



Inria

An Introduction to Scientific Experimentation and Benchmarking

Anne Auger and Nikolaus Hansen
Inria and CMAP, Ecole Polytechnique, IP Paris



Full set of slides: <http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco-2023-benchmarking-tutorial.pdf>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal

© 2023 Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0120-7/23/07...\$15.00

<https://doi.org/10.1145/3583133.3595064>

Please follow this link to access the last version of the slides :

<http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco-2023-benchmarking-tutorial.pdf>

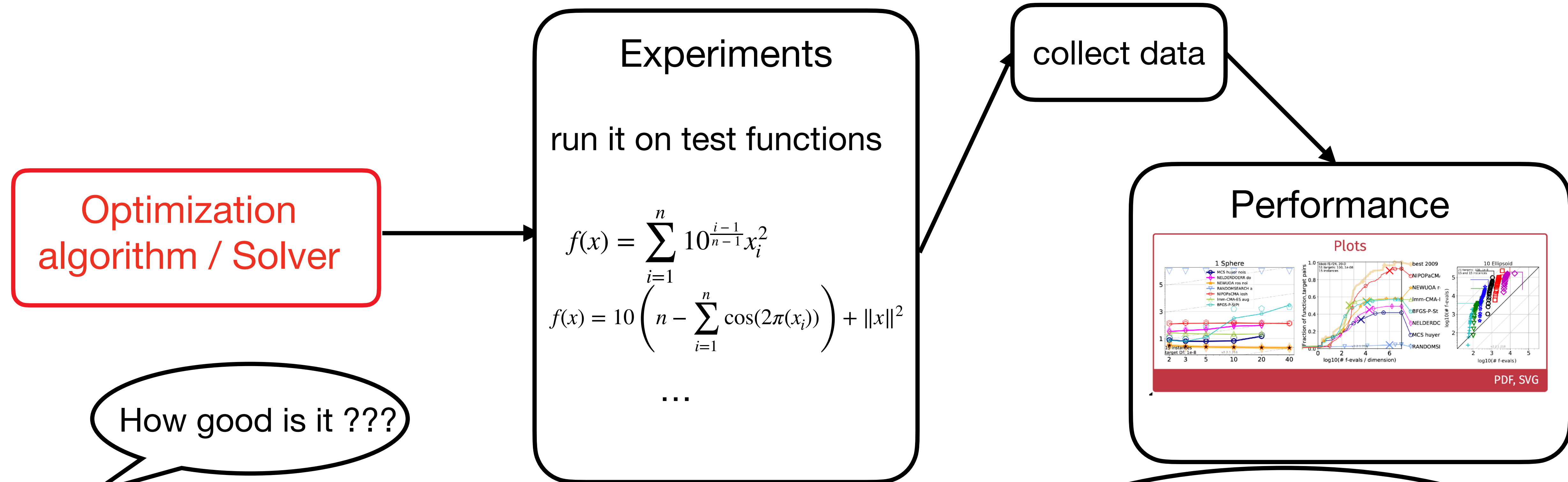
...feel free to ask questions...

...feel free to ask questions...

Scientific Experimentation

- Numerical experiment to answer some (scientific) questions
 - test/validate an hypothesis
 - understand a mechanism
- Often done when designing a new algorithm, but not only

Benchmarking



How good is it ???

Interesting! My algorithm is 10 times faster than CMA-ES on separable functions but 1000 slower on non-separable ones for dimensions between 5 and 40 !



Overview

The first principle is not to fool yourself – and you are the easiest person to fool.

— Richard Feynman

- I Scientific experimentation
 - demo in Python
- II Benchmarking

Why Experimentation?

- The behaviour of many if not most **practically relevant algorithms** is
 - not **amenable** to a (full) theoretical analysis even when applied to simple problems
 - \mathcal{O} -results are not sufficient in practice
 - often, complex algorithms are truthfully represented *only* by source code implementation details matter (at least sometimes)
 - hence we need an *alternative to theory* for investigation
 - not **comprehensible** or predictable without (extensive) empirical examinations
 - even on simple problems (the algorithm is the only source of complexity)
 - comprehension is the main driving force for scientific progress
 - If it disagrees with experiment, it's wrong. [...] And that simple statement is the key to science. — R. Feynman*
- Virtually all algorithms have **parameters**
 - like most (physical/biological/...) models in science
 - we rarely have explicit knowledge about the “right” choice
 - prevent overfitting is a particular challenge
 - this is a *big* obstacle in designing and benchmarking algorithms
- We are interested in solving *black-box* optimisation problems
 - which may be “arbitrarily” complex and (by definition) not well-understood

Why is Experimentation Important?

“In the course of your work, you will from time to time encounter the situation where the facts and the theory do not coincide. In such circumstances [...], it is my earnest advice to respect the facts.”

— Igor Sikorsky

“If it disagrees with experiment, it's wrong. And that simple statement is the key to science. [...] That's all there is to it.”

— Richard P. Feynman
<https://youtu.be/b240PGCMwV0>

Scientific Experimentation in a Nutshell

- start from a (good) **scientific question**
- design an **experiment** to test (try to “falsify”/“rule out”/render unlikely) one or several answers
- **visualize** (all) what make sense and interpret
- iterate ...

Effective scientific experimentation requires a healthy mixture of **creativity** and technique, comparable to the arts.

Scientific Experimentation (dos and don'ts)

- What is the aim? *Answer a question*, ideally quickly (minutes, seconds) and with little ambiguity
 - consider in advance what the question is and in which way the/an experiment could answer the question
- practice to **make quantitative predictions** of the possible/expected outcomes
 - to develop a mental model of the object of interest
 - quantitative in a) effect size and b) error probability of the prediction
 - to practice to make the clear distinction between a guess from intuition and observations
 - to practice being proven wrong by observations and overcoming *confirmation bias*
- do not (blindly) trust in code, claims, ... that you rely upon without *good* reasons
 - check/test “everything” yourself, practice stress testing (e.g. a weird parameter setting) which (also) boosts understanding
 - this is a key element for success
 - interpreted/scripted languages have an advantage (quick test of code snippets)
 - Why Most Published Research Findings Are False* [Ioannidis 2005]
- run **rather many than fewer (different) experiments iteratively**, practice **online experimentation** (see demonstration)
 - to run many experiments they must be *quick to implement and run*, ideally **seconds** rather than minutes (start with small dimension/budget)
 - develops a sense for the effect of setup changes*

Scientific Experimentation (dos and don'ts)

- run any experiment at least **twice**
 - assuming that the outcome is stochastic
or with a different initialization
 - thereby getting an **estimator of variation/dispersion/variance**
- **display**: *the more the better, the more polished the better*
 - figures are *intuition pumps* (not only for presentations or publications)
 - it is hard to overestimate the value of a good figure
 - data is the *only* way experimentation can help to answer questions,
therefore **look at the data, stare at them, study them carefully!**
- **don't** make **minimising CPU-time** a primary objective
 - avoid spending time in implementation *details* to tweak performance
 - prioritize code clarity (minimize time to *change* code, to debug code, to maintain code)
 - yet code optimization may be necessary to run experiments efficiently
- *Testing Heuristics: We Have it All Wrong* [Hooker 1995]
 - “The **emphasis on competition** is fundamentally anti-intellectual and does not build the sort of insight that in the long run is conducive to more effective algorithms”*

Scientific Experimentation (dos and don'ts)

- It is usually (much) more important to **understand why** algorithm A performs badly on function f , than to make algorithm A faster (for unknown, unclear or trivial reasons)
 - mainly because an algorithm is applied to *unknown* functions, not to f , and the “why” allows to predict the effect of design decisions
- Remain aware: many **devils are in the details**, results or their interpretation may crucially depend on (simple or intricate) subtleties or bugs
 - yet another reason to run many (slightly) different experiments
check limit settings to give consistent results
- **Invariance** is a very powerful, almost indispensable tool

Scientific Questions and Objectives (**bad** and good)

- Invent a new algorithm that is **better than a well-established one, say outperforms an algorithm developed for 20 years** (e.g. CMA-ES) in the domain where it excels (unconstrained, difficult problems)
 - likely to be **very difficult, not very specific** on the class of problems (where do we start?), given the algorithm framework (comparison-based, derivative-free) isn't the algorithm already close to optimal performance?
- Have a CMA-ES version with a mixture of Gaussians to perform well on multimodal functions
 - not very specific**: which functions do we want to solve better ? **Why do we expect to solve problems better** than CMA-ES with restarts? **How can the algorithm scale** with the number of local optima? why should it help to maintain several modes to approach a global optimum?
- what people do afterwards: do not compare with baseline (algorithm with restart), test on a few functions with few local optima
- This **new concept** used in physics (or ...) is **fancy**, many famous people use it, I want to write a paper using that in our domain as **it will look good and cool**

Scientific Questions and Objectives (bad and good)

- I want to understand why the algorithm fails in solving this problem
 - specific, leads to subquestions** like what is the difficulty of the problem, can another method solve it ...
 - understanding why it fails is often the first step to solve the problem
- I want to design a large-scale CMA-ES variant scaling linearly that solves problem with sparse structure
 - not widely explored**, possible to identify classes of problems to experiment with
- I want to know how this algorithm scales with the dimension to solve this class of problems
 - specific**, easy to design an experiment that will give a clear answer
- How do I add errors bars that are meaningful when plotting ECDF graphs?
 - specific, unexplored, useful...**

Desirable properties of scientific questions

What are good scientific questions / objectives:

- question/objective should be **solvable** (e.g., an algorithm is *not* already close enough to its optimal performance)
- **realistic**
- have a mechanism why it could succeed / be answered (before to start experimentation)
- not too general / specific enough
- parts can be answered (quickly)
- related to an unexplored subdomain
- not motivated by fashion
- relate to **improve methods where it is needed**

- Demo on scientific experimentation

What did we illustrate - Take home messages

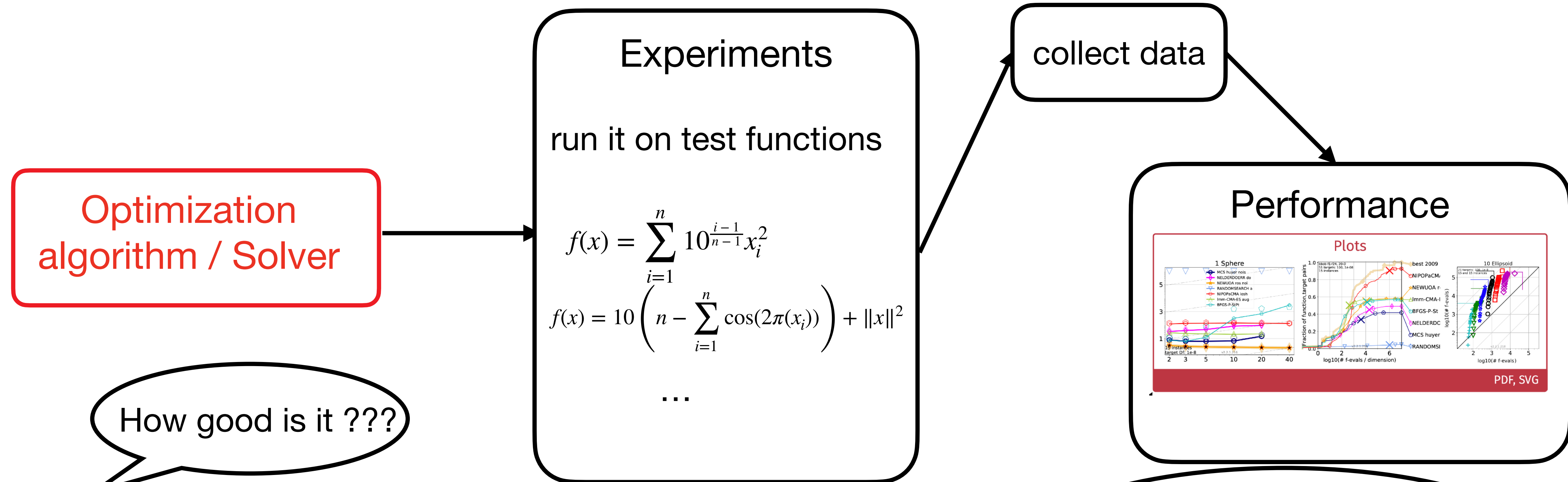
- Interpretation of data can be difficult
- trying various ideas may be necessary

Overview

The first principle is not to fool yourself – and you are the easiest person to fool. Richard Feynman

- I Scientific experimentation
 - demo in Python
- II Benchmarking

Benchmarking



How good is it ???

Interesting! My algorithm is 10 times faster than CMA-ES on separable functions but 1000 slower on non-separable ones for dimensions between 5 and 40 !



About Generalization

Does benchmarking make sense at all?

After all there is no free lunch, right? Or is there?

- A benchmark should attempt to **model commonplace and relevant** “real-world” optimization **problems**

*The set of all observable optimization problems is WAY smaller than most sets of **mathematically constructible** problems.*

*NFL theorems hold on sets of functions that are “closed under permutation”.
Whether all functions in such set are (equally often) observed in reality is **an empirical question**.
Practical evidence suggests: some algorithms are vastly worse than others.*

- The function or instance ID can not be input to the algorithm

*We shall not set algorithm parameters depending on each function!
AKA overfitting.*

*The benchmarking setup: an algorithm that needs to repeatedly solve “new” problems or instances.
possible but not recommended: Crafting Effort correction for using different parameter settings on different functions¹.*

- **Invariance** of algorithms is a relevant aspect to interpret (generalizability of) benchmarking results

- Comparable data

*depends on the benchmarking setup
across publications
across functions (e.g. speedup factor)*

¹: Price KV. Differential evolution vs. the functions of the 2nd ICEO. In Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97) 1997 (pp. 153-157). IEEE.

What about Competitions?

“The emphasis on competition is fundamentally anti-intellectual and does not build the sort of insight that in the long run is conducive to more effective algorithms”.

Hooker (1995) distinguishes “scientific testing” from “competitive testing” in *Testing Heuristics: We Have it All Wrong*.

Miscellaneous

- A trivial (serial) algorithm portfolio: K algorithms can solve each and every problem as fast as the fastest of these algorithms multiplied by K .
Run in parallel, they become as fast as the fastest algorithm
- What differences are we interested in?
2%, 20%, 200%, 2000%,...
- Function/problem *instances*
versus different functions
- Search domain: discrete and continuous
Examples come from the continuous domain.

(Specific) Goals of Benchmarking

We may think of benchmarking as **measuring algorithm performance in a systematic and standardized way**

thereby creating a performance “profile” of an algorithm for a standardized assessment and for simplified comparison

Specific goals can be:

1. Comparing against the “state-of-the-art” or against a baseline

any comparison between two or more algorithms

2. Understanding algorithms

benchmarking usually raises questions, dedicated experimentation is often necessary to answer them

3. Predict performance on real-world problems

4. Selecting algorithms to solve a given problem

5. Regression testing after changes of an algorithm or an implementation

6. Running a competition

a competition setup needs to hide information from the competitor/experimenter

“Everybody” has to do it and it is tedious: choosing (and implementing) problems, performance measures, visualization, statistical tests, ...

which suggests to consider using tools

Benchmarking How To: The Global Picture

Two *surprisingly* (but not completely) *independent* components:

- **Which benchmark, which suite of functions/problems** do we run the algorithms on?
For example and in particular, which collection of test problems?
- **How to assess performance?**
 - experimental setup
 - data collection
 - measures used and presented

COCO/BBOB: The Global Picture

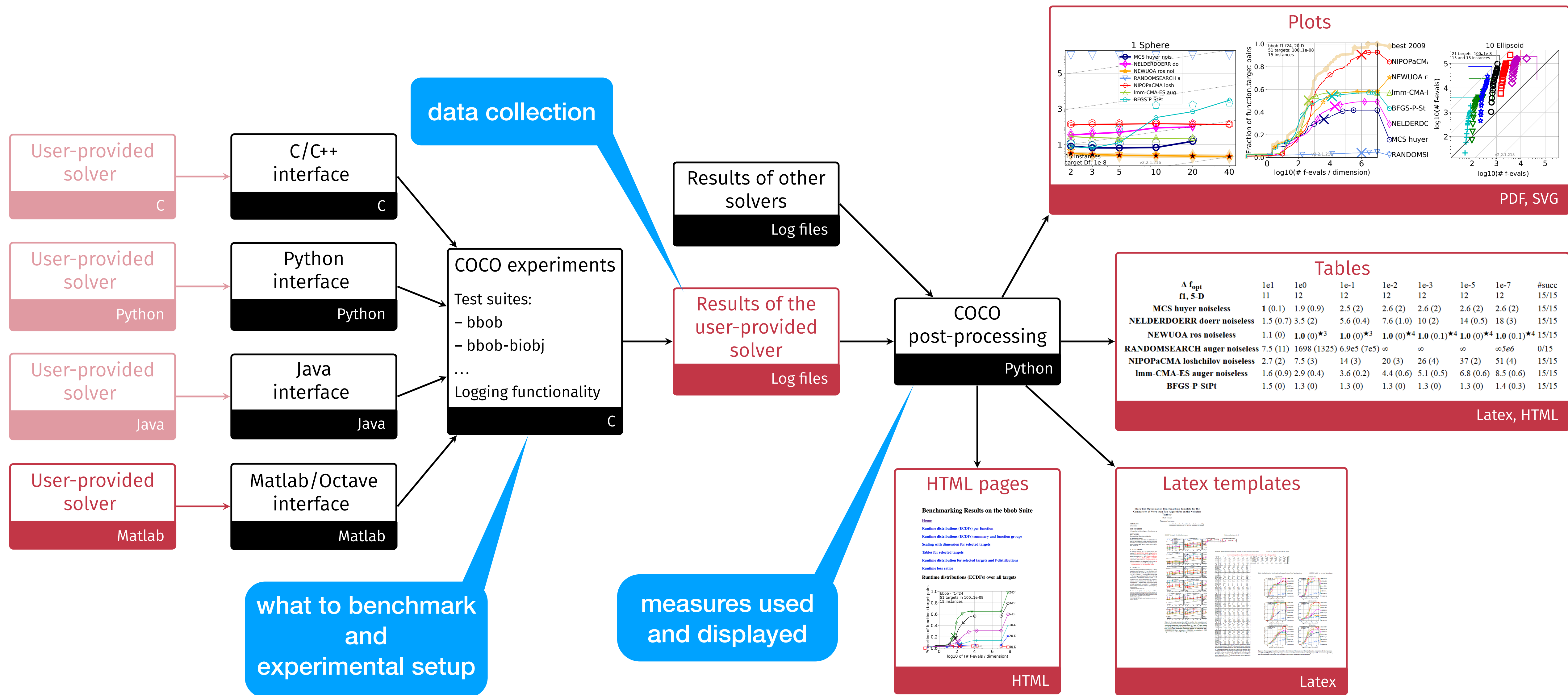


Figure by Tea Tušar in Hansen et al (2021), COCO: A platform for comparing continuous optimizers in a

What is the Benchmark?

Choice of Test Problems

What to Benchmark?

Furious activity is no substitute for understanding (H.H. Williams)

- Taking all possible functions from a repository?
- Bad idea if
 - function difficulties are **unbalanced**
too many small dimensional problems, convex problems...
 - and performance is **aggregated**
- Leads to **bias** in the performance assessment

What to Benchmark?

- test functions should be representative of difficulties we want to test
 - therefore NFL has no relevance as assumption of being closed under permutation has no relevance wrt real world problems*
- related to real-world difficulties
 - for performance to be generalizable to RW*
- scalable
 - dimension plays a big role in performance*
curse of dimensionality
- comprehensible but not too easy
 - BB optimization does not mean BB benchmarking*
- we should still hide properties from the solver (hide optimum, ...)
 - solvers should not be able to exploit the benchmark intentionally or not*

Experimental Setup

- should allow as many **algorithm types/interfaces** as possible
bounded, unbounded, different input options, deterministic, randomized,...
- defines the **information** an algorithm is allowed to use
*search domain (and hence dimension), initial solution,
regions of “interest”, function as back-box
not: function name/ID*
- pure repetitions only work for randomized algorithms
- should define **what is recorded** to afterwards measure performance
- may define **a budget** (or not)
anytime vs targeted budget

Handling and Displaying Empirical Data

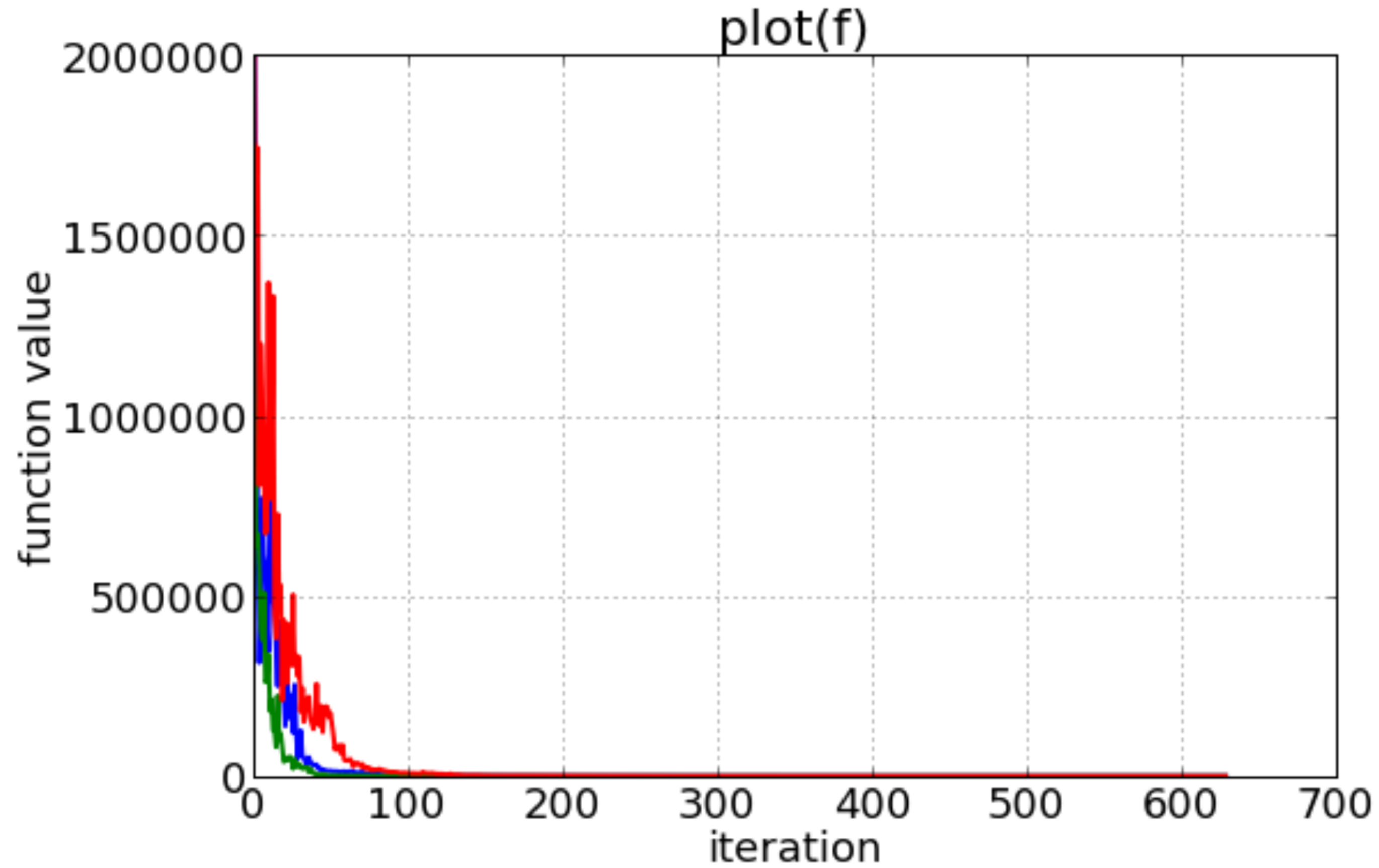
Displaying (Performance) Results

Empirically

convergence graphs is all we have to start with

the right presentation is important!

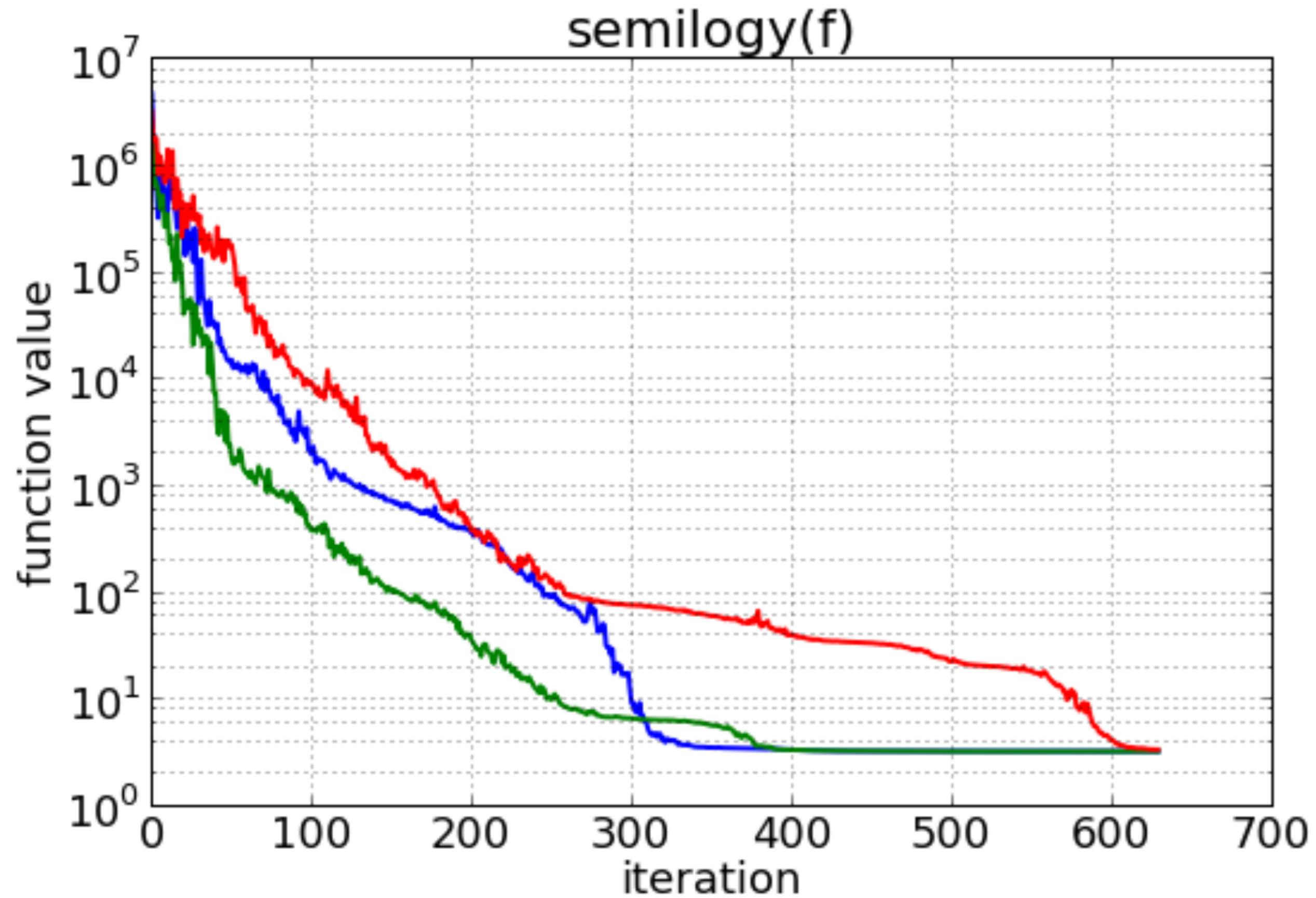
Displaying Three Runs



not like this (it's unfortunately not an extremely uncommon picture)

why not, what's wrong with it?

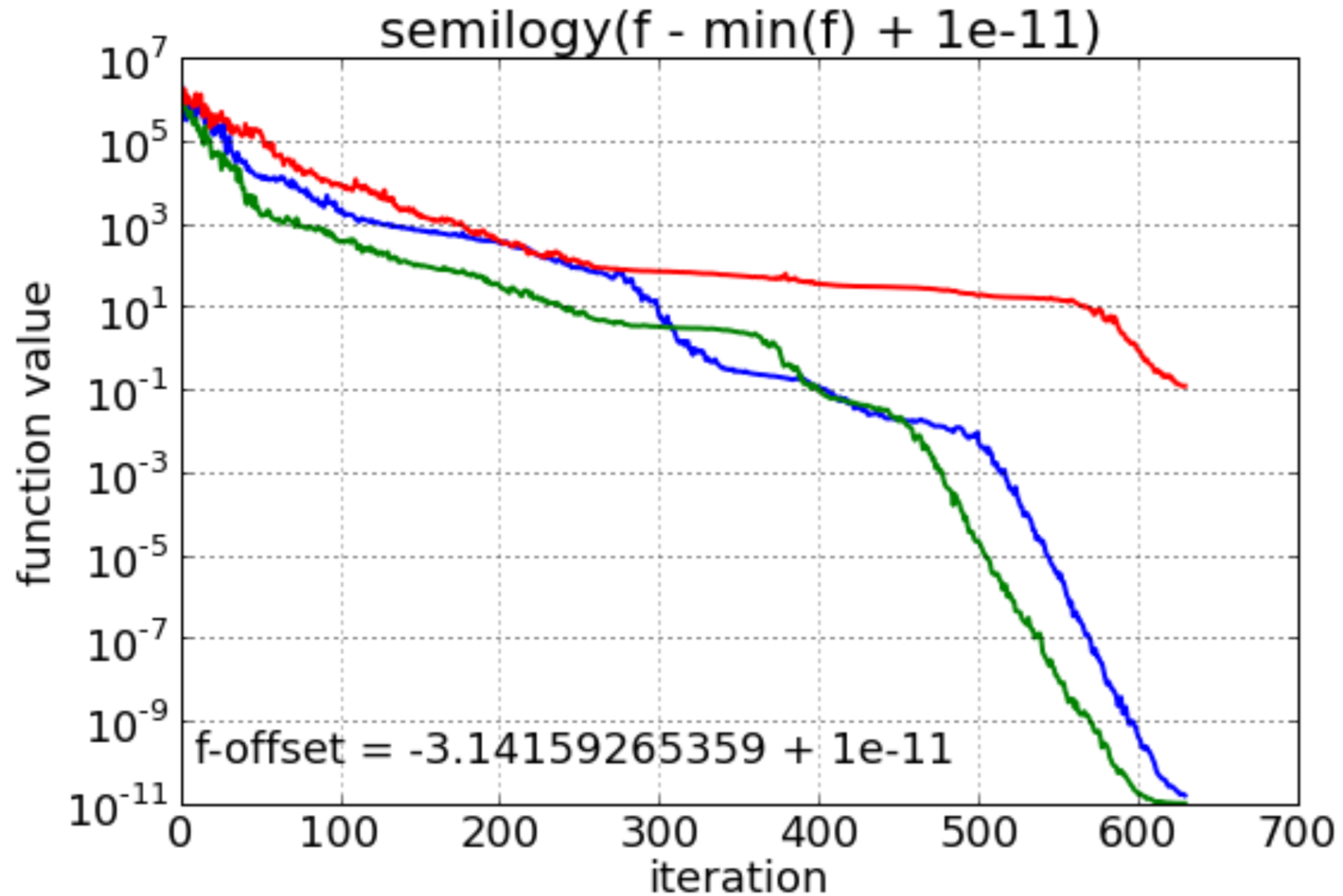
Displaying Three Runs



better like this (shown are the same data),

caveat: fails with negative f-values

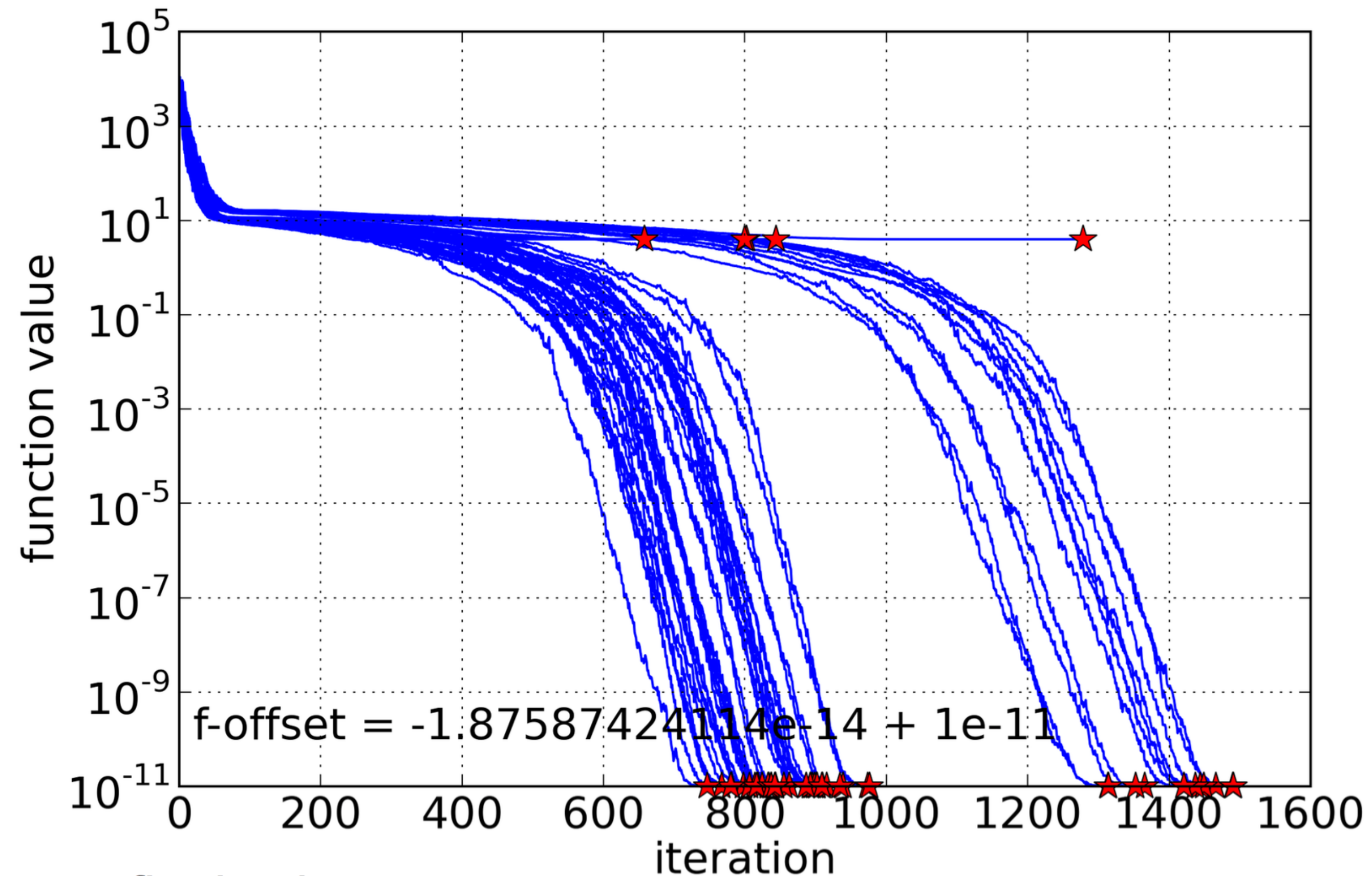
Displaying Three Runs



even better like this: subtract minimum value over all runs

Displaying 51 Runs

don't hesitate to display all data (the appendix is your friend)



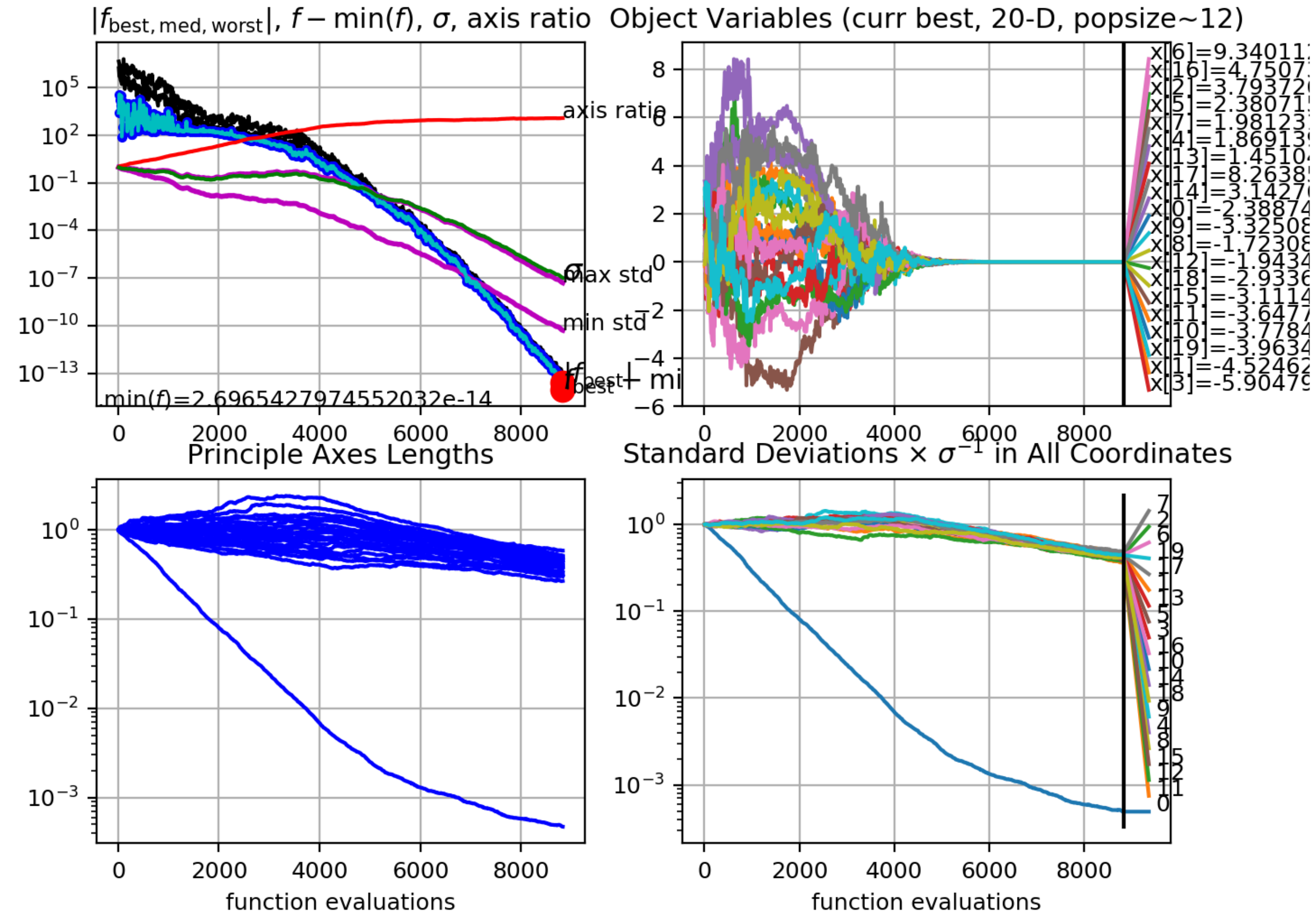
★ : final value

observation: three different "modes", which would be difficult to represent or recover in single statistics

