scikits-symbolic

Release 0.0

Laurent Pezard, Jean-Luc Blanc and others

Oct 09, 2023

CONTENTS:

1 Symbolic sequences 1

2 Discretizers 7

3 Algorithmic complexity procedures 9

4 Sequence generators 11

5 Contributors 13

6 Tutorial 15

7 Indices and tables 17

Python Module Index 19

CHAPTER

ONE

INTRODUCTION

Marcus and Williams [MW08] describe symbolic dynamics as:

"Symbolic Dynamics is the study of shift spaces, which consist of infinite or bi-infinite sequences defined by a shift-invariant constraint on the finite-length sub-words. Mappings between two such spaces can be regarded as codes or encodings. Shift spaces are classified, up to various kinds of invertible encodings, by combinatorial, algebraic, topological and measure-theoretic invariants. The subject is intimately related to dynamical systems, ergodic theory, automata theory and information theory."

[**?**] provides a review of symbolic dynamics

[**?**] is a Python package with more theoretic approach than this one.

Find the old reference…

[**?**] gives an example of using scikits-symbolic in the context of behavioral studies.

CHAPTER TWO

SYMBOLIC SEQUENCES

A symbolic sequence is a list of symbols taken from a finite alphabet of length k

Internally they are encoded according to integers from 0 to $k-1$

Alphabet should be bidirectional dictionary. . .

This is the doc!

class sequence.**Alphabet**(*nsymb*)

The set of states or symbols that can be visited for a sequence or Markov process realization.

Tests on State and Alphabet

```
>>> state1 = State('One')
>>> state1
State(- | One)
```

An integer representation of a state is only attributed once the state is inserted in an alphabet.

```
>>> state2 = State('Two')
>>> state3 = State('Three')
```

Alphabets can be created with a list of states:

```
>>> alpha = Alphabet([state1, state2, state3])
>>> alpha
Alphabet[State(0 | One), State(1 | Two), State(2 | Three)]
>>> print(alpha)
Alphabet[State(0 | One), State(1 | Two), State(2 | Three)]
>>> len(alpha)
3
```

Alphabets can also be created using only the length as argument.

```
>>> beta = Alphabet(3)
>>> beta
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2)]
>>> alpha[0]
State(0 | One)
```

States can be changed but the integer value is kept:

```
>>> alpha[1] = State('Deux')
>>> alpha
Alphabet[State(0 | One), State(1 | Deux), State(2 | Three)]
```

Alphabet's states can be changed using a dictionary representation.

```
>>> alpha.rename({0 : 'Uno', 2 : 'Tre'})
>>> alpha
Alphabet[State(0 | Uno), State(1 | Deux), State(2 | Tre)]
>>> beta.rename({0 : 'Uno', 1 : 'Deux', 2 : 'Tre'})
>>> alpha == beta
True
```

sequence.**DEFAULT_DTYPE**
alias of `numpy.uint16`

class sequence.**Sequence**(*symbols*, *alphabet*, *dtype=<class 'numpy.uint16'>*, *check=True*)
Defines a symbolic sequence coded using integers in $\{0, k-1\}$ and their methods.

Test for the Sequence class and its methods

```
>>> a = [1,0,0,0,1,0,1,0,1,1,0,1]
>>> b = [0,1,0,0,1,1,1,1,0,0,1,0]
>>> A = Alphabet(['a','b'])
>>> s1 = Sequence(a,A)
>>> s2 = Sequence(b,A)
>>> s1
Sequence: [1 0 0 0 1 0 1 0 1 1 0 1]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> print(s1)
[1 0 0 0 1 0 1 0 1 1 0 1]
```

The length of the alphabet is a property named *k* of the sequence.

```
>>> s1.k
2
>>> s1.alphabet
Alphabet[State(0 | a), State(1 | b)]
```

Slices return Sequence object

```
>>> s1[0]
Sequence: [1]
Alphabet[State(0 | a), State(1 | b)]
N = 1 ; k = 2
>>> s1[4:8]
Sequence: [1 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 4 ; k = 2
>>> s1[s1.ivals < 1]
Sequence: [0 0 0 0 0 0]
Alphabet[State(0 | a), State(1 | b)]
N = 6 ; k = 2
```

We can access to the lenght of a sequence

```
>>> len(s1)
12
```

Concatenation of two sequences return Sequence object if the alphabet of the two sequences are the same

```
>>> s1 + s2
Sequence: [1 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 24 ; k = 2
```

Comparison between two sequences with the same length returns a Sequence object with the results of the comparison

```
>>> s1 == s2
Sequence: [0 0 1 1 1 0 1 0 0 0 0 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 > s2
Sequence: [1 0 0 0 0 0 0 0 1 1 0 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 >= s2
Sequence: [1 0 1 1 1 0 1 0 1 1 0 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 < s2
Sequence: [0 1 0 0 0 1 0 1 0 0 1 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 <= s2
Sequence: [0 1 1 1 1 1 1 1 0 0 1 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 != s2
Sequence: [1 1 0 0 0 1 0 1 1 1 1 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
```

With binary sequences, logical operators return e Sequence object with the result

```
>>> s1 & s2
Sequence: [0 0 0 0 1 0 1 0 0 0 0 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 ^ s2
Sequence: [1 1 0 0 0 1 0 1 1 1 1 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 | s2
Sequence: [1 1 0 0 1 1 1 1 1 1 1 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
```

It is possible to transform a sequence in place

```
>>> s1.roll(2)
>>> s1
Sequence: [0 1 1 0 0 0 1 0 1 0 1 1]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s1.reverse()
>>> s1
Sequence: [1 1 0 1 0 1 0 0 0 1 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s1.reduce()
>>> s1
Sequence: [1 0 1 0 1 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 8 ; k = 2
```

And we can do the same creating a new sequence modified from a base sequence

```
>>> s3 = roll(s2,12)
>>> s3
Sequence: [0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s3 = reduce(s2)
>>> s3
Sequence: [0 1 0 1 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 7 ; k = 2
>>> s3 = reverse(s2)
>>> s3
Sequence: [0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
```

This functions provide us information about a sequence

```
>>> s1.count()
array([4, 4])
>>> s1.frequency()
array([0.5, 0.5])
>>> issequence(s2)
True
```

From a list of sequences of the same length but that can have different alphabets, we can recode them creating a new sequence with new symbols and a new alphabet

```
>>> B = Alphabet(['aa','bb','cc'])
>>> s4 = Sequence([2,2,1,0,2,0,0,1,2,1,0,0],B)
>>> s4
Sequence: [2 2 1 0 2 0 0 1 2 1 0 0]
Alphabet[State(0 | aa), State(1 | bb), State(2 | cc)]
N = 12 ; k = 3
```

