A Practical Guide to Experimentation (and Benchmarking)

Nikolaus Hansen
Inria
Research Centre Saclay, CMAP, Ecole polytechnique

Overview

- Scientific experimentation
- Invariance
- Display results
- Statistical analysis
- Performance assessment
  - What to measure
  - How to display
  - Aggregation
  - Empirical distributions
- Benchmarking with COCO
- Using theory
- Approaching an unknown problem
- A practical experimentation session

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 of 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

Do not hesitate to ask questions!
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
  - Interpreted/scripted languages have an advantage (quick test of code snippets)
- **practice to make predictions of the (possible/expected) 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) to develop 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 the data, study them carefully!
- **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 decisions
- **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**

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) to develop 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 the data, study them carefully!
- **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 decisions
- **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 \( x_i \mapsto -x_i + 1 \)

\[ x_i \mapsto -x_i + 1 \]

\[ \text{this transformation or the identity are the coding choice in each variable} \]

\[ \text{in continuous domain: norm-preserving (isotropic, "rigid") transformation} \]

- does not change the function “structure”
- all level sets \( \{ x | f(x) = \text{const} \} \) 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)
- may affect the neighbourhood depending on the operators (recombination, mutation)

\[ f = h \]

\[ f = g_1 \circ h \]

\[ f = g_2 \circ h \]

A permutation introduces structure that we may want to exploit

\[ \text{even at the cost of abandoning invariance} \]

Instead of 1 function, we now consider \( n! \) different but equivalent functions

\( n! \gg 2^n \), that is, the number of permutations is even (much) larger than the size of the search space

Invariance Under Order Preserving Transformations

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”

for example, invariance under search space rotation

(separable vs non-separable)
Invariance Under Rigid Search Space Transformations

\[ f = h \circ R \circ h^{-1} \]

\[ f = h \circ R \]

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
- 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.

Overview

- Scientific experimentation
- Invariance
  - Display results
    - Statistical analysis
    - Performance assessment
      - What to measure
      - How to display
      - Aggregation
      - Empirical distributions
    - Benchmarking with COCO
    - Using theory
    - Approaching an unknown problem
    - A practical experimentation session

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 uncommon picture)

why not, what's wrong with it?

better like this (shown are the same data),

caution: fails with negative f-values

Displaying 51 Runs

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

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

even better like this: subtract minimum value over all runs
There is more to display than convergence graphs

Aggregation: Which Statistics?

- to reliably estimate an expectation (from the average) we need to make assumptions on the tail of the underlying distribution
- these can not be implied from the observed data
- AKA: the average is well-known to be (highly) sensitive to outliers (extreme events)
- rare events can only be analyzed by collecting a large enough number of data

mean/average function value

- tends to emphasize large values
Aggregation: Which Statistics?

- geometric average function value $\exp(\text{mean}(\log(f_i)))$
  - reflects "visual" average
  - depends on offset

- average iterations
  - reflects "visual" average
  - here: incomplete

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
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"

Scientific experimentation
Invariance
Display results

Statistical analysis
Performance assessment
What to measure
How to display
Aggregation
Empirical distributions

Benchmarking with COCO
Using theory
Approaching an unknown problem

What to measure
How to display
Aggregation
Empirical distributions

Benchmarking with COCO
Using theory
Approaching an unknown problem

A practical experimentation session

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 scientists. 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, 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?

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: 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
  - 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 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
- given a found/observed p-value, fewer data are better
  - not assuming that the data come from a parametrised family of probability distributions
- 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 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 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
- given a found/observed p-value, fewer data are better
  - not assuming that the data come from a parametrised family of probability distributions
- 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

- **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.ranksums(..., alternative='two-sided')`

- **Null hypothesis** (nothing relevant is observed if):
  $P(X < Y) = P(Y < X)$
  $H_0$: 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 the sum of ranks in the ranking of all (combined) data values
  $\text{Alg1} = [400, 422, 440]$ vs $\text{Alg2} = [444, 490, 555] \implies$ 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 Significance: How many data do we need?

**AKA as test efficiency**

$P_{\text{min}} = \frac{2 \prod_{i=1}^{n_1} i}{i + n_2}$

*assumption*: data are fully "separated", that is,
$Y_i, j: x_i < y_j$ or $Y_i, j: x_i > y_j$ (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: multiplying the computed $p$-value by the number of tests

- 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?

