
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

**TUTORIAL**

# COUSIN (COdon Usage Similarity INdex): A normalized measure of Codon Usage Preferences.

*Authors:*
JÉRÔME BOURRET
SAMUEL ALIZON
IGNACIO G. BRAVO

*Version :*
1.0 (May 13, 2019)

May 13, 2019

# Contents

# 1 Prerequisites

- ***Web-application*** `COUSIN` :
  - An up-to-date web browser. The tool has been tested and works on the following browsers :
    - **FireFox**
    - **Chrome**
    - **Chromium**
    - **Opera**
    - **Safari**
    - **...**

    While tests have been launched on **Edge** and **Internet Explorer**, bugs could still be seen on these browsers. It is not advised to use these browsers for COUSIN.
  - to visualize graphical outputs online, `Adobe Flash Player` may be needed.
- ***Command-line*** `COUSIN` :
  - A UNIX Operating System (OS or Linux).
  - A Python3+ version of Python with the following modules :
    - pandas
    - numpy
    - scipy
    - sklearn
    - pyclustering
    - matplotlib
    - seaborn

# 2 Introduction

`COUSIN` is an informatic tool developed to calculate the Codon Usage Preferences (CUPrefs) — known as well as Codon Usage Bias (CUB) — of CDSs thanks to the use of a large set of existing indexes including the new COUSIN index. In addition, this tool performs many features such as statistical analysis of CUPrefs, sequence optimization, clustering of queries following criterias, ...

### 2.1 basic architecture of COUSIN

COUSIN performs a CUPrefs analysis on query sequences using either a Null hypothesis (*e.g* ENC index) or a reference (*e.g* COUSIN, CAI indexes). A COUSIN analysis can be separated in 2 steps : A first one where basic calculations are performed and a second one where a non-mandatory option is performed following user's will. These options will be discussed in the section 2.3.2 of this tutorial. At the end of the analysis, COUSIN displays graphical and textual results.

### 2.2 Implemented measures

- CUB measures
  - COUSIN
  - CAI
  - ENC
  - FOP
  - RSCU
  - CBI
  - ICDI
  - $\chi^2$
  - SCUO
  - RSCU values (non mandatory - see parameter "-J")
  - codons and dicodons occurrences and frequencies. (non mandatory - see parameter "-J")
- Amino-acid composition measures
  - GRAVY
  - AROMA
  - Euclidian distances between query and reference

### 2.3 COUSIN steps

The COUSIN manual is accessible by entering the following command :

```
$ cousin.py -h
```

### 2.3.1 Routine tasks (basic calculation)

COUSIN firstly performs a routine step. Please note that this step is launched regardless of the option chosen by the user (section 2.3.2). It consists of the following calculations:

- ATGC composition of the sequence.
- Length of the sequence.
- The calculation of the scores described in the previous section

In addition, COUSIN can perform a simulation that determines if the score of a query is statistically close to the reference. To do so, COUSIN create 500 sequences following a "random-guided" selection of amino acids and codons (Puigbò *et al.*, 2007). The principle of such task is to create sequences with a CUPref close to the one of the reference dataset. Once it is done, the CUPref scores are calculated for each simulated sequence. Then, COUSIN determines the 95% and 99% confidence intervals of these scores and compare them to the score of each query. At the end of this step, COUSIN displays a graphical output which represents the range of values obtained during the simulation (Figure **??**). This simulation step is launched with the parameter $-S$ (section 3).

To launch the routine step without any further option, use the following command line :

```
$ cousin.py calculation -s <input file sequence> -c <codon usage table> -g <genetic code>
    -b <optimal codons file> -S <routine_simulation> -X <modified_COUSIN> -J <vectors> -o
    <output file>
```

Table 1: Definition of the parameters used in the "basic calculation" step of the COUSIN tool

| **Parameters** | **Description** |
|---|---|
| **calculation** | keyword to call the function |
| $-s$ | Path to the input file containing the queries. Must be in a DNA FASTA format. |
| $-c$ | Path to the input file containing the reference. Must be in a kazusa-like format. |
| $-g$ | The translation table of both the query and the reference (standard = 1). |
| $-b$ | A list of optimal codons. Must be in a "comma-separated" (csv) format. |
| $-S$ | Perform simulation step (Yes \| No) (standard = No) |
| $-X$ | Modify the COUSIN calculation by deleting all amino acid with little CUB (Yes \| No) |
| $-J$ | Create a vector of occurences, frequencies and RSCU values for each query (Yes \| No) |
| $-o$ | output folder of your analysis (directly linked to where the code is launched.) |

### 2.3.2 Options

COUSIN displays a broad range of specific functions that can be launched regarding user's choice. These options, and their usage, are described in this section.

