file containing the count, phase per phase, in alphabetical order
dest_count destination count (must be _phase X _nPi)

6.4.3.8 `double PhasedMarkov::log_likelihood (const Sequence & seq, short initial_phase = 0, short numphase = -1) const`

loglikelihood of a sequence

Parameters:

seq sequence

initial_phase phase of the first element of each sequence

numphase likelihood in only the numphase phase, -1 if sum of the phases

6.4.3.9 `double PhasedMarkov::log_likelihood (const SequenceSet & seqset, short initial_phase = 0, short numphase = -1) const`

loglikelihood of a set of sequence

Parameters:

seqset set of sequence

initial_phase phase of the first element of each sequence

numphase likelihood in only the numphase phase, -1 if sum of the phases

6.4.3.10 `double PhasedMarkov::log_ratio_likelihood (const Sequence & seq, const PhasedMarkov & M, short initial_phase1 = 0, short initial_phase2 = 0) const`

Calculation of the logarithm of the ratio of the probability of observing "seq" under "this" distribution and "M".

Parameters:

seq sequence

M alternative **Markov**(p. 11) chain

initial_phase1 phase of the first element of each sequence considered in model *this

initial_phase2 phase of the first element of each sequence considered in model M

REMARKS : NO verification is done on the compatibility of the **Markov**(p. 11) chains

CAREFUL: this calculus is performed by scanning the sequence to avoid pbm of too low likelihood

6.4.3.11 `double PhasedMarkov::log_ratio_likelihood (const SequenceSet & seqset, const PhasedMarkov & M, short initial_phase1 = 0, short initial_phase2 = 0) const`

Calculation of the logarithm of the ratio of the probability of observing "seq" under "this" distribution and "M".

Parameters:

seqset set of sequences

M alternative **Markov**(p. 11) chain

initial_phase1 phase of the first element of each sequence considered in model *this

initial_phase2 phase of the first element of each sequence considered in model M

REMARKS : NO verification is done on the compatibility of the **Markov**(p.11) chains

CAREFUL: this calculus is performed by scanning the sequence to avoid pbm of too low likelihood

6.4.3.12 double PhasedMarkov::Mu (int *index*, int *p* = 0) const [inline]

Access to stationnary vector Mu elements.

Parameters:

index index of the word

p selected phase

6.4.3.13 double& PhasedMarkov::operator() (int *index*, int *p* = 0) [inline]

() operator for **Markov**(p.11) matrix Pi elements

Parameters:

index index of the word

p selected phase

6.4.3.14 double PhasedMarkov::Pi (int *index*, int *p* = 0) const [inline]

Access to **Markov**(p.11) matrix Pi.

Parameters:

index index of the word

p selected phase

6.4.3.15 template<class TSeq1, class TSeq2> double PhasedMarkov::post_log_likelihoood (const TSeq1 & *tseq_train*, const TSeq2 & *tseq_eval*, bool *force* = false, short *initial_phase_train* = 0, short *initial_phase_eval* = 0) [inline]

compute the mean posterior likelihood over the parameters

Parameters:

tseq_train [set of] sequence for the training step

tseq_eval [set of] sequence for the evaluation step

force "true" to force the re-calculation on tseq_train. Default => "false"

initial_phase_train phase of the first element of each sequence for the training step

initial_phase_eval phase of the first element of each sequence for the evaluation step

6.4.3.16 void PhasedMarkov::print (const string & *FileOut*) [inline]

Print a summary of the object.

The estimation results can be saved in such a representation:

```
# 1 <- Order of the phased Markov chain
# 2 <- Phase
# 4 <- Alphabet size
# 19 steps <- Convergence to the stationary distribution
# Phase nř0
# Transition matrix:
0.3945322543    0.1652811616    0.1535033485    0.2866832356
etc.....
# Stationary Probability:
0.3127105148    0.2114684268    0.1783495332    0.2974715251
# Phase nř1
# Transition matrix:
0.3923961961    0.163516403     0.1521005152    0.2919868858
etc.....
# Stationary Probability:
0.3135417652    0.2089660861    0.1771006767    0.300391472
```

6.4.3.17 double PhasedMarkov::proba (const long * *seq*, long *tbeg*, long *tend*, short *numphase* = 0) const

Stationary proba of the word *seq*[*tbeg*...*tend*] with *seq*[*tend*] in phase *numphase*.

Parameters:

- seq* Sequence-like coded sequence(see *Sequence*). The Markov-order for the code must be the same than *_order*
- tbeg* begin position of the word
- tend* end position of the word
- numphase* phase of the last letter of word

6.4.3.18 double PhasedMarkov::proba (long *word*, int *lw* = -1, long *jump* = -1, short *numphase* = 0) const

Stationary proba of a word.

Parameters:

- word* word as a *Sequence*-coded-like integer (see *Sequence*(p. 65))
- lw* length of the word. Default => *order*+1
- jump* *Sequence*-coded-like jump (see *Sequence*(p. 65)). Default => *jump*[*order*]
- numphase* phase of the last letter of word

6.4.3.19 double PhasedMarkov::proba (const vector< short > & *word*, *Coder* & *coder*, short *numphase* = 0) const

Stationary proba of a word.

Parameters:

word word as a vector of short
coder required Coder object, from a **PrimaryCount**(p. 55) object for exple
numphase phase of the last letter of word

6.4.3.20 double **PhasedMarkov::proba** (const string & *word*, Coder & *coder*, short *numphase* = 0) const

Stationnary proba of a word.

Parameters:

word word as a string -a **Translator**(p. 70) object is required to process string to int-
coder required Coder object, from a **PrimaryCount**(p. 55) object for exple
numphase phase of the last letter of word

6.4.3.21 double **PhasedMarkov::proba_c** (const long * *seq*, long *tbeg*, long *tend*, short *numphase* = 0) const

Stationnary proba of the word seq[*tbeg*...*tend*](size greater than *_order*) conditionnaly of its first letters with seq[*tend*] in phase *numphase*.

Parameters:

seq Sequence-like coded sequence(see Sequence). The Markov-order for the code must be the same than *_order*
tbeg begin position of the word
tend end position of the word
numphase phase of the last letter of word

6.4.3.22 double **PhasedMarkov::proba_c** (long *word*, int *lw* = -1, long *jump* = -1, short *numphase* = 0) const

Stationnary proba of a word(size greater than *_order*) conditionnaly of its first letters.

Parameters:

word word as a Sequence-coded-like integer (see **Sequence**(p. 65))
lw length of the word. Default => order+1
jump Sequence-coded-like jump (see **Sequence**(p. 65)). Default => jump[order]
numphase phase of the last letter of word

6.4.3.23 double **PhasedMarkov::proba_c** (const vector< short > & *word*, Coder & *coder*, short *numphase* = 0) const

Stationnary proba of a word(size greater than *_order*) conditionnaly of its first letters.

Parameters:

word word as a vector of short
coder required Coder object, from a **PrimaryCount**(p. 55) object for exple
numphase phase of the last letter of word

6.4.3.24 double PhasedMarkov::proba_c (const string & *word*, Coder & *coder*, short *numphase* = 0) const

Stationnary proba of a word(size greater than `_order`) conditionnaly of its first letters (!use `link_` - `to_translator` before!).

Parameters:

word word as a string -a **Translator**(p. 70) object is required to process string to int-
coder required Coder object, from a **PrimaryCount**(p. 55) object for exple
numphase phase of the last letter of word

The documentation for this class was generated from the following files:

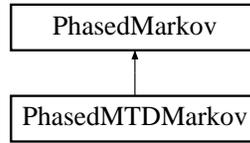
- **PhasedMarkov.h**
- **PhasedMarkov.cc**

6.5 PhasedMTDMarkov Class Reference

Phased Mixture Transition Distribution **Markov**(p. 11) modelling, estimation and simulation.

```
#include <seqpp/PhasedMTDMarkov.h>
```

Inheritance diagram for PhasedMTDMarkov::



Public Member Functions

- `template<class TSeq> PhasedMTDMarkov (const TSeq &tseq, short mkv_order, short phase, short initial_phase=0, short nbseed=NBSEED, int nbiter_max=NBITERMAX, double eps=EPS, bool log=false)`

*Constructor 1 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65).*

- `PhasedMTDMarkov (unsigned long **count, short size, short mtd_order, short mkv_order, short phase, short initial_phase=0, short nbseed=NBSEED, int nbiter_max=NBITERMAX, double eps=EPS, bool log=false)`

Constructor 2 from a coded-word count.

- `void estimate (unsigned long **count, bool decal_required, short mkv_order, short phase, short initial_phase, short nbseed, int nbiter_max, double eps, bool log)`

performs the estimation [used by each constructor]

- `~PhasedMTDMarkov ()`

Destructor.

6.5.1 Detailed Description

Phased Mixture Transition Distribution **Markov**(p. 11) modelling, estimation and simulation.

PhasedMTDMarkov is a **PhasedMarkov**(p. 23) object with a different estimation step. This object performs the estimation with the Mixture Transition Distribution algorithm and then transforms, once per phase, the MTD model in a markovian matrix.

6.5.2 Constructor & Destructor Documentation

- #### 6.5.2.1 `template<class TSeq> PhasedMTDMarkov::PhasedMTDMarkov (const TSeq & tseq, short mkv_order, short phase, short initial_phase = 0, short nbseed = NBSEED, int nbiter_max = NBITERMAX, double eps = EPS, bool log = false) [inline]`

Constructor 1 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65).

Parameters:

tseq a set of sequences or a sequence for estimation
mkv_order markovian order on the markovian matrix of the MTD
phase phase short
initial_phase phase of the first element of each sequence
nbseed number of seeds for the EM algorithm
nbiter_max maximum iterations number of the EM algorithm
eps value of the epsilon of the EM algorithm
log true to log the successive likelihood values

6.5.2.2 PhasedMTDMarkov::PhasedMTDMarkov (unsigned long ** *count*, short *size*, short *mtd_order*, short *mkv_order*, short *phase*, short *initial_phase* = 0, short *nbseed* = NBSEED, int *nbiter_max* = NBITERMAX, double *eps* = EPS, bool *log* = false) [inline]

Constructor 2 from a coded-word count.

