simple_benchmark Documentation

Release 0.1.0

Michael Seifert

Dec 04, 2020

Contents:

CHAPTER 1

Installation

Using `pip`:

```
python -m pip install simple_benchmark
```

Or installing the most recent version directly from `git`:

```
python -m pip install git+https://github.com/MSeifert04/simple_benchmark.git
```

To utilize the all features of the library (for example visualization) you need to install the optional dependencies:

- NumPy
- pandas
- matplotlib

Or install them automatically using:

```
python -m pip install simple_benchmark[optional]
```

CHAPTER 2

Getting started

Suppose you want to compare how NumPys sum and Pythons sum perform on lists of different sizes:

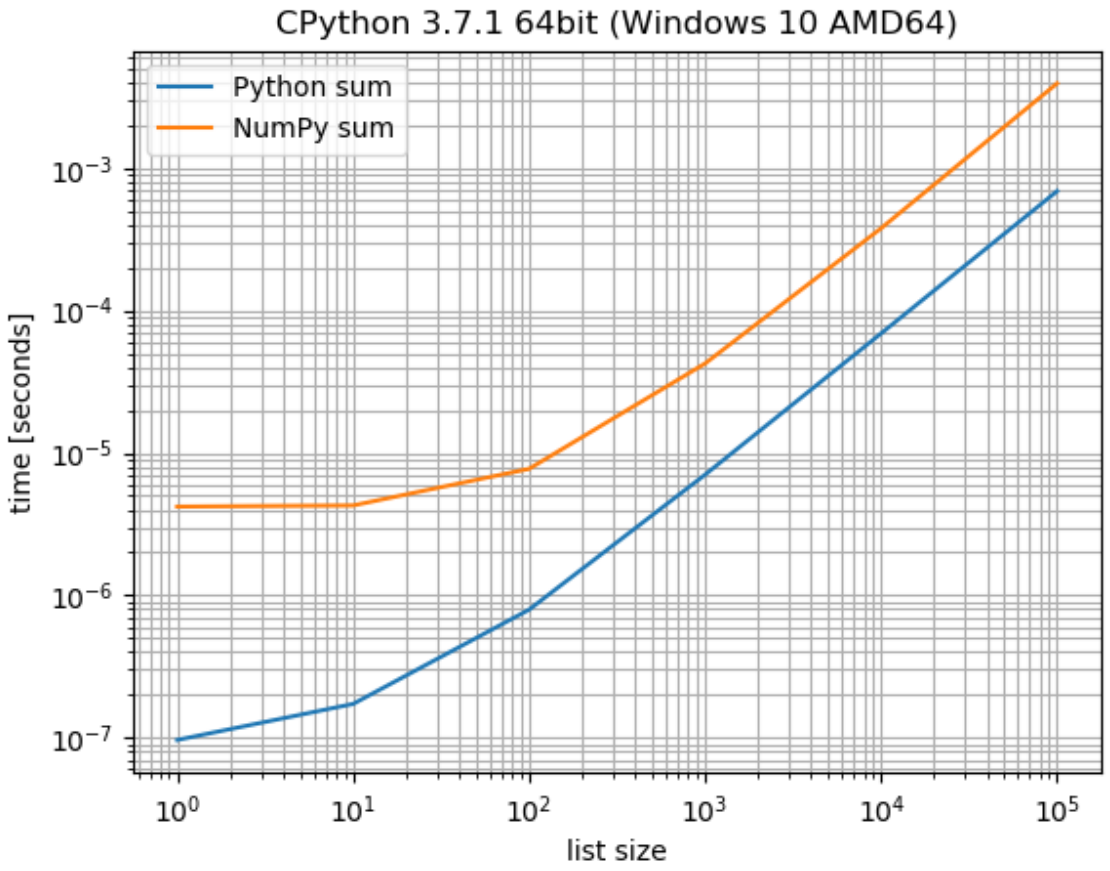
```
>>> from simple_benchmark import benchmark
>>> import numpy as np
>>> funcs = [sum, np.sum]
>>> arguments = {i: [1]*i for i in [1, 10, 100, 1000, 10000, 100000]}
>>> argument_name = 'list size'
>>> aliases = {sum: 'Python sum', np.sum: 'NumPy sum'}
>>> b = benchmark(funcs, arguments, argument_name, function_aliases=aliases)
```

The result can be visualized with `pandas` (needs to be installed):

```
>>> b
           Python sum  NumPy sum
1        9.640884e-08   0.000004
10       1.726930e-07   0.000004
100      7.935484e-07   0.000008
1000     7.040000e-06   0.000042
10000    6.910000e-05   0.000378
100000   6.899000e-04   0.003941
```

Or with `matplotlib` (has to be installed too):

```
>>> b.plot()

>>> # To save the plotted benchmark as PNG file.
>>> import matplotlib.pyplot as plt
>>> plt.savefig('sum_example.png')
```


CPython 3.7.1 64bit (Windows 10 AMD64)
Python sum
NumPy sum
time [seconds]
list size
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}

CHAPTER 3

Command-Line interface

Warning: The command line interface is highly experimental. It's very likely to change its API.