(continues on next page)

(continued from previous page)

```
>>> s2
Sequence: [0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s3 = recode([s2,s4], new_alphabet=True, names=['seq2','seq4'])
>>> s3
Sequence: [2 5 1 0 5 3 3 4 2 1 3 0]
Alphabet[State(0 | seq2_a+seq4_aa), State(1 | seq2_a+seq4_bb), State(2 | seq2_
↪a+seq4_cc), State(3 | seq2_b+seq4_aa), State(4 | seq2_b+seq4_bb), State(5 | seq2_
↪b+seq4_cc)]
N = 12 ; k = 6
```

count(*ival=None*)
Counts the number of each symbol in $\{0, k-1\}$ if code is None or the number of the code symbol

Returns a numpy.ndarray of integers

frequency()
Returns the probability of each symbol in $\{0, k-1\}$

Returns a numpy.ndarray of floats

reduce()
Delete the repetitions of symbols in a sequence *in place*

reverse()
Reverse the sequence *in place*

See also:

numpy.flipud function

roll(*step*)
Roll the sequence *in place*

See also:

numpy.roll function

shuffle()
Shuffle the order of the sequence *in place*.

See also:

numpy.random.shuffle function

class sequence.**State**(*strval*)
A state (or symbol) is used to define the state of the system at time t.

It has two properties:

- *strval*: its name which can be accessed, changed but not deleted
- *ival*: its associated integer value which can be accessed but neither changed nor deleted

setter and deleter raise exception for explicit behavior.

sequence.**issequence**(*obj*)
Returns True if x is a symbolic sequence

sequence.**recode**(*lseq*, *new_alphabet=False*, *sep='+'*, *names=None*)
Recodes a list of sequences with (possibly) different alphabets but with the same length (This is an error to pass Sequences with different length.) A new dictionnary is built for the new sequence.

Parameters `lseq` – a list of Sequences

Raises `LengthError`: when the length of the Sequences are different.

Returns a Sequence

`sequence.reduce(seq)`
Returns a reduced sequence (ie only keep the transitions)

`sequence.reverse(seq)`
Reverse the sequence

`sequence.roll(seq, step)`
Roll the sequence

`sequence.shuffle(seq)`
Shuffle the sequence

`sequence.transform(seq, correspondance, new_alphabet=None)`
Transforms the initial sequence according to the correspondence iterable

`sequence.words(seq, wlen, new_alphabet=False)`
Returns a sequence encoded according to the m-words in seq

Todo: Write the doc of "words"

CHAPTER

THREE

DISCRETIZERS

discretize.**partition**(*arr*, *method='histogram'*, *nbin=10*, *d=None*)

Discretize a continuous series according to method.

Methods are described in Hlavackova-Schindler et al. Physics Reports 441 (2007) 1–46 pages 14–19

method = 'histogram' simple histogram method with equidistant binning

method = 'marginal_equiquantization' marginal equiquantization ie does its best to let equal number of observation in each bin.

Parameters

- **x** – a continuous series
- **method** – a string in *["histogram", "marginal_equiquantization"]*
- **nbin** – the number of bins ie the length of the alphabet
- **d** – a dictionary

Raises `NotImplementedError` if method is not in the list above.

Returns A symbolic Sequence

Todo: To be completed with the other methods described in Hlavackova-Schindler (2007)

Hint: look at R implementation of histogram function.

Tests and examples of the functionnement of the module

```
>>> x = np.linspace(0,10,11)
>>> x
array([ 0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9., 10.])
>>> seq = partition(x, method='histogram', nbin=6)
>>> seq
Sequence: [0 0 1 1 2 3 3 4 4 5 5]
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2), State(3 | 3), State(4 | 4),␣
↪State(5 | 5)]
N = 11 ; k = 6
>>> seq = partition(x, method='marginal_equiquantization',nbin=6)
>>> seq
Sequence: [0 0 1 1 2 2 3 3 4 4 5]
```

(continues on next page)

(continued from previous page)

```
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2), State(3 | 3), State(4 | 4),␣
↪State(5 | 5)]
N = 11 ; k = 6
```

discretize.**phase_cluster**(*data*, *nb_symb*, *target_dim=10*)

This function provides the symbolic dynamic of a multivariate data It is based on the clusterisation of the "phase space" of the channels of MEG temporal signal

Parameters

- **data** – The input matrix, the lines are the channels and the columns are the time, must be an array
- **nb_symb** – The number of bins used for the clusterisation i.e. the number of symbols of the symbolic sequences that will be created

:param nb_vp : The number of eigen vectors that we want to conserve to project our data on it

discretize.**subdivision**(*data*, *iter_max*)

Ulam method Adaptive subdivision technique based on:

Set oriented numerical methods for dynamical systems Dellnitz M. and Junge O. Handbook of dynamical systems vol. 2 p. 221-264 Elsevier 2002.

and

Numerical approximation of random attractors Keller H. and Ochs G. in "Stochastic dynamics" Crauel H. and Gundlach M. Eds Springer 1999. p. 93-115

Input

`x`: numpy array `kmax`: integer, maximum number of boxes

discretize.**symbolize**(*arr*, *bins*, *d=None*)

Todo: is this funtion *symbolize* useful? (duplicate numpy.digitize?) Qu'est-ce que nbin ? Peut-être d ?

```
>>>
```

CHAPTER

FOUR

ALGORITHMIC COMPLEXITY PROCEDURES

Algorithms and procedures related to algorithmic approach of complexity

`algorithmic.`**`contains_sublist`**(*lst*, *sublst*)

Check wether a sublist appears in a list

found at: http://stackoverflow.com/questions/3313590/… … check-for-presence-of-a-sublist-in-python

`algorithmic.`**`lempel_ziv`**(*seq*, *parsing='lz76'*, *norm=False*, *nbsur=None*)

Returns the Lempel-Ziv normalized complexity using either lz76 or lz77 parsing.

Parameters

- **seq** – a Sequence object
- **parsing** – a string in *["lz76", "lz77"]*
- **norm** – a bolean (should the complexity be normalized?)
- **ns** – the number of surrogate data used in the normalization.

Raise `NotImplementedError` if *parsing* is not in the list above.

Returns a float (the Lempel-Ziv complexity)

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> lempel_ziv(seq)
0.6
```

`algorithmic.`**`lz76`**(*arr*, *summary=False*)

Returns Lempel-Ziv complexity according to LZ76 parsing.

Parameters

- **arr** – an array of integers
- **summary** – A boolean (should the dictionary be returned)

Returns either an integer (*summary=False*) or a tuple (*summary=True*) with an integer and a list of strings.

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> lz76(a)
92
```

`algorithmic.lz77`(*arr*, *summary=False*)

Returns Lempel-Ziv complexity according to LZ77 parsing.