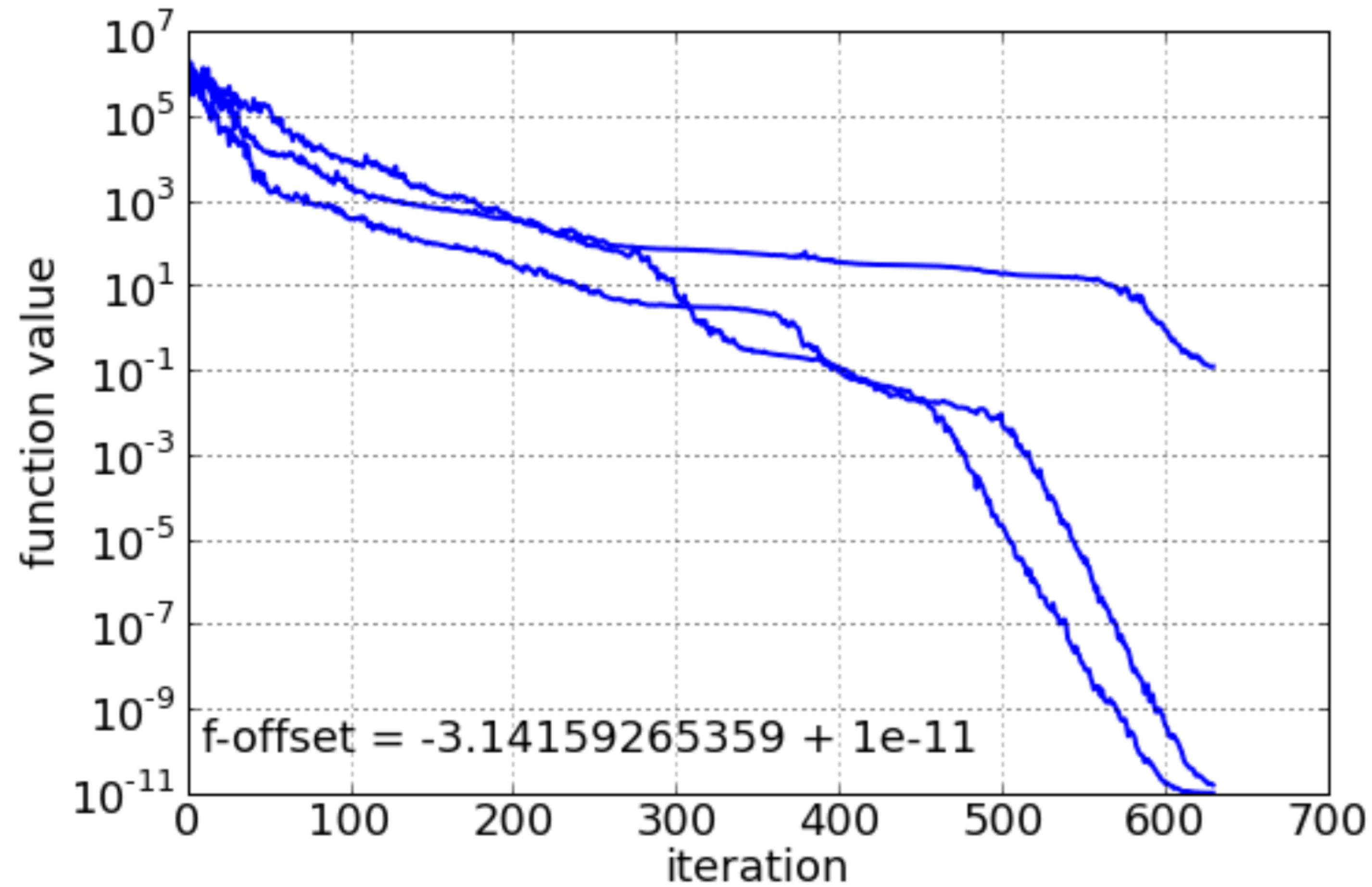
There is more to display than convergence graphs

cma.plot()

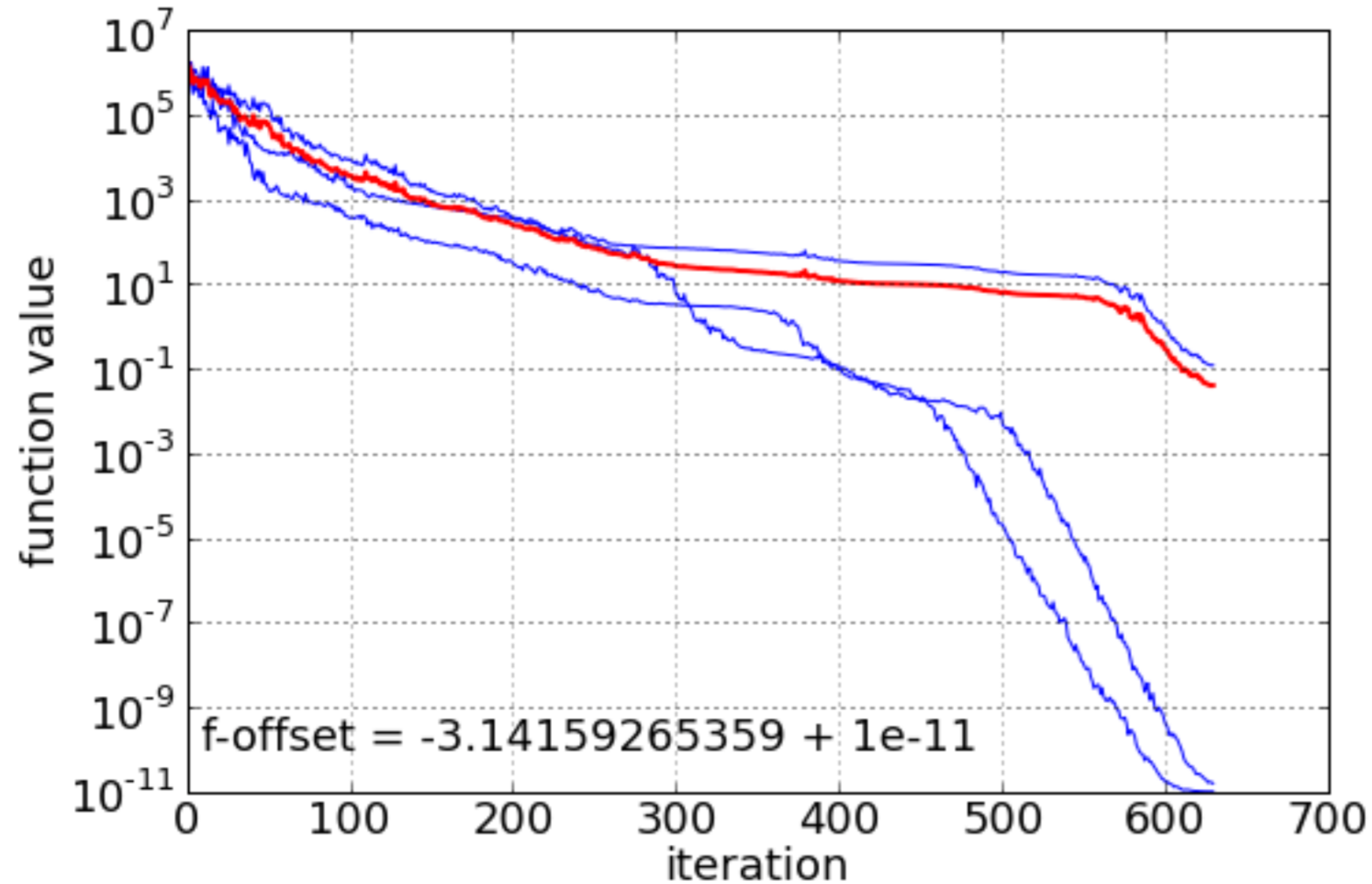
Figure 328



Aggregation: Which Statistics?



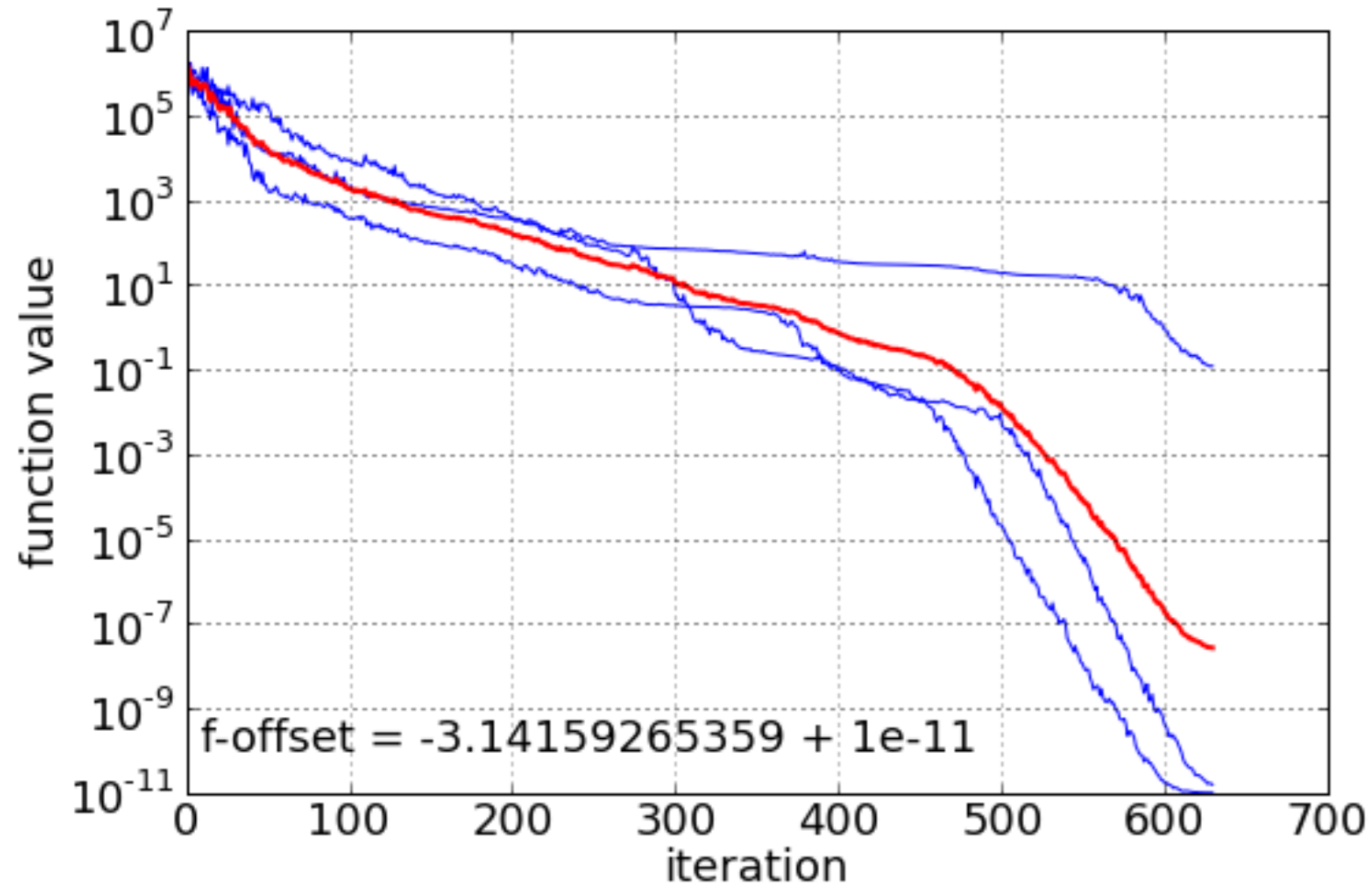
Aggregation: Which Statistics?



mean/average function value

- tends to emphasize large values

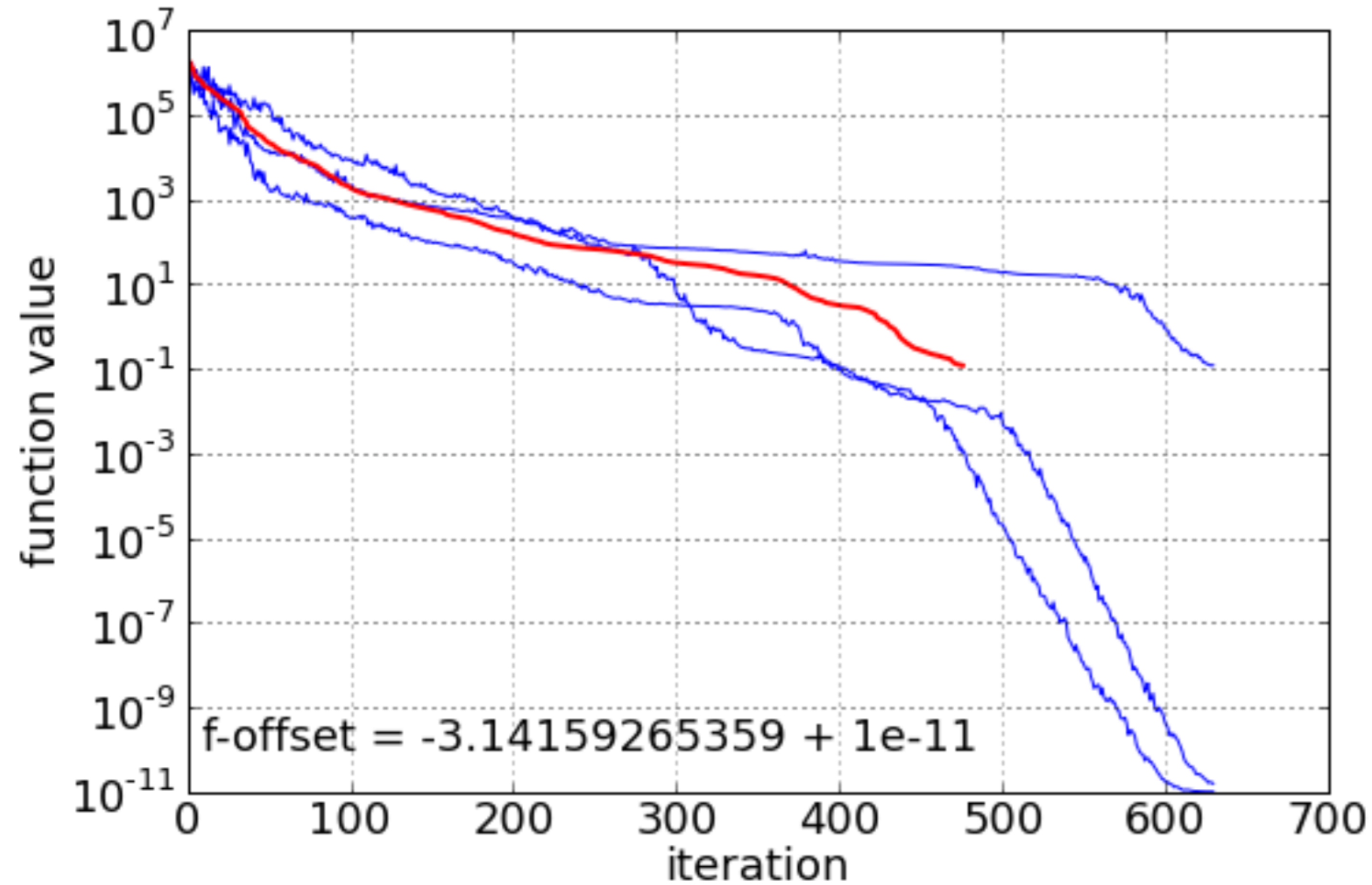
Aggregation: Which Statistics?



geometric average function value $\exp(\text{mean}_i(\log(f_i)))$:

- reflects "visual" average
- depends on offset

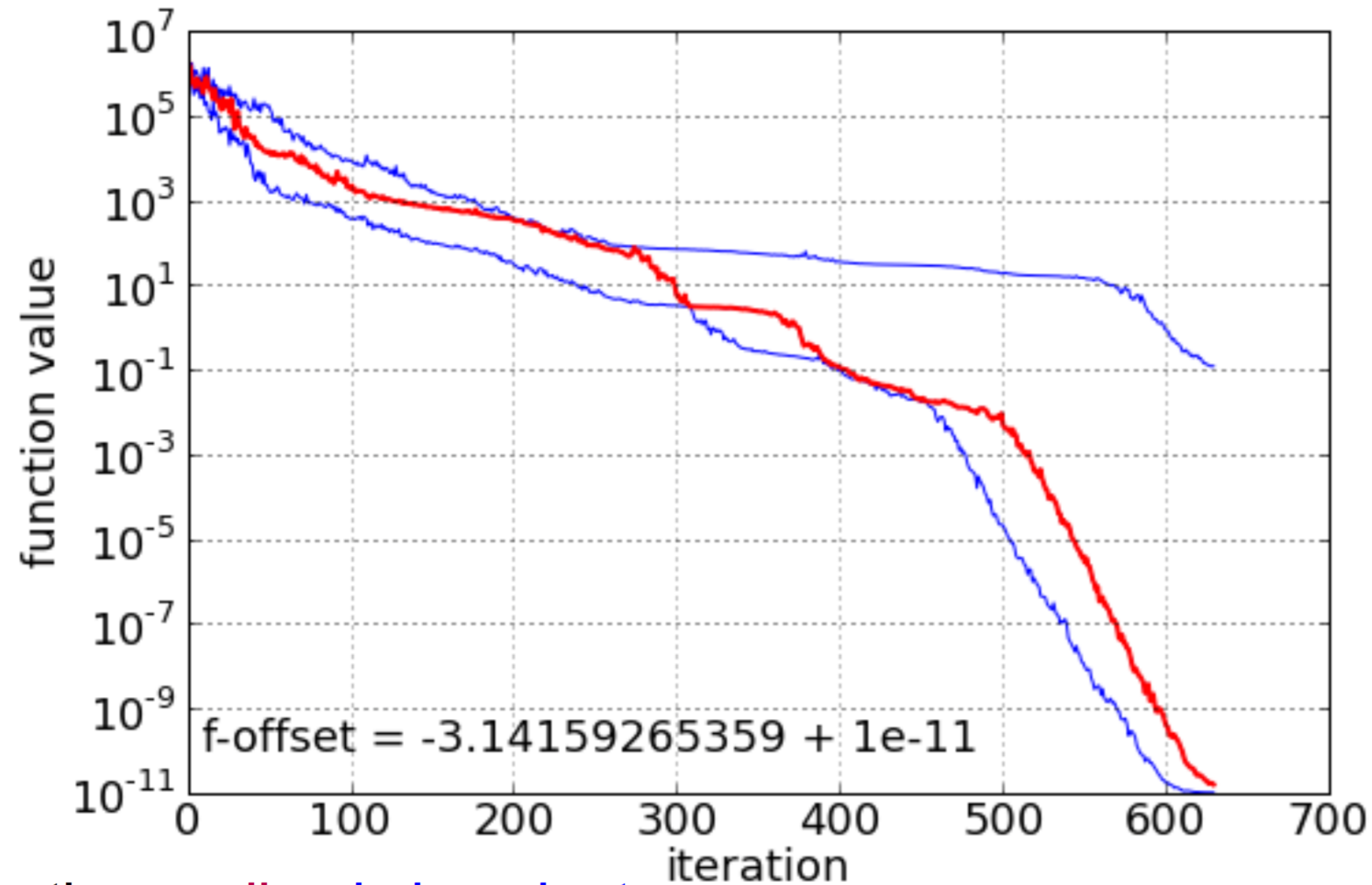
Aggregation: Which Statistics?



average iterations

- reflects "visual" average
- here: incomplete

Aggregation: Which Statistics?



the **median** is invariant

- unique for uneven number of data
- independent of log-scale, offset...

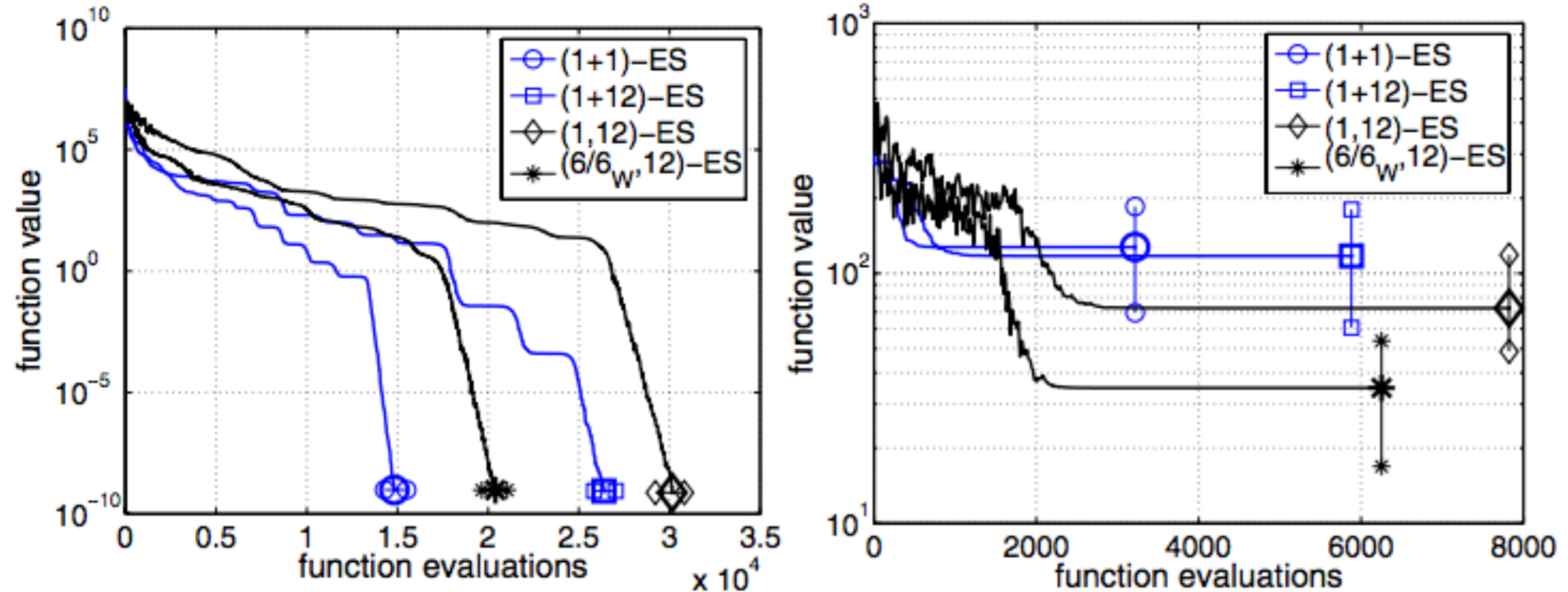
$$\text{median}(\log(\text{data})) = \log(\text{median}(\text{data}))$$

- same when taken over x- or y-direction

Implications

- preferably, use the **median** as summary datum
 - unless there are good reasons for a different statistics
 - out of practicality: use an odd number of repetitions
- more general: use quantiles as summary data
 - for example out of 15 data: 2nd, 8th, and 14th value represent the 10%, 50%, and 90%-tile

Two More Examples



Comparison of 4 algorithms using the "median run" and the 90% central range of the final value on two different functions (Ellipsoid and Rastrigin)

caveat: this range display with simple error bars fails, if, e.g., 30% of all runs "converge"

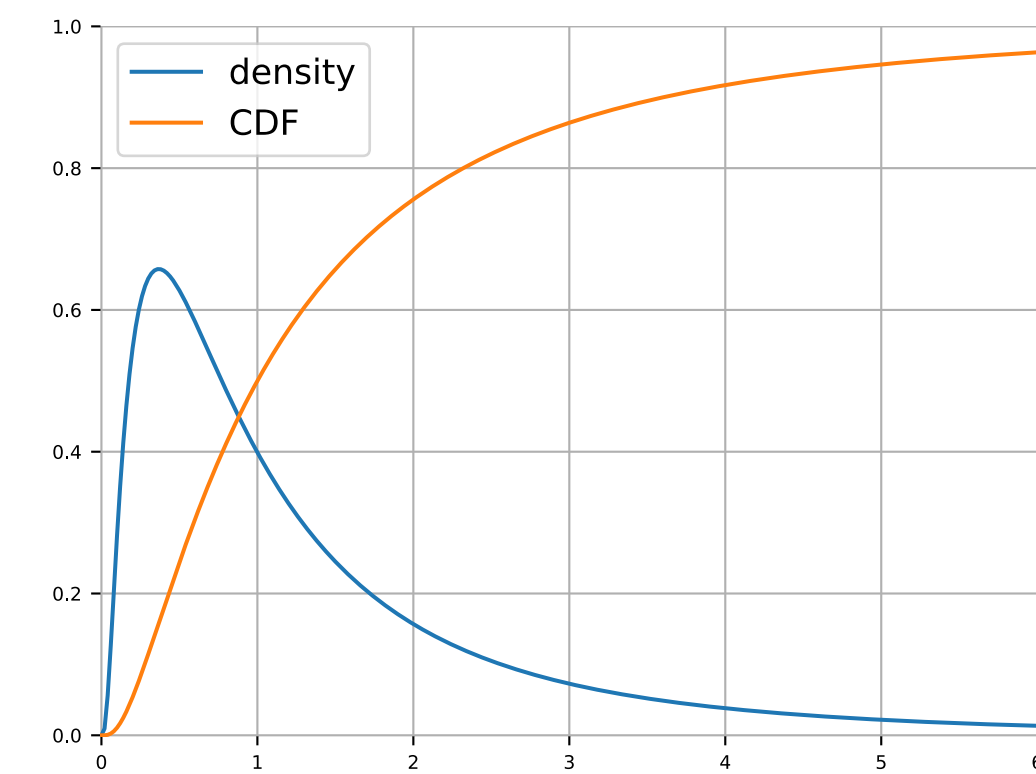
Cumulative Distribution Function (CDF)

Given a random variable T , the cumulative distribution function (CDF) is defined as

$$\text{CDF}_T(t) = \Pr(T \leq t) \text{ for all } t \in \mathbb{R}$$

It characterizes the probability distribution of T

If two random variables have the same CDF, they have the same probability distribution



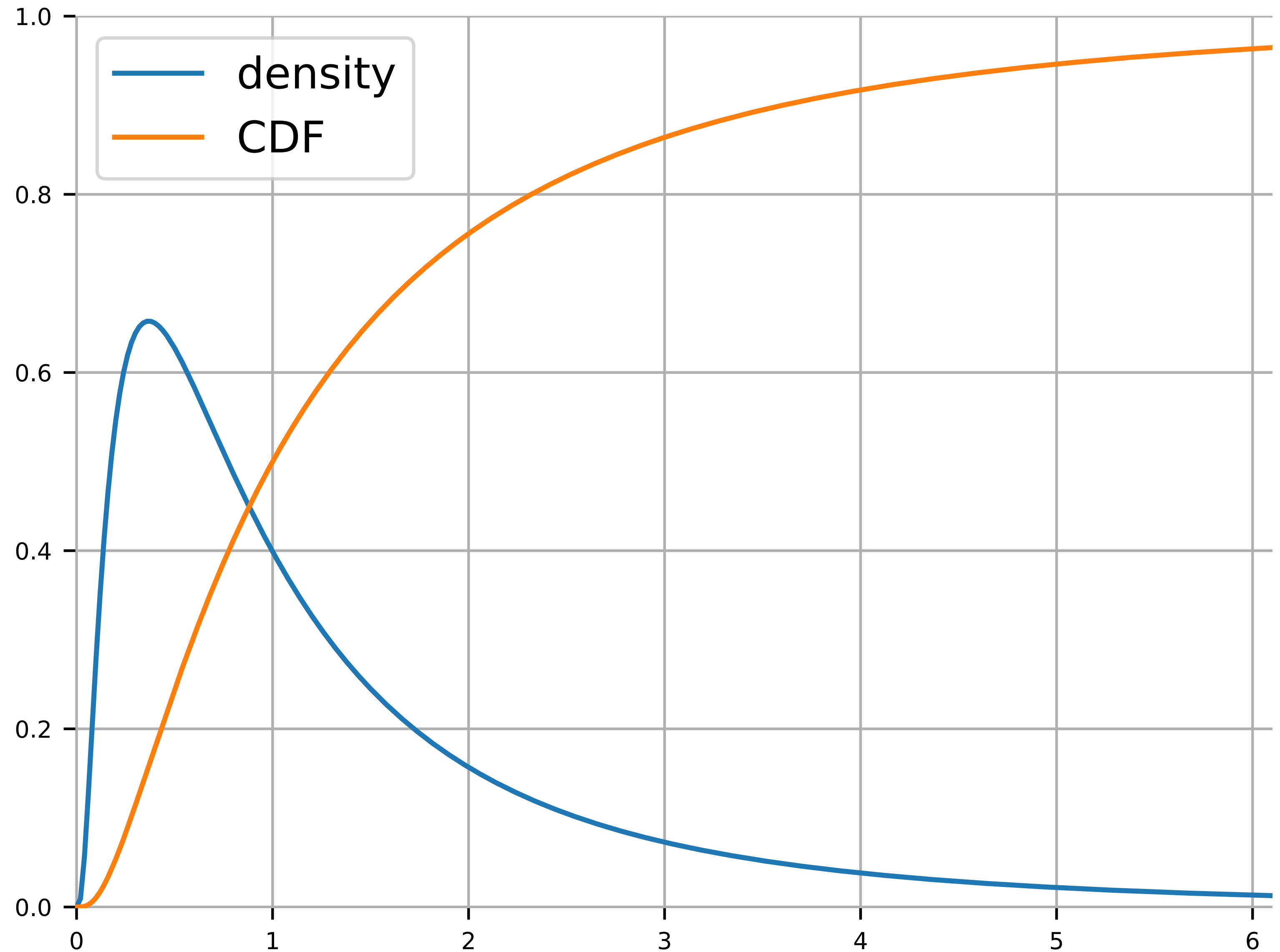
Cumulative Distribution Function (CDF)

Given a random variable T , the cumulative distribution function (CDF) is defined as

$$\text{CDF}_T(t) = \Pr(T \leq t) \text{ for all } t \in \mathbb{R}$$

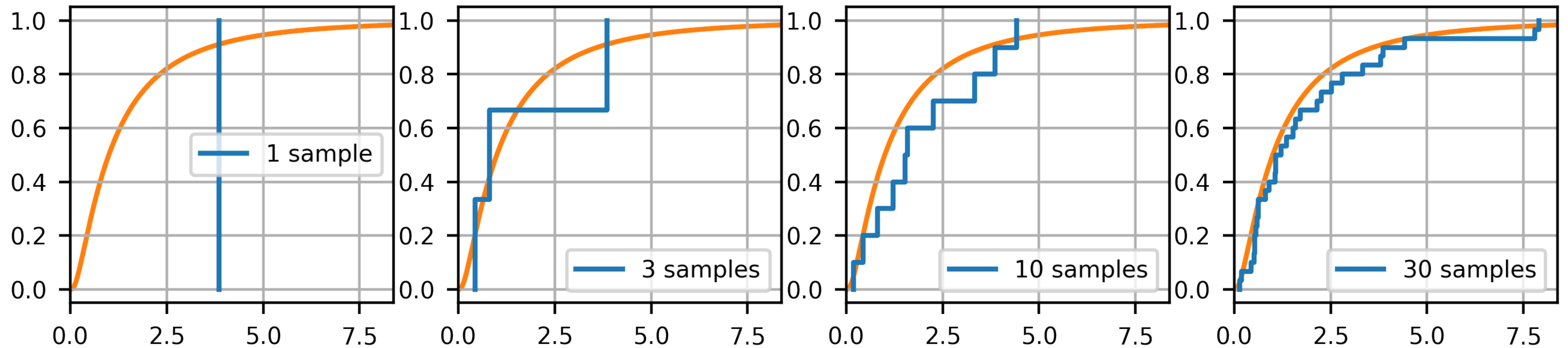
It characterizes the probability distribution of T

If two random variables have the same CDF, they have the same probability distribution



Empirical Cumulative Distribution Function

- Given a collection of data T_1, T_2, \dots, T_k (e.g. an empirical sample of a random variable) the *empirical* cumulative distribution function (ECDF) is a step function that jumps by $1/k$ at each value in the data.



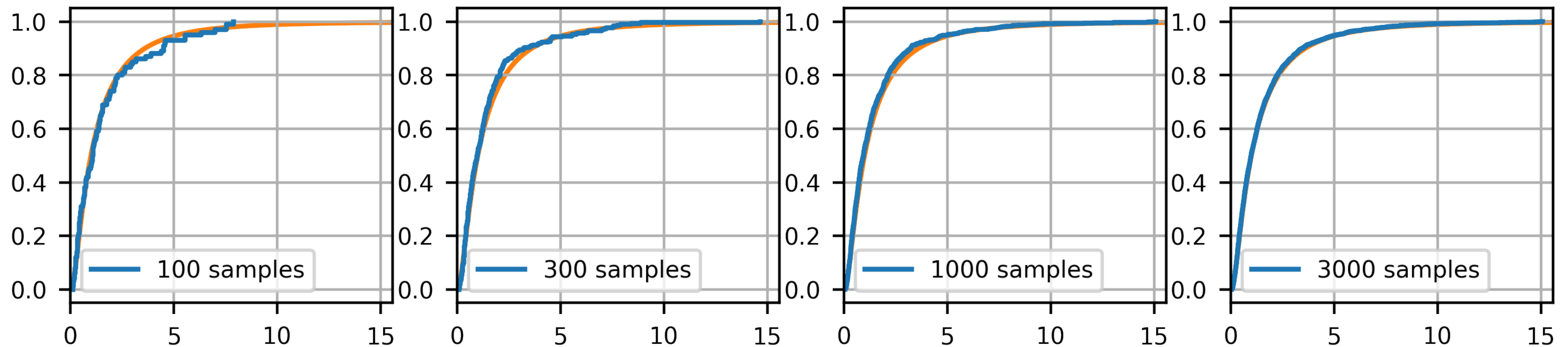
- It is an estimate of the CDF that generated the points in the sample.

