



**HAL**  
open science

# A Practical Guide to Experimentation (and Benchmarking)

Nikolaus Hansen

► **To cite this version:**

Nikolaus Hansen. A Practical Guide to Experimentation (and Benchmarking). GECCO '18 Companion: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Jul 2018, Kyoto, Japan. hal-01959453

**HAL Id: hal-01959453**

**<https://inria.hal.science/hal-01959453v1>**

Submitted on 18 Dec 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A Practical Guide to Experimentation (and Benchmarking)

Nikolaus Hansen  
Inria

Research Centre Saclay, CMAP, Ecole polytechnique, Université Paris-Saclay

- Installing IPython is *not* a prerequisite to follow the tutorial
- for downloading the material, see  
[slides](http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco2018-experimentation-guide-slides.pdf): <http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco2018-experimentation-guide-slides.pdf> at <http://www.cmap.polytechnique.fr/~nikolaus.hansen/invitedtalks.html>  
[code](https://github.com/nikohansen/GECCO-2018-experimentation-guide-notebooks): <https://github.com/nikohansen/GECCO-2018-experimentation-guide-notebooks>
-

# Overview

- Scientific experimentation
- Invariance
- Statistical Analysis
- A practical experimentation session
- Approaching an unknown problem
- Performance Assessment
  - What to measure
  - How to display
  - Aggregation
  - Empirical distributions

Do not hesitate to ask questions!

# Why Experimentation?

- The behaviour of many if not most **interesting algorithms** is
  - not **amenable** to a (full) theoretical analysis even when applied to simple problems
    - calling for an alternative to theory for investigation
  - not fully **comprehensible** or even predictable without (extensive) empirical examinations
    - even on simple problems
    - 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
  - 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



# Scientific Experimentation (dos and don'ts)

- What is the aim? *Answer a question*, ideally quickly and comprehensively

consider in advance what the question is and in which way the experiment can answer the question

What are the dos and don'ts?

- what is most helpful to do?
- what is better to avoid?

# Scientific Experimentation (dos and don'ts)

- What is the aim? *Answer a question*, ideally quickly (minutes, seconds) and comprehensively
  - consider in advance what the question is and in which way the experiment can answer the question
- do not (blindly) trust in what one needs to rely upon (code, claims, ...) without *good* reasons
  - check/test “everything” yourself, practice stress testing (e.g. weird parameter setting) which also boosts understanding one key element for success
  - interpreted/scripted languages have an advantage
  - Why Most Published Research Findings Are False* [Ioannidis 2005]
- practice to **make predictions** of the (possible) outcome(s)
  - to develop a mental model of the object of interest
  - to practice being proven wrong, to overcome *confirmation bias*
- run *rather many than few 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 feeling for the effect of setup changes*

# Scientific Experimentation (dos and don'ts)

- run *rather many than few 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 feeling for the effect of setup changes*
- run any experiment at least *twice*
  - assuming that the outcome is stochastic  
get an *estimator of variation/dispersion/variance*
- *display*: *the more the better, the better the better*
  - figures are *intuition pumps* (not only for presentation or publication)  
it is hardly possible to overestimate the value of a good figure  
data is the only way experimentation can help to answer questions,  
therefore *look 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

# Scientific Experimentation (dos and don'ts)

- don't make **minimising CPU-time** a primary objective
  - avoid spending time in implementation details to tweak performance
  - 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”*
- 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 changes
- there are many **devils in the details**, results or their interpretation may crucially depend on simple or intricate bugs or subtleties
  - yet another reason to run many (slightly) different experiments
  - check limit settings to give consistent results

# Scientific Experimentation (dos and don'ts)

- there are many **devils in the details**, results or their interpretation may crucially depend on simple or intricate bugs or subtleties
  - yet another reason to run many (slightly) different experiments
  - check limit settings to give consistent results
- **Invariance** is a very powerful, almost indispensable tool

# Invariance: binary variables

Assigning 0/1 (for example minimize  $\sum_i x_i$  vs  $\sum_i 1 - x_i$ )

- is an “arbitrary” and “trivial” encoding choice and
- amounts to the affine linear transformation  $x_i \mapsto -x_i + 1$   
this transformation or the identity are the **coding choice in each variable**  
in continuous domain: norm-preserving (isotropic, “rigid”) transformation
- does not change the function “structure”
  - all level sets  $\{x \mid f(x) = \text{const}\}$  have the same size (number of elements, same volume)
  - the same neighbourhood
  - no variable dependencies are introduced (or removed)

Instead of 1 function, we now consider  **$2^{**}n$  different but equivalent functions**

$2^{**}n$  is non-trivial, it is the size of the search space itself

# Invariance: binary variables

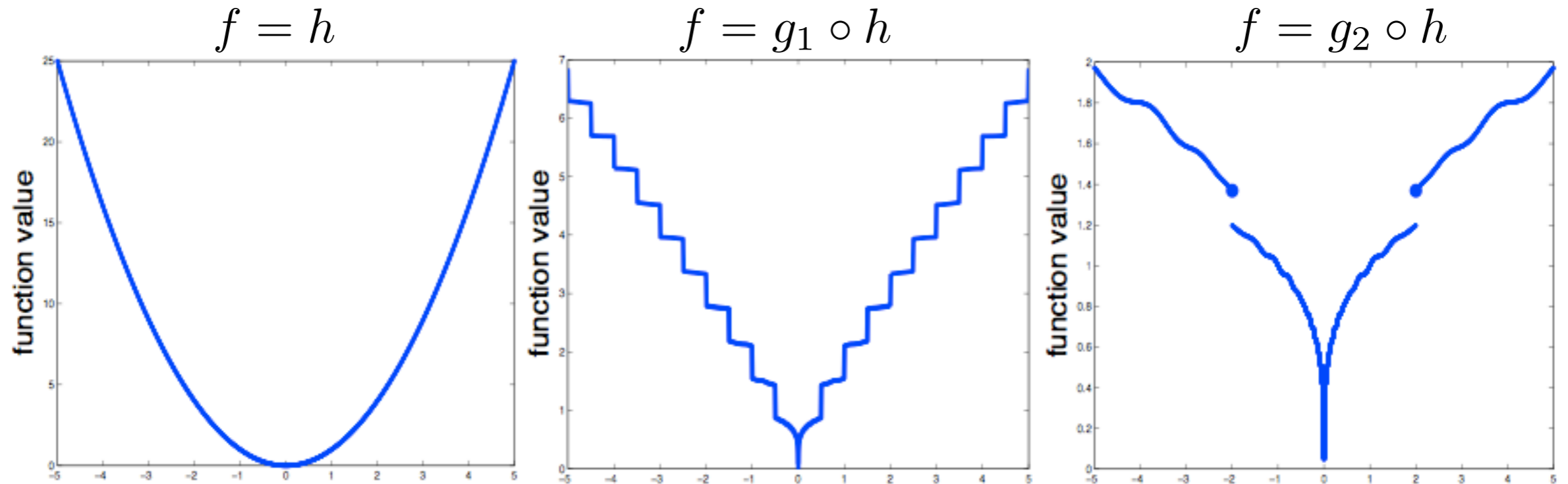
## Permutation of variables

- is another “arbitrary” and “trivial” encoding choice and
- is another norm-preserving transformation
- does not change the function “structure” (as above)
- consider one-point vs two-point crossover: which is better choice?  
only two-point crossover is invariant to variable permutation

Instead of 1 function, we now consider  $n!$  different but equivalent functions

$n! \gg 2^n$  is much larger than the size of the search space

# Invariance Under Order Preserving Transformations



Three functions belonging to the same equivalence class

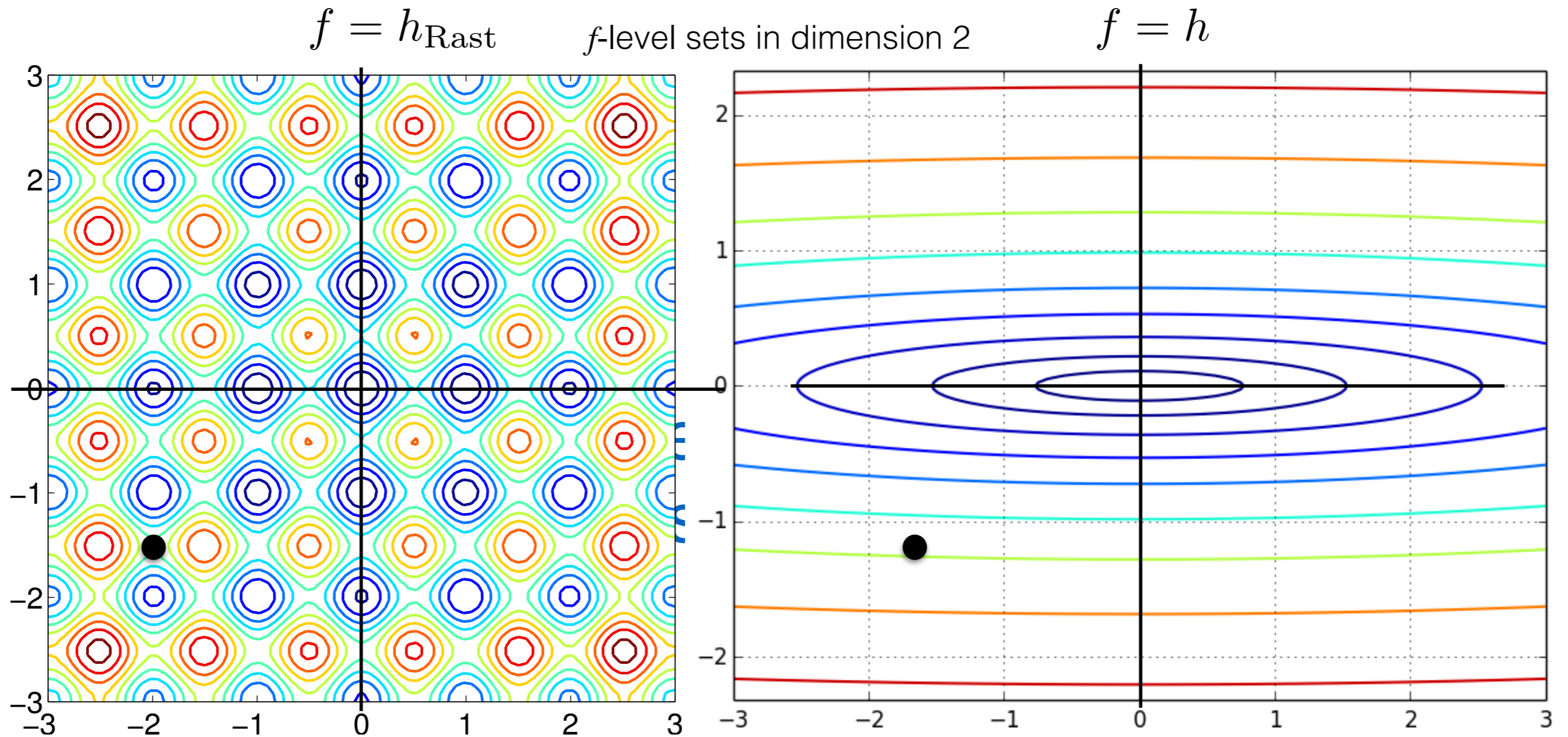
A *function-value free search algorithm* is invariant under the transformation with any **order preserving** (strictly increasing)  $g$ .

Invariances make

- observations meaningful                      as a rigorous notion of generalization
- algorithms predictable and/or "robust"



# Invariance Under Rigid Search Space Transformations

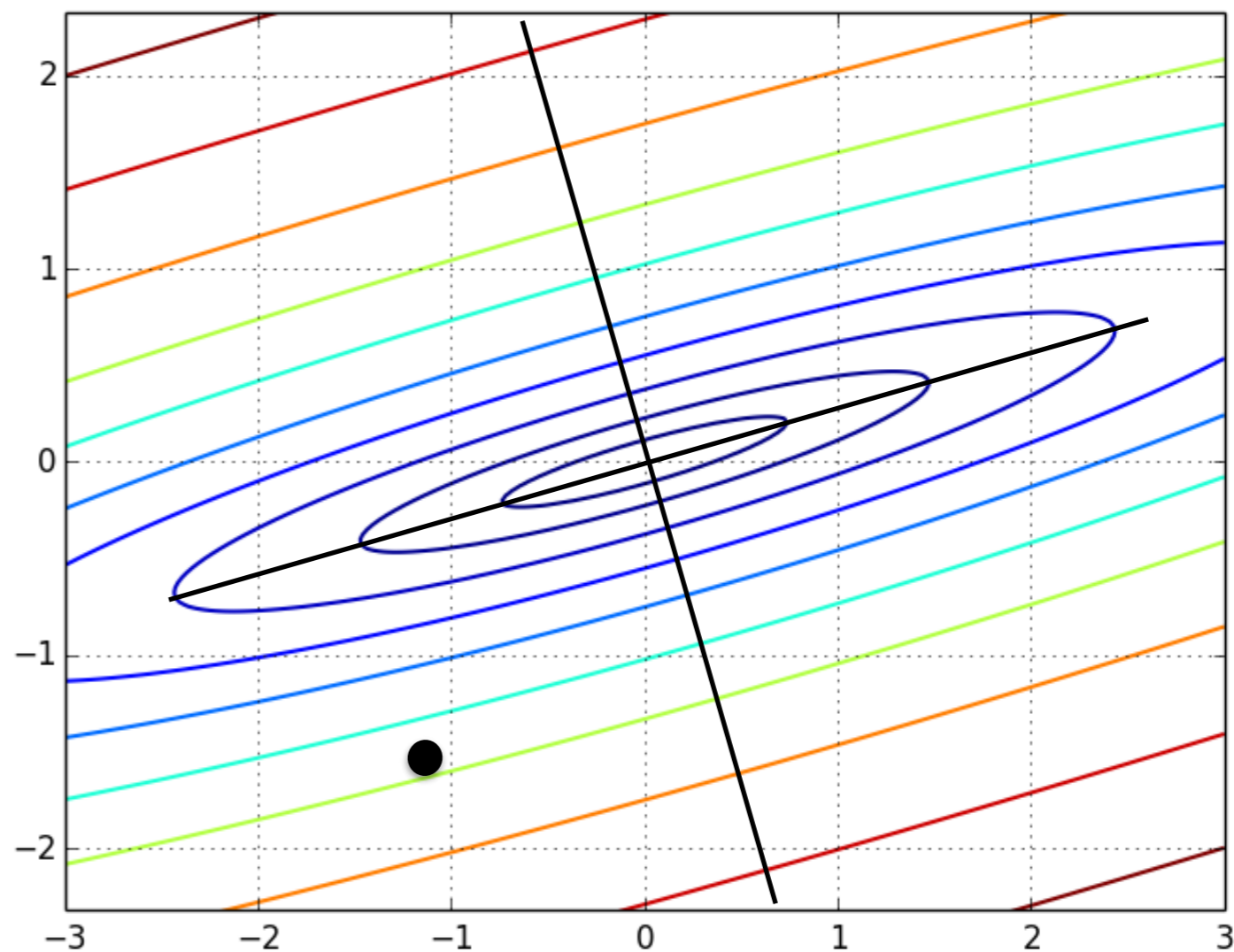
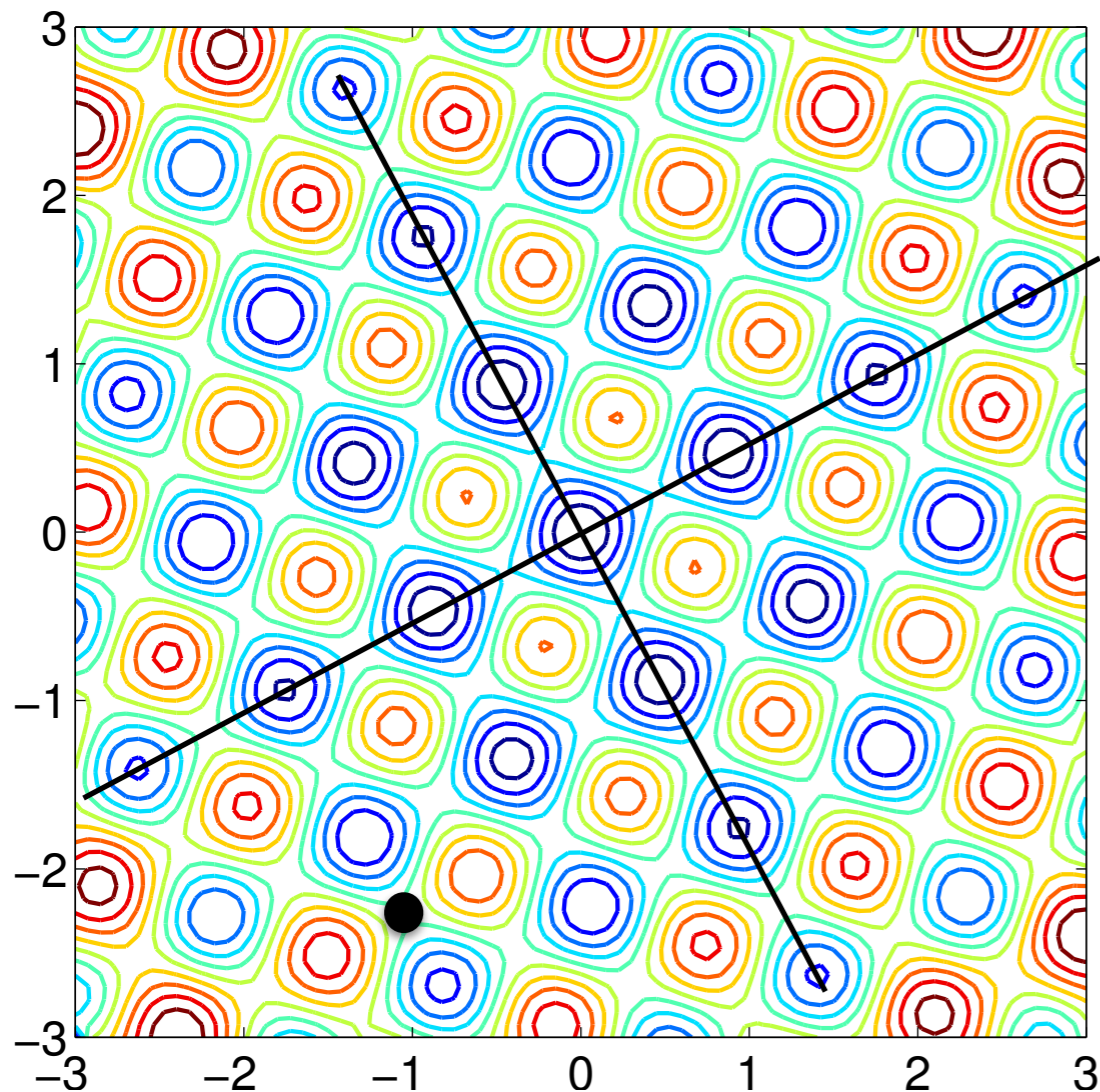


for example, invariance under search space rotation  
(**separable** vs non-separable)

# Invariance Under Rigid Search Space Transformations

$$f = h_{\text{Rast}} \circ R \quad f\text{-level sets in dimension 2}$$

$$f = h \circ R$$



for example, invariance under search space rotation  
(separable vs **non-separable**)

# Invariance

*The grand aim of all science is to cover the greatest number of empirical facts by logical deduction from the smallest number of hypotheses or axioms.*

— Albert Einstein

- Empirical performance results
  - ▶ from benchmark functions
  - ▶ from solved real world problems

are only useful if they do **generalize** to other problems

- **Invariance** is a strong **non-empirical** statement about generalization
  - generalizing (identical) performance from a single function to a whole class of functions

Consequently, invariance is of greatest importance for the **assessment of search algorithms.**

# 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

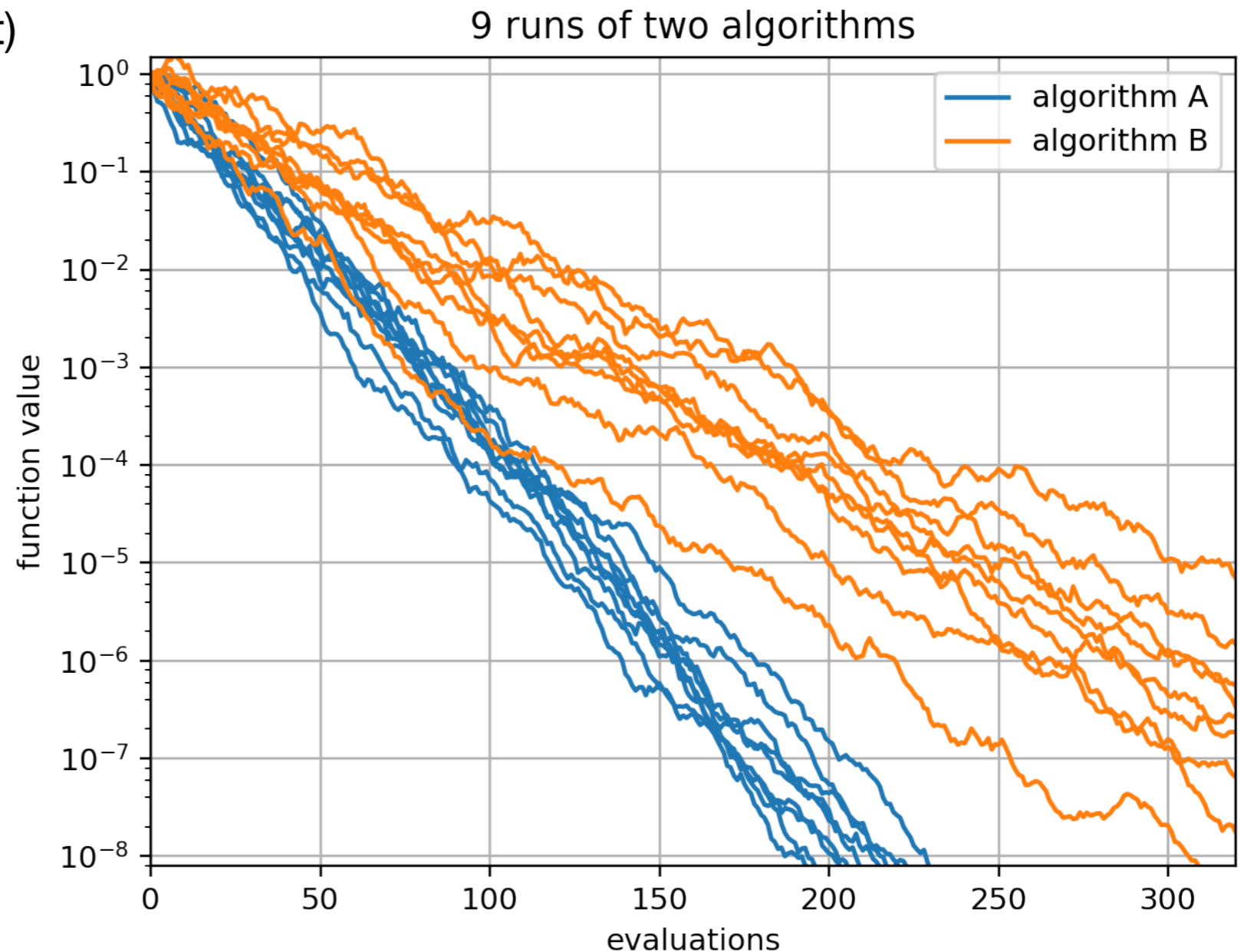
*“experimental results lacking proper statistical analysis must be considered anecdotal at best, or even wholly inaccurate”*

*— M. Wineberg*

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

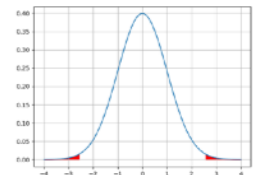
an experimental result (shown are *all data obtained*):

Do we (even) need a statistical analysis?