Parameters:

count count of all the coded word(base size) of size order+1 for each phase, for estimation
size alphabet size
mtd_order markovian order for the resulting model
mkv_order markovian order on the markovian matrices used in the MTD
phase phase
initial_phase phase of the first element of each sequence
nbseed number of seeds for the EM algorithm
nbiter_max maximum iterations number of the EM algorithm
eps value of the epsilon of the EM algorithm
log true to log the successive likelihood values

The documentation for this class was generated from the following file:

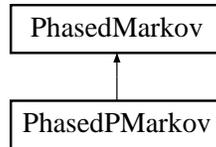
- **PhasedMTDMarkov.h**

6.6 PhasedPMarkov Class Reference

Phased Parsimonious **Markov**(p. 11) modelling, estimation and simulation.

```
#include <seqpp/PhasedPMarkov.h>
```

Inheritance diagram for PhasedPMarkov::



Public Member Functions

- **PhasedPMarkov** (**Partition** &part, const **SequenceSet** &seqset, short phase, short initial_phase=0, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const string &xmlfile=string())
Constructor 1 from a SequenceSet(p. 68).
- **PhasedPMarkov** (**Partition** &part, const **Sequence** &seq, short phase, short initial_phase=0, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const **Translator** &trans=**Translator**(), const string &xmlfile=string())
Constructor 2 from a Sequence(p. 65).
- **PhasedPMarkov** (**Partition** &part, unsigned long **count, short size, short order, short phase, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const **Translator** &trans=**Translator**(), const string &xmlfile=string())
Constructor 3 from a coded-word count.
- **PhasedPMarkov** (**Partition** &part, const string &count_file, short size, short order, short phase, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const **Translator** &trans=**Translator**(), const string &xmlfile=string())
Constructor 4 from a file containing coded-word count.
- **PhasedPMarkov** (**Partition** &part, short size, short order, short phase, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., bool alloc=true)
Basic Constructor 5. No estimation.
- void **select** (unsigned long **count, bool decal_required, const **Translator** &trans=**Translator**(), const string &xmlfile=string())
performs the A Posteriori Maximisation
- template<class TSeq1, class TSeq2> double **mean_post_log_likelihood** (const TSeq1 &tseq_train, const TSeq2 &tseq_eval, short initial_phase_train=0, short initial_phase_eval=0)
- double **mean_post_log_likelihood** (unsigned long **count_train, bool decal_required_t, unsigned long **count_eval, bool decal_required_e)
return the mean post likelihood over the parameters and the trees

- `template<class TSeq> double mean_post_log_likelihood (const TSeq &tseq_eval, short initial_phase_eval=0)`
- `double mean_post_log_likelihood (unsigned long **count_eval, bool decal_required_e)`
return the mean post likelihood over the parameters and the trees
- `double mean_post_log_likelihood ()`
return the mean post likelihood over the parameters and the trees
- `template<class TSeq> double post_log_likelihood (const TSeq &tseq_eval, short initial_phase_eval=0)`
compute the mean posterior likelihood over the parameters
- `double post_log_likelihood (unsigned long **count_eval, bool decal_required_e)`
compute the mean posterior likelihood over the parameters
- `void draw (unsigned long **count, bool decal_required, gsl_rng *r, const Translator &trans=Translator(), const string &xmlfile=string())`
draws a model
- `void draw (gsl_rng *r, const Translator &trans=Translator(), const string &xmlfile=string())`
draws a model
- `void info_nb_leaves () const`
returns info on the number of leaves for each phase
- `~PhasedPMarkov ()`
Destructor.

Protected Attributes

- `vector< pmm_forest * > _p_f`
Parcimonious Context Trees.

6.6.1 Detailed Description

Phased Parcimonious **Markov**(p. 11) modelling, estimation and simulation.

PhasedPMarkov is a **PhasedMarkov**(p. 23) object with a different estimation step. This object performs the estimation with the Parcimonious **Markov**(p. 11) algorithm and then transforms, once per phase, the parcimonious context tree in a markovian matrix. xml outputs can be activated to save the associated trees.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 `PhasedPMarkov::PhasedPMarkov (Partition & part, const SequenceSet & seqset, short phase, short initial_phase = 0, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const string & xmlfile = string())` [inline]

Constructor 1 from a `SequenceSet`(p. 68).

Parameters:

- part* associated partition
- seqset* a set of sequences for estimation
- phase* phase
- initial_phase* phase of the first element of each sequence
- prior_alpha_file* file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")
- motif_prior* activates a weight-function on the priors, weight proportionnal on each motif in the tree
- penalty* penalty on the leaves number, by default 0
- xmlfile* xmlfile for tree saving (if xml2 activated)

6.6.2.2 `PhasedPMarkov::PhasedPMarkov (Partition & part, const Sequence & seq, short phase, short initial_phase = 0, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const Translator & trans = Translator(), const string & xmlfile = string())` [inline]

Constructor 2 from a `Sequence`(p. 65).

Parameters:

- part* associated partition
- seq* sequence for estimation
- phase* phase
- initial_phase* phase of the first element of each sequence
- prior_alpha_file* file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")
- motif_prior* activates a weight-function on the priors, weight proportionnal on each motif in the tree
- penalty* penalty on the leaves number, by default 0
- trans* a `Translator`(p. 70) is required only for the xml saving
- xmlfile* xmlfile for tree saving (if xml2 activated)

6.6.2.3 `PhasedPMarkov::PhasedPMarkov (Partition & part, unsigned long ** count, short size, short order, short phase, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const Translator & trans = Translator(), const string & xmlfile = string())` [inline]

Constructor 3 from a coded-word count.

Parameters:

part associated partition
count count of all the coded word(base size) of size order+1 for each phase, for estimation
size alphabet size
order markovian order associated to the word count
phase phase
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")
motif_prior activates a weight-function on the priors, weight proportionnal on each motif in the tree
penalty penalty on the leaves number, by default 0
trans a **Translator**(p. 70) is required only for the xml saving
xmlfile xmlfile for tree saving (if xml2 activated)

6.6.2.4 PhasedPMarkov::PhasedPMarkov (Partition & part, const string & count_file, short size, short order, short phase, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const Translator & trans = Translator(), const string & xmlfile = string()) [inline]

Constructor 4 from a file containing coded-word count.

Parameters:

part associated partition
count_file file with count
size alphabet size
order markovian order associated to the word count
phase phase
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")
motif_prior activates a weight-function on the priors, weight proportionnal on each motif in the tree
penalty penalty on the leaves number, by default 0
trans a **Translator**(p. 70) is required only for the xml saving
xmlfile xmlfile for tree saving (if xml2 activated)

6.6.2.5 PhasedPMarkov::PhasedPMarkov (Partition & part, short size, short order, short phase, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., bool alloc = true) [inline]

Basic Constructor 5. No estimation.

Parameters:

part associated partition
size alphabet size

order markovian order associated to the word count

phase phase

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element, and for each phase (separated by a "#Phase i")

motif_prior activates a weight-function on the priors, weight proportionnal on each motif in the tree

penalty penalty on the leaves number, by default 0

alloc true for matrices memory allocation

6.6.3 Member Function Documentation

6.6.3.1 `void PhasedPMarkov::draw (gsl_rng * r, const Translator & trans = Translator(), const string & xmlfile = string()) [inline]`

draws a model

Parameters:

r random generator

trans a `Translator`(p. 70) is required for xml tree saving (if xml2 activated)

xmlfile xmlfile for tree saving (if xml2 activated)

6.6.3.2 `void PhasedPMarkov::draw (unsigned long ** count, bool decal_required, gsl_rng * r, const Translator & trans = Translator(), const string & xmlfile = string()) [inline]`

draws a model

Parameters:

count count of all the coded word(base size) of size order+1 for each phase, for estimation

decal_required necessary when using a count of word from 1-word to (`_max_depth`+1)-word

r gsl random generator

trans a `Translator`(p. 70) is required for xml tree saving (if xml2 activated)

xmlfile xmlfile for tree saving (if xml2 activated)

6.6.3.3 `double PhasedPMarkov::mean_post_log_likelihood (unsigned long ** count_eval, bool decal_required_e) [inline]`

return the mean post likelihood over the parameters and the trees

Parameters:

count_eval count of all the coded word(base size) of size order+1 for each phase, for evaluation step

decal_required_e necessary when using a count of word from 1-word to (`_max_depth`+1)-word

6.6.3.4 `template<class TSeq> double PhasedPMarkov::mean_post_log_likelihood (const TSeq & tseq_eval, short initial_phase_eval = 0) [inline]`

Parameters:

tseq_eval sequence(s) (set) for the evaluation step

initial_phase_eval phase of the first element of each sequence

6.6.3.5 `double PhasedPMarkov::mean_post_log_likelihood (unsigned long ** count_train, bool decal_required_t, unsigned long ** count_eval, bool decal_required_e) [inline]`

return the mean post likelihood over the parameters and the trees

Parameters:

count_train count of all the coded word(base size) of size order+1 for each phase, for training step

decal_required_t necessary when using a count of word from 1-word to (*_max_depth*+1)-word

count_eval count of all the coded word(base size) of size order+1 for each phase, for evaluation step

decal_required_e necessary when using a count of word from 1-word to (*_max_depth*+1)-word

6.6.3.6 `template<class TSeq1, class TSeq2> double PhasedPMarkov::mean_post_log_likelihood (const TSeq1 & tseq_train, const TSeq2 & tseq_eval, short initial_phase_train = 0, short initial_phase_eval = 0) [inline]`

Parameters:

tseq_train sequence(s) (set) for the training step

initial_phase_train phase of the first element of each sequence

tseq_eval sequence(s) (set) for the evaluation step

initial_phase_eval phase of the first element of each sequence

6.6.3.7 `double PhasedPMarkov::post_log_likelihood (unsigned long ** count_eval, bool decal_required_e) [inline]`

compute the mean posterior likelihood over the parameters

Parameters:

count_eval count of all the coded word(base size) of size order+1 for each phase, for evaluation step

decal_required_e necessary when using a count of word from 1-word to (*_max_depth*+1)-word

6.6.3.8 `template<class TSeq> double PhasedPMarkov::post_log_likelihood (const TSeq & tseq_eval, short initial_phase_eval = 0) [inline]`

compute the mean posterior likelihood over the parameters

Parameters:

tseq_eval sequence(s) (set) for the evaluation step

initial_phase_eval phase of the first element of each sequence

6.6.3.9 `void PhasedPMarkov::select (unsigned long ** count, bool decal_required, const Translator & trans = Translator(), const string & xmlfile = string()) [inline]`

performs the A Posteriori Maximisation

Parameters:

count count of all the coded word(base size) of size order+1 for each phase, for estimation

decal_required necessary when using a count of word from 1-word to (*_max_depth*+1)-word

trans a **Translator**(p. 70) is required for xml tree saving (if xml2 activated)

xmlfile xmlfile for tree saving (if xml2 activated)

The documentation for this class was generated from the following file:

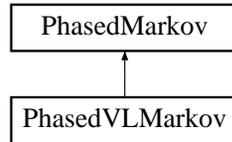
- **PhasedPMarkov.h**

6.7 PhasedVLMarkov Class Reference

Phased Variable Length **Markov**(p. 11) modelling, estimation and simulation.

```
#include <seqpp/PhasedVLMarkov.h>
```

Inheritance diagram for PhasedVLMarkov::



Public Member Functions

- `template<class TSeq> PhasedVLMarkov (const TSeq &tseq, vector< double > &init, double cutoff, short phase, short initial_phase=0)`
Constructor 1 from a SequenceSet(p.68) or a Sequence(p.65).
- `template<class TSeq> PhasedVLMarkov (const TSeq &tseq, double cutoff, short phase, short initial_phase=0)`
Constructor 2 from a SequenceSet(p.68) or a Sequence(p.65) with random init.
- `~PhasedVLMarkov ()`
destructor

Protected Attributes

- `vlm_tree * _vlmtree`
Variable Length Context Trees.