Empirical Cumulative Distribution Function

$$\text{ECDF}_{(T_1, \dots, T_k)}(t) = \frac{\text{number of } T_i \leq t}{k} = \frac{1}{k} \sum_{i=1}^k \mathbf{1}_{\{T_i \leq t\}}$$

For $\{T_i : i \geq 1\}$ i.i.d. realization of a random variable T , by the LLN

$$\text{ECDF}_{T_1, \dots, T_k}(t) \xrightarrow[k \rightarrow \infty]{} \text{CDF}_T(t) \text{ a.s. for all } t$$



On Performance Measure

- When comparing algorithms:

- ➔ Algorithm A is better than Algorithm B?

we want more than that

- ➔ Algorithm A is 100 times faster than Algorithm B

*We want **quantitative** statements*

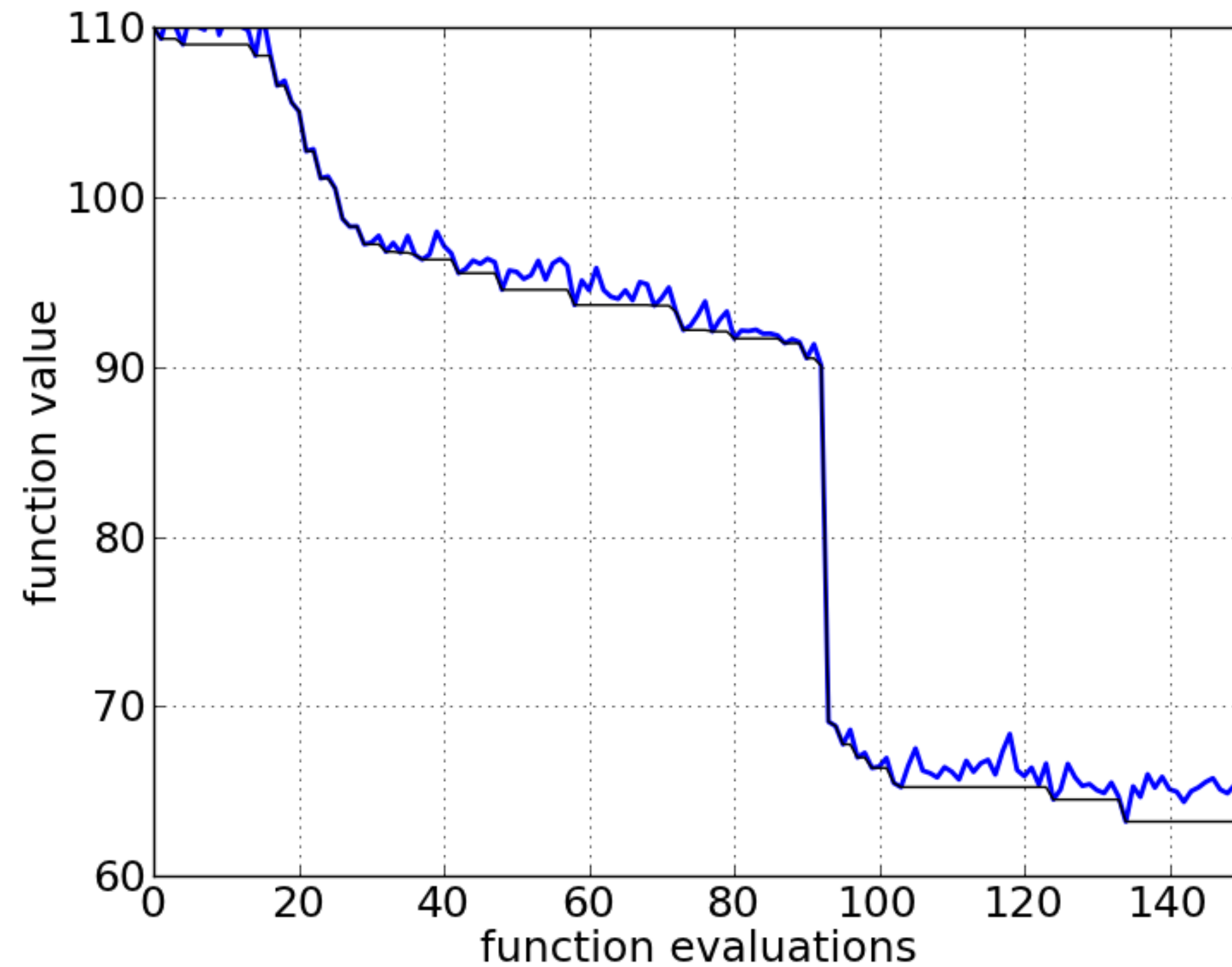
- Requires

- ➔ adequate **performance measure**

- ➔ adequate **data collection**

Collecting Empirical Data

Convergence Graphs is All We Have

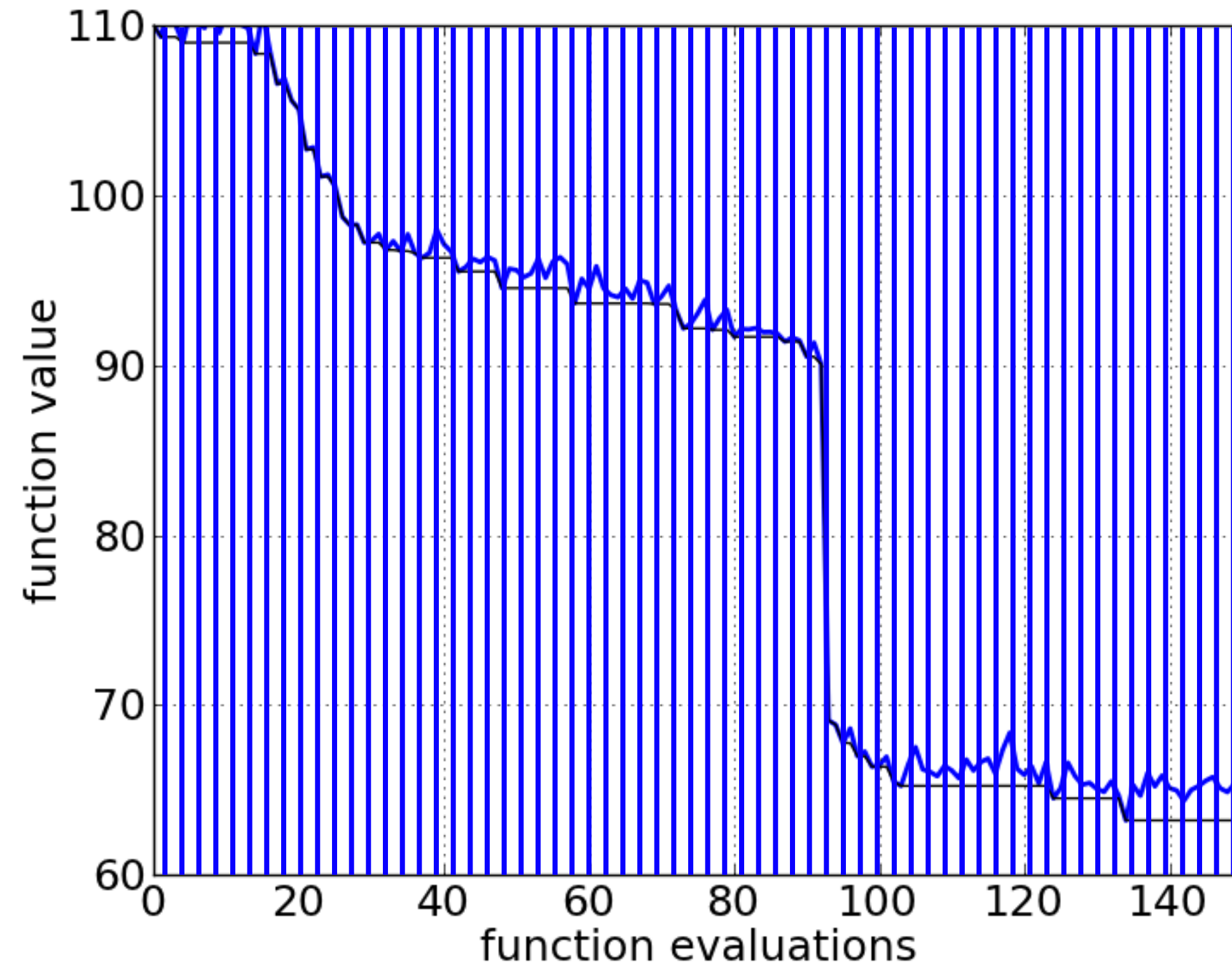


- a convergence graph
- lower envelope (a monotonous graph), best so-far solution

We measure #fevals: quantitative, comparable across papers

using the lower envelope is a practical choice that relates to the first hitting time

Discretization: Two Possibilities



- a convergence graph
- lower envelope (a monotonous graph), best so-far solution

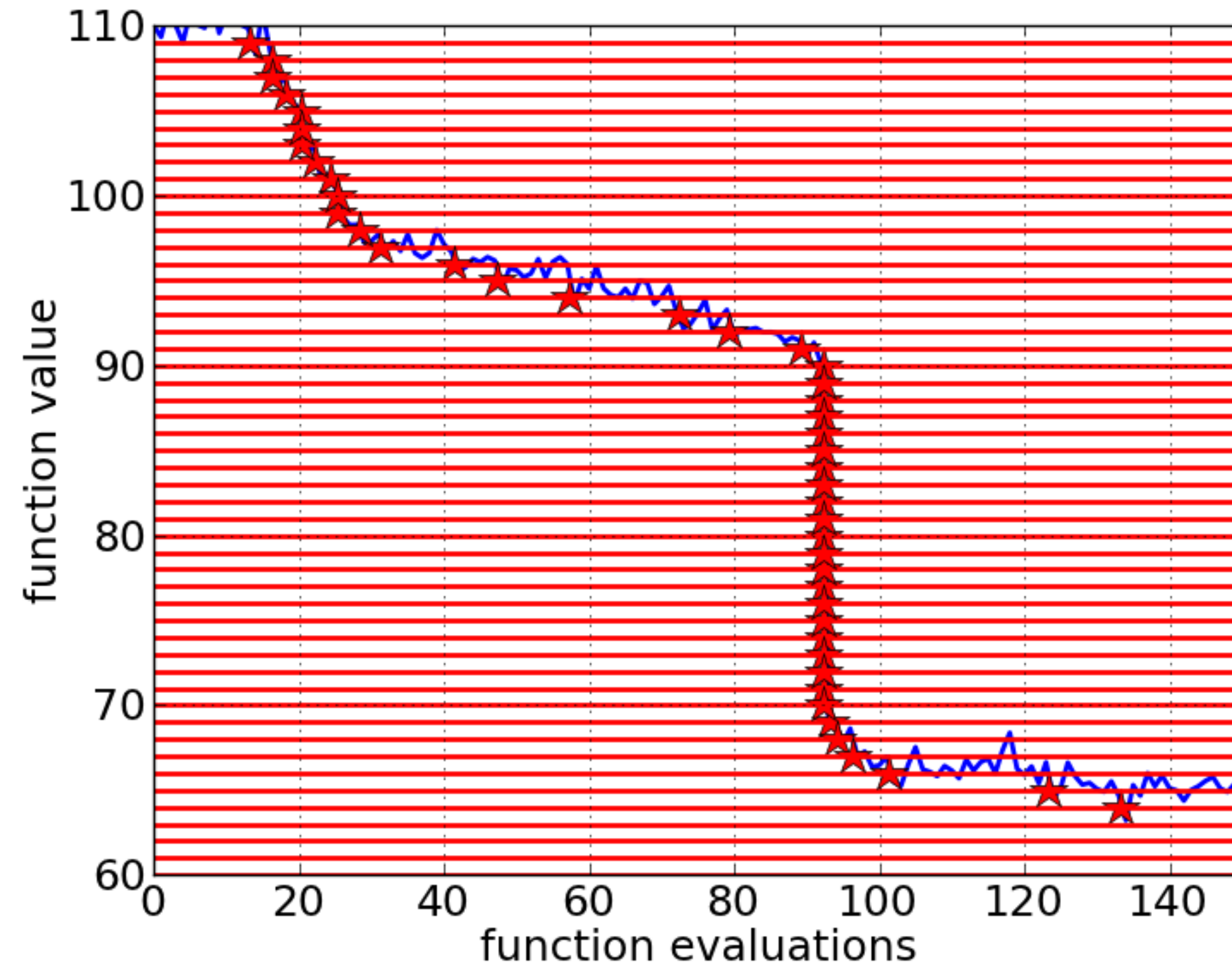
- **vertical:** by evaluation is a natural discretization

for wall clock or CPU time we would need to determine discretization intervals

- evaluations are the independent variable

function value is the dependent variable, the measurement

Discretization: Two Possibilities



- a convergence graph
- lower envelope (a monotonous graph), best so-far solution

- **horizontal:** not a “natural” discretization

we need to determine discretization intervals

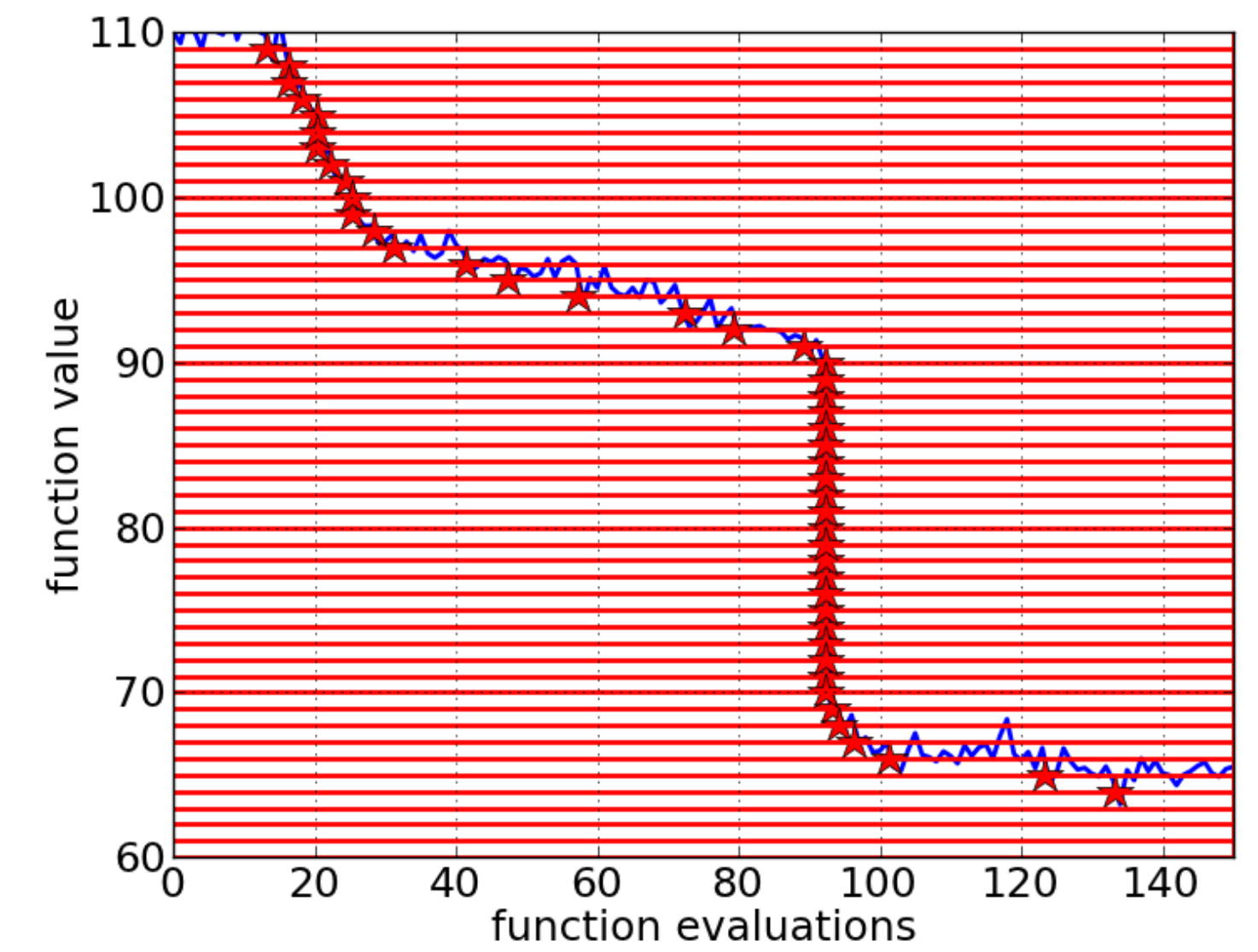
- function “target” values are the independent variable

time is the dependent variable, the measurement

- still recovers the original data

a time measurement for each discretization function value, these measurements can be plotted as ECDF

using the



horizontal discretization

is

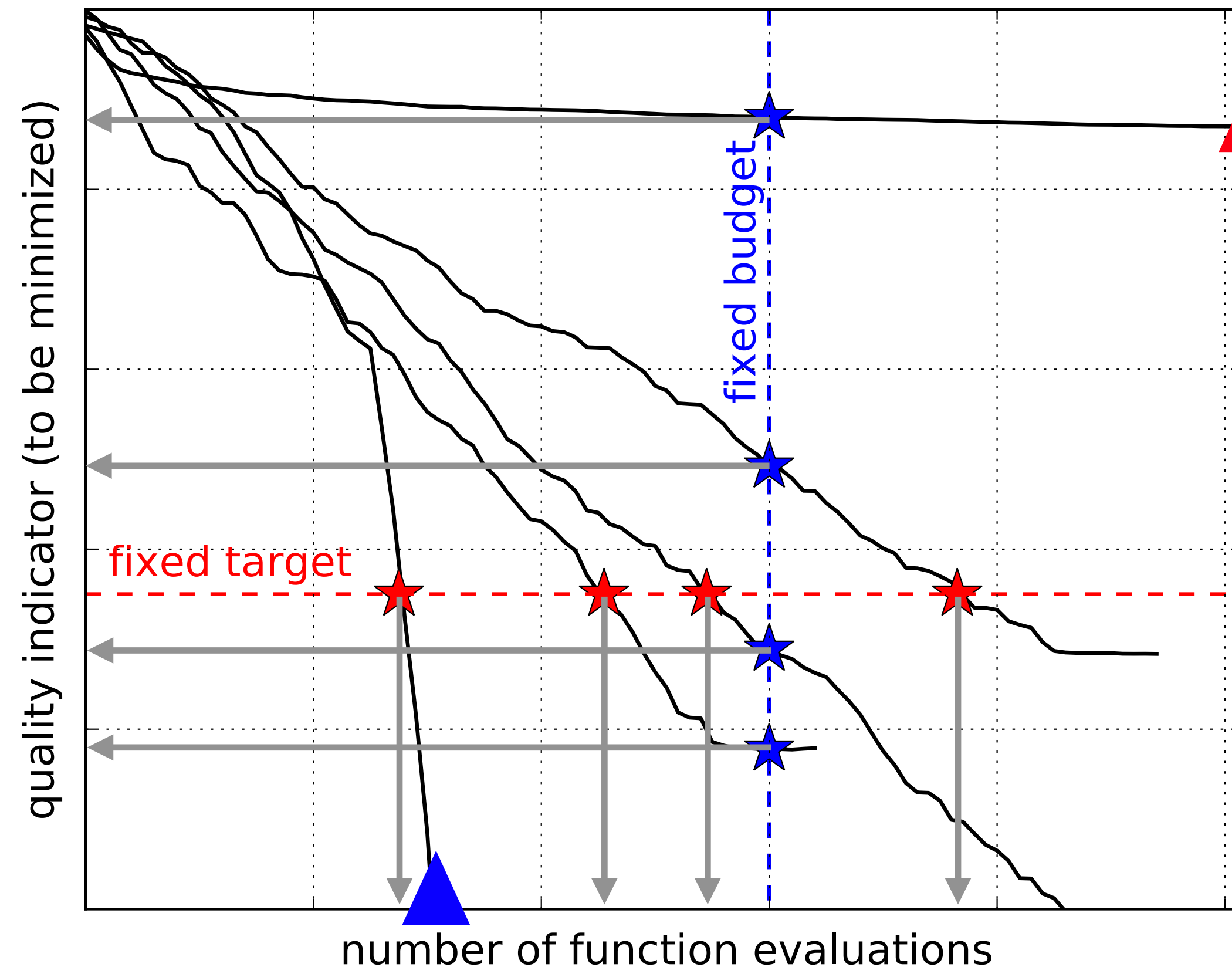
not

just

a technical subtlety

because it crucially determines the measurement we are looking at in the end

Fixed Target(s) versus Fixed Budget



- five convergence graphs
“quality indicator” versus “time”

- **Both** can lead to *imprecise data (a bound)* in some cases

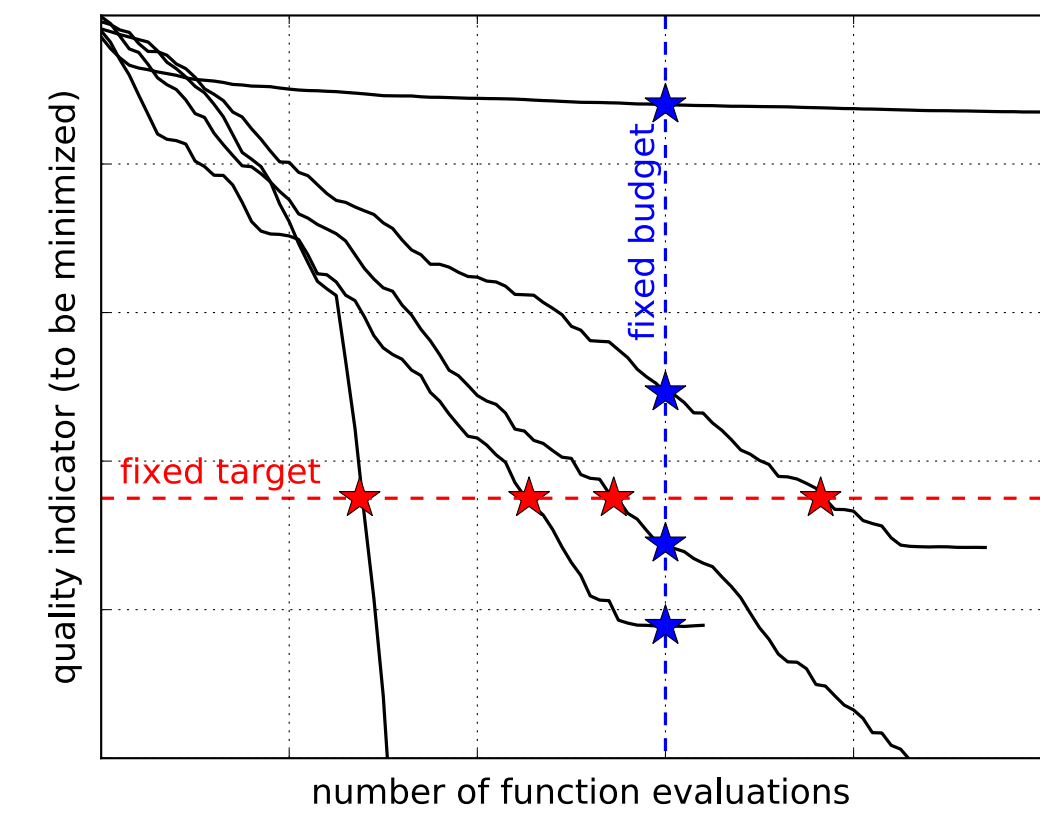
- “too good” performance (often overlooked)

(reached global optimum up to the relevant or numerical precision before the given budget)
quick and dirty fix: assign best possible (or measured) function value

- “too bad” performance

then the data only provide a lower bound estimate for the runtime (and a fixed budget measure at the maximum budget)
quick and dirty fix: assign 10 x time_out_budget

Fixed Target(s) versus Fixed Budget



The resulting measurement

- Fixed budget (vertical, target-free) design: **function values (quality)**
- Fixed target design (budget-free) design: **evaluations (runtime)**

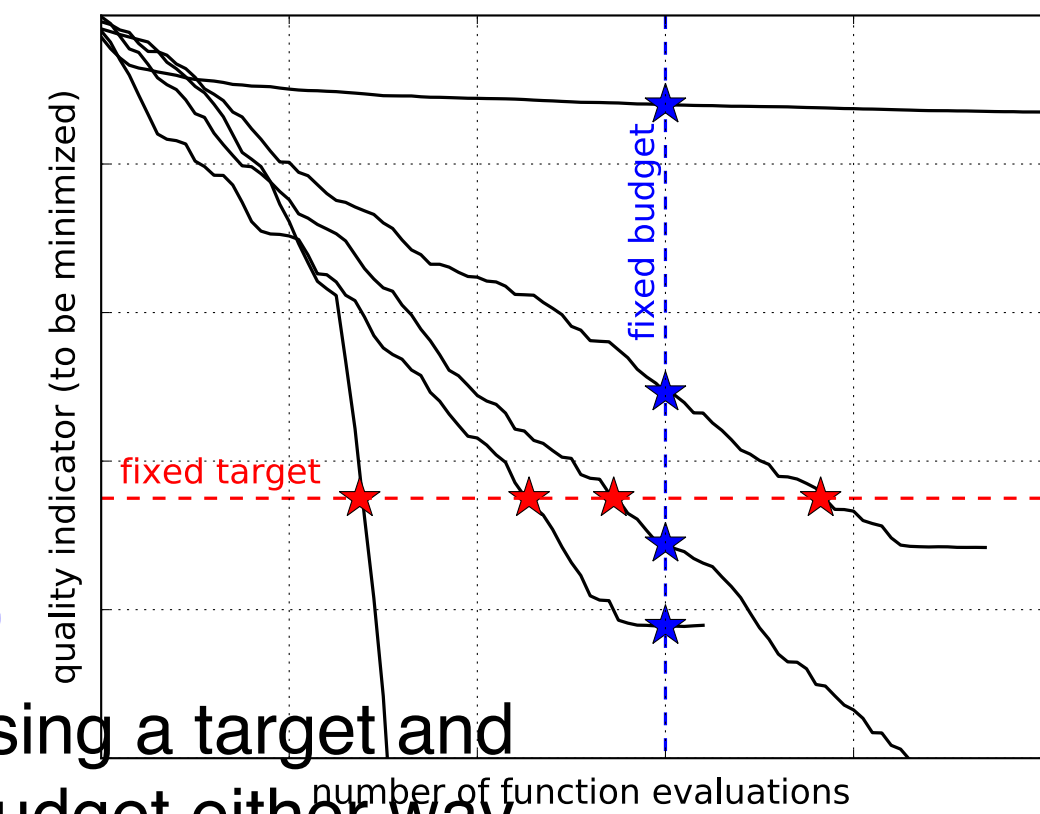
Does this make a difference?

Scales of Measurement (“Level” of Data)

- Nominal - categorical, define a classification
- Ordinal - define an order, ranks, function *values* (fixed budget)
- Interval - differences are meaningful
- Rational - ratios are meaningful, we usually can take the logarithm, function *evaluations* (fixed target)

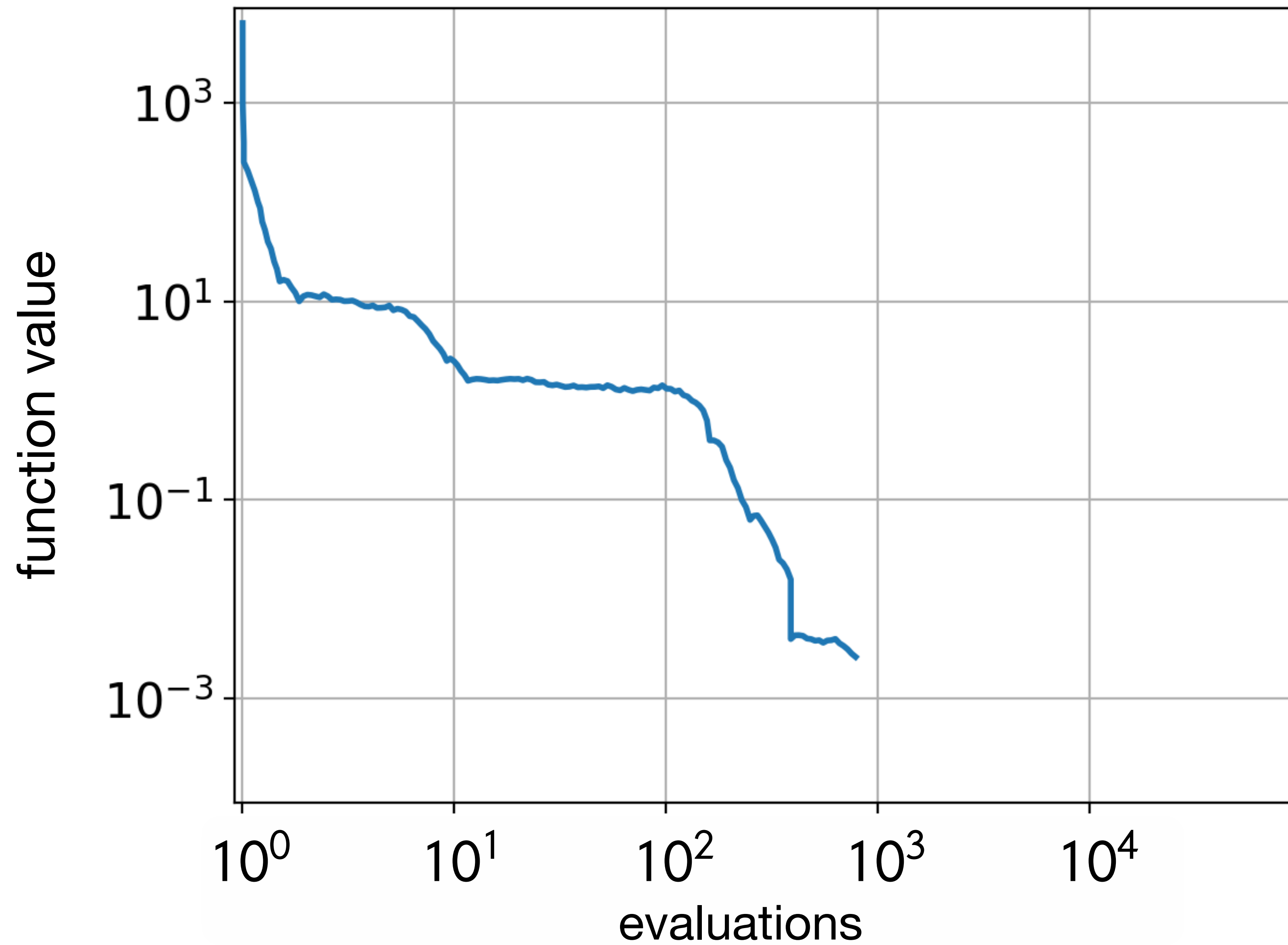
CAVEAT: mathematical and semantic treatment of data is not the same. From a classification with values $\{1, 2\}$ we can *mathematically* take differences and ratios of the values, but they have no meaningful *semantic interpretation*. Fahrenheit or Celsius versus Kelvin describe temperatures, however only Kelvin is on a rational scale of measurement.

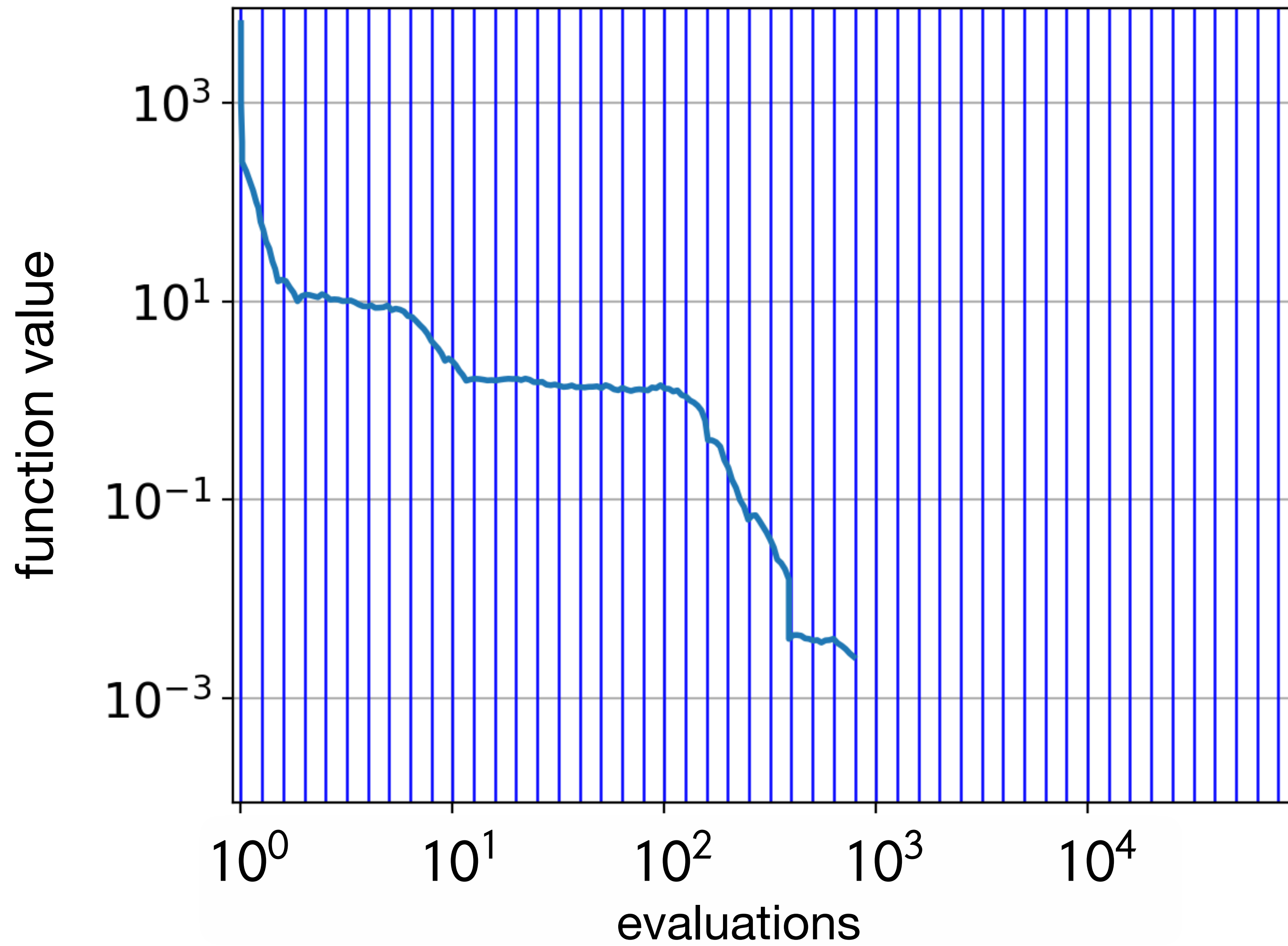
Summarizing Fixed Target(s) versus Fixed Budget

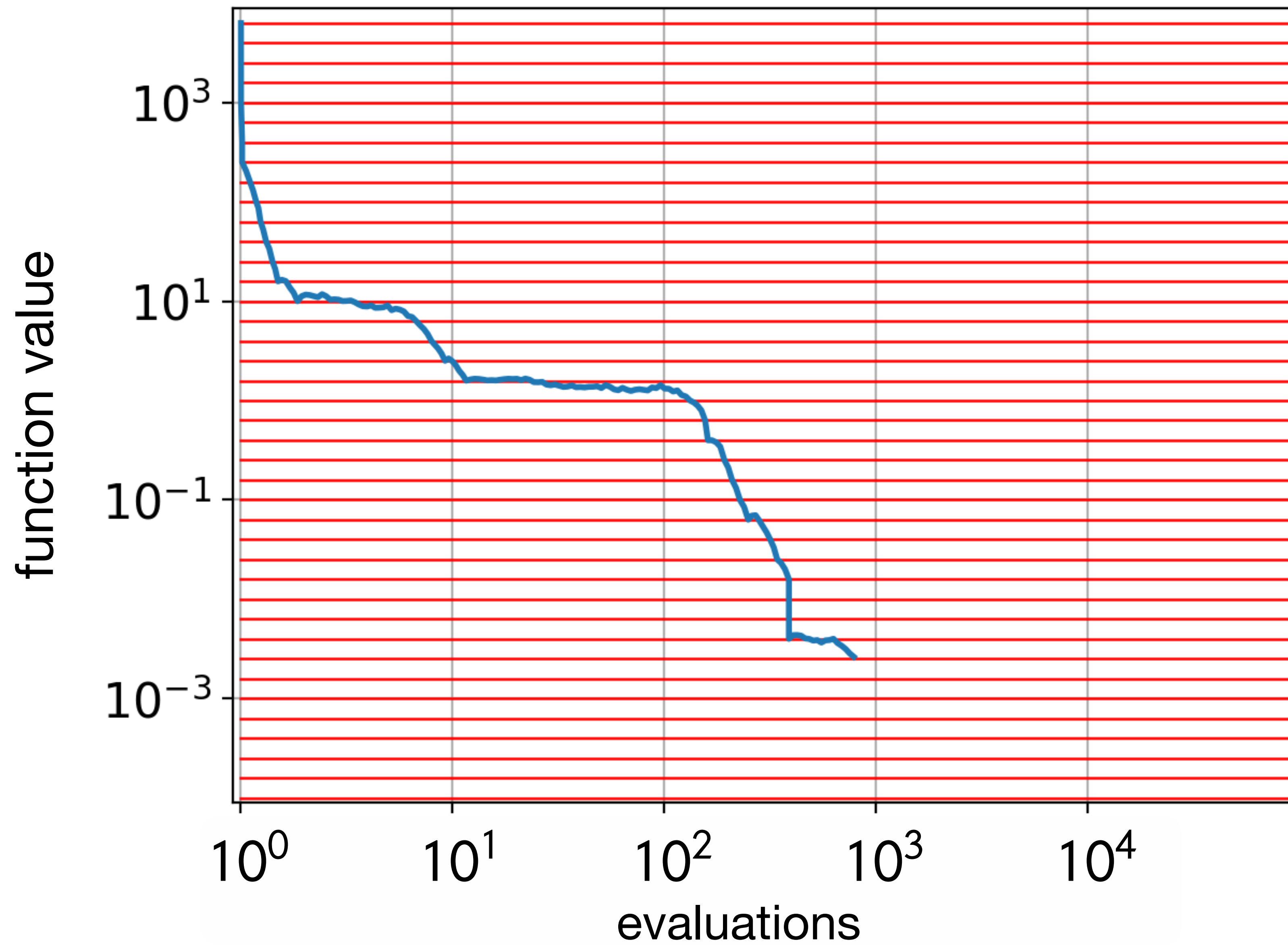


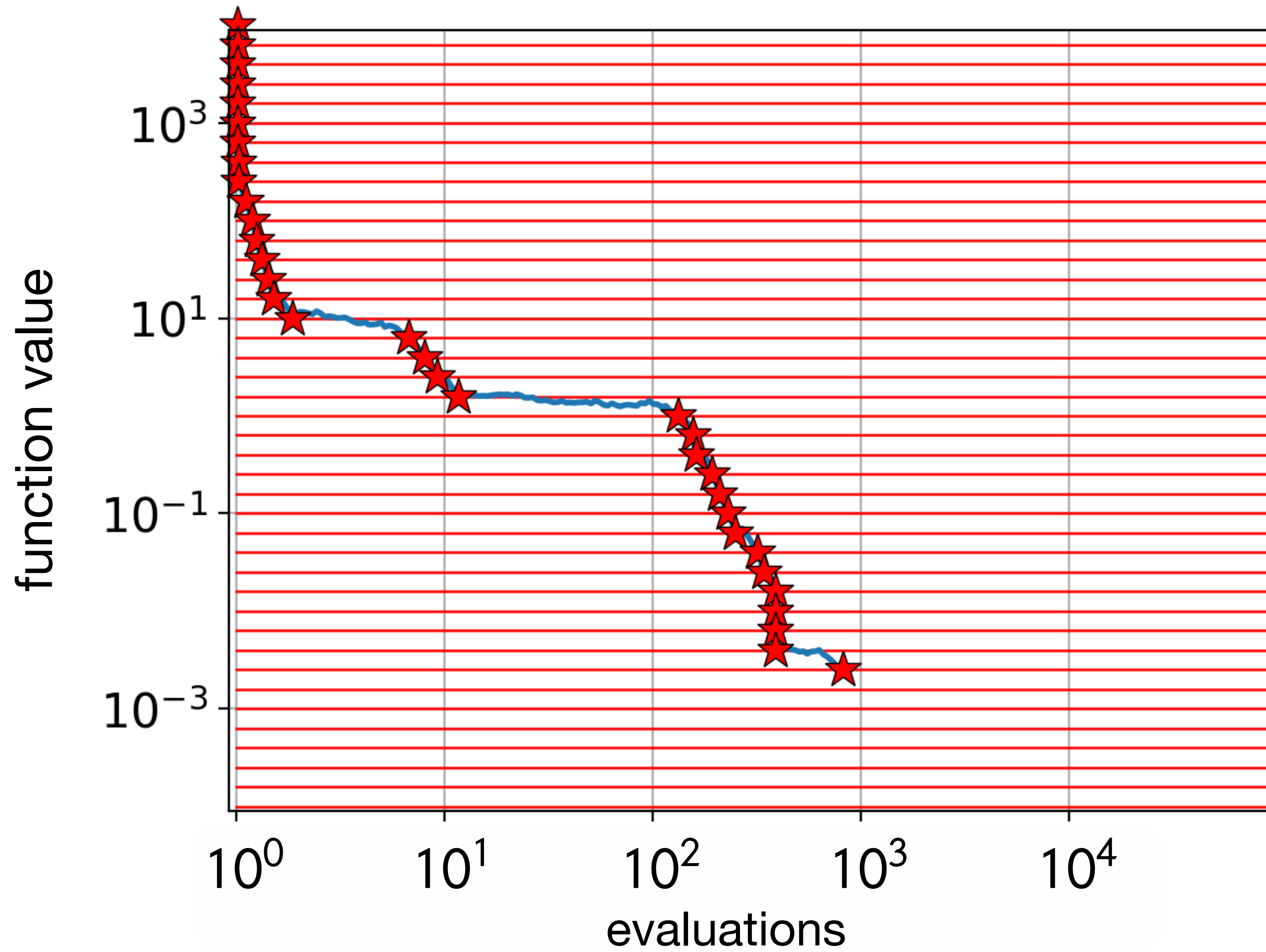
- The fixed budget (vertical) design is (much) **easier to set up**
target-free: choosing a budget is simpler than choosing a target and we need to choose a maximal “timeout” budget either way
- For the (very) same reason, results from the fixed target (horizontal) design are (much) **simpler to interpret and usually more conclusive**
without specific knowledge/insight, a function value is impossible to interpret beyond ordering
- Runtimes have a **quantitative interpretation**
“Algorithm A is 100 times faster than Algorithm B”
- Fixed target results can be **meaningfully aggregated** in ECDFs and geometric averages
whereas function values from different functions are in general not commensurable
- Fixed target results are **“budget-free”**
we can compare results with different maximal “timeout” budgets

