BFO, a trainable and versatile Brute Force Optimizer

Philippe Toint (with Margherita Porcelli)

Namur Center for Complex Systems (naXys), University of Namur, Belgium

(philippe.toint@unamur.be)

Oliver Smithies and Leverhume Lecture II, November 2015

Thanks

- Leverhulme Trust, UK
- Balliol College, Oxford
- Belgian Fund for Scientific Research (FNRS)
- University of Namur, Belgium

How it happened...

- working on an interpolation-based derivative free optimizer for

$$\min_{x \text{ subject to bounds}} f(x)$$

(more on that at the very end)

- needed something quick and dirty to improve its parameter settings
- wrote a "Brute Force" tool ...
- ... which (after some years of tweaking) has turned into BFO

simple ideas + computing power $\stackrel{?}{\Longrightarrow}$ robust/useful tool?

The context

Two common preoccupations in algorithm design/usage:

- For algorithms designers:

  > How to tune the parameters of an algorithm in order to ensure the best possible performance on the *largest possible class* of applications?

- For algorithm/code users:

  > How to tune the parameters of a code in order to ensure the best possible performance on a *specialized class* of applications?

Does achieving the first does help the second?

A way out ?

Some flexibility is needed !

- Provide a tuning methodology which is applicable to many algorithms
- Provide code which allows user-tuning for his/her pet problem class

$\Longrightarrow$ optimization?

- Need to define an objective function
  (how to measure algorithm performance in this context?)
- Need to define the constraints (on algorithmic parameters)
  - simple bounds (algorithm dependent)
  - continuous/integer/categorical variables + mix
    (ex: blocking size, model type, . . .)

Which objective function?

Assume that the (negative) performance perf(params, prob) can be measured by running the considered algorithm with parameters params on problem prob (ex: number of function evaluations).

- First model: optimize the total/average performance (AO, OPAL):

$$\min_{\text{params}} \sum_{\text{problems}} \text{perf}(\text{params}, \text{prob})$$

- Second model: optimize the robust performance (RO):

$$\min_{\text{params}} \max_{\text{perturbed params}} \sum_{\text{problems}} \text{perf}(\text{perturbed params}, \text{prob})$$

where

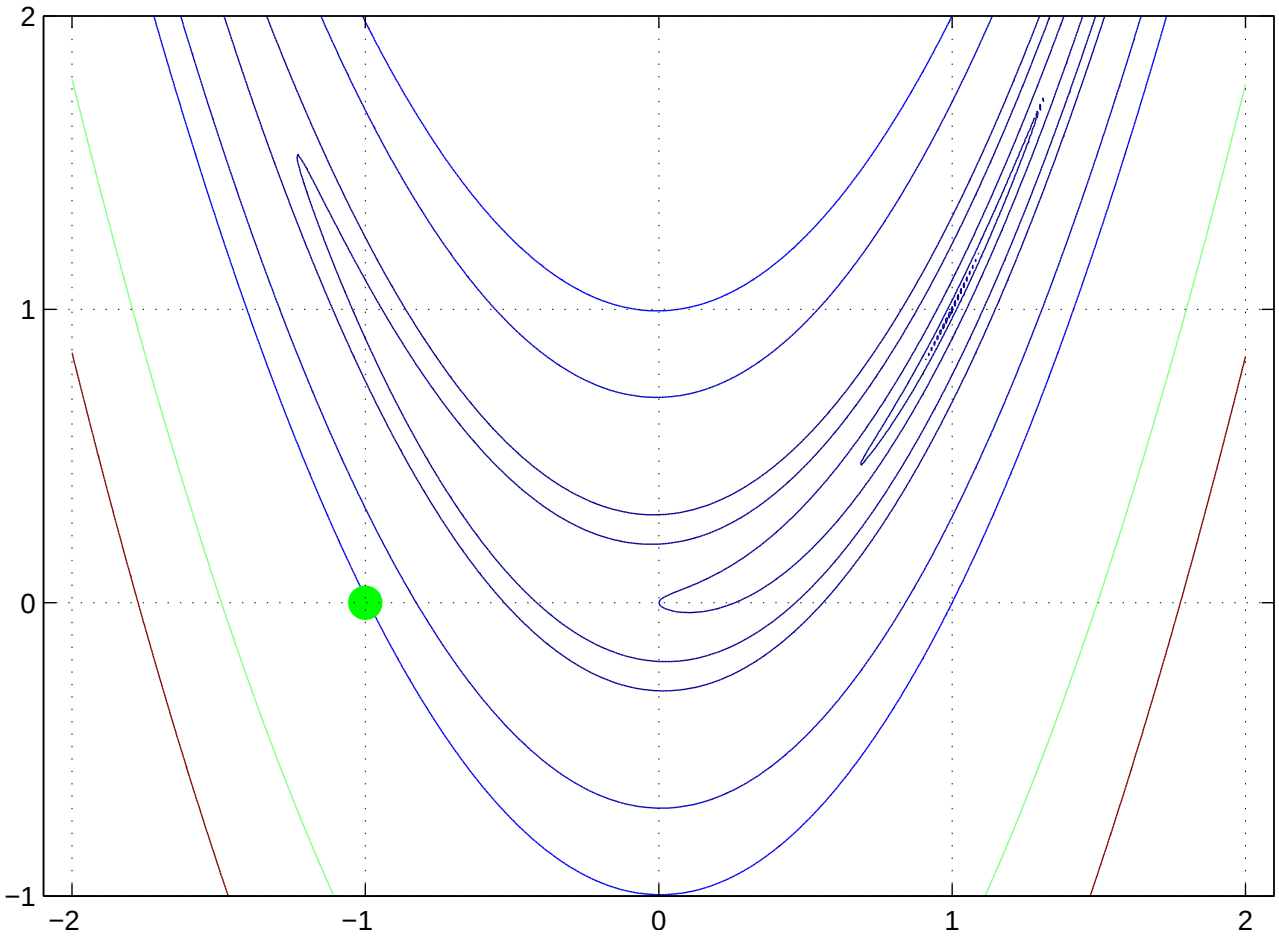
$$0.95 * \text{params} \leq \text{perturbed params} \leq 1.05 * \text{params}$$

A new tool: BFO (the Brute Force Optimizer)

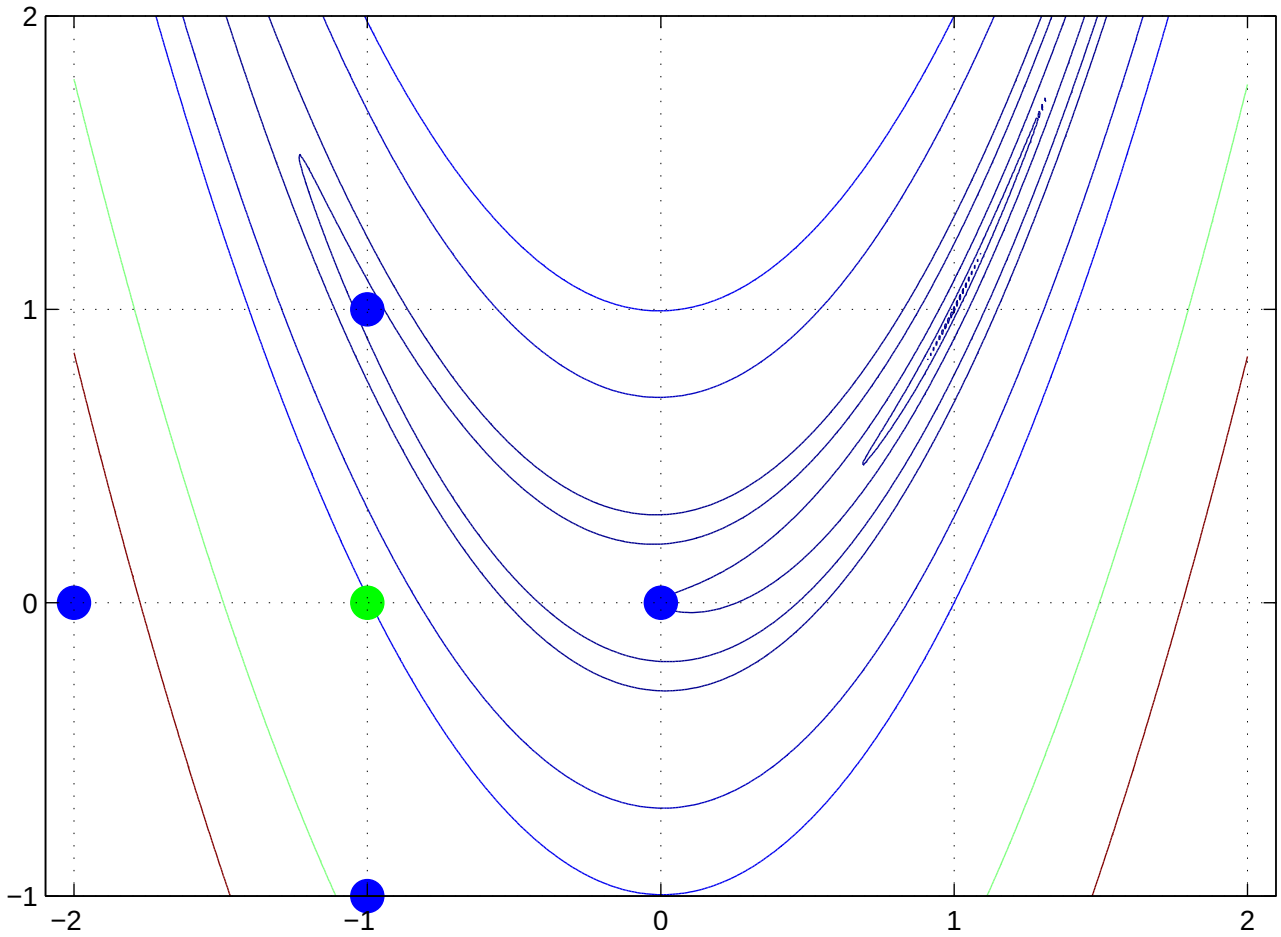
BFO: a new *local* optimization package with

