

# Timing and Benchmarking Scientific Python

EuroSciPy 2023

---

Kai Striega

2023-08-16

Software Developer & SciPy Maintainer

# What we're going to cover

Why this talk?

Why does time matter?

Thinking of measurement as an experiment

Taking a single measurement

Running a single Benchmark

What's out there?

Benchmark Design

Our benchmark

Comparing Benchmarks

Why this talk?

---

What this talk was supposed to be

## What this talk was supposed to be

- Look at timing and benchmarking in the SciPy ecosystem

# What this talk was supposed to be

- Look at timing and benchmarking in the SciPy ecosystem
- Analyse the methodology of different articles & papers in the scientific Python ecosystem

# What this talk was supposed to be

- Look at timing and benchmarking in the SciPy ecosystem
- Analyse the methodology of different articles & papers in the scientific Python ecosystem
- Discuss what is done well and where improvements could be made

# What this talk was supposed to be

- Look at timing and benchmarking in the SciPy ecosystem
- Analyse the methodology of different articles & papers in the scientific Python ecosystem
- Discuss what is done well and where improvements could be made
- Apply the points learnt to *SciPy*'s benchmarking suite



That didn't happen

# That didn't happen

- Dearth of papers looking at performance of Scientific Python

# That didn't happen

- Dearth of papers looking at performance of Scientific Python
- Many lack adequate analysis of their results

# That didn't happen

- Dearth of papers looking at performance of Scientific Python
- Many lack adequate analysis of their results
- Many did not even state their methodology

# What this talk is

# What this talk is

- Advocate for a statistically rigorous approach to timing

# What this talk is

- Advocate for a statistically rigorous approach to timing
- Cover topics **you** should consider when timing

# What this talk is not

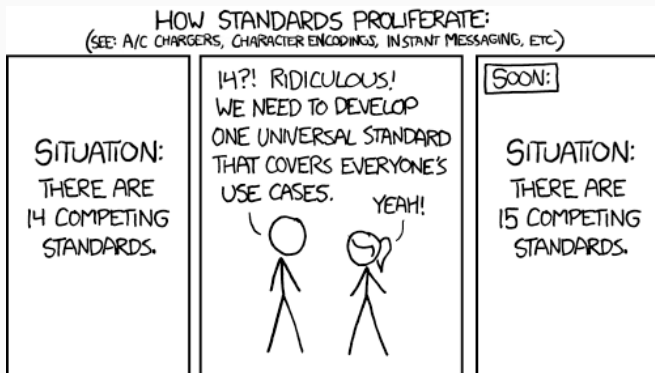


Figure 1: standards



Why does time matter?

---

## A common problem...

## A common problem...

- You have two solutions  $S$  and  $S'$

## A common problem...

- You have two solutions  $S$  and  $S'$
- You are told that:
  - $S$  runs in 100s
  - $S'$  runs in 95s

## A common problem...

- You have two solutions  $S$  and  $S'$
- You are told that:
  - $S$  runs in 100s
  - $S'$  runs in 95s
- Which is faster?

## A common problem...

- You have two solutions  $S$  and  $S'$
- You are told that:
  - $S$  runs in 100s
  - $S'$  runs in 95s
- Which is faster?
- How sure are you that it is faster?

## Computers can reproduce answers, not performance

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 343 msec per loop
```

## Computers can reproduce answers, not performance

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 343 msec per loop
```

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 310 msec per loop
```



## Computers can reproduce answers, not performance

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 343 msec per loop
```

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 310 msec per loop
```

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 312 msec per loop
```

## Computers can reproduce answers, not performance

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 343 msec per loop
```

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 310 msec per loop
```

```
$ python -m timeit "sum(n*n for n in range(10000000))"  
1 loop, best of 5: 312 msec per loop
```

Which run gives the **true** time?