It's an experiment to run it as command-line tool, especially useful if you want to run it on multiple files and don't want the boilerplate.

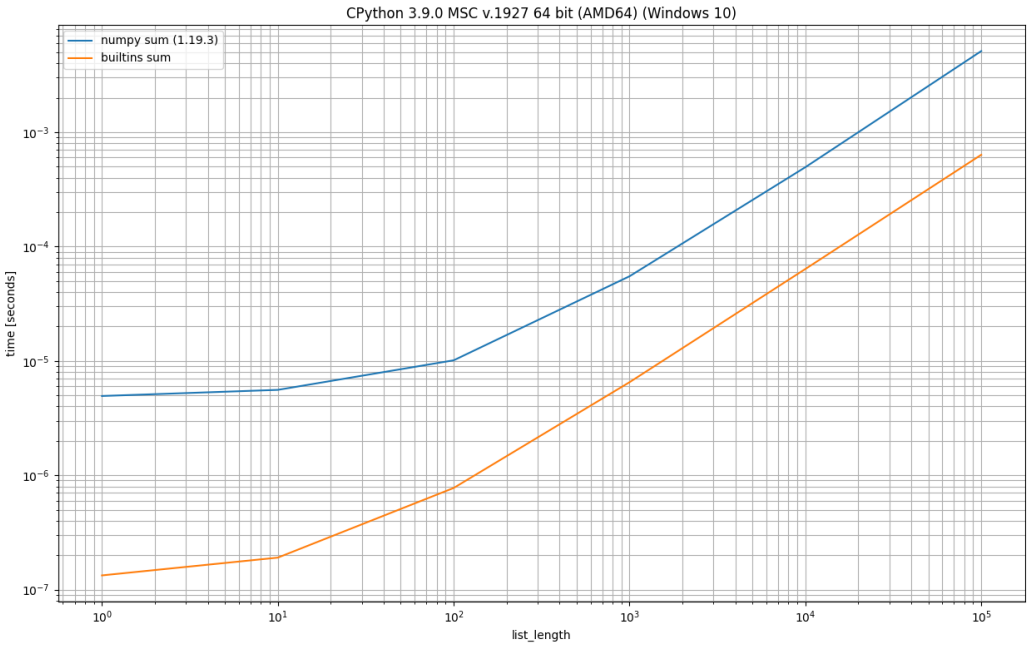
File `sum.py`:

```
import numpy as np

def bench_sum(l, func=sum):
   return func(l)

def bench_numpy_sum(l, func=np.sum):
   return np.sum(l)

def args_list_length():
   for i in [1, 10, 100, 1000, 10000, 100000]:
      yield i, [1]*i
```

Then run:

```
$ python -m simple_benchmark sum.py sum.png
```

With a similar result `sum.png`:



3.1 Extended examples

3.1.1 BenchmarkBuilder

The *simple_benchmark.BenchmarkBuilder* class can be used to build a benchmark using decorators, essentially it is just a wrapper around *simple_benchmark.benchmark()*.

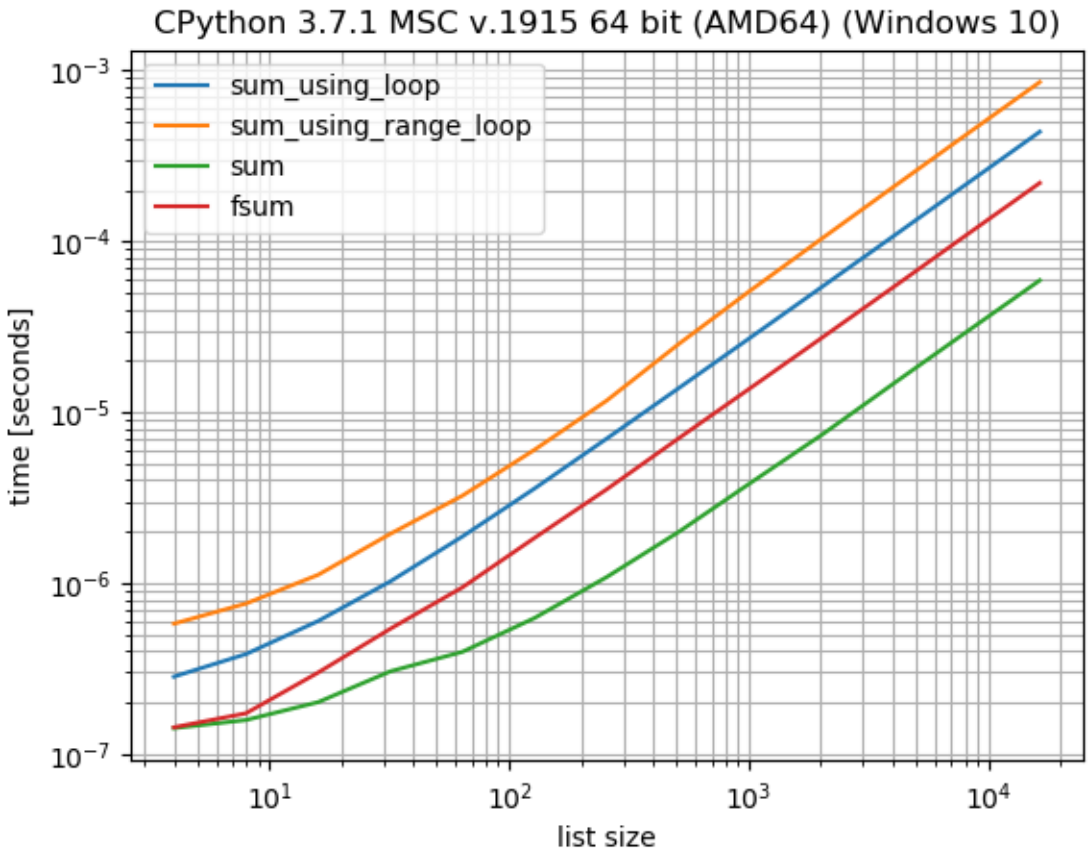
For example to compare different approaches to calculate the sum of a list of floats:

```
from simple_benchmark import BenchmarkBuilder
import math

bench = BenchmarkBuilder()

@bench.add_function()
def sum_using_loop(lst):
    sum_ = 0
    for item in lst:
        sum_ += item
    return sum_

@bench.add_function()
def sum_using_range_loop(lst):
    sum_ = 0
    for idx in range(len(lst)):
        sum_ += lst[idx]
    return sum_

bench.use_random_lists_as_arguments(sizes=[2**i for i in range(2, 15)])
```