Parameters

- **arr** – an array of integers
- **summary** – A boolean (should the dictionary be returned)

Returns either an integer (*summary=False*) or a tuple (*summary=True*) with an integer and a list of strings.

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> lz77(a)
158
```

CHAPTER FIVE

STOCHASTIC

Defines stochastic matrices

`stochastic.conditional_matrix(seq1, seq2)`

Returns the conditional matrix ie P(s1=j | x2=i).

This is estimated using the maximum likelihood estimator.

Parameters

- **seq1** – a symbolic Sequence object
- **seq2** – a symbolic Sequence object

Returns A numpy.matrix of floats

..todo:

```
check the doc and implementation of conditional_matrix
```

NB: lines should sum to one (one should go somewhere) see markov_sequence in generate.py ie np.sum(matrix, axis=1) == [[1]. . . [1]]

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> conditional_matrix(seq1, seq2)
[[0.71947674 0.28052326]
 [0.69551282 0.30448718]] [688 312]
(array([], dtype=int64),)
matrix([[0.71947674, 0.28052326],
        [0.69551282, 0.30448718]])
```

`stochastic.influence_matrix(seq1, seq2, time=1)`

Returns the influence matrix ie P(x1(T+t)=j | x2(T)=i).

This is estimated using the maximum likelihood estimator.

Parameters

- **seq1** – a symbolic Sequence object

- **seq2** – a symbolic Sequence object

Returns A numpy.matrix of floats

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> influence_matrix(seq1, seq2)
[[0.70887918 0.29112082]
 [0.71794872 0.28205128]] [687 312]
(array([], dtype=int64),)
matrix([[0.70887918, 0.29112082],
        [0.71794872, 0.28205128]])
```

`stochastic.`**`transition_matrix`**(*seq*, *time=1*)

Returns the transition matrix.

This is estimated using the maximum likelihood estimator.

Parameters **seq** – a symbolic Sequence object

Returns A numpy.matrix of floats

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> transition_matrix(seq)
[[0.70224719 0.29775281]
 [0.73519164 0.26480836]] [712 287]
(array([], dtype=int64),)
matrix([[0.70224719, 0.29775281],
        [0.73519164, 0.26480836]])
```

CHAPTER

SIX

INFORMATION

`information.H(seq)`

Returns Shannon's (metric) entropy of sequence

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

On fixe la seed pour pouvoir contrôler la génération de vecteur "aléatoires"

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> metric_entropy(seq)
0.6003511877776578
```

`information.R(seq, coef)`

Returns the Rényi entropy

..todo:

```
Make the doc of renyi_entropy!
```

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> renyi_entropy(seq, 0.9)
0.6088567303148161
```

`information.T(seq)`

Returns the topological entropy

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
```

(continues on next page)

(continued from previous page)

```
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> topological_entropy(seq)
0.6931471805599453
```

`information.`**`block_entropy`**(*seq*, *wlen*)

Returns the block entropy

Parameters

- **seq** – a symbolic Sequence object
- **n** – the block length
- **method** – a string in *["metric", "shannon", "topological", "renyi", "all"]*

Raises `ValueError` if $n < 0$

`NotImplementedError` if the method is not in the list above.

Returns either the value of the demanded entropy or all their value in a tuple *Tn, Hn, hav*

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> block_entropy(seq, 6)
3.577559335188841
```

`information.`**`effective_complexity`**(*seq*, *n_max*)

Computes the effective complexity defined by Grassberger

..todo:

```
Make the doc of effective complexity
```

`information.`**`entropy_rate`**(*seq*, *wlen*, *method='average'*)

Returns the entropy rate

Parameters

- **seq** – a symbolic Sequence object
- **method** – a string in ['lempel_ziv', 'average']
- **kwargs** – parameter to pass to the method

Returns the entropy rate computed using the method

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> entropy_rate(seq, 6)
0.5962598891981402
```

information.**metric_entropy**(*seq*)

Returns Shannon's (metric) entropy of sequence

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

On fixe la seed pour pouvoir contrôler la génération de vecteur "aléatoires"

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> metric_entropy(seq)
0.6003511877776578
```

information.**multi_information**(*seq1*, *seq2*, *seq3*)

Computes the multi information for 3 symbolic sequences,

A kind of 3 variables mutual information (See Blanc J.L. & Coq J.O., J.Physiol. 2007)

Parameters **z** (*x, y,*) – three symbolic Sequences

Returns the three variable mutual information

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(3)
>>> c = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> seq3 = S.Sequence(c,A)
>>> multi_information(seq1, seq2, seq3)
-4.8757282800737656e-05
```

information.**mutual_information**(*seq1*, *seq2*)

Computes the mutual information for symbolic sequences

Parameters **y** (*x,*) – two symbolic Sequences

Returns the mutual information (float)

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> mutual_information(seq1, seq2)
0.0002988020334349084
```

`information.`**`renyi_entropy`**`(`*seq*, *coef*`)`

Returns the Rényi entropy

..todo:

```
Make the doc of renyi_entropy!
```

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> renyi_entropy(seq, 0.9)
0.6088567303148161
```

`information.`**`shannon_entropy`**`(`*seq*`)`

Returns Shannon's (metric) entropy of sequence

Parameters **`seq`** – a symbolic Sequence object

Returns a float

Example :

On fixe la seed pour pouvoir contrôler la génération de vecteur "aléatoires"

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> metric_entropy(seq)
0.6003511877776578
```