# Variance makes time measurement hard

# Variance makes time measurement hard

- Computers can reproduce answers bit for bit

# Variance makes time measurement hard

- Computers can reproduce answers bit for bit
- Computers cannot reproduce runtime

Time is an important metric

# Time is an important metric

- Who **likes** waiting?

# Time is an important metric

- Who **likes** waiting?
- There are many performance metrics...



# Time is an important metric

- Who **likes** waiting?
- There are many performance metrics...
- ...many of which depend on time

# Time is an important metric

- Who **likes** waiting?
- There are many performance metrics...
- ...many of which depend on time
- Accurate time measurement is crucial for accurate metrics

But we usually do not measure time well

## But we usually do not measure time well

- Rigorous benchmarking and performance analysis methodologies are lacking in Python

## But we usually do not measure time well

- Rigorous benchmarking and performance analysis methodologies are lacking in Python
- Different studies use different, often ad-hoc, methodologies

## But we usually do not measure time well

- Rigorous benchmarking and performance analysis methodologies are lacking in Python
- Different studies use different, often ad-hoc, methodologies
- Many lack statistical treatment of their results

## But we usually do not measure time well

- Rigorous benchmarking and performance analysis methodologies are lacking in Python
- Different studies use different, often ad-hoc, methodologies
- Many lack statistical treatment of their results
- Many do not even state their methodologies

## Thinking of measurement as an experiment

---



# Statistically rigorous?

There are three kinds of lies:  
lies, damned lies, and  
statistics.

---

Unknown

# What makes a good experiment?

# What makes a good experiment?

**Clear Hypothesis** The experiment should have a well-defined objective, and the research question or hypothesis being tested should be explicit.

# What makes a good experiment?

**Clear Hypothesis** The experiment should have a well-defined objective, and the research question or hypothesis being tested should be explicit.

**Relevant** The experiment being run should be relevant to what is being studied.

# What makes a good experiment?

- Clear Hypothesis** The experiment should have a well-defined objective, and the research question or hypothesis being tested should be explicit.
- Relevant** The experiment being run should be relevant to what is being studied.
- Replicable** Repetition of the experiment under similar conditions strengthens the validity of the results.

# What makes a good experiment?

- Clear Hypothesis** The experiment should have a well-defined objective, and the research question or hypothesis being tested should be explicit.
- Relevant** The experiment being run should be relevant to what is being studied.
- Replicable** Repetition of the experiment under similar conditions strengthens the validity of the results.
- Documented** Clearly document all aspects of the experiment, including the study design, methods, results, and limitations.

# What to measure?

# What to measure?

- Clock cycles



# What to measure?

- Clock cycles
- Time

# How to make benchmarks reproducible?

# How to make benchmarks reproducible?

- *My principle:* Benchmarks should require as little human effort as possible to understand and to be run

# How to make benchmarks reproducible?

- *My principle*: Benchmarks should require as little human effort as possible to understand and to be run
- Two main tools:

# How to make benchmarks reproducible?

- *My principle*: Benchmarks should require as little human effort as possible to understand and to be run
- Two main tools:
  - Documentation

# How to make benchmarks reproducible?

- *My principle*: Benchmarks should require as little human effort as possible to understand and to be run
- Two main tools:
  - Documentation
  - Automation







## Taking a single measurement

---

You can't fix by analysis what  
you bungled by design.

---

Light, Singer, and Willett [1990]

## What effects a single measurement (not exhaustive!)

# What effects a single measurement (not exhaustive!)

Observer Effect

# What effects a single measurement (not exhaustive!)

Observer Effect

Hardware Effects

# What effects a single measurement (not exhaustive!)

Observer Effect

Hardware Effects

Garbage Collection

# What effects a single measurement (not exhaustive!)

Observer Effect

Hardware Effects

Garbage Collection

Warmups & Steady State





- All forms of instrumentation may change the result

- All forms of instrumentation may change the result
- Instrumentation normally adds overhead