- randomized pattern search methodology
  (does not require continuity of the objective function)
- allows bounds on the variables
- allows continuous/discrete or mixed integer variables
- handles multilevel/equilibrium problems
  (needed for the robust tuning strategy)
- includes self-tuning facilities

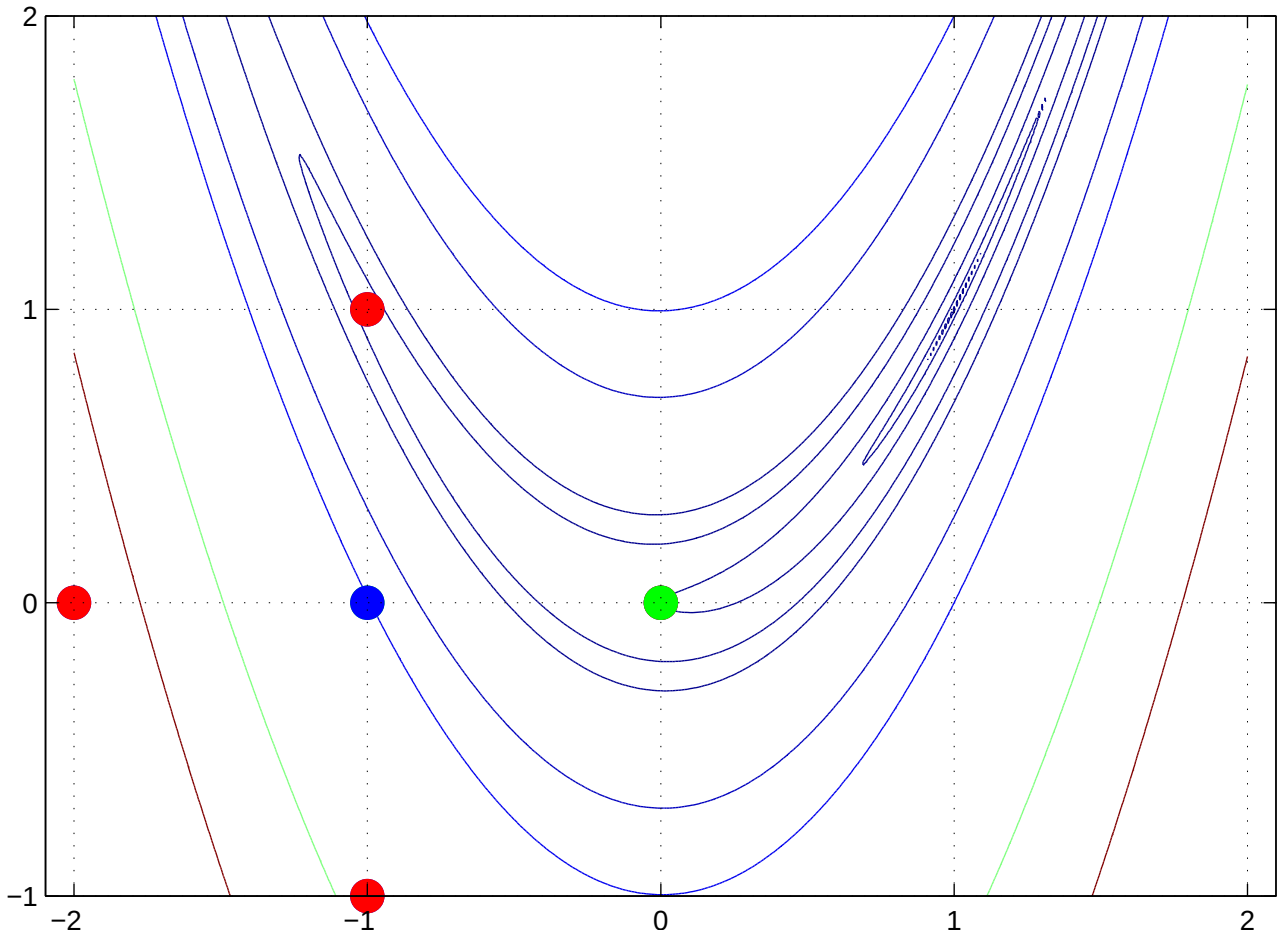
The standard compass-search on Rosenbrock's function



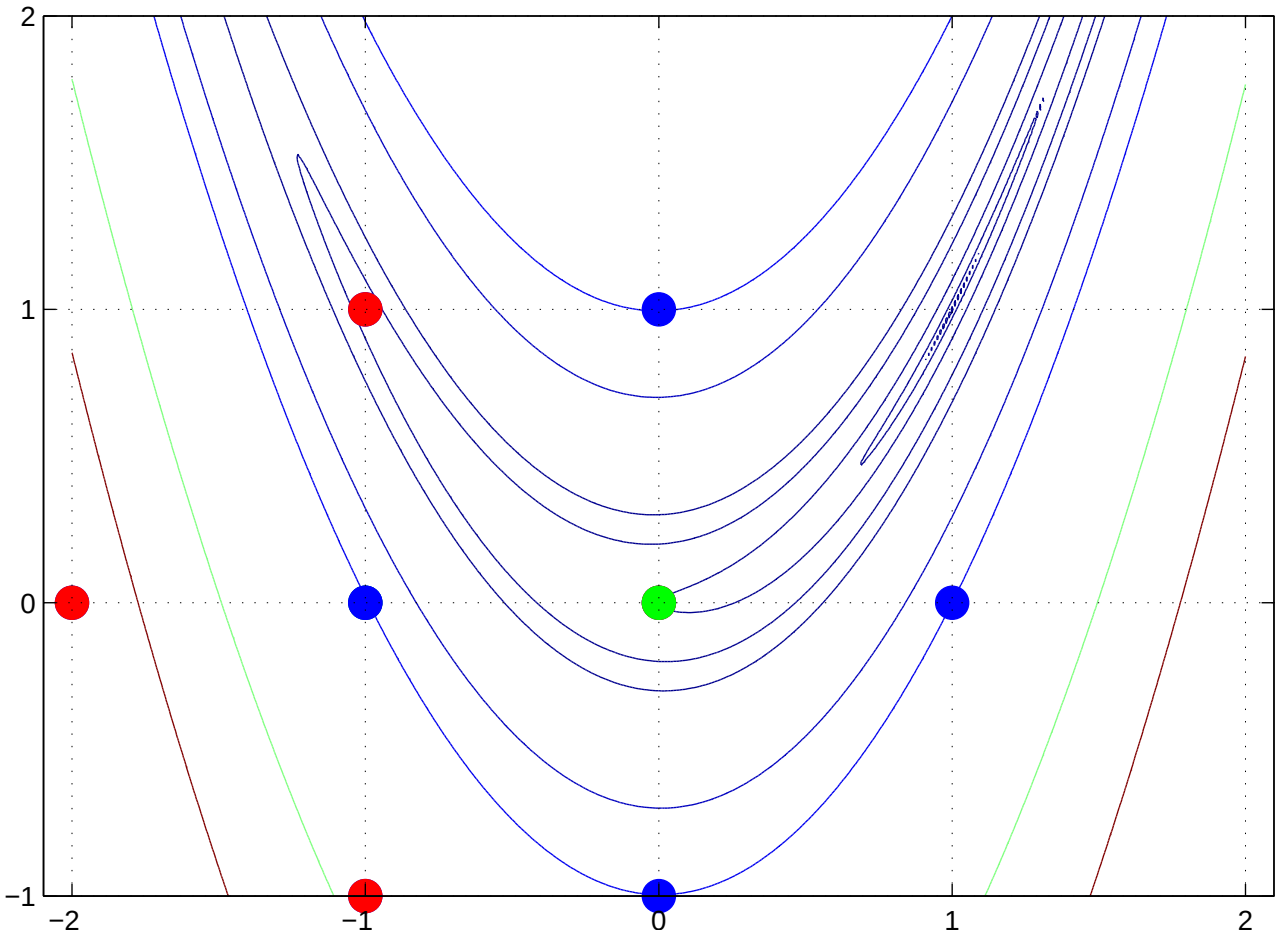
The standard compass-search on Rosenbrock's function