(continues on next page)

(continued from previous page)

```
bench.add_functions([sum, math.fsum])

b = bench.run()
b.plot()
# To save the plotted benchmark as PNG file.
import matplotlib.pyplot as plt
plt.savefig('sum_list_example.png')
```



3.1.2 MultiArgument

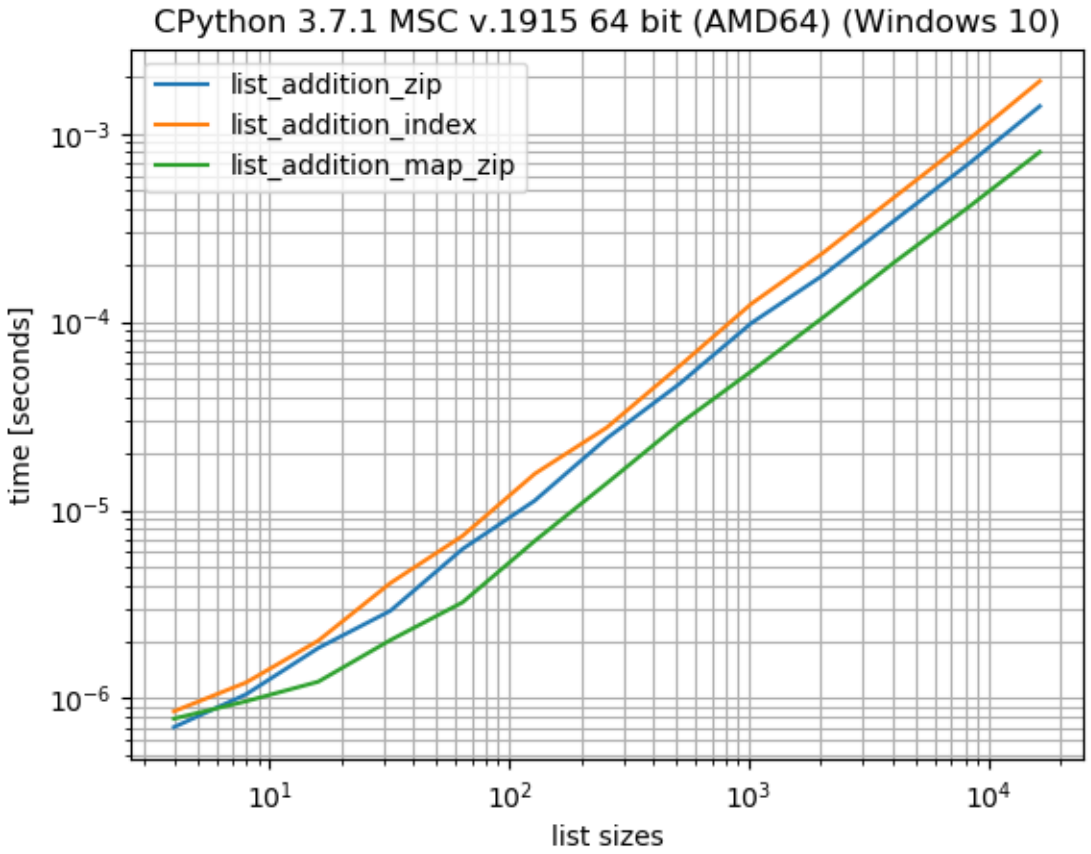
The *simple_benchmark.MultiArgument* class can be used to provide multiple arguments to the functions that should be benchmarked:

```
from itertools import starmap
from operator import add
from random import random

from simple_benchmark import BenchmarkBuilder, MultiArgument

bench = BenchmarkBuilder()

@bench.add_function()
```