# Statistical Significance: 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 *important* or *meaningful*
- prefer “nonparametric” methods
  - not* assuming that the data come from a *parametrised* family of probability distributions
- **p-value** = significance level = probability of a false positive outcome, given H0 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

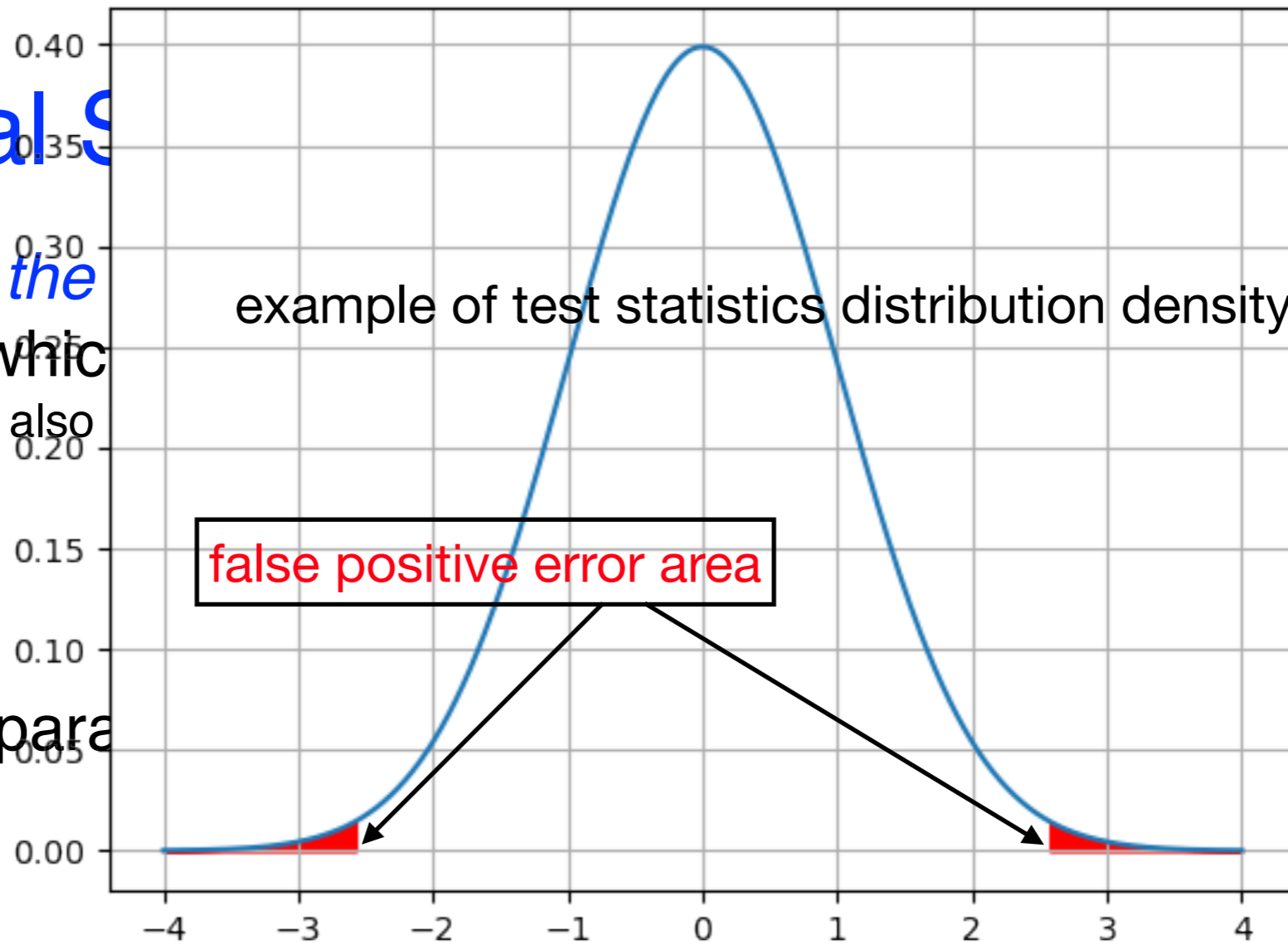


# Statistical

ecure

- *first, check the* difference which

this also



given H0

mbinations)  
*statistically*  
simple trick,  
*meaningful*

- prefer “nonpara

parametrised  
distributions

- **p-value** = significance level = probability of a false positive outcome, given H0 is true

smaller p-values are better

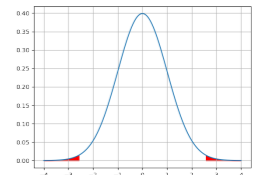
<0.1% or <1% or <5% is usually considered as *statistically significant*

- *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 Significance: 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 *important* or *meaningful*
- prefer “nonparametric” methods
  - not* assuming that the data come from a *parametrised* family of probability distributions
- **p-value** = significance level = probability of a false positive outcome, given H0 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 Significance: Methods

- use the **rank-sum** test (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
  - **Null hypothesis** (nothing relevant is observed if):  $\Pr(x < y) = \Pr(y < x)$ 

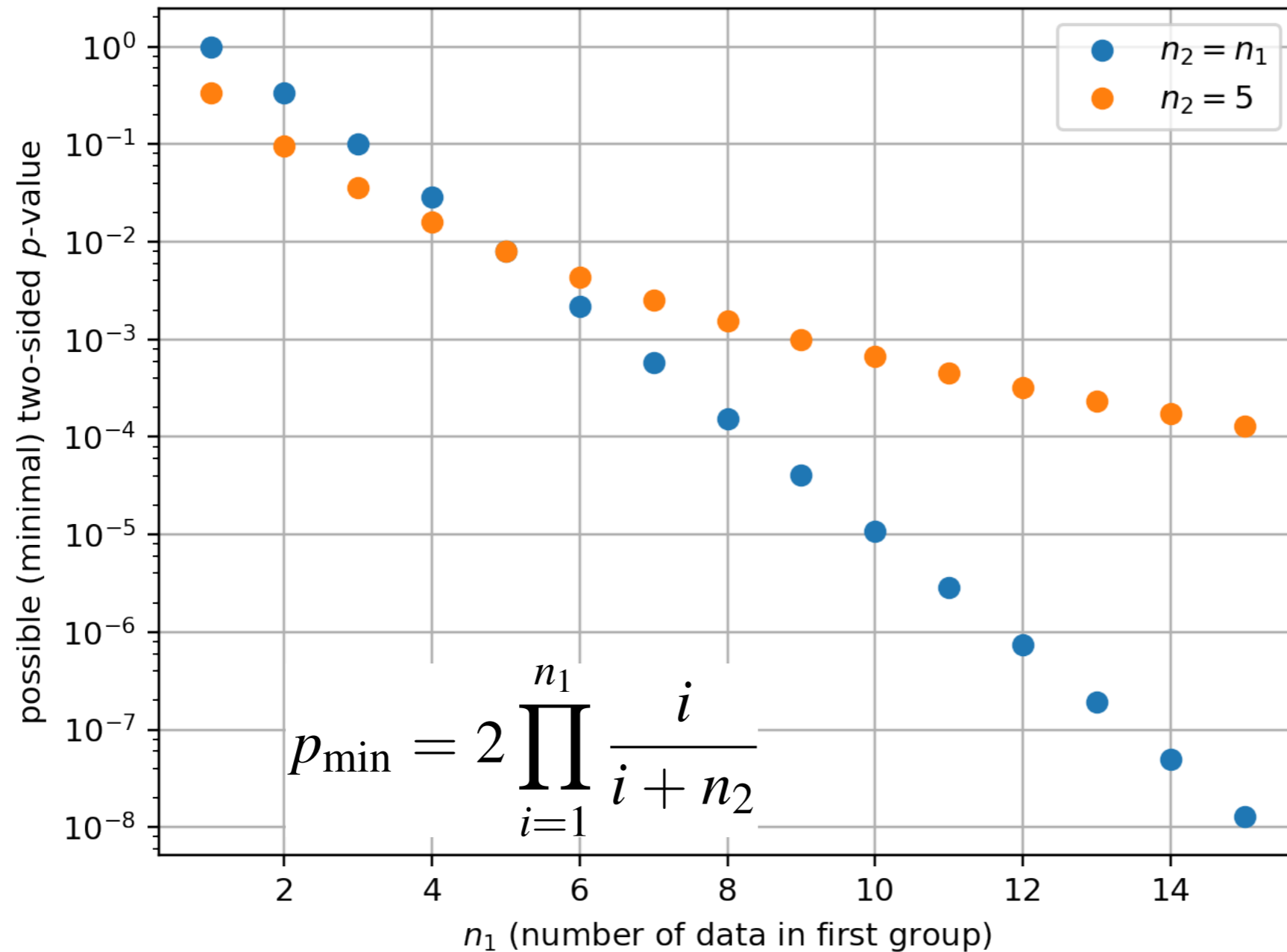
H0: the probability to be greater or smaller (better or worse) is the same the aim is to be able to reject the null hypothesis
  - Procedure: compute the sum of ranks in the ranking of all (combined) data values
  - **Outcome**: a *p*-value

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 Significance: How many data do we need?

AKA as test efficiency

Rank Sum Test



- 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

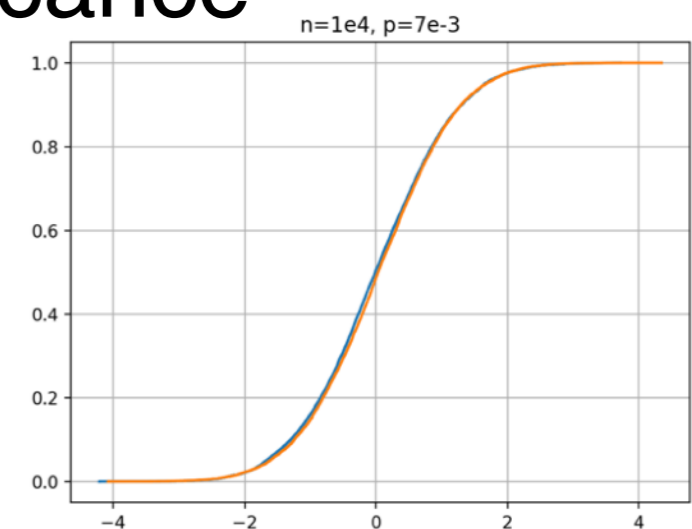
- use the [Bonferroni correction](#) for multiple tests

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

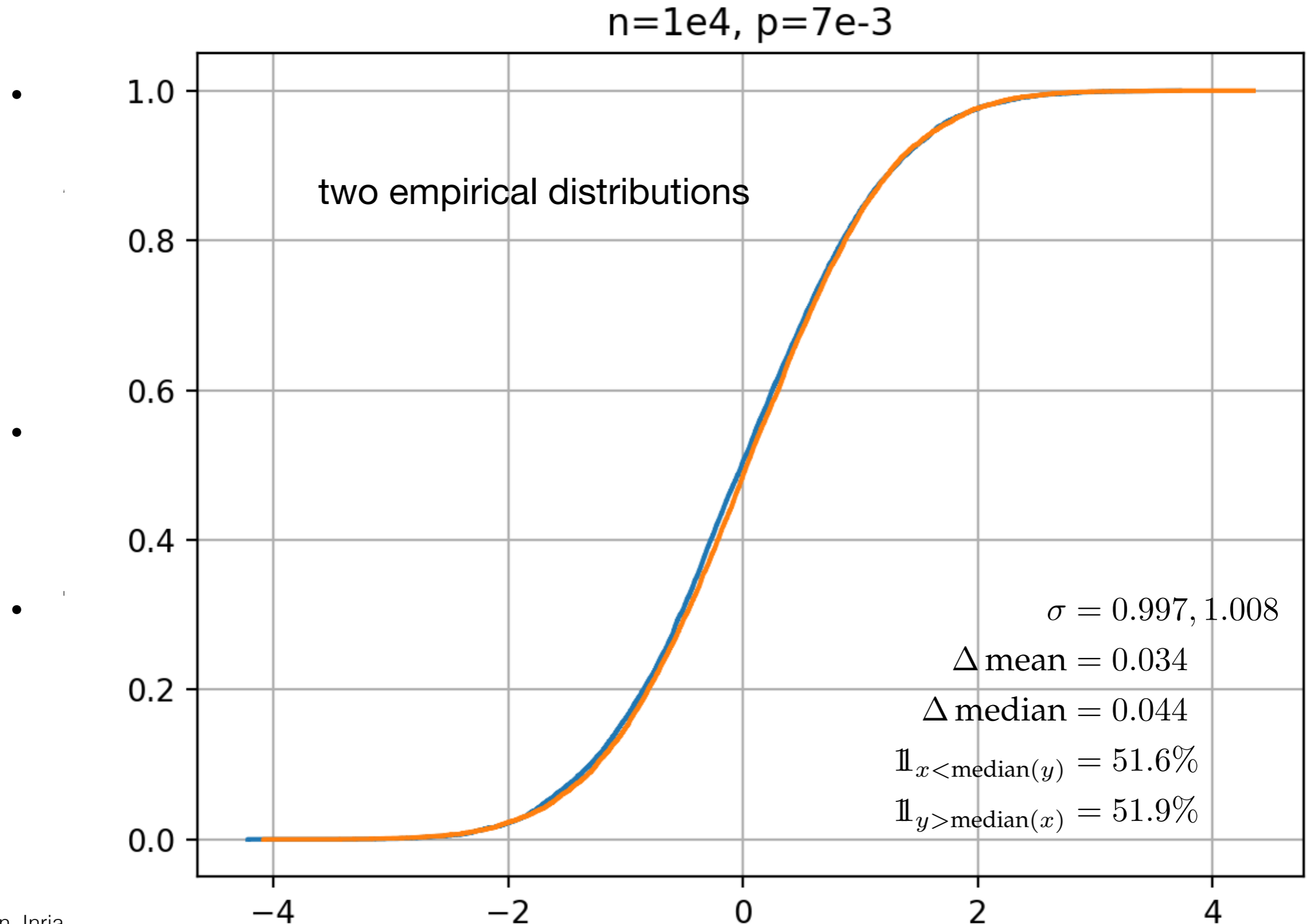
A practical guide to experimentation

# Statistical Significance: 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
- I 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 Significance: How many data do we need?



# Statistical Analysis

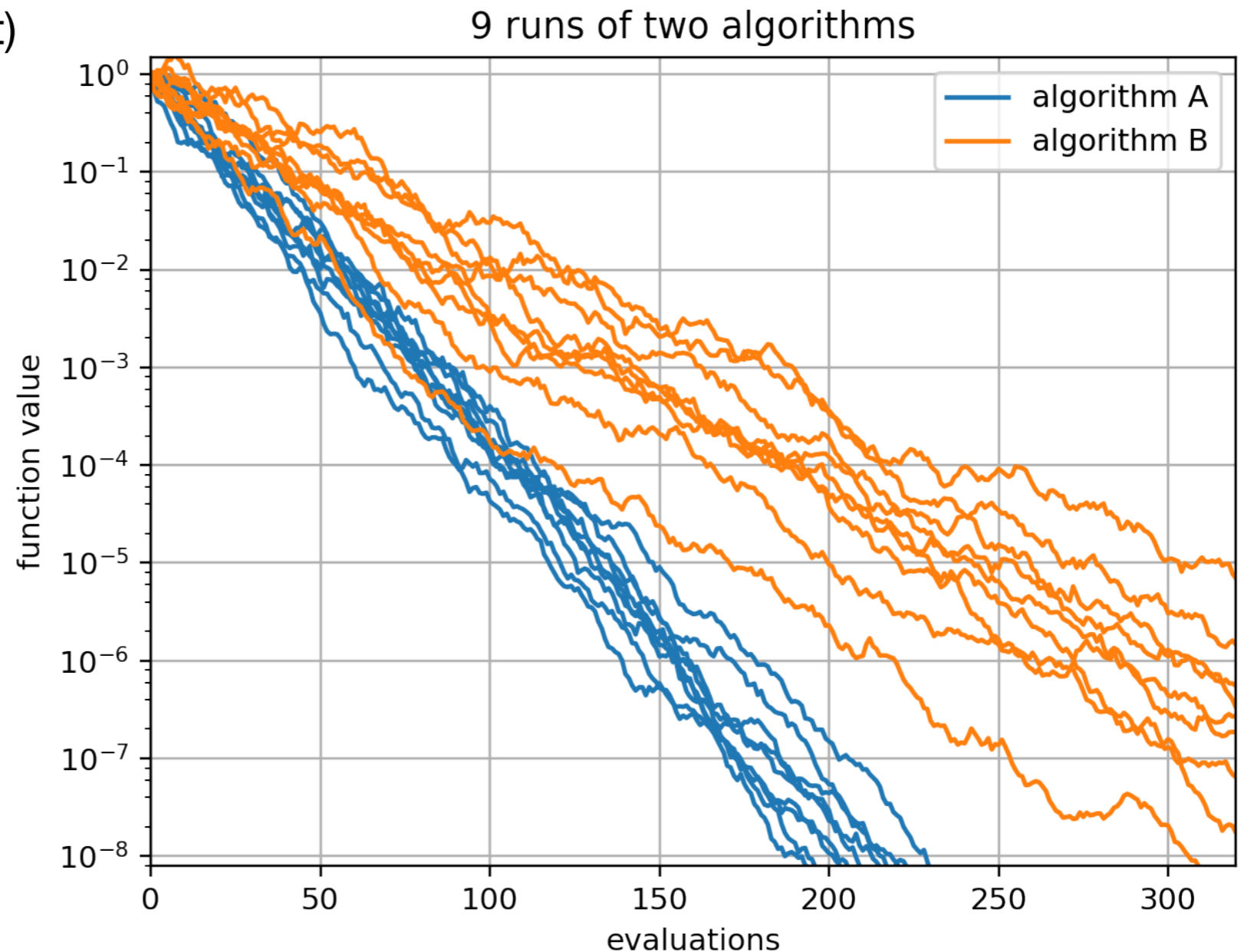
*“experimental results lacking proper statistical analysis must be considered anecdotal at best, or even wholly inaccurate”*

*— M. Wineberg*

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

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

Do we (even) need a statistical analysis?



