**Measuring rhythmic complexity: A primer to quantify and compare temporal structure in speech, movement, and animal vocalizations**

**Name Surname, Name Surname**

**Supplementary materials**

This document describes the code that was written for the article “Measuring rhythmic complexity: A primer to quantify and compare temporal structure in speech, movement, and animal vocalizations. The code was used to carry out all analyses and produce most of the figures. It can be obtained online as supplementary material. With the help of this document the readers will be able to carry out the rhythmic analyses described in the article on their own.

The code was written in Mathworks Matlab. It was tested in Matlab versions 2012b, 2015b and 2016b, but should run in all modern Matlab versions. If you do not have access to Matlab, you can run the code in GNU Octave, a free and open source alternative that is mostly compatible with Matlab's syntax. Octave is available for Windows, OSX and Linux from this address: https://www.gnu.org/software/octave/download.html. The code was tested in Octave versions 3.8.1 and 4.0.3. For the full code to run in Octave the packages 'control', 'signal', 'statistics' and 'io' are needed. These can be downloaded from http://octave.sourceforge.net/packages.php. On Windows they should come pre-installed with Octave. To load all installed packages type the following in the command window after starting Octave (omitting the ‘>>’):

>> pkg load all

The code consists of several functions (Table S1), each contained in a separate text file with the extension '.m'. To access these functions, make sure that either the files are in your current folder (browse to the folder within Matlab/Octave in the left hand panel), or the folder containing the files is in your search path (type e.g. "addpath('C:/exampleFolder')" in the command window). You can call these functions from within the Matlab/Octave command window by typing the function name, followed by one or several arguments in parentheses, e.g.:

>> plotCorr(pattern, 0, 1000)

Multiple arguments are separated by commas. Most functions require as an argument either an array containing timepoints of events (e.g. 2,4,6,9,10, the "pattern"), or an array of inter-onset intervals, i.e. the time intervals between successive events (e.g.2,2,3,1, the "ioi"). Both these arrays can be created in one of three ways:

**1. Typing in a pattern at the command line.** To create a pattern from scratch, assign a comma-separated list of numbers in square brackets to a new array. The array name can be any string of letters (a-z), numbers (0-9) and underscores (\_). They following example creates an array named "examplePattern" with a pattern consisting of events at 2, 4, 6 and 9 seconds:

>> examplePattern = [2, 4, 6, 9]

The corresponding array of IOIs would consist of the numbers 2 (difference between the second and the first event), 2 (difference between the third and the second event), and 3 (difference between the fourth and the third event):

>> exampleIOIs = [2, 2, 3];

Adding a semicolon (;) to the end of a line executes the command like normal, but suppresses the output.

**2. Generating a pattern procedurally.** The patterns that served as examples in the article were generated by the function generatePattern(). This function takes a number of arguments that affect the properties of the resulting pattern. It returns an array of events ("pattern") and an array of inter-onset intervals ("ioi"). To view a description of a function and its arguments and return values, the reader should type either "help" or "doc" followed by the function name in the command window, e.g.:

>> doc generatePattern

This prints the following text:

[pattern,iois] = generatePattern(n,nRep,type,stdev,seed)

generates random, isochronous, or one of three types of non-isochronous rhythmic patterns.

n: number of events (must be divisible by nRep wtithout remainder)

nRep: number of repetitions of the rhythmic pattern

type: either 'random', 'isochronous', 'rhythmic', 'stress' or 'mora'

stdev: standard deviation for gaussian noise

seed: seed for the random number generator

returns a 1-dimensional array of timestamps of events (pattern) and a

1-dimensional array of the corresponding inter-onset intervals (ioi).

The different pattern types (random, isochronous, rhythmic, stress and mora) follow certain rules as described in the main article. The stress pattern for example always produces clusters of four events that have a total duration of 8 seconds. The duration of the events within a cluster, however, is determined by a random number generator. The random number generator can be initialized into a certain state by the argument seed. This allows to reproduce a specific pattern. Calling the function with a seed value of 8 (see example below) will produce the same pattern each time the function is called, if all other arguments are the same. Calling the function with a different seed value will produce a different pattern. Setting the seed value to zero will result in a different pattern each time the function is called. The random number generator also affects the Gaussian noise added to the event timing. Isochronous patterns created with different seed values, for example, will therefore produce slightly different patterns, as the jitter of the events is slightly different. Readers are encouraged to experiment with different randomized patterns to see their effect on the different plots, for example how different permutations of a rhythmic pattern can lead to mirrored phase portraits.

The following command creates a 'rhythmic' pattern (and corresponding list of IOIs) similar to that used in the article, that has 24 events, 4 repetitions of a sub-pattern, an average inter-onset interval of 2s and Gaussian noise added with a standard deviation of 0.04:

>> [patternR, ioiR] = generatePattern(24, 4, 'rhythmic', 0.04, 8);

When a function has multiple return values, they are assigned to a comma-separated list of variables (in this case our two arrays) in square brackets. Again, the names of the variables can be any alphanumeric string.