#### Simulation analysis

This simulation is close to the one described in the section 2.3.1. Here, the simulated sequences are created following two criterias:

- Same length than the query // Same amino acid composition and CUPrefs than the reference.
- Same length AND the same amino acid composition than the query // same CUPrefs than the reference.

The CUPrefs scores of these two datasets are then calculated and the 95% and 99% intervals are estimated. Each query score is then compared to the two sets of intervals. If it is contained in one dataset interval threshold and not in the other, this would suggest that amino acid composition significantly impact the CUPrefs score of the query. In addition, a Wilcoxon-Mann-Whitney U-test is performed on the two simulated datasets to check whether or not they have the same score distribution.

The simulation function is called as following :

```
$ cousin.py simulation_analysis -s <input file sequence> -c <codon usage table> -g <genetic
    code> -b <optimal codons file> -S <routine_simulation> -X <modified_COUSIN> -J
    <vectors> -n <simulation_number> -o <output file>
```

Table 2: Definition of the parameters used in the "simulation analysis" step of the COUSIN tool

| Parameters | Description |
|---|---|
| **simulation_analysis** | keyword to call the function |
| $-s$ | Path to the input file containing the queries. Must be in a DNA FASTA format. |
| $-c$ | Path to the input file containing the reference. Must be in a kazusa-like format. |
| $-g$ | The translation table of both the query and the reference (standard = 1). |
| $-b$ | A list of optimal codons. Must be in a "comma-separated" (csv) format. |
| $-S$ | Perform simulation step (Yes \| No) (standard = No) |
| $-X$ | Modify the COUSIN calculation by deleting all amino acid with little CUB (Yes \| No) |
| $-J$ | Create a vector of occurences, frequencies and RSCU value for each query (Yes \| No) |
| $-n$ | The number of simulation wanted on each query. |
| $-o$ | output folder of your analysis. Is directly linked where the code is launched. |

**Pattern analysis**

The pattern analysis aims to gather and treat query sequences following their headers. In some cases, one may want to analyze, as an example, all the CDSs of a genome containing the pattern "CDS_1". The CUPref of all sequences containing this pattern in their header is then calculated altogether, and a graph showing the range of values is displayed. The pattern analysis function can be called with the following command :

```
$ cousin.py pattern_analysis -s <input file sequence> -c <codon usage table> -g <genetic
    code> -b <optimal codons file> -S <routine_simulation> -X <modified_COUSIN> -J
    <vectors> -z <input file patterns> -o <output file>
```

Table 3: Definition of the parameters used in the "pattern analysis" step of the COUSIN tool

| Parameters | Description |
|---|---|
| **pattern_analysis** | keyword to call the function |
| $-s$ | Path to the input file containing the queries. Must be in a DNA FASTA format. |
| $-c$ | Path to the input file containing the reference. Must be in a kazusa-like format. |
| $-g$ | The translation table of both the query and the reference (standard = 1). |
| $-b$ | A list of optimal codons. Must be in a "comma-separated" (csv) format. |
| $-S$ | Perform simulation step (Yes \| No) (standard = No) |
| $-X$ | Modify the COUSIN calculation by deleting all amino acid with little CUB (Yes \| No) |
| $-J$ | Create a vector of occurences, frequencies and RSCU value for each query (Yes \| No) |
| $-z$ | The list of header patterns (csv/tsv) format. |
| $-o$ | output folder of your analysis. Is directly linked where the code is launched. |

Please note that if no pattern data is given to COUSIN, a pattern analysis will be done on all queries. Thus, a unique group containing all headers will be created.