- All forms of instrumentation may change the result
- Instrumentation normally adds overhead
- "You thought the code was slow to start with, so you made it slower to see how slow it was" - Adelstein-Lelbach [2015]



- Many, many, many possible effects

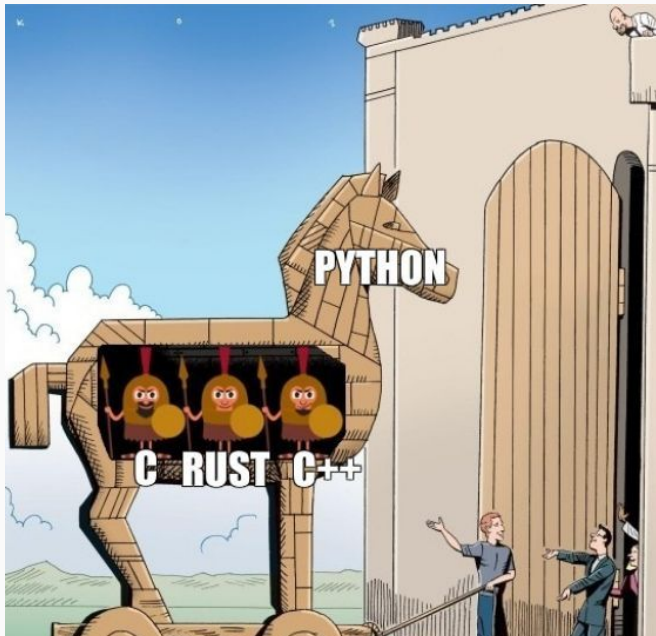
- Many, many, many possible effects
- For example, Mytkowicz et al. [2009] showed that link order and environment variable size can significantly affect measurement

- Many, many, many possible effects
- Fore example, Mytkowicz et al. [2009] showed that link order and environment variable size can significantly affect measurement
- Mainly noticeable in low level languages

# Why care in Python?



# Why care in Python?





- CPython implements a *stop the world* garbage collector

- CPython implements a *stop the world* garbage collector
- Garbage collection can freeze your program at any point

- CPython implements a *stop the world* garbage collector
- Garbage collection can freeze your program at any point
- The `gc` module provides an interface to the garbage collector

Consider performing garbage collection then disabling the garbage collector before taking a measurement:

Consider performing garbage collection then disabling the garbage collector before taking a measurement:

```
>>> import gc  
>>> gc.collect()  
>>> gc.disable()
```

## Warmups & Steady State



- The first values of each worker process are usually slower

- The first values of each worker process are usually slower
- Overhead can vary greatly, especially when using JIT compilers

- The first values of each worker process are usually slower
- Overhead can vary greatly, especially when using JIT compilers
- Many benchmarking suites ignore the first  $n$  values of a run

- The first values of each worker process are usually slower
- Overhead can vary greatly, especially when using JIT compilers
- Many benchmarking suites ignore the first  $n$  values of a run
- Warmup vs steady state is still a work in progress

## Running a single Benchmark

---

# What is a benchmark?

# What is a benchmark?

- A benchmark is a standardised test designed to evaluate and measure the performance of hardware, software, or a combination of both

# What is a benchmark?

- A benchmark is a standardised test designed to evaluate and measure the performance of hardware, software, or a combination of both
- Benchmarks are useful tools for performance analysis



# What is a benchmark?

- A benchmark is a standardised test designed to evaluate and measure the performance of hardware, software, or a combination of both
- Benchmarks are useful tools for performance analysis
- May not always reflect real-world usage scenarios accurately

What's out there?