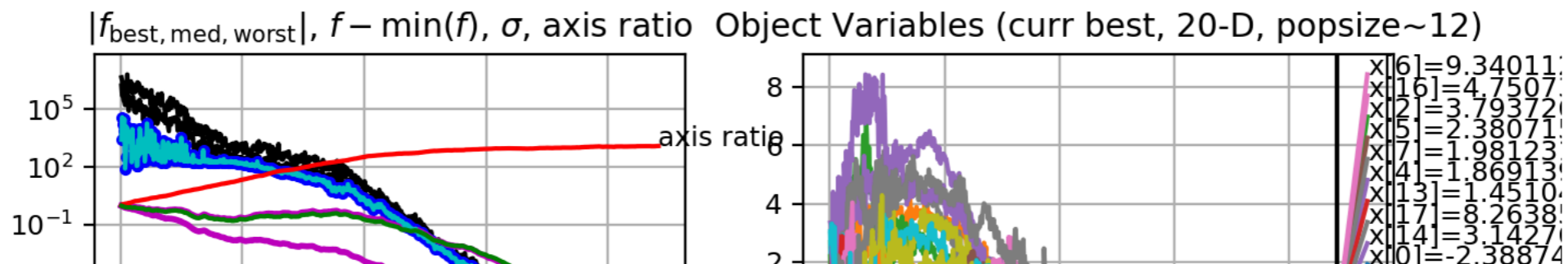
# Jupyter IPython notebook

```
%pylab nbagg
import cma
cma.fmin(cma.ff.tablet, 20 * [1], 1);
```

```
Populating the interactive namespace from numpy and matplotlib
(6_w,12)-aCMA-ES (mu_w=3.7,w_l=40%) in dimension 20 (seed=344737, Wed Jul 5 16:09:44 2017)
Iterat #Fevals  function value  axis ratio  sigma  min&max  std  t[m:s]
  1      12  2.637846492377813e+03  1.0e+00  9.49e-01  9e-01  1e+00  0:00.0
  2      24  3.858353384747645e+04  1.1e+00  9.13e-01  9e-01  9e-01  0:00.0
  3      36  1.589934793439056e+04  1.2e+00  8.94e-01  9e-01  9e-01  0:00.0
100     1200  1.805167565570186e+02  6.6e+00  2.52e-01  6e-02  3e-01  0:00.1
200     2400  9.260486860109009e+01  4.2e+01  2.79e-01  1e-02  4e-01  0:00.3
300     3600  8.460045942108286e+00  2.0e+02  3.20e-01  4e-03  4e-01  0:00.4
400     4800  5.352841113616880e-02  5.2e+02  4.71e-02  2e-04  5e-02  0:00.5
500     6000  1.169838413517761e-04  8.7e+02  2.61e-03  3e-06  2e-03  0:00.7
600     7200  2.232682824828931e-08  9.9e+02  5.00e-05  4e-08  3e-05  0:00.8
700     8400  1.483610308401096e-12  1.2e+03  4.61e-07  3e-10  2e-07  0:00.9
736     8832  2.696542797455203e-14  1.2e+03  1.03e-07  5e-11  5e-08  0:01.0
termination on tolfun=1e-11 (Wed Jul 5 16:09:46 2017)
final/bestever f-value = 1.422957e-14 1.422957e-14
incumbent solution: [ -1.01044748e-11 -3.22608195e-08 -8.75163241e-10 -3.66834969e-08
 2.35485309e-08 -9.59521093e-10  4.23137381e-08  6.92049899e-09 ...]
std deviations: [ 5.07976963e-11  4.52415829e-08  4.67529085e-08  4.36659472e-08
 4.04686177e-08  4.38294341e-08  4.65665203e-08  5.01580767e-08 ...]
```

```
cma.plot()
```

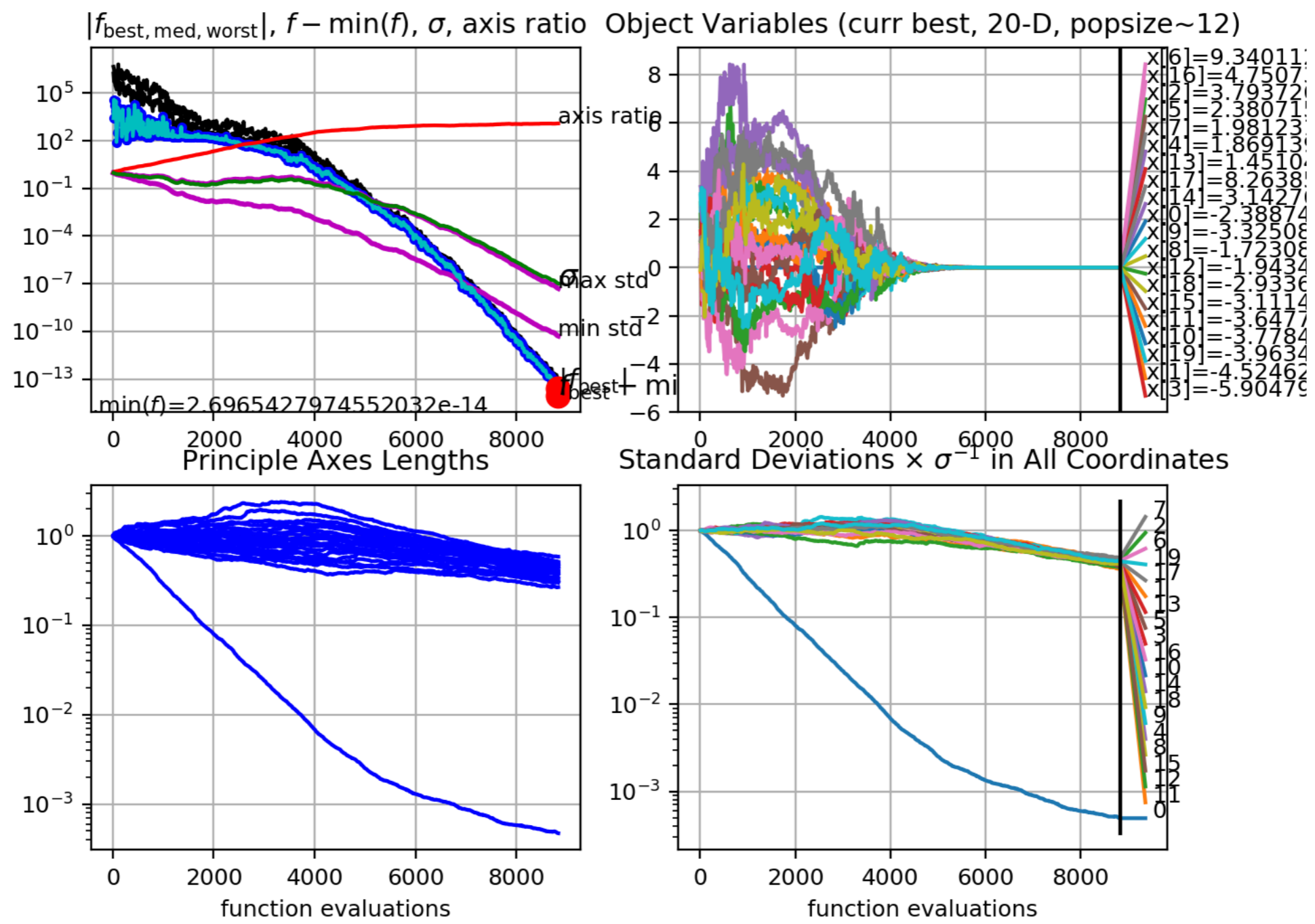
Figure 328





```
cma.plot()
```

Figure 328 ⏻



Questions?



# Jupyter IPython notebook

```
▼ # download&install anaconda python
# shell cmd "conda create" in case a different Python version is needed
# shell cmd "pip install cma" to install a CMA-ES module (or see github)
# shell cmd "jupyter-notebook" and click on compact-ga.ipynb
from __future__ import division, print_function
%pylab nbagg
```

Populating the interactive namespace from numpy and matplotlib

- **SEE** <https://github.com/nikohansen/GECCO-2018-experimentation-guide-notebooks>
- **Demonstrations**
  - A somewhat typical working mode
  - A parameter investigation

# Approaching an unknown problem

- Problem/variable **encoding**  
for example log scale vs linear scale vs quadratic transformation
- Fitness **formulation**  
for example  $\sum_i |x_i|$  and  $\sum_i x_i^2$  have the same optimal (minimal) solution but may be very differently “optimizable”.
- Try to **locally improve** a given (good) solution
- Start local search from **different initial solutions**.  
Ending up always in different solutions? Or always in the same?
- Apply “global search” setting
- **see also** [http://cma.gforge.inria.fr/cmaes\\_sourcecode\\_page.html#practical](http://cma.gforge.inria.fr/cmaes_sourcecode_page.html#practical)

Questions?

# Performance Assessment

- methodology: run an algorithm on a **set of test functions** and extract **performance measures** from the generated data
  - choice of measure and aggregation
- display
  - subtle display changes can make a huge difference
- there are surprisingly many devils in the details

# Why do we want to measure performance?

- compare algorithms and algorithm selection (the obvious)
  - ideally we want standardized comparisons
- regression testing after (small) changes
  - as we may expect (small) changes in behaviour, conventional regression testing may not work
- understanding of algorithms
  - to improve algorithms
  - non-standard experimentation is often preferable or necessary

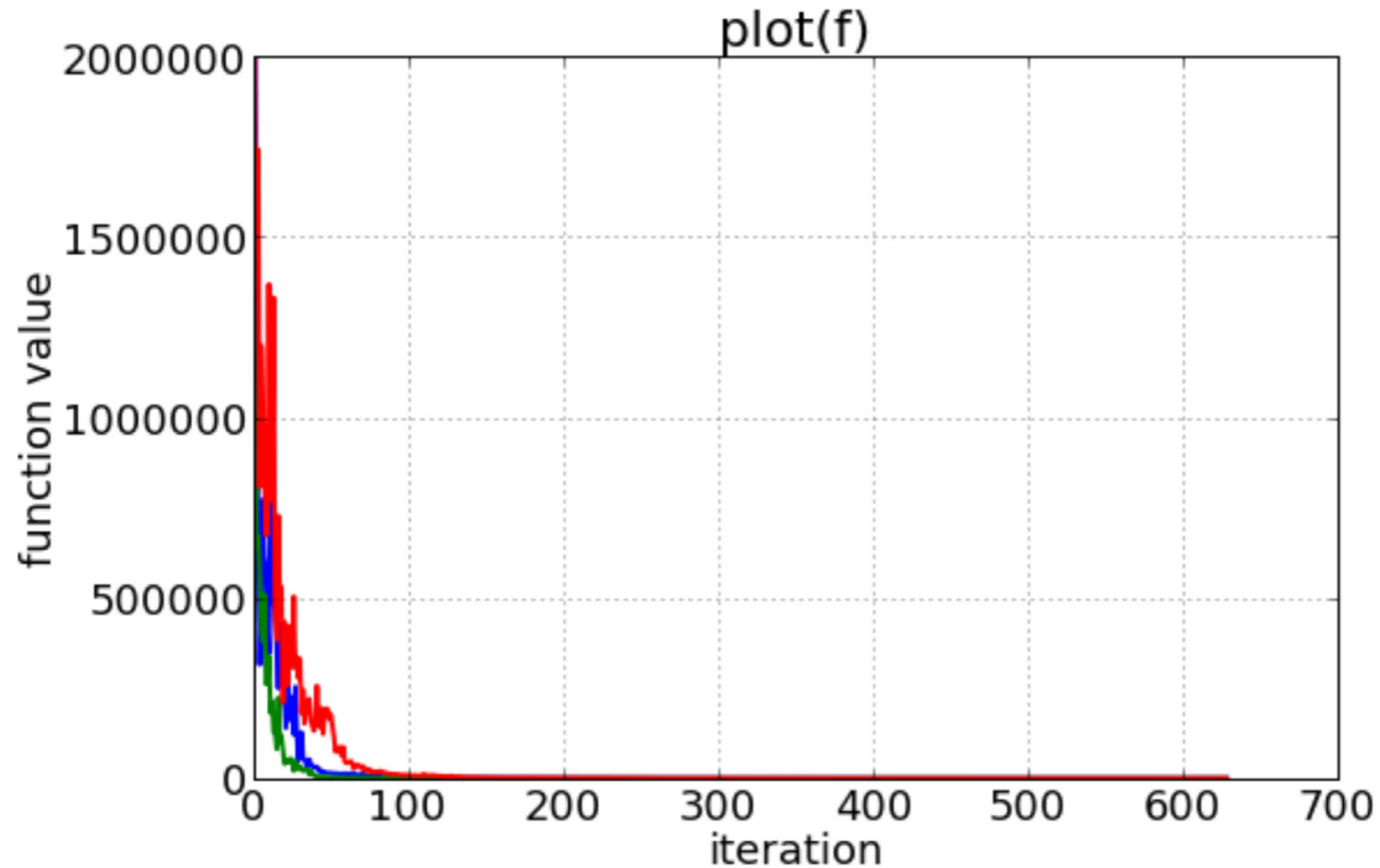
# Measuring Performance

Empirically

convergence graphs is all we have to start with

the right presentation is important!

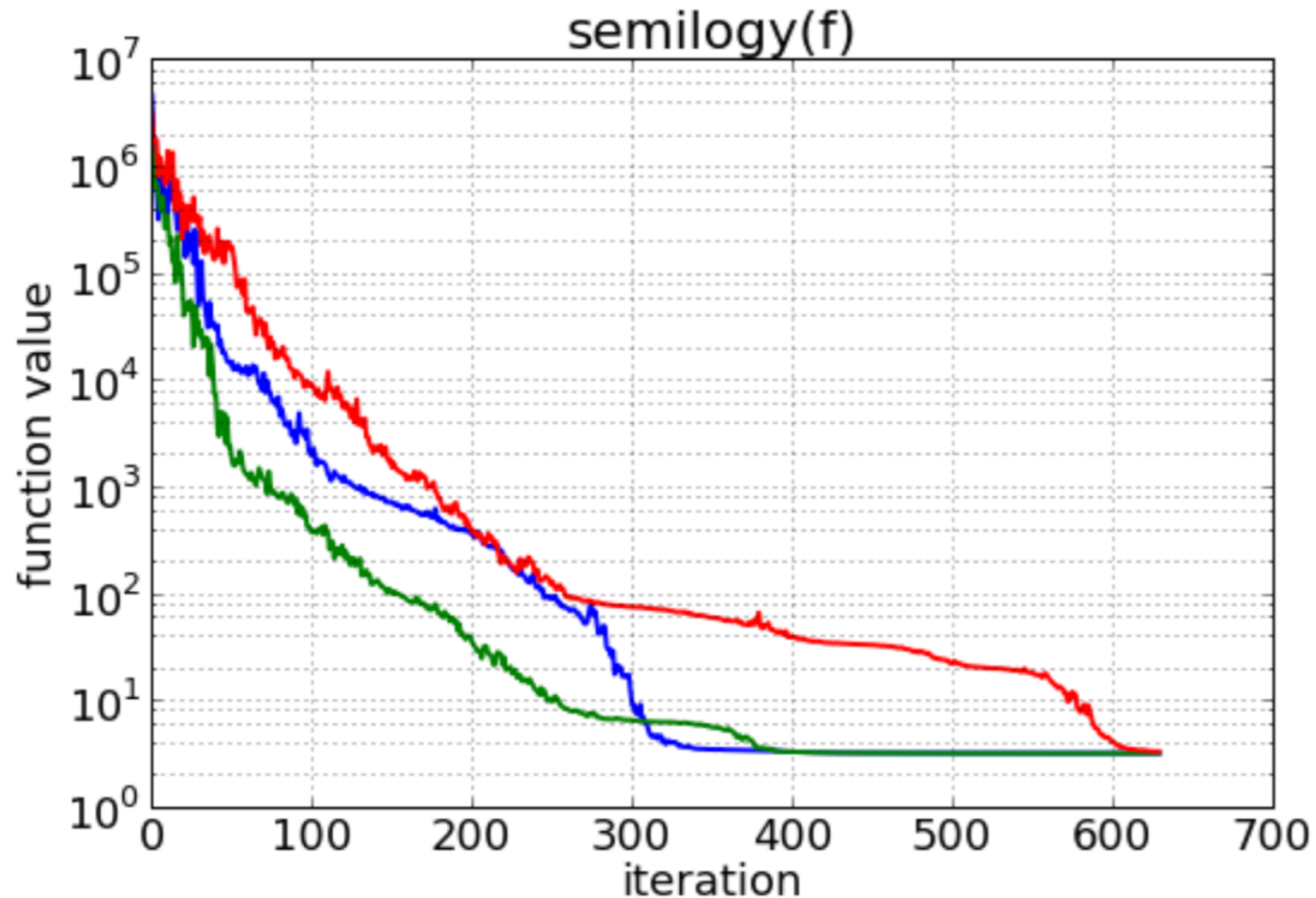
# Displaying Three Runs



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

why not, what's wrong with it?

# Displaying Three Runs

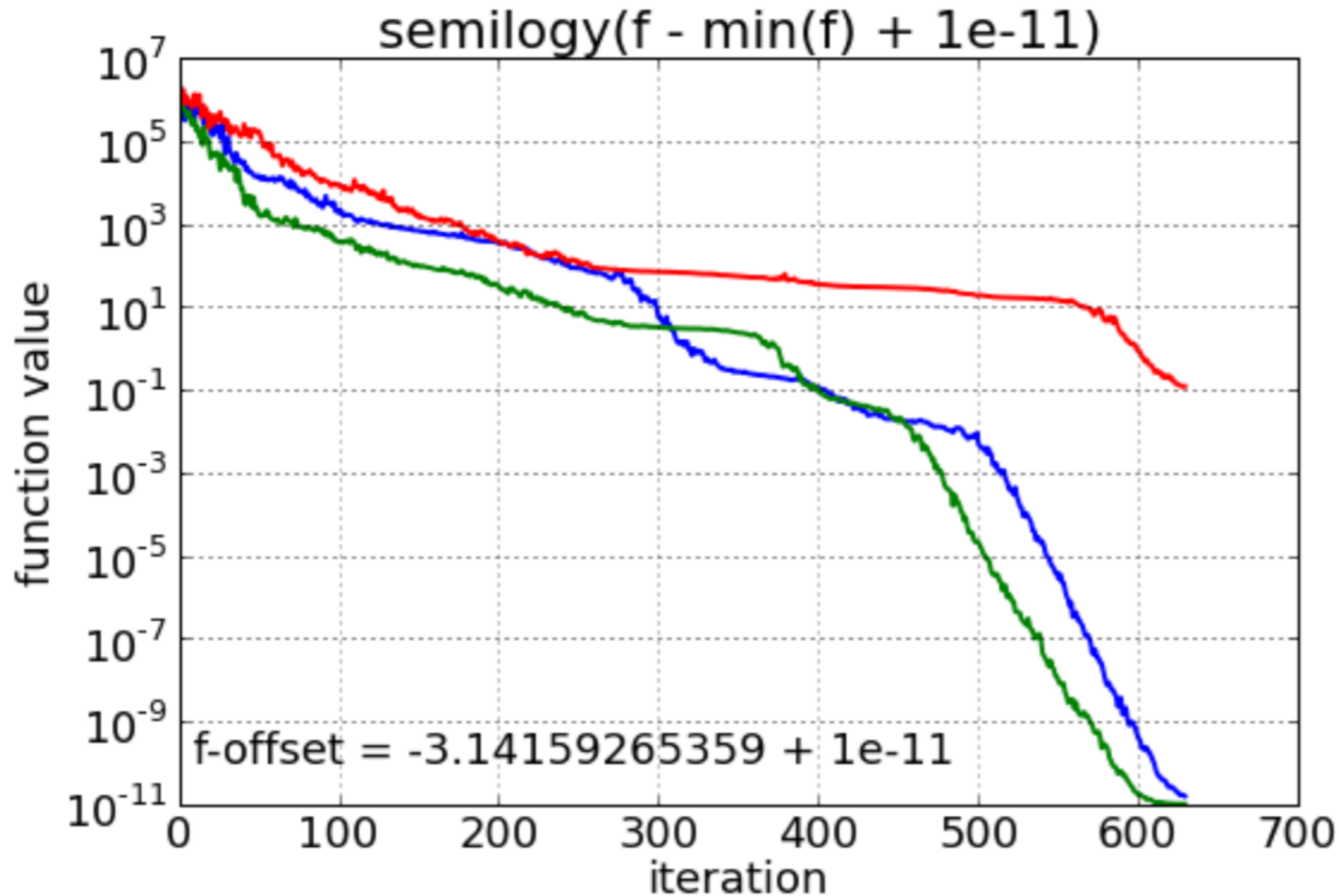


better like this (shown are the same data),

caveat: fails with negative f-values



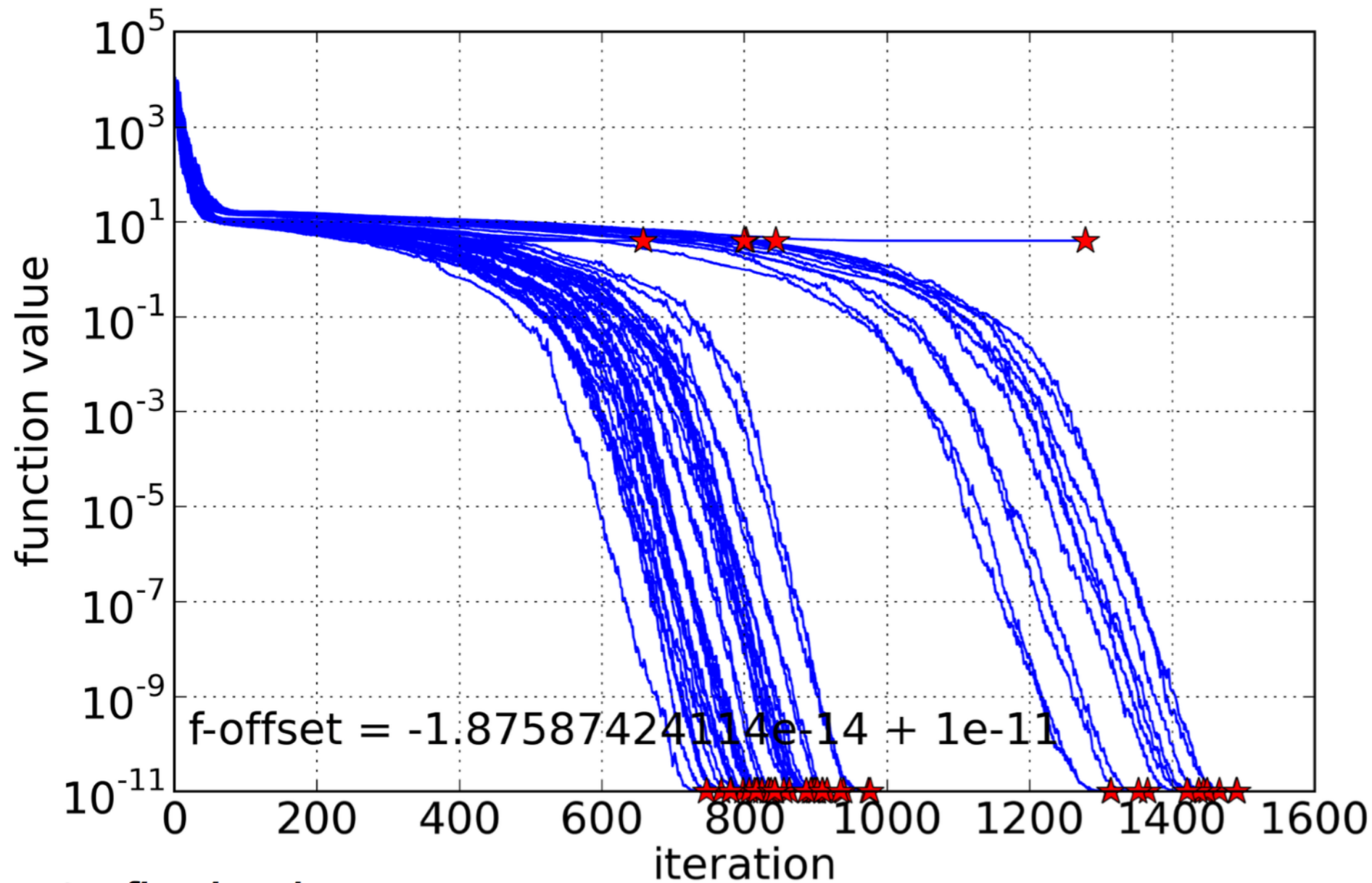
# Displaying Three Runs



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)