`information.`**`topological_entropy`**`(`*seq*`)`

Returns the topological entropy

Parameters **`seq`** – a symbolic Sequence object

Returns a float

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> topological_entropy(seq)
0.6931471805599453
```

`information.`**`transfer_entropy`**`(`*seq1*, *seq1p*, *seq2*`)`

Computes the symbolic transfer entropy T y->x

we can use: P(x|y) = P(x,y) / P(y) in the formula: P(x+, x, y) log (P(x+|x,y) / P(x+|x))

but (see Kugiumtzis, 2011)

-H(x+, x, y) + H(x, y) + H(x+, x) - H(x)

gives a better implementation

see:

Schreiber (2000) Staniek and Lehnertz (2008) Symbolic transfer entropy PRE Kugiumtzis (2011) Journal of Nonlinear Systems and Applications vol. 2 n°3 http://arxiv.org/abs/1007.0357

CHAPTER

SEVEN

SEQUENCE GENERATORS

Generation of specified symbolic sequences

`generator.`**`binary_logistic_sequence`**(*length*, *param*, *xinit*, *threshold=0.5*, *skip=100*)

Returns a binary sequence with logistic dynamics according to the parameter μ.

The equation used here is: $x(t+1) = \mu x(1-x)$

Parameters

- **`N`** – the length of the sequence
- **`mu`** – the paramter for the logistic equation
- **`thresh`** – the threshold value to make a binary sequence

Returns A binary Sequence

`generator.`**`binary_map1d_sequence`**(*length*, *map1d*, *xinit*, *threshold=0.5*, *skip=100*)

Returns a binary sequence with a specified one-dimensional map dynamics

map1d can be specified such as: map1d = lambda x: 3.4 * x * (1 - x) or any function the defines x(t+1) as a function of x(t)

Parameters

- **`N`** – the length of the sequence
- **`thresh`** – the threshold value to make a binary sequence

Returns A binary Sequence

`generator.`**`generate`**(*method*, *N*, *k*, **args*)

Generates a Sequence according to a *method*

Parameters

- **`method`** – a string in *[“uniform”, “markov”, “binary_logistic”]*
- **`N`** (*`integer`*) – the length of the sequence
- **`k`** – the length of the alphabet
- **`k`** – integer
- **`args`** – supplementary parameters (transition matrix and order for *“markov”* and μ for *“binary_logisitic”*)

Raises `NotImplementedError` if *method* is not in the list above.

Returns A Sequence object.

Todo: Should we check the type of *N* and *k*

Todo: Deal with args more clearly (make a dict)?

Todo: should (N,k,d) be replaced by a dictionary dict(N=..., k=...,d=...)

Todo: Check NS code for cantor and cantor_id sequence

Todo: implement binary_tent, Gaussian, Poissonian, etc.??

`generator.markov_sequence`(*length*, *alen*, *markov_matrix*, *order*)

Returns as sequence of a Markov process of order o with transition matrix M.

Parameters

- **N** – the length of the sequence
- **k** – the length of the alphabet
- **M** – the transition matrix
- **order** – the order of the Markov process

Raises `ValueError`: if the shape of M does not correspond to the order of the process ie $k^o imesk$

Returns A sequence object

NB:

- lines of Markov matrix give the probability to transition to one of the k symbols of the alphabet (so sum(markov_matrix[line] == 1) (ie np.sum(matrix, axis=1) == [[1]...[1]]

`generator.uniform_sequence`(*length*, *alen*)

Returns an uniform random sequence.

Parameters

- **N** – the length of the sequence
- **k** – the length of the alphabet

Returns a Sequence object

CHAPTER

EIGHT

INPUT-OUTPUT PROCEDURES

iosymb.**read_codix**(*fname*, *data_only=True*)

Reads data file from the codix-encoder of the codix software suite for behavioral studies.

returns in all cases a dictionary with data['site']['code'] = Sequence

CHAPTER

NINE

VISUALISATION PROCEDURES

`viz.plot`(*seq*, *xlabel='Time'*, *ylabel='States'*, *title='Simple plot'*, *labelsize=15*, *titlesize=25*, *color='blue'*, ***kwargs*)

Simple (discrete / symbolic) time series plot

`viz.plot_bar`(*seq*, *xlabel='Time'*, *ylabel='States'*, *title='Bar plot'*, *labelsize=15*, *titlesize=25*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*, ***kwargs*)

Plots bar code like graph.

`viz.plot_color`(*seq*, *aspect=5*, *title='Sequence'*, *xlabel='Time'*, *labelsize=15*, *titlesize=25*, ***kwargs*)

Plots as ???

`viz.plot_grid`(*seq1*, *seq2*, *xlabel='1st sequence'*, *ylabel='2nd Sequence'*, *title='Grid plot'*, *labelsize=15*, *titlesize=25*, *color='blue'*, *alpha=0.3*, *scale=100*, *jitter=0.4*, ***kwargs*)

Plots state-space grids plots inspired from

Hollenstein T. (2013) State space grids. Springer.

`viz.plot_independence`(*seq1*, *seq2*, *xlabel='1st sequence'*, *ylabel='2nd Sequence'*, *title='Independence plot'*, *labelsize=15*, *titlesize=25*, *color=('blue', 'red')*, *alpha=0.3*, *scale=100*, ***kwargs*)

Plots state-space grids representing the elements of the mutual information between sequences.

CHAPTER

TEN

CONTRIBUTORS

Main developers

- Laurent Pezard (2007-)
- Jean-Luc Blanc (2007-)
- Noelia Montero (XXXX-XXXX)
- Yann Mahnoun (XXXX-XXXX)
- Nicolas Schmidt (XXXX-XXXX)
- Abir Hadriche (XXXX-XXXX)
- Lucas Becquet (2023)
- Florent Boyer-Aymé (XXXX-XXXX)
- Alexandre Veyrié (XXXX-XXXX)
- Inès Bertuzzi (XXXX-XXXX)

CHAPTER

ELEVEN

TUTORIAL

[5]:
```
import sys
sys.path.append('../../../symbolic')

import sequence as S
```

[6]:
```
a = S.Alphabet(3)
s = S.Sequence([0,1,2,0,2,1], a)

print(s)
s
```

```
[0 1 2 0 2 1]
```

[6]:
```
Sequence: [0 1 2 0 2 1]
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2)]
N = 6 ; k = 3
```

[ ]:

[ ]:

CHAPTER TWELVE

INFANT-MOTHER INTERACTION

This tutorial is based on an example extracted from the data analyzed in {cite:p}`DobaEtAl22` article see [**?**].

Behavioral interaction between mother and her infant are video recorded while playing. Behaviors are encoded according to several categories. They are recorded during a first session before the mother leave temporally the room and after the mother comes back. The software used to encode the videos is called codix see [PDPN24]_

12.1 Read data

[14]:
```python
import sys
sys.path.append('../../../symbolic')

import sequence as S # sequence module from scikits.symbolic
import iosymb as IO # IO from scikits.symbolic
import viz as V

data_S1 = IO.read_codix('data/209_S1')
data_S2 = IO.read_codix('data/209_S2')
```

The `read_codix` function returns a dictionary organized as `data[person][code]` which value is a symbolic `Sequence` coded according to a specific `Alphabet`:

[15]:
```python
for person in data_S1.keys():
    print(person+': ')
    for code in data_S1[person].keys():
        print("\t"+code+': \t', data_S1[person][code].alphabet)
```