- **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.ranksums(..., alternative='two-sided')`

- **Null hypothesis** (nothing relevant is observed if):
  $P(X < Y) = P(Y < X)$
  $H_0$: 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 the sum of ranks in the ranking of all (combined) data values
  $\text{Alg1} = [400, 422, 440]$ vs $\text{Alg2} = [444, 490, 555] \implies$ 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

---

Rank Sum Test

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

\[
\begin{align*}
\lambda &= 0.997, 1.018 \\
\Delta \text{mean} &= 0.834 \\
\Delta \text{median} &= 0.844 \\
\eta_{<\text{median}}(\lambda) &= 51.6\% \\
\eta_{>\text{median}}(\lambda) &= 51.9\%
\end{align*}
\]

Rare Events

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

\[
\begin{align*}
(1 - p)^{\# \text{samples}} \approx \exp(-1) \approx 1/3 \\
(1 - p)^{\# \text{samples}} \approx \exp(-3) \approx 0.05
\end{align*}
\]

Testing Frequencies

```
import scipy.stats
p1, n1 = 0, 90  # 10% success
p2, n2 = 20, 100  # 20% success
scipy.stats.chi2_contingency([[p1, n1 - p1], [p2, n2 - p2]])
```

\[(2.930076628352492, 0.869433657247775, \chi^2\text{-statistics}, 0.0, 0.0, \text{p-value})\]

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

"... experimental results lacking proper statistical analysis must be considered anecdotal at best, or even wholly inaccurate." — M. Wineberg, 2016

Statistical Analysis

Do we (even) need a statistical analysis?
Overview

- Scientific experimentation
- Invariance
- Display results
- Statistical analysis
- Performance assessment
  - What to measure
  - How to display
  - Aggregation
  - Empirical distributions
- Benchmarking with COCO
- Using theory
- Approaching an unknown problem
- A practical experimentation session

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
do not display (only) tabulated numbers
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 find where something needs to be understood non-standard experimentation is often additionally necessary

Aggregation: Fixed Budget vs Fixed Target

- for aggregation (and algorithm comparison) 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
Measures for Performance Assessment

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"

- 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

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 (and algorithm comparison) 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

- for aggregation (and algorithm comparison) 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?

\[ p_x(\text{Algo A}) < 1, \text{fast convergence} \]
\[ p_x(\text{Algo B}) \approx 1, \text{slow convergence} \]

Consider artificial (AKA simulated) restarts using the given independent runs.

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

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

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

\[
= \frac{\text{odds ratio}}{N_{\text{unsucc}} N_{\text{succe}}} \times \text{avg}(\text{evals}_{\text{unsucc}})
\]

\[
\approx \frac{\text{avg}(\text{evals}_{\text{succe}}) + \frac{N_{\text{unsucc}}}{N_{\text{succe}}} \times \text{avg}(\text{evals}_{\text{succe}})}{N_{\text{succe}}}
\]

\[
= \frac{N_{\text{unsucc}}}{N_{\text{succe}}} \times \text{avg}(\text{evals}_{\text{succe}})
\]

\[
= \frac{1}{\text{success rate}} \times \text{avg}(\text{evals}_{\text{succe}})
\]

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, \textit{empirical distributions}) are arguably the \textit{single most powerful tool} to “aggregate” data in a display.
- 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
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 [# f-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

Overview

- Scientific experimentation
- Invariance
- Display results
- Statistical analysis
- Performance assessment
  - What to measure
  - How to display
  - Aggregation
  - Empirical distributions
- Benchmarking with COCO
  - Using theory
  - Approaching an unknown problem
  - A practical experimentation session
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, large-scale, mixed-integer, constrained (in beta stage)

- data of already benchmarked algorithms to compare with

COCO: Installation and Benchmarking in Python

```python
from scipy.optimize import fnin
import cocoox, coccop

# prepare
output_folder = "scipy-optimize-fmin"
suite = cocoox.Suite(['bbo', '', ''])
observer = cocoox.Observer('bbo', 'result_folder', *output_folder)

# run benchmarking
for problem in suite:
    observer.observe(problem) # generates the data for cocoox post-processing

# post-process and show data
 cocoox.main(observer.result_folder) # re-run folders look like '....-001' etc
```

Benchmark Functions

should (ideally) 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, in COCO 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

- uses 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

---

Overview

- Scientific experimentation
- Invariance
- Display results
- Statistical analysis
- Performance assessment
  - What to measure
  - How to display
  - Aggregation
  - Empirical distributions
- Benchmarking with COCO
  - Using theory
  - Approaching an unknown problem
  - A practical experimentation session

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
- knowing the limits (optimal bounds)
- utilize invariance
  - theory may tell us what we expect to see
  - for example, we cannot converge faster than optimal
  - 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

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".
- Create sections plots (1 vs x on a line)
  - one-dimensional grid search is cheap
- 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

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
import numpy

Populating the interactive namespace from numpy and matplotlib
```


Demonstrations

- A somewhat typical working mode
- A parameter investigation

FIN
Jupyter IPython notebook

Questions?