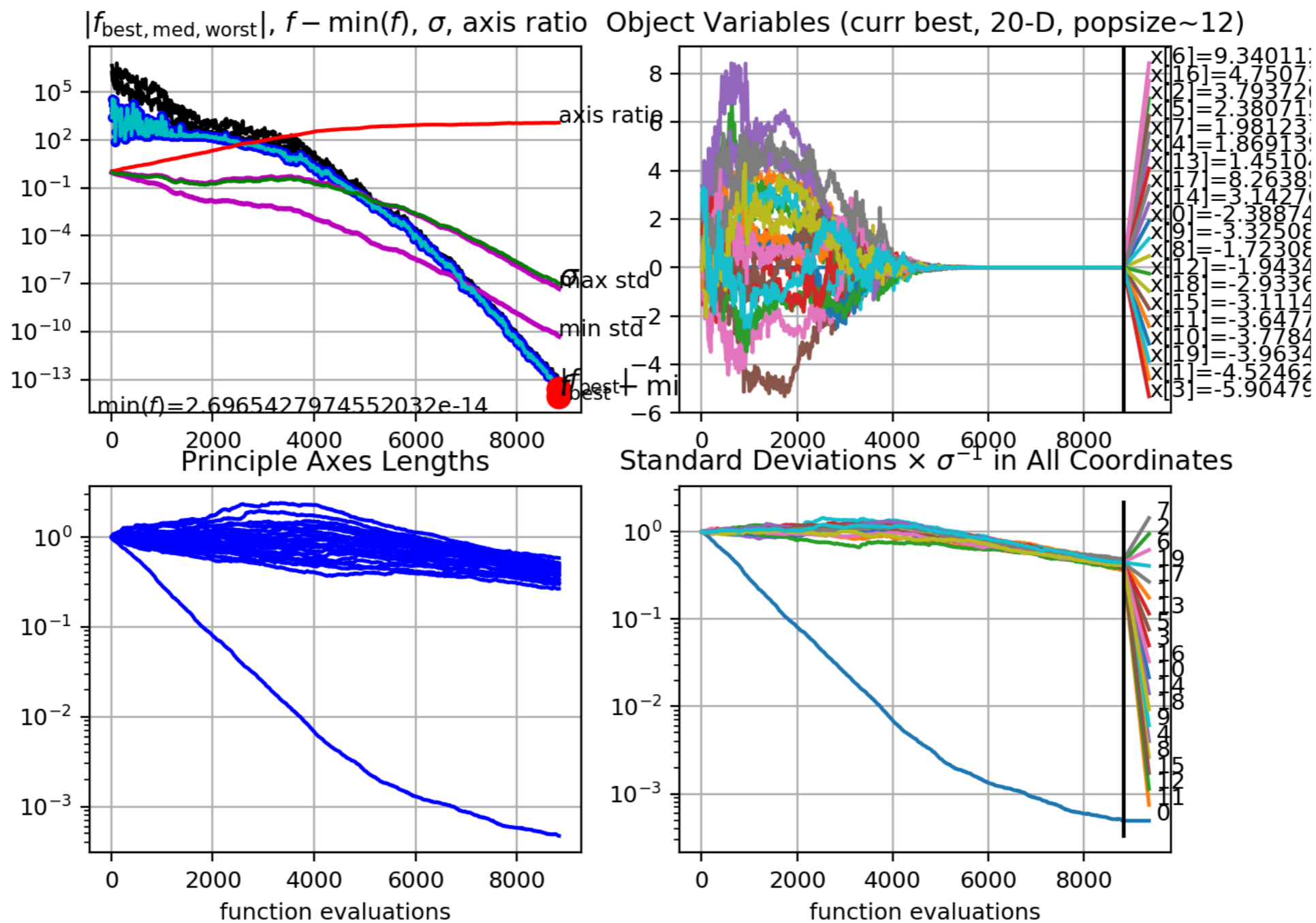
★ : final value

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

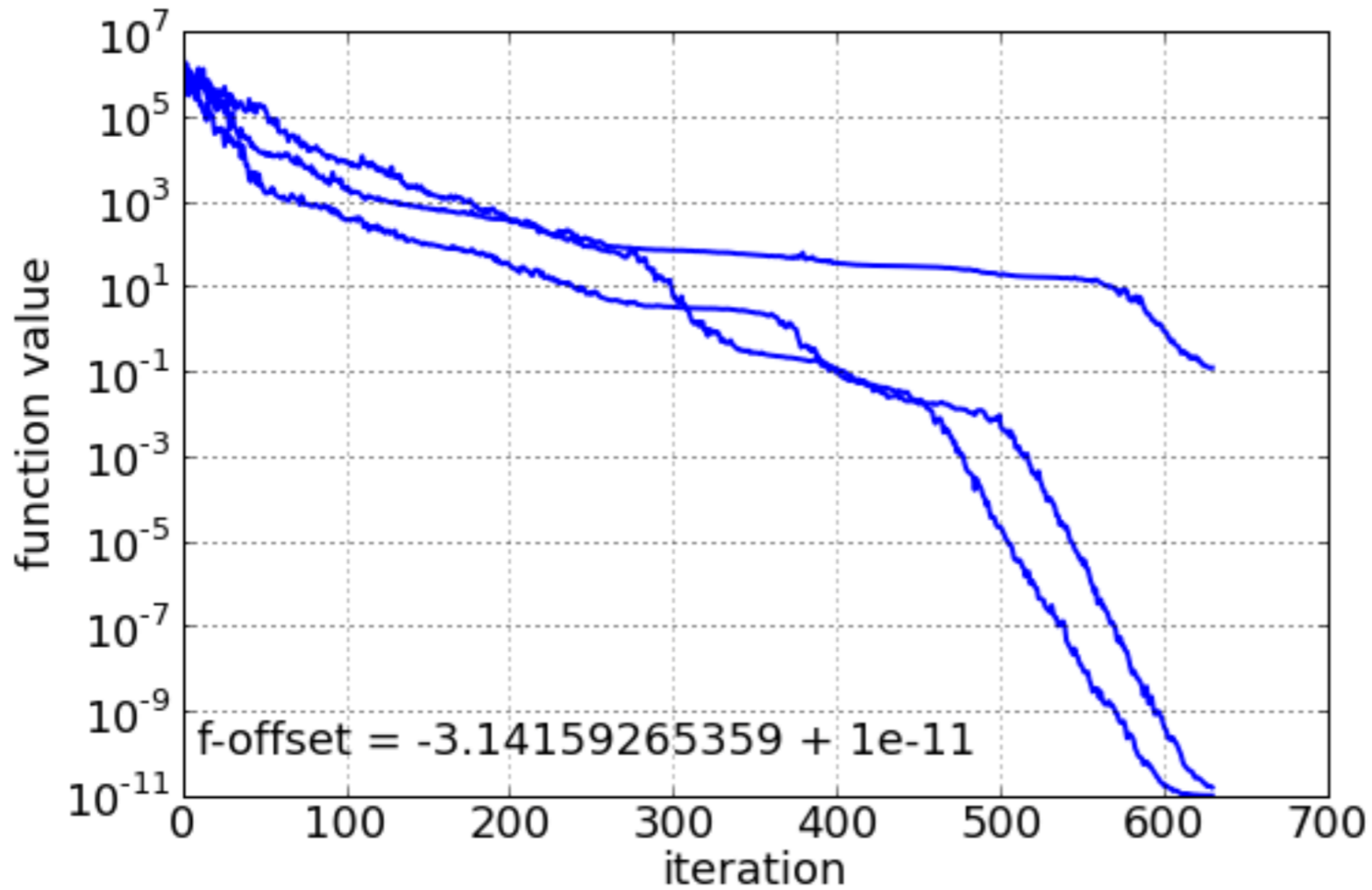
# There is more to display than convergence graphs

cma.plot()

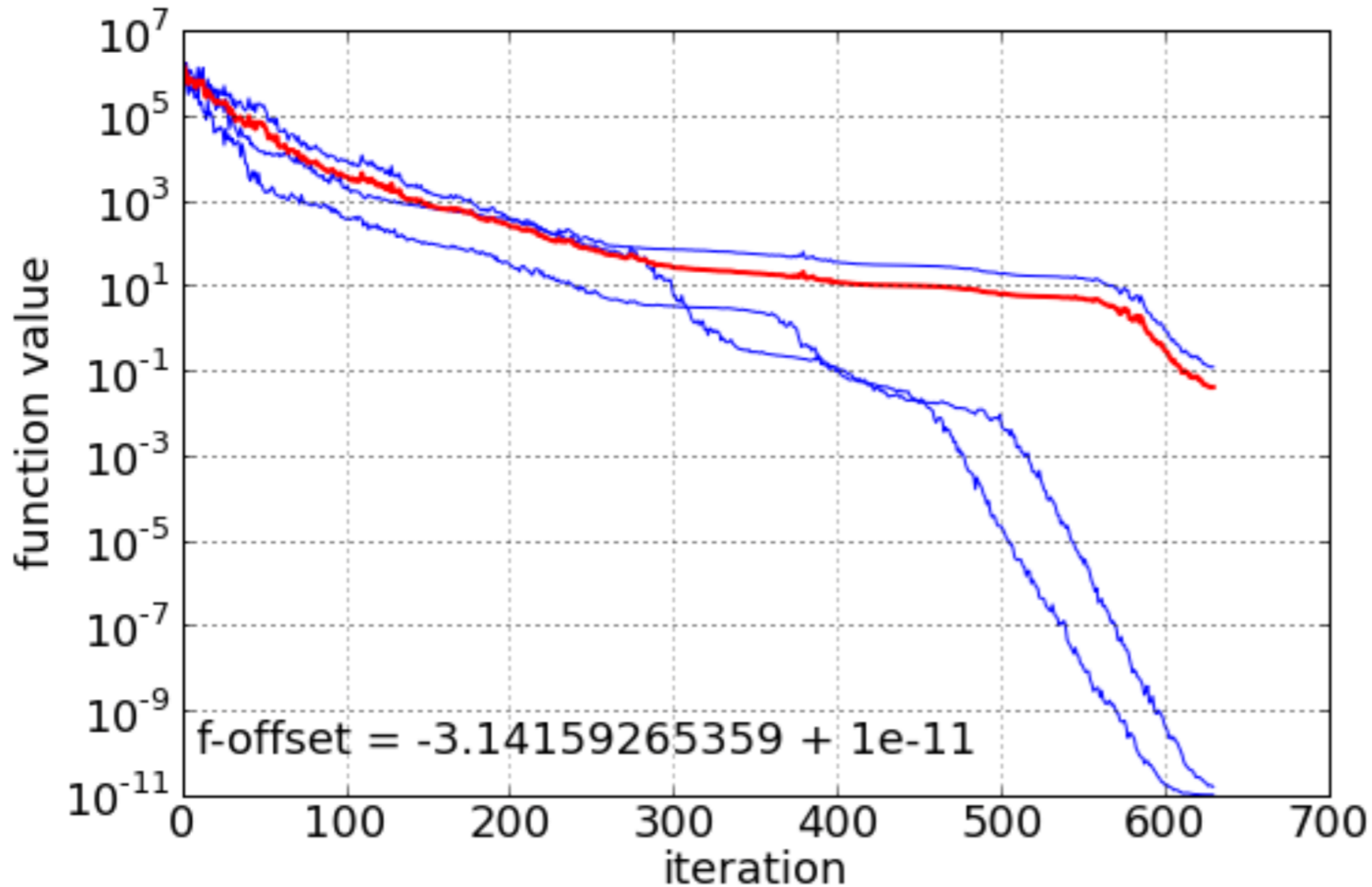
Figure 328



# Aggregation: Which Statistics?



# Aggregation: Which Statistics?

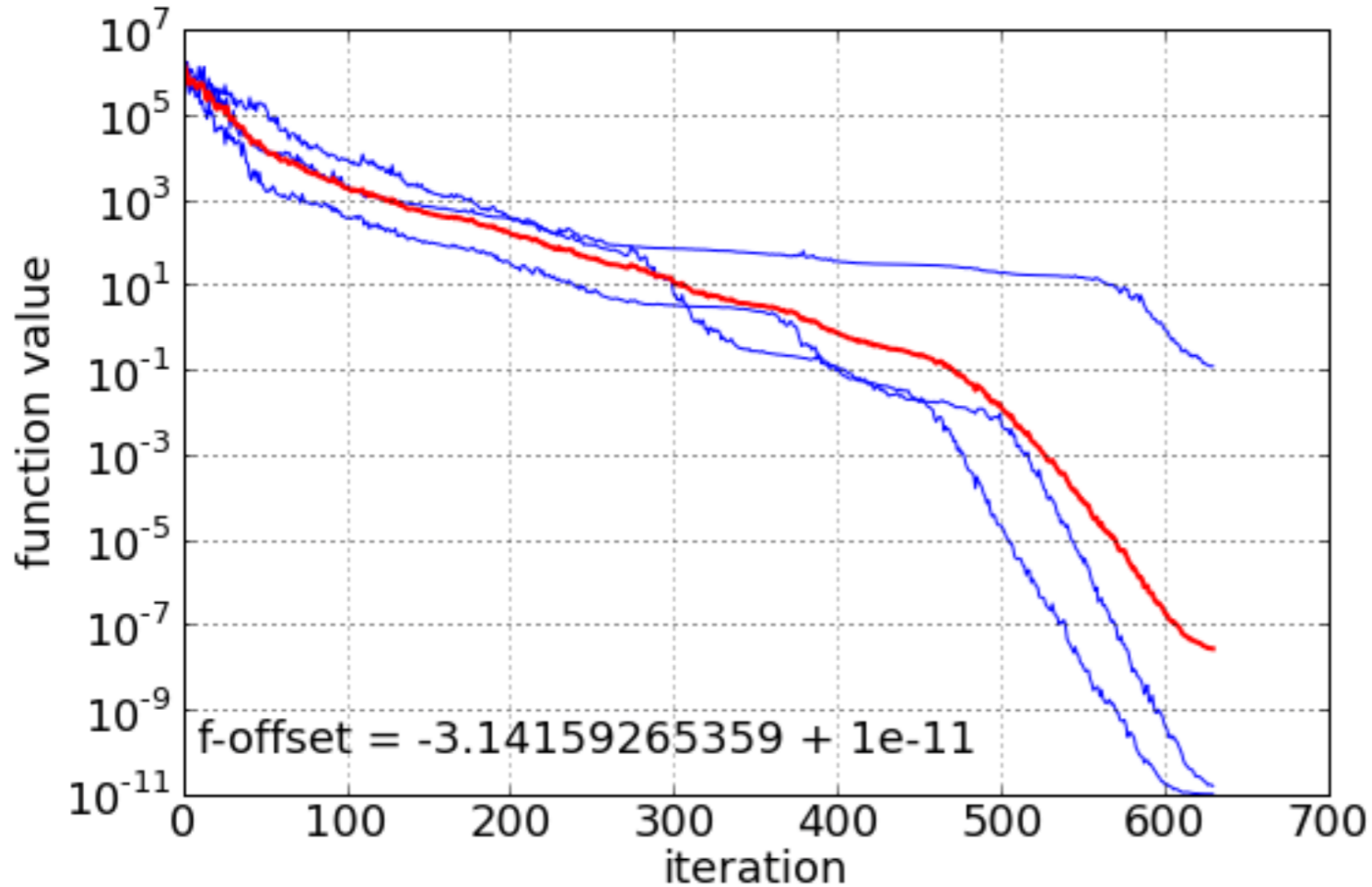


mean/average function value

- tends to emphasize large values



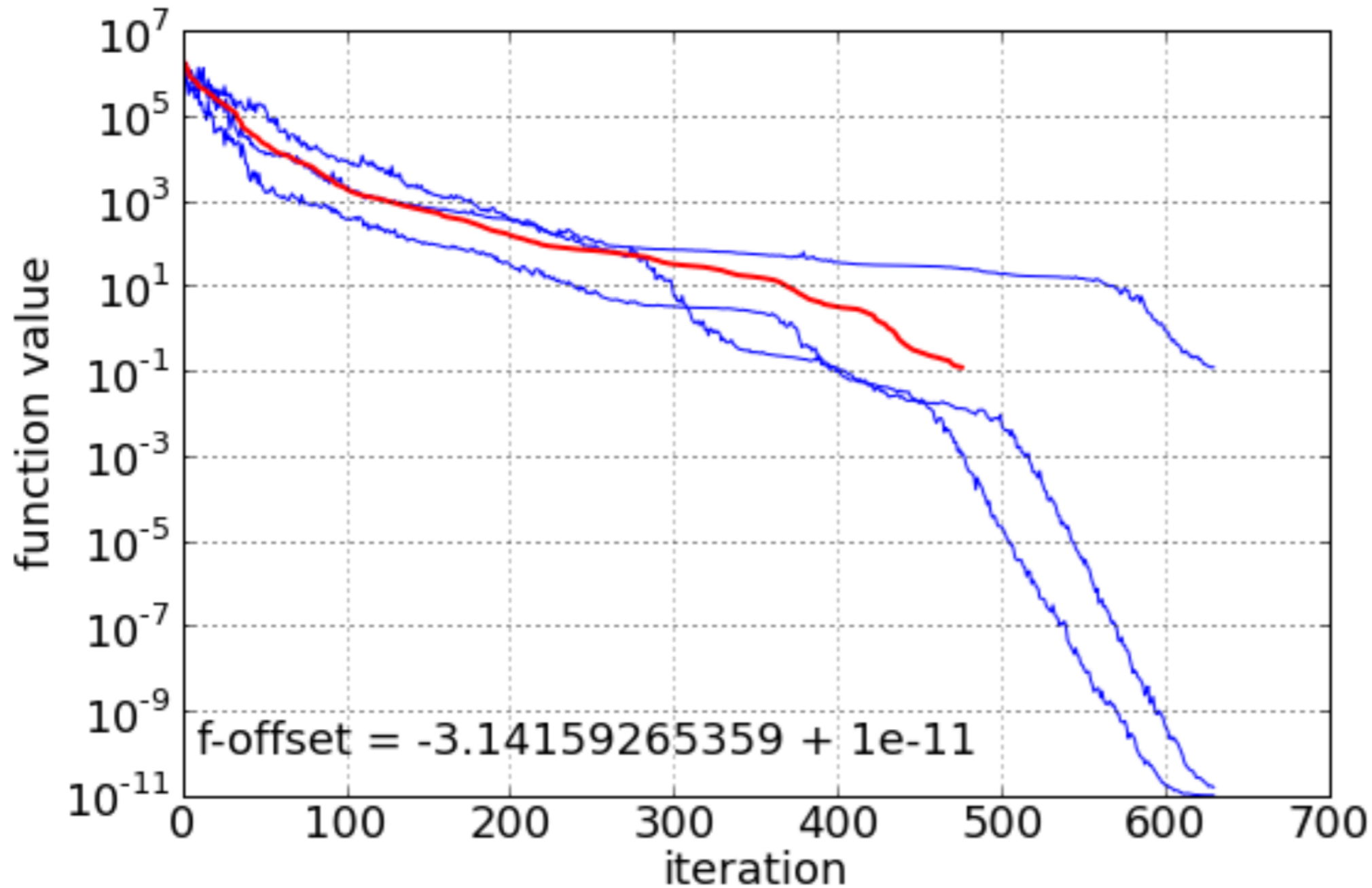
# Aggregation: Which Statistics?



