coinflip

Release 0.0.5

Oct 29, 2020

Contents

CHAPTER 1

coinflip

Randomness testing for humans

coinflip aims to implement the tests recommended by NIST SP800-22 to check random number generators for randomness. A user-friendly command-line interface provided allows you to `load`, `run` and `report` on your data in a step-by-step fashion.

`coinflip.randtests` acts as the public API for notebook users and developers to use the randomness tests directly. The tests are implemented as general solutions, meaning they accept basically any binary sequence you throw at them!

1.1 Setup

coinflip is currently in rapid development, so we recommend building from source.

```
$ pip install git+https://github.com/Honno/coinflip
```

If that means nothing to you, no fret! Please continue reading the instructions below.

1.1.1 Install Python 3.7+

Cross-platform installation instructions for Python are available at realpython.com/installing-python/.

Note `coinflip` only works on **Python 3.7 or above**. Make sure you have Python 3.7 (or higher) by checking the version of your installation:

```
$ python --version
Python 3.7.X
```

1.1.2 Clone repository

You can clone the source code via Git:

```
$ git clone https://github.com/Honno/coinflip
```

1.1.3 Install coinflip

Enter the directory *coinflip* is downloaded to:

```
$ cd coinflip
```

You can install *coinflip* via the `pip` module:

```
$ pip install -e .
```

pip is the standard package manager for Python, which should of installed automatically when installing Python 3.7+.

1.1.4 Trial run

Try running the randomness tests on a generated binary sequence:

```
$ coinflip example-run
```

If the command `coinflip` is "not found", you may need to add your local binaries folder to your shell's path. For example, in bash you would do the following:

```
$ echo "export PATH=~/.local/bin:$PATH" >> ~/.bash_profile
$ source ~/.bash_profile
```

In the worst case, you can execute commands via `python -m`:

```
$ python -m coinflip example-run
```

1.2 Quick start

Output of random number generators can be parsed and serialised into a test-ready format via the `load` command:

```
$ coinflip load DATA
Store name to be encoded as store_<timestamp>
Data stored successfully!
...
```

`DATA` is the path to newline-delimited text file that contains a binary sequence. An example file to use is available on my gist.

Randomness tests can then be ran over the store's data via the `run` command. You should be prompted by a "No STORE argument provided" message, where `coinflip` will assume you want to run the tests over the data you just loaded—type `y` and hit enter.

```
$ coinflip run
No STORE argument provided
  The most recent STORE to be initialised is 'store_<timestamp>'
  Pass it as the STORE argument? [y/N]: y
...
```

Output should comprise of the example sequence, test-specific summaries, and a final overall summary table.

CHAPTER 2

Commands

2.1 coinflip

Randomness tests for RNG output.

Output of random number generators can be parsed and serialised into a test-ready format via the load command. The data is saved in a folder, which coinflip refers to as a "store". This store is located in the local data directory, but can be easily accessed via the store's name in coinflip commands.

Randomness tests can then be ran over the store's data via the run command. Rich documents explaining the test results can be produced via the report command.

```
coinflip [OPTIONS] COMMAND [ARGS]...
```

Commands

cat
Print contents of data in STORE.

example-run
Run randomness tests on example data.

load
Loads DATA into a store.

local-run
Run randomness tests on DATA directly.

ls
List all stores.

report
Generate html report from test results in...

rm
Delete STORE.

rm-all
Delete all stores.

run
Run randomness tests on data in STORE.

2.2 coinflip load

Loads DATA into a store.

DATA is a newline-delimited text file which contains output of a random number generator. The contents are parsed, serialised and saved in local data.

The stored data can then be applied the randomness tests via the run command, where the results of which are also saved.

```
coinflip load [OPTIONS] DATA
```

Options

-n, --name <name>
Specify name of the store.

-d, --dtype <dtype>
Specify data type of the data.

Options bool|byte|short|int|long|float|double

-o, --overwrite
Overwrite existing store with same name.