```
Bebe:
	Mouvement: 	 Alphabet[State(0 | Non), State(1 | Oui)]
	Expression_faciale: 	 Alphabet[State(0 | neutre), State(1 | sourit), State(2 |␣
↪negatif)]
	Regard: 	 Alphabet[State(0 | ailleurs), State(1 | vers_la_mere)]
	Sons: 	 Alphabet[State(0 | Silence), State(1 | vocalisation), State(2 |␣
↪negatif)]
Mere:
	Expression_faciale: 	 Alphabet[State(0 | Neutre), State(1 | Expressif)]
	Sti_motrices: 	 Alphabet[State(0 | Absence), State(1 | Avec_contact), State(2 |␣
↪Sans_contact)]
	Regard: 	 Alphabet[State(0 | Ailleurs), State(1 | vers_bebe)]
```

(continues on next page)

(continued from previous page)

```
        Sti_verbale:    Alphabet[State(0 | Silence), State(1 | Inference), State(2 |␣
↪Sons)]
        Jeu:     Alphabet[State(0 | Absence), State(1 | Avec_objet), State(2 | Sans_
↪objet)]
```

In the study we discarded the facial expression code since the face of the mother and the infant could not be seen all the time.

[16]:
```
bmv1 = data_S1['Bebe']['Mouvement']
bcv1 = data_S1['Bebe']['Sons']
bre1 = data_S1['Bebe']['Regard']
bmv2 = data_S2['Bebe']['Mouvement']
bcv2 = data_S2['Bebe']['Sons']
bre2 = data_S2['Bebe']['Regard']

mmv1 = data_S1['Mere']['Sti_motrices']
mcv1 = data_S1['Mere']['Sti_verbale']
mre1 = data_S1['Mere']['Regard']
mmv2 = data_S2['Mere']['Sti_motrices']
mcv2 = data_S2['Mere']['Sti_verbale']
mre2 = data_S2['Mere']['Regard']

# rename motor behavior in English :-)
eng_mvt = {0:'NoMvt', 1:'Touch', 2:'NoTouch'}
mmv1.alphabet.rename(eng_mvt)
mmv2.alphabet.rename(eng_mvt)
```

12.2 Recode and transform sequences

The infant's behavior was also recoded according to a general level of activity.

First, the three sequences (motor, verbal and gaze) are recoded according to the cartesian product of the alphabets:

[17]:
```
bbstate1 = S.recode([bmv1, bcv1, bre1], new_alphabet=True, names=['Mvt','Verb','Gaz'])
bbstate2 = S.recode([bmv2, bcv2, bre2], new_alphabet=True, names=['Mvt','Verb','Gaz'])
print(bbstate1.alphabet) # some pretty print would be better...
```