(continues on next page)

(continued from previous page)

```
def list_addition_zip(list1, list2):
    res = []
    for item1, item2 in zip(list1, list2):
        res.append(item1 + item2)
    return res

@bench.add_function()
def list_addition_index(list1, list2):
    res = []
    for idx in range(len(list1)):
        res.append(list1[idx] + list2[idx])
    return res

@bench.add_function()
def list_addition_map_zip(list1, list2):
    return list(starmap(add, zip(list1, list2)))

@bench.add_arguments(name='list sizes')
def benchmark_arguments():
    for size_exponent in range(2, 15):
        size = 2**size_exponent
        arguments = MultiArgument([
            [random() for _ in range(size)],
            [random() for _ in range(size)]])
        yield size, arguments

b = bench.run()
b.plot()
# To save the plotted benchmark as PNG file.
import matplotlib.pyplot as plt
plt.savefig('list_add_example.png')
```



3.1.3 Asserting correctness

Besides comparing the timings it's also important to assert that the approaches actually produce the same outcomes and don't modify the input arguments.

To compare the results there is *simple_benchmark.assert_same_results()* (or in case you use BenchmarkBuilder *simple_benchmark.BenchmarkBuilder.assert_same_results()*):

```
import operator
import random
from simple_benchmark import assert_same_results

funcs = [min, max]  # will produce different results
arguments = {2**i: [random.random() for _ in range(2**i)] for i in range(2, 10)}
assert_same_results(funcs, arguments, equality_func=operator.eq)
```

And to compare that the inputs were not modified *simple_benchmark.assert_not_mutating_input()* (or in case you use BenchmarkBuilder *simple_benchmark.BenchmarkBuilder.assert_not_mutating_input()*):

```
import operator
import random
from simple_benchmark import assert_not_mutating_input

```

(continues on next page)

(continued from previous page)

```
def sort(l):
    l.sort()  # modifies the input
    return l

funcs = [sorted, sort]
arguments = {2**i: [random.random() for _ in range(2**i)] for i in range(2, 10)}
assert_not_mutating_input(funcs, arguments, equality_func=operator.eq)
```

Both will produce an `AssertionError` if they gave different results or mutate the input arguments.

Typically the `equality_func` will be one of these:

- `operator.eq()` will work for most Python objects.
- `math.isclose()` will work for `float` that may be close but not equal.
- `numpy.array_equal` will work for element-wise comparison of NumPy arrays.
- `numpy.allclose` will work for element-wise comparison of NumPy arrays containing floats that may be close but not equal.

The *`simple_benchmark.assert_not_mutating_input()`* also accepts an optional argument that needs to be used in case the argument is not trivially copyable. It expects a function that takes the argument as input and should return a deep-copy of the argument.

3.1.4 Times for each benchmark

The benchmark will run each function on each of the arguments for a certain amount of times. Generally the results will be more accurate if one increases the number of times the function is executed during each benchmark. But the benchmark will also take longer.

To control the time one benchmark should take one can use the `time_per_benchmark` argument. This controls how much time each function will take for each argument. The default is 0.1s (100 milliseconds) but the value is ignored for calls that either take very short (then it will finish faster) or very slow (because the benchmark tries to do at least a few calls).

Another option is to control the maximum time a single function call may take `maximum_time`. If the first call of this function exceeds the `maximum_time` the function will be excluded from the benchmark from this argument on.

- To control the quality of the benchmark the `time_per_benchmark` can be used.
- To avoid excessive benchmarking times one can use `maximum_time`.