**Clustering analysis**

A clustering analysis can be done on a set of variables obtained after a basic calculation COUSIN step. Firstly, a k-mean (or x-mean) analysis is performed on the query sequences with the set of variables chosen for the clustering analysis. Then, a PCA on the same variables is performed. A set of graphs showing the first four PC axes is given with the projection of the clustering performed. In parallel, a hierarchical clustering is performed on the selected variables. The dendrogram resulting from this analysis is given in a newick format. As in the pattern analysis, a pattern file can be given to COUSIN in order to propose an "expected" clustering that will replace the k-means clustering.

```
$ cousin.py clustering_analysis -s <input file sequence> -c <codon usage table> -g <genetic
    code> -b <optimal codons file> -S <routine_simulation> -X <modified_COUSIN> -J
    <vectors> -w < selected variables> -n <number_of_clusters> -o <output file>
```

Table 4: Definition of the parameters used in the "clustering analysis" step of the COUSIN tool

| Parameters | Description |
|---|---|
| **clustering_analysis** | keyword to call the function |
| $-s$ | Path to the input file containing the queries. Must be in a DNA FASTA format. |
| $-c$ | Path to the input file containing the reference. Must be in a kazusa-like format. |
| $-g$ | The translation table of both the query and the reference (standard = 1). |
| $-b$ | A list of optimal codons. Must be in a "comma-separated" (csv) format. |
| $-S$ | Perform simulation step (Yes \| No) (standard = No) |
| $-X$ | Modify the COUSIN calculation by deleting all amino acid with little CUB (Yes \| No) |
| $-J$ | Create a vector of occurences, frequencies and RSCU value for each query (Yes \| No) |
| $-w$ | The set of variables used for the clustering (see section 3 for more details) |
| $-n$ | The number of clusters wanted for the k-means analysis (if 0 or 1 ==> x-means).<br>If a pattern file is provided with this parameter, the clusters will be formed from this file. |
| $-o$ | output folder of your analysis. Is directly linked where the code is launched. |

**Optimization**

Among other tools, one of the most implemented and used function is the optimization of a sequence (Puigbò *et al.*, 2007; Grote *et al.*, 2005; Supek and Vlahovicek, 2004). This function, which find primarily an interest in molecular engineering, modifies the CUPrefs of a query to make it follow a specific one (here the reference one). With COUSIN, different kind of optimizations can be performed :

- A "Random" optimization, where for each amino acid of the sequence a codon is randomly selected among its synonymous. COUSIN is able to perform a random selection among either each synonymous codon of an amino acid or the ones maximizing GC or AT content.
- A "Random Guided" optimization, where the codons are selected following their frequencies among the reference. Once again, COUSIN is able to perform a random selection among either each synonymous codon of an amino acid or the ones maximizing GC or AT content
- A "One amino acid — One codon" optimization, where each amino acid is represented by a unique codon (the one with the greatest or least frequency among the reference).

The optimization function can be called with the following command-line :

```
$ cousin.py optimization -s <input file sequence> -c <codon usage table> -g <genetic code>
    -b <optimal codons file> -S <routine_simulation> -X <modified_COUSIN> -J <vectors> -k
    <sequence type> -n <number of optimizations> -x <optimization type> -r < reverse
    frequencies> -o <output file>
```

Table 5: Definition of the parameters used in the "optimization" step of the COUSIN tool

| Parameters | Description |
|---|---|
| **optimization** | keyword to call the function |
| $-s$ | Path to the input file containing the queries. Must be in a DNA/AA FASTA format. |
| $-c$ | Path to the input file containing the reference. Must be in a kazusa-like format. |
| $-g$ | The translation table of both the query and the reference (standard = 1). |
| $-b$ | A list of optimal codons. Must be in a "comma-separated" (csv) format. |
| $-S$ | Perform simulation step (Yes \| No) (standard = No) |
| $-X$ | Modify the COUSIN calculation by deleting all amino acid with little CUB (Yes \| No) |
| $-J$ | Create a vector of occurences, frequencies and RSCU value for each query (Yes \| No) |
| $-k$ | the nature of your dataset (DNA \| AA) |
| $-n$ | Number of optimizations. |
| $-x$ | the optimization wanted (see below for more details). |
| $-r$ | Reverse the frequencies of synonymous codons inside the reference (0 \| 1). |
| $-o$ | output folder of your analysis. Is directly linked where the code is launched. |

**Creation of a Codon Usage Table**

This step consists in the creation of a codon and of a dicodon (pairs of codons) usage tables in a kazusa-like format. The tables are created using the query dataset. A routine step as described in section 2.3.1 is then performed on the CDSs that created the codon usage tables.