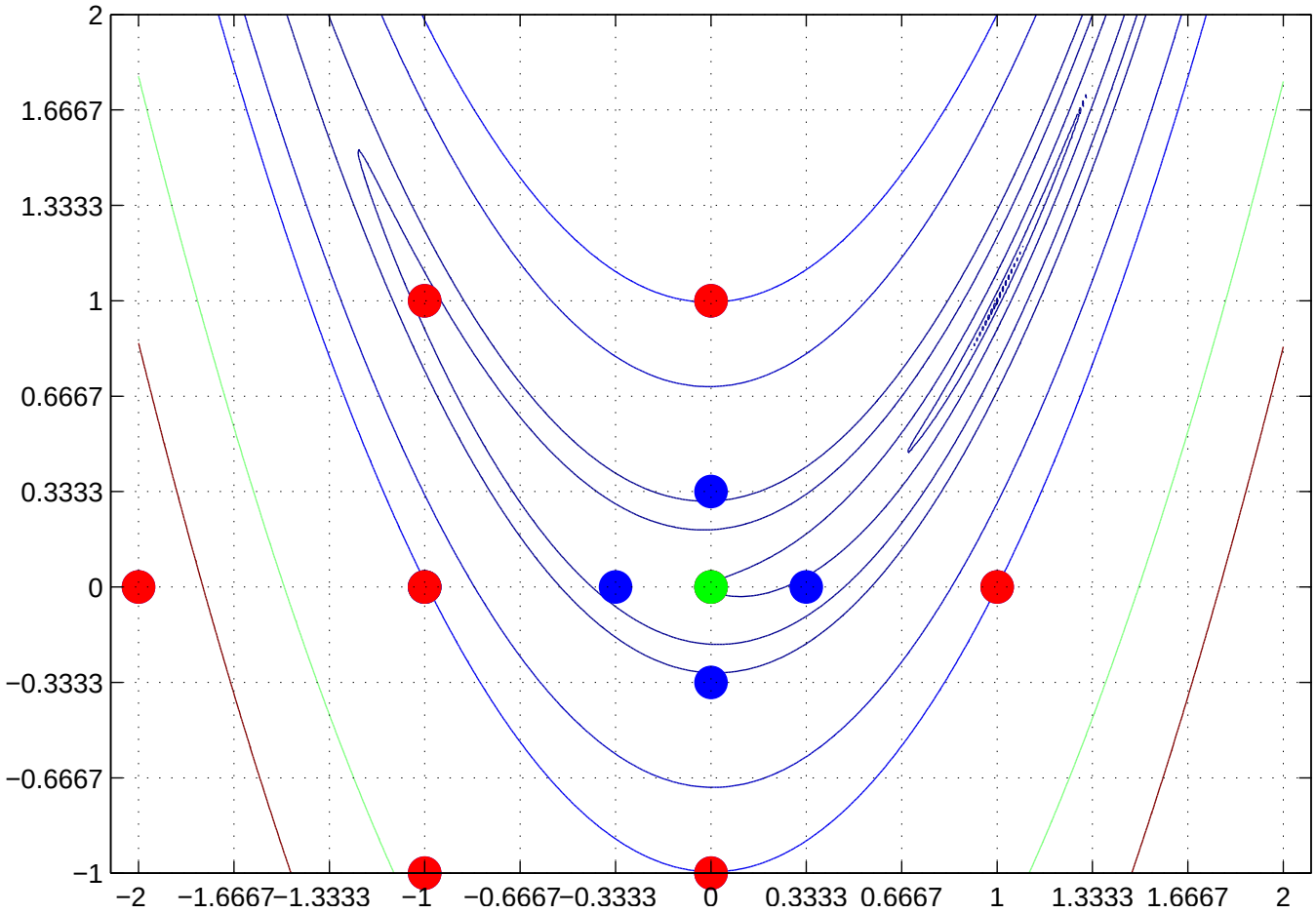
The standard compass-search on Rosenbrock's function



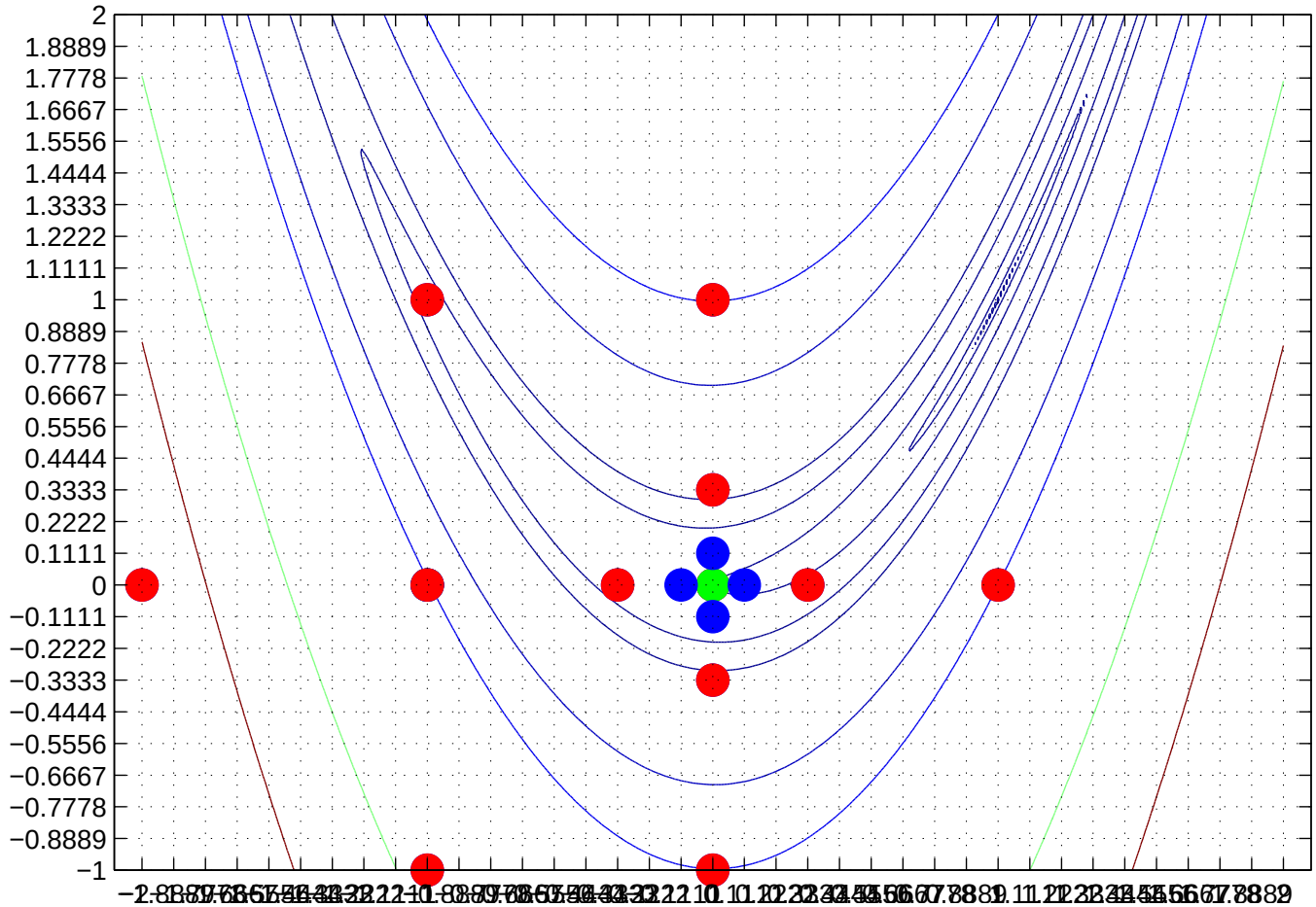
The standard compass-search on Rosenbrock's function



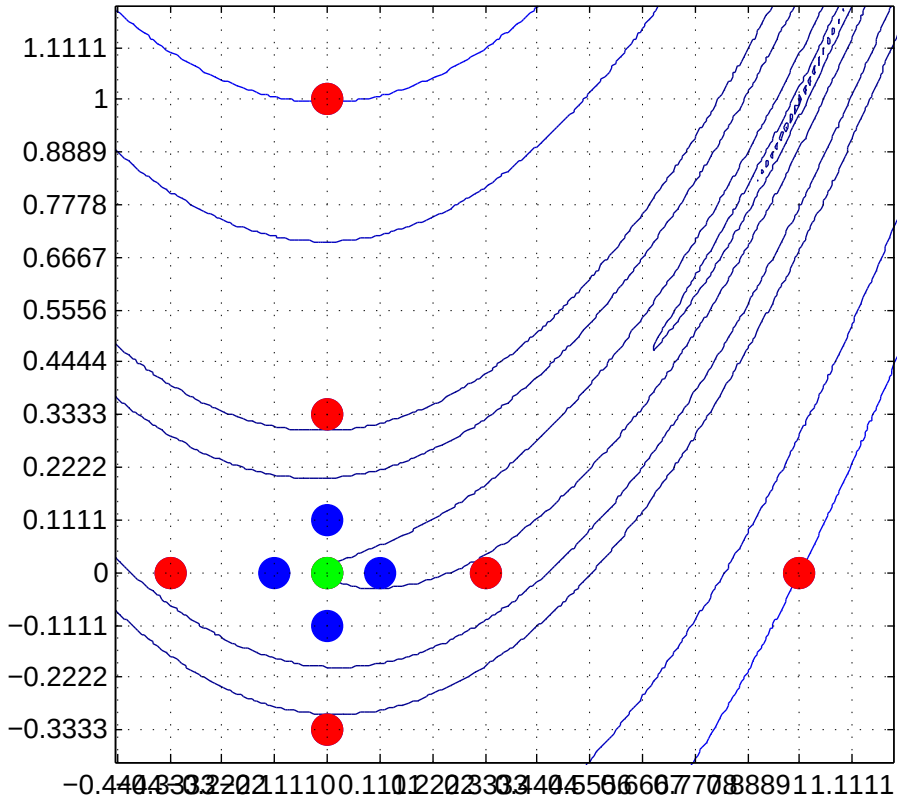
The standard compass-search on Rosenbrock's function