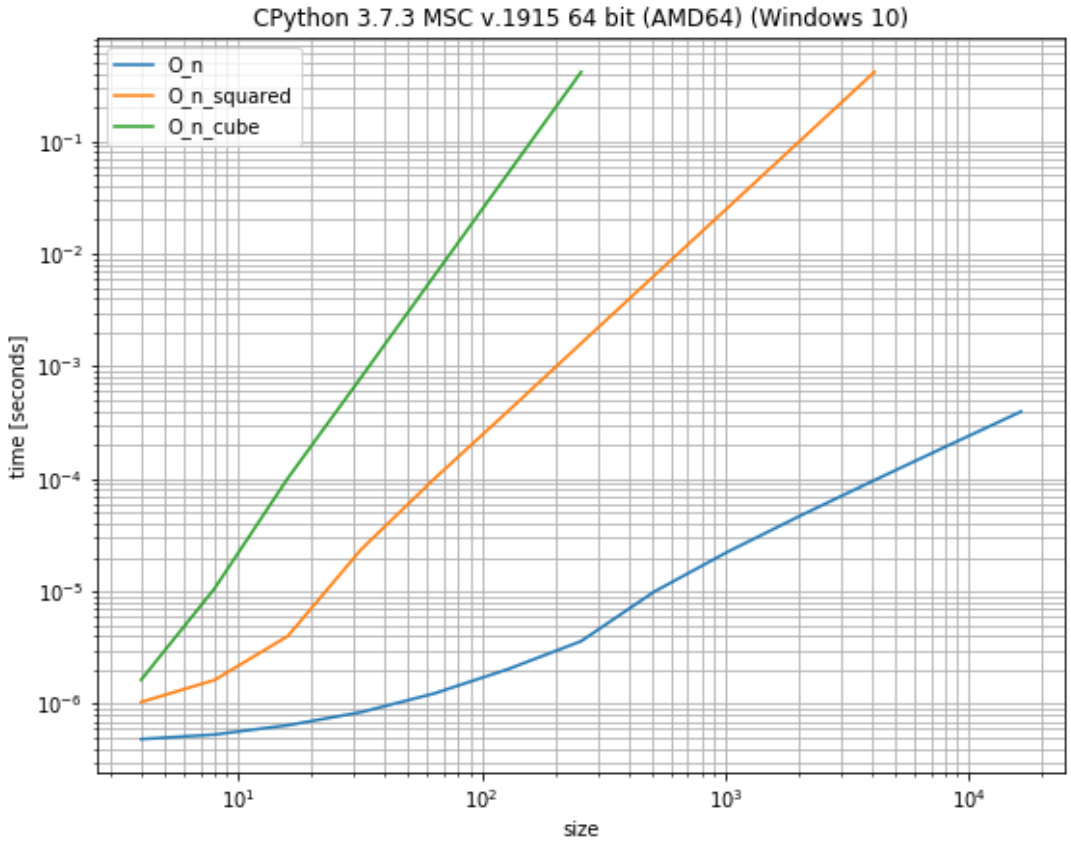
An example showing both in action:

```
from simple_benchmark import benchmark
from datetime import timedelta

def O_n(n):
    for i in range(n):
        pass

def O_n_squared(n):
    for i in range(n ** 2):
        pass

def O_n_cube(n):
    for i in range(n ** 3):
```

(continues on next page)

(continued from previous page)

```
        pass

b = benchmark(
    [O_n, O_n_squared, O_n_cube],
    {2**i: 2**i for i in range(2, 15)},
    time_per_benchmark=timedelta(milliseconds=500),
    maximum_time=timedelta(milliseconds=500)
)

b.plot()
# To save the plotted benchmark as PNG file.
import matplotlib.pyplot as plt
plt.savefig('time_example.png')
```



3.1.5 Examples on StackOverflow

In some cases it's probably best to see how it can be used on some real-life problems:

- Count the number of non zero values in a numpy array in Numba
- When numba is effective?
- Range with repeated consecutive numbers

- Concatenate tuples using sum()
- How to retrieve an element from a set without removing it?
- What exactly is the optimization "functools.partial" is making?
- Nested lambda statements when sorting lists
- How to make a flat list out of list of lists?
- How do you remove duplicates from a list whilst preserving order?
- Iterating over every two elements in a list
- Cython - efficiently filtering a typed memoryview
- Python's sum vs. NumPy's numpy.sum
- Finding longest run in a list
- Remove duplicate dict in list in Python
- How do I find the duplicates in a list and create another list with them?
- Suppress key addition in collections.defaultdict
- Numpy first occurrence of value greater than existing value
- Count the number of times an item occurs in a sequence using recursion Python
- Converting a series of ints to strings - Why is apply much faster than astype?

See also Results for "simple_benchmark" on StackOverflow.

3.2 Command Line

3.2.1 Using the Command Line

Warning: The command line interface is highly experimental. It's very likely to change its API.

When you have all optional dependencies installed you can also run `simple_benchmark`, in the most basic form it would be:

```
$ python -m simple_benchmark INPUT_FILE OUTPUT_FILE
```

Which processes the `INPUT_FILE` and writes a plot to `OUTPUT_FILE`.

However in order to work correctly the `INPUT_FILE` has to fulfill several criteria:

- It must be a valid Python file.
- All functions that should be benchmarked have to have a name starting with `bench_` and everything thereafter is used for the label.
- The function generating the arguments for the benchmark has to start with `args_` and everything thereafter is used for the label of the x-axis.

Also if the benchmarked function has a `func` parameter with a default it will be used to determine the `alias` (the displayed name in the table and plot).