Arguments

DATA
Required argument

2.3 coinflip cat

Print contents of data in STORE.

```
coinflip cat [OPTIONS] STORE
```

Arguments

STORE
Optional argument

2.4 coinflip rm

Delete STORE.

```
coinflip rm [OPTIONS] STORE
```

Arguments

STORE
Required argument

2.5 coinflip rm-all

Delete all stores.

```
coinflip rm-all [OPTIONS]
```

2.6 coinflip run

Run randomness tests on data in STORE.

Results of the tests run are saved in STORE, which can be compiled into a rich document via the report command.

```
coinflip run [OPTIONS] STORE
```

Options

-t, --test <test>
Specify single test to run on data.

Options monobit|frequency_within_block|runs|longest_runs|binary_matrix_rank|spectral|non_overlapping_template_match

Arguments

STORE
Optional argument

2.7 coinflip example-run

Run randomness tests on example data.

```
coinflip example-run [OPTIONS]
```

Options

-e, --example <example>
Example binary output to use.

Options python|primes

-n, --length <length>
Length of binary output.

-t, --test <test>
Specify single test to run on data.

Options monobit|frequency_within_block|runs|longest_runs|binary_matrix_rank|spectral|non_overlapping_template_match

2.8 coinflip local-run

Run randomness tests on DATA directly.

```
coinflip local-run [OPTIONS] DATA
```

Options

-d, --dtype <dtype>
Specify data type of the data.

Options bool|byte|short|int|long|float|double

-t, --test <test>
Specify single test to run on data.

Options monobit|frequency_within_block|runs|longest_runs|binary_matrix_rank|spectral|non_overlapping_template_match

Arguments

DATA
Required argument

2.9 coinflip report

Generate html report from test results in STORE.

```
coinflip report [OPTIONS] STORE
```

Options

-o, --outfile <outfile>

Arguments

STORE
Optional argument

CHAPTER 3

Reference

3.1 store

Store functionality for the CLI

Notes

A store is an abstraction for a folder in the user's local data directory which pertains to a specific dataset that comprises of RNG output. The store can subsequently store test results and report markup for said results.

exception coinflip.store.**DataParsingError**
Base class for parsing-related errors

coinflip.store.**parse_data**(*data_file*, *dtype_str=None*) → pandas.core.series.Series
Reads file containing data into a pandas Series

Reads from file containing RNG output and produces a representitive pandas Series. The appropiate dtype is inferred from the data itself, or optionally from the supplied *dtype_str*.

Parameters

- **data_file** (*file-like object*) – File containing RNG output
- **dtype_str** (*str*, optional) – String representation of desired dtype. If not supplied, it is inferred from the data.

Returns *Series* – A pandas *Series* which represents the data

Raises

- TypeNotRecognizedError – If supplied dtype_str does not recognise a dtype
- MultipleColumnsError – If inputted data contains multiple values per line
- NonBinarySequenceError – If sequence does not contain only 2 values

See also:

pandas.read_csv() The pandas method for reading *data_file*

store_data() Calls this method, and handles subsequent storage of data

exception coinflip.store.**StoreError**(*store_name*)
Base class for store-related errors

coinflip.store.**init_store**(*name=None*, *overwrite=False*)
Creates store in local data

A name supplied or generated is used to initialise a store. If supplied, the name is sanitised to remove invalid characters for filepaths. If generated, the name will be a timestamp of initialisation.

Parameters

- **name** (*str*, optional) – Desired name of the store, which will be sanitised. If not supplied, a name is generated automatically.
- **overwrite** (*boolean*, default *False*) – If a name conflicts with an existing store, this decides whether to overwrite it.

Returns

- **store_name** (*str*) – Internal name of the initialised store
- **store_path** (*Path*) – Path of the initialised store

Raises

- NameConflictError – If attempts at generating a unique name fails
- StoreExistsError – If a store of the same name exists already (and overwrite is set to *False*)
- NonBinarySequenceError – If sequence does not contain only 2 values

See also:

store_data() Parses data and calls this method, to then save data in store

coinflip.store.**store_data**(*data_file*, *name=None*, *dtype_str=None*, *overwrite=False*)
Load and parse RNG output, serialised to a local data directory

Reads from file containing RNG output and produces a representitive pandas Series. The appropiate dtype is inferred from the data itself, or optionally from the supplied *dtype_str*.