```
Alphabet[State(0 | Mvt_Non+Verb_Silence+Gaz_ailleurs), State(1 | Mvt_Non+Verb_
↪Silence+Gaz_vers_la_mere), State(2 | Mvt_Non+Verb_vocalisation+Gaz_ailleurs), State(3␣
↪| Mvt_Non+Verb_vocalisation+Gaz_vers_la_mere), State(4 | Mvt_Non+Verb_negatif+Gaz_
↪ailleurs), State(5 | Mvt_Non+Verb_negatif+Gaz_vers_la_mere), State(6 | Mvt_Oui+Verb_
↪Silence+Gaz_ailleurs), State(7 | Mvt_Oui+Verb_Silence+Gaz_vers_la_mere), State(8 | Mvt_
↪Oui+Verb_vocalisation+Gaz_ailleurs), State(9 | Mvt_Oui+Verb_vocalisation+Gaz_vers_la_
↪mere), State(10 | Mvt_Oui+Verb_negatif+Gaz_ailleurs), State(11 | Mvt_Oui+Verb_
↪negatif+Gaz_vers_la_mere)]
```

Then, states are transformed according to a correspondance table:

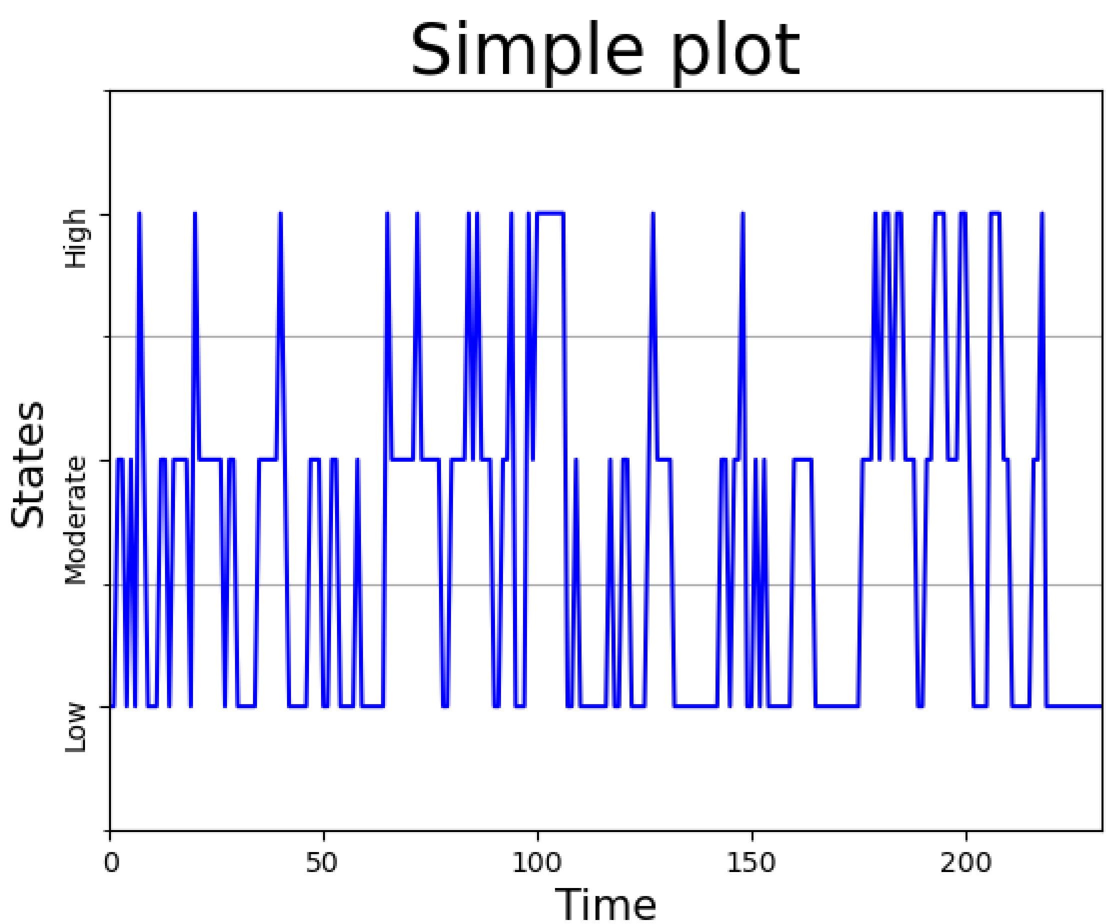
[18]:
```
naint = S.Alphabet(['Low','Moderate', 'High'])
bbglo1 = S.transform(bbstate1, [0,0,0,1,0,1,0,1,1,2,1,2], new_alphabet=naint)
bbglo2 = S.transform(bbstate2, [0,0,0,1,0,1,0,1,1,2,1,2], new_alphabet=naint)
bbglo1
```

[18]:
```
Sequence: [0 0 1 1 0 1 0 2 1 0 0 0 1 1 0 1 1 1 1 0 2 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1