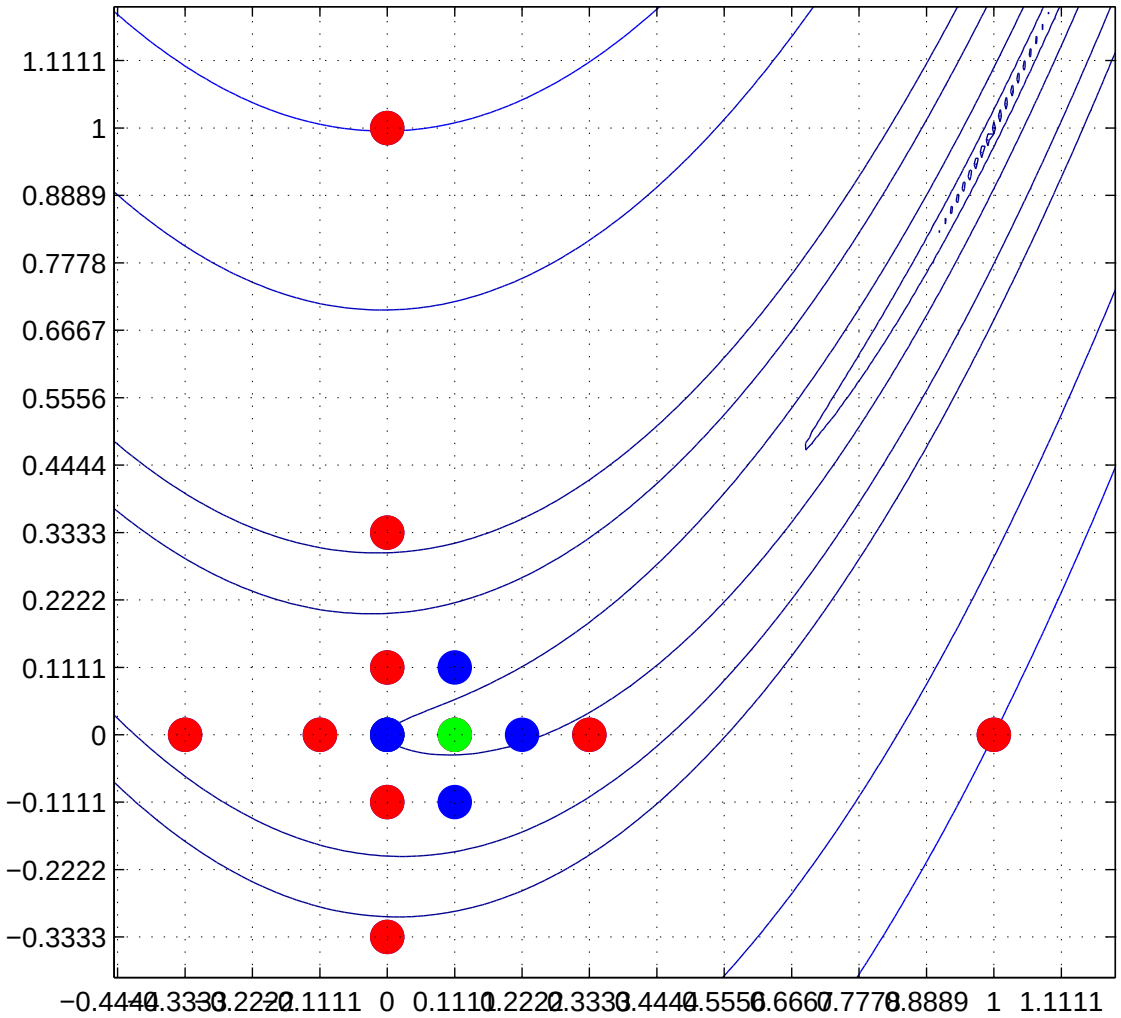
The standard compass-search on Rosenbrock's function



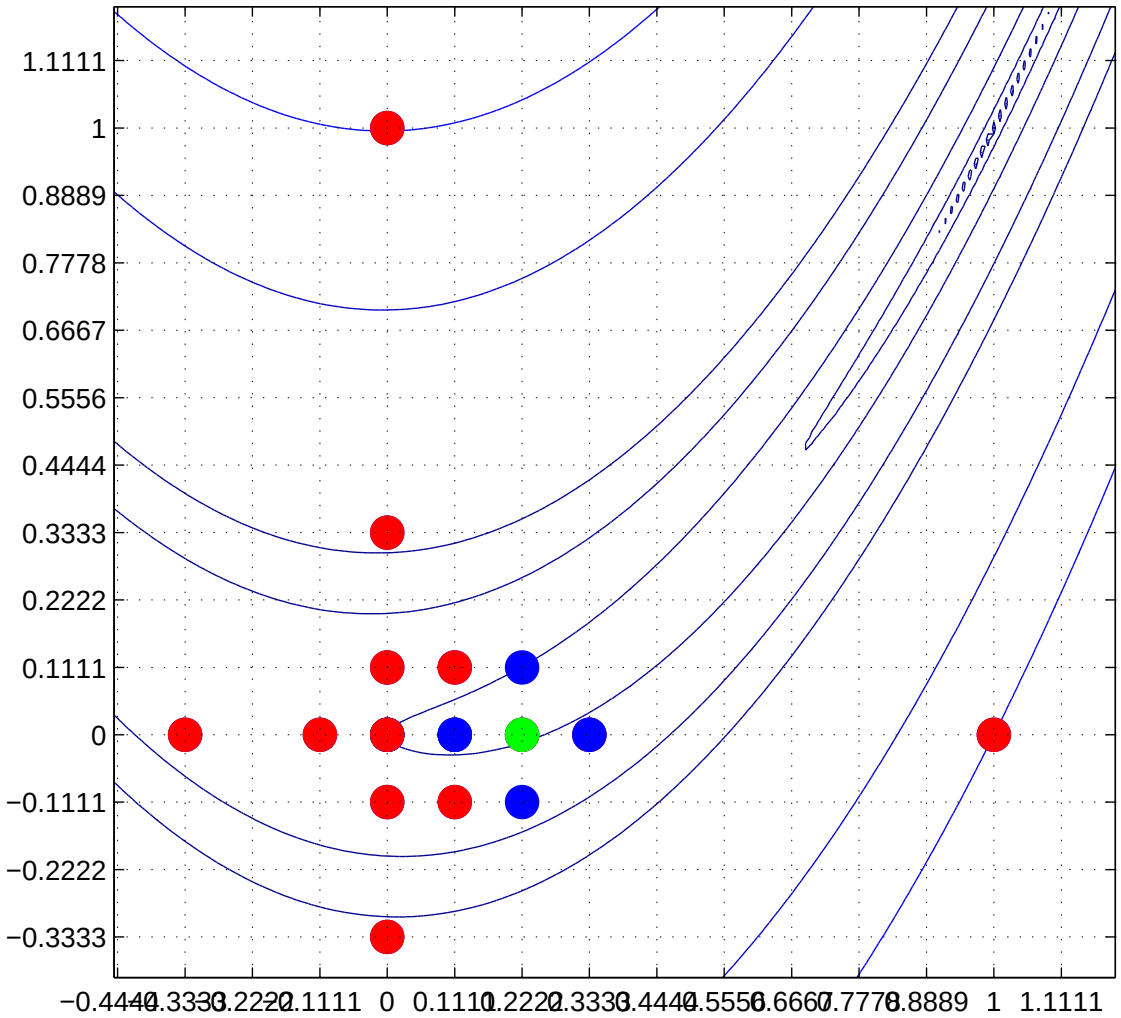
The standard compass-search on Rosenbrock's function