---



- Inbuilt Python module to measure execution time of small code snippets

- Inbuilt Python module to measure execution time of small code snippets
- Performs multiple runs and repeats of the statement

- Inbuilt Python module to measure execution time of small code snippets
- Performs multiple runs and repeats of the statement
- Returns the average of the minimum time of each run



- Is a toolkit to write, run and analyze benchmarks



- Is a toolkit to write, run and analyze benchmarks
- Automatically calibrate a benchmark for a time budget

- Is a toolkit to write, run and analyze benchmarks
- Automatically calibrate a benchmark for a time budget
- Detect if a benchmark result seems unstable



- Tool for benchmarking Python packages over their lifetime

- Tool for benchmarking Python packages over their lifetime
- Runtime, memory consumption and even custom-computed values may be tracked

- Tool for benchmarking Python packages over their lifetime
- Runtime, memory consumption and even custom-computed values may be tracked
- Used by many FOSS projects e.g. Astropy, NumPy and SciPy

- Tool for benchmarking Python packages over their lifetime
- Runtime, memory consumption and even custom-computed values may be tracked
- Used by many FOSS projects e.g. Astropy, NumPy and SciPy
- <https://asv.readthedocs.io/en/stable/>

# Benchmark Design

---



## Average vs Minimum Time

- *Minimum* time allows us to get a "frictionless model" of performance, this is used by *timeit*

## Average vs Minimum Time

- *Minimum* time allows us to get a "frictionless model" of performance, this is used by *timeit*
- *Average* time allows us to mathematically increase the accuracy of the measure by taking more samples, this is used by *pyperformance*

## Average vs Minimum Time

- *Minimum* time allows us to get a "frictionless model" of performance, this is used by *timeit*
- *Average* time allows us to mathematically increase the accuracy of the measure by taking more samples, this is used by *pyperformance*
- Also, which average do you use?

## Average vs Minimum Time

- *Minimum* time allows us to get a "frictionless model" of performance, this is used by *timeit*
- *Average* time allows us to mathematically increase the accuracy of the measure by taking more samples, this is used by *pyperformance*
- Also, which average do you use?
- There is not yet a consensus on which measure should be used

Can we assume a normal distribution?

# Can we assume a normal distribution?

- Normality is assumed in many benchmarking suites

# Can we assume a normal distribution?

- Normality is assumed in many benchmarking suites
- The distribution of results might not always follow a normal distribution



# Can we assume a normal distribution?

- Normality is assumed in many benchmarking suites
- The distribution of results might not always follow a normal distribution
- In this case we must adopt different statistical tools

# Testing for a normal distribution

# Testing for a normal distribution

- It's essential to inspect the distribution of benchmark

# Testing for a normal distribution

- It's essential to inspect the distribution of benchmark
- Visually "eyeballing" the test

# Testing for a normal distribution

- It's essential to inspect the distribution of benchmark
- Visually "eyeballing" the test
- Visualising with a QQ-plot

# Testing for a normal distribution

- It's essential to inspect the distribution of benchmark
- Visually "eyeballing" the test
- Visualising with a QQ-plot
- Statistical tests can be employed to formally assess the normality assumption, such as the Shapiro-Wilk test

## A simple error check

## A simple error check

- I like to measure the minimum and the average.



## A simple error check

- I like to measure the minimum and the average.
- You can then use the distance between the average and the minimum as an easy-to-measure error metric

## A simple error check

- I like to measure the minimum and the average.
- You can then use the distance between the average and the minimum as an easy-to-measure error metric
- If the average is far from the minimum, then your real-world performance could differ drastically from the minimum

## A simple error check

- I like to measure the minimum and the average.
- You can then use the distance between the average and the minimum as an easy-to-measure error metric
- If the average is far from the minimum, then your real-world performance could differ drastically from the minimum
- see Lemire [2023]



- The standard deviation is greater than 10% of the mean

- The standard deviation is greater than 10% of the mean
- The minimum or the maximum is 50% smaller or greater than the mean

## More error checks

- The standard deviation is greater than 10% of the mean
- The minimum or the maximum is 50% smaller or greater than the mean
- The shortest raw value takes less than 1 millisecond