A name supplied or generated is used to initialise a store. If supplied, the name is sanitised to remove invalid characters for filepaths. If generated, the name will be a timestamp of initialisation.

The representive Series is serialised using Python's pickle module, saved in the initialised store.

The store's name is also written to a file in the user data directory, to be accessed later when identifying the last initialised store.

Parameters

- **data_file** (*file-like object*) – File containing RNG output
- **name** (*str*, optional) – Desired name of the store, which will be sanitised. If not supplied, a name is generated automatically.
- **dtype_str** (*str*, optional) – String representation of desired dtype. If not supplied, it is inferred from the data.

- **overwrite** (*bool*, default *False*) – If a name conflicts with an existing store, this decides whether to overwrite it.

Raises

- `TypeNotRecognizedError` – If supplied dtype_str does not recognise a dtype
- `MultipleColumnsError` – If inputted data contains multiple values per line
- `NameConflictError` – If attempts at generating a unique name fails
- `StoreExistsError` – If a store of the same name exists already (and overwrite is set to *False*)

See also:

parse_data() Loads and parses *data_file*

init_store() Initialises the store

find_latest_store() Accesses the name of the last initialised store

exception coinflip.store.**NoLatestStoreRecordedError**
Error for when latest store cannot be identified

coinflip.store.**find_latest_store**() → str
Find out the last initialised store

A file is kept in the root user data directory to record the last initialised store's name, which this method reads to identify the store.

Returns **store_name** (*str*) – Name of the last initialised store

Raises *NoLatestStoreRecordedError* – When no last initialised store is found

coinflip.store.**get_data**(*store_name*) → pandas.core.series.Series
Access data of a store

Parameters **store_name** (*str*) – Name of the store

Returns *Series* – A pandas *Series* which represents the data

Raises

- `StoreNotFoundError` – If requested store does not exist
- `DataNotFoundError` – If requested store has no data

coinflip.store.**drop**(*store_name*)
Remove store from local data

Parameters **store_name** (*str*) – Name of store to remove

coinflip.store.**list_stores**()
List all stores in local data

coinflip.store.**store_result**(*store_name*, *randtest_name*, *result: coinflip.randtests._result.TestResult*)
Store result of a statistical test

Parameters

- **store_name** (*str*) – Name of store to save result in
- **randtest_name** (*str*) – Name of statistical test the result came from
- **result** (*TestResult*) – Result of the statistical test

See also:

store_results() Store multiple results from multiple statistical tests

coinflip.store.**store_results**(*store_name*, *results_dict: Dict[str, coinflip.randtests._result.TestResult]*)
Store results of multiple statistical tests

Parameters

- **store_name** (*str*) – Name of store to save result in
- **results_dict** (*Dict[str, TestResult]*) – Mapping of statistical tests to their respective results

See also:

store_result() Store a single results from a single statistical test

coinflip.store.**open_results**(*store_name*)
Context manager to read/write results of a store

Parameters **store_name** (*str*) – Name of store to access results in

Yields **results** (*Dict[str, TestResult]*) – Previously stored results of statistical tests

Raises `StoreNotFoundError` – If requested store does not exist

3.2 tests_runner

Methods used to interact with the randtests subpackage.

coinflip.tests_runner.**list_tests**() → Iterator[Tuple[str, Callable]]
List all available statistical tests

Yields

- **randtest_name** (*str*) – Name of statistical test
- **randtest_func** (*Callable*) – Function object of the statistical test

exception coinflip.tests_runner.**TestNotFoundError**
Error for when a statistical test is not found

coinflip.tests_runner.**run_test**(*series: pandas.core.series.Series*, *randtest_name*, ***kwargs*) → coinflip.randtests._result.TestResult
Run a statistical test on RNG output

Parameters

- **series** (*Series*) – Output of the RNG being tested
- **randtest_name** (*str*) – Name of statistical test
- ****kwargs** – Keyword arguments to pass to statistical test

Returns **result** (*TestResult*) – Dataclass that contains the test's statistic and p-value as well as other relevant information gathered.