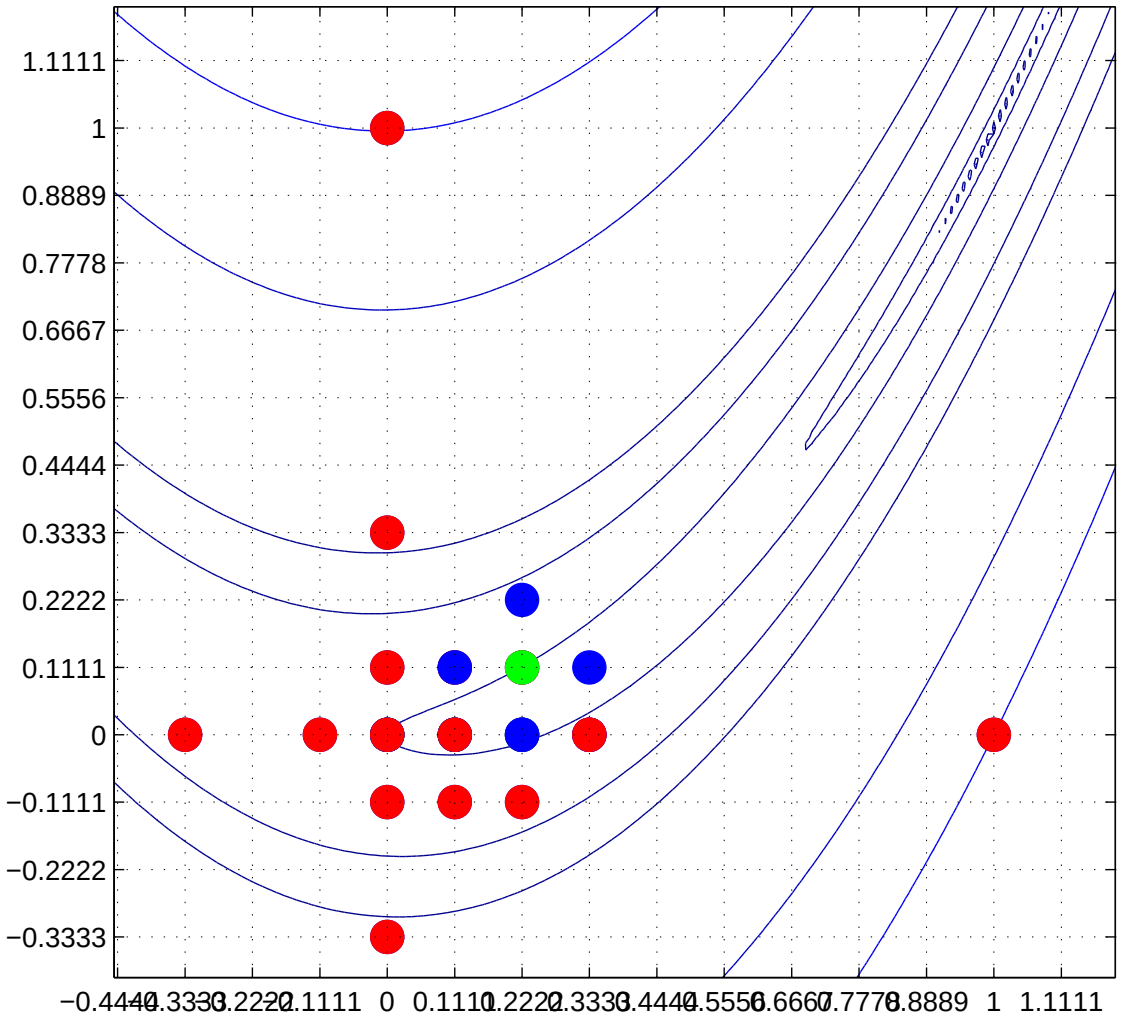
The standard compass-search on Rosenbrock's function



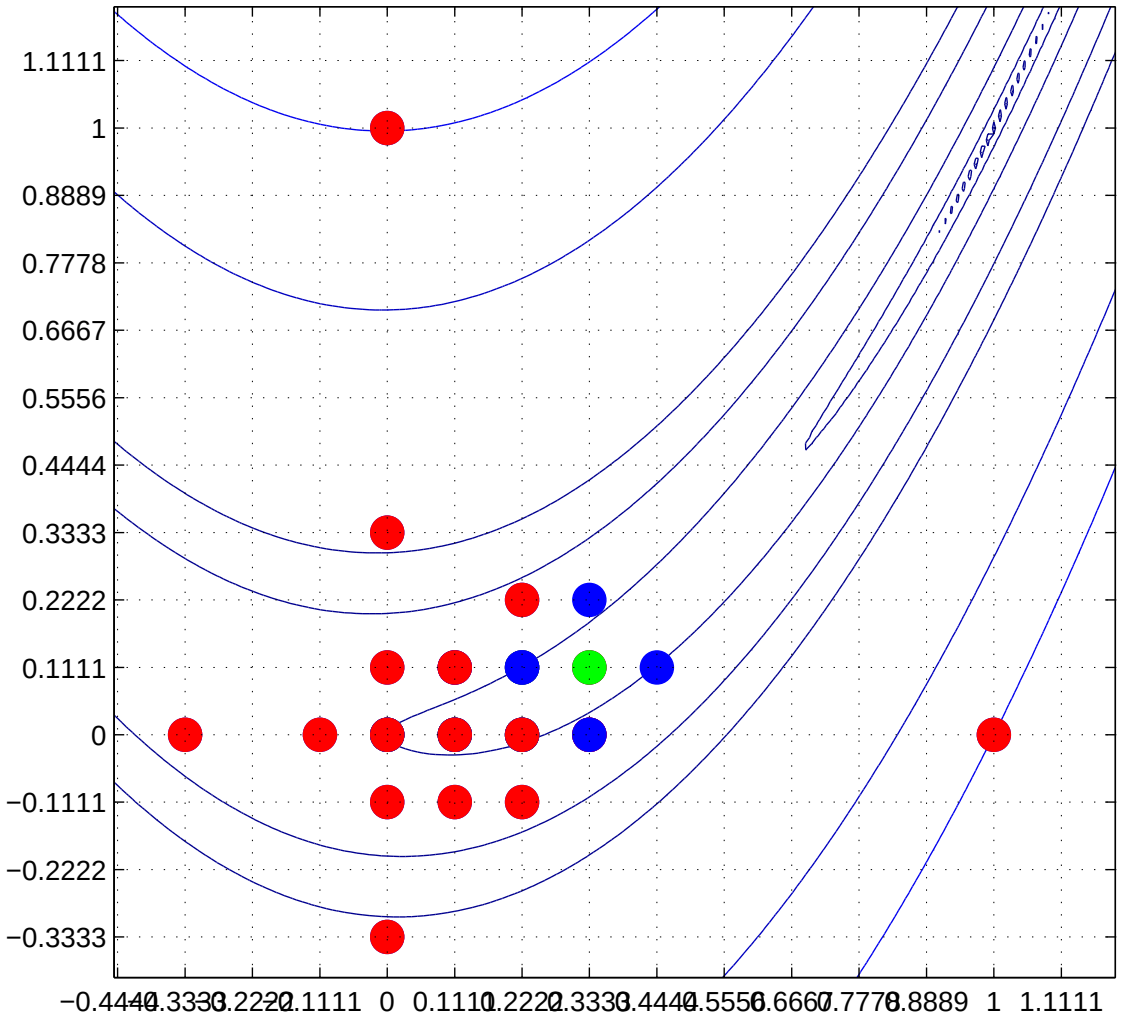
The standard compass-search on Rosenbrock's function



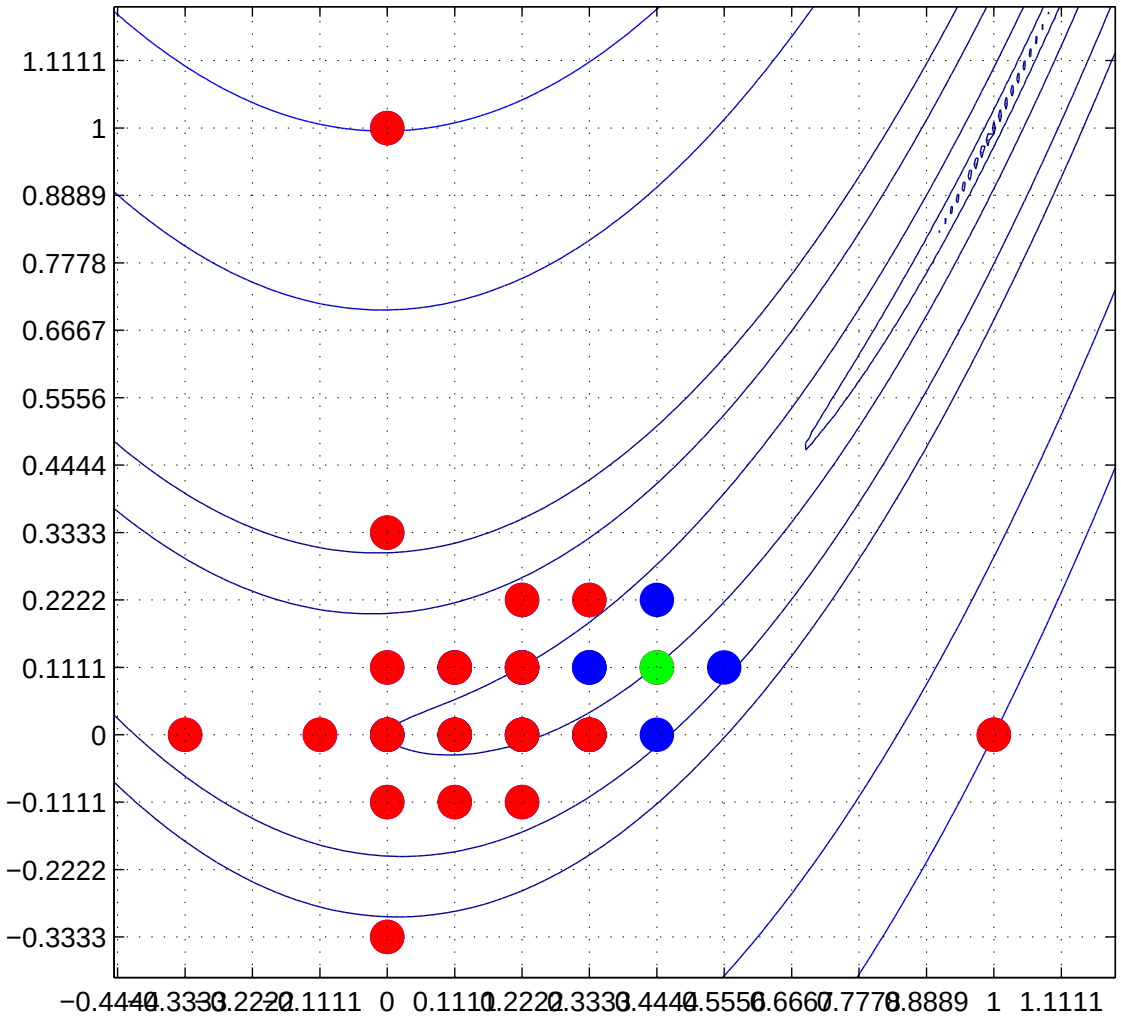
The standard compass-search on Rosenbrock's function