 1 1 1 2 1 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 2 1 1 1 1 1 1 2 1
 1 1 1 1 0 0 1 1 1 1 2 1 2 1 1 1 0 0 1 1 2 0 0 0 2 1 2 2 2 2 2 2 2 0 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 2 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 2 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 2 1 2 2 1 2
 2 1 1 1 0 0 1 1 2 2 2 1 1 1 2 2 1 0 0 0 0 2 2 2 1 1 0 0 0 0 0 1 1 2 0 0 0
 0 0 0 0 0 0 0 0 0 0 0]
Alphabet[State(0 | Low), State(1 | Moderate), State(2 | High)]
N = 233 ; k = 3
```

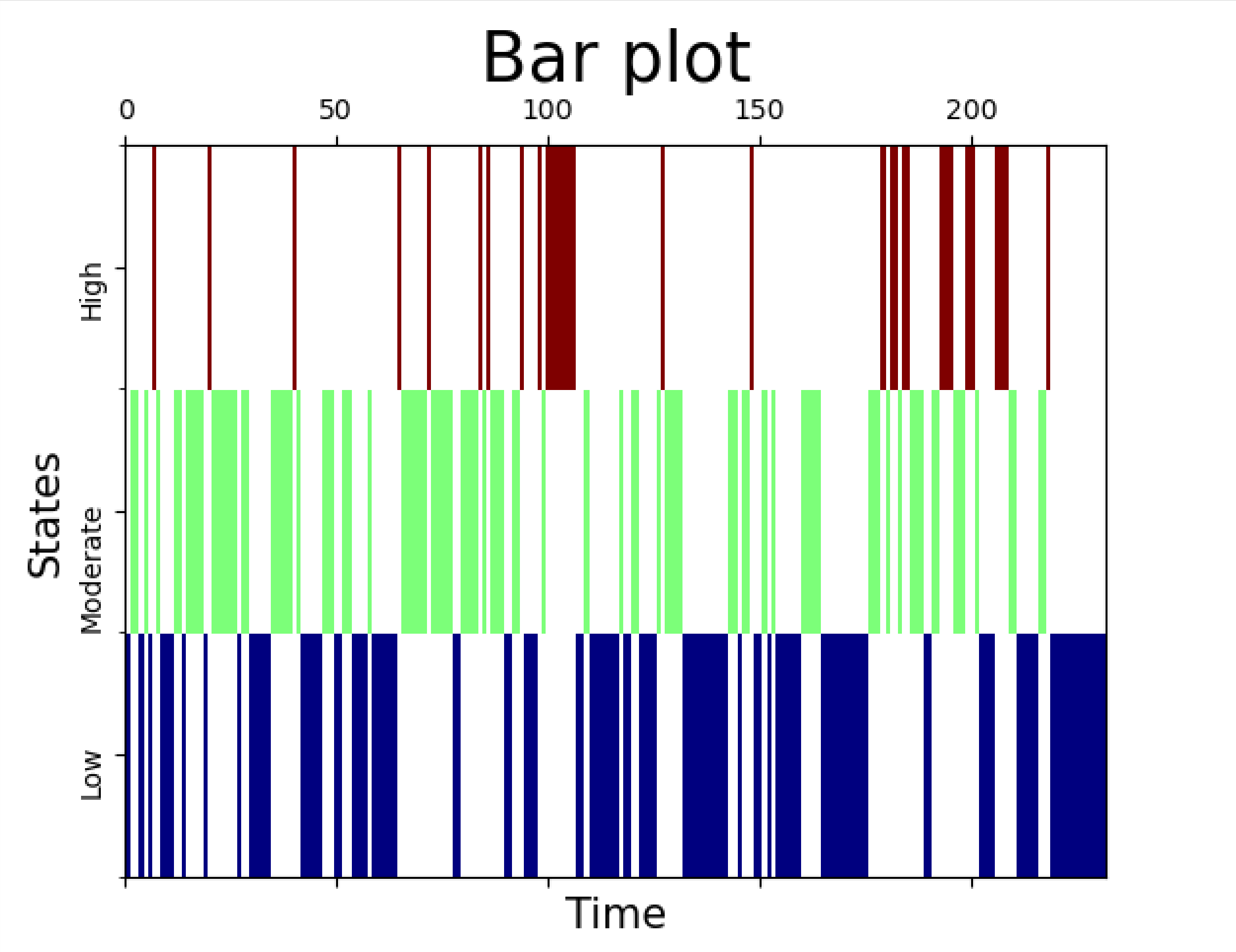
12.3 Visualise sequences

12.3.1 One sequence

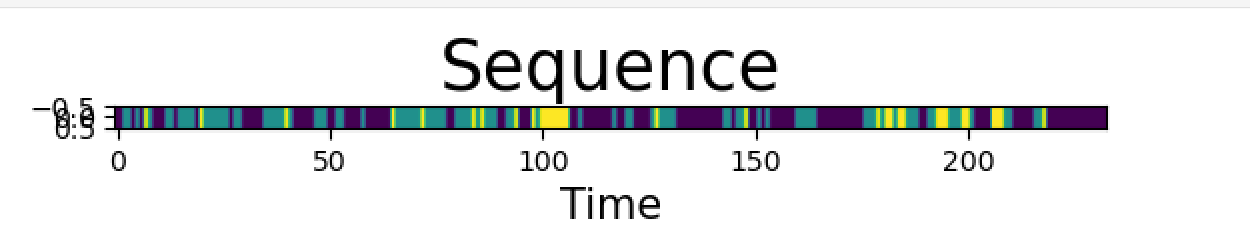
[19]:
```
V.plot(bbglo1)
```



[20]:
```
V.plot_bar(bbglo1)
```

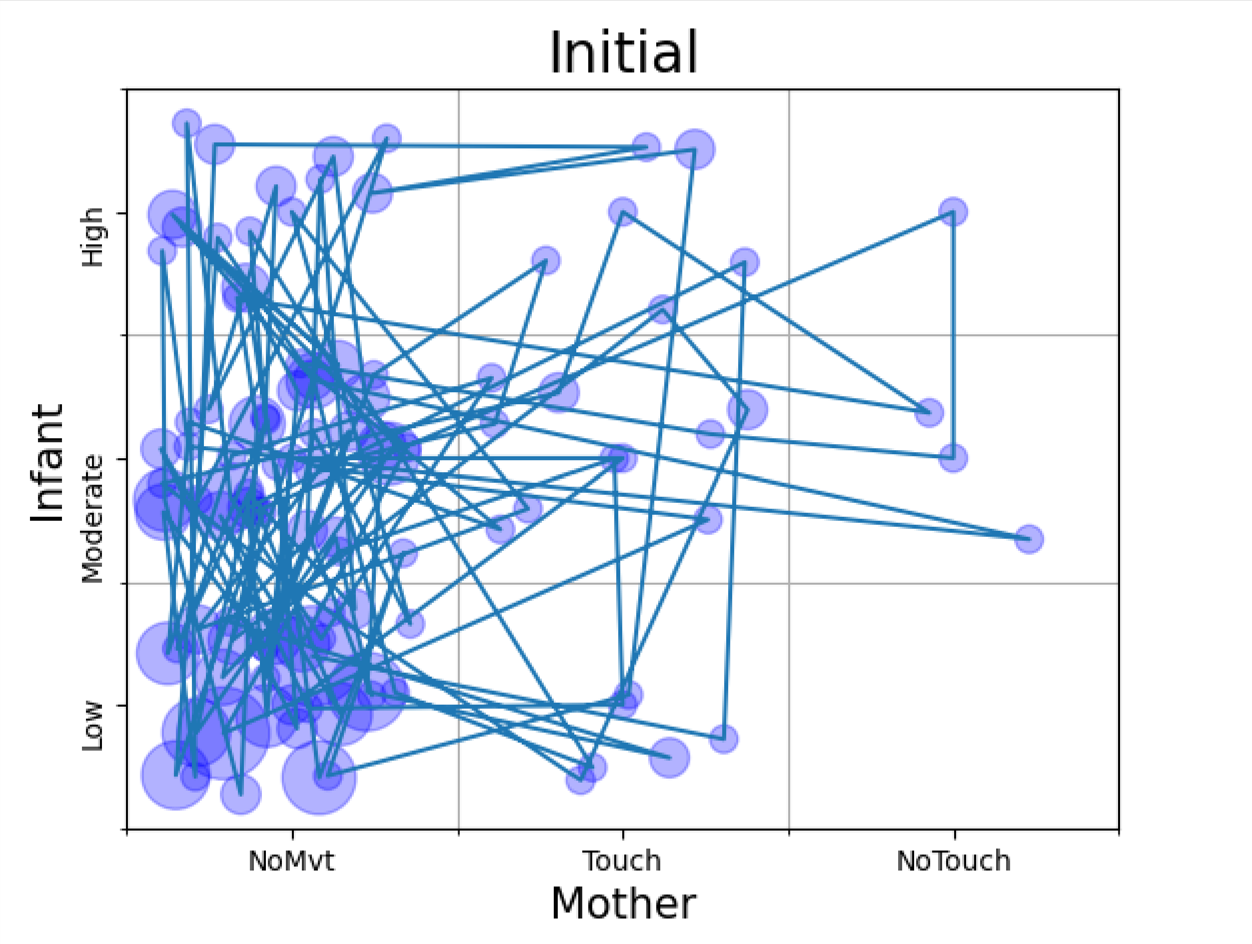


[21]:
```
V.plot_color(bbglo1)
```

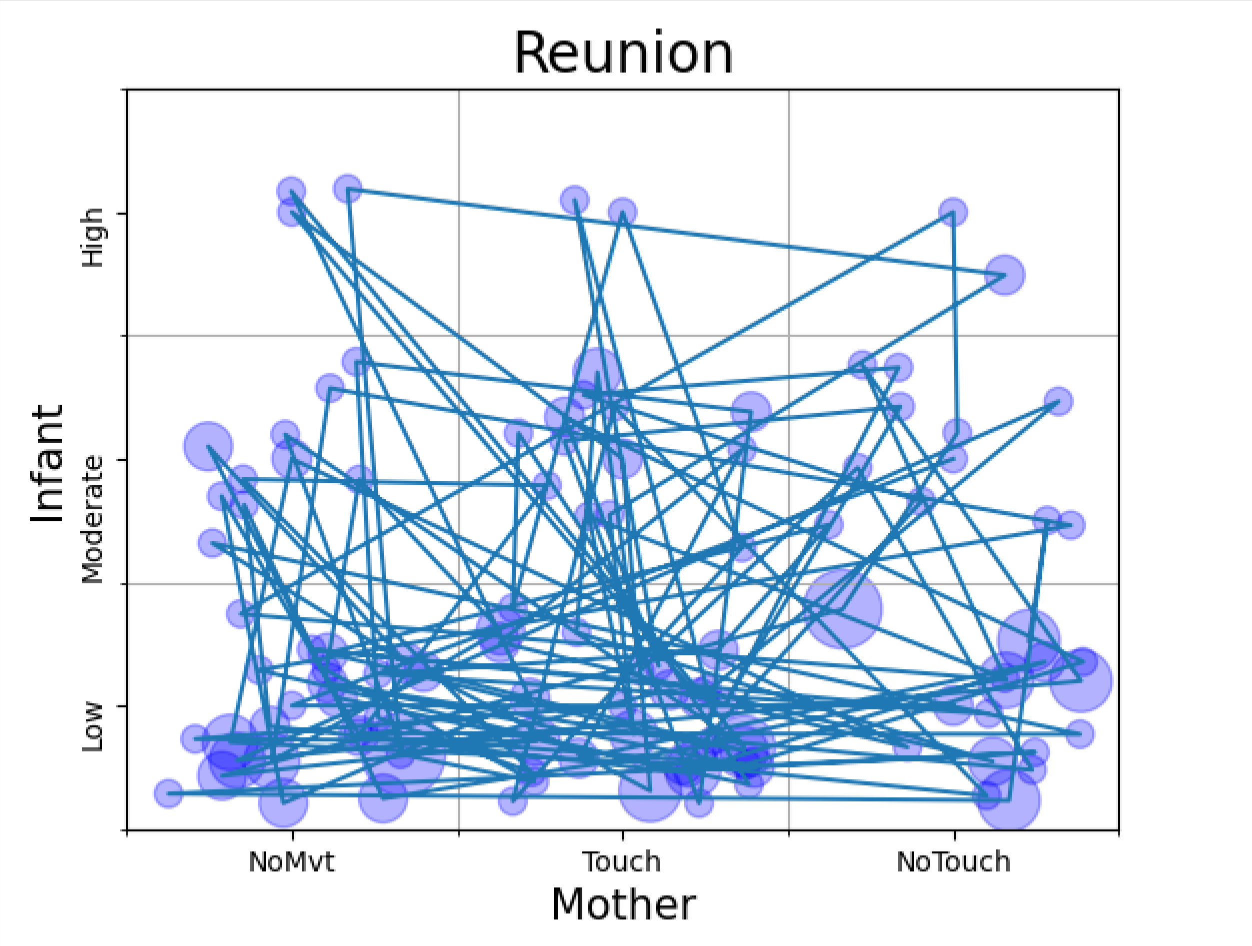


12.3.2 Two sequences

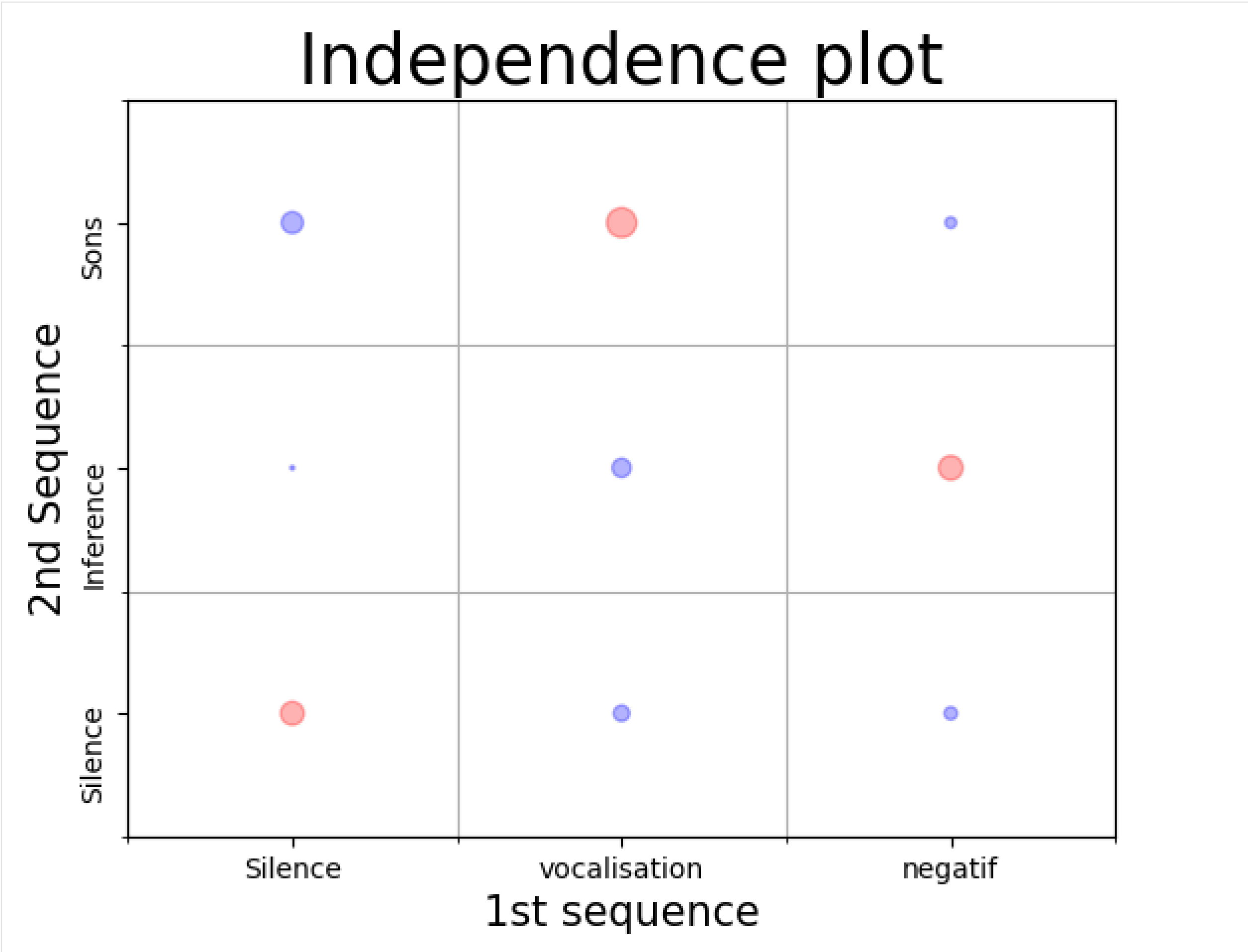
[26]:
```
V.plot_grid(mmv1, bbglo1, xlabel='Mother', ylabel='Infant', title='Initial',␣
↪titlesize=20)
```