Raises

- *TestNotFoundError* – If *randtest_name* does not match any available statistical tests
- `TestError` – Errors raised when running *randtest_name*

coinflip.tests_runner.**run_all_tests**(*series: pandas.core.series.Series*) → Iterator[Tuple[str, coinflip.randtests._result.TestResult, Exception]]

Run all available statistical test on RNG output

Parameters **series** (*Series*) – Output of the RNG being tested

Yields

- **randtest_name** (*str*) – Name of statistical test
- **result** (*TestResult*) – Dataclass that contains the test's statistic and p-value as well as other relevant information gathered.
- **exception** (*NotImplementedError* or *MinimumInputError*) – The exception raised when running *randtest_name*, otherwise *None*.

Raises `NonBinarySequenceError` – If series contains a sequence made of non-binary values

3.3 generators

Generators of binary sequences

Methods infinitely generate *0* and *1* integers to represent a binary sequence.

coinflip.generators.**python**()

Generates random bits using python's *random* module

Yields **bit** (*0* or *1*) – Random bit

See also:

`random.getrandbits()` Method used to generate bits

coinflip.generators.**primes**()

Generates bits representing if natural numbers are prime

Yields **bit** (*0* or *1*) – Whether next number is prime: *0* represents number is a composite (i.e. not a prime), *1* represents number is a prime.

3.4 randtests

Statistical tests for randomness

This subpackage comprises of implementations of statistical tests laid out in a comprehensive paper from NIST[1] in regards to assessing (pseudo-)random number generators.

Notes

A copy of the NIST paper can be found at the root of the *coinflip* repository as SP800-22.pdf.

The test themselves are defined in section 2., "Random Number Generation Tests", p. 23-62. Further detail of the tests are provided in section 3. "Technical Descriptions of Tests", p. 63-87.

These tests were implemented in a complimentary program *sts*, which can be downloaded from the NIST website.

[1] National Institute of Standards and Technology <Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo, Lawrence E Bassham II>, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", *Special Publication 800-22 Revision 1a*, April 2010.

Note that the paper assumes a great amount of familiarity with certain concepts in statistics. It also uses some constants and algorithms without any explaination. Part of the purpose for *coinflip* is to "describe" the NIST tests more wholly than in the paper itself, whilst also reducing the noise of some non-idiomatic programming conventions used in *sts*.

coinflip.randtests.**monobit**(*sequence*)
Proportion of values is compared to expected 1:1 ratio

The difference between the frequency of the two values is found, and referenced to a hypothetically truly random RNG.

Parameters **sequence** (*array-like*) – Output of the RNG being tested

Returns *MonobitTestResult* – Dataclass that contains the test's statistic and p-value as well as other relevant information gathered.

coinflip.randtests.**frequency_within_block**(*sequence*, *candidate=None*, *blocksize=8*)
Proportion of values per block is compared to expected 1:1 ratio

The difference between the frequency of the two values in each block is found, and referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block
- **blocksize** (*int*) – Size of the blocks that partition the given series

Returns *FrequencyWithinBlockTestResult* – Dataclass that contains the test's statistic and p-value as well as other relevant information gathered.

coinflip.randtests.**runs**(*sequence*, *candidate=None*)
Actual number of runs is compared to expected result

The number of runs (uninterrupted sequence of the same value) is found, and referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

coinflip.randtests.**longest_runs**(*sequence*, *candidate=None*)
Longest runs per block is compared to expected result

The longest number of runs (uninterrupted sequence of the same value) per block is found, and referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

coinflip.randtests.**binary_matrix_rank**(*sequence*, *candidate=None*, *matrix_dimen: Tuple[int, int] = None*)
Independence of neighbouring sequences is compared to expected result