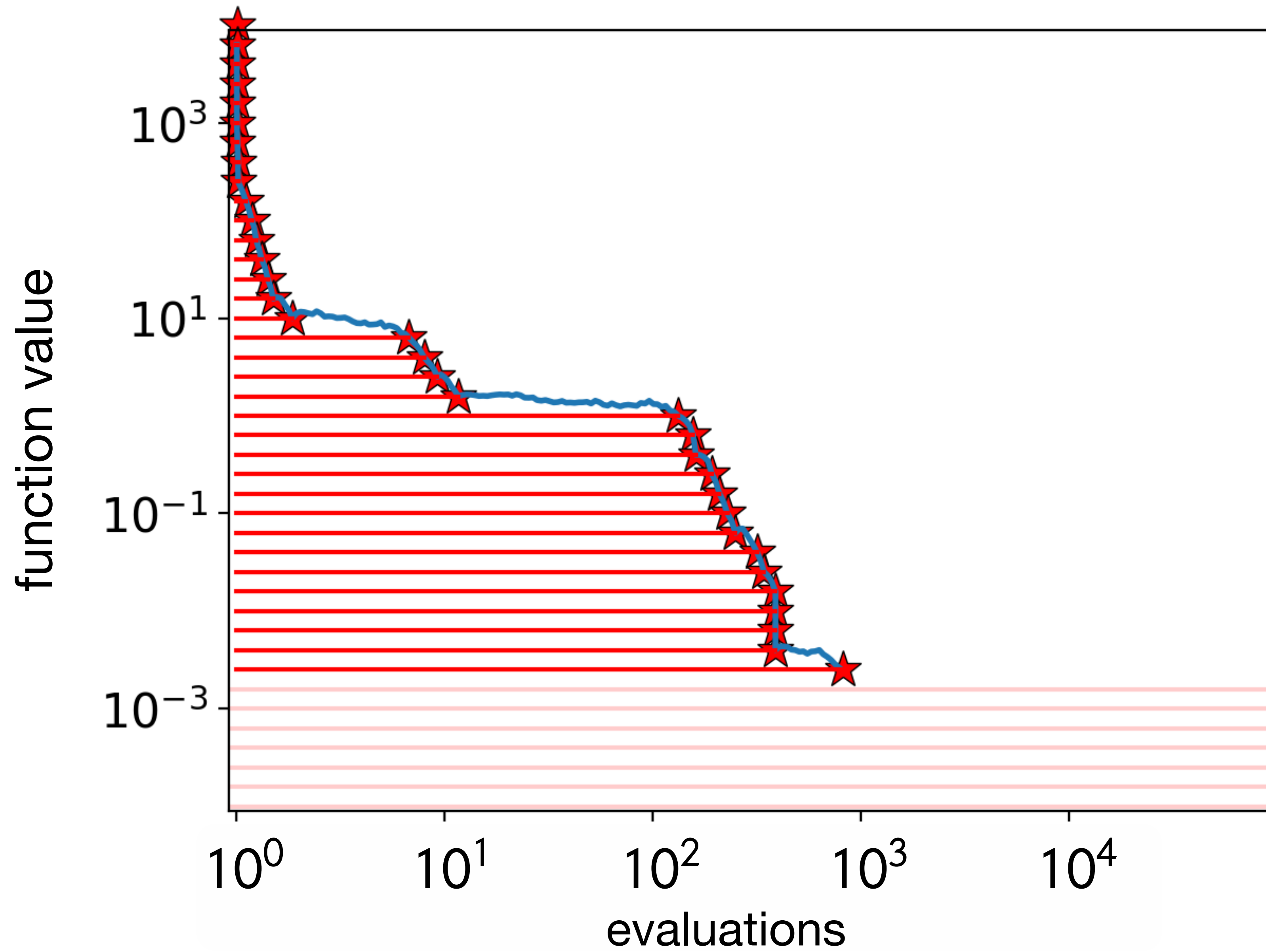
From a Convergence Graph to the Empirical Runtime Distribution