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

- reflects "visual" average
- depends on offset

# Aggregation: Which Statistics?

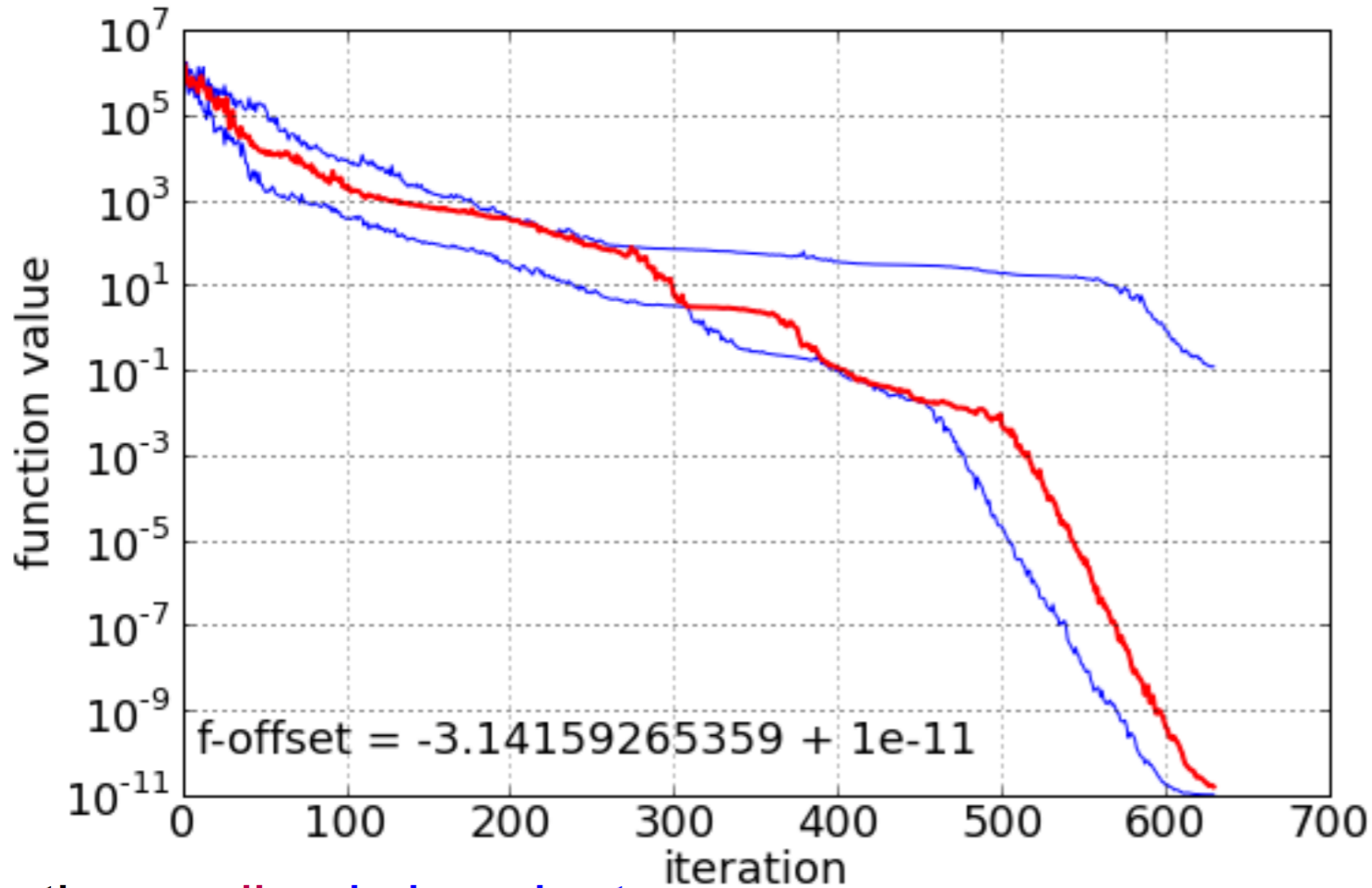


## average iterations

- reflects "visual" average
- here: incomplete



# Aggregation: Which Statistics?



the **median** is invariant

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

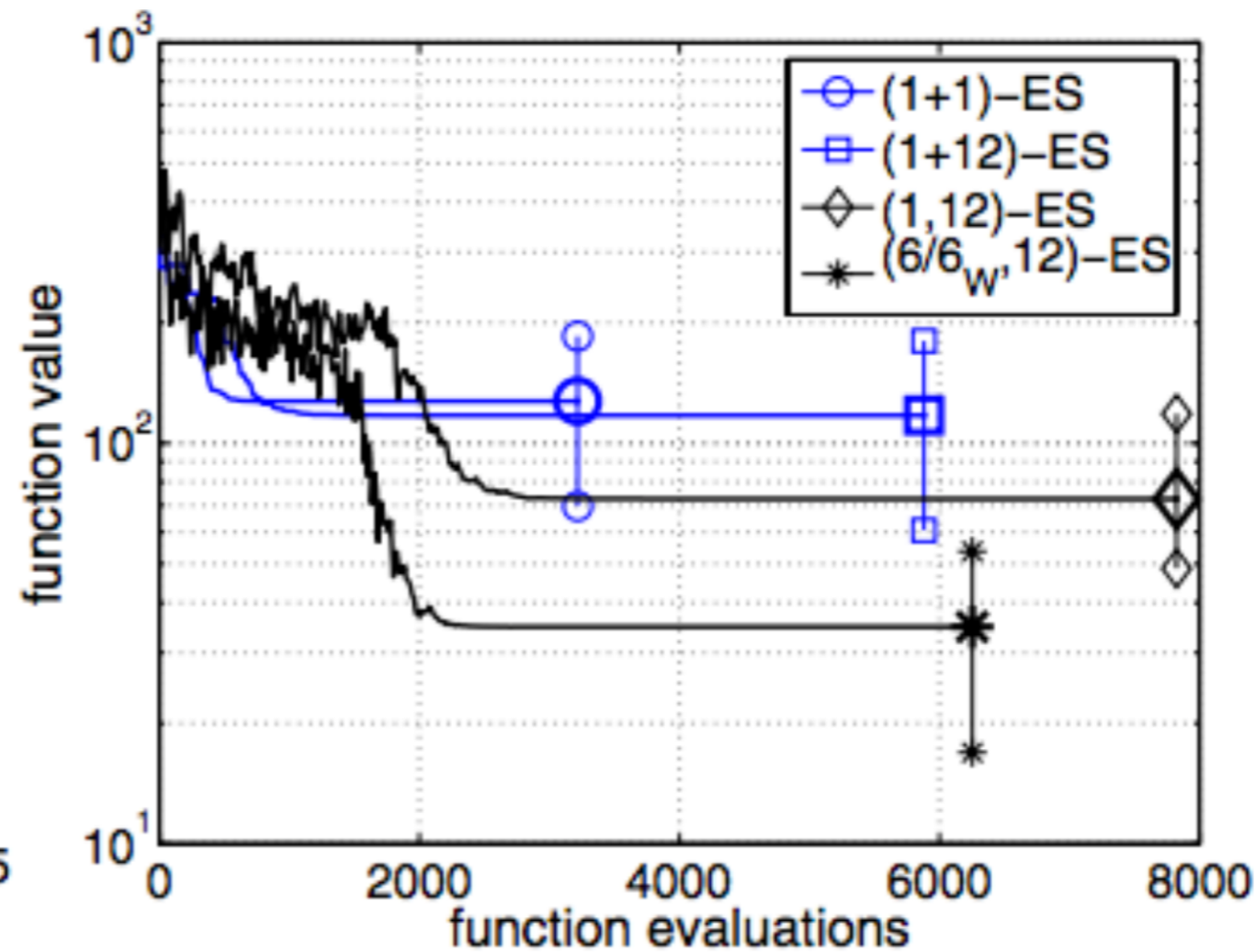
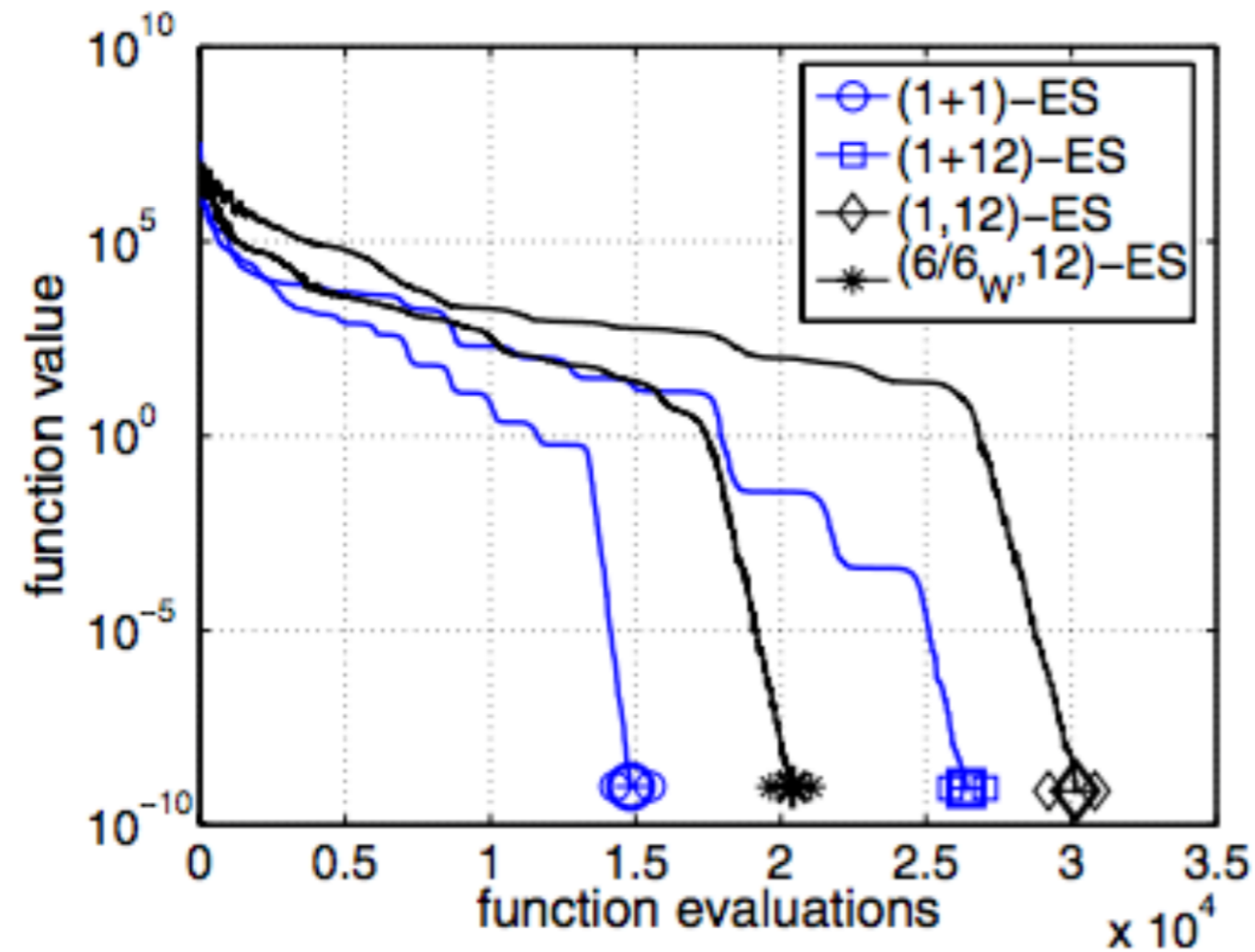
$$\text{median}(\log(\text{data})) = \log(\text{median}(\text{data}))$$

- same when taken over x- or y-direction

# Implications

- 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

# Examples

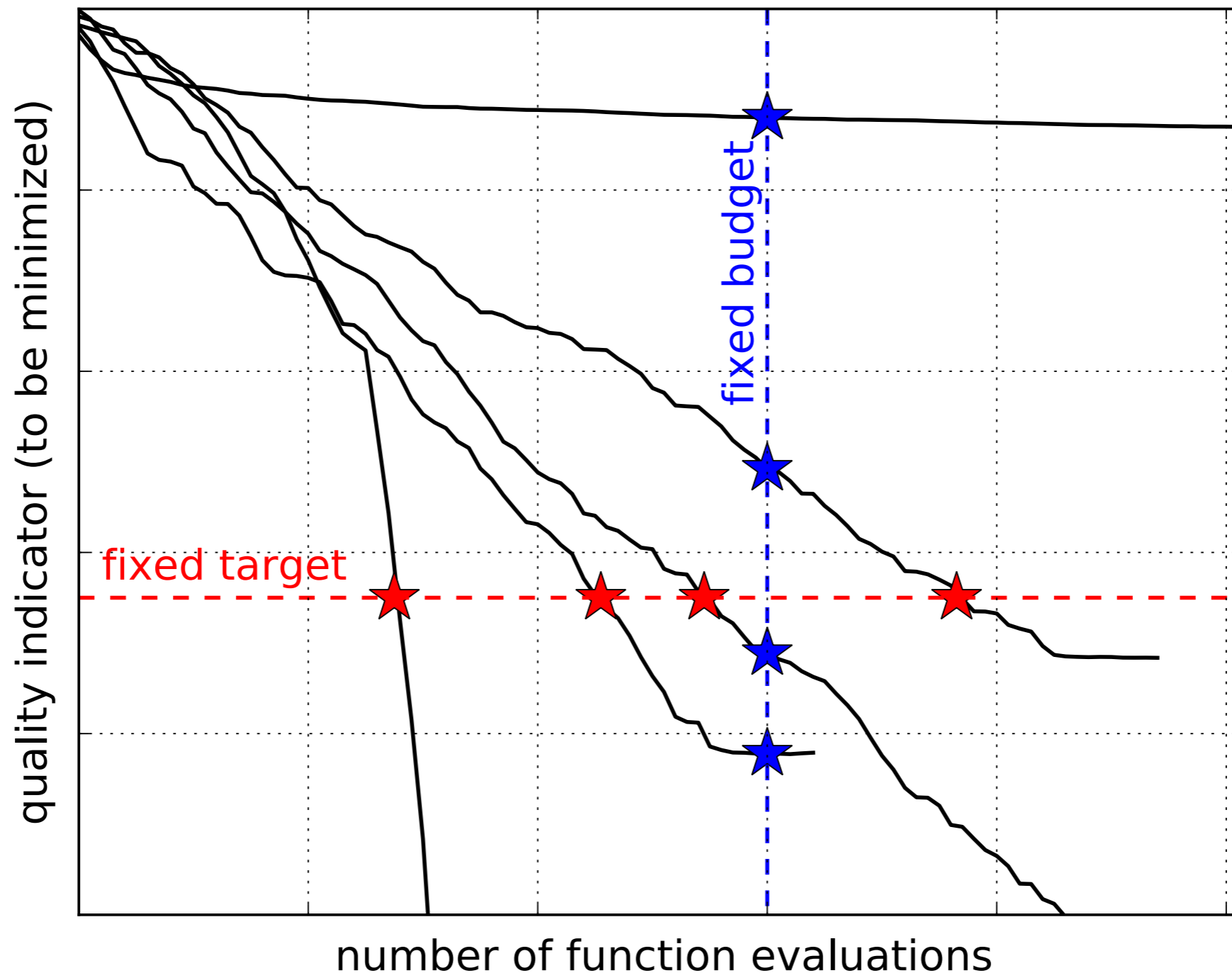


Comparison of 4 algorithms using the "median run" and the 90% central range of the final value on two different functions (Ellipsoid and Rastrigin)

**caveat: the range display with error bars fails, if, for example, only 30% of all runs "converge"**

How can we deal with large variations?

# Aggregation: Fixed Budget vs Fixed Target



- for aggregation we need **comparable** data
- missing data: problematic when many runs lead to missing data
  - fixed target approach misses out on bad results (we may correct for this to some extent)
  - fixed budget approach misses out on good results

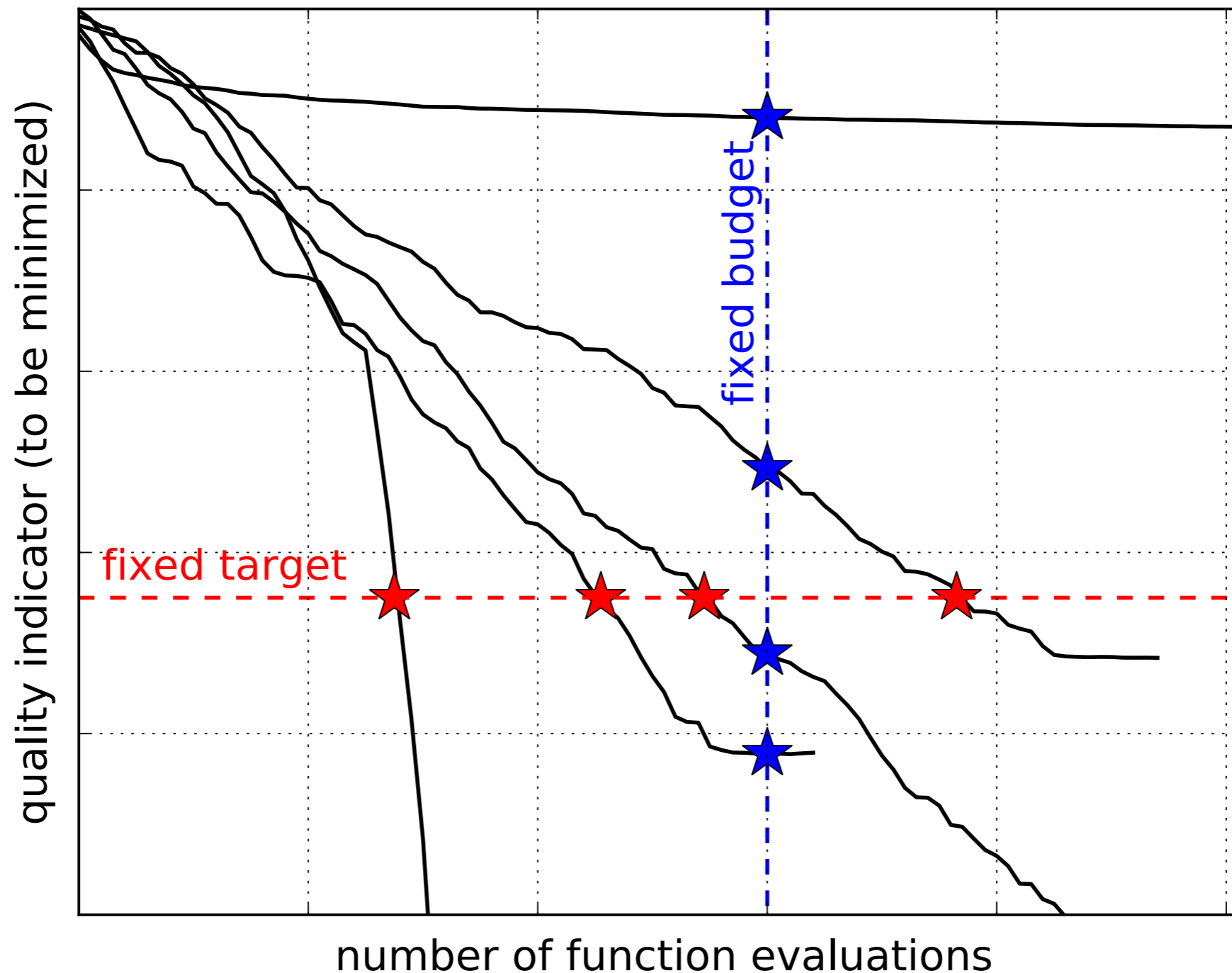
# Performance Measures for Evaluation

Generally, a performance measure should be

- **quantitative** on the ratio scale (highest possible)  
“algorithm A is two *times* better than algorithm B”  
as “performance(B) / performance(A) = 1/2 = 0.5”  
should be meaningful statements
- assuming a wide range of values
- **meaningful (interpretable)** with regard to the real world  
transfer the measure from benchmarking to real world

**runtime** or **first hitting time** is the prime candidate

# Aggregation: Fixed Budget vs Fixed Target



- for aggregation we need **comparable** data
- missing data: problematic when many runs lead to missing data
  - fixed target approach misses out on bad results (we may correct for this to some extent)
  - fixed budget approach misses out on good results

# Fixed Budget vs Fixed Target

Fixed budget => *measuring/display* **final/best  $f$ -values**

Fixed target => *measuring/display* **needed budgets (#evaluations)**

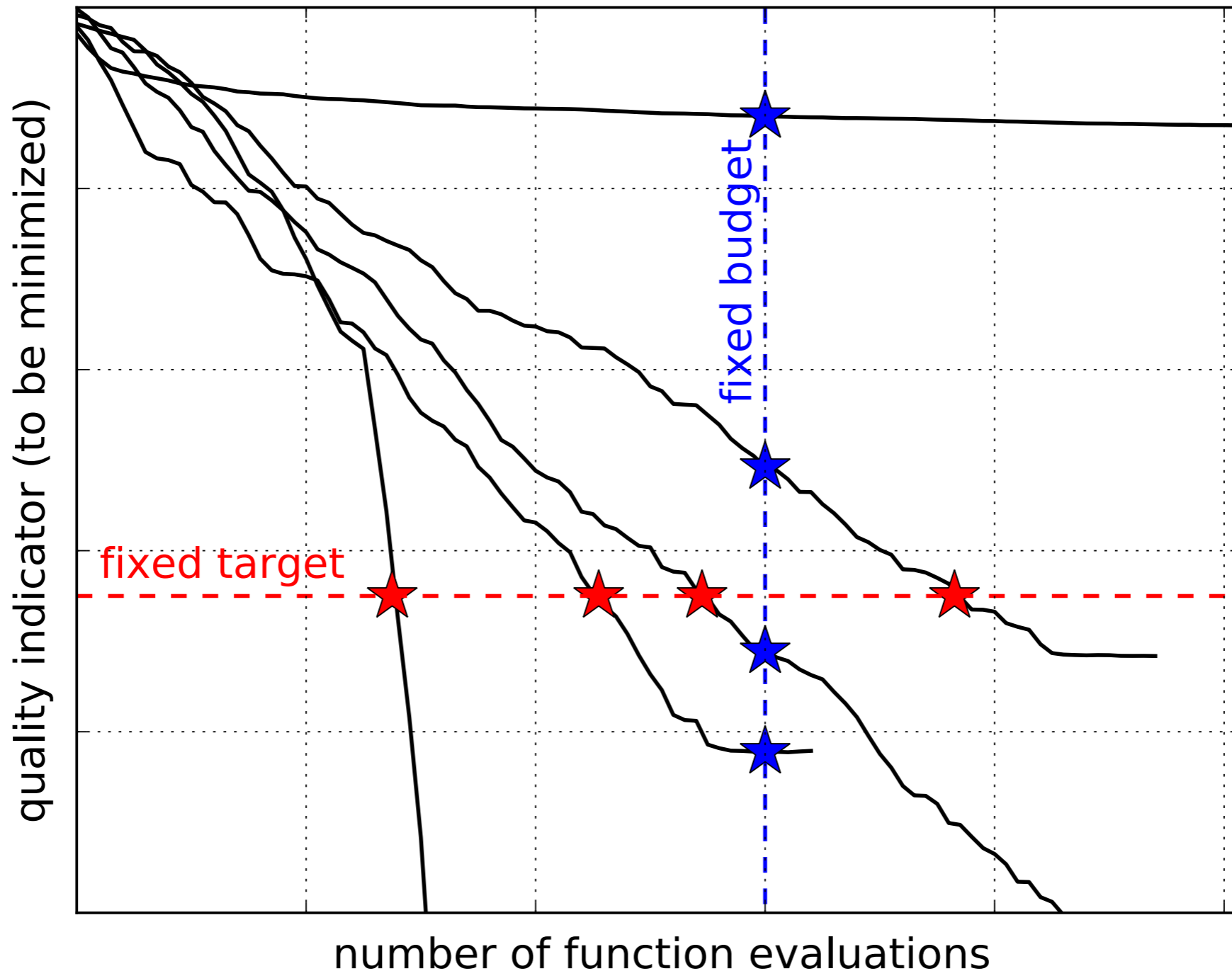
Number of function evaluations:

- are **quantitatively comparable** (on a ratio scale)  
ratio scale: “A is 3.5 times faster than B”,  
 $A/B = 1/3.5$  is a meaningful notion
- the measurement itself is **interpretable** *independently of the function*  
time remains the same time regardless of the underlying problem  
*3 times faster is 3 times faster is 3 times faster* on every problem
- there is a clever way to account for **missing data**  
via restarts

=> fixed target is (much) preferable



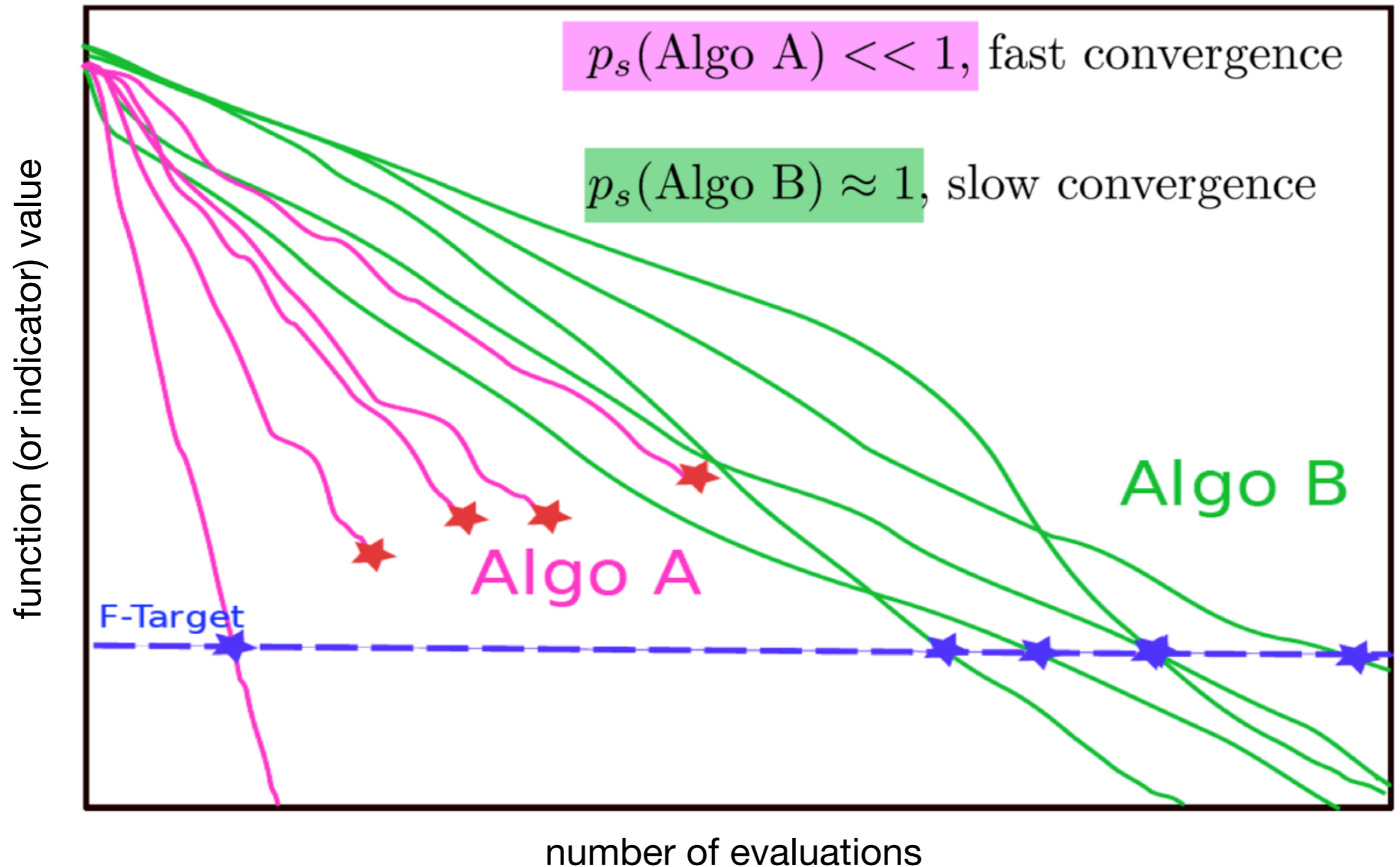
# The Problem of Missing Values



- for aggregation we need **comparable** data
- missing data: problematic when many runs lead to missing data
  - fixed target approach misses out on bad results (we may correct for this to some extent)
  - fixed budget approach misses out on good results

# The Problem of Missing Values

how can we compare the following two algorithms?



# The Problem of Missing Values

Consider 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)

# The Problem of Missing Values

The **expected runtime** (ERT, aka SP2, aRT) to hit a target value in  $\#evaluations$  is computed (estimated) as:


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

unsuccessful runs  
count (only) in the  
nominator

$$= \text{avg}(evals_{succ}) + \overbrace{\frac{N_{unsucc}}{N_{succ}}}^{\text{odds ratio}} \times \text{avg}(evals_{unsucc})$$

$$\approx \text{avg}(evals_{succ}) + \frac{N_{unsucc}}{N_{succ}} \times \text{avg}(evals_{succ})$$

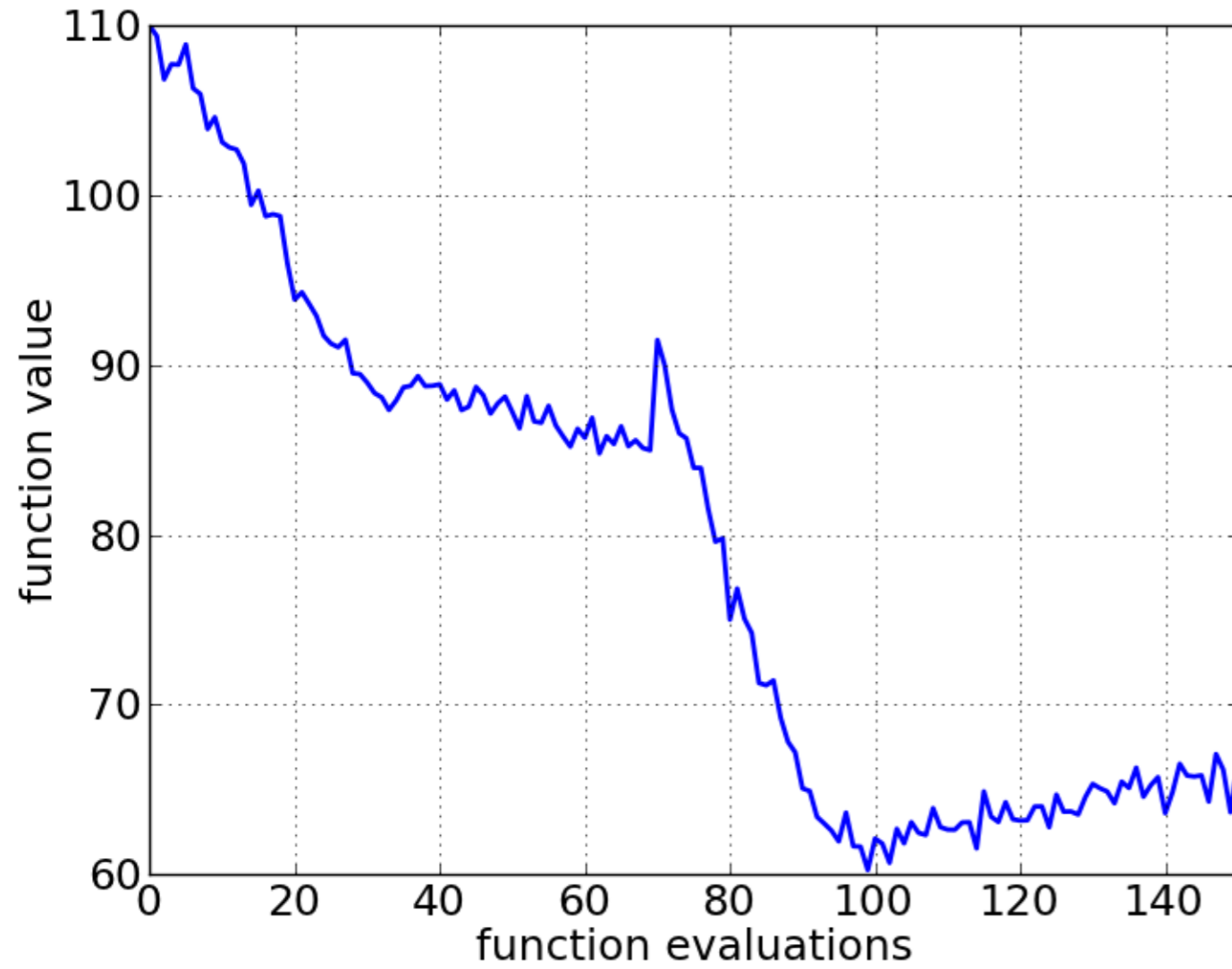
$$= \frac{N_{succ} + N_{unsucc}}{N_{succ}} \times \text{avg}(evals_{succ})$$

$$= \frac{1}{\text{success rate}} \times \text{avg}(evals_{succ})$$

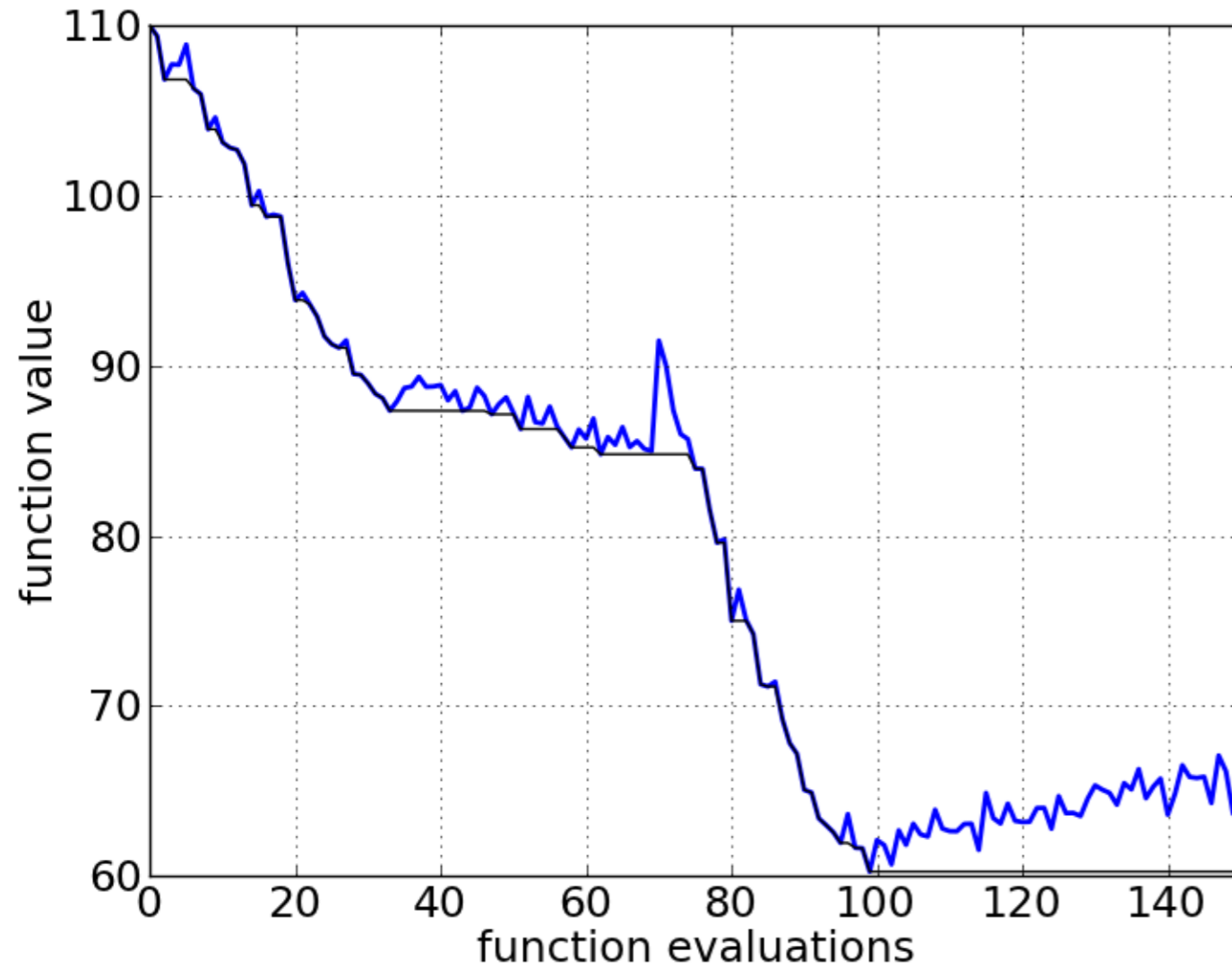
defined (only) for  $\#successes > 0$ . The last three lines are aka Q-measure or SP1 (success performance).

# Empirical Distribution Functions

- Empirical cumulative distribution functions (ECDF, or in short, *empirical distributions*) are arguably the single most powerful tool to “aggregate” data in a display.

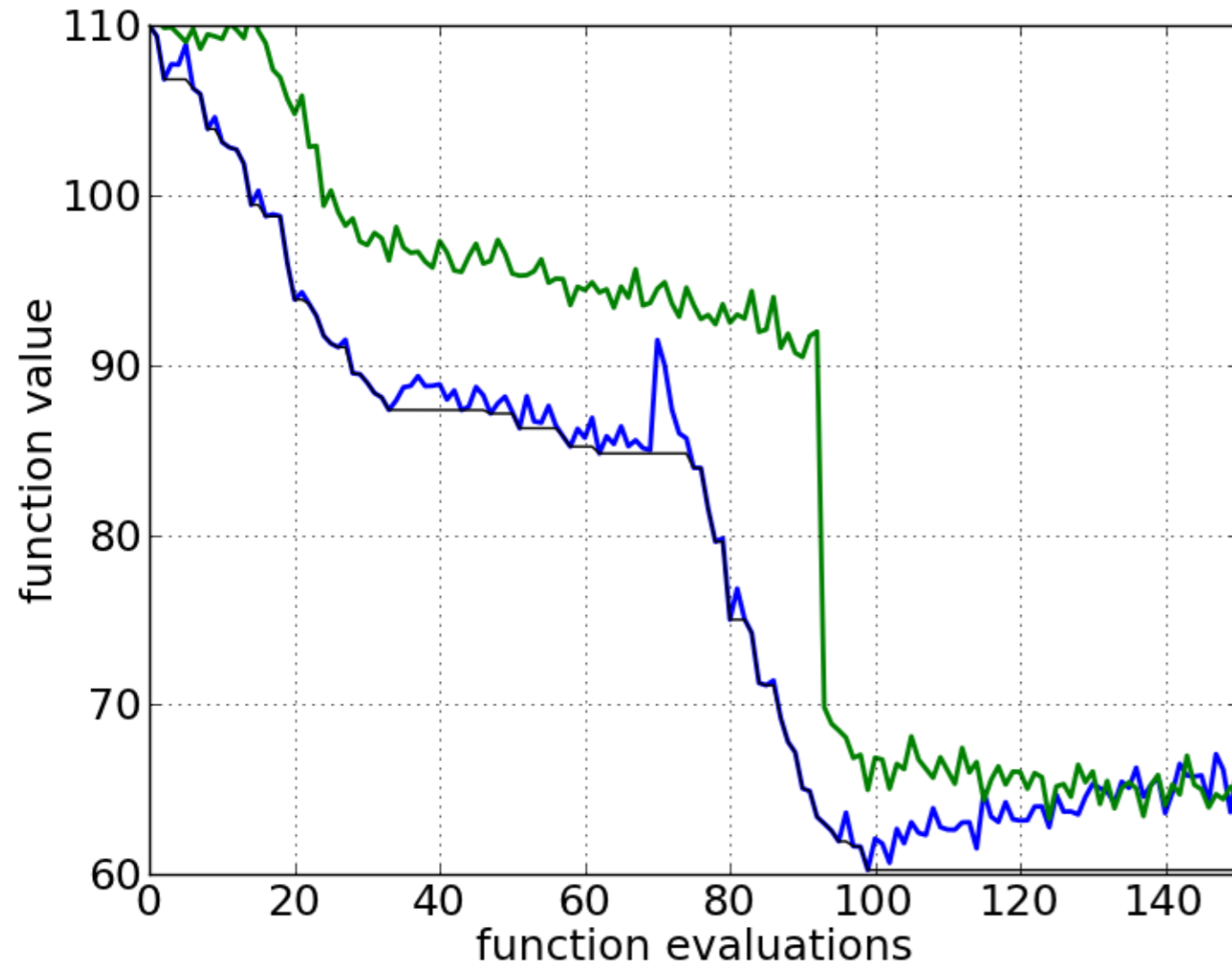


- a convergence graph

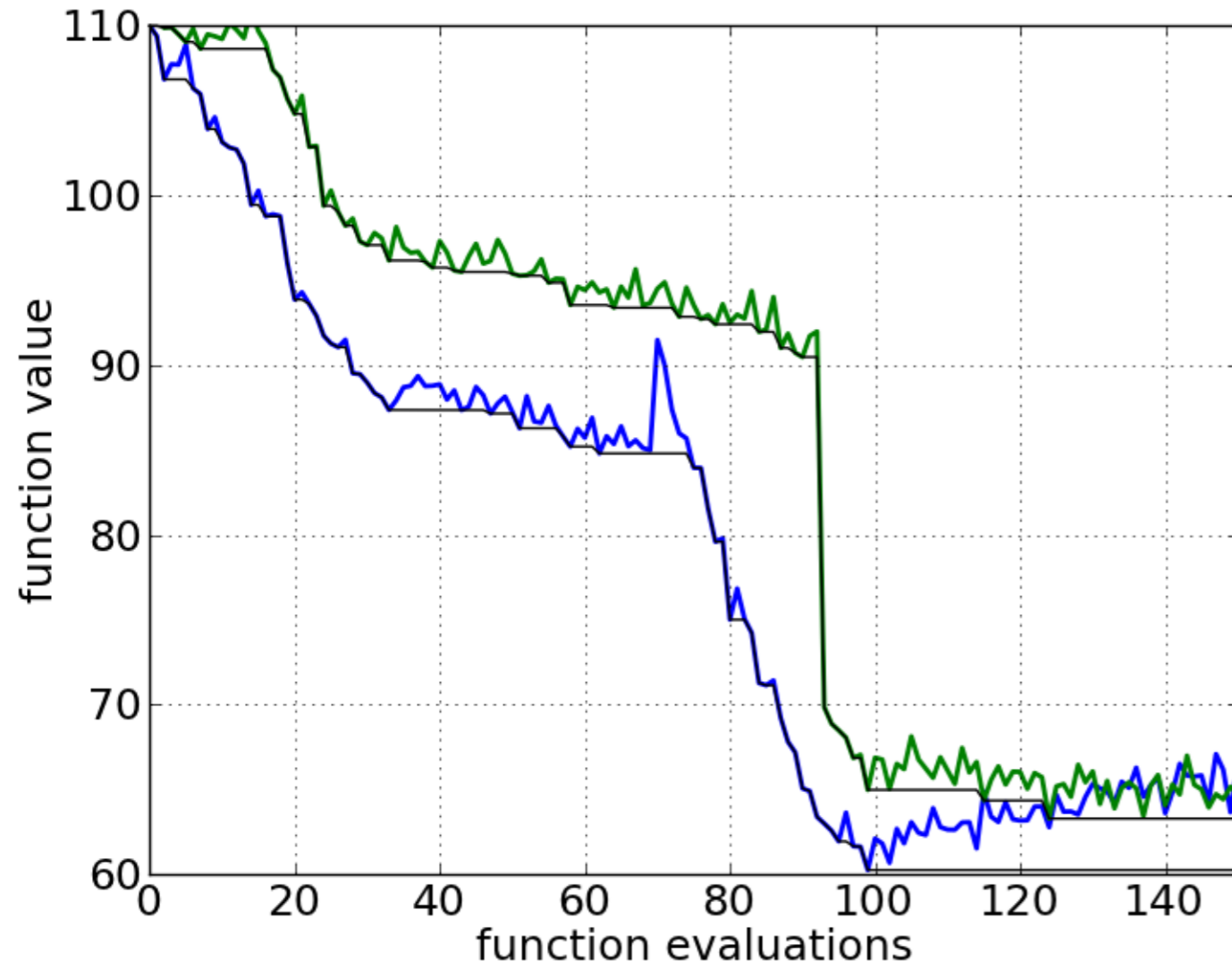


- a convergence graph
- first hitting time (black): lower envelope, a monotonous graph