Independence is determined by the matrix rank of a subsequence, where it is split into multiple rows to form a matrix. The counts of different rank bins is referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block
- **matrix_dimen** (*Tuple[int, int]*) – Number of rows and columns in each matrix

Returns *BinaryMatrixRankTestResult* – Dataclass that contains the test's statistic and p-value

coinflip.randtests.**spectral**(*sequence*, *candidate=None*)

Potency of periodic features in sequence is compared to expected result

The binary values are treated as the peaks and troughs respectively of a signal, which is applied a Fourier transform so as to find constituent periodic features. The strength of these features is referenced to the expected potent periodic features present in a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is considered the peak in oscillations

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

Raises `NonBinaryTruncatedSequenceError` – When odd-lengthed sequence is truncated there is only one distinct value present

coinflip.randtests.**non_overlapping_template_matching**(*sequence*, *template: List = None*, *nblocks=None*)

Matches of template per block is compared to expected result

The sequence is split into blocks, where the number of non-overlapping matches to the template in each block is found. This is referenced to the expected mean and variance in matches of a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **template** (*List*, optional) – Template to match with the sequence, randomly generated if not provided.
- **nblocks** (*int*) – Number of blocks to split sequence into

Returns *TestResult* – Dataclass that contains the test's statistic and p-value.

Raises `TemplateContainsElementsNotInSequenceError` – If template contains values not present in sequence

coinflip.randtests.**overlapping_template_matching**(*sequence*, *template: List = None*, *nblocks=None*, *df=5*)

Overlapping matches of template per block is compared to expected result

The sequence is split into *nblocks* blocks, where the number of overlapping matches to the template in each block is found. This is referenced to the expected mean and variance in matches of a hypothetically truly random RNG.

Deprecated since version 0: *df* will be removed once I figure out the correct value, as I don't quite understand what NIST wants (or if they're even correct!)

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **template** (*List*, optional) – Template to match with the sequence, randomly generated if not provided.

- **nblocks** (*int*) – Number of blocks to split sequence into
- **df** (*int*, default *5*) – Degrees of freedom to use in p-value calculation

Returns *TestResult* – Dataclass that contains the test's statistic and p-value.

Raises `TemplateContainsElementsNotInSequenceError` – If template contains values not present in sequence

`coinflip.randtests.`**`maurers_universal`**(*sequence*, *blocksize=None*, *init_nblocks=None*)

Distance between patterns is compared to expected result

Unique permutations in an initial sequence are identified, and the distances of aforementioned permutations in a remaining sequence are accumulated. The normalised value for the accumulated distances is then compared to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **blocksize** (*int*) – Size of the blocks that form a permutation
- **init_nblocks** (*int*) – Number of initial blocks to identify permutations

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

Warning: This section is for the private API of *coinflip.randtests*, intended for developers of randomness tests. If you wish to simply run randomness tests on your data (e.g. in a Juypter Notebook session), please use the public API as specified above.

3.4.1 frequency

`coinflip.randtests.frequency.`**`monobit`**(*series*)

Proportion of values is compared to expected 1:1 ratio

The difference between the frequency of the two values is found, and referenced to a hypothetically truly random RNG.

Parameters **sequence** (*array-like*) – Output of the RNG being tested

Returns *MonobitTestResult* – Dataclass that contains the test's statistic and p-value as well as other relevant information gathered.

`coinflip.randtests.frequency.`**`frequency_within_block`**(*series*, *candidate*, *blocksize=8*)

Proportion of values per block is compared to expected 1:1 ratio

The difference between the frequency of the two values in each block is found, and referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block
- **blocksize** (*int*) – Size of the blocks that partition the given series

Returns *FrequencyWithinBlockTestResult* – Dataclass that contains the test's statistic and p-value as well as other relevant information gathered.