3.2.2 Parameters

The first two parameters are the input and output file. However there are a few more parameters. These can be also seen when running:

```
$ python -m simple_benchmark -h
usage: __main__.py [-h] [-s FIGSIZE] [--time-per-benchmark TIME_PER_BENCHMARK] [-v] [-
↪-write-csv] filename out

Benchmark a file

positional arguments:
filename              the file to run the benchmark on.
out                   Specifies the output file for the plot

optional arguments:
-h, --help            show this help message and exit
-s FIGSIZE, --figsize FIGSIZE
                        Specify the output size in inches, needs to be wrapped in␣
↪quotes on most shells, e.g. "15, 9" (default: 15, 9)
--time-per-benchmark TIME_PER_BENCHMARK
                        The target time for each individual benchmark in seconds␣
↪(default: 0.1)
-v, --verbose         prints additional information on stdout (default: False)
--write-csv           Writes an additional CSV file of the results (default: False)
```

3.3 API Reference

Warning: This package is under active development. API changes are very likely.

This package aims to give an easy way to benchmark several functions for different inputs and provide ways to visualize the benchmark results.

To utilize the full features (visualization and post-processing) you need to install the optional dependencies:

- NumPy
- pandas
- matplotlib

simple_benchmark.**assert_same_results**(*funcs*, *arguments*, *equality_func*)

Asserts that all functions return the same result.

New in version 0.1.0.

Parameters

- **funcs** (*iterable of callables*) – The functions to check.
- **arguments** (*dict*) – A dictionary containing where the key represents the reported value (for example an integer representing the list size) as key and the argument for the functions (for example the list) as value. In case you want to plot the result it should be sorted and ordered (e.g. an collections.OrderedDict or a plain dict if you are using Python 3.7 or later).

- **equality_func** (*callable*) – The function that determines if the results are equal. This function should accept two arguments and return a boolean (True if the results should be considered equal, False if not).

Raises AssertionError – In case any two results are not equal.

simple_benchmark.**assert_not_mutating_input**(*funcs*, *arguments*, *equality_func*, *copy_func=<function deepcopy>*)

Asserts that none of the functions mutate the arguments.

New in version 0.1.0.

Parameters

- **funcs** (*iterable of callables*) – The functions to check.
- **arguments** (*dict*) – A dictionary containing where the key represents the reported value (for example an integer representing the list size) as key and the argument for the functions (for example the list) as value. In case you want to plot the result it should be sorted and ordered (e.g. an collections.OrderedDict or a plain dict if you are using Python 3.7 or later).
- **equality_func** (*callable*) – The function that determines if the results are equal. This function should accept two arguments and return a boolean (True if the results should be considered equal, False if not).
- **copy_func** (*callable, optional*) – The function that is used to copy the original argument. Default is copy.deepcopy().

Raises AssertionError – In case any two results are not equal.

Notes

In case the arguments are *MultiArgument* then the copy_func and the equality_func get these *MultiArgument* as single arguments and need to handle them appropriately.

simple_benchmark.**benchmark**(*funcs*, *arguments*, *argument_name=''*, *warmups=None*, *time_per_benchmark=datetime.timedelta(microseconds=100000)*, *function_aliases=None*, *estimator=<built-in function min>*, *maximum_time=None*)

Create a benchmark suite for different functions and for different arguments.

Parameters

- **funcs** (*iterable of callables*) – The functions to benchmark.
- **arguments** (*dict*) – A dictionary containing where the key represents the reported value (for example an integer representing the list size) as key and the argument for the functions (for example the list) as value. In case you want to plot the result it should be sorted and ordered (e.g. an collections.OrderedDict or a plain dict if you are using Python 3.7 or later).
- **argument_name** (*str, optional*) – The name of the reported value. For example if the arguments represent list sizes this could be *"size of the list"*. Default is an empty string.
- **warmups** (*None or iterable of callables, optional*) – If not None it specifies the callables that need a warmup call before being timed. That is so, that caches can be filled or jitters to kick in. Default is None.
- **time_per_benchmark** (*datetime.timedelta, optional*) – Each benchmark should take approximately this time. The value is ignored for functions that take very little time or very long. Default is 0.1 seconds.

Changed in version 0.1.0: Now requires a `datetime.timedelta` instead of a `float`.

- **function_aliases** (*None or dict, optional*) – If not None it should be a dictionary containing the function as key and the name of the function as value. The value will be used in the final reports and plots. Default is None.
- **estimator** (*callable, optional*) – Each function is called with each argument multiple times and each timing is recorded. The benchmark_estimator (by default `min()`) is used to reduce this list of timings to one final value. The minimum is generally a good way to estimate how fast a function can run (see also the discussion in `timeit.Timer.repeat()`). Default is `min()`.
- **maximum_time** (*datetime.timedelta or None, optional*) – If not None it represents the maximum time the first call of the function may take. If exceeded the benchmark will stop evaluating the function from then on. Default is None.

  New in version 0.1.0.

Returns **benchmark** – The result of the benchmarks.

Return type *BenchmarkResult*

See also:

BenchmarkBuilder()

class simple_benchmark.**BenchmarkBuilder**(*time_per_benchmark=datetime.timedelta(microseconds=100000)*, *estimator=<built-in function min>*, *maximum_time=None*)

A class useful for building benchmarks by adding decorators to the functions instead of collecting them later.