6.7.1 Detailed Description

Phased Variable Length **Markov**(p. 11) modelling, estimation and simulation.

PhasedVLMarkov is a **PhasedMarkov**(p. 23) object with a different estimation step. This object performs the estimation with the Variable Length **Markov**(p. 11) algorithm and then transforms, once per phase, the variable length context tree in a markovian matrix. One difference is that a cutoff is required for the selection of the tree.

6.7.2 Constructor & Destructor Documentation

- ##### 6.7.2.1 `template<class TSeq> PhasedVLMarkov::PhasedVLMarkov (const TSeq & tseq, vector< double > & init, double cutoff, short phase, short initial_phase = 0) [inline]`

Constructor 1 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65).

Parameters:

tseq a (set of) sequence(s) for estimation

init initial law in the leaves

cutoff tree selection cutoff

phase phase

initial_phase phase of the first element of each sequence

6.7.2.2 `template<class TSeq> PhasedVLMarkov::PhasedVLMarkov (const TSeq & tseq, double cutoff, short phase, short initial_phase = 0) [inline]`

Constructor 2 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65) with random init.

Parameters:

tseq a (set of) sequence(s) for estimation

cutoff tree selection cutoff

phase phase

initial_phase phase of the first element of each sequence

The documentation for this class was generated from the following file:

- **PhasedVLMarkov.h**

6.8 PMarkov Class Reference

Parcimonious **Markov**(p.11) modelling, estimation and simulation.

```
#include <seqpp/PMarkov.h>
```

Inheritance diagram for PMarkov::



Public Member Functions

- **PMarkov** (**Partition** &p, const **SequenceSet** &seqset, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const string &xmlfile=string())
Constructor 1 from a SequenceSet(p.68).
- **PMarkov** (**Partition** &p, const **Sequence** &seq, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const **Translator** &trans=**Translator**(), const string &xmlfile=string())
Constructor 2 from a Sequence(p.65).
- **PMarkov** (**Partition** &p, unsigned long *count, short size, short order, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const **Translator** &trans=**Translator**(), const string &xmlfile=string())
Constructor 3 from a coded-word count.
- **PMarkov** (**Partition** &part, const string &count_file, short size, short order, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., const **Translator** &trans=**Translator**(), const string &xmlfile=string())
Constructor 4 from a file containing coded-word count.
- **PMarkov** (**Partition** &part, short size, short order, const string &prior_alpha_file=string(), bool motif_prior=true, double penalty=0., bool alloc=true)
Basic Constructor 5. No estimation.
- void **select** (unsigned long *count, bool decal_required, const **Translator** &trans=**Translator**(), const string &xmlfile=string())
performs the A Posteriori Maximisation
- template<class TSeq1, class TSeq2> double **mean_post_log_likelihood** (const TSeq1 &tseq_train, const TSeq2 &tseq_eval, short initial_phase_train=0, short initial_phase_eval=0)
return the mean post likelihood over the parameters and the trees

- `template<class TSeq> double mean_post_log_likelihood (const TSeq &tseq_eval, short initial_phase_eval=0)`
return the mean post likelihood over the parameters and the trees
- `double mean_post_log_likelihood (unsigned long *count_train, bool decal_required_t, unsigned long *count_eval, bool decal_required_e)`
return the mean post likelihood over the parameters and the trees
- `double mean_post_log_likelihood (unsigned long *count_eval, bool decal_required_e)`
return the mean post likelihood over the parameters and the trees
- `double mean_post_log_likelihood ()`
return the mean post likelihood over the parameters and the trees
- `template<class TSeq> double post_log_likelihood (const TSeq &tseq_eval)`
compute the mean posterior likelihood over the parameters
- `double post_log_likelihood (unsigned long *count_eval, bool decal_required_e)`
compute the mean posterior likelihood over the parameters
- `void draw (unsigned long *count, bool decal_required, gsl_rng *r, const Translator &trans=Translator(), const string &xmlfile=string())`
draws a model
- `void draw (gsl_rng *r, const Translator &trans, const string &xmlfile=string())`
draws a model
- `void info_nb_leaves () const`
returns info on the number of leaves

6.8.1 Detailed Description

Parcimonious **Markov**(p. 11) modelling, estimation and simulation.

This is a special case of a phased Parcimonious **Markov**(p. 11) [**PhasedPMarkov**(p. 40)] model when only one phase is considered, and also a **Markov**(p. 11) object.

6.8.2 Constructor & Destructor Documentation

- 6.8.2.1 **PMarkov::PMarkov (Partition & p, const SequenceSet & seqset, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const string & xmlfile = string()) [inline]**

Constructor 1 from a **SequenceSet**(p. 68).

Parameters:

- p* associated partition
- seqset* a set of sequences for estimation

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

motif_prior activate a weight-function on the priors, weight proportionnal on each motif in the tree

penalty penalty on the leaves number, by default 0

xmlfile xmlfile for tree saving (if xml2 activated)

6.8.2.2 PMarkov::PMarkov (Partition & p, const Sequence & seq, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const Translator & trans = Translator(), const string & xmlfile = string()) [inline]

Constructor 2 from a **Sequence**(p. 65).

Parameters:

p associated partition

seq sequence for estimation

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

motif_prior activate a weight-function on the priors, weight proportionnal on each motif in the tree

penalty penalty on the leaves number, by default 0

trans a **Translator**(p. 70) is required only for the xml saving

xmlfile xmlfile for tree saving (if xml2 activated)

6.8.2.3 PMarkov::PMarkov (Partition & p, unsigned long * count, short size, short order, const string & prior_alpha_file = string(), bool motif_prior = true, double penalty = 0., const Translator & trans = Translator(), const string & xmlfile = string()) [inline]

Constructor 3 from a coded-word count.

Parameters:

p associated partition

count count of all the coded word(base size) of size order+1 for each phase, for estimation

size alphabet size

order markovian order associated to the word count

prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element

motif_prior activate a weight-function on the priors, weight proportionnal on each motif in the tree

penalty penalty on the leaves number, by default 0

trans a **Translator**(p. 70) is required only for the xml saving

xmlfile xmlfile for tree saving (if xml2 activated)

6.8.2.4 PMarkov::PMarkov (Partition & *part*, const string & *count_file*, short *size*, short *order*, const string & *prior_alpha_file* = string(), bool *motif_prior* = true, double *penalty* = 0., const Translator & *trans* = Translator(), const string & *xmlfile* = string()) [inline]

Constructor 4 from a file containing coded-word count.

Parameters:

part associated partition
count_file file with count
size alphabet size
order markovian order associated to the word count
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element
motif_prior activate a weight-function on the priors, weight proportionnal on each motif in the tree
penalty penalty on the leaves number, by default 0
trans a Translator(p. 70) is required only for the xml saving
xmlfile xmlfile for tree saving (if xml2 activated)

6.8.2.5 PMarkov::PMarkov (Partition & *part*, short *size*, short *order*, const string & *prior_alpha_file* = string(), bool *motif_prior* = true, double *penalty* = 0., bool *alloc* = true) [inline]

Basic Constructor 5. No estimation.

Parameters:

part associated partition
size alphabet size
order markovian order associated to the word count
prior_alpha_file file containing the alpha for the a priori law, one value per alphabet element
motif_prior activate a weight-function on the priors, weight proportionnal on each motif in the tree
penalty penalty on the leaves number, by default 0
alloc true for matrices memory allocation

6.8.3 Member Function Documentation

6.8.3.1 void PMarkov::draw (gsl_rng * *r*, const Translator & *trans*, const string & *xmlfile* = string()) [inline]

draws a model

Parameters:

r random generator
trans a Translator(p. 70) is required for xml tree saving (if xml2 activated)
xmlfile xmlfile for tree saving (if xml2 activated)

6.8.3.2 `void PMarkov::draw (unsigned long * count, bool decal_required, gsl_rng * r, const Translator & trans = Translator(), const string & xmlfile = string()) [inline]`

draws a model

Parameters:

count count of all the coded word(base size) of size order+1, for estimation
decal_required necessary when using a count of word from 1-word to (*_max_depth*+1)-word
r random generator
trans a Translator(p. 70) is required for xml tree saving (if xml2 activated)
xmlfile xmlfile for tree saving (if xml2 activated)

6.8.3.3 `double PMarkov::mean_post_log_likelihood (unsigned long * count_eval, bool decal_required_e) [inline]`

return the mean post likelihood over the parameters and the trees

Parameters:

count_eval count of all the coded word(base size) of size order+1 for evaluation step
decal_required_e necessary when using a count of word from 1-word to (*_max_depth*+1)-word

6.8.3.4 `double PMarkov::mean_post_log_likelihood (unsigned long * count_train, bool decal_required_t, unsigned long * count_eval, bool decal_required_e) [inline]`

return the mean post likelihood over the parameters and the trees

Parameters:

count_train count of all the coded word(base size) of size order+1, for learning step
decal_required_t necessary when using a count of word from 1-word to (*_max_depth*+1)-word
count_eval count of all the coded word(base size) of size order+1 for each phase, for evaluation step
decal_required_e necessary when using a count of word from 1-word to (*_max_depth*+1)-word

6.8.3.5 `template<class TSeq> double PMarkov::mean_post_log_likelihood (const TSeq & tseq_eval, short initial_phase_eval = 0) [inline]`

return the mean post likelihood over the parameters and the trees

Parameters:

tseq_eval sequence(s) (set) for the evaluation step
initial_phase_eval phase of the first element of each sequence