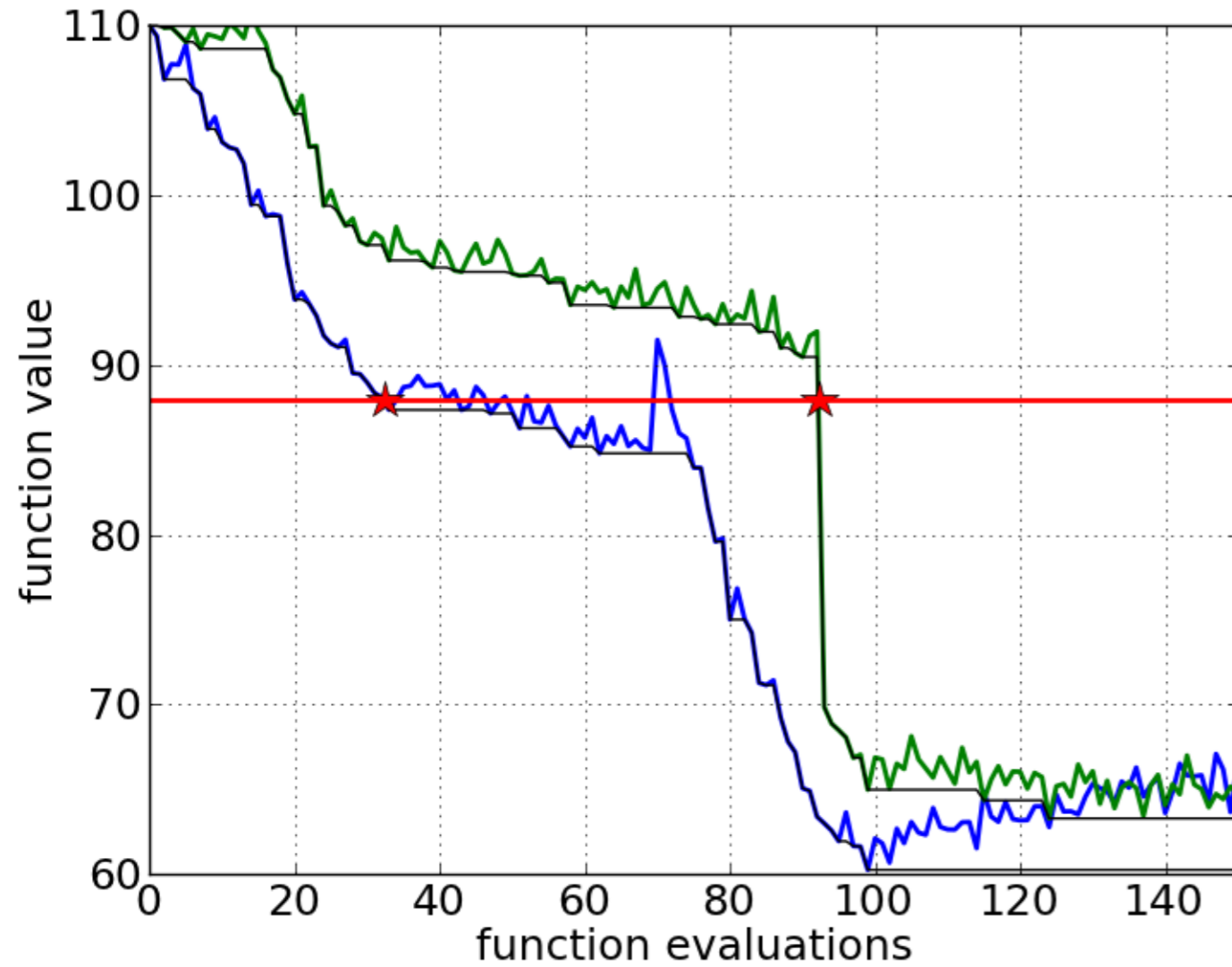




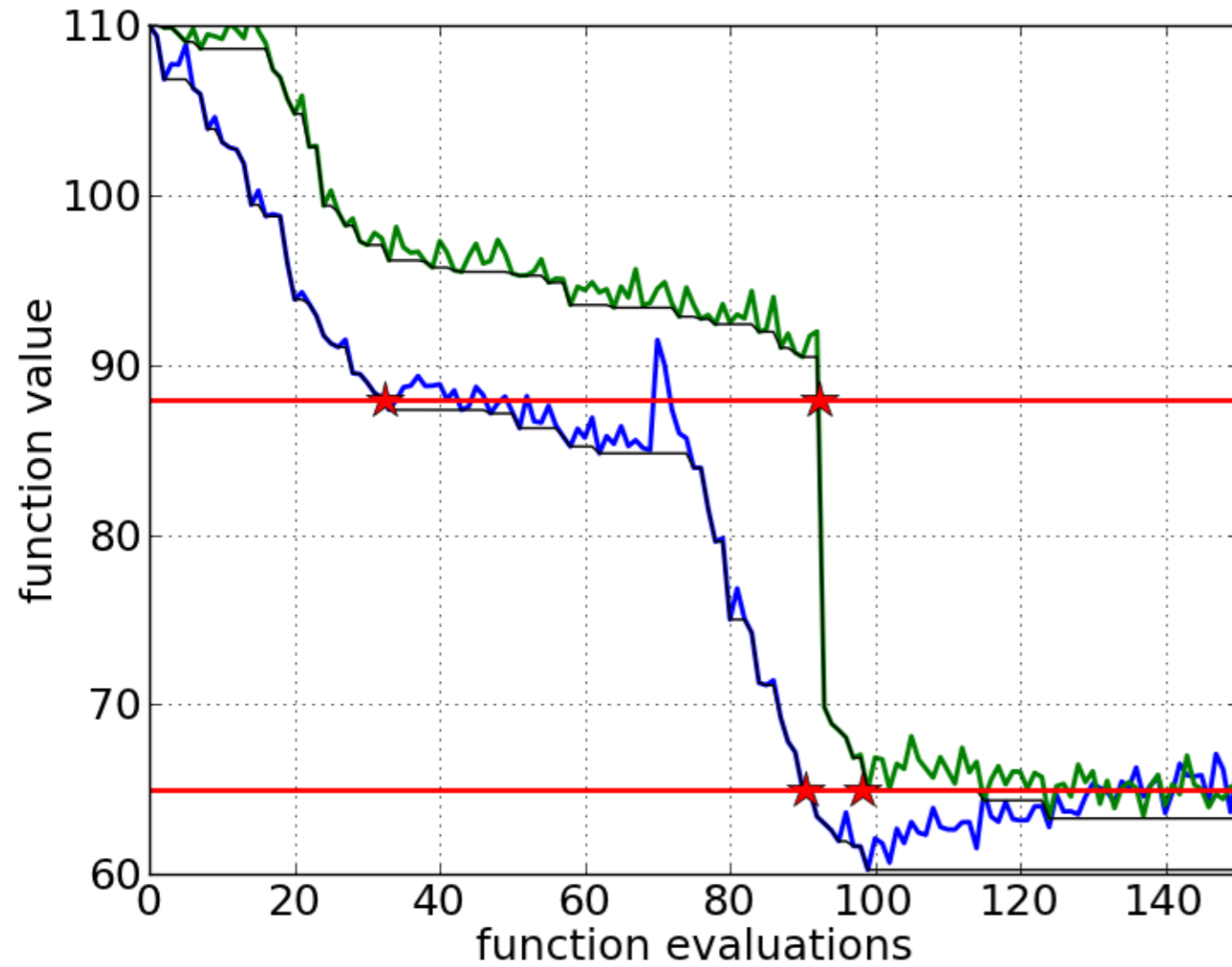
- another convergence graph



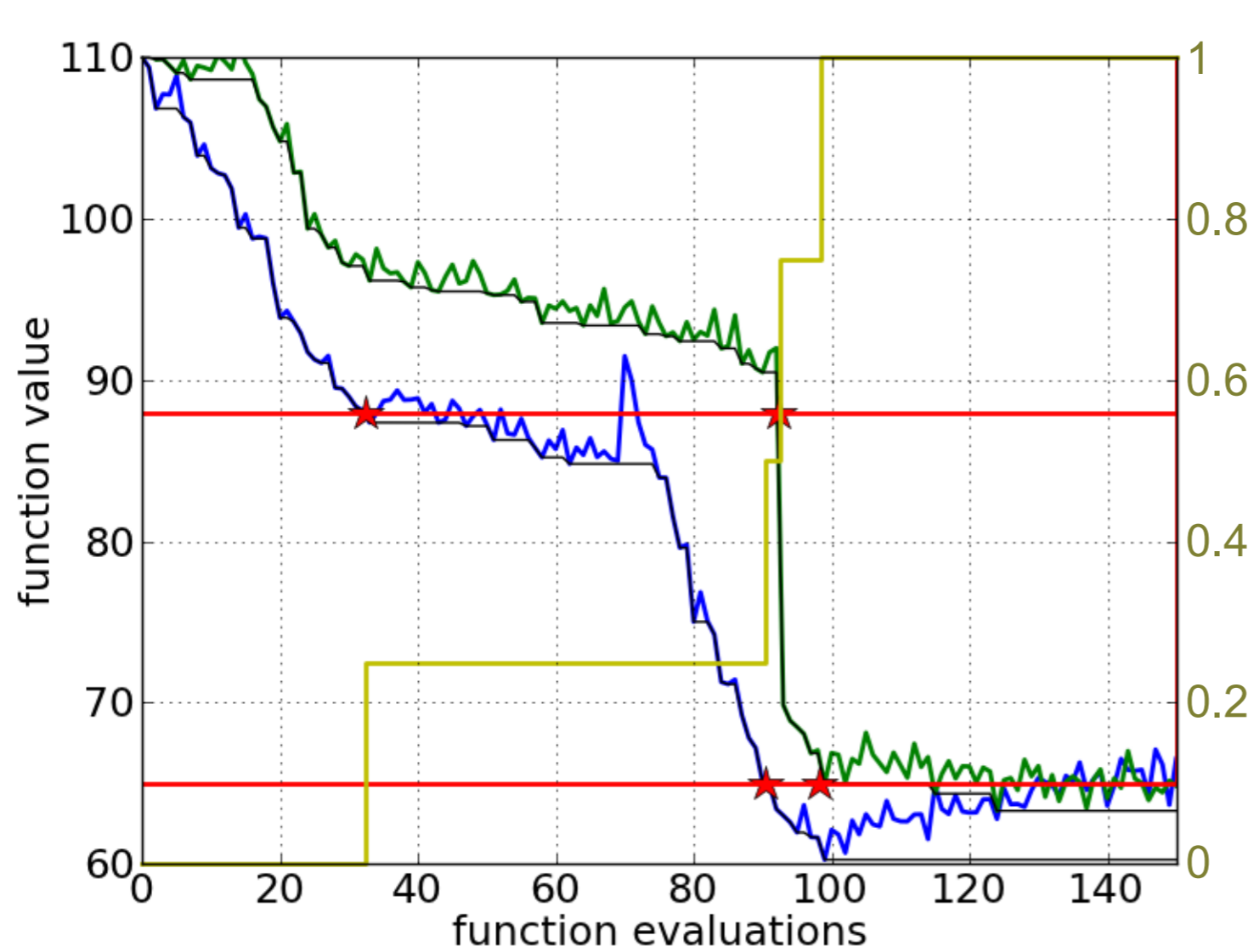
- another convergence graph with hitting time



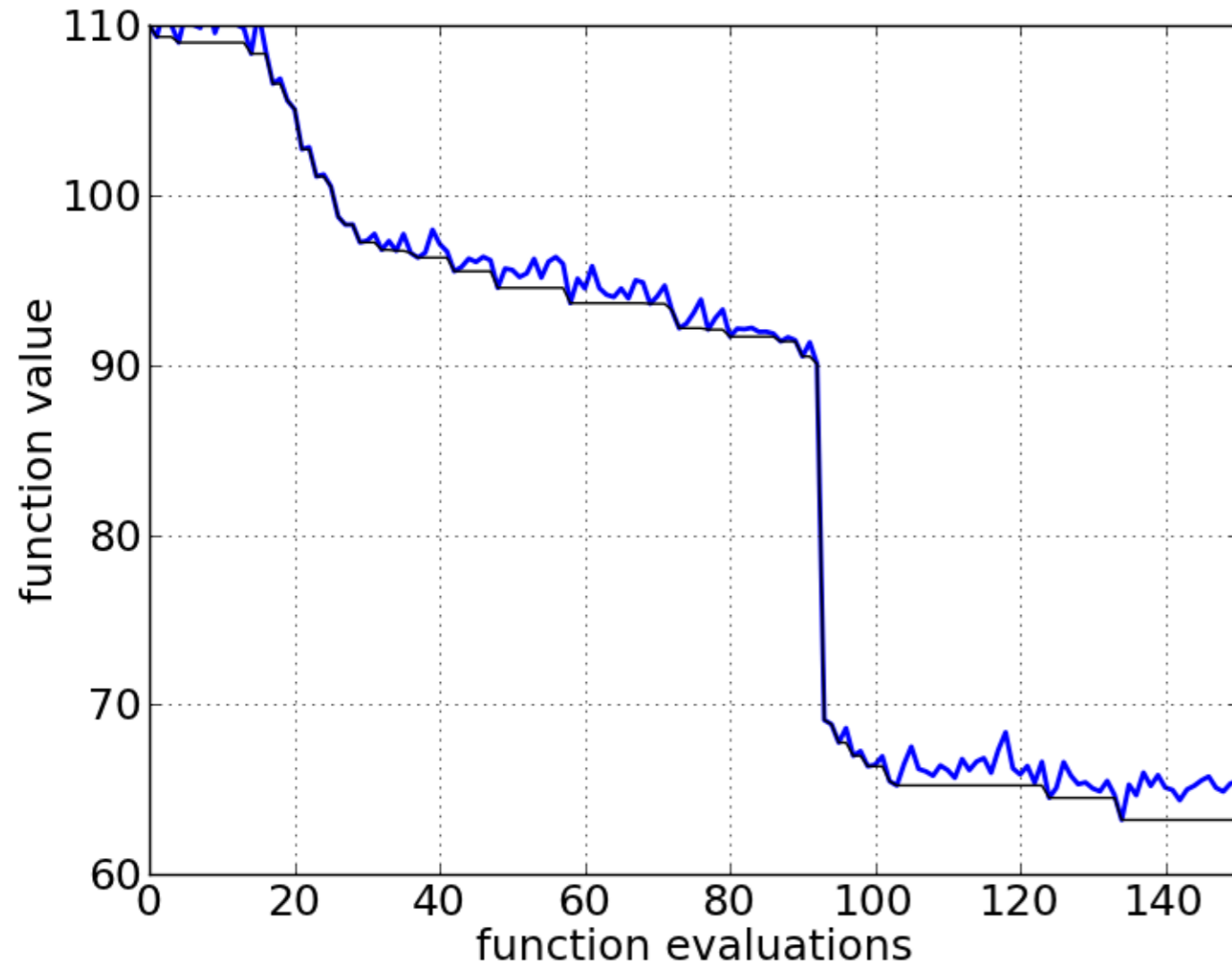
- a target value delivers two data points (or possibly missing values)



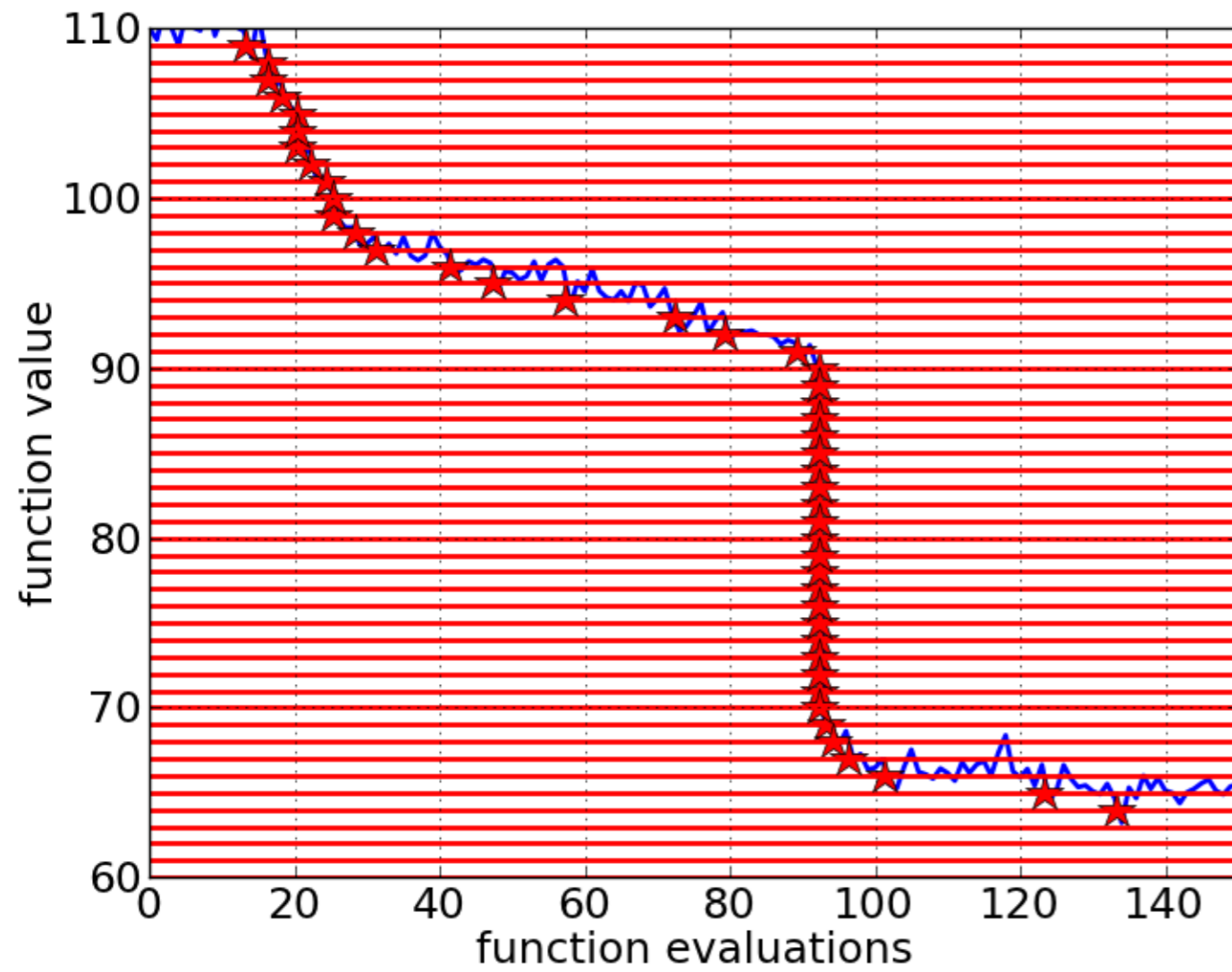
- another target value delivers two more data points



- the ECDF with four steps (between 0 and 1)