```
[23]: V.plot_grid(mmv2, bbglo2, xlabel='Mother', ylabel='Infant', title='Reunion',␣
      ↪titlesize=20)
```




```
[24]: V.plot_independence(bcv1,mcv1, scale=2500)
```


Independence plot
2nd Sequence
Sons
Inference
Silence
Silence
vocalisation
negatif
1st sequence

CHAPTER

THIRTEEN

REFERENCES

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[MaWi08] Brian Marcus and Susan Williams (2008) Symbolic dynamics. Scholarpedia, 3(11):2923.

[HiAm23] Yoshito Hirata and José M. Amigó (2023) A review of symbolic dynamics and symbolic reconstruction of dynamical systems. Chaos, 33, 052101.

[Cai23] https://pypi.org/project/symbolic-dynamics/

[DoPN22] Karyn Doba, Laurent Pezard and Jean-Louis Nandrino (2022) How do maternal emotional regulation difficulties modulate the mother–infant behavioral synchrony? Infancy, 27(3):582-608

[MW08] Brian Marcus and Susan Williams. Symbolic dynamics. *Scholarpedia*, 3:2923, 2008.

PYTHON MODULE INDEX

a

algorithmic, 9

d

discretize, ??

g

generator, 11

i

information, ??
iosymb, ??

s

sequence, 1
stochastic, ??

v

viz, ??