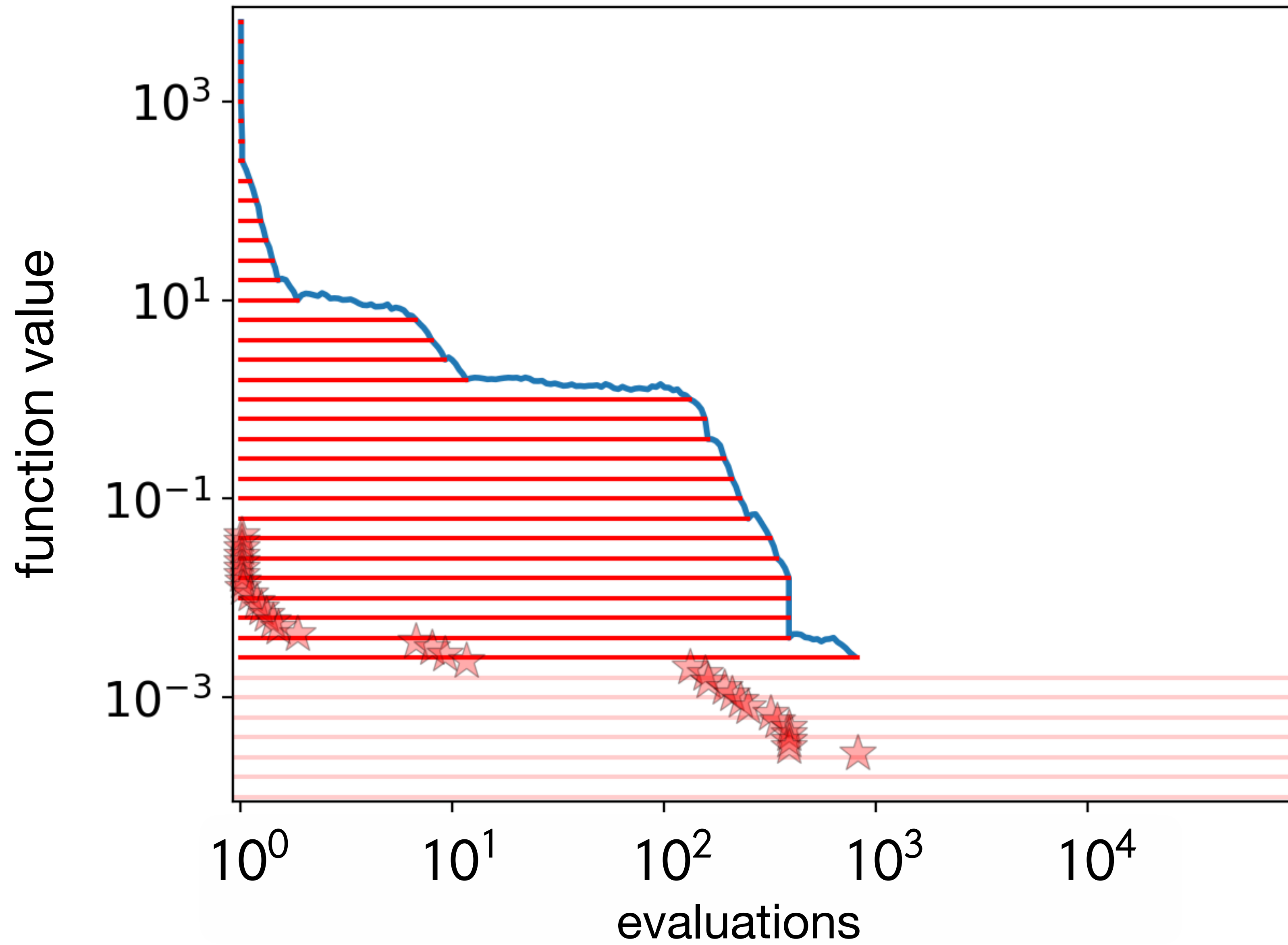


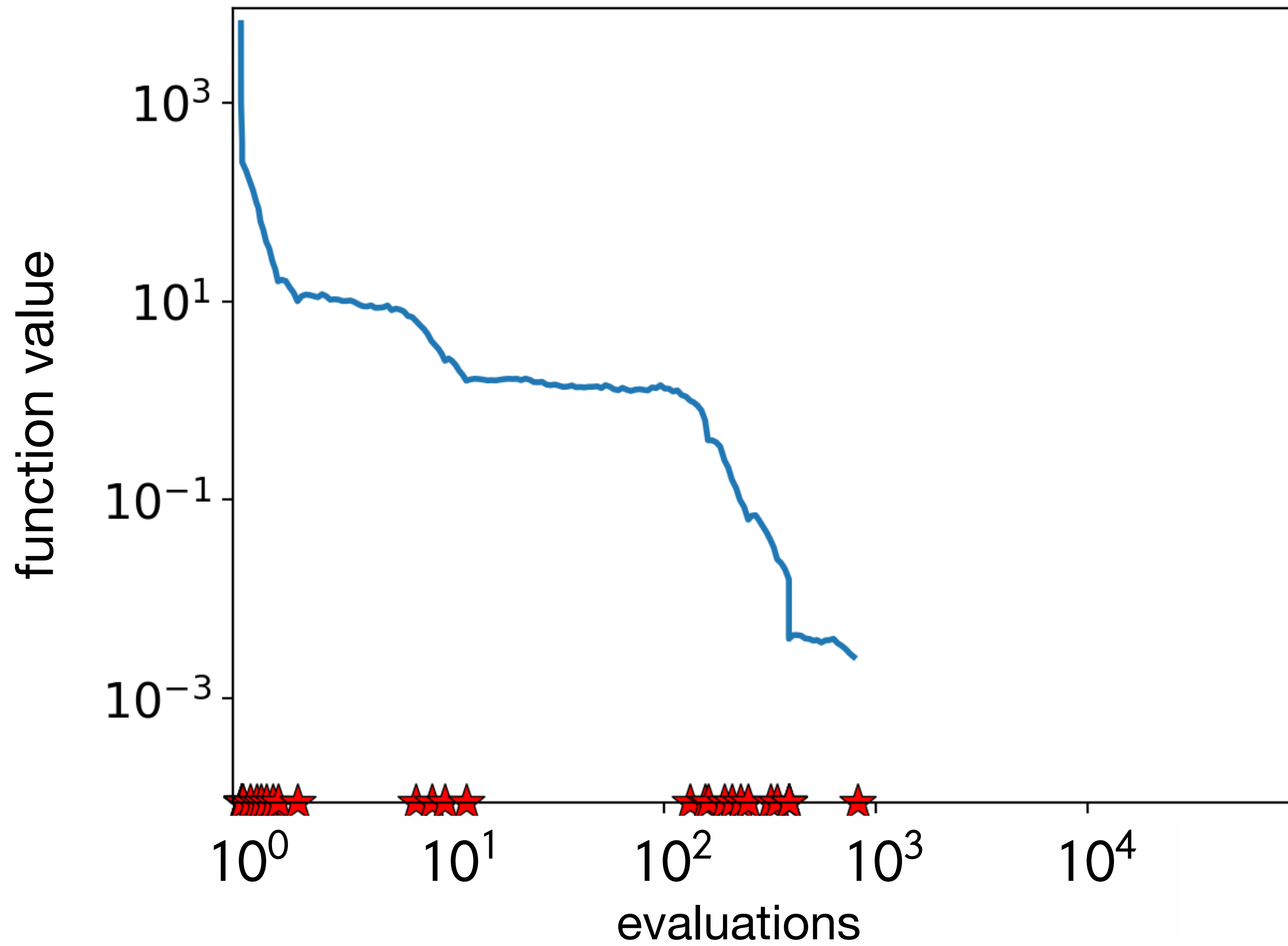


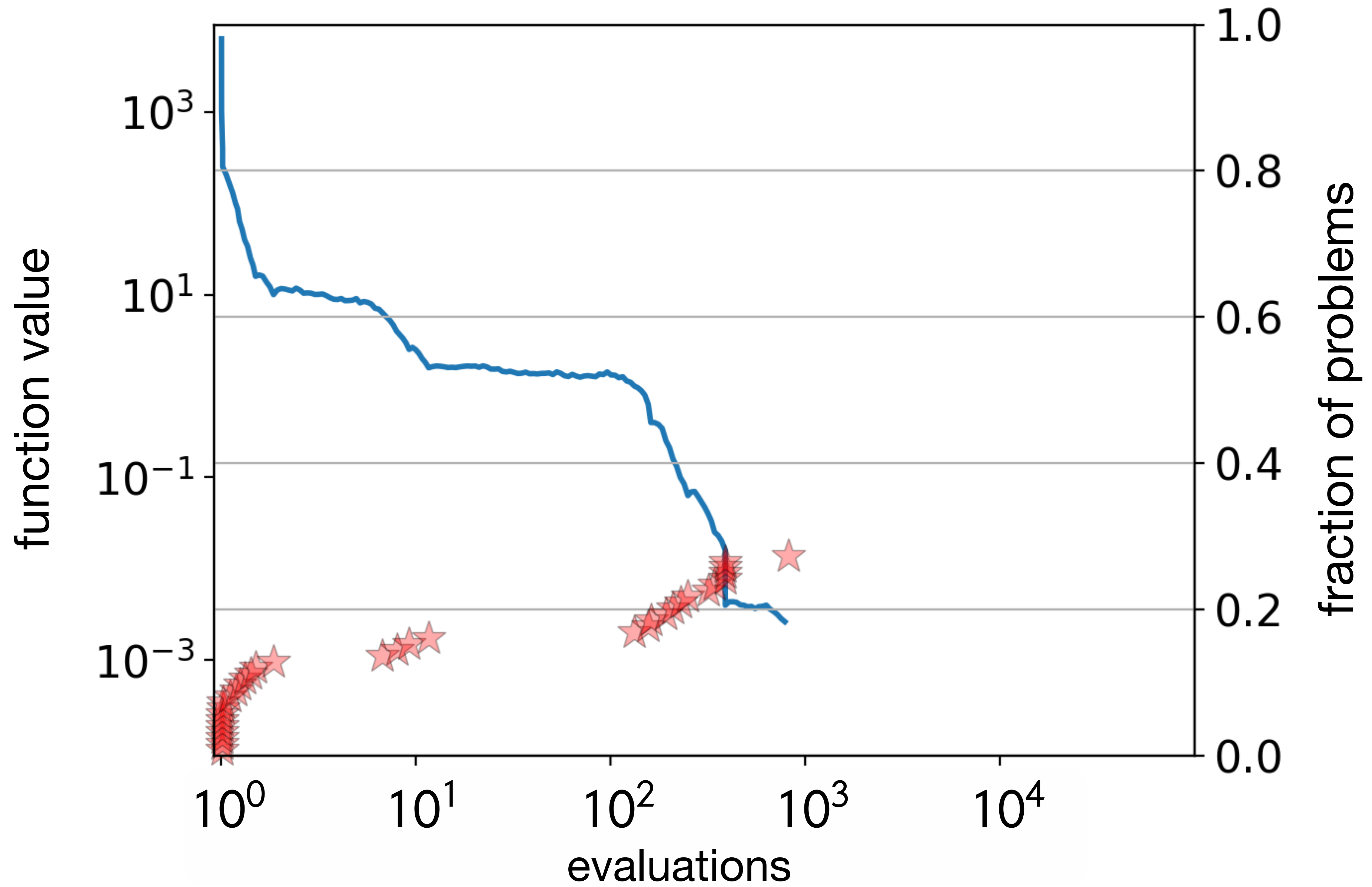


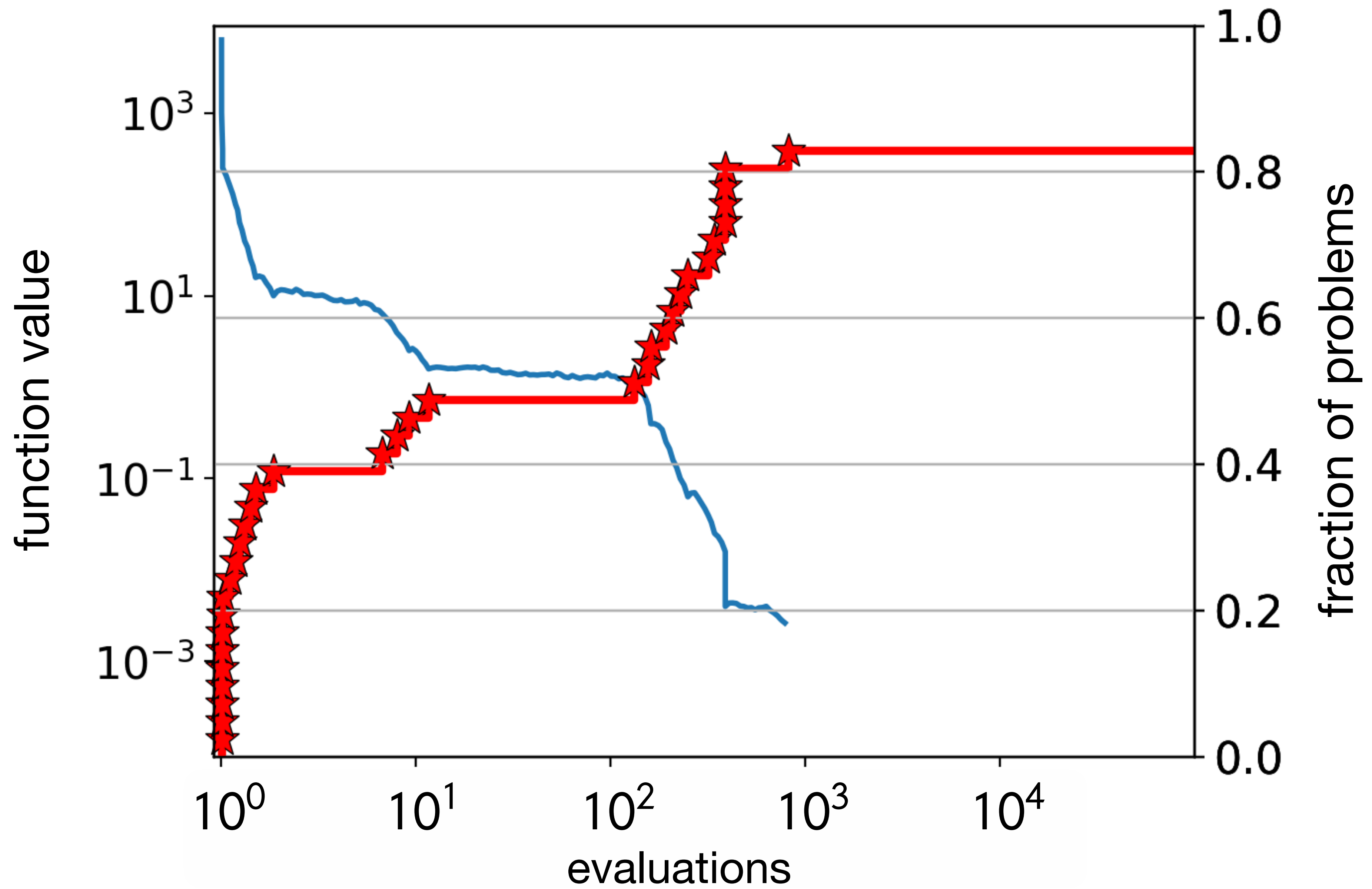


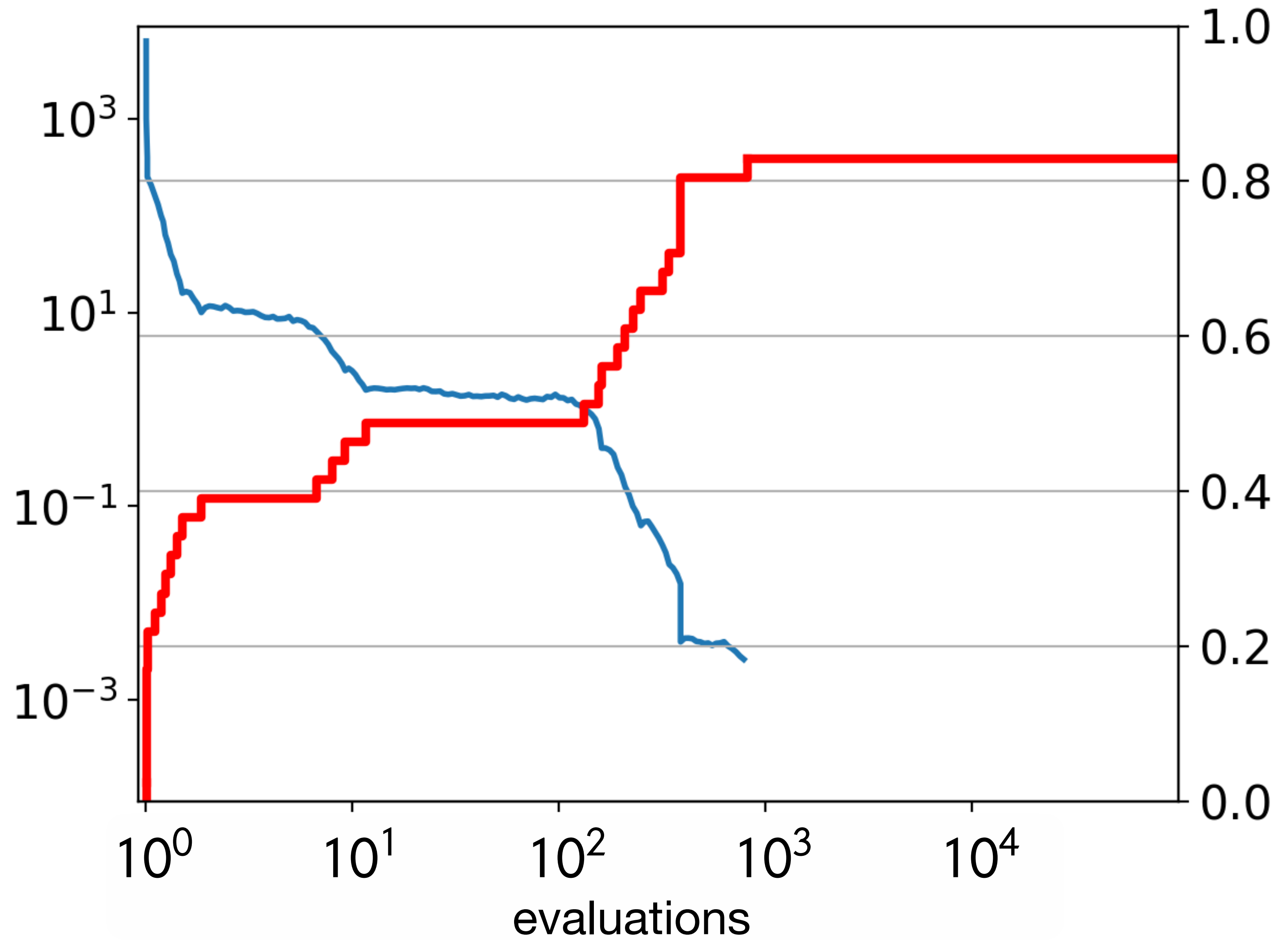


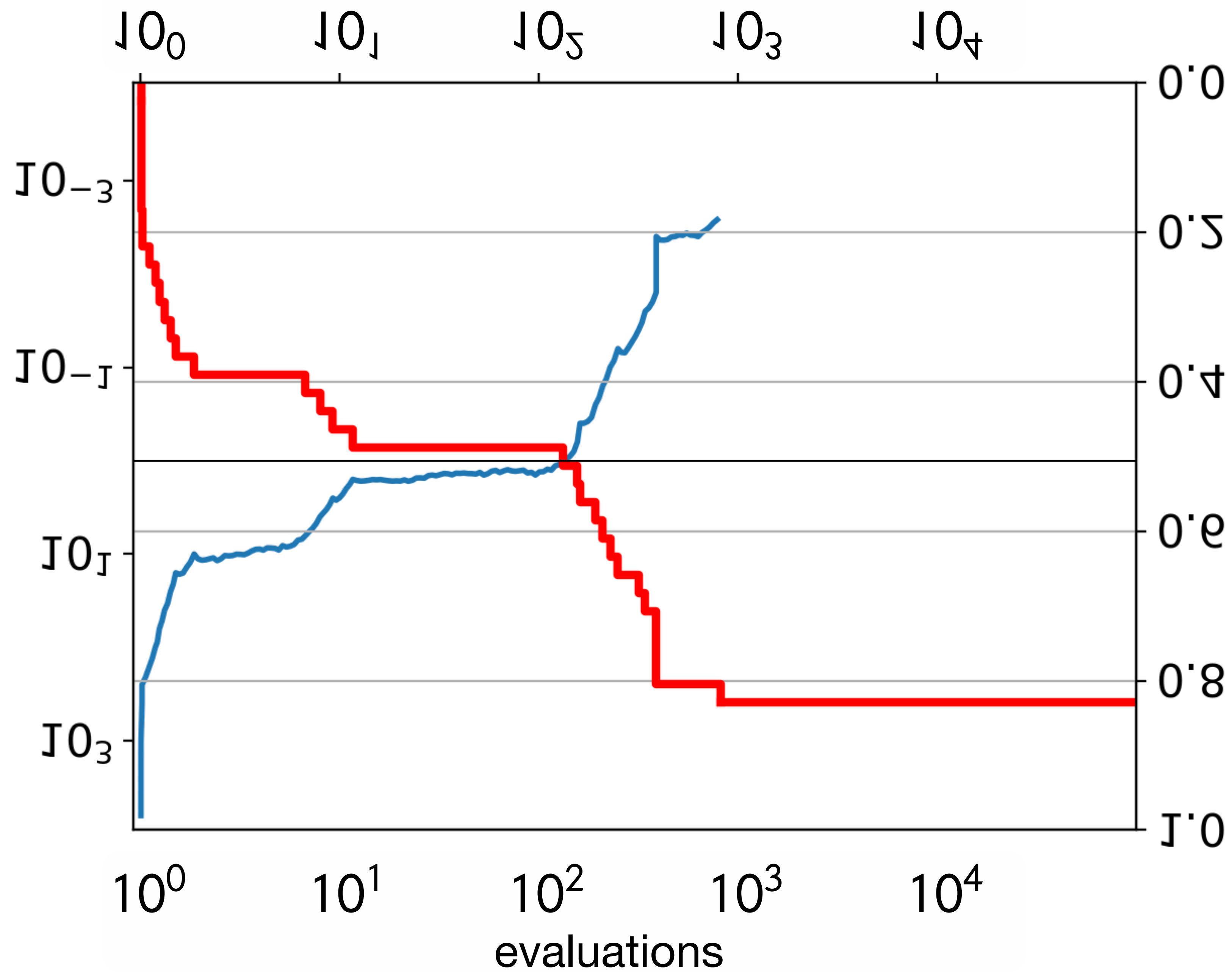


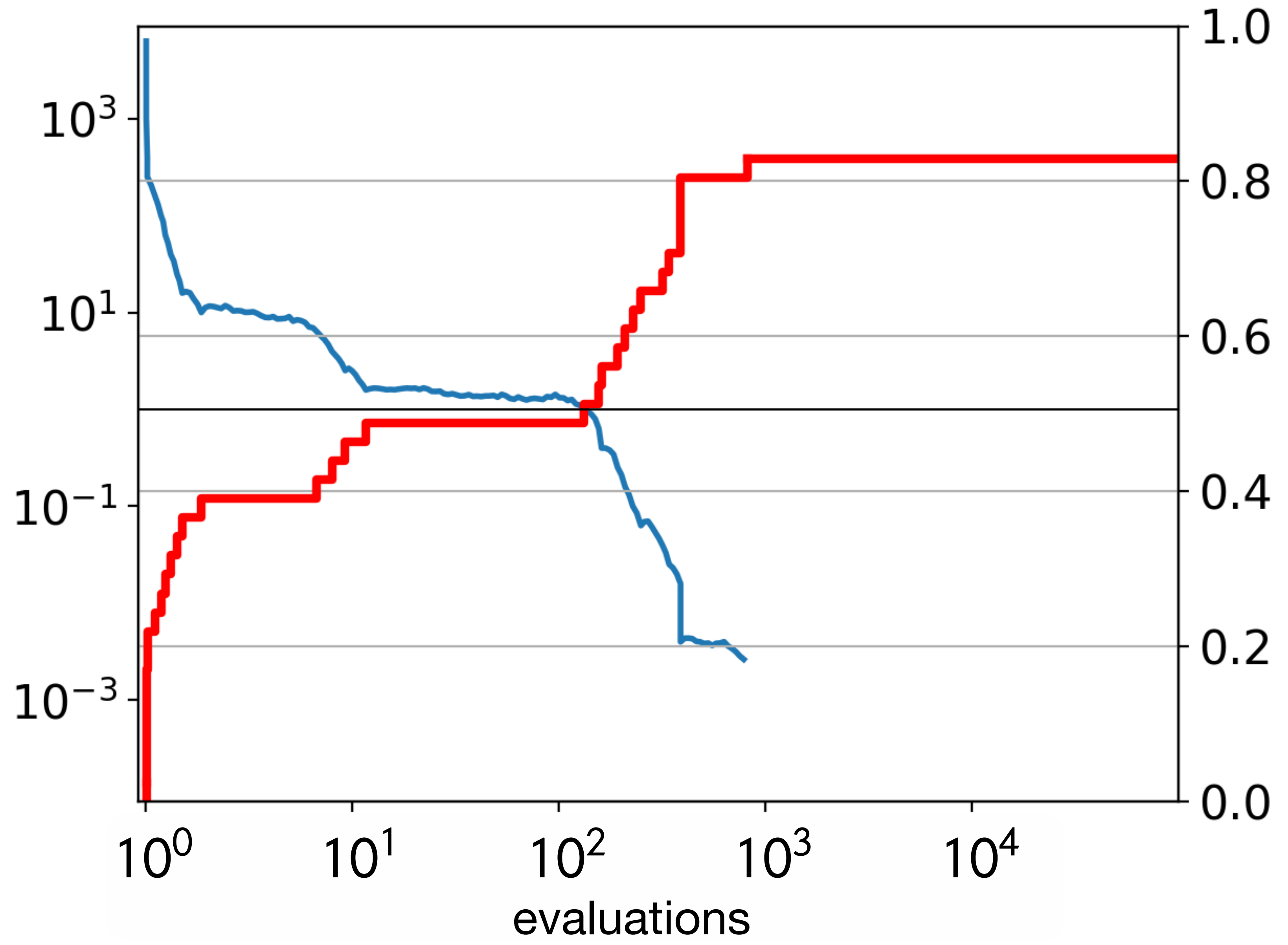


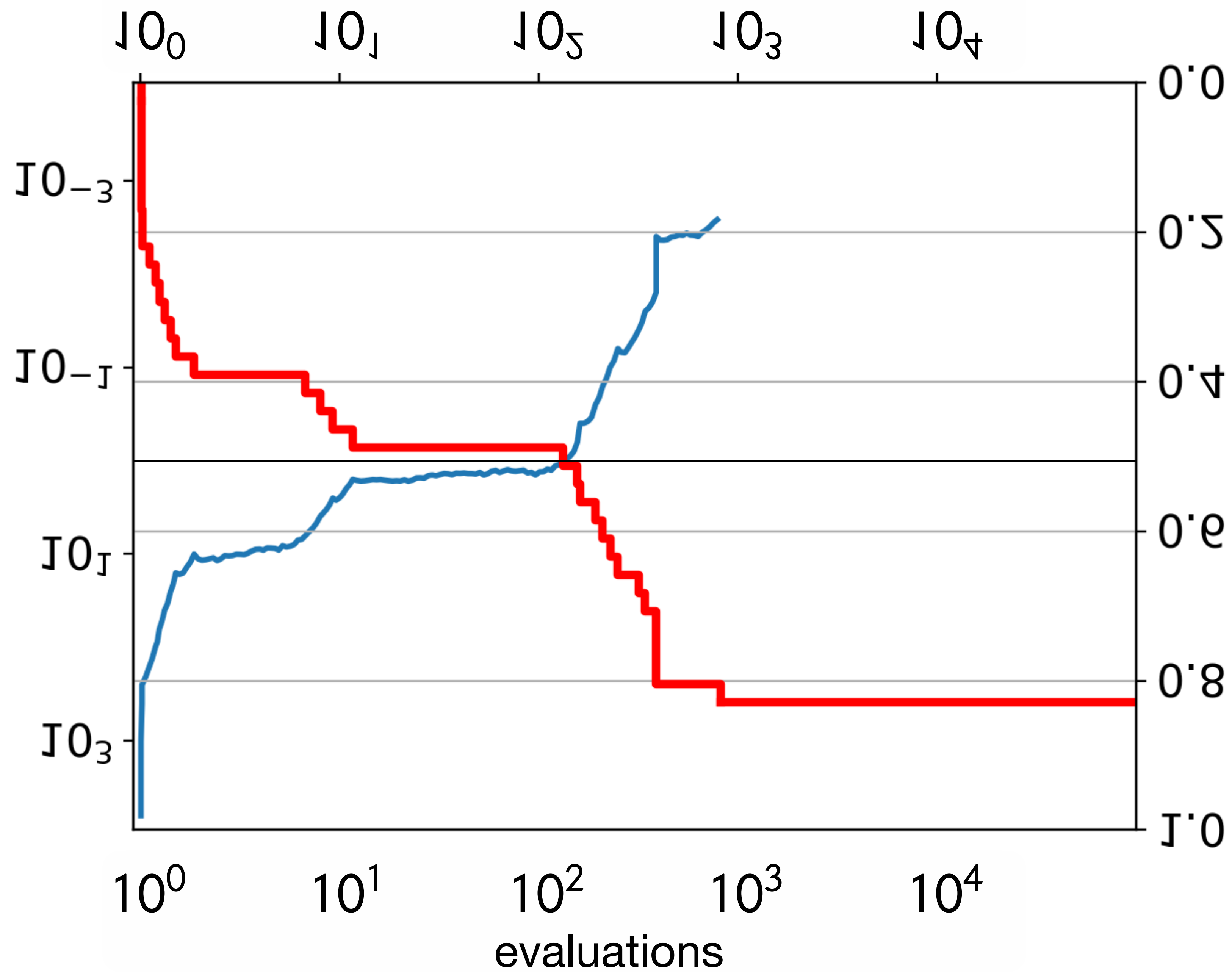




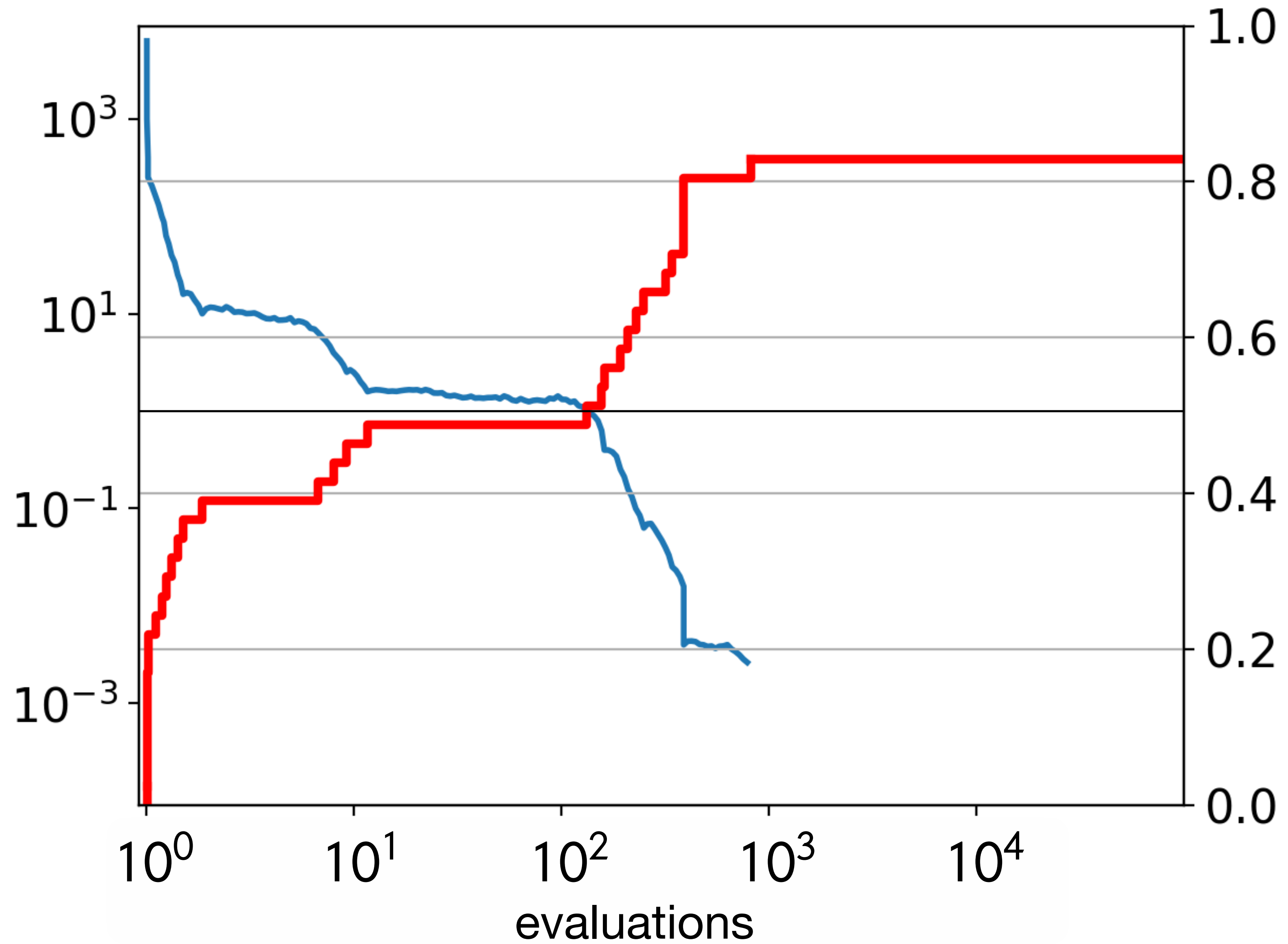


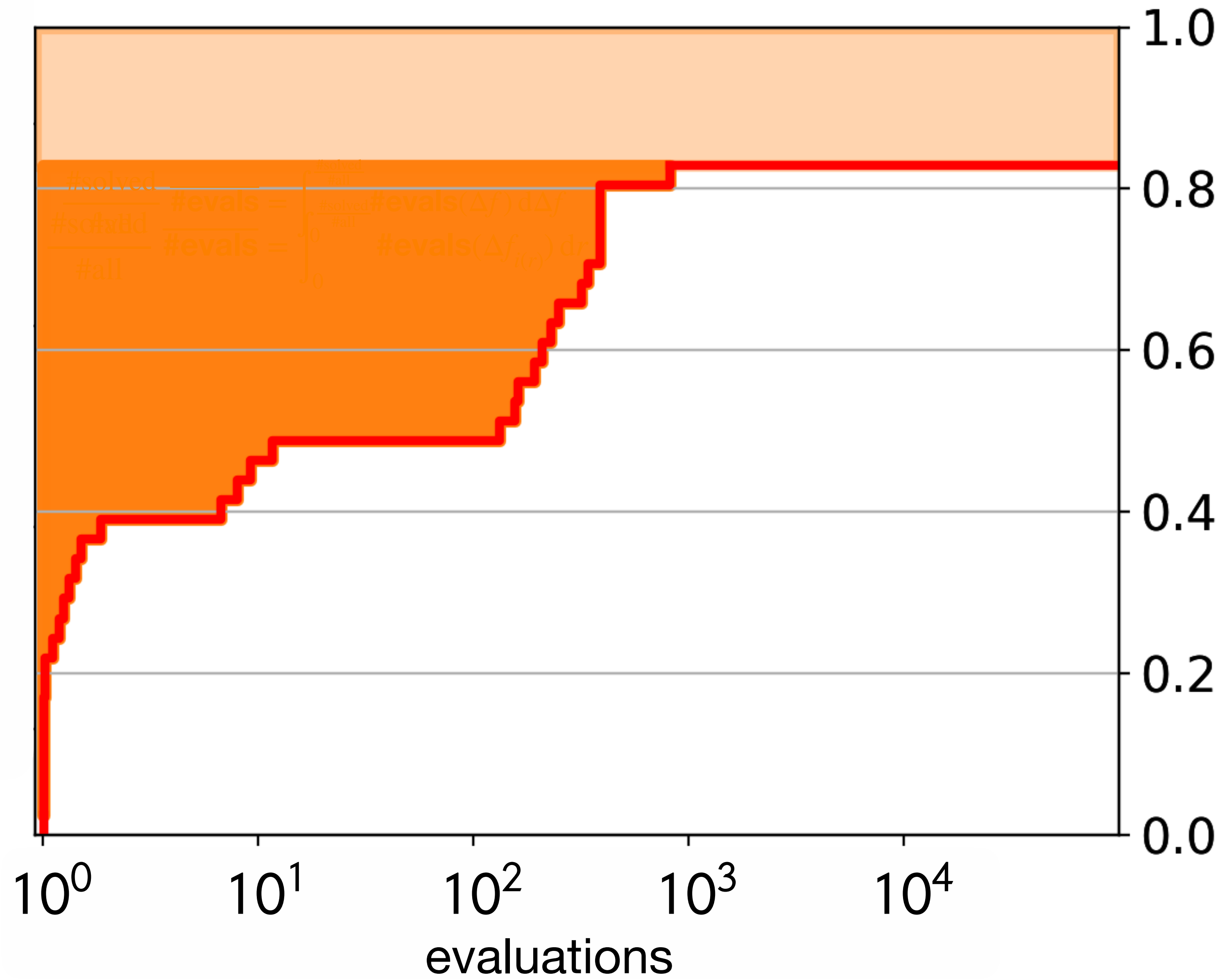






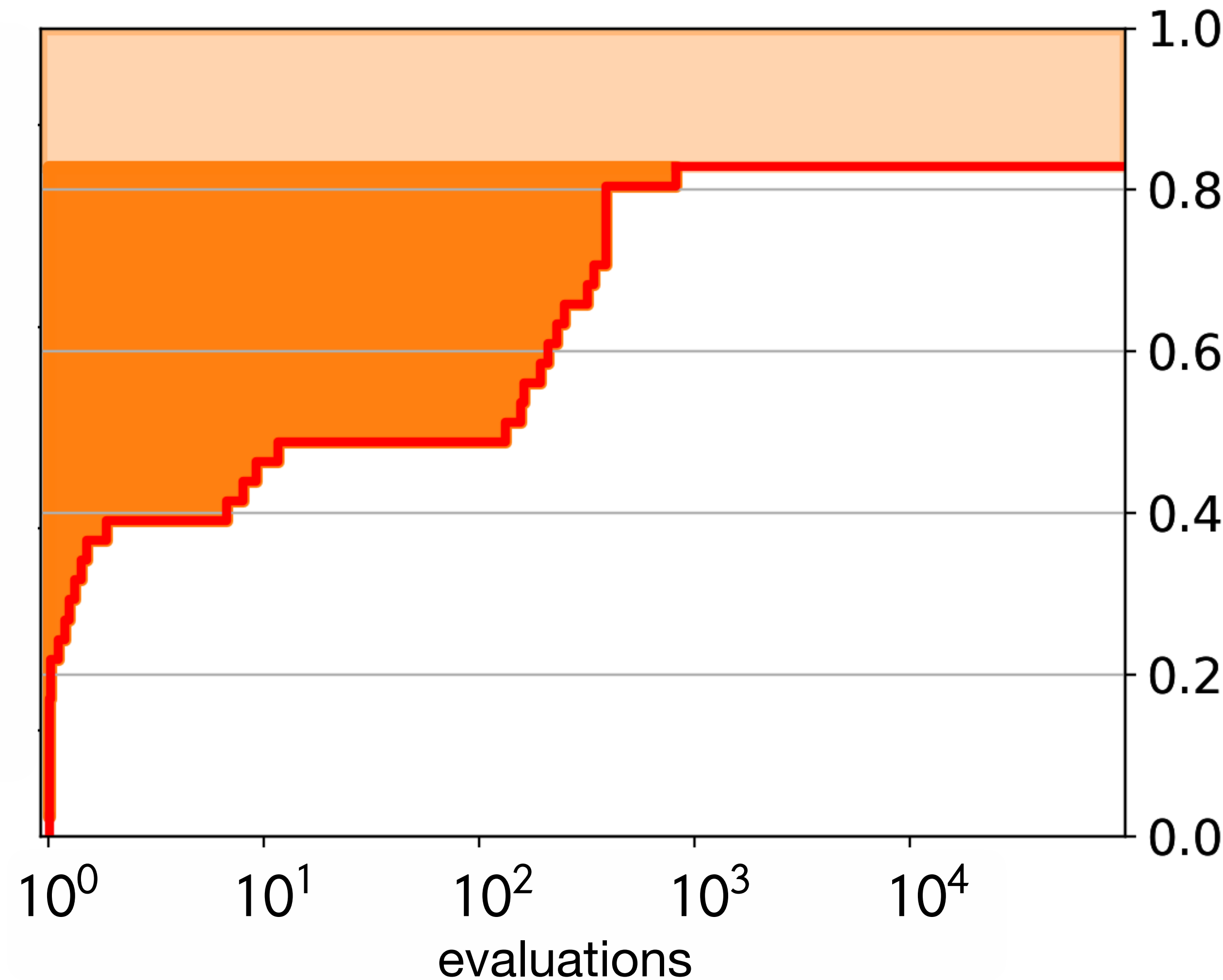
**when we maximize
(instead of minimize),
the graph can be considered as an
empirical runtime
distribution as is**

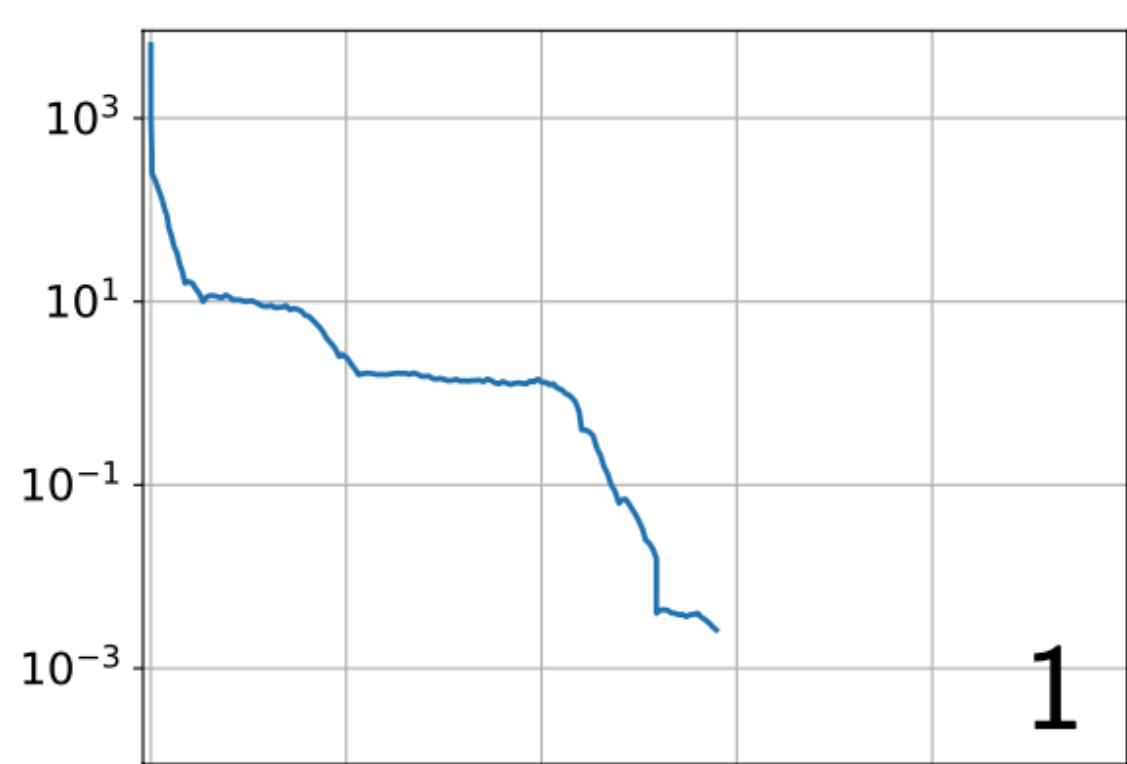




$$\overline{\#evals} = \frac{\#all}{\#solved} \int_0^{\frac{\#solved}{\#all}} \#evals(\Delta f_{i(r)}) dr$$

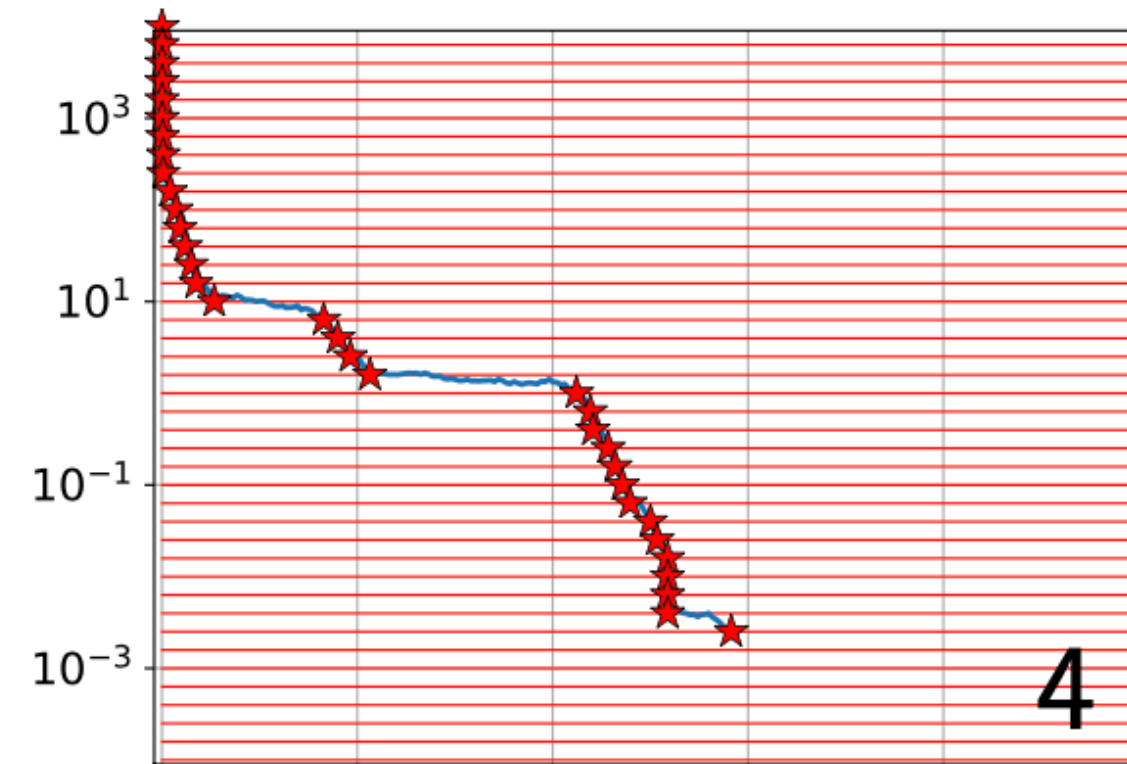
When the x-axis is in log-scale, it is the geometric average