6.8.3.6 `template<class TSeq1, class TSeq2> double PMarkov::mean_post_log_likelihood (const TSeq1 & tseq_train, const TSeq2 & tseq_eval, short initial_phase_train = 0, short initial_phase_eval = 0) [inline]`

return the mean post likelihood over the parameters and the trees

Parameters:

tseq_train sequence(s) (set) for the training step
initial_phase_train phase of the first element of each sequence
tseq_eval sequence(s) (set) for the evaluation step
initial_phase_eval phase of the first element of each sequence

6.8.3.7 `double PMarkov::post_log_likelihood (unsigned long * count_eval, bool decal_required_e) [inline]`

compute the mean posterior likelihood over the parameters

Parameters:

count_eval count of all the coded word(base size) of size order+1 for each phase, for evaluation step
decal_required_e necessary when using a count of word from 1-word to (*_max_depth*+1)-word

6.8.3.8 `template<class TSeq> double PMarkov::post_log_likelihood (const TSeq & tseq_eval) [inline]`

compute the mean posterior likelihood over the parameters

Parameters:

tseq_eval sequence(s) (set) for the evaluation step

6.8.3.9 `void PMarkov::select (unsigned long * count, bool decal_required, const Translator & trans = Translator(), const string & xmlfile = string()) [inline]`

performs the A Posteriori Maximisation

Parameters:

count count of all the coded word(base size) of size order+1 for each phase, for estimation
decal_required necessary when using a count of word from 1-word to (*_max_depth*+1)-word
trans a **Translator**(p. 70) is required for xml tree saving (if xml2 activated)
xmlfile xmlfile for tree saving (if xml2 activated)

The documentation for this class was generated from the following file:

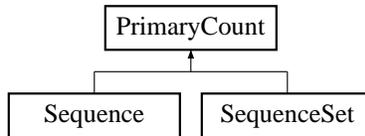
- **PMarkov.h**

6.9 PrimaryCount Class Reference

Abstract class designed for the (coded)word count.

```
#include <seqpp/PrimaryCount.h>
```

Inheritance diagram for PrimaryCount::



Public Member Functions

- virtual `~PrimaryCount ()`
destructor
- void `clear_count () const`
clear the count
- void `null_count () const`
put all counts down to 0
- short `tell_order () const`
returns the markovian order
- int `tell_nb_value () const`
returns the nber of codes/words taking into interest
- long `tell_jump () const`
returns the jump to the codes of _order+1 letters
- long * `get_jump () const`
returns the vector of the jumps to the codes of i letters, 0<i<=order+2
- Coder & `get_coder () const`
allows a const access to the coder
- bool `is_count_ok () const`
checks if the count is available
- unsigned long ** `get_p_count () const`
get the total count, in the phased case
- unsigned long * `get_count (short p=0) const`
get the total count in a phase p, or in the unphased case
- short `tell_phase () const`

returns nb of phase

- void **count_p_occurencies** (short phase, short initial_phase, unsigned long beg, unsigned long end) const
Phased count the number of occurencies of all the nvalue codes (in the sub-sequence [beg...end])... beware: beg>=0 but not >0.
- void **count_p_occurencies** (unsigned long **extern_count, short phase, short initial_phase, unsigned long beg, unsigned long end) const
Phased extern count the number of occurencies of all the nvalue codes (in the sub-sequence [beg...end])... beware: beg>=0 but not >0.
- void **count_p_occurencies** (unsigned long **extern_count, short phase, short initial_phase=0) const
Phased extern count the number of occurencies of all the nvalue codes in the whole sequence.
- void **count_p_occurencies** (short phase, short initial_phase=0) const
Phased count the number of occurencies of all the nvalue codes in the whole sequence.
- void **count_occurencies** (unsigned long beg, unsigned long end) const
Extern count the number of occurencies of all the nvalue codes (in the sub-sequence [beg...end])... beware: beg>=0 but not >0.
- void **count_occurencies** () const
Count the number of occurencies of all the nvalue codes in the whole sequence.
- unsigned long **tell_p_occurencies** (long code, short numphase) const
extract the number of occurencies of code in phase
- unsigned long **tell_occurencies** (long code) const
extract the number of occurencies of obs

Protected Member Functions

- void **init_count** () const
initialize the count
- virtual unsigned long **tell_max_count** () const =0
return the max length for the count
- virtual void **compute_count** (unsigned long **extern_count, short phase, short initial_phase, unsigned long beg, unsigned long end) const =0
compute the count

Protected Attributes

- short **_order**
Order of the model.

- long **_nbvalue**
nber of codes/words taking into interest
- short **_nbinv**
number of invalid negative code
- Coder * **_coder**
markovian coder associated to the order of the sequence
- bool **_coder_in**
true if _coder is build in the class
- unsigned long ** **_count**
count of each code/word, even invalid (-1)
- bool **_count_ok**
if the count is available
- short **_phase**
phase for the count

6.9.1 Detailed Description

Abstract class designed for the (coded)word count.

This object is based on integer codes to store words as required for a markov modelling (see **Sequence**(p.65)). The words are coded in `alphabet_size` base, coding the 1-token words, then the 2-tokens words, the 3-tokens words... Let call `jump(i)` the first code of a `i`-tokens words, a `PrimaryCount` object is designed to count all words:

```
(example with alphabet AGCT and order=2)
0          first word ( 0 -> A)
..
..          1-token words (from A to T)
..
jump(1)    (4 -> AA)
..
..          2-tokens words (from AA to TT)
..
jump(2)    (20 -> AAA)
..
..          3-tokens words (from AAA to TTT)
..
..
..          last word (TTT)
jump(order+1) (64 -> /end)
```

The count can also be phased.

6.9.2 Member Function Documentation

6.9.2.1 void PrimaryCount::count_occurencies (unsigned long *beg*, unsigned long *end*) const [inline]

Extern count the number of occurencies of all the nbvalue codes (in the sub-sequence [beg...end])... beware: beg>=0 but not >0.

Parameters:

beg the sub-sequence [beg...

end the sub-sequence ...end]

6.9.2.2 void PrimaryCount::count_p_occurencies (short *phase*, short *initial_phase* = 0) const [inline]

Phased count the number of occurencies of all the nbvalue codes in the whole sequence.

Parameters:

phase phase

initial_phase phase of the first element of associated sequence

6.9.2.3 void PrimaryCount::count_p_occurencies (unsigned long ** *extern_count*, short *phase*, short *initial_phase* = 0) const [inline]

Phased extern count the number of occurencies of all the nbvalue codes in the whole sequence.

Parameters:

extern_count extern count storage

phase phase

initial_phase phase of the first element of associated sequence

6.9.2.4 void PrimaryCount::count_p_occurencies (unsigned long ** *extern_count*, short *phase*, short *initial_phase*, unsigned long *beg*, unsigned long *end*) const [inline]

Phased extern count the number of occurencies of all the nbvalue codes (in the sub-sequence [beg...end])... beware: beg>=0 but not >0.

Parameters:

extern_count extern count storage

phase phase

initial_phase phase of the first element of associated sequence

beg the sub-sequence [beg...

end the sub-sequence ...end]

6.9.2.5 `void PrimaryCount::count_p_occurencies (short phase, short initial_phase, unsigned long beg, unsigned long end) const [inline]`

Phased count the number of occurencies of all the nbvalue codes (in the sub-sequence [*beg*...*end*])... beware: *beg* ≥ 0 but not >0.

Parameters:

phase phase

initial_phase phase of the first element of associated sequence

beg the sub-sequence [*beg*...

end the sub-sequence ...*end*]

6.9.2.6 `unsigned long* PrimaryCount::get_count (short p = 0) const [inline]`

get the total count in a phase *p*, or in the unphased case

Parameters:

p selected phase, default 0 if unphased case.

The documentation for this class was generated from the following files:

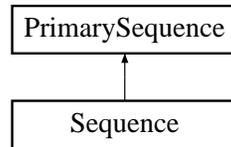
- **PrimaryCount.h**
- PrimaryCount.cc

6.10 PrimarySequence Class Reference

Virtual class corresponding to a generic sequence.

```
#include <seqpp/PrimarySequence.h>
```

Inheritance diagram for PrimarySequence::



Public Member Functions

- virtual `~PrimarySequence ()=0`
Destructor.
- const `Translator & ref_translator () const`
allows a const access to the alphabet
- short `tell_alphabet_size () const`
returns the alphabet size
- short `tell_nb_inv () const`
returns the nber of invalid codes in alphabet
- virtual unsigned long `tell_length () const`
returns the length
- string `tell_seq_name () const`
return the sequence name
- string `tell_file_name () const`
return the file name
- void `set_pos (unsigned long t)`
sets the current position
- unsigned long `tell_pos ()`
returns the current position
- virtual short `tell_int (unsigned long pos) const =0`
returns the int value of the single letter on position pos
- short `tell_int ()`
Return the code on current position.
- short `tell_int_compl (unsigned long pos) const`

Returns the complementary code on position i.

- short **tell_int_compl** () const
Returns the complementary code on position i.
- string **tell_pattern** (unsigned long pos) const
returns the pattern for a position
- string **tell_pattern_compl** (unsigned long pos)
returns complementary pattern on position pos
- string **tell_patterns** (unsigned long beg, unsigned long stop) const
returns the patterns between position beg and stop
- string **tell_label** (unsigned long pos) const
return the label(if defined in the alphabet) for a position
- string **tell_labels** (unsigned long beg, unsigned long stop) const
returns the label(if defined in the alphabet)-sequence between position beg and stop
- const string **strseq** () const
returns the sequence as a string
- void **print** (const string &file, ios_base::openmode mode=ios::out) const
print the sequence to file in fasta format

Protected Attributes

- const **Translator** * **_transl**
a Translator(p. 70) alphabet pointer
- short **_sizecode**
the number of char for a letter of the alphabet (alphabet mode)
- short **_nmodal**
number of possible observable value
- long * **_obs**
Vector of the observed datas.
- unsigned long **_length**
length of the sequence
- unsigned long **_ipos**
current position
- string **_name**
name of the sequence

- string **_name_file**
name of the file

6.10.1 Detailed Description

Virtual class corresponding to a generic sequence.

Note that a Primarysequence is associated with a **Translator**(p. 70) object.