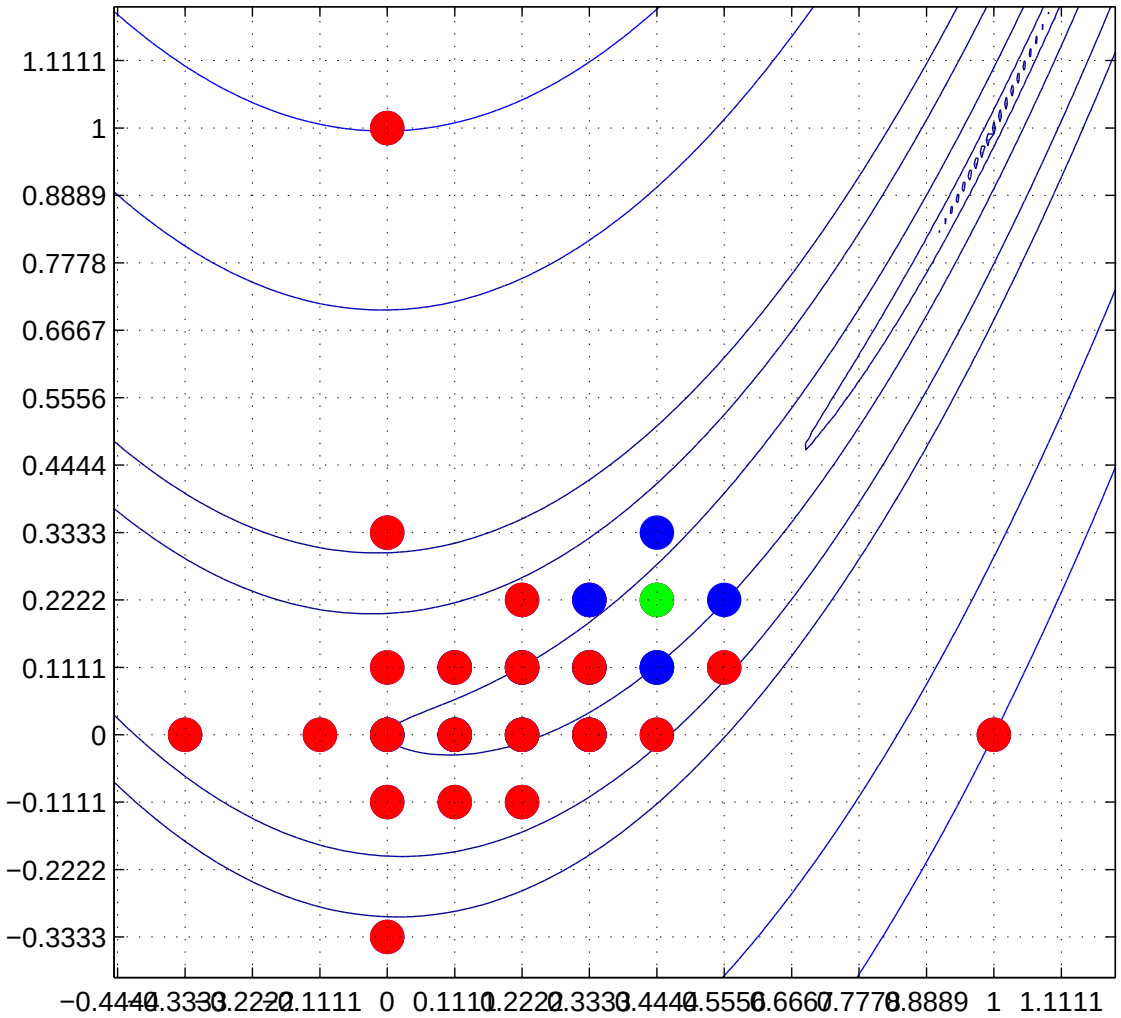
The standard compass-search on Rosenbrock's function



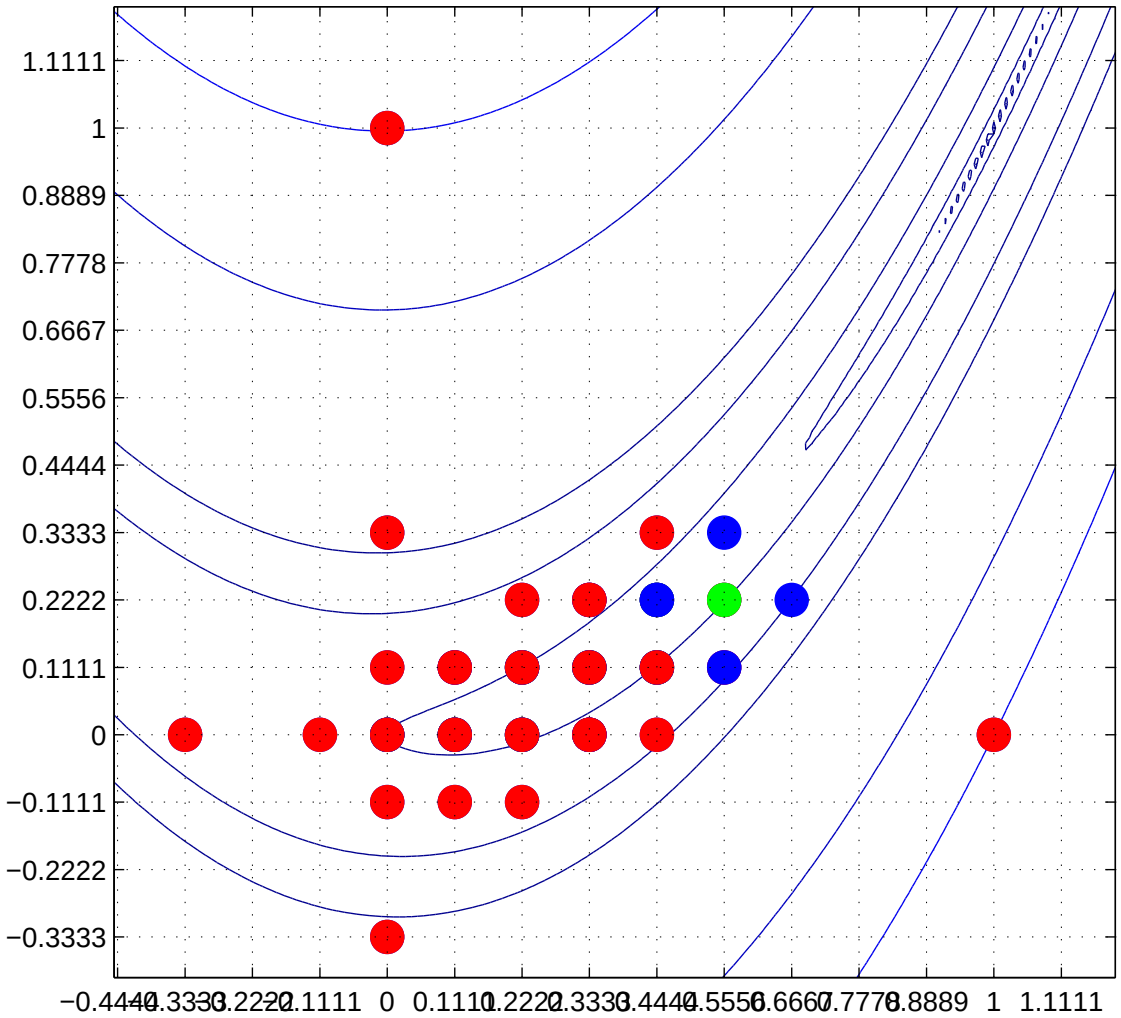
The standard compass-search on Rosenbrock's function