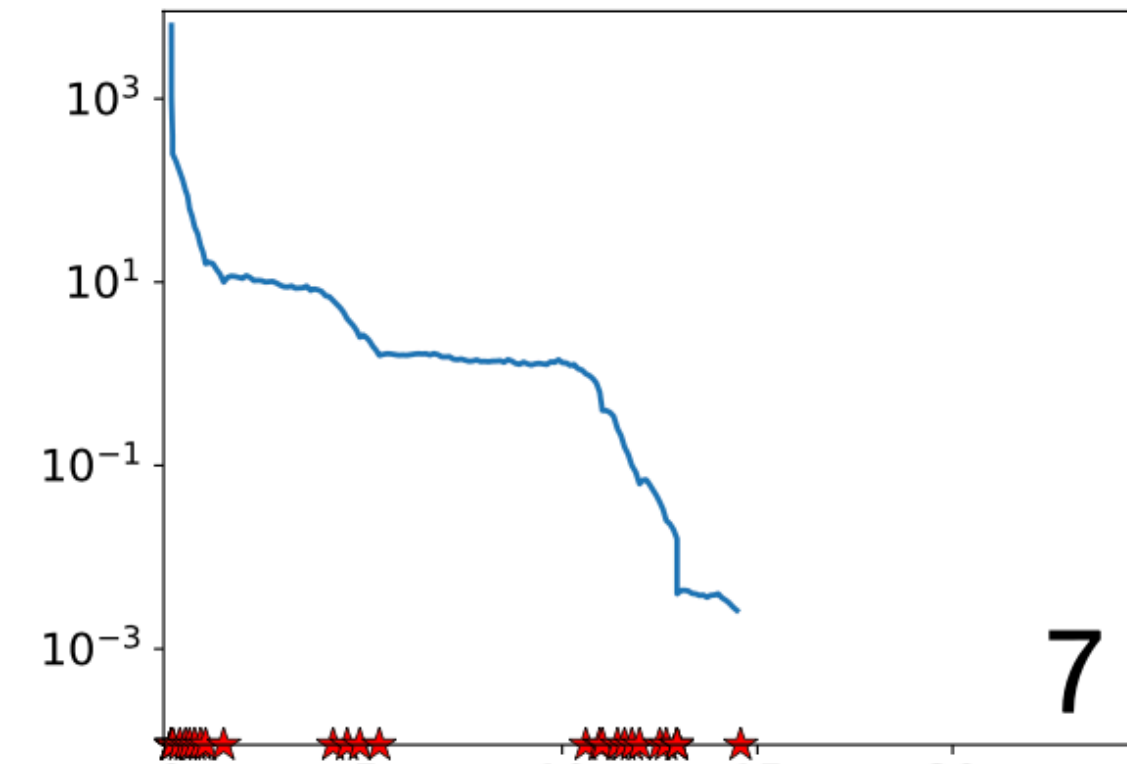
evaluations

1



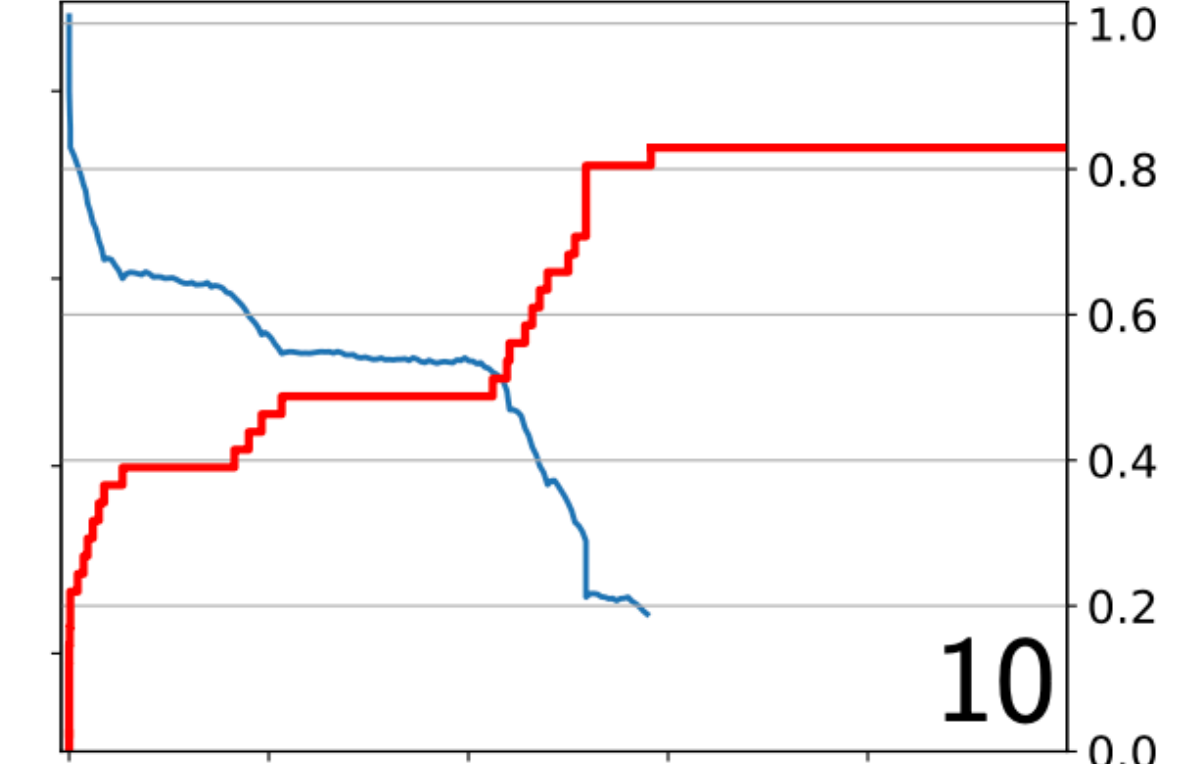
evaluations

4



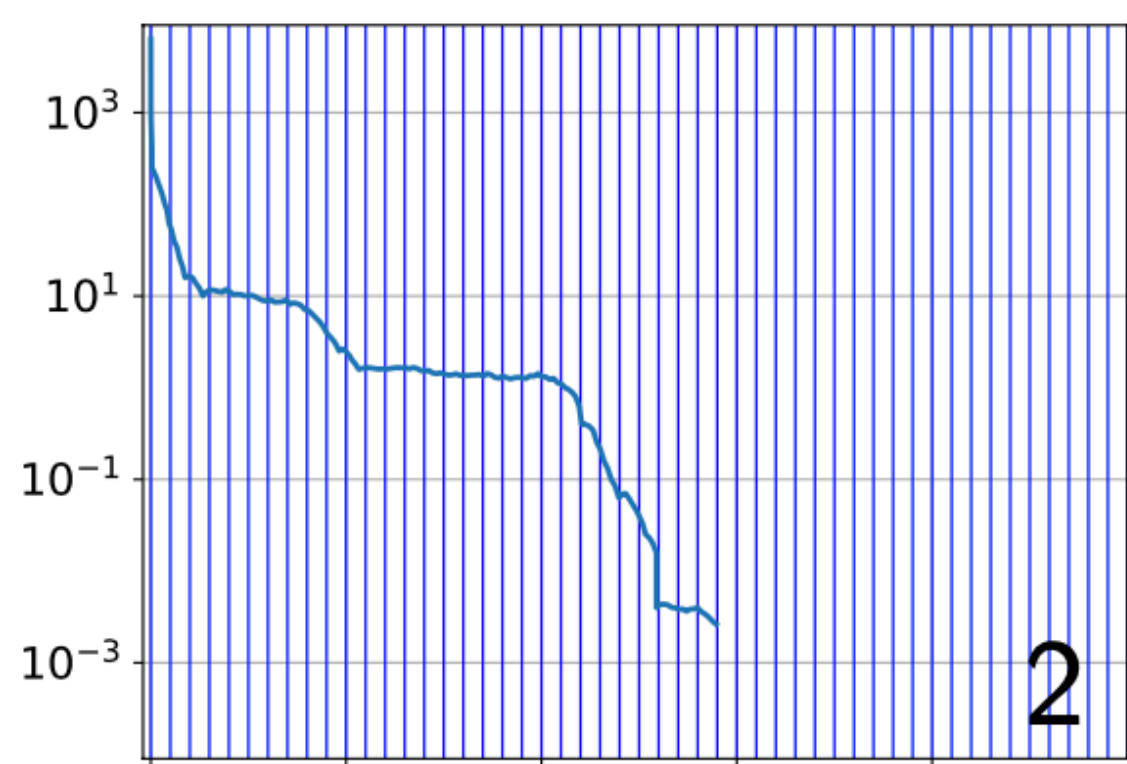
evaluations

7



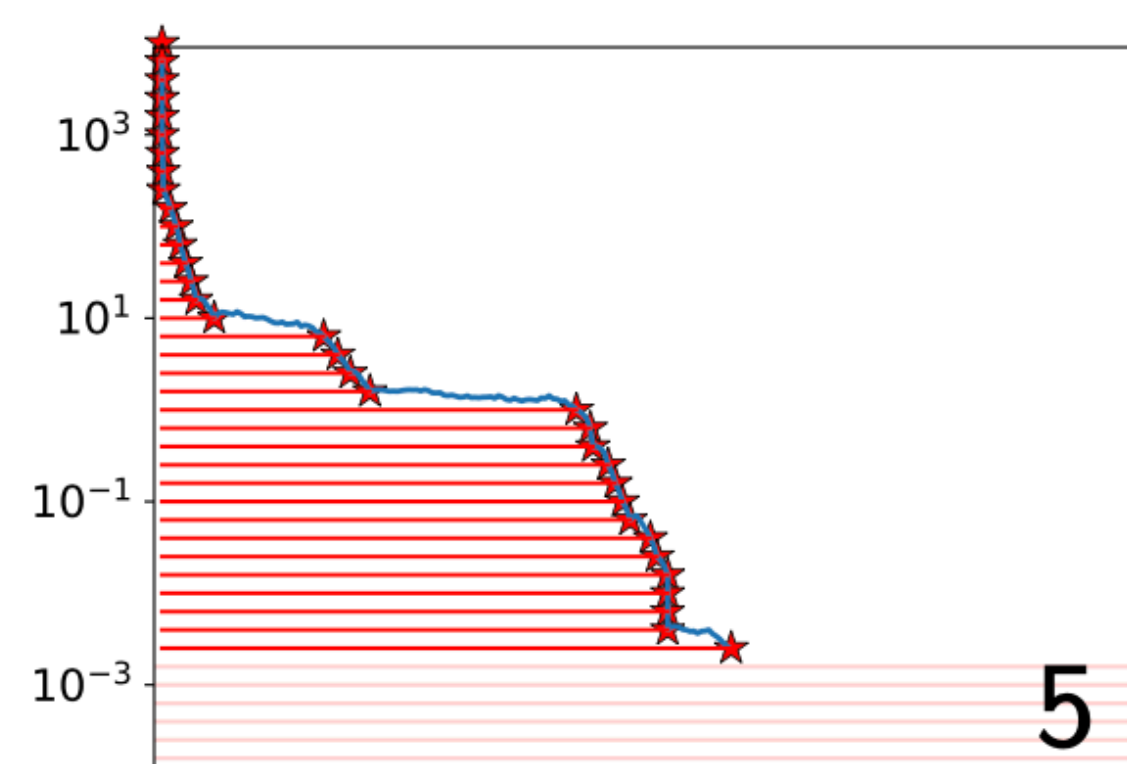
evaluations

10



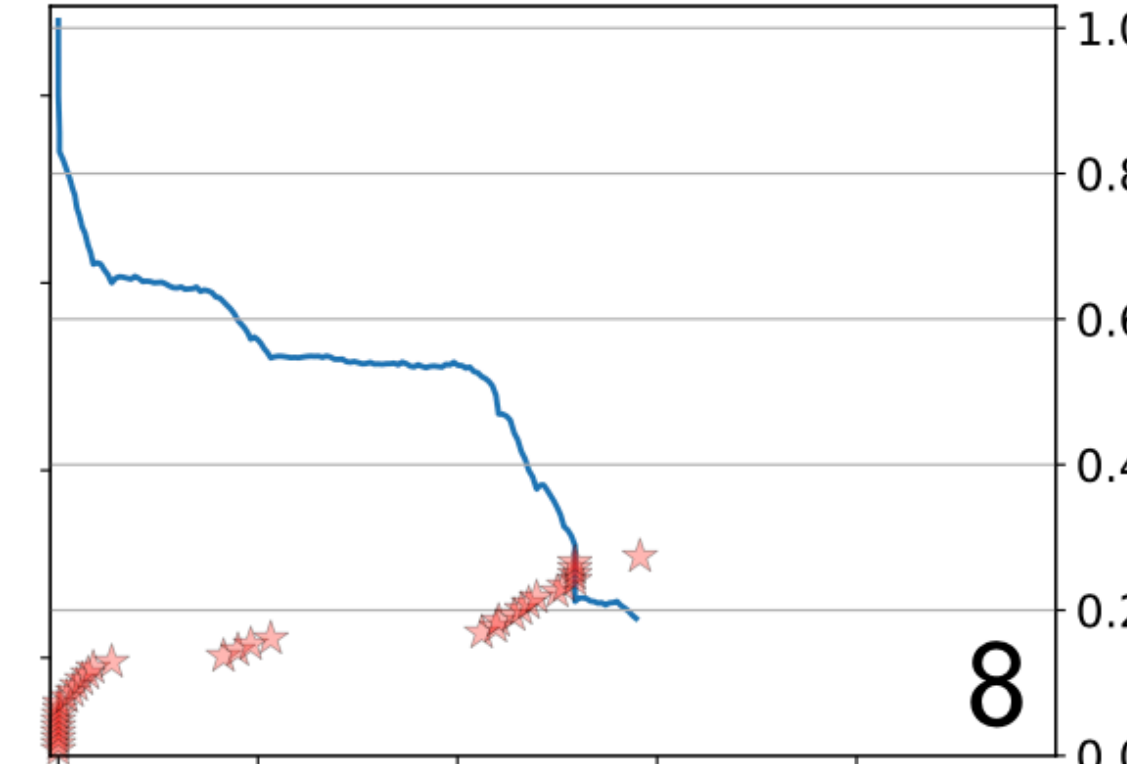
evaluations

2



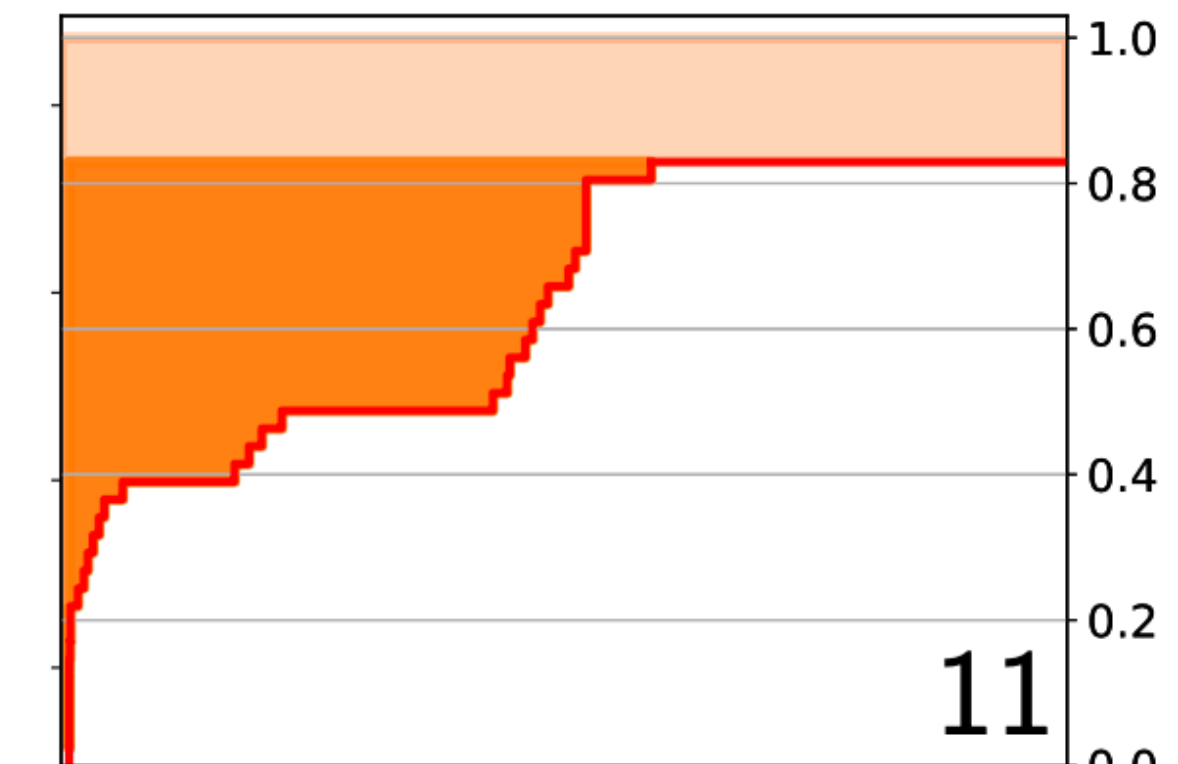
evaluations

5



evaluations

8



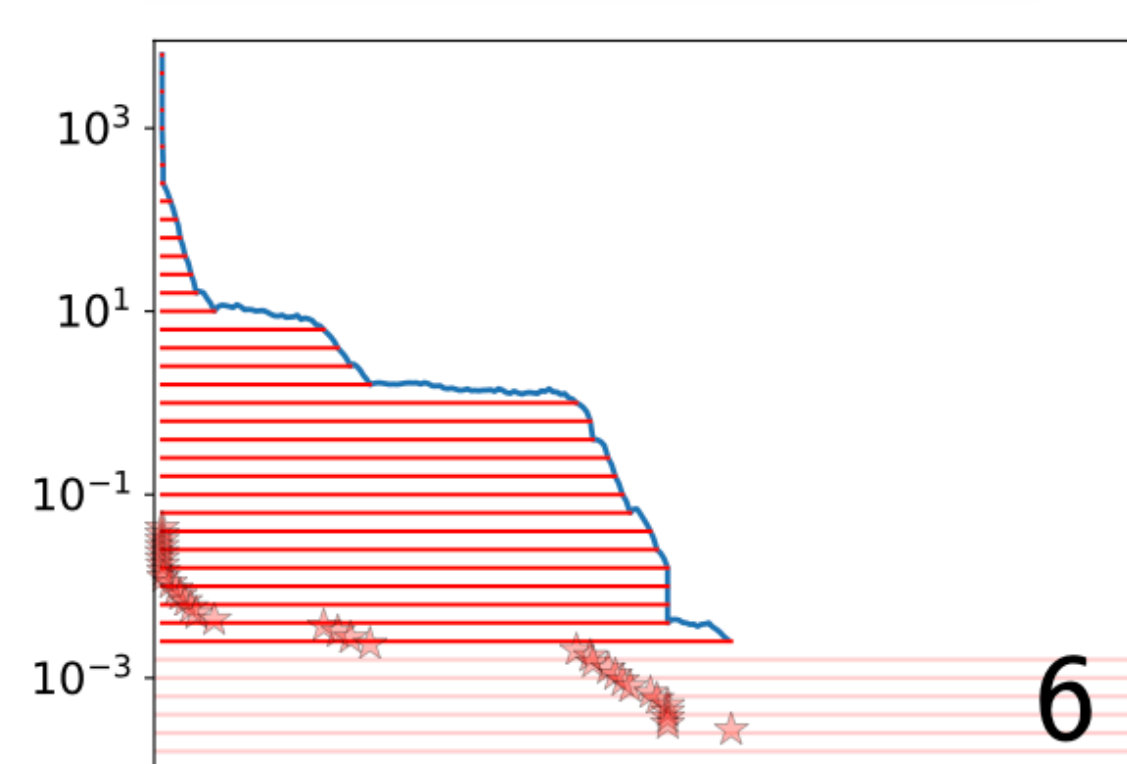
evaluations

11



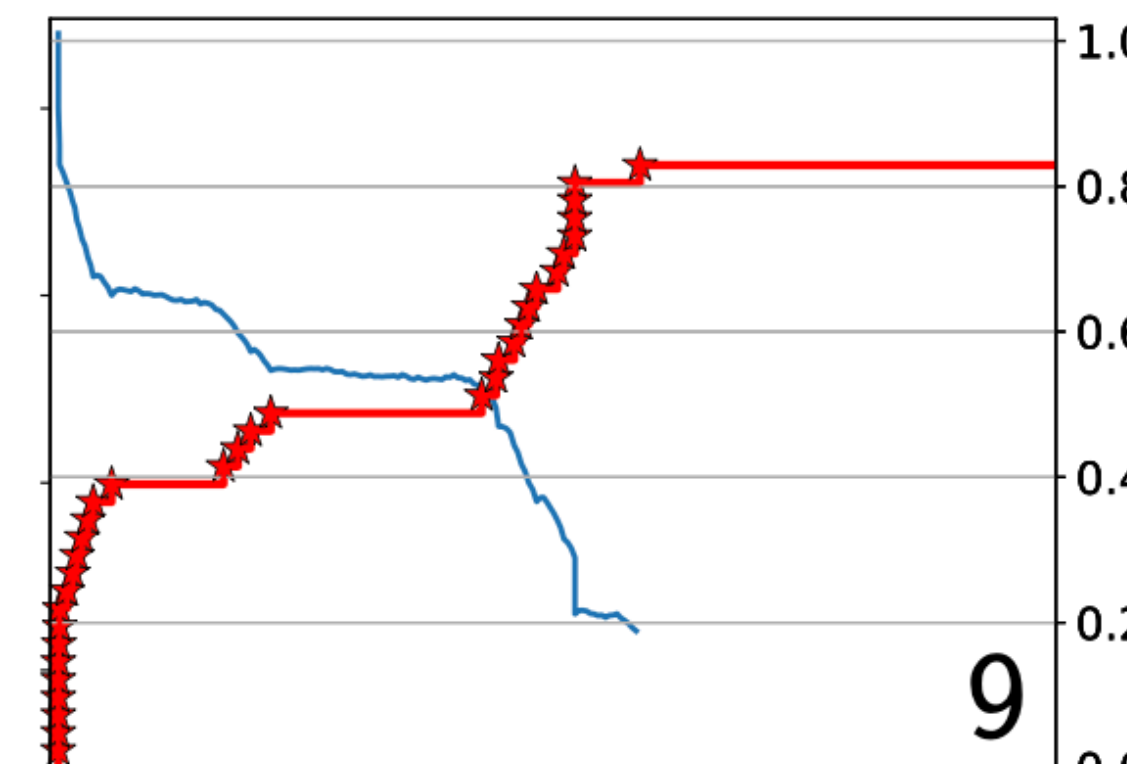
evaluations

3



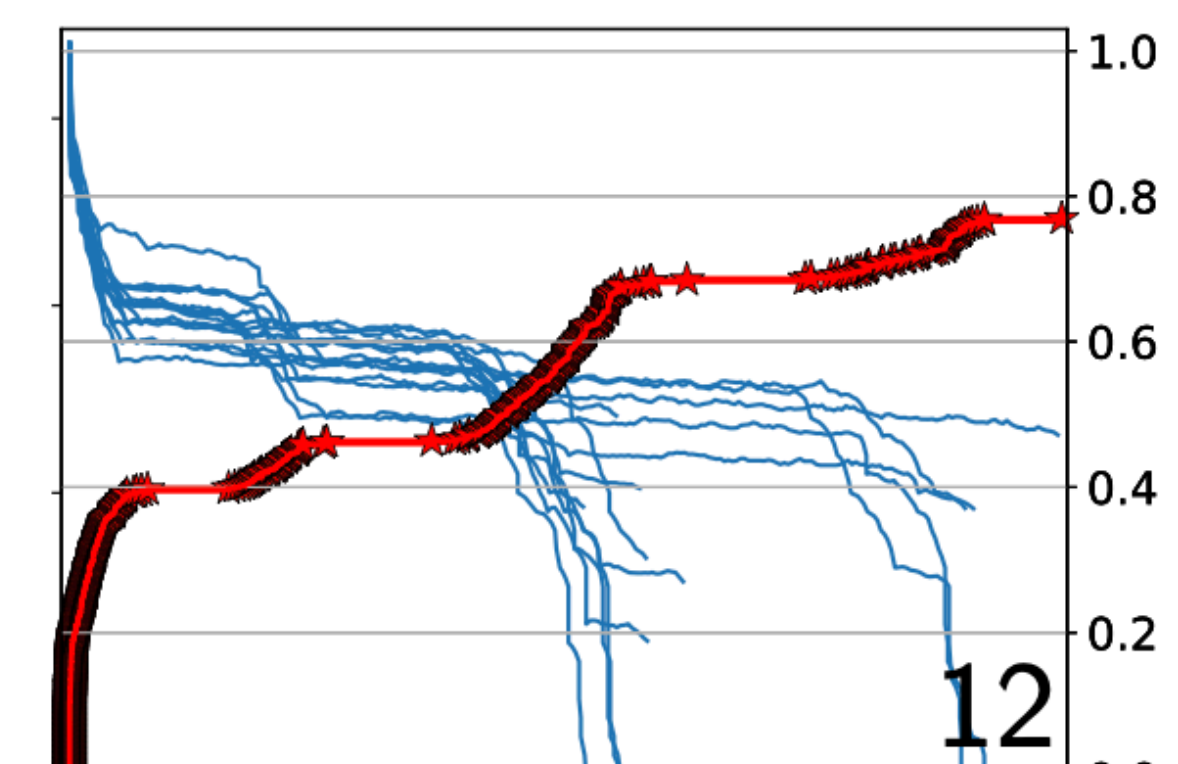
evaluations

6



evaluations

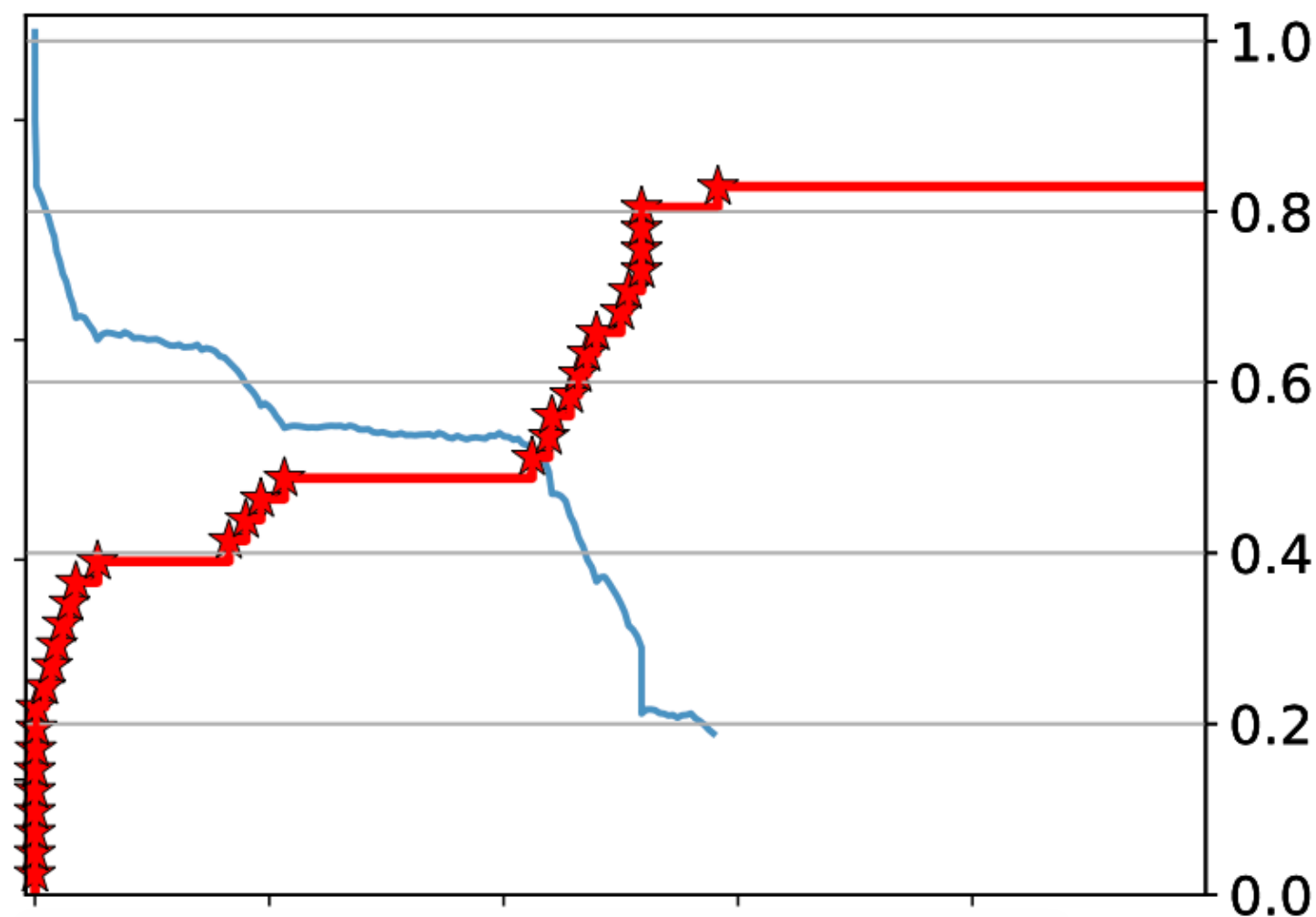
9



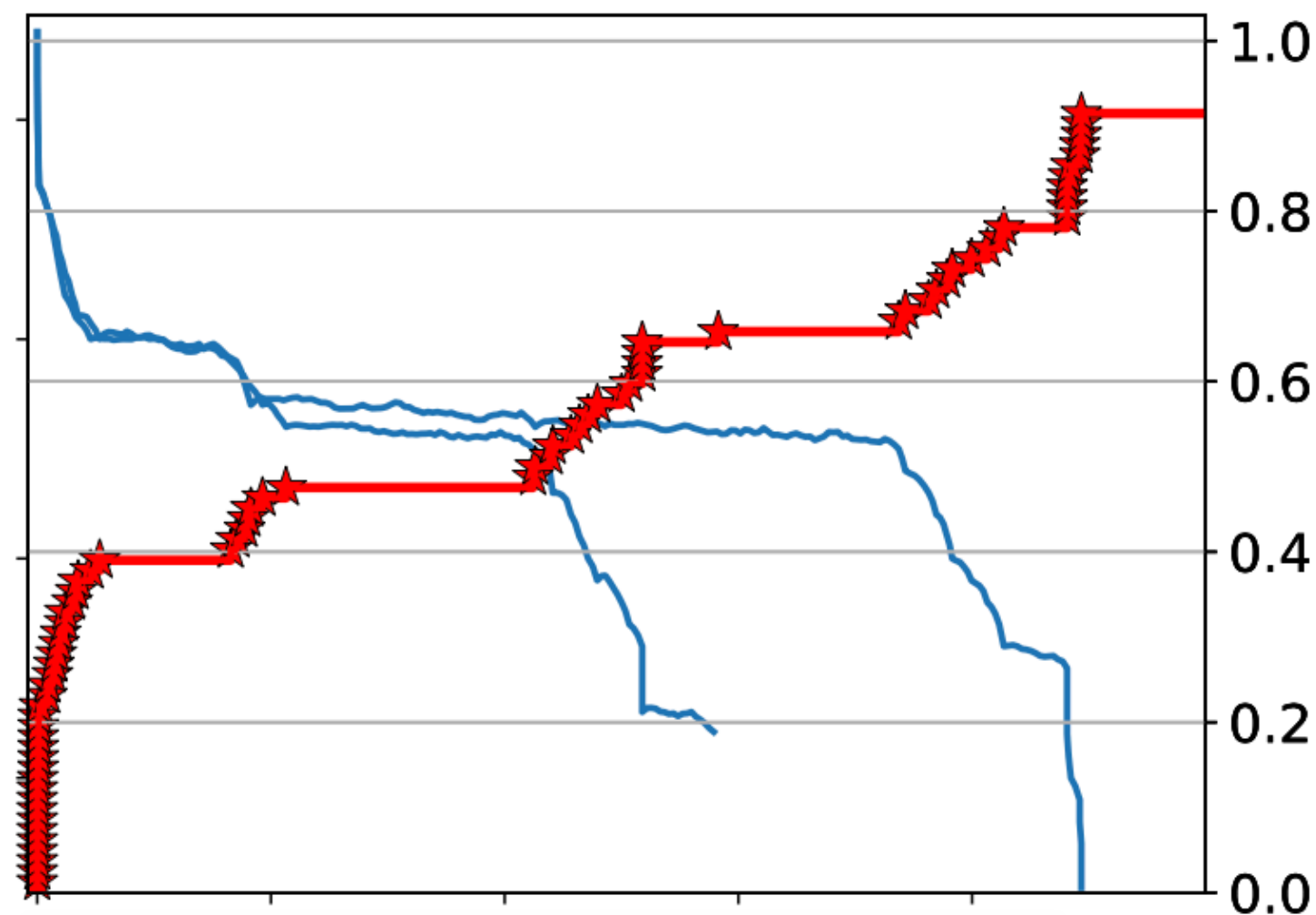
evaluations

12

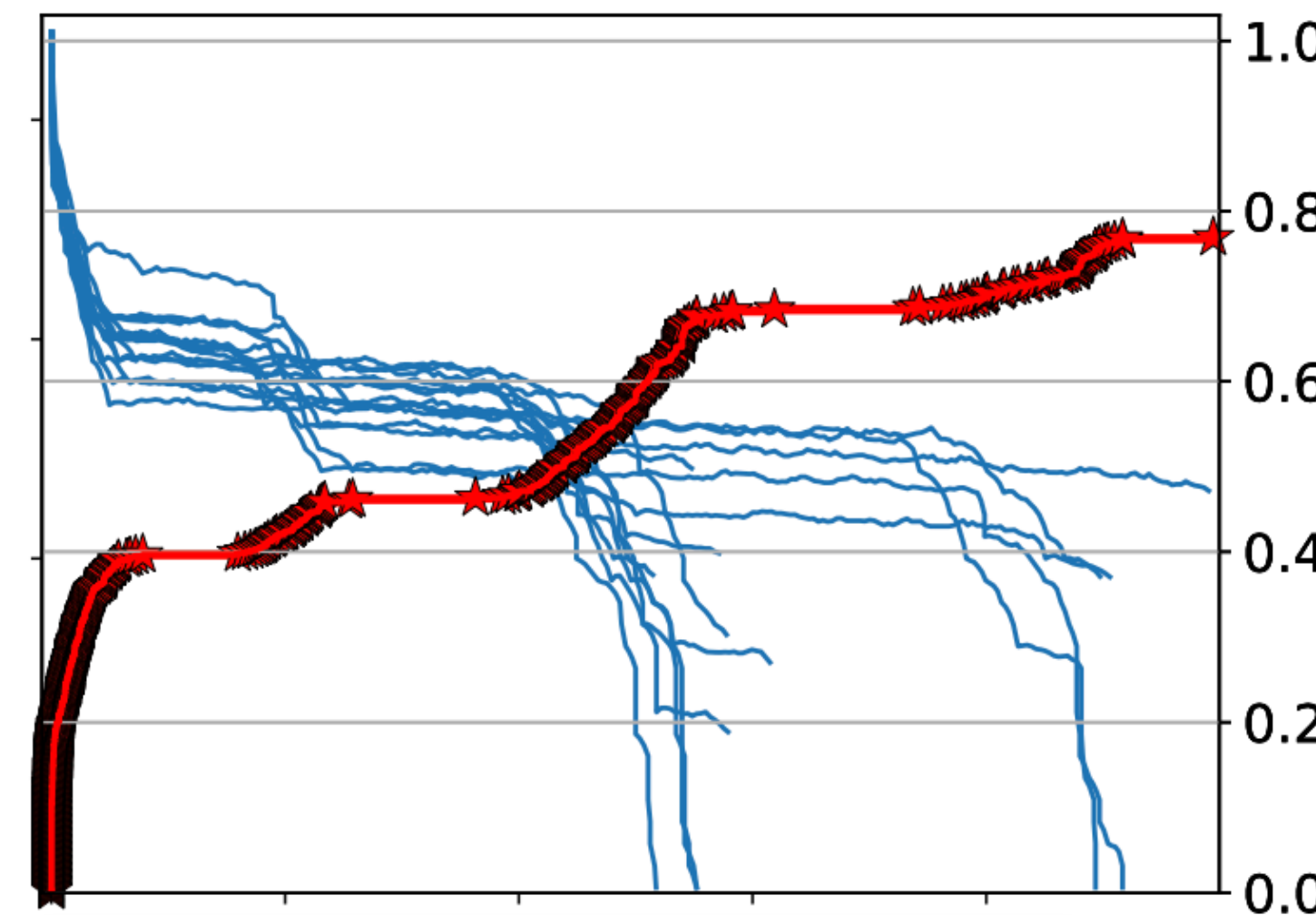
Aggregation of Several Convergence Graphs



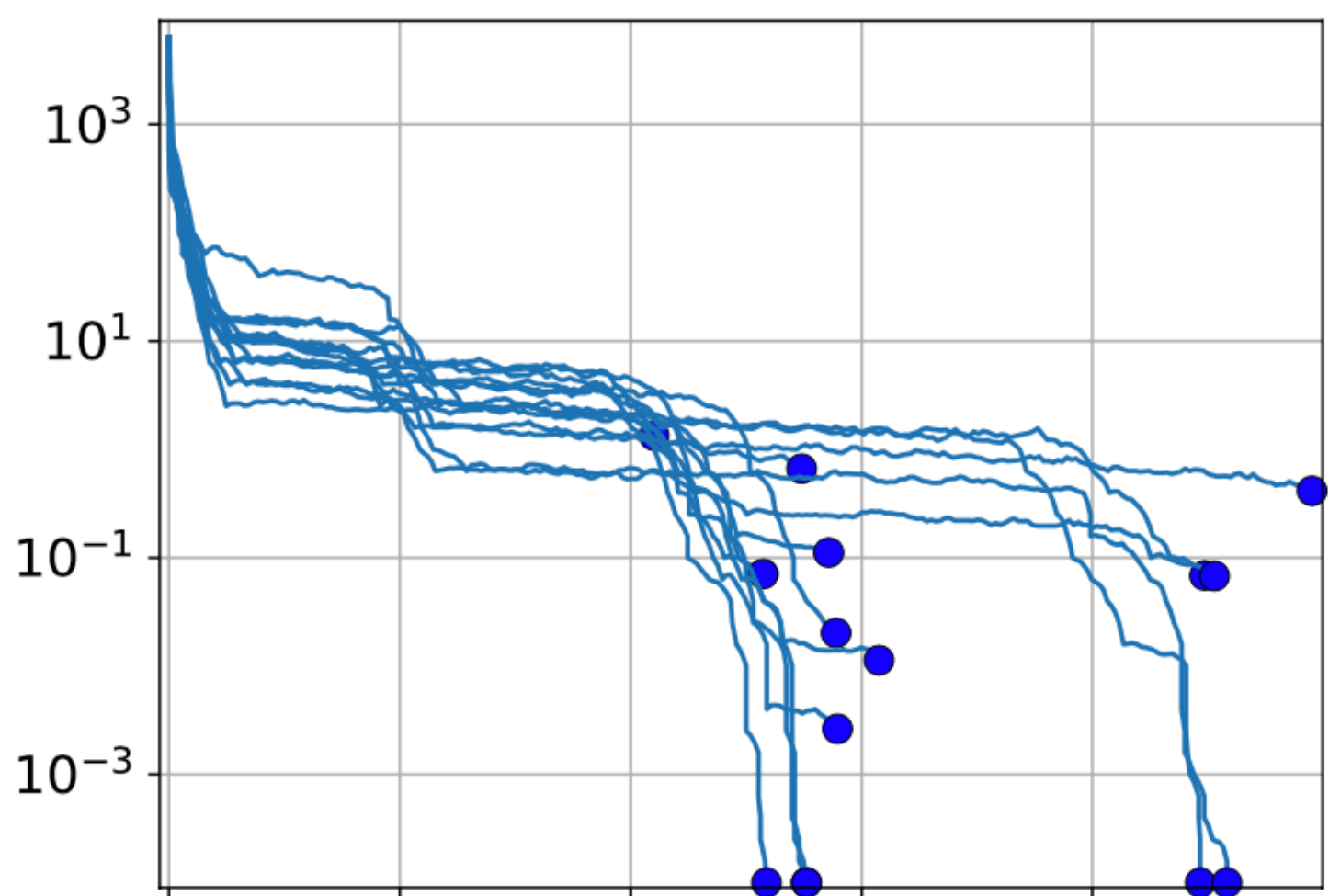
evaluations



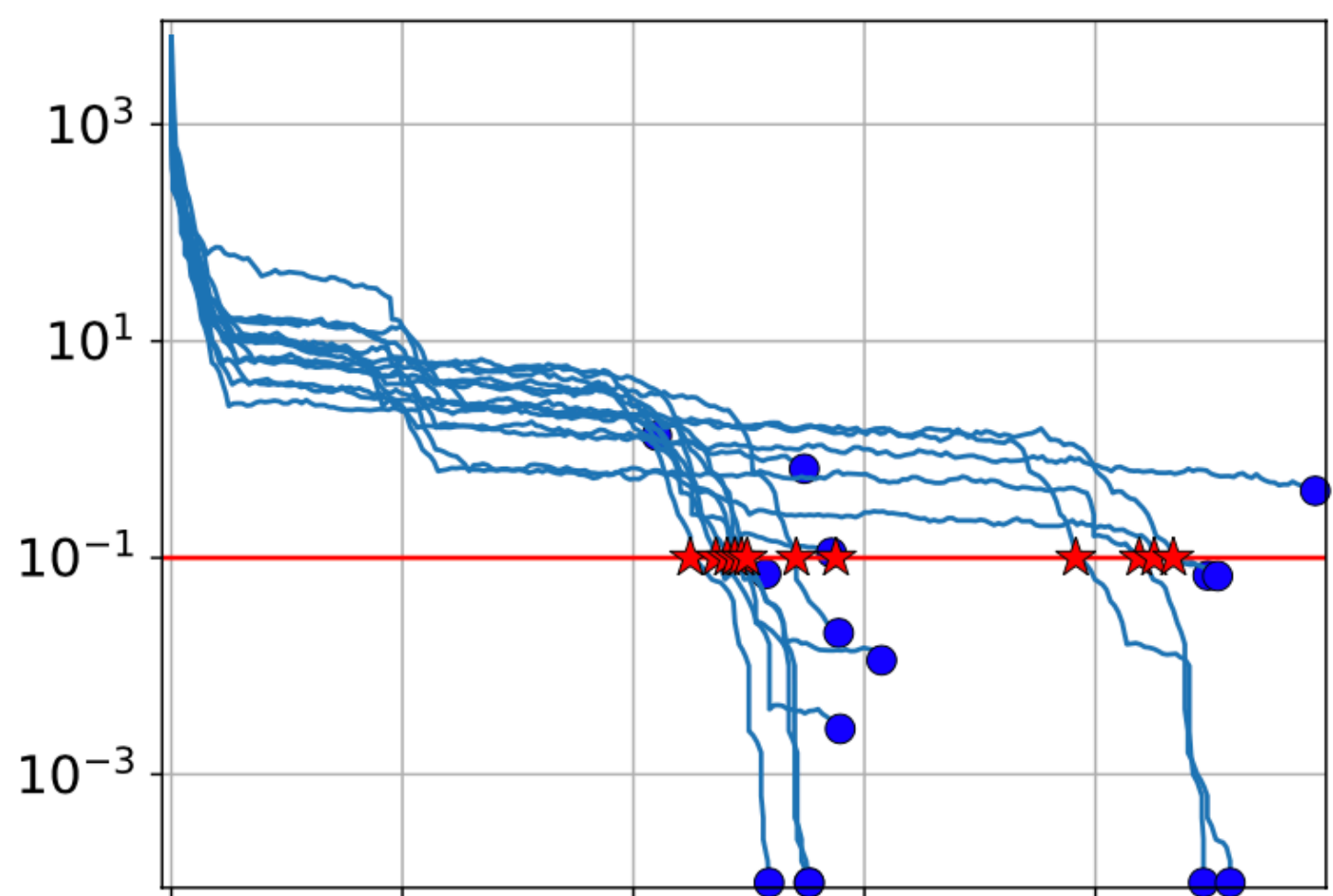
evaluations



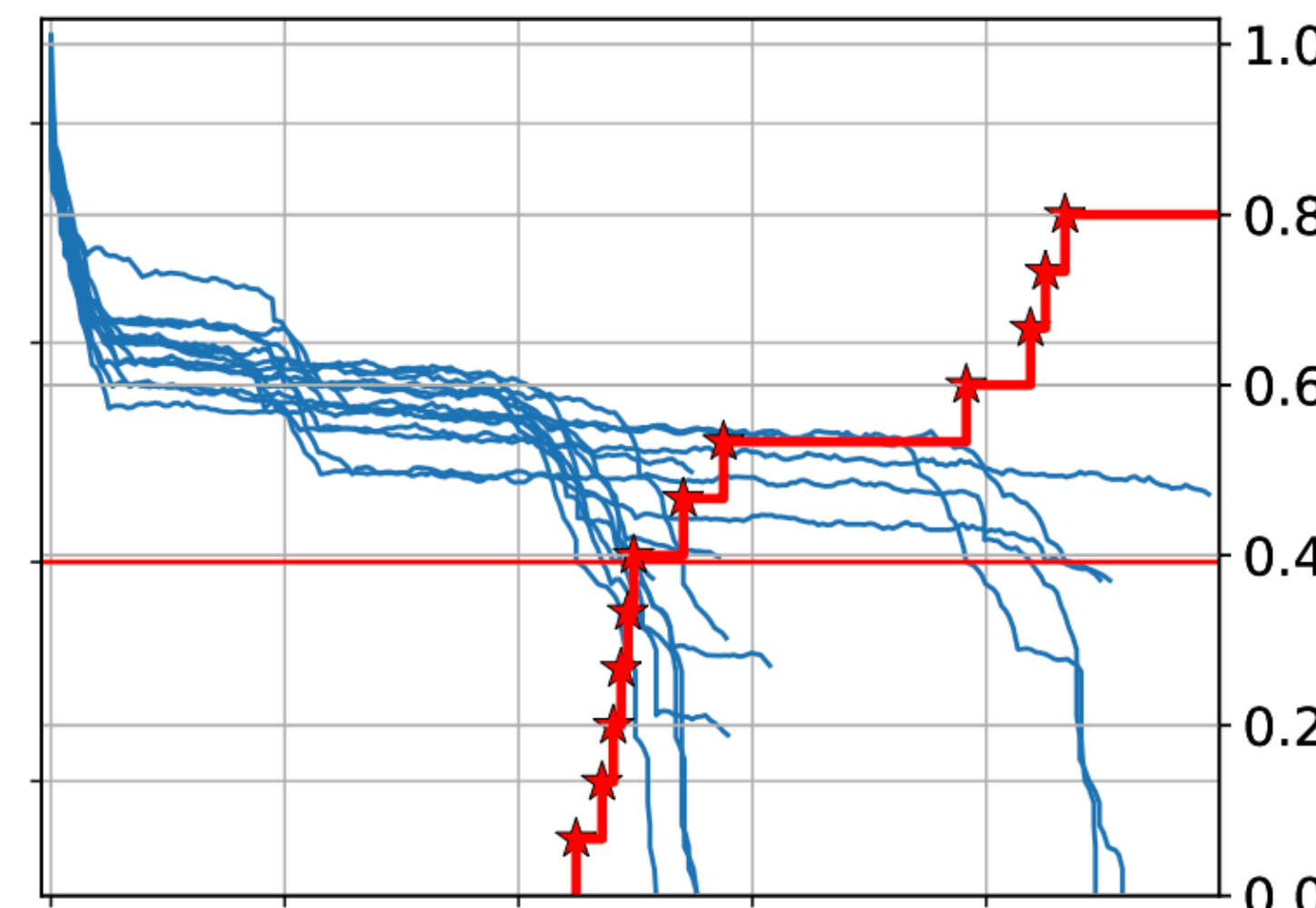
evaluations



evaluations

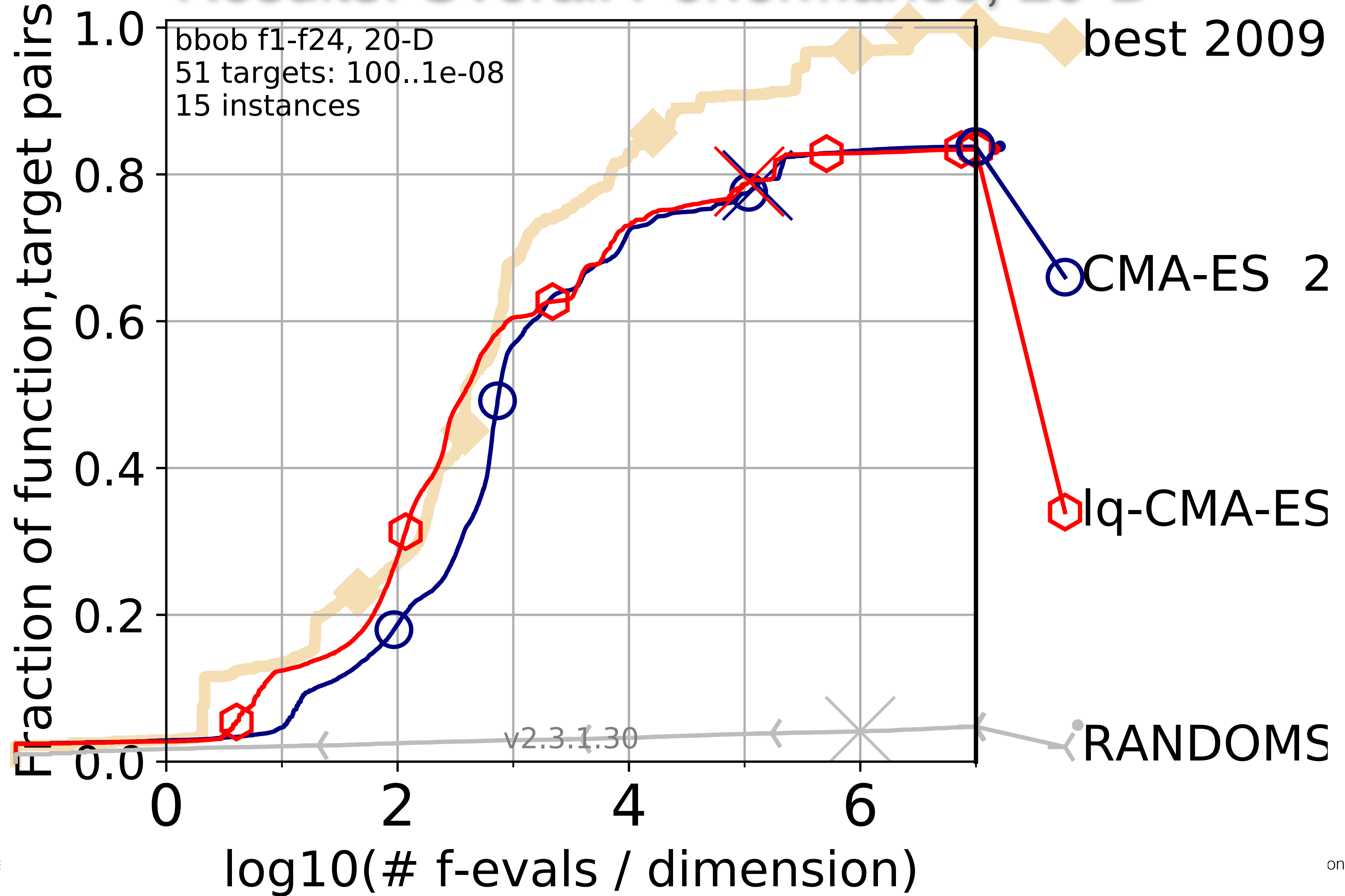


evaluations



evaluations

Results: Overall Performance, 20-D



Data and Performance Profiles

Data Profile

Given $T_{p,s}$ a collection of runtime (#of f-evals) for a solver s to reach **a certain target** on a problem $p \in \mathcal{P}$.

The data profile is the ECDF of $\{T_{p,s}/(n+1), p \in \mathcal{P}\}$:

Normalization is done because runtime associated to different dimensions are put together

•
Benchmarking Derivative-Free Optimization Algorithms by J. Moré and S. Wild. SIAM J. Optimization, Vol. 20 (1), pp.172-191, 2009.

Data Profile

Given $T_{p,s}$ a collection of runtime (#of f-evals) for a solver s to reach **a certain target** on a problem $p \in \mathcal{P}$.

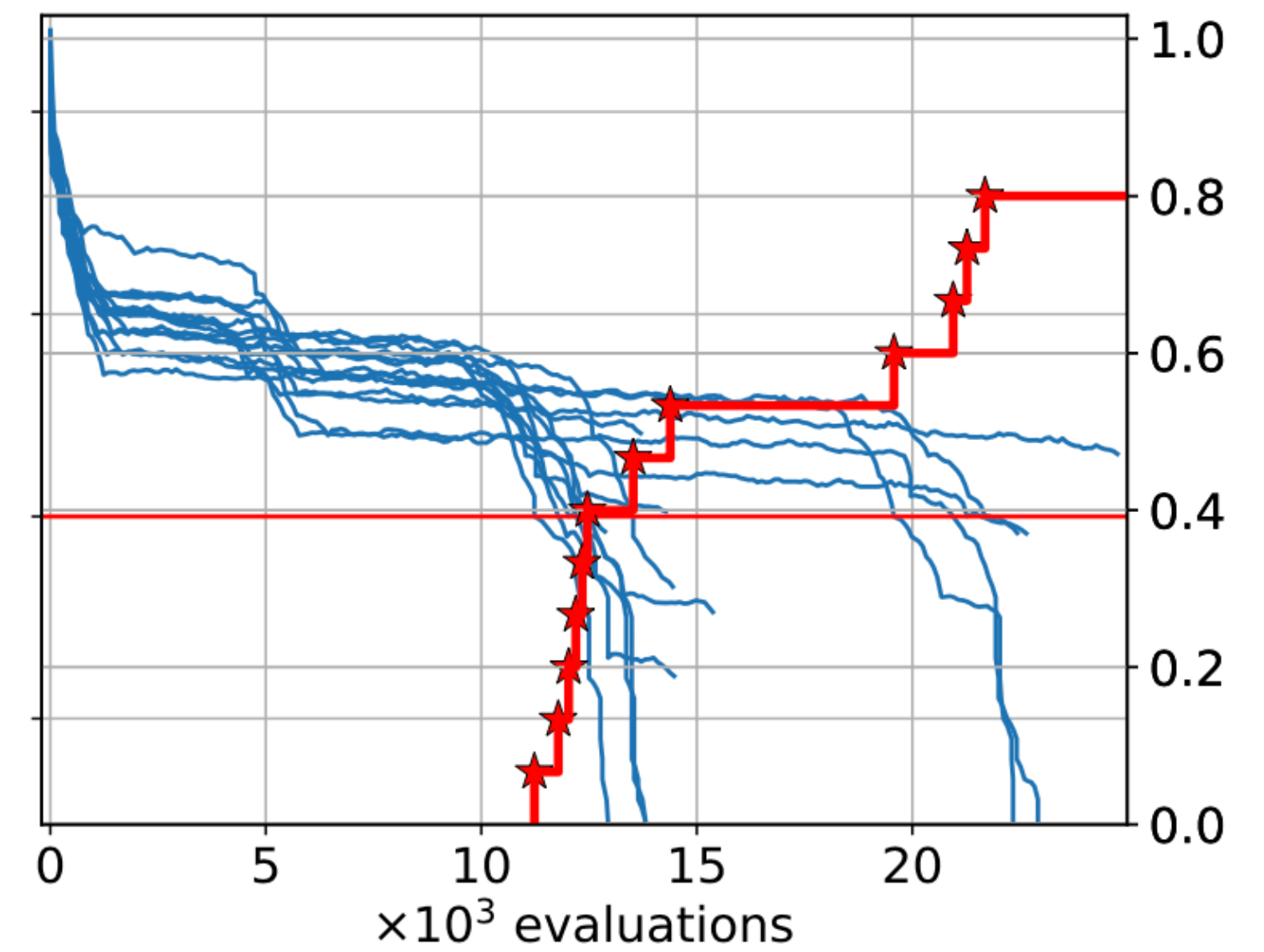
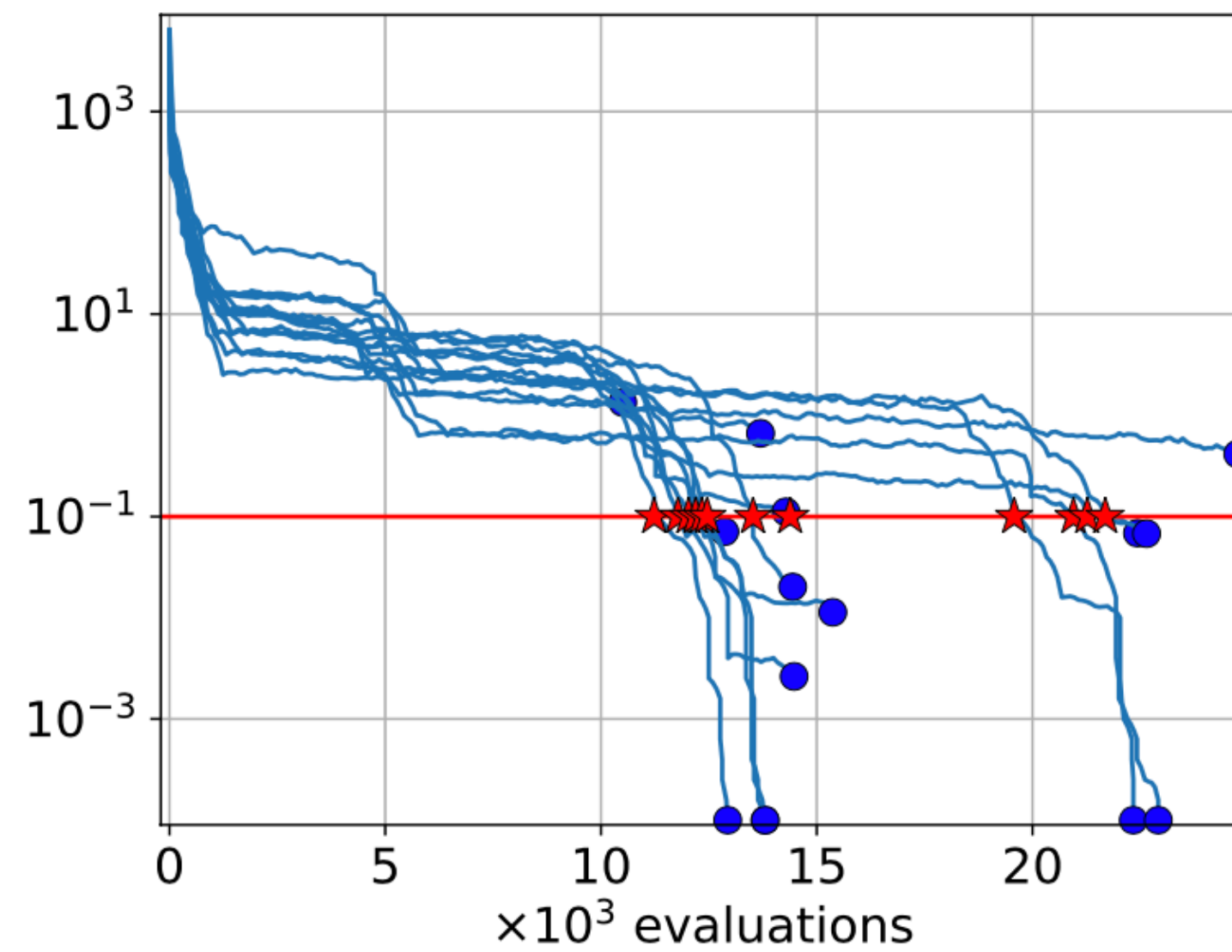
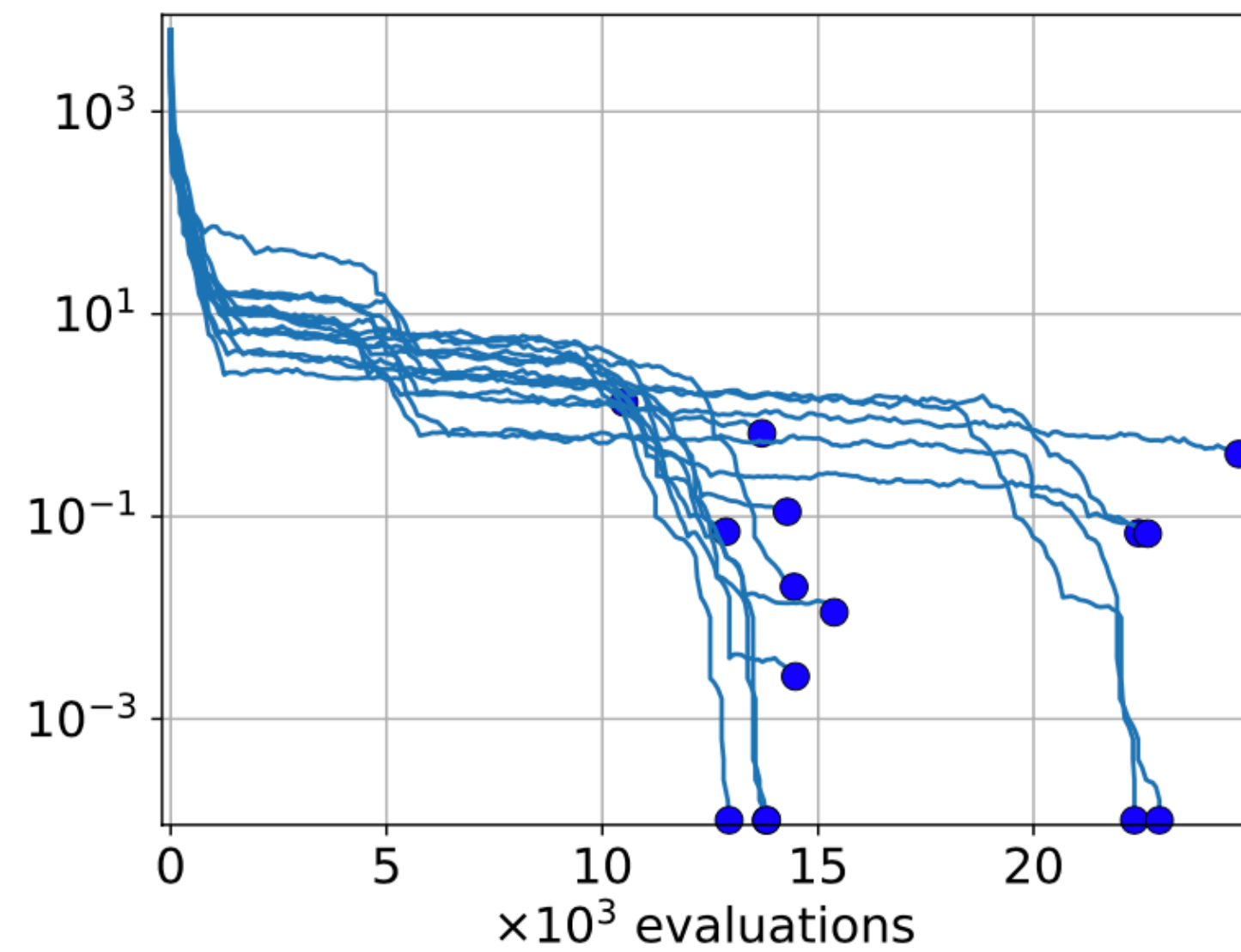
The data profile is the ECDF of $\{T_{p,s}/(n+1), p \in \mathcal{P}\}$:

Normalization is done because runtime associated to different dimensions are put together

$$\text{ECDF}_{\{T_{p,s}/(n+1), p \in \mathcal{P}\}}(t) = \frac{1}{|\mathcal{P}|} \sum_{p=1}^{|\mathcal{P}|} \mathbf{1}_{\left\{\frac{T_{p,s}}{n+1} \leq t\right\}}$$

•
Benchmarking Derivative-Free Optimization Algorithms by J. Moré and S. Wild. SIAM J. Optimization, Vol. 20 (1), pp.172-191, 2009.

Data Profile



Targets may be different for each function, but choosing a different target or shifting the respective graph vertically is the same

Performance Profile

Normalize runtime by performance of best solver: Define the performance on a problem p by a solver s as the runtime divided by the runtime of best solver among a set of solvers \mathcal{S}

$$r_{p,s} = \frac{T_{p,s}}{\min\{T_{p,s} : s \in \mathcal{S}\}}$$

! It “Removes” the order of magnitude of $T_{p,s}$ and thus the information of difficulty

The **performance profile of a solver s** is the ECDF of $\{r_{p,s}, p \in \mathcal{P}\}$:

$$\text{ECDF}_{\{r_{p,s}, p \in \mathcal{P}\}}(t) = \frac{1}{|\mathcal{P}|} \sum_{p=1}^{|\mathcal{P}|} 1_{\{r_{p,s} \leq t\}}$$

E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, Math. Program., 91 (2002), pp. 201–213.