3.4.2 runs

coinflip.randtests.runs.**runs**(*series*, *candidate*)
Actual number of runs is compared to expected result

The number of runs (uninterrupted sequence of the same value) is found, and referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

coinflip.randtests.runs.**longest_runs**(*series*, *candidate*)
Longest runs per block is compared to expected result

The longest number of runs (uninterrupted sequence of the same value) per block is found, and referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

coinflip.randtests.runs.**asruns**(*series*) → Iterator[Tuple[Any, int]]
Iterator of runs in a *Series*

Parameters **series** (*Series*) – *Series* to represent as runs

Yields

- **value** (*Any*) – Value of the run
- **length** (*int*) – Length of the run

Notes

A "run" is an uninterrupted sequence of the same value.

3.4.3 matrix

coinflip.randtests.matrix.**binary_matrix_rank**(*series*, *candidate*, *matrix_dimen: Tuple[int, int] = None*)
Independence of neighbouring sequences is compared to expected result

Independence is determined by the matrix rank of a subsequence, where it is split into multiple rows to form a matrix. The counts of different rank bins is referenced to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is counted in each block
- **matrix_dimen** (*Tuple[int, int]*) – Number of rows and columns in each matrix

Returns *BinaryMatrixRankTestResult* – Dataclass that contains the test's statistic and p-value

coinflip.randtests.matrix.**gf2_rank**(*matrix: Iterable[Iterable[int]]*) → int
Finds the rank of a binary matrix

Parameters **matrix** (*List[Tuple[int, ...]]*) – Binary matrix to rank

Returns **rank** (*int*) – Rank of *matrix*

Notes

Implementaton inpisred by a StackOverflow answer from Mark Dickinson.

3.4.4 fourier

coinflip.randtests.fourier.**spectral**(*series*, *candidate*)
Potency of periodic features in sequence is compared to expected result

The binary values are treated as the peaks and troughs respectively of a signal, which is applied a Fourier transform so as to find constituent periodic features. The strength of these features is referenced to the expected potent periodic features present in a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **candidate** (*Value present in given sequence*) – The value which is considered the peak in oscillations

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

Raises `NonBinaryTruncatedSequenceError` – When odd-lengthed sequence is truncated there is only one distinct value present

coinflip.randtests.fourier.**fft**(*array*) → pandas.core.series.Series
Performs fast fourier transform

Parameters **array** (*array-like*) – Input array

Returns *Series* – Fourier transformation of *array*

See also:

numpy.fft.fft() Method adapted to return a *Series* as opposed to an *ndarray*

3.4.5 template

coinflip.randtests.template.**non_overlapping_template_matching**(*series*, *template: List[T] = None*, *nblocks=None*)
Matches of template per block is compared to expected result

The sequence is split into blocks, where the number of non-overlapping matches to the template in each block is found. This is referenced to the expected mean and variance in matches of a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **template** (*List*, optional) – Template to match with the sequence, randomly generated if not provided.

- **nblocks** (*int*) – Number of blocks to split sequence into

Returns *TestResult* – Dataclass that contains the test's statistic and p-value.

Raises `TemplateContainsElementsNotInSequenceError` – If template contains values not present in sequence

`coinflip.randtests.template.`**`overlapping_template_matching`**(*series*, *template: List[T] = None*, *nblocks=None*, *df=5*)

Overlapping matches of template per block is compared to expected result

The sequence is split into *nblocks* blocks, where the number of overlapping matches to the template in each block is found. This is referenced to the expected mean and variance in matches of a hypothetically truly random RNG.

Deprecated since version 0: *df* will be removed once I figure out the correct value, as I don't quite understand what NIST wants (or if they're even correct!)

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **template** (*List*, optional) – Template to match with the sequence, randomly generated if not provided.
- **nblocks** (*int*) – Number of blocks to split sequence into
- **df** (*int*, default *5*) – Degrees of freedom to use in p-value calculation

Returns *TestResult* – Dataclass that contains the test's statistic and p-value.