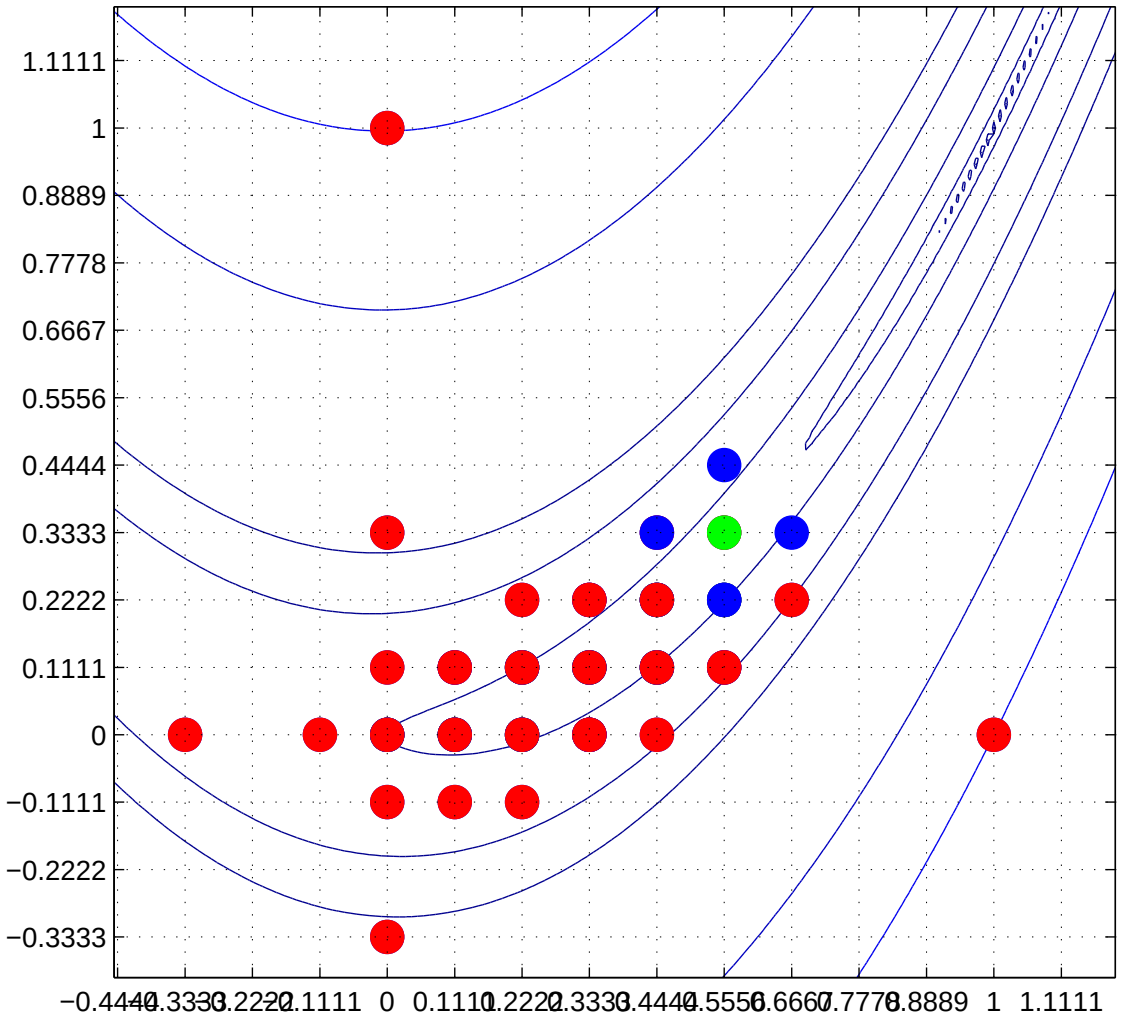
The standard compass-search on Rosenbrock's function



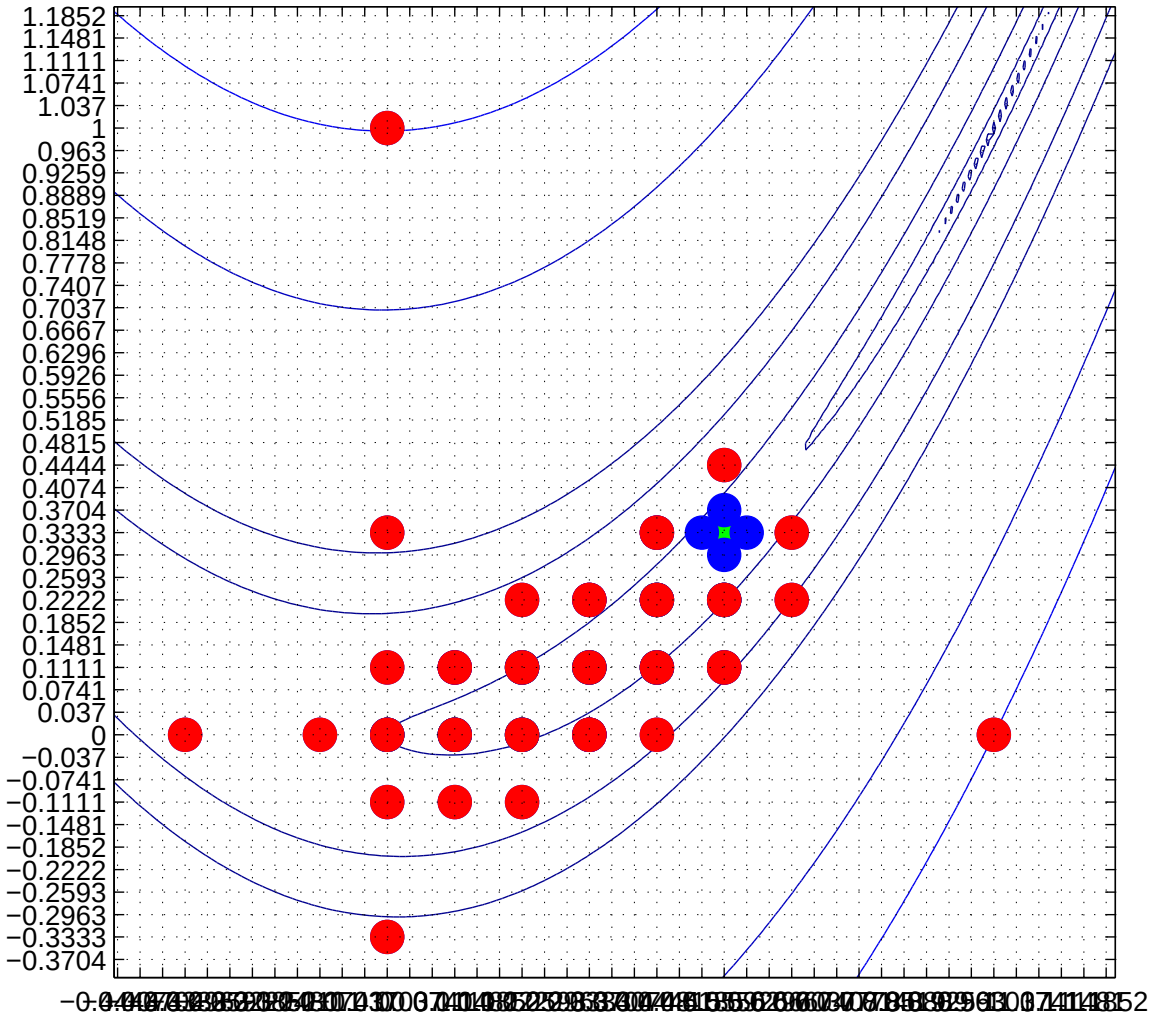
The standard compass-search on Rosenbrock's function



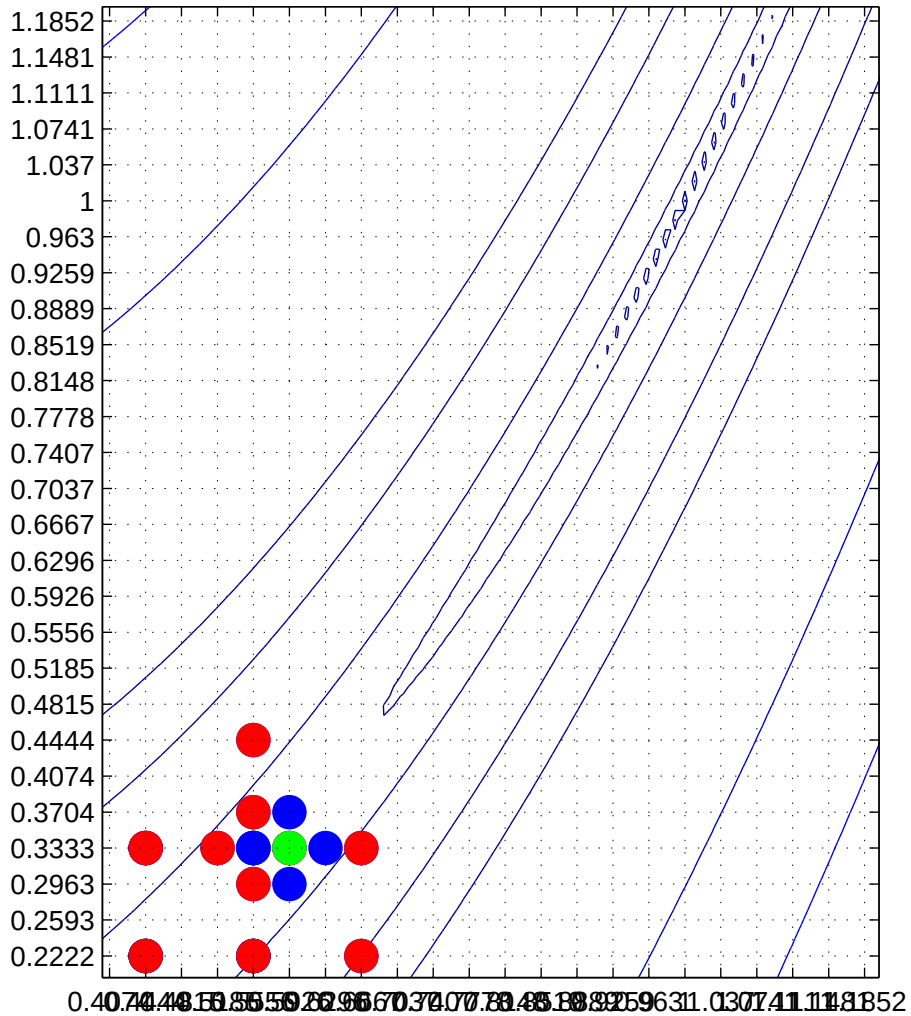
The standard compass-search on Rosenbrock's function