Data and Performance Profile: Discussion

- Performance and Data profiles are just **ECDF of (normalized) runtime** associated to a single target per problem
- Performance profile
 - normalized by the smallest (best) runtime
 - relative to the set of solvers benchmarked
 - difficult to compare across papers*
- we do not see the problem difficulty anymore: normalization removes absolute value

Aggregation of Data

- is necessary

e.g., BBOB takes about $24 \times 15 \times 30 \approx 10,000$ single measurements for each algorithm in each dimension

- **implicit assumption**: uniform distribution over all problems we aggregate over shall somewhat reflect the problem distribution in reality

- properties that can be **inexpensively probed** should not (never) be aggregated over different values

For example: dimensionality. Why?

- any **runtimes** can be meaningfully aggregated

Assuming they come in the same unit of measurement (here evaluations).

However: not all ways to aggregate runtimes are meaningful.

We should use a log scale when they come from different distributions.

- **successful and unsuccessful runs** can be meaningfully aggregated,

solving the fast vs successful comparison “dilemma” once and for all.

Using simulated restarts or Enes/ERT/SP2, see “Treating success probabilities”.

Aggregation of Data

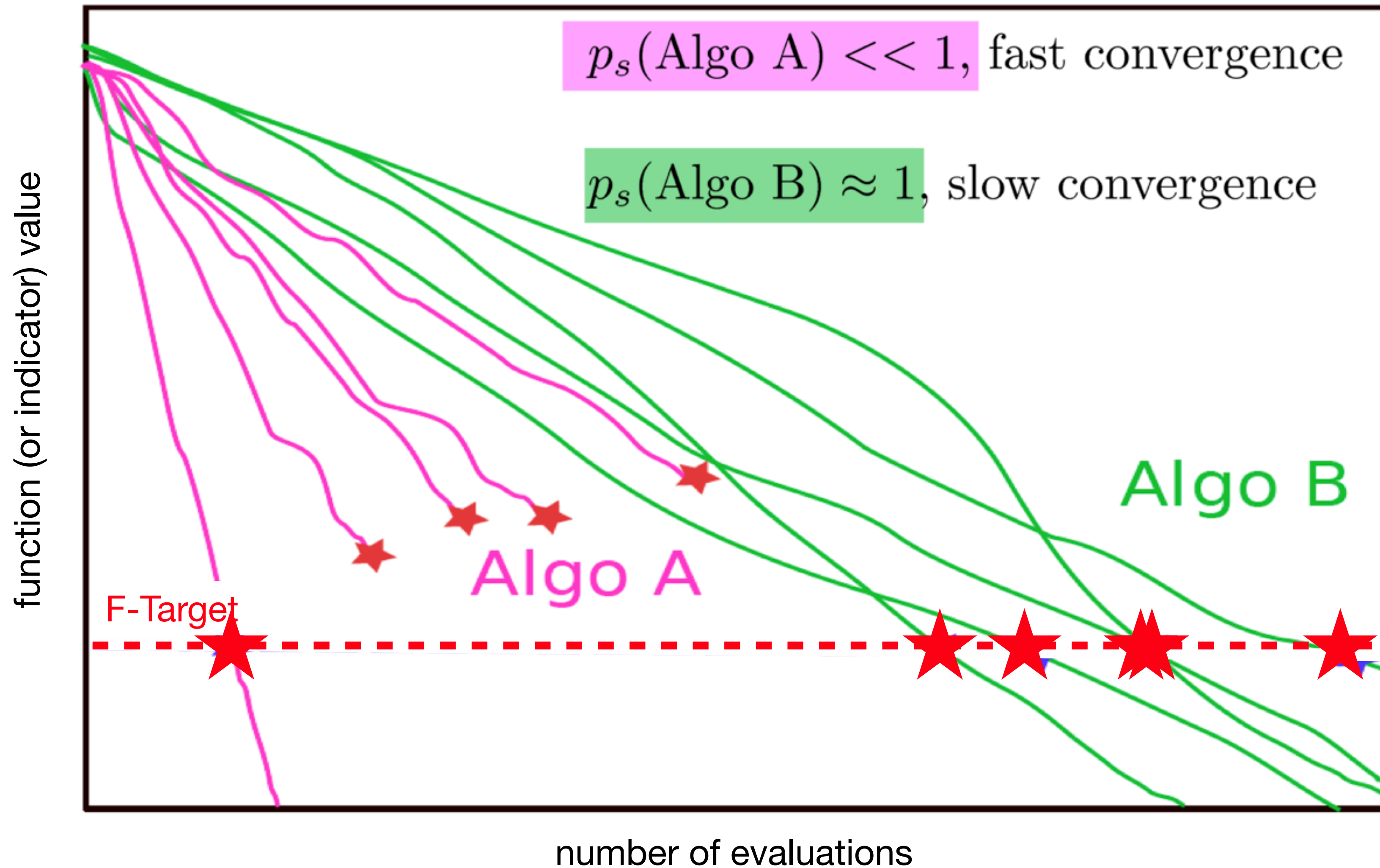
1. aggregating **repetitions** over the same (or very similar) problem(s)
in particular with unsuccessful trials
2. aggregating data from **different problems**

Expected RunTime (ERT)

Aggregated measurement of repetitions

Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma



Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma

We can **simulate a runtime distribution** by simulated (artificial) restarts using the given independent runs

Algo Restart A:



$$p_s(\text{Algo Restart A}) = 1$$

Algo Restart B:



$$p_s(\text{Algo Restart B}) = 1$$

Caveat: the performance of algorithm A critically depends on termination methods (before to hit the target)

which reflects the situation on a practical problem unless many runs can be done in parallel

Expected Runtime of Restart Algorithm

Algo Restart A:



$$p_s(\text{Algo Restart A}) = 1$$

Algo Restart B:



$$p_s(\text{Algo Restart B}) = 1$$

comparable runtimes

Expected Runtime of Restart Algorithm:

$$\mathbb{E}[RT^r] = \left(\frac{1}{p_s} - 1 \right) \mathbb{E}[RT_{\text{unsucc}}] + \mathbb{E}[RT_{\text{success}}]$$

Expected time to see the first success

Expected RunTime - ERT

Expected runtime (ERT, aka Enes, SP2, aRT) estimates $\mathbb{E}[RT^r]$

$$\text{ERT} = \frac{\text{\#evaluations(until to hit target or stop)}}{\text{\#successes}}$$

unsuccessful runs count
(only) in the nominator

defined (only) for $\text{\#successes} > 0$

$$\mathbb{E}[RT^r] = \left(\frac{1}{p_s} - 1 \right) \mathbb{E}[RT_{\text{unsucc}}] + \mathbb{E}[RT_{\text{succ}}]$$

$$\text{ERT} = \left(\frac{N_{\text{success}} + N_{\text{unsucc}}}{N_{\text{success}}} - 1 \right) \text{avg}(\text{eval}_{\text{unsucc}}) + \text{avg}(\text{eval}_{\text{succ}})$$

$$= \left(\frac{N_{\text{unsucc}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{unsucc}}) + \text{avg}(\text{eval}_{\text{succ}})$$

odds ratio

ERT Related Performance Measures

$$\text{ERT} = \left(\frac{N_{\text{unsuccess}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{unsucc}}) + \text{avg}(\text{eval}_{\text{succ}})$$

$$\approx \left(\frac{N_{\text{unsuccess}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{succ}}) + \text{avg}(\text{eval}_{\text{succ}})$$

$$= \left(\frac{N_{\text{unsuccess}} + N_{\text{success}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{succ}})$$

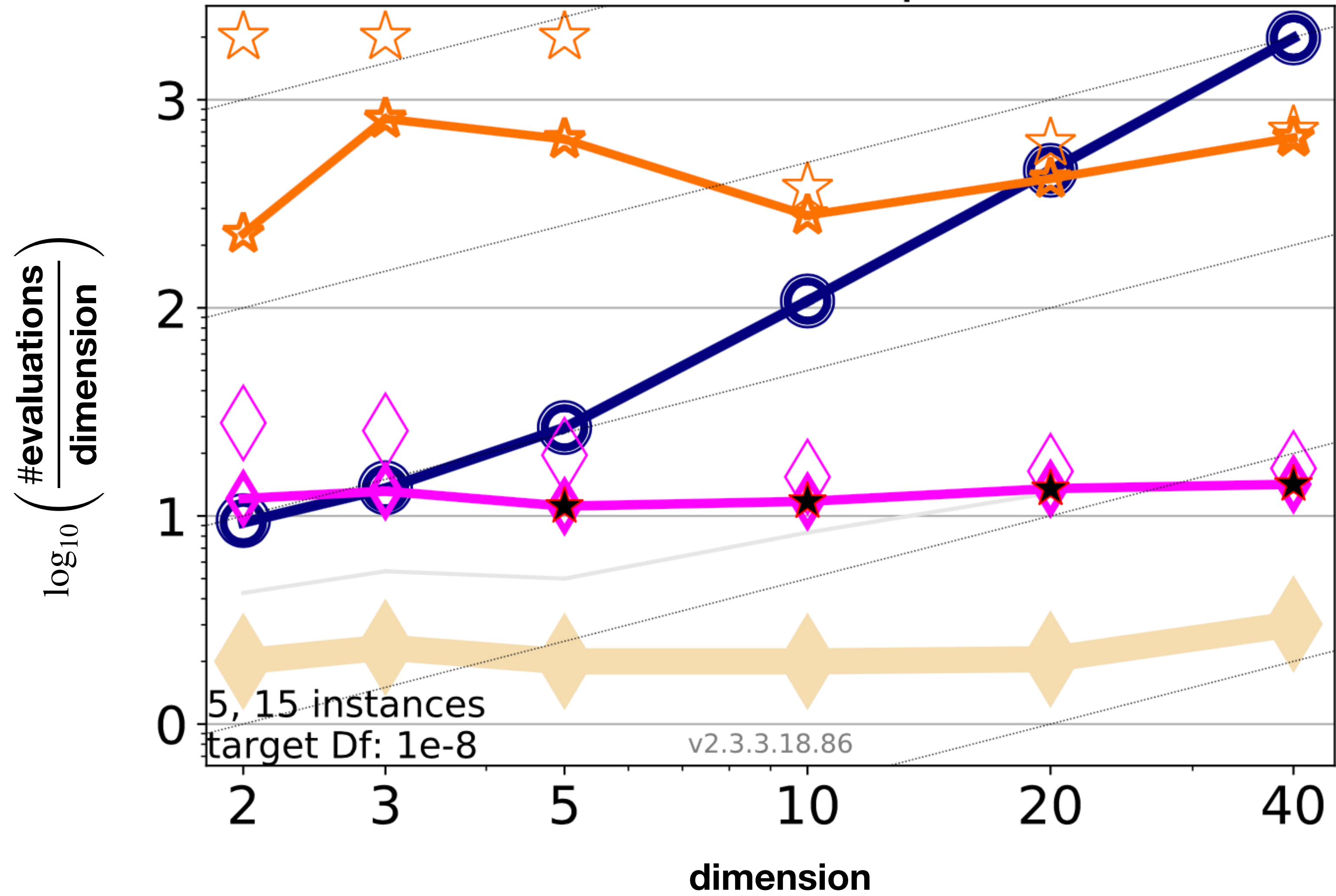
$$= \left(\frac{1}{\text{success rate}} \right) \text{avg}(\text{eval}_{\text{succ}})$$

may or may not
be the case

The last three lines are AKA Q-measure or SP1 (success performance).

See [Price 1997] and [Auger&Hansen 2005]

On Scaling



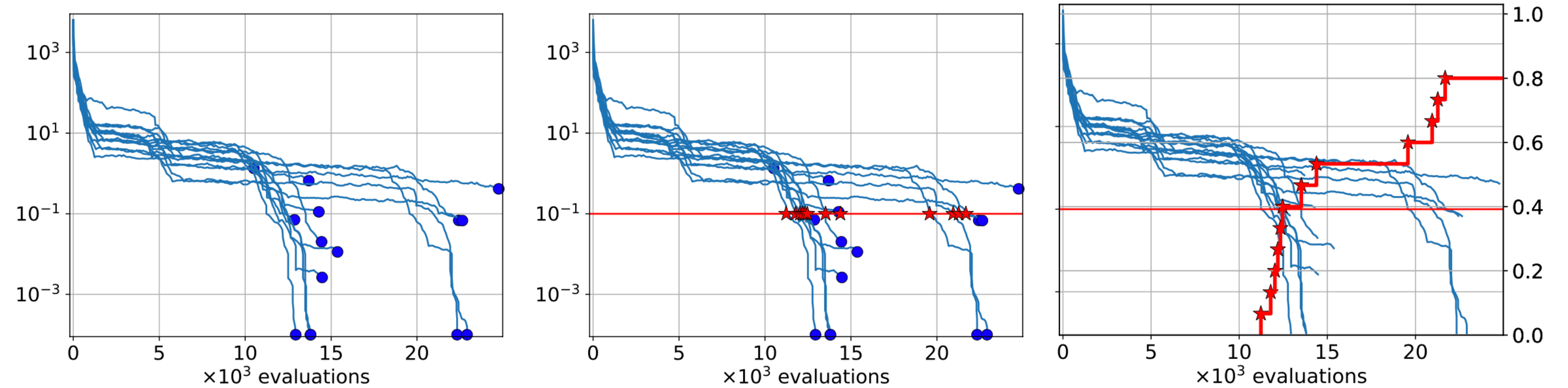
Aggregation of Data

1. aggregating **repetitions** over the same (or very similar) problem(s)
in particular with unsuccessful trials
2. aggregating data from **different problems**
already a single convergence graph contains different problems

Aggregation of Data: ECDFs With Different Problems

- ECDFs (re-)order the data (sort the data)

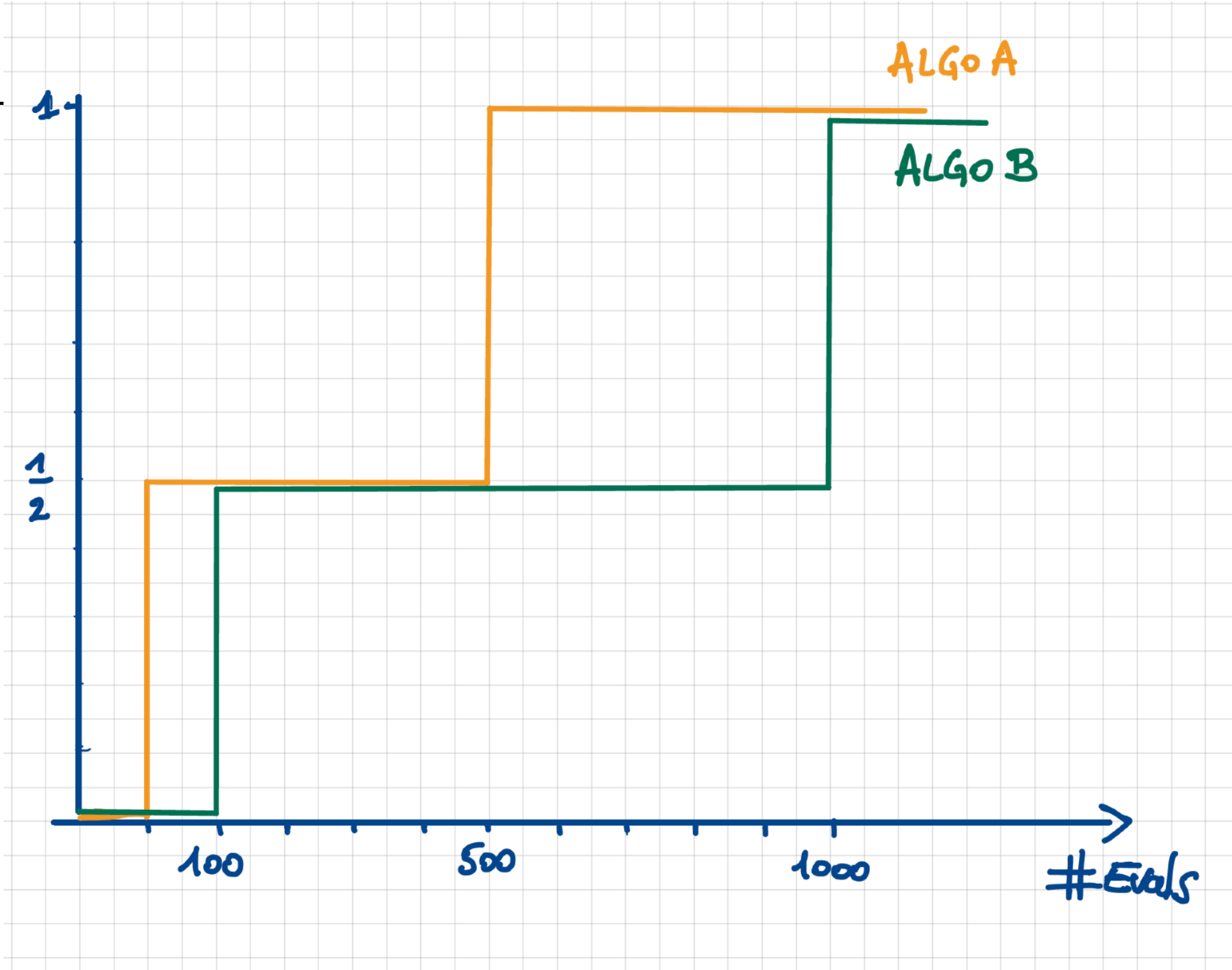
*hence we **lose the problem label**
single convergence graph ECDFs are not affected*



Discussion of Aggregation (Caveat)

	Problem 1	Problem 2
Algorithm A	50	500
Algorithm B	100	1000

Algorithm A = 2 x faster



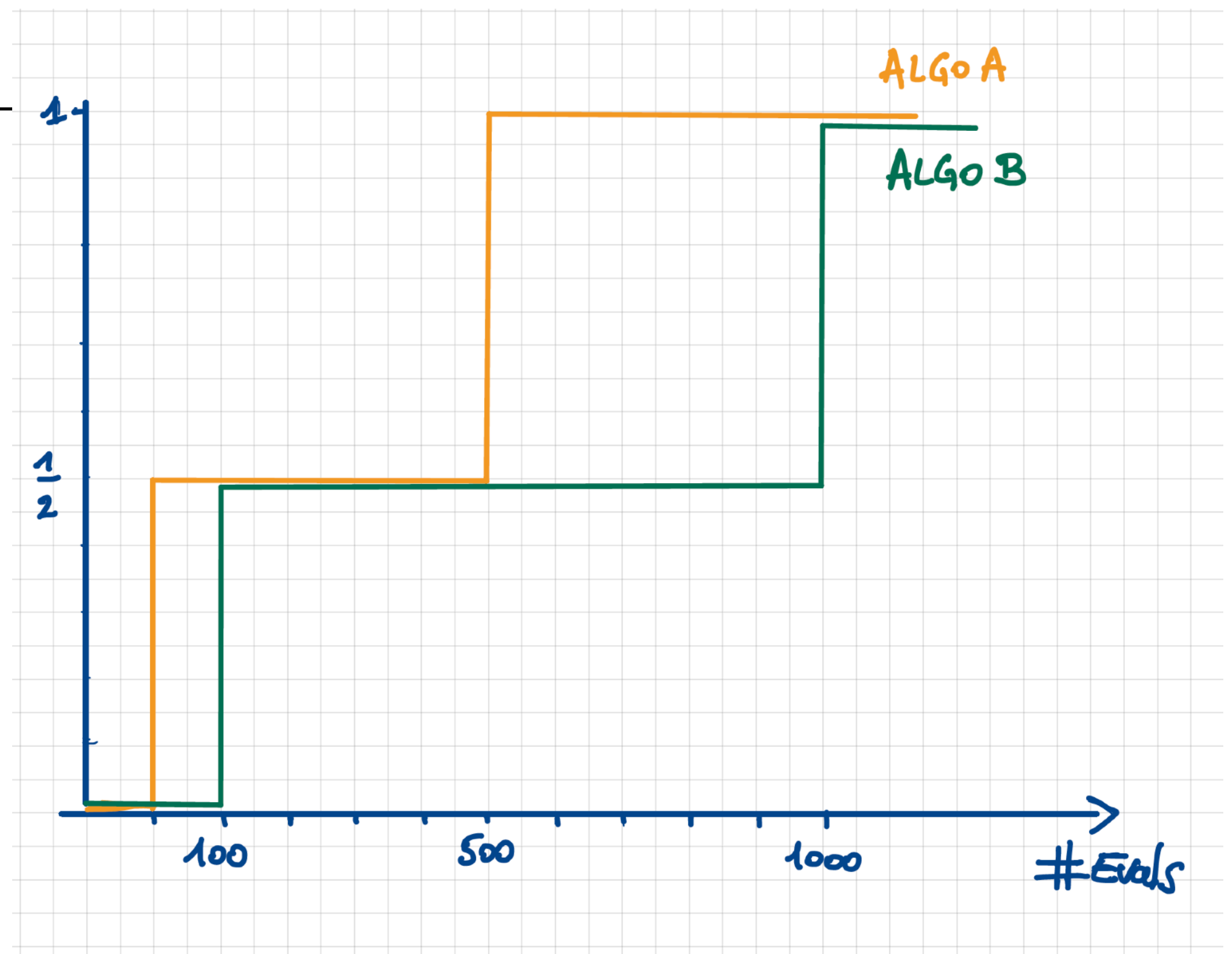
Discussion of Aggregation (Caveat)

	Problem 1	Problem 2
Algorithm A	50	500
Algorithm B	1000	100

Algorithm B = 5 x faster Algorithm A

Algorithm A = 20 x faster Algorithm B

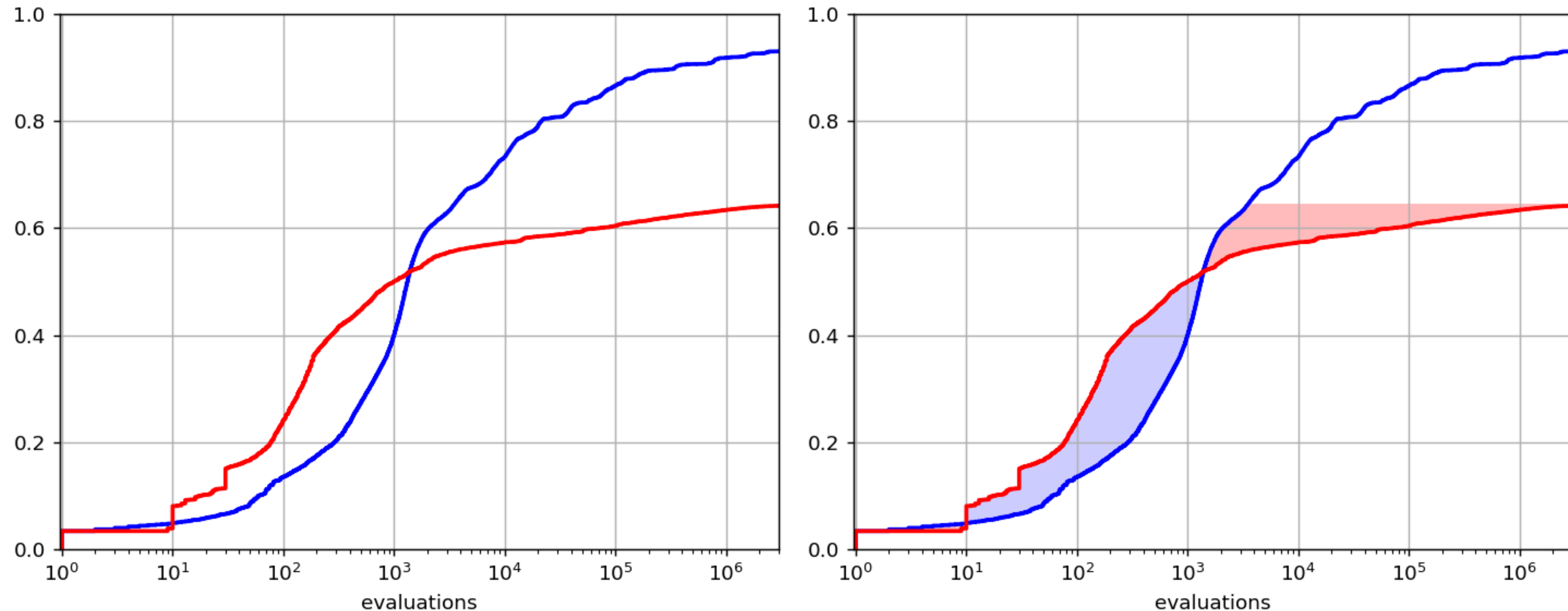
Domination in each point of an empirical runtime distribution does not imply better performance on each problem!



- The **average runtime ratio**

$$\frac{\exp\left(\frac{1}{k} \sum_i \log(B_i)\right)}{\exp\left(\frac{1}{k} \sum_i \log(A_i)\right)} = \exp\left(\frac{1}{k} \sum_i \log(B_i) - \frac{1}{k} \sum_i \log(A_i)\right) = \exp\left(\frac{1}{k} \sum_i \log\left(\frac{B_i}{A_i}\right)\right)$$

is the area between the runtime distribution graphs of two algorithms A,B
is the geometric average when the x-axis is in log-scale
with the geometric average it is invariant under the exchange of operators:
the ratio of the averages equals the average of the ratios



Take Home Messages

- Select a **balanced testbed**

*furious activity is no substitute for understanding
using “all functions” is likely to introduce a bias
(too many simple or low dimensional problems)*

- Use **quantitative measurements**

*which should preferably be comparable across publications
empirical CDFs are a very useful tool*

- Don't aggregate over attributes that are simple to determine

like dimension

- Benchmarking is **tedious but necessary**

use a provided software platform?

Statistical Analysis

*“The first principle is that **you must not fool yourself**, and you are the easiest person to fool. So you have to be very careful about that. After you've not fooled yourself, it's easy not to fool other[scientist]s. You just have to be honest in a conventional way after that. ”*

— Richard P. Feynman

- Statistical analysis is secondary!
useful only after a (quantitative) conclusion has been drawn from the data.
- Don't confuse statistical (significance) testing (hypothesis *testing*) with statistical data exploration (searching for which test *of many* has a small p-value, HARKing, hypothesis *generating*)

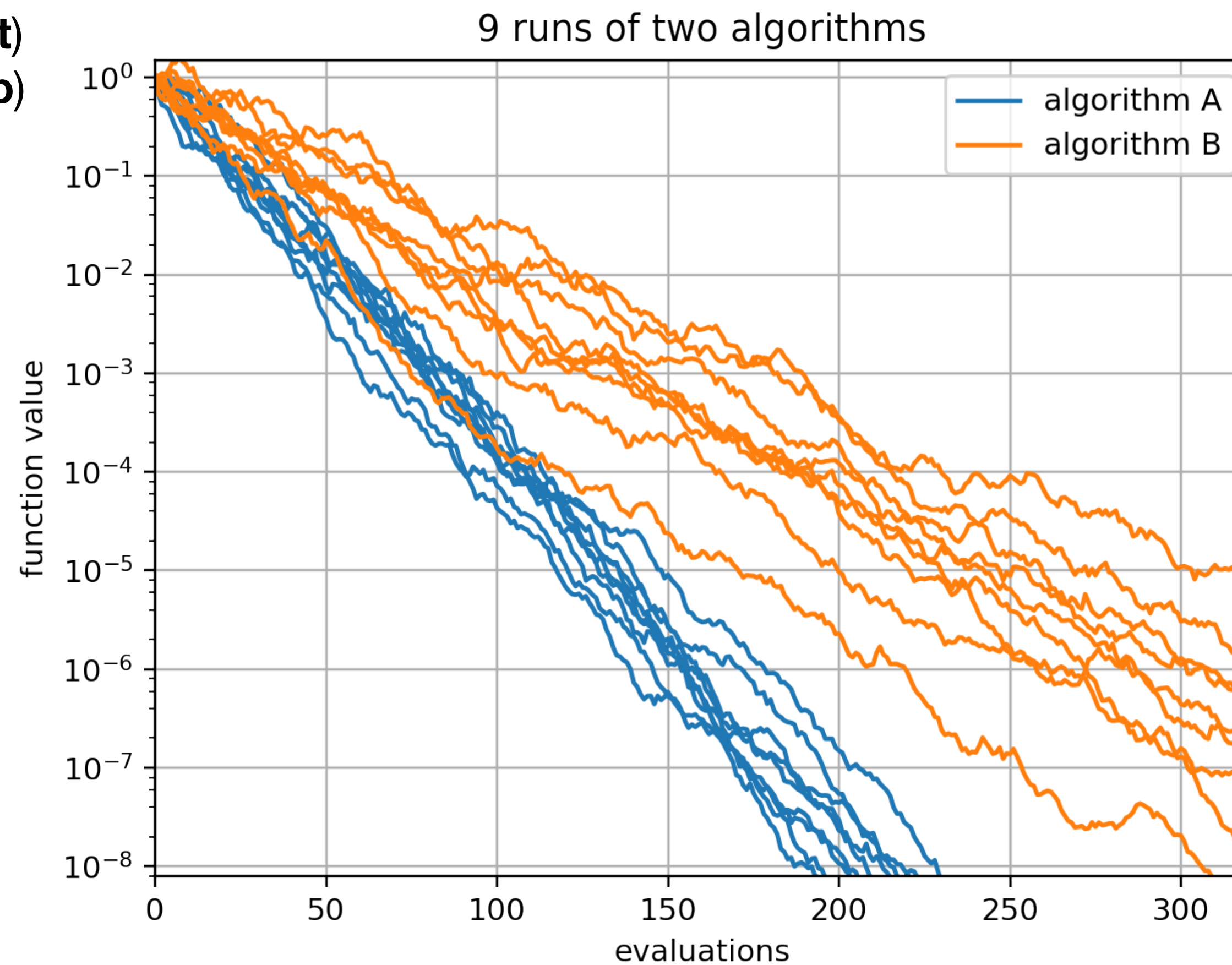
Statistical Analysis

“[...] experimental results lacking proper statistical analysis must be considered anecdotal at best, or even wholly inaccurate.”

— M. Wineberg, 2016

Do you agree (**sounds about right**) or disagree (**is a little over the top**) with the quote?

an experimental result (shown are *all data obtained*):



Statistical Analysis

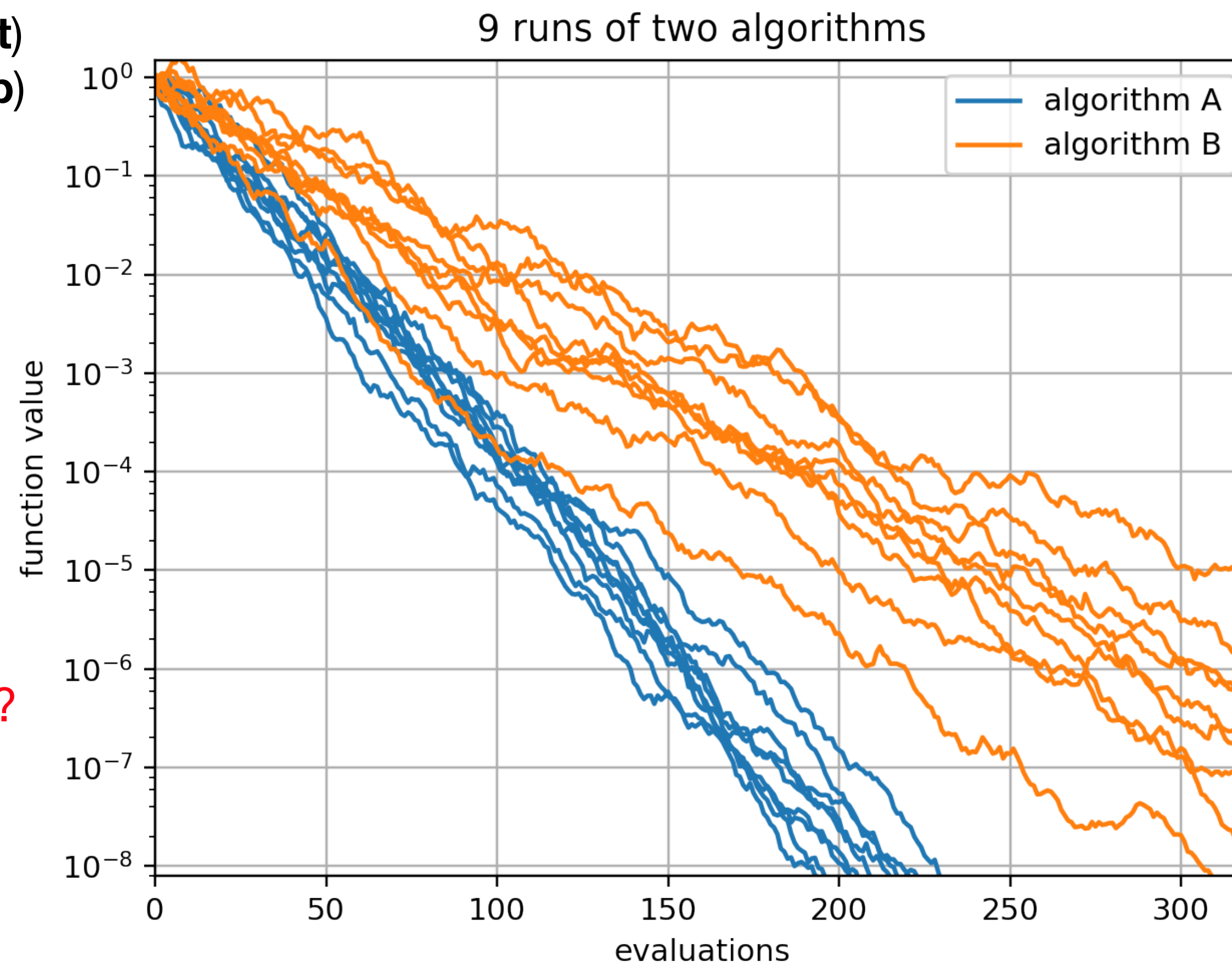
“[...] experimental results lacking proper statistical analysis must be considered anecdotal at best, or even wholly inaccurate.”

— M. Wineberg, 2016

Do you agree (**sounds about right**) or disagree (**is a little over the top**) with the quote?

an experimental result (shown are *all data obtained*):

Do we need a statistical analysis?



What about the p -value?

The “ p -value of statistical significance” is the *probability that we will observe a “false positive” outcome, $P(D | H_0)$*

i.e., the probability that the observed (or more extreme “favorable”) data D would be observed under the null hypothesis H_0 “by chance” (while in reality there is no “effect” to be seen).