# Outliers and Perturbing Events



- Perturbing events should be investigated carefully

# Outliers and Perturbing Events

- Perturbing events should be investigated carefully
- It is important to understand *how* it effects the benchmark result

# Outliers and Perturbing Events

- Perturbing events should be investigated carefully
- It is important to understand *how* it effects the benchmark result
- *pyperformance* chooses to include outliers, as it wants to reflect real world usage

# Outliers and Perturbing Events

- Perturbing events should be investigated carefully
- It is important to understand *how* it effects the benchmark result
- *pyperformance* chooses to include outliers, as it wants to reflect real world usage
- Outliers due to perturbing events may or may not be included in your analysis

## Our benchmark

---



- The *pyperformance* project is intended to be an authoritative source of benchmarks for all Python implementations.

- The *pyperformance* project is intended to be an authoritative source of benchmarks for all Python implementations.
- Focus is on real-world benchmarks



- The *pyperformance* project is intended to be an authoritative source of benchmarks for all Python implementations.
- Focus is on real-world benchmarks
- Includes the ability to add your own benchmarks

- The *pyperformance* project is intended to be an authoritative source of benchmarks for all Python implementations.
- Focus is on real-world benchmarks
- Includes the ability to add your own benchmarks
- <https://github.com/python/pyperformance>

# The n-body benchmark

# The n body benchmark

- N-body benchmark from the Computer Language Benchmarks Game

# The n body benchmark

- N-body benchmark from the Computer Language Benchmarks Game
- Model the orbits of Jovian planets, using a simple integrator

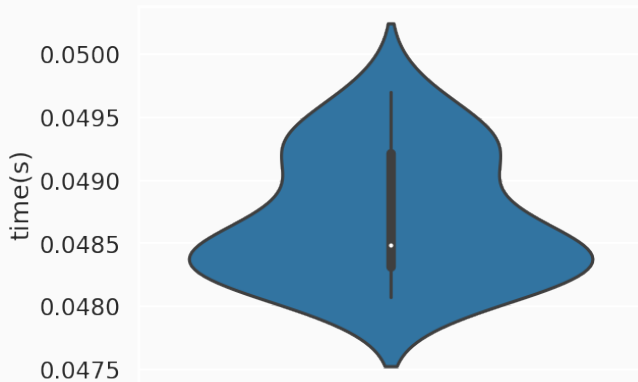
# The n body benchmark

- N-body benchmark from the Computer Language Benchmarks Game
- Model the orbits of Jovian planets, using a simple integrator
- There does not exist an analytical solution

# The n body benchmark

- N-body benchmark from the Computer Language Benchmarks Game
- Model the orbits of Jovian planets, using a simple integrator
- There does not exist an analytical solution
- Microbenchmark on floating point operations

## Eyeballing the distribution



**Figure 3:** violinplot of n-body runtimes (s)



# The summary statistics

count	20
mean	48.706
std	0.495
min	48.071
50%	48.489
max	49.701

**Table 1:** Summary statistics for the n-body benchmark (ms)

## What about our simple error check?

## What about our simple error check?

- Minimum of the runs is 48.071 ms

## What about our simple error check?

- Minimum of the runs is 48.071 ms
- Mean of the runs is 48.706 ms
- ✓ Very close together

## Let's look at our stability checks

## Let's look at our stability checks

- ✓ The standard deviation is 1% of the mean

## Let's look at our stability checks

- ✓ The standard deviation is 1% of the mean
- ✓ The minimum and the maximum are very close to the mean

## Let's look at our stability checks

- ✓ The standard deviation is 1% of the mean
- ✓ The minimum and the maximum are very close to the mean
- ✓ The shortest raw value took 48 milliseconds



# Comparing Benchmarks

---

# Does Windows or Linux run faster?

# Does Windows or Linux run faster?

- Want to compare how a change in the OS affects our runtime performance

# Does Windows or Linux run faster?

- Want to compare how a change in the OS affects our runtime performance
- Ran the benchmark on Linux and Windows

# Does Windows or Linux run faster?

- Want to compare how a change in the OS affects our runtime performance
- Ran the benchmark on Linux and Windows
- Was careful to present a fair and unbiased approach

# Does Windows or Linux run faster?

- Want to compare how a change in the OS affects our runtime performance
- Ran the benchmark on Linux and Windows
- Was careful to present a fair and unbiased approach
- Let's compare the results!