Parameters

- **time_per_benchmark** (*datetime.timedelta, optional*) – Each benchmark should take approximately this time. The value is ignored for functions that take very little time or very long. Default is 0.1 seconds.

  Changed in version 0.1.0: Now requires a `datetime.timedelta` instead of a `float`.
- **estimator** (*callable, optional*) – Each function is called with each argument multiple times and each timing is recorded. The benchmark_estimator (by default `min()`) is used to reduce this list of timings to one final value. The minimum is generally a good way to estimate how fast a function can run (see also the discussion in `timeit.Timer.repeat()`). Default is `min()`.
- **maximum_time** (*datetime.timedelta or None, optional*) – If not None it represents the maximum time the first call of the function may take. If exceeded the benchmark will stop evaluating the function from then on. Default is None.

  New in version 0.1.0.

See also:

benchmark

add_arguments(*name=''*)

A decorator factory that returns a decorator that can be used to add a function that produces the x-axis values and the associated test data for the benchmark.

Parameters **name** (*str, optional*) – The label for the x-axis.

Returns **decorator** – The decorator that adds the function that produces the x-axis values and the test data to the benchmark.

Return type callable

Raises `TypeError` – In case `name` is a callable.

add_function(*warmups=False*, *alias=None*)
A decorator factory that returns a decorator that can be used to add a function to the benchmark.

Parameters

- **warmups** (*bool, optional*) – If true the function is called once before each benchmark run. Default is False.
- **alias** (*str or None, optional*) – If None then the displayed function name is the name of the function, otherwise the string is used when the function is referred to. Default is None.

Returns **decorator** – The decorator that adds the function to the benchmark.

Return type callable

Raises `TypeError` – In case `name` is a callable.

add_functions(*functions*)
Add multiple functions to the benchmark.

Parameters **functions** (*iterable of callables*) – The functions to add to the benchmark

assert_not_mutating_input(*equality_func*, *copy_func=<function deepcopy>*)
Asserts that none of the stored functions mutate the arguments.

New in version 0.1.0.

Parameters

- **equality_func** (*callable*) – The function that determines if the results are equal. This function should accept two arguments and return a boolean (True if the results should be considered equal, False if not).
- **copy_func** (*callable, optional*) – The function that is used to copy the original argument. Default is `copy.deepcopy()`.

Warns **UserWarning** – In case the instance has no arguments for the functions.

Raises `AssertionError` – In case any two results are not equal.

Notes

In case the arguments are *MultiArgument* then the copy_func and the equality_func get these *MultiArgument* as single arguments and need to handle them appropriately.

assert_same_results(*equality_func*)
Asserts that all stored functions return the same result.

New in version 0.1.0.

Parameters **equality_func** (*callable*) – The function that determines if the results are equal. This function should accept two arguments and return a boolean (True if the results should be considered equal, False if not).

Warns **UserWarning** – In case the instance has no arguments for the functions.

Raises `AssertionError` – In case any two results are not equal.

run()
Starts the benchmark.

Returns **result** – The result of the benchmark.

Return type *BenchmarkResult*

Warns **UserWarning** – In case the instance has no arguments for the functions.

New in version 0.1.0.

use_random_arrays_as_arguments(*sizes*)
Alternative to *add_arguments()* that provides random arrays of the specified sizes as arguments for the benchmark.

Parameters **sizes** (*iterable of int*) – An iterable containing the sizes for the arrays (should be sorted).

Raises ImportError – If NumPy isn't installed.

use_random_lists_as_arguments(*sizes*)
Alternative to *add_arguments()* that provides random lists of the specified sizes as arguments for the benchmark.

Parameters **sizes** (*iterable of int*) – An iterable containing the sizes for the lists (should be sorted).

class simple_benchmark.**BenchmarkResult**(*timings*, *function_aliases*, *arguments*, *argument_name*)
A class holding a benchmarking result that provides additional printing and plotting functions.

plot(*relative_to=None*, *ax=None*)
Plot the benchmarks, either relative or absolute.

Parameters

- **relative_to** (*callable or None, optional*) – If None it will plot the absolute timings, otherwise it will use the given *relative_to* function as reference for the timings.
- **ax** (*matplotlib.axes.Axes or None, optional*) – The axes on which to plot. If None plots on the currently active axes.

Raises ImportError – If matplotlib isn't installed.

plot_both(*relative_to*)
Plot both the absolute times and the relative time.

Parameters **relative_to** (*callable or None*) – If None it will plot the absolute timings, otherwise it will use the given *relative_to* function as reference for the timings.

Raises ImportError – If matplotlib isn't installed.

plot_difference_percentage(*relative_to*, *ax=None*)
Plot the benchmarks relative to one of the benchmarks with percentages on the y-axis.

Parameters

- **relative_to** (*callable*) – The benchmarks are plotted relative to the timings of the given function.
- **ax** (*matplotlib.axes.Axes or None, optional*) – The axes on which to plot. If None plots on the currently active axes.