Raises `TemplateContainsElementsNotInSequenceError` – If template contains values not present in sequence

3.4.6 universal

`coinflip.randtests.universal.`**`maurers_universal`**(*series*, *blocksize=None*, *init_nblocks=None*)

Distance between patterns is compared to expected result

Unique permutations in an initial sequence are identified, and the distances of aforementioned permutations in a remaining sequence are accumulated. The normalised value for the accumulated distances is then compared to a hypothetically truly random RNG.

Parameters

- **sequence** (*array-like*) – Output of the RNG being tested
- **blocksize** (*int*) – Size of the blocks that form a permutation
- **init_nblocks** (*int*) – Number of initial blocks to identify permutations

Returns *TestResult* – Dataclass that contains the test's statistic and p-value

3.4.7 _collections

`class` `coinflip.randtests._collections.`**`FloorDict`**

Subclassed *dict* where invalid keys floor to the smallest real key

If a key is accessed that does not exist, the nearest real key that is the less-than of the passed key is used.

class coinflip.randtests._collections.**RoundingDict**
Subclassed *dict* where invalid keys are rounded to the nearest real key

If a key is accessed that does not exist, the nearest real key to the passed key is used.

class coinflip.randtests._collections.**Bins**(*intervals: Iterable[int]*)
Subclassed *RoundingDict* to initialise intervals as empty bins

3.4.8 _decorators

coinflip.randtests._decorators.**randtest**(*min_input=2*, *rec_input=2*)
Decorator factory for parsing sequences in randomness tests

Returns a decorator (a method which returns a wrapper method). The wrapper checks if passed *sequence* is a pandas *Series*, attempting to convert it if not.

The length of the *sequence* is then checked to see if it meets the passed minimum input requirement, raising an error if not. Subsequently the length is checked with the passed recommended input requirement, issuing a warning if not.

Parameters

- **min_input** (*int*, default *2*) – Absolute minimum length of sequence the test can handle
- **rec_input** (*int*, default *2*) – Recommended minimum length of sequence for the test

Returns **decorator** (*Callable[[Callable], Callable]*) – Decorator method for parsing sequences in randomness tests

Raises MinimumInputError – If *sequence* length exceeds *min_input*

Warns **UserWarning** – If *sequence* length exceeds *rec_input*

See also:

pandas.Series() Initialises with *sequence* if not already a *Series*

coinflip.randtests._decorators.**infer_candidate**(*unique_values*)
Infers the candidate from a list of unique values

An equality check between the values is attempted, where the "largest" value is chosen. If the values can not be compared, then the first element of *unique_values* is chosen.

Parameters **unique_values** (*List*) – List of a unique values

Returns *candidate* – Inferred candidate of *unique_values*

See also:

max() Built-in method used to pick the "largest" value

coinflip.randtests._decorators.**elected**(*func*)
Decorator for parsing candidate arguments in randomness tests

If no *candidate* value is passed, a candidate is inferred from the unique values of the passed *series*.

If a *candidate* value is passed, it is checked to see the value is present in the passed *series*.

Parameters **func** (*Callable*) – Randomness test with candidate kwarg to parse

Returns **wrapper** (*Callable*) – Decorated *func*

Raises CandidateNotInSequenceError – If passed *candidate* value is not present in *series*

See also:

infer_candidate() Method that infers the value of *candidate*

3.4.9 _exceptions

Base exception classes and common exceptions for randomness tests.

exception coinflip.randtests._exceptions.**TestError**
Base class for test-related errors

exception coinflip.randtests._exceptions.**TestNotImplementedError**
Error if test is not implemented to handle valid parameters

exception coinflip.randtests._exceptions.**TestInputError**
Error if test cannot handle (invalid) parameters

exception coinflip.randtests._exceptions.**NonBinarySequenceError**
Error if sequence does not contain only 2 distinct values

3.4.10 _plots

3.4.11 _pprint

Coloured ASCII art representations of binary sequences.

coinflip.randtests._pprint.**determine_rep**
Determine single-character representations of each binary value