The documentation for this class was generated from the following files:

- **PrimarySequence.h**
- PrimarySequence.cc

6.11 PrimarySequenceSet< TSequence > Class Template Reference

Class corresponding to a generic sequence set.

```
#include <seqpp/PrimarySequenceSet.h>
```

Public Member Functions

- **PrimarySequenceSet** ()
default constructor
- virtual **~PrimarySequenceSet** ()
destructor
- const **Translator** & **get_translator** () const
allows a const access to the alphabet via a reference
- short **tell_alphabet_size** () const
returns the alphabet size
- short **tell_nb_inv** () const
returns the nber of invalid codes
- TSequence & **get_sequence** (int i) const
allows a const access to ith sequence via a reference
- TSequence & **operator**() (int i) const
allows a const access to ith sequence via a reference
- int **tell_nb_sequence** () const
returns the number of sequences
- virtual unsigned long **tell_length** () const
returns the total length
- unsigned long **tell_length_max** () const
returns the length max
- unsigned long **tell_length_seq** (int i) const
returns the length of sequence i
- string **tell_seq_name** (int i) const
gives the name oh the ith file
- string **tell_file_name** (int i) const
gives the name oh the ith file
- void **weight_matrix** (double **wmat) const
compute the weight matrix associated to the sequences wmat[token]/[position in the sequence]

Protected Attributes

- `const Translator * _transl`
Associated alphabet.
- `short _nmodal`
number of possible observable value
- `int _nb_seq`
Number of sequences in the array.
- `unsigned long _total_length`
Total length.
- `unsigned long _length_max`
Maximum length of a sequence.
- `TSequence ** _array_seq`
Array of sequences.

6.11.1 Detailed Description

`template<class TSequence> class PrimarySequenceSet< TSequence >`

Class corresponding to a generic sequence set.

The documentation for this class was generated from the following file:

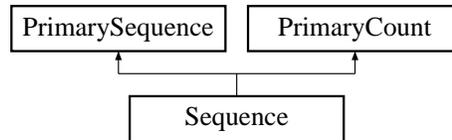
- **PrimarySequenceSet.h**

6.12 Sequence Class Reference

The Sequence class has been designed for Markovian analysis of a single sequence.

```
#include <seqpp/Sequence.h>
```

Inheritance diagram for Sequence::



Public Member Functions

- **Sequence** (ifstream &ifile, short order, const **Translator** &transl, Coder &coder, const string &name_file="noname")
Constructor from a flow ifile from a fasta file.
- **Sequence** (short order, unsigned long length, const **Translator** &transl)
minimal constructor
- **Sequence** (const string &name_file, short order, const **Translator** &transl)
Constructor from a Fasta or Genbank file containing a single sequence.
- **Sequence** (const **Sequence** &seq)
Copy constructor.
- **Sequence** (const **Sequence** &seq, short order)
Deduction constructor from an order 0-sequence.
- virtual ~**Sequence** ()
destructor
- short **tell_int** (unsigned long pos) const
Returns the int value of the token on position pos.
- long **tell_code** (unsigned long pos) const
Returns the code on position pos.
- long & **operator**() (unsigned long pos) const
idem "tell_code" Returns the code on position pos
- long **tell_code** () const
Returns the code on current position.
- const long * **seq** () const
Returns the coded sequence as a vector.

- const long * **get_code** () const
Returns the coded sequence as a vector.
- void **simule** (const double *m_trans, const double *v_init, const gsl_rng *r)
simulate the observations from a transition matrix and a vector for initial_order-word proba
- void **simule** (const double *m_trans, const gsl_rng *r)
simulate the observations from a transition matrix, random initial law
- void **simule_phased** (short phase, const double **m_trans, const double *v_init, const gsl_rng *r)
simulate the observations from a vector of transition matrices and a vector for initial_order word proba
- void **simule** (unsigned long position, double *proba, const gsl_rng *r)
simulate the observation on position from a alphabet size vector

Protected Member Functions

- virtual unsigned long **tell_max_count** () const
return the max length for the count
- virtual void **compute_count** (unsigned long **extern_count, short phase, short initial_phase, unsigned long beg, unsigned long end) const

6.12.1 Detailed Description

The Sequence class has been designed for Markovian analysis of a single sequence.

On each position, the markovian predictor is coded and store (see **PrimaryCount**(p. 55)):

```
Exple with the alphabet AGCT:
sequence position  .... 7  8  9  ...
token             .... A  G  A  ...
code stored       .....24
```

Sequence object of integer values corresponding to the code of the word considered on a position t for a **Markov**(p. 11) model of order_order... this code corresponds to the number of smaller words and the calculus in base |A| of this word, with A the alphabet.

ex : from A to GGG for order 2 with dna alphabet

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Sequence::Sequence (ifstream & ifile, short order, const Translator & transl, Coder & coder, const string & name_file = "noname")

Constructor from a flow ifile from a fasta file.

on the position of the char '>', create an object until the next '>' generating his name by the concatenation of the sentence after the char '>'. The flow is or eof or on a '>' at the end.

Parameters:

ifile ifstream flow
order markovian order
transl reference to a **Translator**(p. 70)
coder reference to a **Coder**
name_file file name corresponding to ifile

6.12.2.2 Sequence::Sequence (short *order*, unsigned long *length*, const **Translator & *transl*)**

minimal constructor

Parameters:

order markovian order
length default length
transl reference to a **Translator**(p. 70)

6.12.2.3 Sequence::Sequence (const string & *name_file*, short *order*, const **Translator & *transl*)**

Constructor from a Fasta or Genbank file containing a single sequence.

Parameters:

name_file file name to open
order markovian order
transl reference to a **Translator**(p. 70)

6.12.3 Member Function Documentation**6.12.3.1 void Sequence::compute_count (unsigned long ** *extern_count*, short *phase*, short *initial_phase*, unsigned long *beg*, unsigned long *end*) const**
[protected, virtual]

total length case

Implements **PrimaryCount** (p. 56).

The documentation for this class was generated from the following files:

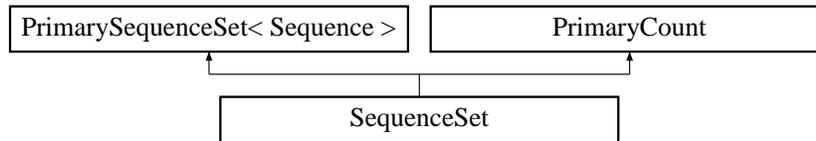
- **Sequence.h**
- **Sequence.cc**

6.13 SequenceSet Class Reference

Set of **Sequence**(p. 65) objects.

```
#include <seqpp/SequenceSet.h>
```

Inheritance diagram for SequenceSet::



Public Member Functions

- **SequenceSet** ()
Default constructor.
- **SequenceSet** (const string &seq_list_file, const string &alpha_file, short order, int nbseq=BUFFERSEQ)
constructor with a alphabet-needed file
- **SequenceSet** (const string &seq_list_file, short mode, const string &line2parse, bool case_sensitivity, short order, int nbseq=BUFFERSEQ)
constructor with a alphabet-needed line2parse

6.13.1 Detailed Description

Set of **Sequence**(p. 65) objects.

from :

- a Fasta file (with 1 or more sequences inside separated by a line beginning with a SKIPCHAR) or a Genbank file
- or a file containing the name of many Fasta (or a Genbank) sequence files

6.13.2 Constructor & Destructor Documentation

6.13.2.1 SequenceSet::SequenceSet (const string & seq_list_file, const string & alpha_file, short order, int nbseq = BUFFERSEQ)

constructor with a alphabet-needed file

Parameters:

- seq_list_file* file, among the 3 possibilities of files.
- alpha_file* file for initializing the alphabet (see **Translator**(p. 70) doc)
- order* Markovian order,
- nbseq* number of sequences

6.13.2.2 SequenceSet::SequenceSet (const string & *seq_list_file*, short *mode*, const string & *line2parse*, bool *case_sensitivity*, short *order*, int *nbseq* = BUFFERSEQ)

constructor with a alphabet-needed line2parse

Parameters:

seq_list_file file, among the 3 possibilities of files.

mode number of char for a token in the alphabet

line2parse string for initializing the alphabet (see **Translator**(p.70) doc)

case_sensitivity true for being case sensitive

order Markovian order,

nbseq number of sequences

The documentation for this class was generated from the following files:

- **SequenceSet.h**
- SequenceSet.cc

6.14 Translator Class Reference

Alphabet designer tool.

```
#include <seqpp/Translator.h>
```

Public Member Functions

- **Translator** (const string &alpha_file)
Constructor from a configuration file.
- **Translator** (const string &line2parse, short mode, bool case_sensitivity=true)
Constructor from an expression.
- **Translator** (const **Translator** &t)
Copy constructor.
- **Translator** ()
Default constructor.
- **~Translator** ()
Destructor.
- short **tell_alphabet_size** () const
Returns alphabet size.
- short **tell_nb_inv** () const
Return the number of invalid code.
- short **tell_alphabet_mode** () const
Return alphabet mode, the number of characters of a token.
- bool **is_complementary_ok** () const
Checks if complementary codes are defined.
- bool **is_label_ok** () const
Checks if labels are defined.
- short int **char_to_int** (const char *source) const
*Returns the int code for a given char *.*
- short int **str_to_int** (const string &source) const
Returns the int code for a given string.
- short **complementary_char_to_int** (const char *source) const
*Returns the complementary int code for a given char *.*
- short int **complementary_str_to_int** (const string &source) const
Returns the complementary int code for a given string.

- short **complementary_int** (short source) const
Returns the complementary int code for a given code.
- void **int_to_char** (char *dest, short source) const
Initialize dest to the word corresponding to the int source.
- string **int_to_str** (short source) const
returns the word corresponding to the int source
- void **complementary_char** (char *dest, const char *source) const
Initialize dest to the word corresponding to the complementary of word source.
- string **complementary_str** (const string &source) const
returns the token corresponding to the inverse complementary of a token source
- string **complementary_int_to_str** (short source) const
returns the string corresponding to the complementary of int source
- string **complementary_strw** (const string &wordsource) const
returns the inverse complementary string-word of source
- void **vect_to_word** (char *worddest, const short *vectsource, short sizeword) const
a char-word <- an int-vect
- void **vect_to_word** (char *worddest, const vector< short > &vectsource) const
a char-word <- an int-vect
- string **vect_to_strw** (const short *vectsource, short sizeword) const
a string-word <- an int-vect
- string **vect_to_strw** (const vector< short > &vectsource) const
a string-word <- an int-vect
- void **word_to_vect** (short *vectdest, const char *wordsource, short sizeword) const
a char-word -> an int-vect
- void **word_to_vect** (vector< short > &vectdest, const char *wordsource) const
a char-word -> an int-vect
- void **strw_to_vect** (vector< short > &vectdest, const string &wordsource) const
a char-word -> an int-vect
- long **word_to_coded_int** (const char *wordsource, int &l) const
a char-word -> an int code (base alphabet_size)
- long **strw_to_coded_int** (const string &wordsource) const
a string-word -> an int code (base alphabet_size)
- long **complementary_coded_int** (long codesource, short sizeword) const
Returns the complementary int code for a given coded int.