The command-line used to call this function is the following one :

```
$ cousin.py create_table -s <input file sequence> -g <genetic code> -b <optimal codons
    file> -S <routine_simulation> -X <modified_COUSIN> -J <vectors> -o <output file>
```

Table 6: Definition of the parameters used in the "create table" step of the COUSIN tool

| Parameters | Description |
|---|---|
| **create_table** | keyword to call the function |
| $-s$ | Path to the input file containing the queries. Must be in a DNA FASTA format. |
| $-g$ | The translation table of both the query and the reference (standard = 1). |
| $-b$ | A list of optimal codons. Must be in a "comma-separated" (csv) format. |
| $-S$ | Perform simulation step (Yes \| No) (standard = No) |
| $-X$ | Modify the COUSIN calculation by deleting all amino acid with little CUB (Yes \| No) |
| $-J$ | Create a vector of occurences, frequencies and RSCU value for each query (Yes \| No) |
| $-o$ | output folder of your analysis. Is directly linked where the code is launched. |

**Comparison of two datasets**

The last step proposed by COUSIN allow the user to compare two dataset by calculating the euclidian distances of each codons and amino acids frequencies between codon usage tables and / or sequences. The main goal of this step is to denote quickly differences between two (or more) datasets.

This function is called as following :

```
$ cousin.py compare_data -c <first_dataset> -d <second_dataset> -C <nature_of_datasets> -o
    <output file>
```

Table 7: Definition of the parameters used in the "data comparison" step of the COUSIN tool

| Parameters | Description |
|---|---|
| **data_comparison** | keyword to call the function |
| $-c$ | Path to the first dataset that will be compared (DNA Fasta file or Codon Usage Table). |
| $-d$ | Path to the second dataset that will be compared (DNA Fasta file or Codon Usage Table). |
| $-C$ | The nature of the two dataset under comparison (see below for more details). |
| $-o$ | output folder of your analysis. Is directly linked where the code is launched. |

# 3 Definition of parameters and common usage

## 3.1 All functions

### 3.1.1 parameter −s

⚠ **MANDATORY** ⚠

The path access and the filename of the file containing the queries. Here, only DNA or AA (only for optimization) FASTA sequences are accepted. Each sequence must have a unique FASTA header.

Example :

```
$ cousin.py calculation -s path/to/file.fasta ...
```

### 3.1.2 parameter −c (all functions except create_table)

⚠ **MANDATORY** ⚠

The path access and the filename of a file containing the codon usage table you want to use for this analysis. The only accepted format is the "kazusa-like" one (`http://www.kazusa.or.jp/codon/`). COUSIN can take

as an input codon usage tables with or without amino acids, but not in a way like presented in the Codon-Frequency output of GCG.

Example :

```
$ cousin.py simulation_analysis ... -c /path/to/file.txt ...
```

#### 3.1.3 parameter –g

**⚠ IF NOT SPECIFIED : VALUE = 1 ⚠**

The genetic code used for your analysis. For now, you can choose between 13 genetic codes. Please note that with `COUSIN`, the names of these codes follow the NCBI translation tables nomenclature :

- 1 : The Standard Code
- 2 : The Vertebrate Mitochondrial Code
- 3 : The Yeast Mitochondrial Code
- 4 : The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
- 5 : The Invertebrate Mitochondrial Code
- 6 : The Ciliate, Dasycladacean and Hexamita Nuclear Code
- 9 : The Echinoderm Mitochondrial Code
- 10 : The Euplotid Nuclear Code
- 11 : The Bacterial "Code"
- 12 : The Alternative Yeast Nuclear Code
- 13 : The Ascidian Mitochondrial Code
- 14 : The Flatworm Mitochondrial Code
- 15 : The Blepharisma Nuclear Code