Parameters

- **candidate** (*Any*) – One of the two values in a sequence
- **noncandidate** (*Any*) – Value in a sequence which is not *candidate*

Returns

- **c_rep** (*str*) – Character representation of the *candidate*
- **nc_rep** (*str*) – Character representation of the *noncandidate*

See also:

lru_cache Method used for caching results

coinflip.randtests._pprint.**pretty_subseq**(*series*, *candidate*, *noncandidate*) → str
Produce a one-line pretty representation of a subsequence

Parameters

- **series** (*Series*) – Subsequence to represent
- **candidate** (*Any*) – One of the two values in *series*
- **noncandidate** (*Any*) – Value in a sequence which in *series*

Returns **series_rep** (*str*) – Pretty representation of *series*

See also:

determine_rep() Method used to determine the *series* character representations

coinflip.randtests._pprint.**pretty_seq**(*series*, *cols*) → str
Produce a multi-line representation of a sequence

Parameters

- **series** (*Series*) – Sequence to represent
- **cols** (*int*) – Maximum number of characters to use per line

Returns **series_rep** (*str*) – Pretty represented of a sequence

See also:

infer_candidate() Method used to infer the candidate value of the *series*

pretty_subseq() Method wrapped to generate rows

coinflip.randtests._pprint.**dim**(*string*) → str
Wrap string in dim character codes

Parameters **string** (*str*) – String to wrap

Returns *str* – Wrapped string

coinflip.randtests._pprint.**bright**(*string*) → str
Wrap string in bright character codes

Parameters **string** (*str*) – String to wrap

Returns *str* – Wrapped string

3.4.12 _result

class coinflip.randtests._result.**TestResult**(*statistic: Union[int, float], p: float*)
Base container for test result data and subsequent representation methods

Variables

- **statistic** (*int* or *float*) – Statistic of the test
- **p** (*float*) – p-value of the test

coinflip.randtests._result.**smartround**(*num: Union[int, float], ndigits=1*) → Union[int, float]
Round number only if it's a float

3.4.13 _tabulate

3.4.14 _testutils

Utility methods for randomness tests.

coinflip.randtests._testutils.**blocks**(*series*, *blocksize=None*, *nblocks=None*, *truncate=True*) → Iterable[pandas.core.series.Series]
Chunking method for *Series* objects

Parameters

- **series** (*Series*) – The pandas *Series* to chunk
- **blocksize** (*int*, required if no *nblocks* passed) – Size of the chunks

- **nblocks** (*int*, required if no *blocksize* passed) – Number of chunks
- **truncate** (*bool*, default *True*) – Whether to discard remaning series

Yields **block** (*Series*) – Chunk of the passed *series*

Raises `ValueError` – When neither *blocksize* or *nblocks* is passed

`coinflip.randtests._testutils.`**`rawblocks`**`(*args, **kwargs)` → Iterable[Tuple[Any]]

Tuple chunking method for *Series* objects

Parameters

- ***args** – Positional arguments to pass to *blocks*
- ****kwargs** – Keyword arguments to pass to *blocks*

Yields **block_tup** (*Tuple*) – Tuple representation of the block

Raises `ValueError` – When neither *blocksize* or *nblocks* is passed

See also:

`blocks()` The method *rawblocks* adapts

`coinflip.randtests._testutils.`**`check_recommendations`**`(recommendations: Dict[str, bool])`

Warns on recommendation failures

Parameters **recommendations** (*Dict[str, bool]*) – Map of recommendation string representations to the actual recommendation outcomes

Warns **UserWarning** – When one or more recommendations fail

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

4.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.2 Documentation improvements

coinflip could always use more documentation, whether as part of the official coinflip docs, in docstrings, or even on the web in blog posts, articles, and such.

4.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Honno/coinflip/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

4.4 Development

To set up *coinflip* for local development:

1. Fork coinflip (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:Honno/coinflip.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

4.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

4.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 5

Authors

- Matthew Barber - https://matthewbarber.io

CHAPTER 6

Changelog

6.1 0.0.0 (2020-04-21)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T