## Looking at the statistics

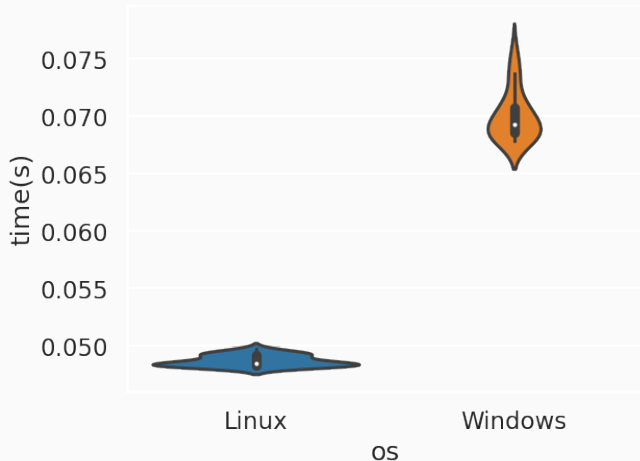
- Linux ran with a mean time of 49 ms



## Looking at the statistics

- Linux ran with a mean time of 49 ms
- Windows ran with a mean time of 70 ms

For the  $n$ -body problem it's obvious...



**Figure 4:** Runtime of the  $n$ -body benchmark

## How comfortable are you saying this speedup is significant?

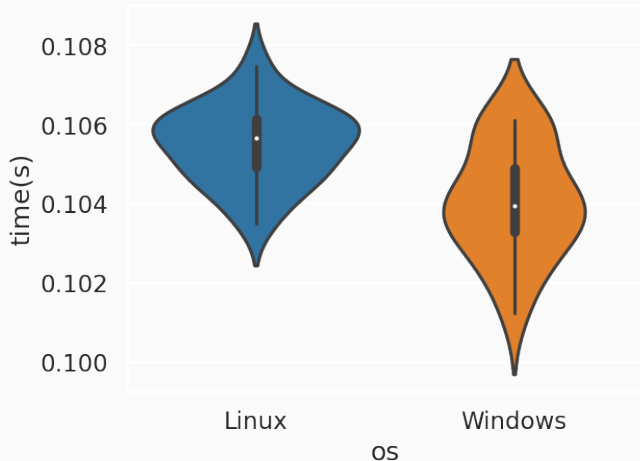


Figure 5: Runtime of the `sympy_sum` benchmark

# Compare distributions, not summaries

# Compare distributions, not summaries

- Sample size and data quality

# Compare distributions, not summaries

- Sample size and data quality
- Distributions capture variability

# Compare distributions, not summaries

- Sample size and data quality
- Distributions capture variability
- Skewness and asymmetry

## References

---

- B. Adelstein-Lelbach. Benchmarking c++ code, 2015. URL <https://youtu.be/zWxSZcpeS8Q?t=534>.
- D. Lemire. Are your memory-bound benchmarking timings normally distributed?, 2023. URL <https://lemire.me/blog/2023/04/06/are-your-memory-bound-benchmarking-timings-normally-distributed/>
- R. J. Light, J. D. Singer, and J. B. Willett. *By Design: Planning Research on Higher Education*. Harvard University Press, 1990.



T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! *SIGARCH Comput. Archit. News*, 37(1):265–276, mar 2009. ISSN 0163-5964. doi: 10.1145/2528521.1508275. URL <https://doi.org/10.1145/2528521.1508275>.