Example with the standard genetic code :

```
$ cousin.py create_table ... -g 1 ...
```

#### 3.1.4 parameter −b

**⚠ IF NOT SPECIFIED : DETECTION OF OPTIMAL CODONS ⚠**

The path access and the filename of a file containing informations on optimal codons. This file is used for the FOP and CBI methods. The data must be stocked on a file in a unique line where each codon is separated by a comma ",".

Example :

```
$ cousin.py ... -b path/to/file.csv
```

If no file is specified, optimal codons are automatically detected from the reference table. In this case, the most frequent synonymous codons of each amino acid are selected.

#### 3.1.5 parameter −S

**⚠ IF NOT SPECIFIED : VALUE = NO ⚠**

Allow `COUSIN` to perform ("Yes") or not ("No") the "basic simulation step as described in the section 2.3.1.

Example :

```
$ cousin.py clustering_analysis ... -S path/to/output_dir
```

#### 3.1.6 parameter −X

**⚠ IF NOT SPECIFIED : VALUE = NO ⚠**

In some reference data, the synonymous codons of an amino acid could be used almost identically. In this case, the calculation of COUSIN can be modified to either take in account these amino acids in the final score ("No" for "No modification in the calculation") or not ("Yes"). Indeed, since these amino acids display little to no differences from the Null Hypothesis, they could alter slightly the final result.

Example :

```
$ cousin.py calculation ... -X No
```

#### 3.1.7 parameter −J

**⚠ IF NOT SPECIFIED : VALUE = NO ⚠**

Create ("Yes") or not ("No") a file for each query containing vectors of codons and dicodons occurences, frequencies along with RSCU values.

Example :

```
$ cousin.py create_table ... -J No
```

#### 3.1.8 parameter −o

**⚠ MANDATORY ⚠**

The path and the name of the output folder where all of the results will be stocked.

Example :

```
$ cousin.py compare_data ... -o path/to/output_dir
```

### 3.2 Simulation analysis option

#### 3.2.1 parameter −n

**⚠ IF NOT SPECIFIED : VALUE = 500 ⚠**

The number of simulations performed during the simulation analysis step.

Example :

```
$ cousin.py ... -n 100 ...
```

### 3.3 Pattern analysis option

#### 3.3.1 parameter −z

**⚠ IF NOT SPECIFIED : GLOBAL ANALYSIS ⚠**

The path access and the filename of a file containing the patterns. The file must be organized as following : each pattern must be separated by a comma ",", a semi-colon ";" or a tabulation. Group of patterns can be used in this step. Theses groups defines a way to aggregate data following their headers :

1. Let's suppose two groups of patterns : [Group 1 : CDS_1, CDS_2, CDS_3] [Group 2 : CDS_a, CDS_b, CDS_c]
2. For each header containing the pattern "CDS_1", `COUSIN` will perform a classic pattern analysis with the following patterns : "CDS_a", "CDS_b" and "CDS_c". The same task will be performed with the patterns "CDS_2" and "CDS_3".
3. The same step is repeated for the headers containing the patterns "CDS_a", "CDS_b" and "CDS_c". Here, `COUSIN` performs a classic pattern analysis with the following patterns : "CDS_1", "CDS_2" and "CDS_3" for all queries with "CDS_a", "CDS_b" and "CDS_c" in their headers.

If no file is specified, all headers will be analyzed altogether, and the software will output a unique graph : the density curve of all queries scores.

Example :

```
$ cousin.py ... -z path/to/file.csv ...
```

## 3.4 Clustering analysis

### 3.4.1 parameter −n

**⚠ If not specified : value = 0 ⚠**

The number of clusters set for the analysis. Please note that the number of clusters must be comprised between 0 (x-means analysis) and the total number of queries minus 1. This parameter can also accept a pattern file, similar to the one described in the "pattern analysis" parameters, as a set of predefined clusters.