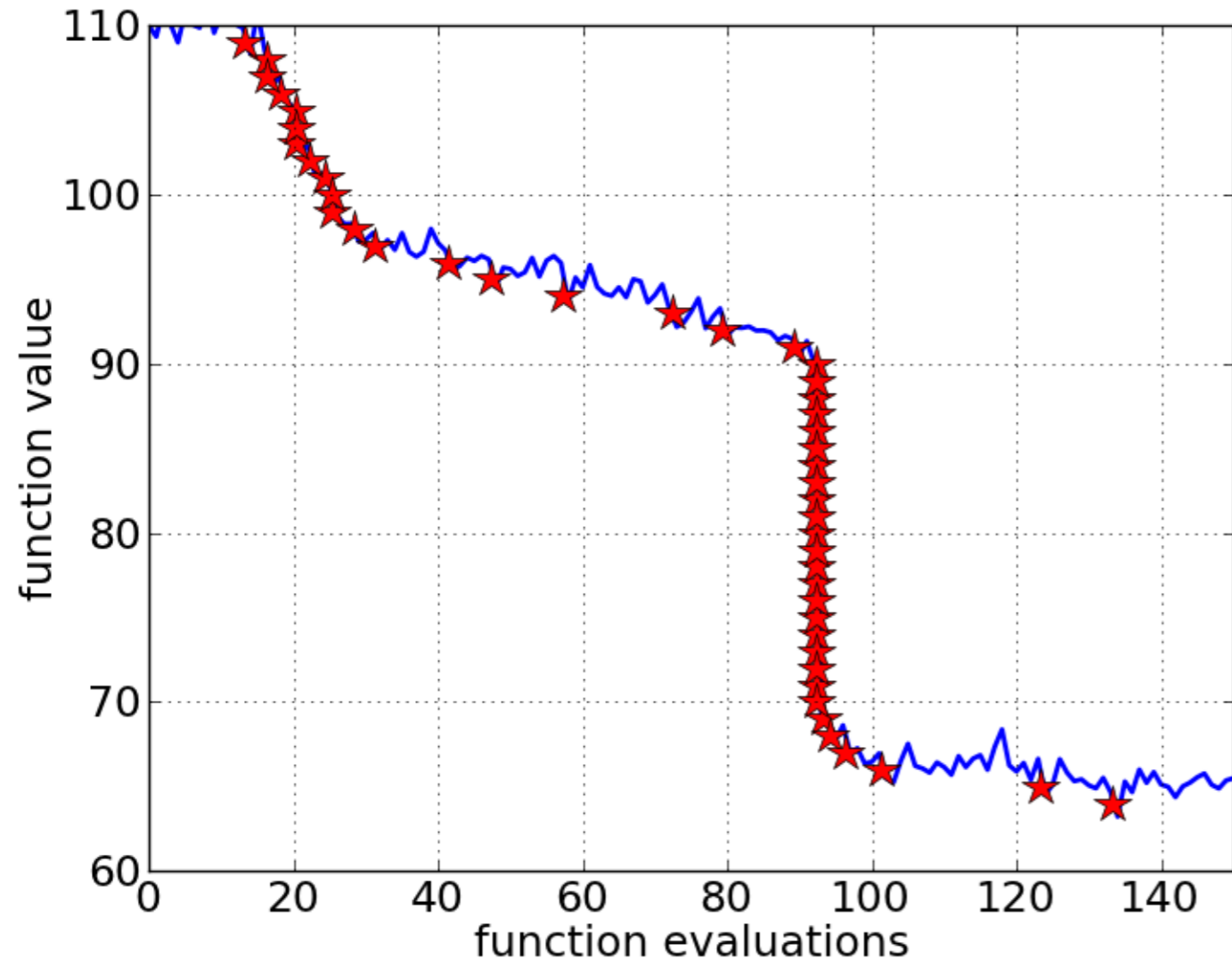


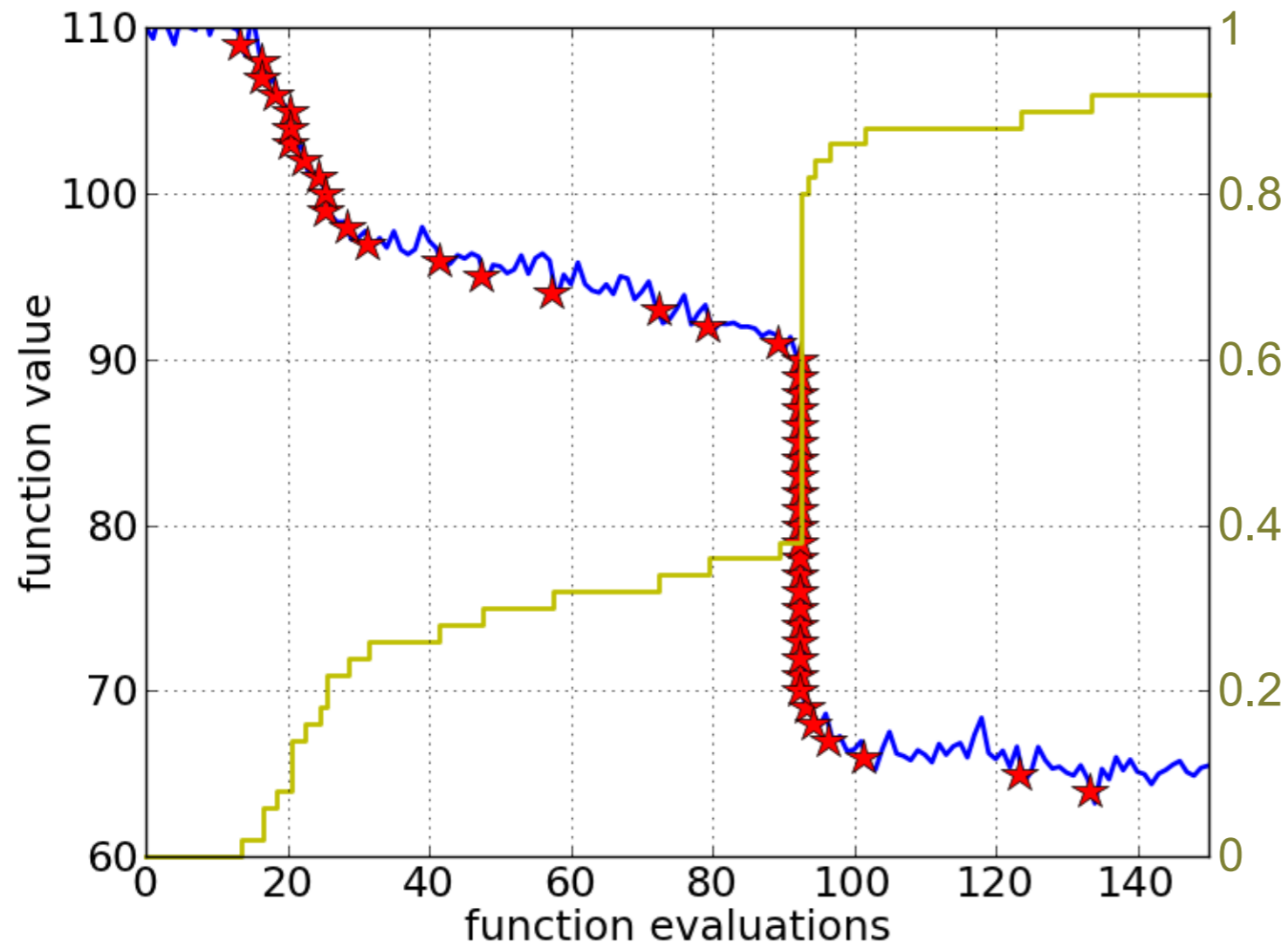
- reconstructing a single run



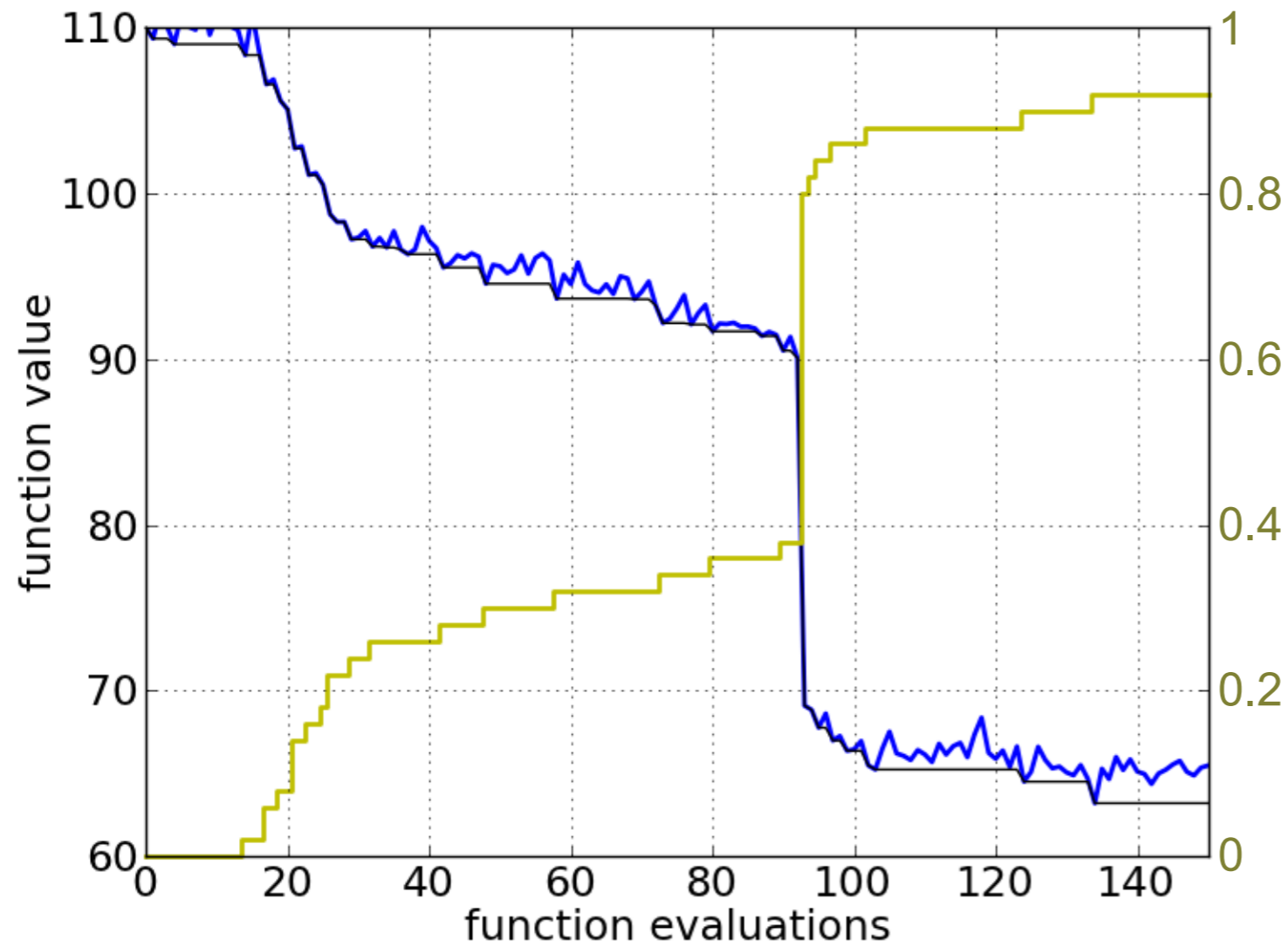
50 equally spaced targets



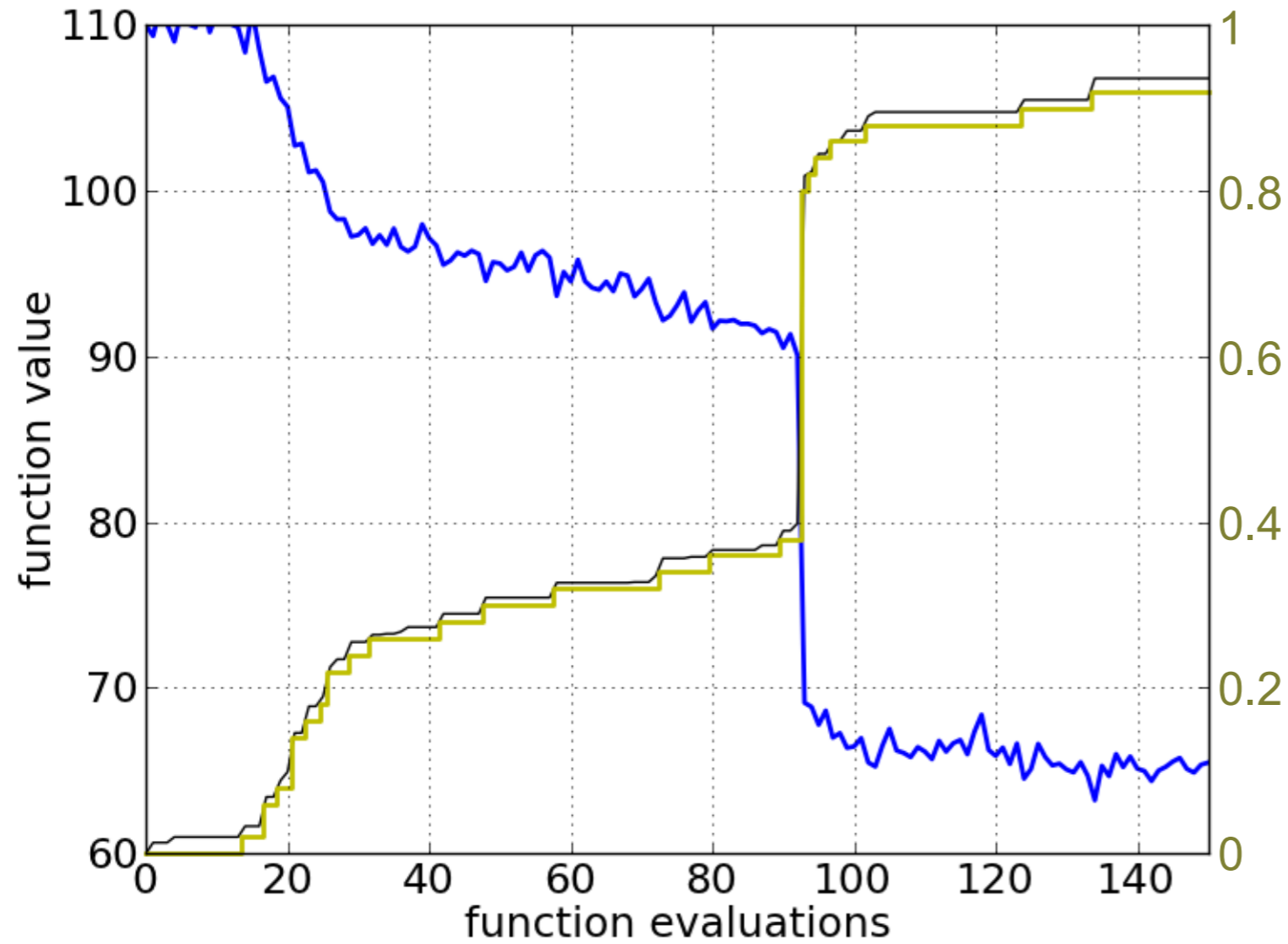




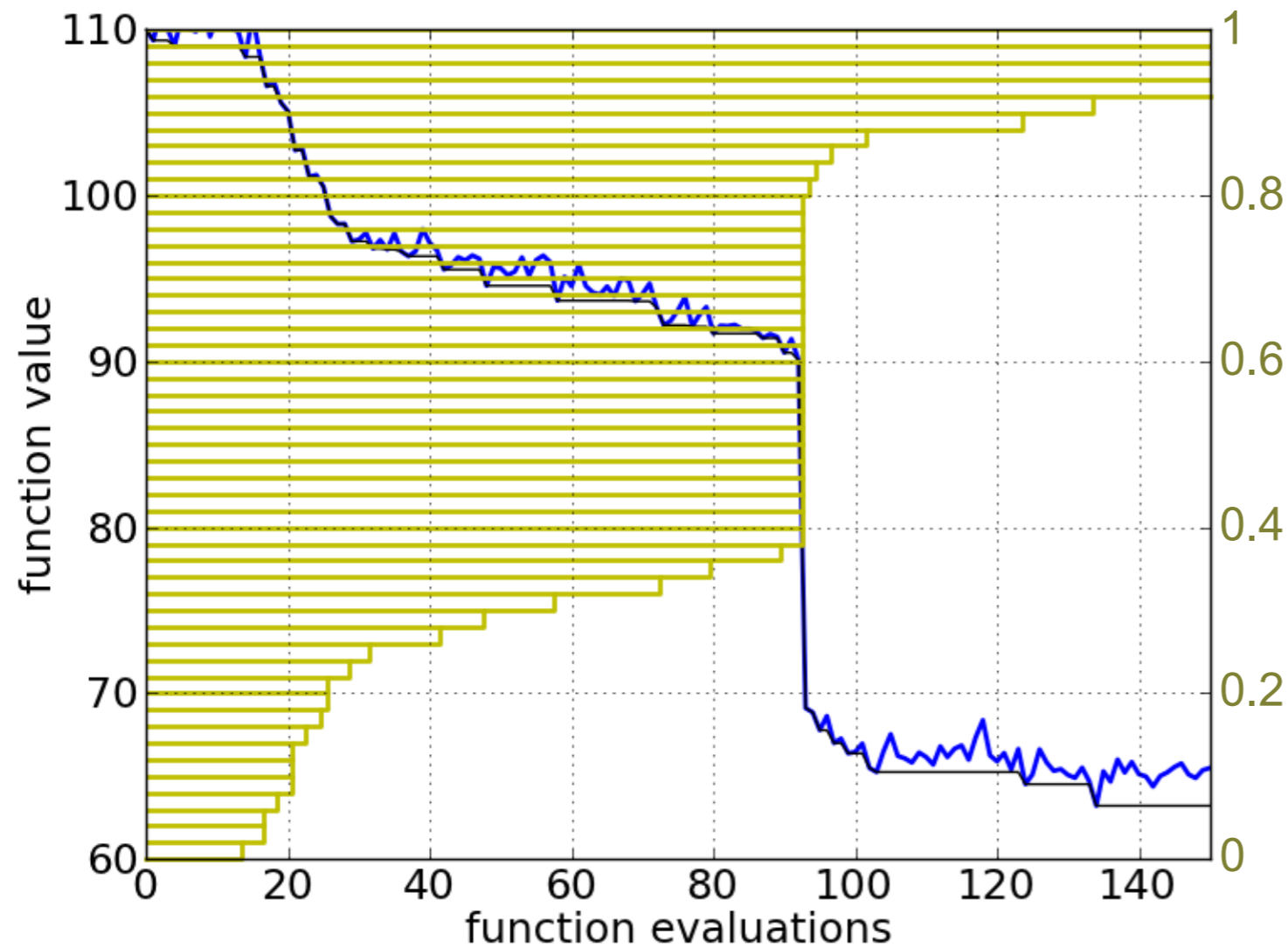
the ECDF recovers  
the monotonous  
graph



the ECDF recovers  
the monotonous  
graph, discretised  
and flipped



the ECDF recovers  
the monotonous  
graph, discretised  
and flipped



the ECDF recovers  
the monotonous  
graph, discretised  
and flipped

the **area over the  
ECDF** curve is the  
**average runtime**  
(the geometric  
average if the x-axis  
is in log scale)

# Data and Performance Profiles

- so-called *Data Profiles* (Moré and Wild 2009) are empirical distributions of runtimes [# evaluations] to achieve a given single target  
usually divided by dimension + 1
- so-called *Performance profiles* (Dolan and Moré 2002) are empirical distributions of *relative* runtimes [# evaluations] to achieve a given single target  
normalized by the runtime of the fastest algorithm on the respective problem

# Benchmarking with COCO

## COCO — Comparing Continuous Optimisers

- is a (software) platform for comparing continuous optimisers in a black-box scenario

<https://github.com/numbbo/coco>

- *automatises* the tedious and repetitive task of *benchmarking numerical optimisation algorithms in a black-box setting*
- advantage: saves time and **prevents** common (and not so common) **pitfalls**

## COCO provides

- experimental and measurement *methodology*  
main decision: what is the end point of measurement
- suites of benchmark functions  
single objective, bi-objective, noisy, constrained (in beta stage)
- **data** of already benchmarked algorithms **to compare with**



# COCO: Installation and Benchmarking in Python

```
$ ### 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 post-processing :-)
```

```
import os, webbrowser
from scipy.optimize import fmin
import cocoex, cocopp

# prepare
output_folder = "scipy-optimize-fmin"
suite = cocoex.Suite("bbob", "", "")
observer = cocoex.Observer("bbob", "result_folder: " + output_folder)

# run benchmarking
for problem in suite: # this loop will take several minutes
    observer.observe(problem) # generates the data for cocopp post-processing
    fmin(problem, problem.initial_solution)

# post-process and show data
cocopp.main(observer.result_folder) # re-run folders look like "...-001" etc
webbrowser.open("file://" + os.getcwd() + "/ppdata/index.html")
```

# Benchmark Functions

should be

- comprehensible
- **difficult to defeat** by “cheating”  
examples: optimum in zero, separable
- scalable with the input dimension
- **reasonably quick to evaluate**  
e.g. 12-36h for one full experiment
- **reflect reality**  
specifically, we model **well-identified difficulties**  
encountered also in real-world problems

# The COCO Benchmarking Methodology

- budget-free

larger budget means more data to investigate  
any budget is comparable  
termination and restarts are or become relevant

- using runtime as (almost) single performance measure  
measured in [number of function evaluations](#)

- runtimes are aggregated

- in empirical (cumulative) distribution functions

- by taking averages

*geometric average* when aggregating over different problems

# Benchmarking Results for Algorithm ALG on the bbob Suite

[Home](#)

[Runtime distributions \(ECDFs\) per function](#)

[Runtime distributions \(ECDFs\) summary and function groups](#)

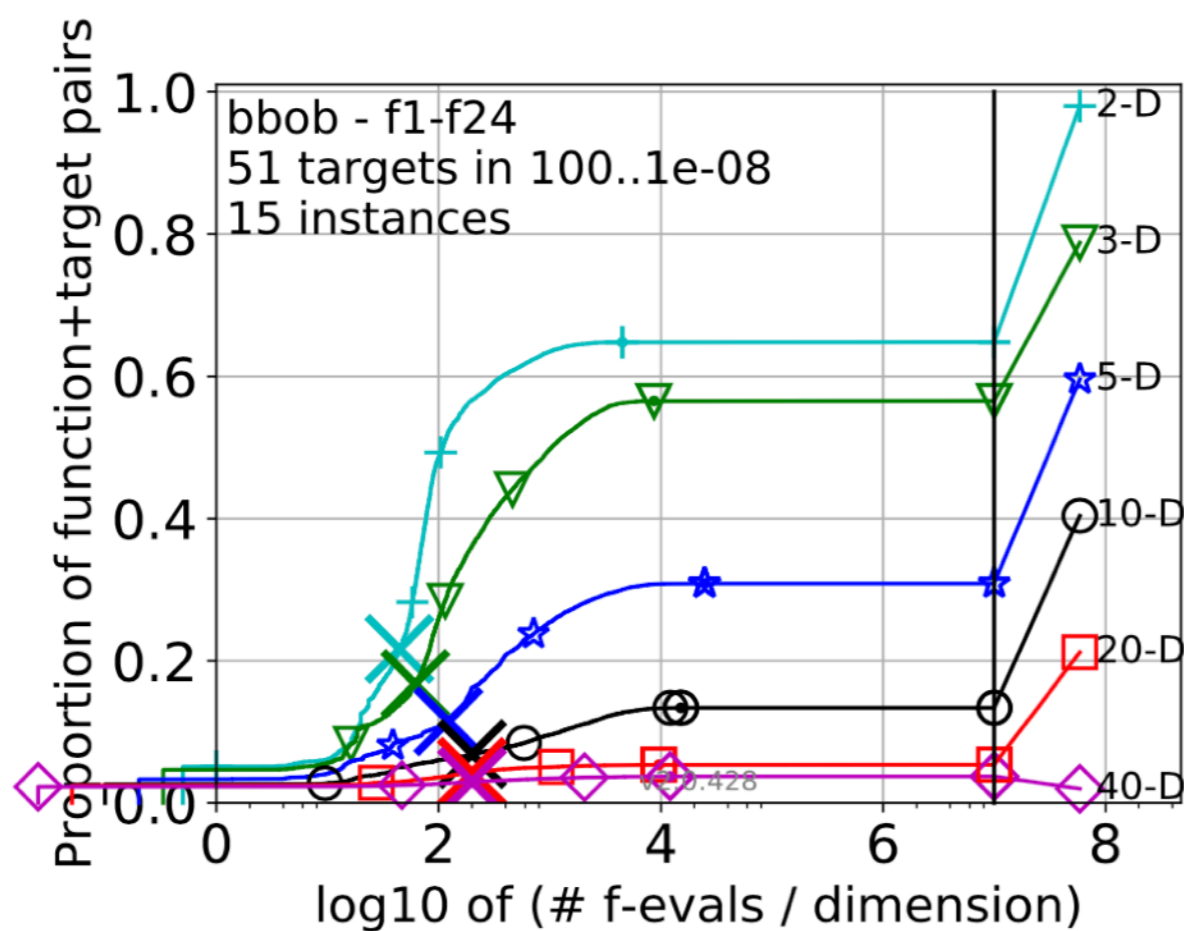
[Scaling with dimension for selected targets](#)

[Tables for selected targets](#)

[Runtime distribution for selected targets and f-distributions](#)

[Runtime loss ratios](#)

## Runtime distributions (ECDFs) over all targets



# Using Theory

*“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, young gentlemen, it is my earnest advice to respect the facts.”*

*— Igor Sikorsky, airplane and helicopter designer*

# Using Theory in Experimentation

- shape our expectations and objectives
- debugging / consistency checks
  - theory may tell us what we *expect* to see
- knowing the limits (optimal bounds)
  - for example, we cannot converge faster than optimal
  - trying to improve is a waste of time
- utilize **invariance**
  - it may be possible to design a much simpler experiment and get to the same or stronger conclusion by invariance considerations
  - change of coordinate system is a powerful tool

**FIN**