It is straight forward to *compute the posterior odds* that H_0 is true *from the observed p -value*:

posterior odds for $H_0 \approx$ prior odds for H_0 times $2p$

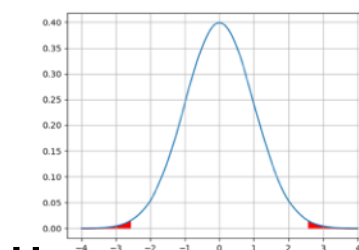
$$\underbrace{\frac{P(H_0 | D)}{P(\neg H_0 | D)}}_{\text{posterior odds}} = \underbrace{\frac{P(H_0)}{P(\neg H_0)}}_{\text{prior odds}} \underbrace{\frac{p}{P(D | \neg H_0)}}_{\text{update (Bayes factor)}}$$

- if we try to improve a state-of-the-art algorithm, *the prior odds of H_0 is usually high (the odds of success is low)*
as a rule of thumb, the higher the prior odds for H_0 , the more interesting is the scientific question to begin with
to accept $\neg H_0$ with the same confidence we had before in H_0 , we need $p \approx P(\neg H_0)^2$
- if we are not willing to *estimate the prior odds of H_0* , we are not justified to conclude *anything* about
whether *rejecting H_0* is a reasonable decision

we can only conclude that the odds for H_0 became about $2p$ times worse

Statistical Testing: General Procedure

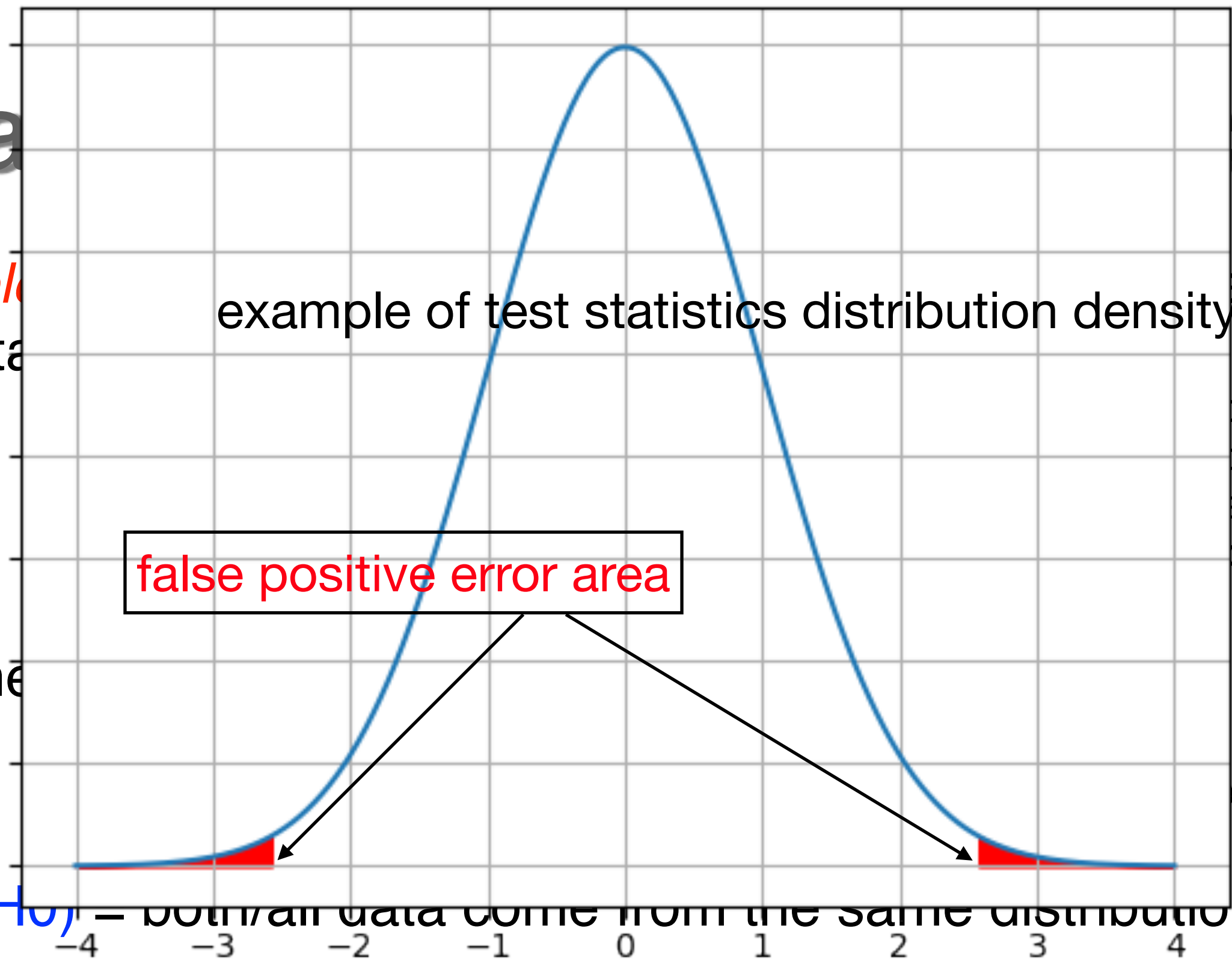
- *first, check the **relevance** of the result*, for example, of the difference which is tested for statistical significance
 - this also means: preferably do not *explorative testing* (e.g. test *all* pairwise combinations), any ever so small difference can be made *statistically* significant with a simple trick, but *not made* significant in the sense of *relevant* or *important* or *meaningful*
- we prefer “nonparametric” methods
 - not* assuming that the statistics come from a *parametrised* family of probability distributions
- **Null hypothesis (H0)** = both/all data come from the same distribution
- **p-value** = significance level = probability of a *false* positive outcome given H0 is true = probability “H0 is rejected” given H0 is true
 - smaller p-values are better
 - obsolete practice: <0.1% or <1% or <5% is usually considered as *statistically significant*
- *given* an observed p-value, **fewer data are better**
 - more data (almost inevitably) lead to smaller p-values, hence to achieve the same p-value with fewer data, the *between*-difference must be larger compared to the *within*-variation



Statistical

ture

- first, check the relationship to be tested for statistical significance



which is
combinations)
le statistically
a simple trick,
or meaningful
parametrised
y distributions

- prefer “nonparametric”

- Null hypothesis (H_0) = both/all data come from the same distribution

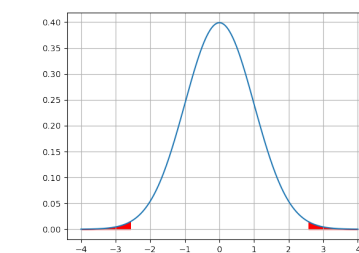


- p-value = significance level = probability of a false positive outcome given H_0 is true = probability H_0 is rejected given H_0 is true
smaller p-values are better
<0.1% or <1% or <5% is usually considered as statistically significant

- given a found/observed p-value, fewer data are better
more data (almost inevitably) lead to smaller p-values, hence to achieve the same p-value with fewer data, the between-difference must be larger compared to the within-variation

Statistical Testing: General Procedure

- *first, check the **relevance** of the result*, for example of the difference which is to be tested for statistical significance
 - this also means: do not *explorative testing* (e.g. test *all* pairwise combinations)
any ever so small difference can be made *statistically* significant with a simple trick,
but *not made* significant in the sense of *relevant* or *important* or *meaningful*
- prefer “nonparametric” methods
 - not* assuming that the data come from a *parametrised* family of probability distributions
- **Null hypothesis (H0)** = both/all data come from the same distribution
- **p-value** = significance level = probability of a *false* positive outcome given H0 is true = probability H0 is rejected given H0 is true
 - smaller p-values are better
<0.1% or <1% or <5% is usually considered as *statistically significant*
- *given* an observed p-value, ***fewer data are better***
 - more data (almost inevitably) lead to smaller p-values, hence to achieve the same p-value with fewer data, the *between*-difference must be larger compared to the *within*-variation



Statistical Testing: Methods

- we use the **rank-sum** test (`scipy.stats.ranksums`, aka Wilcoxon or Mann-Whitney U test)
 - **Assumption**: all observations (data values) are obtained independently and no equal values are observed
 - The “*lack*” of necessary preconditions is the main reason to use the rank-sum test. even a few equal values are not detrimental
 - the rank-sum test is *nearly as efficient* as the t-test which requires certain distributions for empirical mean and variance
 - for discrete data with ties: `scipy.stats.mannwhitneyu(..., alternative='two-sided')`
 - **Null hypothesis** (nothing relevant is observed if): $\Pr(x < y) = \Pr(y < x)$
 - H0: the probability to be greater or smaller (better or worse) is the same
 - the aim is to be able to reject the null hypothesis
 - Procedure: computes subsums of ranks in the ranking of all (combined) data values
 - `Alg1 = [400, 422, 440]` vs `Alg2 = [444, 490, 555]` \implies ranks: `Alg1 = [1, 2, 3]` vs `Alg2 = [4, 5, 6]`
 - **Outcome**: a *p*-value
 - the probability that the observed *or a more extreme* data set was generated under the null hypothesis; the probability to *mistakenly* reject the null hypothesis

Statistical Testing: Methods

- use the **rank-sum** test (`scipy.stats.ranksums`, aka Wilcoxon or Mann-Whitney U test)

- **Assumption**: all observations (data values) are obtained independently and no equal values are observed

The “*lack*” of necessary preconditions is the main reason to use the rank-sum test. even a few equal values are not detrimental the rank-sum test is *nearly as efficient* as the t-test which requires normal distributions for discrete data with ties: `scipy.stats.mannwhitneyu(..., alternative='two-sided')`

- **Null hypothesis** (nothing relevant is observed if): $\Pr(x < y) = \Pr(y < x)$

H0: the probability to be greater or smaller (better or worse) is the same

Alg1 = [400, 422, 440] vs Alg2 = [444, 490, 555] \implies ranks: Alg1 = [1, 2, 3] vs Alg2 = [4, 5, 6]

- Procedure: computes the sum of ranks in the ranking of all (combined) data values

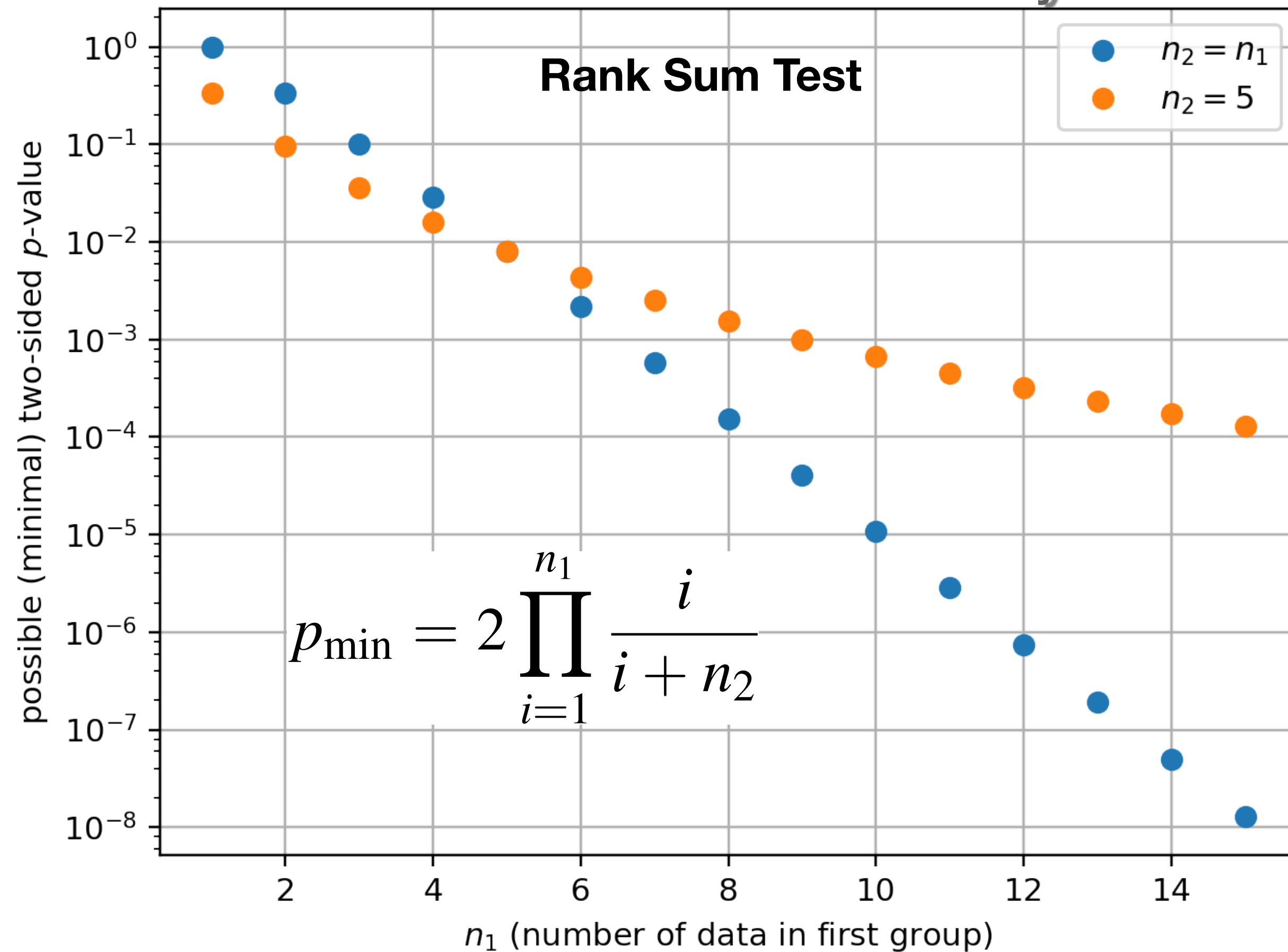
Alg1 = [400, 422, 440] vs Alg2 = [444, 490, 555] \implies ranks: Alg1 = [1, 2, 3] vs Alg2 = [4, 5, 6]

- **Outcome**: a p -value

the probability that the observed *or a more extreme* data set was generated under the null hypothesis; the probability to *mistakenly* reject the null hypothesis

Statistical Testing: How many data do we need?

AKA as test efficiency



- assumption: data are fully “separated”, that is,
 $\forall i, j : x_i < y_j$ or $\forall i, j : x_i > y_j$ (two-sided)
- observation: adding 2 data points in each group gives about one additional order of magnitude

- use the [Bonferroni correction](#) for multiple tests

simple and conservative: multiply the computed p -value by the number of tests

Statistical Testing: How many data do we need?

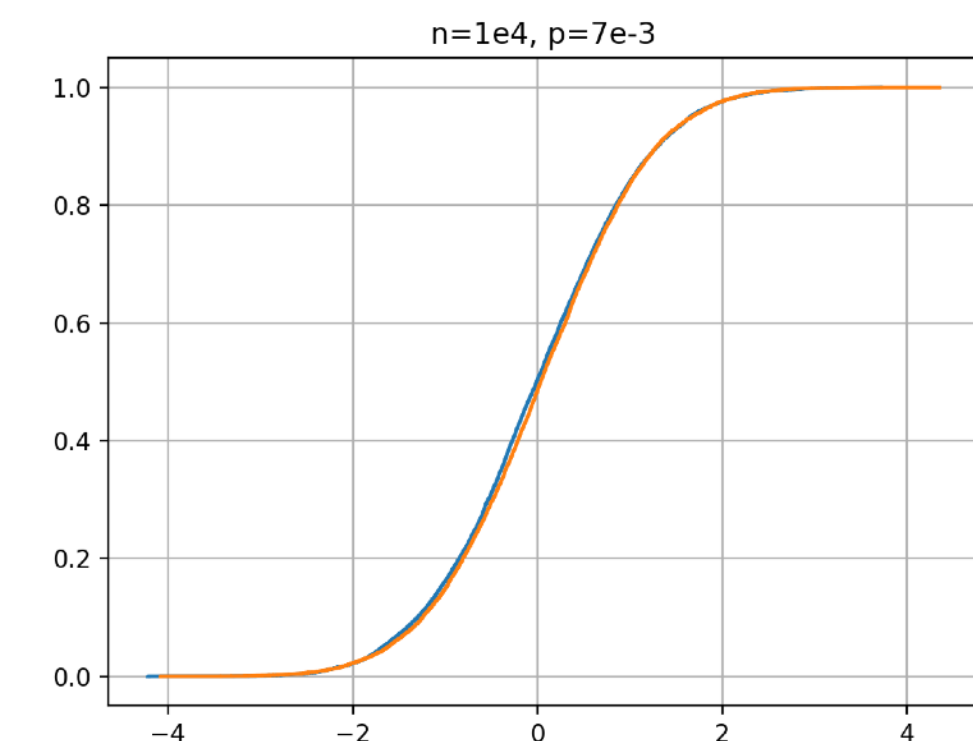
- In the best case: **at least ten** (two times five) and *two times nine is plenty*

minimum number of data to possibly get two-sided
 $p < 1\%$: **5+5** or 4+6 or 3+9 or 2+19 or 1+200
and $p < 5\%$: 4+4 or 3+5 or 2+8 or 1+40

- We often take two times **11 or 31 or 51**

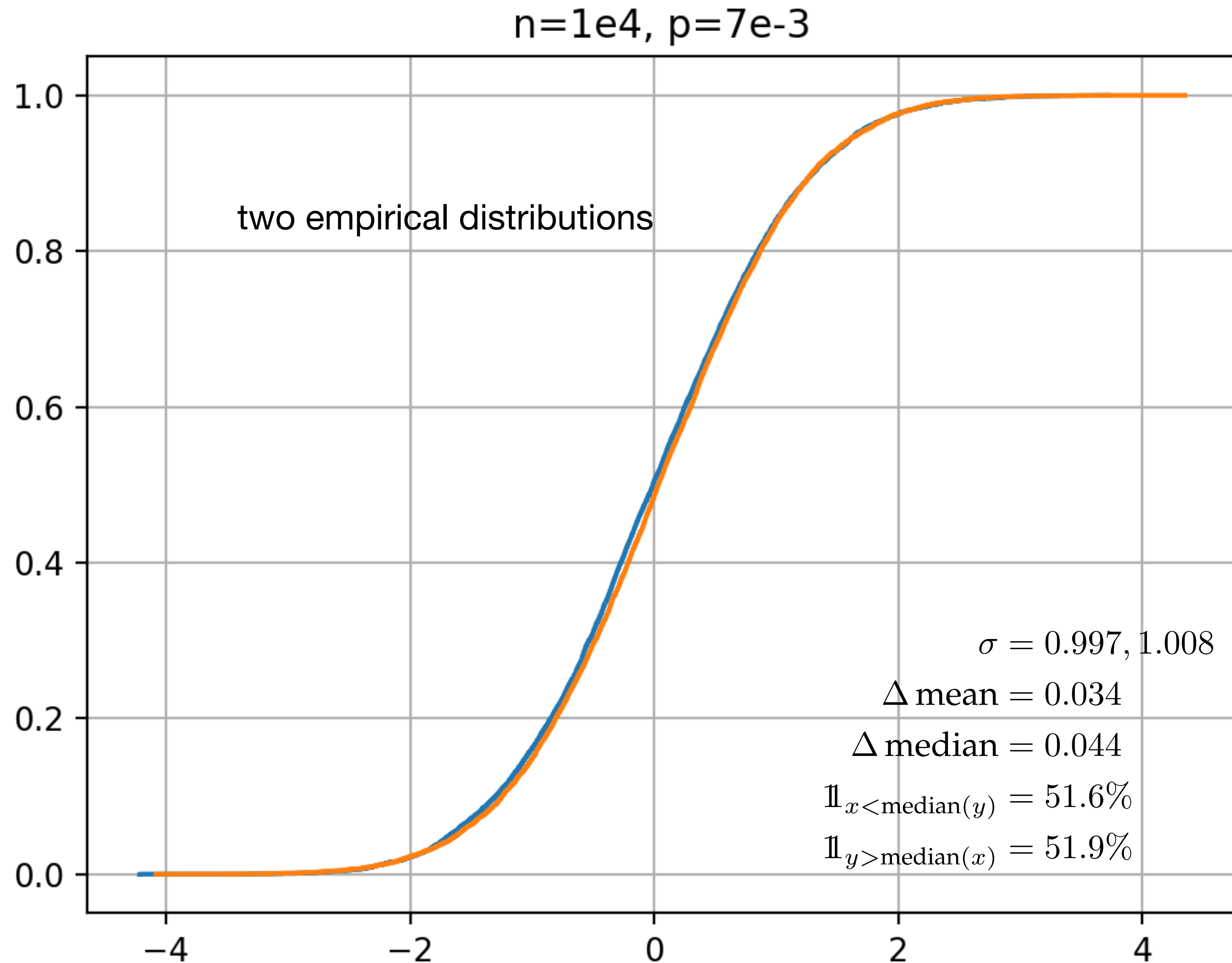
median, 5%-tile and 95%-tile are easily accessible
with 11 or 31 or 51... data

- Too many data make statistical significance meaningless



Statistical Testing: How many data do we need?

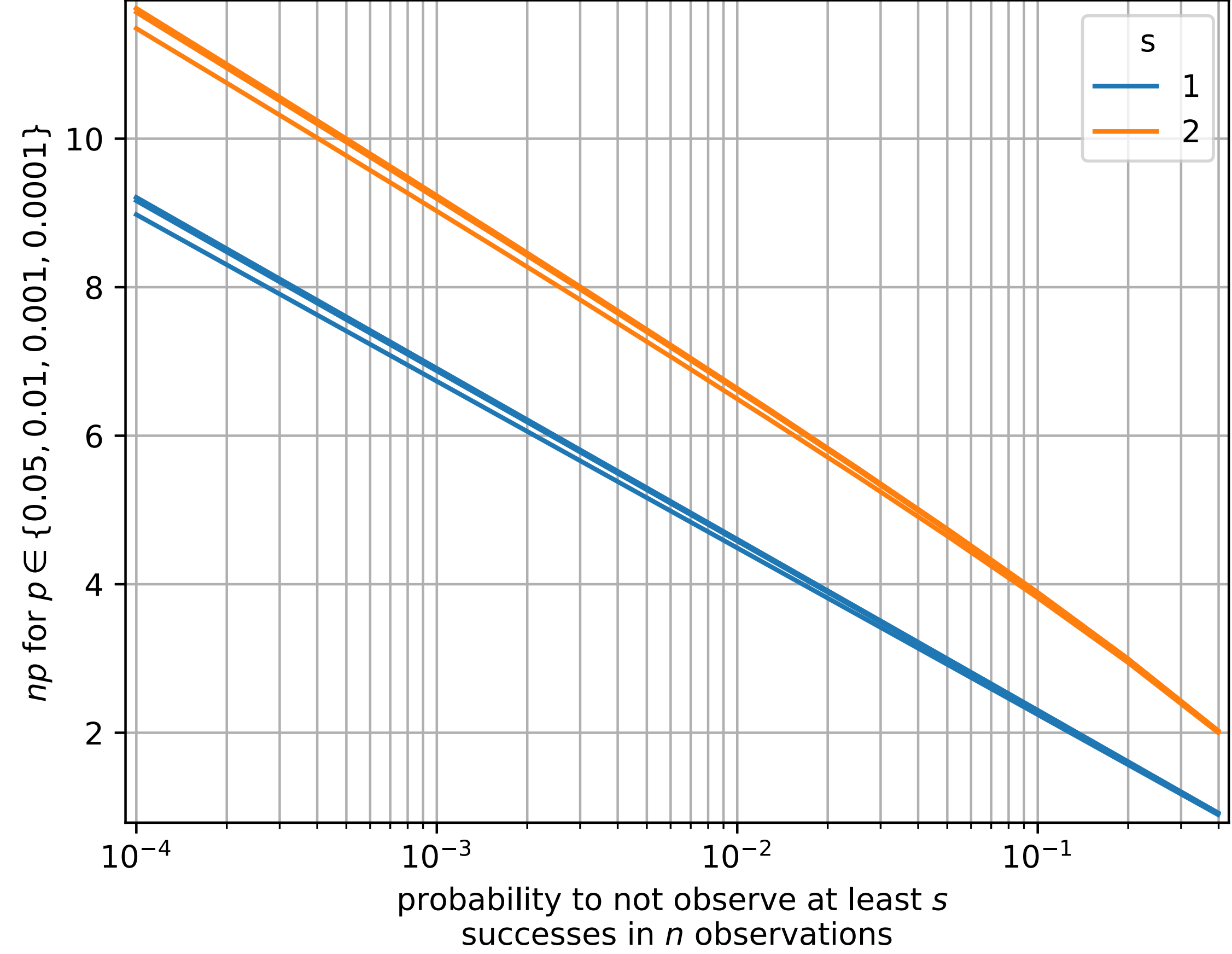
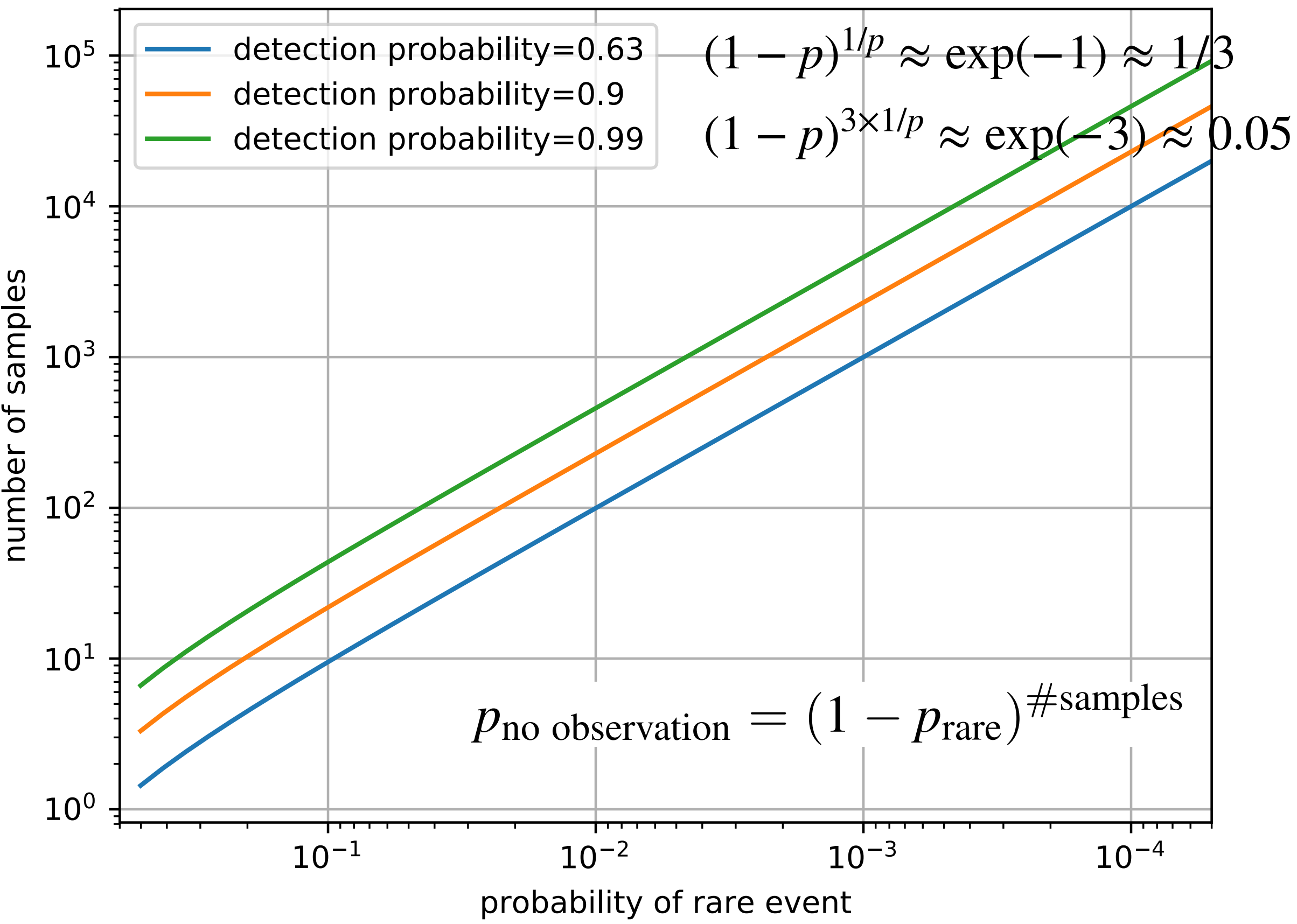
- In the |
is plen
- I often
- Too ma



led
:00
.40
ble
ata

Rare Events

The obvious: if we consider rare events to be important, we have to sample many data



Testing Frequencies

```
1 import scipy.stats
2 p1, n1 = 9, 90 # 10% success
3 p2, n2 = 20, 100 # 20% success
4 scipy.stats.chi2_contingency([[p1, n1 - p1],
5                               [p2, n2 - p2]])
```

(2.930076628352492 χ^2 -statistics

0.0869433657247775 p -value

1

[[13.73684211 76.26315789]

[15.26315789 84.73684211]])

Statistical Analysis

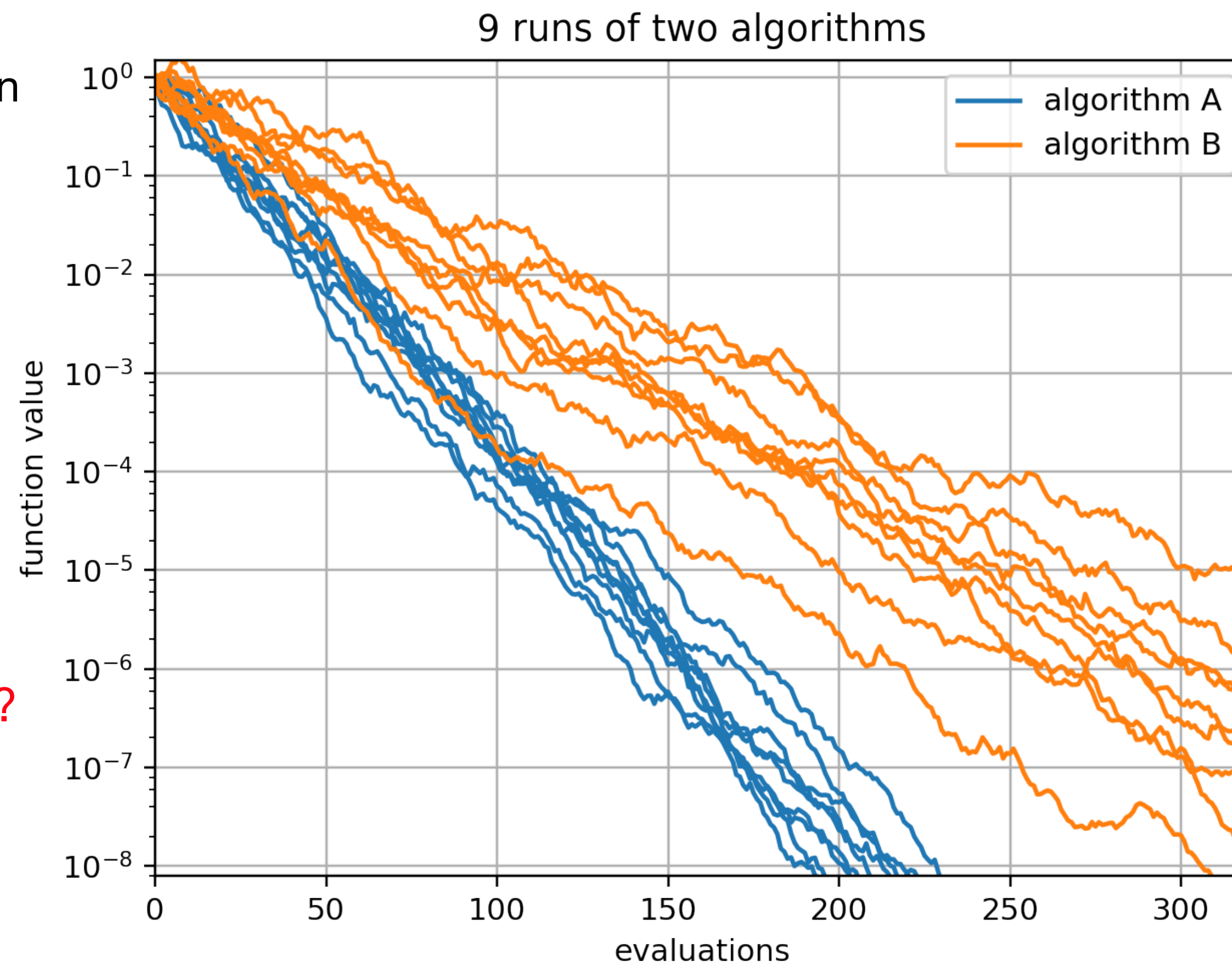
“[...] experimental results lacking proper statistical analysis must be considered anecdotal at best, or even wholly inaccurate.”

— M. Wineberg, 2016

Do you agree (sounds about right) with the quote or disagree (is taken a little over the top)?

an experimental result (shown are *all data obtained*):

Do we need a statistical analysis?



Using COCO

Running an experiment

```
$ ### get and install the code
$ git clone https://github.com/numbbo/coco.git # get coco using git
$ cd coco
$ python do.py run-python # install Python experimental module cocoex
$ python do.py install-postprocessing install-user # install postprocessing :-)
```

```
$ ### (optional) run an example from the shell
$ mkdir my-first-experiment
$ cd my-first-experiment
$ cp ../code-experiments/build/python/example_experiment2.py .
$ python example_experiment2.py # run the current "default" experiment
$ # and the post-processing
$ # and open browser when finished
```

```
#!/usr/bin/env python
"""Python script to benchmark fmin of scipy.optimize"""
from __future__ import division # not needed in Python 3
import cocoex, cocopp # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### input
suite_name = "bbob"
output_folder = "scipy-optimize-fmin"
```

```
#!/usr/bin/env python
"""Python script to benchmark fmin of scipy.optimize"""
from __future__ import division # not needed in Python 3
import cocoex, cocopp # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### input
suite_name = "bbob"
output_folder = "scipy-optimize-fmin"
fmin = scipy.optimize.fmin
budget_multiplier = 2 # increase to 10, 100, ...

### prepare
suite = cocoex.Suite(suite_name, "", "")
observer = cocoex.Observer(suite_name, "result_folder: " + output_folder)

### go
for problem in suite: # this loop will take several minutes or longer
    problem.observe_with(observer) # will generate the data for cocopp
    # restart until the problem is solved or the budget is exhausted
    while (not problem.final_target_hit and
           problem.evaluations < problem.dimension * budget_multiplier):
        fmin(problem, problem.initial_solution_proposal())
    # we assume that 'fmin' evaluates the final/returned solution
```

```
import cocoex, cocopp # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### input
suite_name = "bbob"
output_folder = "scipy-optimize-fmin"
fmin = scipy.optimize.fmin
budget_multiplier = 2 # increase to 10, 100, ...

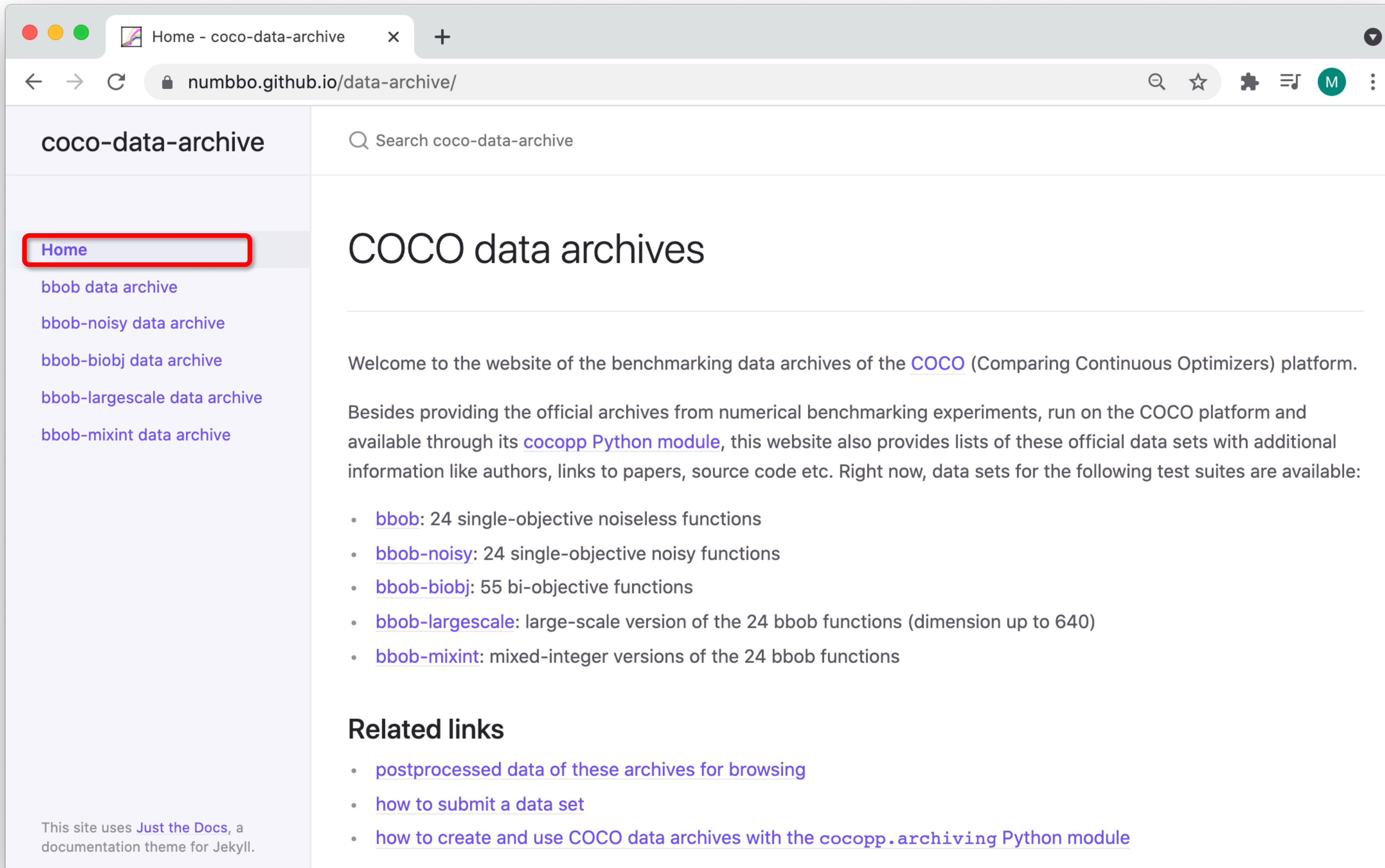
### prepare
suite = cocoex.Suite(suite_name, "", "")
observer = cocoex.Observer(suite_name, "result_folder: " + output_folder)

### go
for problem in suite: # this loop will take several minutes or longer
    problem.observe_with(observer) # will generate the data for cocopp
    # restart until the problem is solved or the budget is exhausted
    while (not problem.final_target_hit and
           problem.evaluations < problem.dimension * budget_multiplier):
        fmin(problem, problem.initial_solution_proposal())
        # we assume that 'fmin' evaluates the final/returned solution

### post-process data
cocopp.main(observer.result_folder) # re-run folders look like "...-001" etc
```

Selecting algorithms for comparison

Using COCO



The screenshot shows a web browser window with the URL `numbbo.github.io/data-archive/`. The page title is "coco-data-archive". The left sidebar contains a navigation menu with "Home" highlighted in a red box, and other links: "bbob data archive", "bbob-noisy data archive", "bbob-biobj data archive", "bbob-largescale data archive", and "bbob-mixint data archive". The main content area has a search bar and a heading "COCO data archives". Below the heading, there is a welcome message and a list of available test suites. At the bottom, there are "Related links" including "postprocessed data of these archives for browsing", "how to submit a data set", and "how to create and use COCO data archives with the cocopp.archiving Python module".

Home - coco-data-archive

numbbo.github.io/data-archive/

coco-data-archive

Search coco-data-archive

COCO data archives

Welcome to the website of the benchmarking data archives of the [COCO](#) (Comparing Continuous Optimizers) platform.

Besides providing the official archives from numerical benchmarking experiments, run on the COCO platform and available through its [cocopp Python module](#), this website also provides lists of these official data sets with additional information like authors, links to papers, source code etc. Right now, data sets for the following test suites are available:

- [bbob](#): 24 single-objective noiseless functions
- [bbob-noisy](#): 24 single-objective noisy functions
- [bbob-biobj](#): 55 bi-objective functions
- [bbob-largescale](#): large-scale version of the 24 bbob functions (dimension up to 640)
- [bbob-mixint](#): mixed-integer versions of the 24 bbob functions

Related links

- [postprocessed data of these archives for browsing](#)
- [how to submit a data set](#)
- [how to create and use COCO data archives with the cocopp.archiving Python module](#)

This site uses [Just the Docs](#), a documentation theme for Jekyll.

Using COCO

Home

bbob data archive

bbob-noisy data archive

bbob-biobj data archive

bbob-largescale data archive

bbob-mixint data archive

Search coco-data-archive

Algorithm data sets for the bbob test suite

In the first table below, you will find all official algorithm data sets on the bbob test suite, together with their year of publication, the authors, and related PDFs for each data set. Links to the source code to run the corresponding experiments/algorithms are provided whenever available.

A second table mentions data sets that have been collected on the bbob suite, but which are not complete in the sense that they miss at least one of the requested dimensions 2, 3, 5, 10, 20.

To sort the tables, simply click on the table header of the corresponding column.

Number	Algorithm Name	Year	Author(s)	link to data	related PDFs, source code, etc.
000	ALPS	2009	Hornby	data	pdf
001	AMALGAM	2009	Bosman et al.	data	pdf noiseless - pdf noisy
002	BAYEDA	2009	Gallagher	data	pdf noiseless - pdf noisy
003	BFGS	2009	Ros	data	pdf noiseless - pdf noisy
004	BIPOP-CMA-ES	2009	Hansen	data	pdf noiseless - pfd noisy

This site uses [Just the Docs](#), a documentation theme for Jekyll.

Visit <https://numbbo.github.io/data-archive/>

https://numbbo.github.io/ppdata x +

numbbo.github.io/ppdata-archive/

COCO ppdata-archive

This archive contains *postprocessed* data displaying benchmarking experiments of various numerical optimization algorithms on the various test suites provided by the [Comparing Continuous Optimizers platform](#). Experiments are conducted in a blackbox setting and data are collected by year.

bbob	bbob-noisy	bbob-biobj	bbob-largescale	bbob-mixint
24 functions single-objective continous domain 200+ algorithm data sets	30 functions noisy evaluations single-objective 45 algorithm data sets	55 functions bi-objective noiseless 32 algorithm data sets	24 bbob functions single-objective dimensions 20 to 640 11 algorithm data sets	24 functions 80% discrete variables single-objective no official data yet
2009 2010 2012 2013 2014 2015-CEC 2015-GECCO 2016 2017 2018 2019	2009 2010 2012 2016	2016 2017 2019	2019	

Visit <https://numbbo.github.io/ppdata-archive/>