- short **isInvalid** (const char *source) const
checks if source is invalid
- short **isInvalid** (const string &source) const
checks if source is invalid
- string **int_to_label** (short source) const
returns the label corresponding to the source
- string **vect_to_labelstrw** (const vector< short > &vectsource) const
a string-word of label <- an int-vect

Protected Attributes

- short **_alphabet_size**
number of tokens describing the alphabet
- short **_alphabet_mode**
number of characters of each token
- short **_nb_inv**
number of invalid tokens
- bool **_complementary_ok**
checks if complementary is defined
- bool **_label_ok**
checks if labels are defined

6.14.1 Detailed Description

Alphabet designer tool.

The key of the seq++ flexibility is the translator object. It defines a mapping between the alphabet and integer values. In seq++, a letter of an alphabet, called a token, can be made of one or more characters, for instance to manage n-uplets like codons. It is associated to its rank integer position in the alphabet.

A simple configuration file is required, containing synonymous tokens, complementary tokens (if exist) and labels. A label aims to be the explicit name of a token. Moreover, invalid tokens can be declared and associated to negative values.

Use a initial file containing same size(alphabet_mode characters) tokens (the letters of the alphabet)

- on a same line, all the synonymous tokens (beware lower and upper case)
- (if necessary) on a line beginning with a "?", single characters or tokens (same size than others) to declare a class of invalid tokens. NB: a default class contains all the tokens not declared (neither belonging to alphabet nor to known invalids)

- (if necessary) their associated label (formally the common name) after a " → "
 - (if necessary) their complementary after a ":"
- Beware the spaces between each tokens !

exple :

```
#codons
ttt TTT -> F1
ttc TTC -> F2
tta TTA -> L1
...
ggg GGG -> G4
? taa TAA tag TAG tga TGA
```

```
#dna
a A : t T
g G : c C
c C : g G
t T : a A
? n N
```

```
#secondary structures
H -> HELIX
B -> BETA
C -> COIL
```

OR

Use an expression to parse ("line2parse") where all the same size ("mode") tokens are concatenated, with a ":" to define the inverse complementary with same size tokens. No synonyms supported.
exple :

```
acgt:tgca          # dna alphabet, mode = 1
R1R2T1T2T3X1Y1Y2 # other alphabet, with mode = 2 (R1 is the first token...)
```

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Translator::Translator (const string & *alpha_file*)

Constructor from a configuration file.

Parameters:

alpha_file configuration file

6.14.2.2 Translator::Translator (const string & *line2parse*, short *mode*, bool *case_sensitivity* = true)

Constructor from an expression.

Parameters:

line2parse alphabet expression

mode number of characters of a token

case_sensitivity false if no case sensitivity required

6.14.3 Member Function Documentation

6.14.3.1 `short int Translator::char_to_int (const char * source) const`

Returns the int code for a given char *.

Parameters:

source pointer to the first character of a token to translate in int Beware that source must be the beginning of a word of size `_alphabet_mode`, or -1 is returned

6.14.3.2 `void Translator::complementary_char (char * dest, const char * source) const [inline]`

Initialize dest to the word corresponding to the complementary of word source.

Parameters:

dest must be allocated with `_alphabet_mode+1` space, empty word if bad query.

source pointer to the first position of the source token

6.14.3.3 `short int Translator::complementary_char_to_int (const char * source) const`

Returns the complementary int code for a given char *.

Parameters:

source pointer to the first character of a token to complementary translate in int Beware that source must be the beginning of a word of size `_alphabet_mode`, or -1 is returned

6.14.3.4 `short Translator::complementary_int (short source) const [inline]`

Returns the complementary int code for a given code.

Parameters:

source integer code

6.14.3.5 `short int Translator::complementary_str_to_int (const string & source) const`

Returns the complementary int code for a given string.

Parameters:

source string-token to complementary translate in int

6.14.3.6 void Translator::int_to_char (char * *dest*, short *source*) const

Initialize *dest* to the word corresponding to the int *source*.

Parameters:

dest must be allocated with `_alphabet_mode+1` space, empty word if bad query.
source integer to translate

6.14.3.7 short int Translator::str_to_int (const string & *source*) const

Returns the int code for a given string.

Parameters:

source string-token to translate in int

6.14.3.8 long Translator::strw_to_coded_int (const string & *wordsource*) const

a string-word -> an int code (base `alphabet_size`)

Parameters:

wordsource word to code

6.14.3.9 void Translator::strw_to_vect (vector< short > & *vectdest*, const string & *wordsource*) const

a char-word -> an int-vect

a string-word -> an int-vect

6.14.3.10 void Translator::vect_to_word (char * *worddest*, const short * *vectsource*, short *sizeword*) const

a char-word <- an int-vect

Parameters:

worddest word (must be allocated `sizeword*_alphabet_mode + 1` space) corresponding to the sequential translation of *source*
vectsource vector of integer to translate
sizeword size of the *vectsource* to translate

6.14.3.11 long Translator::word_to_coded_int (const char * *wordsource*, int & *l*) const

a char-word -> an int code (base `alphabet_size`)

Parameters:

wordsource pointer to the character word to code
l its length

6.14.3.12 void Translator::word_to_vect (short * *vectdest*, const char * *wordsource*, short *sizeword*) const

a char-word -> an int-vect

Parameters:

vectdest integer vector (must be sizeword-allocated)

wordsource character word to sequentially translate

sizeword size of the word

The documentation for this class was generated from the following files:

- **Translator.h**
- Translator.cc

6.15 VLMarkov Class Reference

Variable **Markov**(p. 11) modelling, estimation and simulation.

```
#include <seqpp/VLMarkov.h>
```

Inheritance diagram for VLMarkov::



Public Member Functions

- `template<class TSeq> VLMarkov (const TSeq &tseq, vector< double > &init, double cutoff)`
Constructor 1 from a SequenceSet(p. 68) or a Sequence(p. 65).
- `template<class TSeq> VLMarkov (const TSeq &tseq, double cutoff)`
Constructor 2 from a SequenceSet(p. 68) or a Sequence(p. 65) with random init.

Protected Attributes

- `vlm_tree * _vlmtree`
Variable Length Context Tree.

6.15.1 Detailed Description

Variable **Markov**(p. 11) modelling, estimation and simulation.

This is a special case of a phased Variable LengthMarkov [**PhasedVLMarkov**(p. 47)] model when only one phase is considered, and also a **Markov**(p. 11) object.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 `template<class TSeq> VLMarkov::VLMarkov (const TSeq & tseq, vector< double > & init, double cutoff) [inline]`

Constructor 1 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65).

Parameters:

- tseq* a (set of) sequence(s) for estimation
- init* initial law in the leaves
- cutoff* tree selection cutoff

6.15.2.2 `template<class TSeq> VLMarkov::VLMarkov (const TSeq & tseq, double cutoff) [inline]`

Constructor 2 from a **SequenceSet**(p. 68) or a **Sequence**(p. 65) with random init.

Parameters:

tseq a (set of) sequence(s) for estimation

cutoff tree selection cutoff

The documentation for this class was generated from the following file:

- **VLMarkov.h**

Chapter 7

seqpp File Documentation

7.1 Coder.h File Reference

Coder.

```
#include <vector>
```

```
#include <iostream>
```

7.1.1 Detailed Description

Coder.

Author:

Vincent Miele

07/2004

Coded abstract representation of a word

7.2 const.h File Reference

Configuration file.

Namespaces

- namespace `seqplusplus_space`

Variables

- const int **BUFFERSEQ** = 100
for `SequenceSet`(p.68), number of sequence per buffer
- const int **LGBLOC** = 1000
for `Sequence`(p.65), length of a bloc used to memory the sequence
- const char **SKIPCHAR** = '>'
for `Sequence`(p.65), character defining the line to skip on top of a sequence (cf. fasta file)
- const char **INVCHAR** = '?'
for `Translator`(p.70), for invalid declaration file
- const char **COMCHAR** = '#'
comments characters
- const char **LABPATT** [3] = " → "
label prefix before a declaration
- const char **LABINV** [4] = "INV"
invalid label
- const int **LGSTRING** = 256
string name max length
- const int **LG_FILENAME** = 200
file name max length
- const int **STREAM_PRECISION** = 8
precision for the o/ofstream
- const double **PRECISION** = 1e-7
precision for comparing a double to 0, must be = 1e-(STREAM_PRECISION-1)
- const double **EPS** = 1e-5
for MTD E.M. Algorithm
- const short **DEBUG** = 0
debug level, for the compilation

7.2.1 Detailed Description

Configuration file.

Author:

Vincent Miele

12/2003

7.3 DOXYMAINPAGE.h File Reference

7.3.1 Detailed Description

Main page

7.4 Markov.h File Reference

Markov(p. 11) model.

```
#include <seqpp/PhasedMarkov.h>
```

```
#include <seqpp/arnoldi.h>
```

Classes

- class **Markov**

Markov modelling, estimation and simulation.

7.4.1 Detailed Description

Markov(p. 11) model.

Author:

David Robelin - Vincent Miele

01/2004

7.5 MTDMarkov.h File Reference

Mixture Transition Distribution **Markov**(p. 11) model.

```
#include <seqpp/Markov.h>
```

```
#include <seqpp/mtd_core.h>
```

Classes

- class **MTDMarkov**

*Mixture Transition Distribution **Markov**(p. 11) modelling, estimation and simulation.*

7.5.1 Detailed Description

Mixture Transition Distribution **Markov**(p. 11) model.

Author:

Vincent Miele

01/2004

7.6 Partition.h File Reference

Partition(p. 20).

```
#include <seqpp/Partition_box.h>
```

```
#include <seqpp/Translator.h>
```

```
#include <list>
```

```
#include <set>
```

```
#include <map>
```

Classes

- class **Partition**

Dealer of alphabet partitions.

7.6.1 Detailed Description

Partition(p. 20).

Author:

Vincent Miele

23/02/2004

7.7 Partition_box.h File Reference

Partition_box.

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <vector>
```

```
#include <cmath>
```

7.7.1 Detailed Description

Partition_box.

Author:

Vincent Miele

23/02/2004

display the set of $|A|$ -vectors containing the num of group for each letter. exple: a c g t 1 1 1 1 1
2 1 1 1 2 2 1 etc... 1 2 3 4 and those vector must be processed and stored in a Tpart

7.8 PhasedMarkov.h File Reference

Phased **Markov**(p. 11) model.