Raises ImportError – If matplotlib isn't installed.

to_pandas_dataframe()
Return the timing results as pandas DataFrame. This is the preferred way of accessing the text form of the timings.

Returns **results** – The timings as DataFrame.

Return type pandas.DataFrame

Warns **UserWarning** – In case multiple functions have the same name.

Raises ImportError – If pandas isn't installed.

class simple_benchmark.**MultiArgument**
Class that behaves like a tuple but signals to the benchmark that it should pass multiple arguments to the function to benchmark.

3.4 Changelog

3.4.1 0.1.0 (not released)

- Added a command-line command for performing benchmarks on carefully constructed Python files (experimental)
- Added the functions `assert_same_results` and `assert_not_mutating_input`.
- The argument `time_per_benchmark` of `benchmark` and `BenchmarkBuilder` now expects a `datetime.timedelta` instead of a `float`.
- Added `maximum_time` for `benchmark` and `BenchmarkBuilder` to control the maximum time for a single function execution. If exceeded the function will not be not be benchmarked anymore.
- Added info-level based logging during benchmark runs.

3.4.2 0.0.9 (2019-04-07)

- Fixed wrong name for optional dependencies in `extras_require` of `setup.py`
- Added development documentation.

3.4.3 0.0.8 (2019-04-06)

- Removed `benchmark_random_list` and `benchmark_random_array` in favor of the static methods `use_random_lists_as_arguments` and `use_random_arrays_as_arguments` on `BenchmarkBuilder`.
- Added `BenchmarkBuilder` class that provides a decorator-based construction of a benchmark.
- Added a title to the plot created by the `plot` functions of `BenchmarkResult` that displays some information about the Python installation and environment.

3.4.4 0.0.7 (2018-04-30)

- Added optional `estimator` argument to the benchmark functions. The `estimator` can be used to calculate the reported runtime based on the individual timings.

3.4.5 0.0.6 (2018-04-30)

- Added `plot_difference_percentage` to `BenchmarkResult` to plot percentage differences.

3.4.6 0.0.5 (2018-04-22)

- Print a warning in case multiple functions have the same name
- Use `OrderedDict` to fix issues on older Python versions where `dict` isn't ordered.

3.4.7 0.0.4 (2018-04-19)

- Added `MultiArgument` class to provide a way to pass in multiple arguments to the functions.

3.4.8 0.0.3 (2018-04-16)

- Some bugfixes.

3.4.9 0.0.2 (2018-04-16)

- General restructuring.

3.4.10 0.0.1 (2018-02-19)

- Initial release.

3.5 License

(continues on next page)

(continued from previous page)

```
    "You" (or "Your") shall mean an individual or Legal Entity
    exercising permissions granted by this License.

    "Source" form shall mean the preferred form for making modifications,
    including but not limited to software source code, documentation
    source, and configuration files.

    "Object" form shall mean any form resulting from mechanical
    transformation or translation of a Source form, including but
    not limited to compiled object code, generated documentation,
    and conversions to other media types.

    "Work" shall mean the work of authorship, whether in Source or
    Object form, made available under the License, as indicated by a
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    "Derivative Works" shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    "Contribution" shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, "submitted"
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
```

(continues on next page)

(continued from previous page)

```
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
```

(continues on next page)

(continued from previous page)

```
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright since 2018 Michael Seifert

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```

3.6 Development

Prerequisites:

- Cloned or downloaded source repository. For example `git clone https://github.com/MSeifert04/simple_benchmark.git`.
- You're in the root directory of the cloned (or downloaded) repository.
- Have an installed Python with pip and setuptools (the following will assume that the Python executable is in your path!).

3.6.1 Building the package locally

Navigate to the root directory of the repository (the directory where the `setup.py` file is) and then run one of these commands:

```
python setup.py develop
```

or:

```
python -m pip install -e .
```

In case you want to install all the optional dependencies automatically (**recommended**):

```
python -m pip install -e .[optional]
```

3.6.2 Building the documentation locally

This requires that the package was installed with all development dependencies:

```
python -m pip install -e .[development]
```

Then just run:

```
python setup.py build_sphinx
```

The generated HTML documentation should then be available in the `./build/sphinx/html` folder.

3.6.3 Running the tests locally

This requires that the package was installed with all development dependencies:

```
python -m pip install -e .[development]
```

Then use `pytest`:

```
python -m pytest tests
```

Or to exclude the tests marked as slow:

```
python -m pytest tests -m "not slow"
```

3.6.4 Publishing the package to PyPI

Note: This is maintainer-only!

To install the necessary packages run:

```
python -m pip install -e .[maintainer]
```

First clean the repository to avoid outdated artifacts:

```
git clean -dfX
```

Then build the source distribution, since it's a very small package without compiled modules, we can omit building wheels:

```
python setup.py sdist
```

Then upload to PyPI:

```
python -m twine upload --repository-url https://upload.pypi.org/legacy/ dist/*
```

You will be prompted for the username and password.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index