Example :

```
$ cousin.py ... -n 4 ...
```

### 3.4.2 parameter −w

**⚠ If not specified : value = "basic**$_{a}nalysis"B$

The set of variables used for the clustering analysis.

- ***"all_variables"***: All the variables found in a basic COUSIN calculation (A-T-G-C content, CUPref scores, ...).
- ***"codons"***: the frequencies of the 59 synonymous codons.
- ***"GC_content"***: GC-content (overall, first, second and third base)
- ***"ATGC_content"***: ATGC-content (third base)
- ***"all_indexes"***: all the CUPrefs and amino acid composition indexes used during a COUSIN analysis.
- ***"basic_analysis"***: COUSIN, CAI and GC content.

Example :

```
$ cousin.py ... -w codons ...
```

## 3.5 Optimization option

### 3.5.1 parameter −k

**⚠ If not specified : value = "DNA" ⚠**

The sequence type of your input. Type « DNA » if your sequences are in a DNA format or « AA » if they are related to proteins. Naturally, it is not possible to have both AA AND DNA sequences in your input file.

Example :

```
$ cousin.py ... -k DNA ...
```

### 3.5.2 parameter −r

**⚠ IF NOT SPECIFIED : VALUE = 0 ⚠**

"r" is used to create the reversal frequencies of codons in the codon usage table for your analysis. When this parameter has a "0" value, the codon usage table is used in its basic format, but when "r" is equal to "1", the reverse frequencies of the same codon usage table are used for the analysis.

Example :

```
$ cousin.py ... -r 0 ...
```

#### 3.5.3 parameter −x

**⚠ IF NOT SPECIFIED : VALUE = 0 ⚠**

the type of simulation you want to make. The choices can be made as following :

- 0 : Random simulation
- 1 : Random guided simulation
- 2 : Random with max GC content
- 3 : Random guided with max GC content
- 4 : Random with max AT content
- 5 : Random guided with max AT content
- 6 : One amino acid - One codon simulation with the most frequent codons
- 7 : One amino acid - One codon simulation with the least present codons

Example :

```
$ cousin.py ... -x 1 ...
```

#### 3.5.4 parameter −n (optimization function)

**⚠ IF NOT SPECIFIED : VALUE = 1 ⚠**

The parameter "n" describes the number of simulations you want to make. If this parameter has a value equal to 0, no simulation will be done.

Please note that if your sequences are in a AA format, no further analysis will be done if you give a 0 value to the "n" parameter. Indeed, at least one simulation is needed to transform an AA sequence in a DNA one.

Example :

```
$ cousin.py ... -n 100 ...
```

### 3.6 Data comparison option

#### 3.6.1 Parameters −c and −d

**⚠ MANDATORY ⚠**

The path access and the filename of a file containing either a Codon Usage Table or a FASTA file containing DNA sequences.

Example :

```
$ cousin.py ... -c path/to/codon_usage_table -d path/to/FASTA_sequences ...
```

#### 3.6.2 Parameters −C

⚠ **MANDATORY** ⚠

This parameters gives an indication about the nature of the datasets. 4 values can be given :

- CVC : CU table against CU table
- CVS : CU table against Fasta file
- SVC : Fasta file against CU table
- SVS : Fasta file against Fasta file

Example :

```
$ cousin.py ... -c path/to/codon_usage_table -d path/to/FASTA_sequences -C CVS...
```

### 3.7 Creation of a Codon Usage Table option

The launching of the creation of a Codon Usage Table does not require specific parameters. Moreover, the parameter "-c" is also not required for this step.

## References

Grote, A. *et al.* (2005). JCat: a novel tool to adapt codon usage of a target gene to its potential expression host. *Nucleic Acids Research*, **33**(suppl_2), W526–W531.

Puigbò, P. *et al.* (2007). OPTIMIZER: a web server for optimizing the codon usage of DNA sequences. *Nucleic Acids Research*, **35**(suppl_2), W126–W131.

Supek, F. and Vlahovicek, K. (2004). INCA: synonymous codon usage analysis and clustering by means of self-organizing map. *Bioinformatics (Oxford, England)*, **20**(14), 2329–2330.