```
#include <seqpp/SequenceSet.h>
```

```
#include <seqpp/arnoldi.h>
```

```
#include <vector>
```

```
#include <string>
```

```
#include <gsl/gsl_sf_gamma.h>
```

Classes

- class **PhasedMarkov**

Phased Markov(p. 11) *modelling, estimation and simulation.*

7.8.1 Detailed Description

Phased **Markov**(p. 11) model.

Author:

David Robelin - Vincent Miele

01/2004

7.9 PhasedMTDMarkov.h File Reference

Phased Mixture Transition Distribution **Markov**(p. 11) model.

```
#include <seqpp/PhasedMarkov.h>
```

```
#include <seqpp/mtd_core.h>
```

Classes

- class **PhasedMTDMarkov**

*Phased Mixture Transition Distribution **Markov**(p. 11) modelling, estimation and simulation.*

7.9.1 Detailed Description

Phased Mixture Transition Distribution **Markov**(p. 11) model.

Author:

Vincent Miele

01/2004

7.10 PhasedPMarkov.h File Reference

Phased Parsimonious **Markov**(p. 11) model.

```
#include <seqpp/PhasedMarkov.h>
```

```
#include <seqpp/pmm_forest.h>
```

Classes

- class **PhasedPMarkov**

*Phased Parsimonious **Markov**(p. 11) modelling, estimation and simulation.*

7.10.1 Detailed Description

Phased Parsimonious **Markov**(p. 11) model.

Author:

Vincent Miele

01/2004

7.11 PhasedVLMarkov.h File Reference

Phased Variable Length **Markov**(p. 11) model.

```
#include <seqpp/PhasedMarkov.h>
```

```
#include <seqpp/vlm_tree.h>
```

Classes

- class **PhasedVLMarkov**

*Phased Variable Length **Markov**(p. 11) modelling, estimation and simulation.*

7.11.1 Detailed Description

Phased Variable Length **Markov**(p. 11) model.

Author:

Vincent Miele

01/2004

7.12 PMarkov.h File Reference

Parcimonious **Markov**(p. 11) model.

```
#include <seqpp/Markov.h>
```

```
#include <seqpp/PhasedPMarkov.h>
```

```
#include <seqpp/pmm_tree.h>
```

Classes

- class **PMarkov**

Parcimonious Markov(p. 11) *modelling, estimation and simulation.*

7.12.1 Detailed Description

Parcimonious **Markov**(p. 11) model.

Author:

Vincent Miele

01/2004

7.13 PrimaryCount.h File Reference

Abstract class designed for the (coded)word count.

```
#include <seqpp/Coder.h>
```

Classes

- class **PrimaryCount**

Abstract class designed for the (coded)word count.

7.13.1 Detailed Description

Abstract class designed for the (coded)word count.

Author:

Vincent Miele

23/04/2003

7.14 PrimarySequence.h File Reference

Abstract generic sequence.

```
#include <seqpp/Translator.h>
```

```
#include <fstream>
```

```
#include <iostream>
```

```
#include <string>
```

Classes

- class **PrimarySequence**

Virtual class corresponding to a generic sequence.

7.14.1 Detailed Description

Abstract generic sequence.

Author:

Vincent Miele - Florence Muri - Pierre Nicolas

23/04/2003

7.15 PrimarySequenceSet.h File Reference

Generic set of sequence objects.

```
#include <seqpp/Sequence.h>
```

```
#include <seqpp/Translator.h>
```

```
#include <iomanip>
```

Classes

- class **PrimarySequenceSet**< **TSequence** >
Class corresponding to a generic sequence set.

7.15.1 Detailed Description

Generic set of sequence objects.

Author:

Vincent Miele

23/04/2003

7.16 Sequence.h File Reference

Sequence(p. 65) class has been designed for Markovian analysis.

```
#include <seqpp/PrimarySequence.h>
#include <seqpp/PrimaryCount.h>
#include <seqpp/Coder.h>
#include <set>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
```

Classes

- class **Sequence**

The Sequence class has been designed for Markovian analysis of a single sequence.

7.16.1 Detailed Description

Sequence(p. 65) class has been designed for Markovian analysis.

Author:

Vincent Miele - Florence Muri - Pierre Nicolas

23/04/2003

7.17 SequenceSet.h File Reference

Set of **Sequence**(p. 65) objects.

```
#include <seqpp/PrimarySequenceSet.h>
```

```
#include <iomanip>
```

Classes

- class **SequenceSet**
*Set of **Sequence**(p. 65) objects.*

7.17.1 Detailed Description

Set of **Sequence**(p. 65) objects.

Author:

Vincent Miele

23/04/2003

7.18 Translator.h File Reference

General Alphabet class.

```
#include <seqpp/const.h>
```

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <string>
```

```
#include <map>
```

```
#include <vector>
```

Classes

- class **Translator**

Alphabet designer tool.

7.18.1 Detailed Description

General Alphabet class.

Author:

Vincent Miele - Pierre Yves Bourguignon

23/04/2003

7.19 VLMarkov.h File Reference

Variable Length **Markov**(p. 11) model 01/2004.

```
#include <seqpp/Markov.h>
#include <seqpp/PhasedVLMarkov.h>
#include <seqpp/vlm_tree.h>
```

Classes

- class **VLMarkov**
Variable Markov(p. 11) *modelling, estimation and simulation.*

7.19.1 Detailed Description

Variable Length **Markov**(p. 11) model 01/2004.

Author:

Vincent Miele

Chapter 8

seqpp Page Documentation

8.1 Library

The inheritance links between classes are essential to understand the design of seq++ and the behaviour of each object. Please browse the top task bar to discover the **Class Hierarchy**.

Basically seq++ focuses on both aspects:

- Sequence manipulation:
 1. the key of the seq++ flexibility is the possibility to customize any alphabet thanks to the **Translator**(p. 70) class. Therefore a seq++-based software can support alphabets of DNA, codons, proteins, secondary structure, hydrophobicity class and so on.
 2. the **SequenceSet**(p. 68) and **Sequence**(p. 65) classes are dedicated to the storage and the manipulation of external sequences provided by the user. This is done keeping in mind the purpose of modeling sequences with Markov models: at each position of the sequences the associated markovian-predictor (word, see **PrimaryCount**(p. 55)) is coded and stored.

- Markovian modelling (estimation, simulation...)
 1. the **PhasedMarkov**(p. 23) object represents the general Markov model: the order and the number of phases have to be selected. **Markov**(p. 11) focuses on non-phased models.
 2. **PhasedPMarkov**(p. 40) and **PMarkov**(p. 49) are dedicated to Parsimonious Markov models. Note that the definition of a set of partitions of the alphabet (see **Partition**(p. 20)) can be required.
 3. **PhasedMTDMarkov**(p. 38) and **MTDMarkov**(p. 18) are dedicated to Mixture Transition Distribution Markov models. Note that the estimation is performed by a E.M. algorithm so that E.M. parameters have to be set up.
 4. **PhasedVLMarkov**(p. 47) and **VLMarkov**(p. 77) are dedicated to Variable Length Markov models. Beware that these models require a cutoff value which can be estimated by optimizing the BIC criteria.

Please watch the **References**(p. 108) page for more details on the models.

8.1.1 The correct version of the published Fig.2.

```
ord = 2;
SequenceSet(p. 68) mot("motifs.fna", "dna", ord);
nphase = mot(0).tell_length();
PhasedMarkov(p. 23) m_motif(mot, nphase);
SequenceSet(p. 68) set("target.gb", "dna", ord);
Sequence(p. 65) & seq = set(0);
Markov(p. 11) m_backg(seq);
for(t=nphase-1;t<seq.tell_length();t++){
a = m_motif.proba(seq,t-nphase+1,t);
b = m_backg.proba(seq,t-nphase+1,t);
cout<<t<<" score: "<<a/b<<endl;
}
```

8.2 Programs

When installing the package seq++, both the library and a set of binary programs are available on your system:

1. `estim_m` is dedicated to Phased Markov model estimation (based on the **Phased-Markov**(p.23) object).
2. `estim_vlm` is dedicated to Variable Length Markov model estimation (based on the **PhasedVLMMarkov**(p.47) object).
3. `estim_pm` is dedicated to Parsimonious Markov model estimation (based on the **Phased-PMarkov**(p.40) object).
4. `estim_mtd` is dedicated to Mixture Transition Distribution Markov model estimation (based on the **PhasedMTDMarkov**(p.38) object).
5. `simul_m` is dedicated to sequence simulation from a markovian matrix (previously estimated) or a base sequence (based on the **PhasedMarkov**(p.23) object).
6. `dist_m` computes the Total Variation Distance between two Markov models previously estimated with `estim_*`.

Nota bene: the `estim_*` programs differs only on the way the estimation of the markovian matrix is performed.

Basicaly the best way to discover the possibilities of these programs is the Unix way:

```
program_name --help OR man program_name
```

The estimation results can be saved in such a representation:

```
# 1 <- Order of the phased Markov chain
# 2 <- Phase
# 4 <- Alphabet size
# Phase nř0
# Transition matrix:
0.3945322543    0.1652811616    0.1535033485    0.2866832356
0.2758032675    0.208204727    0.2605722039    0.2554198016
0.3496152294    0.2310568942    0.1708121974    0.248515679
0.2312234494    0.2503993203    0.1515293118    0.3668479184
# Stationnary Probability:
0.3127105148    0.2114684268    0.1783495332    0.2974715251
# Phase nř1
# Transition matrix:
0.3923961961    0.163516403    0.1521005152    0.2919868858
0.2789719005    0.2082462898    0.2570550517    0.2557267579
0.352109549    0.2297549901    0.1679259742    0.2502094867
0.2320996136    0.2447917561    0.1520437956    0.3710648347
# Stationnary Probability:
0.3135417652    0.2089660861    0.1771006767    0.300391472
```


8.3 What's new

	[07/05/05]		Version 4.1.5. New options <code>-draw</code> to draw a model. Also new <code>-prior</code> behaviour: a file containing one prior value per alphabet element is required.
	[07/05/05]		Version 4.1.3. New options <code>-k</code> , <code>-M</code> , <code>-m</code> , <code>-nbleaves</code> . Please try <code>-h</code> for help.
	[03/27/05]		Version 4.0.2. Methods "proba" and "proba_c" are modified, please watch the Phased-Markov (p. 23) documentation. Bug fixed in <code>simul_m</code> (phase inversion).
	[03/17/05]		Miele V, Bourguignon PY, Robelin D, Nuel G, Richard H. seq++: analyzing biological sequences with a range of Markov-related models. Bioinformatics. 2005 Mar 17
	[02/17/05]		Version 4.0.0: Mixture Transition Distribution models are available in the library, as the program <code>estim_mtd</code> and the Phased-MTDMarkov (p. 38) and MTD-Markov (p. 18) objects.
	[02/02/05]		New version 3.1.3, with man pages. No more limitations for SequenceSet (p. 68) in terms of sequences number.
Generated on Thu Aug 4 20:33:11 2005 for seqpp by Doxygen			
	[01/24/05]		MTD model development starts with 1.75 (1 + 1/2 + 1/4...) developers

8.4 Bug and Feedback

- Bug report:

If you experiment any bug with seq++, please send a mail to Vincent Miele name@genopole.cnrs.fr with the following subject "bug report". At last, feel free to add any relevant comments to that bug report. We will try to answer to most of those reports.