**3. Reading a pattern from an Excel table.** The five example patterns from the article are included with the code as Excel table files (e.g. example\_pattern\_isochronous.xls). You can read these, or your own data, using the function readPatternFromXls(). Tables must contain the time values of several events either in a single row or a single column. The function takes the filename of the table as an argument and returns arrays for the pattern and the IOIs. If the table file is in your current folder or in a folder in the search path you can supply just its filename in single quotes (e.g. 'example\_pattern\_isochronous.xls'). Otherwise "filename" must contain the whole path to the file (e.g. 'C:/exampleFolder/example\_pattern\_isochronous.xls'). The following command reads the values of the mora sequence from the excel file:

>> [myPattern, myIOIs] = readPatternFromXls('example\_pattern\_mora.xls');

**4. Plotting and analyzing patterns.** The function plotAllSingle() calls several of the other functions and in doing so creates separate figures for all the single pattern analyses presented in the article. It takes as arguments the arrays "pattern" and "ioi" that you created through one of the three methods described above. The function also prints the Kolmogorov-Smirnov D (K-S D) and the normalized pairwise variability index (nPVI) to the command window. In addition, it returns a structure that contains the outputs of some of the functions called by plotAllSingle(), which can be used for further analyses. Type "doc plotAllSingle" or open the file "plotAllSingle.m" to view a list of all outputs contained in the results structure.

>> myResults = plotAll(myPattern, myIOIs)

To access any of the outputs in the results structure, type the name of your results structure, followed by a period and the name of the output variable you are interested in, for example:

>> myResults.nPVI

All of the analysis functions can also be called separately. Each function takes either the pattern or the IOIs as an argument. The functions that transform the pattern to a time series – plotCorr() and plotFFT() – additionally take the temporal resolution of the time series as an argument. These functions transform the pattern into a time series by creating an array with a certain temporal resolution (e.g. one value per millisecond). This array has the value 1 at at the time points of the events and the value 0 elsewhere. For the article a temporal resolution value of 1000 (time points per second) was used and the plotAllSingle() function uses 1000 as a default. Readers who wish to change this value should edit the appropriate line in the plotAllSingle.m file.

To only plot the autocorrelation function for the pattern "myPattern", type:

>> plotCorr(myPattern, 0, 1000);

This function takes either one or two patterns as arguments. If the user inputs a single pattern, this pattern will be autocorrelated. In this case set the second argument to zero. The correlation function can also be used to cross-correlate two different patterns. To do this, readers should supply a different pattern as the second argument, for instance:

>> plotCorr(patternOne, patternTwo, 1000);

The function plotRose() creates a circular histogram of deviations of a pattern from a previously defined strictly isochronous pattern. The following commands create a rhythmic pattern and an isochronous pattern without Gaussian noise added and creates a rose plot:

>> [patternR,ioiR] = generatePattern(24, 2, 4, 'rhythmic', 0.04, 0);

>> [patternI,ioiI] = generatePattern(24, 2, 0, 'isochronous', 0, 0);

>> plotRose(patternR, patternI);

To view any function, readers can open it in the Matlab or Octave editor or any other text editor. There, they will find a description of the function and its usage, as well as comments explaining each step (lines beginning with %). Users can change all functions according to their needs. Some internal parameters of the functions, for example, depend somewhat on the magnitude of your IOIs, like the standard deviation and width of the normal distribution in the plotCorr() function, or the threshold for the recurrence plots in plotRecurrence(). Other values that one might want to adjust include the width of the histogram bins in plotHistogram() and the freuquency limits in plotFFT() and plotGAT().

Table S1. Overview of all functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Arguments** | **Returns** | **Description** |
| generatePattern | n, nRep, type,  stdev, seed | pattern, ioi | Generates one of five different types of patterns (random, isochronous, rhythmic, stress, mora) and inter-onset intervals (IOI). |
| getKolmogorovSmirnovD | ioi | d | Calculates the Kolmogorov-Smirnov D (K-S D) from a list of IOIs. |
| getNPVI | ioi | nPVI | Calculates the normalized pairwise variability index (nPVI). |
| getPhaseFromPatterns | pattern, patternIso | phaseRad | Calculates the phase (in radians) of each event in a pattern, relative to a second, isochronous pattern. This function gets internally called by plotRose(). |
| getTimeSeriesFromPattern | pattern, resolution | timeSeries | Constructs a time series from a pattern. This func-tion gets internally called by plotCorr() and plotFFT(). |
| plotAllSingle | pattern, ioi | results | Calls all plot functions consecutively for a pattern and prints the K-S D and nPVI. |
| plotCorr | patternOne, patternTwo,  resolution | correlation, sigConvOne,  sigConvTwo | Performs and plots either autocorrelation (if patternTwo = 0) or crosscorrelation. The pattern is first convoluted with a Gauss curve. |
| plotFFT | pattern, resolution | freq, power, timeSeries | Calculates the fast Fourier transform (FFT) of a pattern and plots the power spectrum. |
| plotGAT | pattern | freq, frmsd | Performs the pulse generate-and-test (GAT) method and plots the frmsd for the tested frequencies (default: 0.2-5Hz; set within the function). |
| plotHistogram | ioi | - | Plots a histogram of IOIs. (default bin width: 0.075) |
| plotPattern | pattern | - | Plots the events of a pattern as vertical lines on a timeline. |
| plotPhasePortrait | ioi | - | Creates a phase portrait of IOIs. |
| plotRecurrence | ioi | - | Creates a recurrence plot of IOIs. (default threshold: 0.3) |
| plotRose | pattern,  patternIso | phaseRad | Creates a rose plot (circular histogram) of phases of a pattern relative to a pre-determined strictly isochronous pattern. |
| readPatternFromXls | filename | pattern, ioi | Reads a pattern from an Excel table file (\*.xls/\*.xlsx). |