- Other feedback:

seq++ is an emerging project which is still in evolution. Thus, your feedback is more than welcome to help us to get seq++ better than it is. For any suggestion or comment, please send a mail to Vincent Miele name@genopole.cnrs.fr with the following subject "feedback".

8.5 ToDo

Forthcoming developments:

- implementation of Mixture Transition Model (MTD) in release seq++-4.0 [in progress, released in February].

Please send a mail to Sophie Lebre [name@genopole.cnrs.fr](mailto:lebre@genopole.cnrs.fr) for information on the topic.

- algorithm for Transcription factor binding site detection using seq++ modules [in test, released in February 2005].

8.6 Download and Installation

The package is written in ANSI C++ and developed on x86 GNU/Linux systems with GCC 3.3. It has been successfully tested with Intel ICC 8.0, on Sun systems using GCC 3.3 and Apple Mac OSX systems with GCC 3.1. Compilation and installation are compliant with the GNU standard procedure. seq++ is licensed under the GNU General Public License (**Licence**(p.107)). Please visit the bottom links to download seq++.

NOTICE: The following procedure installs on your system both the library as a shared library and associated include files and the binaries such as estimation and simulation programs.

0/ Requirements : compilers C++ and F77, library GSL and XML2(optionnal).

1/ The installation procedure is explained in ./INSTALL. The classical way, compliant with the GNU standard procedure:

- ./configure (add "--enable-xml" to activate xml options, libxml2 required)
- make
- make install
- make docs (if you want to generate the html doc)

NB: for a debug compilation, add "--enable-debug and then le library name becomes "seqpp_g"

2/ You can then include files from seqpp by such an include

- exple: #include <seqpp/.....>

2'/ You can then use the binary programs.

- exple: estim_m -help

3/ You now must compile with the -lseqpp or use in your configure.in the set of instructions :

- AC_CHECK_LIB(seqpp, main)

4/ Don't forget to fit your \$LD_LIBRARY_PATH for the linkage

8.7 Licence

Copyright (C) 2005 Laboratoire Statistique & Genome

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Please visit the GNU licences web site.

8.8 References

Please feel free to contact:

- Pierre-Yves Bourguignon `name@genopole.cnrs.fr` for details on Parsimonious models.
- David Robelin `name@genopole.cnrs.fr` for details on phased models and statistical calculus.
- Gregory Nuel `name@genopole.cnrs.fr` for details on Arnoldi algorithms.
- Sophie Lèbre `name@genopole.cnrs.fr` for details on Mixture Transition Distribution models.

Main reference:

- Miele V, Bourguignon PY, Robelin D, Nuel G, Richard H. (2005) `seq++`: analyzing biological sequences with a range of Markov-related models. *Bioinformatics*.

Several references:

- Bourguignon,P.Y., Robelin,D. (2004) `Modeles de Markov parcimonieux` , Actes de JO-BIM.
- Bourguignon,P.Y. (2005)`Parsimonious Markov Models`, submitted.
- Bühlmann,P., Wyner,A.J. (1999) Variable length Markov chains. *Annals of Statistics*, 27, 480-513.
- Bulyk M.L., (2003) Computational prediction of transcription-factor binding site locations, *Genome Biology*, 5:201.
- Borodovsky,M., McIninch,J.D., Koonin,E.V., Rudd,K.E., Medigue,C., Danchin,A., (1995) Detection of new genes in a bacterial genome using Markov models for three gene classes, *Nucleic Acids Res.*, 25, 3554-62.
- Lebre,S. (2004) Estimation de modèle MTD, Evry Univ. Internal Report.
- Lehoucq R.B., Sorensen B., Yang C. (1996) `ARPACK` user's guide: solution of large-scale eigenvalue problems by implicitly restarted Arnoldi methods. Technical report, Rice University.
- Richard H., Nuel G., (2003) `SPA`: simple web tool to assess statistical significance of DNA patterns. *Nucl. Acids. Res.*, 31(13):3679-81.
- Robelin D., Richard H., Prum B., (2003) `SIC: a tool to detect short inverted segments in a biological sequence`. *Nucl. Acids. Res.*, 31(13):3669-71.
- Ron D., Singer Y., Tishby N., (1996) The power of amnesia: learning probabilistic automata with Variable Memory Length, *Machine Learning*, 25.
- Schbath S., Prum B., Turckheim E. d. (1995) Exceptional motifs in different Markov chain models for a statistical analysis of DNA sequences. *J. Comp. Biol.*, 2:417-437.
- Stormo G.D. (1990) Consensus patterns in DNA. *Methods Enzymol.*, 183:211-221.

Index

- AIC
 - PhasedMarkov, 31
- BIC
 - PhasedMarkov, 31
- char_to_int
 - Translator, 74
- Coder.h, 79
- complementary_char
 - Translator, 74
- complementary_char_to_int
 - Translator, 74
- complementary_int
 - Translator, 74
- complementary_str_to_int
 - Translator, 74
- compute_count
 - Sequence, 67
- const.h, 80
- count_occurencies
 - PrimaryCount, 58
- count_p_occurencies
 - PrimaryCount, 58
- DOXYMAINPAGE.h, 82
- draw
 - PhasedPMarkov, 44
 - PMarkov, 52
- draw_markov_matrices
 - PhasedMarkov, 31
- draw_markov_matrix
 - Markov, 15
- estimate
 - Markov, 15, 16
 - PhasedMarkov, 31, 32
- file_to_count
 - PhasedMarkov, 32
- get_count
 - PrimaryCount, 59
- int_to_char
 - Translator, 74
- isMu
 - Markov, 16
- isPi
 - Markov, 16
- isPow
 - Markov, 16
- log_likelihood
 - PhasedMarkov, 32, 33
- log_ratio_likelihood
 - PhasedMarkov, 33
- Markov, 11
 - draw_markov_matrix, 15
 - estimate, 15, 16
 - isMu, 16
 - isPi, 16
 - isPow, 16
 - Markov, 13–15
 - Mu, 16
 - operator(), 16
 - proba_step, 17
- Markov.h, 83
- mean_post_log_likelihood
 - PhasedPMarkov, 44, 45
 - PMarkov, 53
- MTDMarkov, 18
 - MTDMarkov, 19
- MTDMarkov.h, 84
- Mu
 - Markov, 16
 - PhasedMarkov, 34
- operator()
 - Markov, 16
 - PhasedMarkov, 34
- Partition, 20
 - Partition, 21
- Partition.h, 85
- Partition_box.h, 86
- PhasedMarkov, 23
 - PhasedMarkov, 28–30
- PhasedMarkov
 - AIC, 31
 - BIC, 31

- draw_markov_matrices, 31
- estimate, 31, 32
- file_to_count, 32
- log_likelihood, 32, 33
- log_ratio_likelihood, 33
- Mu, 34
- operator(), 34
- PhasedMarkov, 28–30
- Pi, 34
- post_log_likelihood, 34
- print, 34
- proba, 35, 36
- proba_c, 36
- PhasedMarkov.h, 87
- PhasedMTDMarkov, 38
 - PhasedMTDMarkov, 38, 39
- PhasedMTDMarkov
 - PhasedMTDMarkov, 38, 39
- PhasedMTDMarkov.h, 88
- PhasedPMarkov, 40
 - PhasedPMarkov, 42, 43
- PhasedPMarkov
 - draw, 44
 - mean_post_log_likelihood, 44, 45
 - PhasedPMarkov, 42, 43
 - post_log_likelihood, 45
 - select, 46
- PhasedPMarkov.h, 89
- PhasedVLMarkov, 47
 - PhasedVLMarkov, 47, 48
- PhasedVLMarkov
 - PhasedVLMarkov, 47, 48
- PhasedVLMarkov.h, 90
- Pi
 - PhasedMarkov, 34
- PMarkov, 49
 - draw, 52
 - mean_post_log_likelihood, 53
 - PMarkov, 50–52
 - post_log_likelihood, 54
 - select, 54
- PMarkov.h, 91
- post_log_likelihood
 - PhasedMarkov, 34
 - PhasedPMarkov, 45
 - PMarkov, 54
- PrimaryCount, 55
- PrimaryCount
 - count_occurencies, 58
 - count_p_occurencies, 58
 - get_count, 59
- PrimaryCount.h, 92
- PrimarySequence, 60
- PrimarySequence.h, 93
- PrimarySequenceSet, 63
- PrimarySequenceSet.h, 94
- print
 - PhasedMarkov, 34
- proba
 - PhasedMarkov, 35, 36
- proba_c
 - PhasedMarkov, 36
- proba_step
 - Markov, 17
- select
 - PhasedPMarkov, 46
 - PMarkov, 54
- Sequence, 65
 - compute_count, 67
 - Sequence, 66, 67
- Sequence.h, 95
- SequenceSet, 68
 - SequenceSet, 68
- SequenceSet
 - SequenceSet, 68
- SequenceSet.h, 96
- str_to_int
 - Translator, 75
- strw_to_coded_int
 - Translator, 75
- strw_to_vect
 - Translator, 75
- Translator, 70
 - char_to_int, 74
 - complementary_char, 74
 - complementary_char_to_int, 74
 - complementary_int, 74
 - complementary_str_to_int, 74
 - int_to_char, 74
 - str_to_int, 75
 - strw_to_coded_int, 75
 - strw_to_vect, 75
 - Translator, 73
 - vect_to_word, 75
 - word_to_coded_int, 75
 - word_to_vect, 75
- Translator.h, 97
- vect_to_word
 - Translator, 75
- VLMarkov, 77
 - VLMarkov, 77
- VLMarkov.h, 98
- word_to_coded_int
 - Translator, 75

word_to_vect
Translator, 75