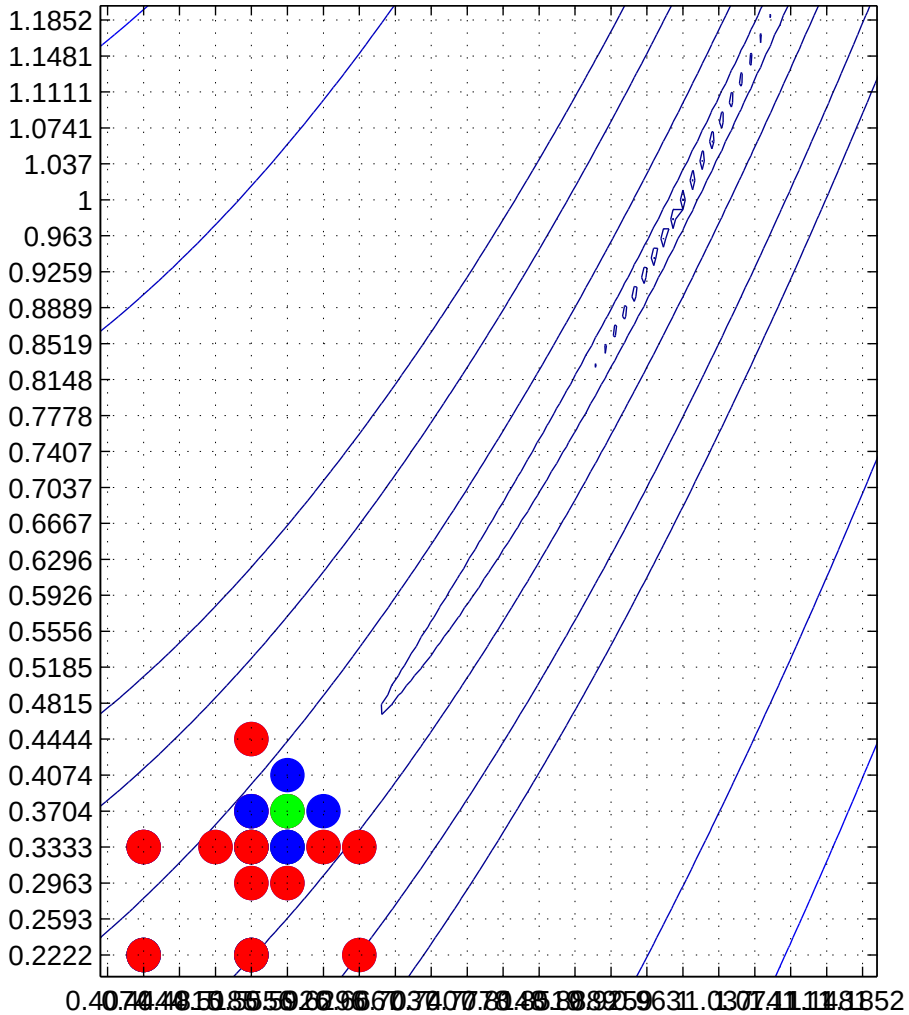
The standard compass-search on Rosenbrock's function



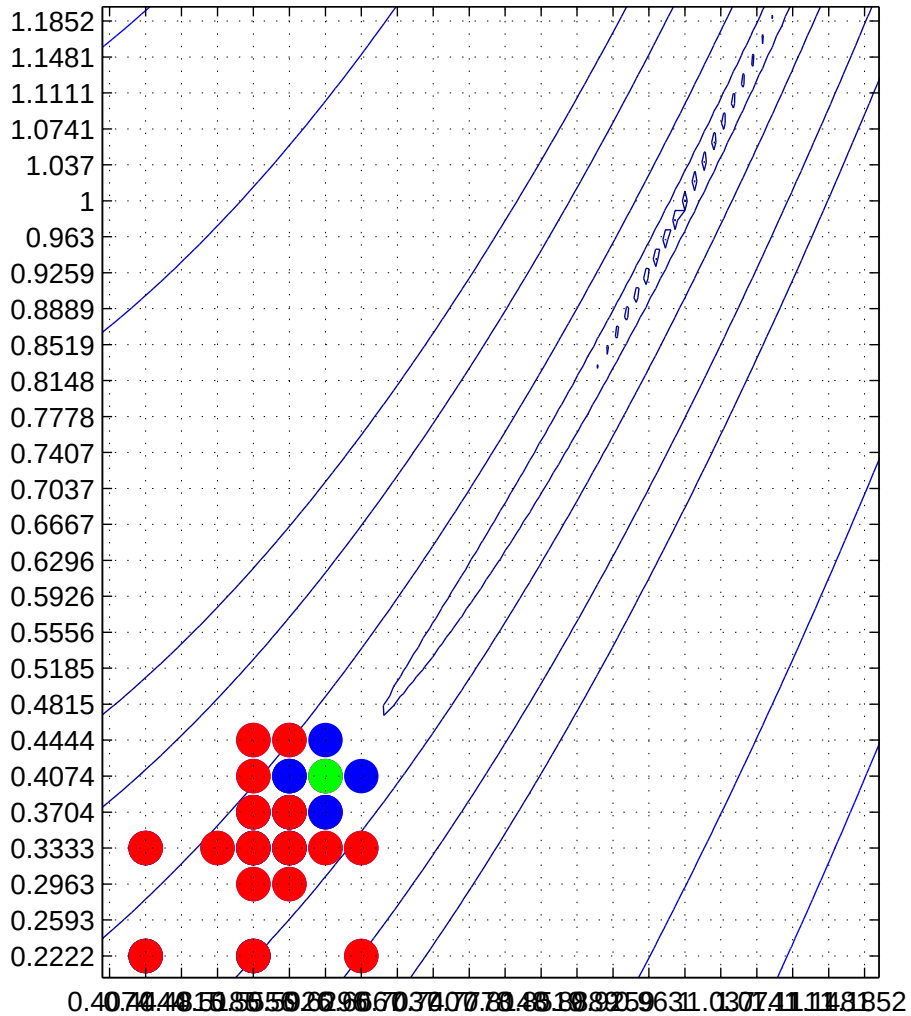
The standard compass-search on Rosenbrock's function



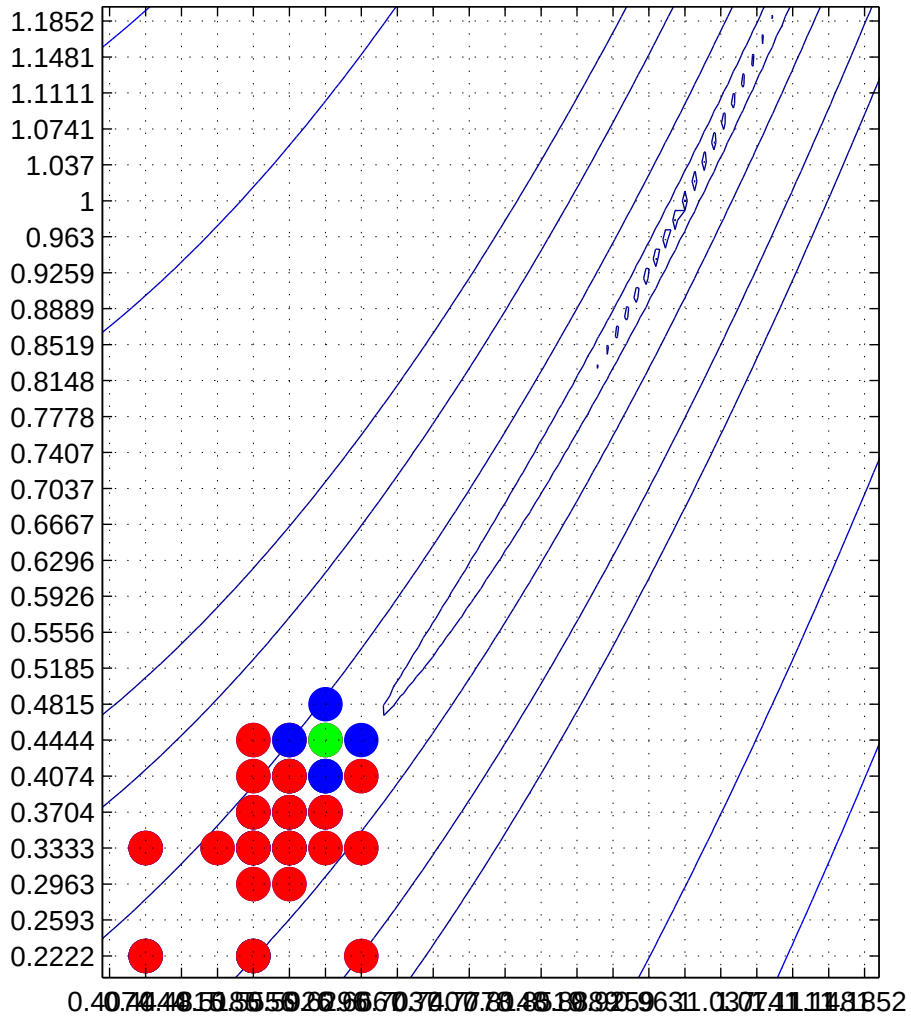
The standard compass-search on Rosenbrock's function



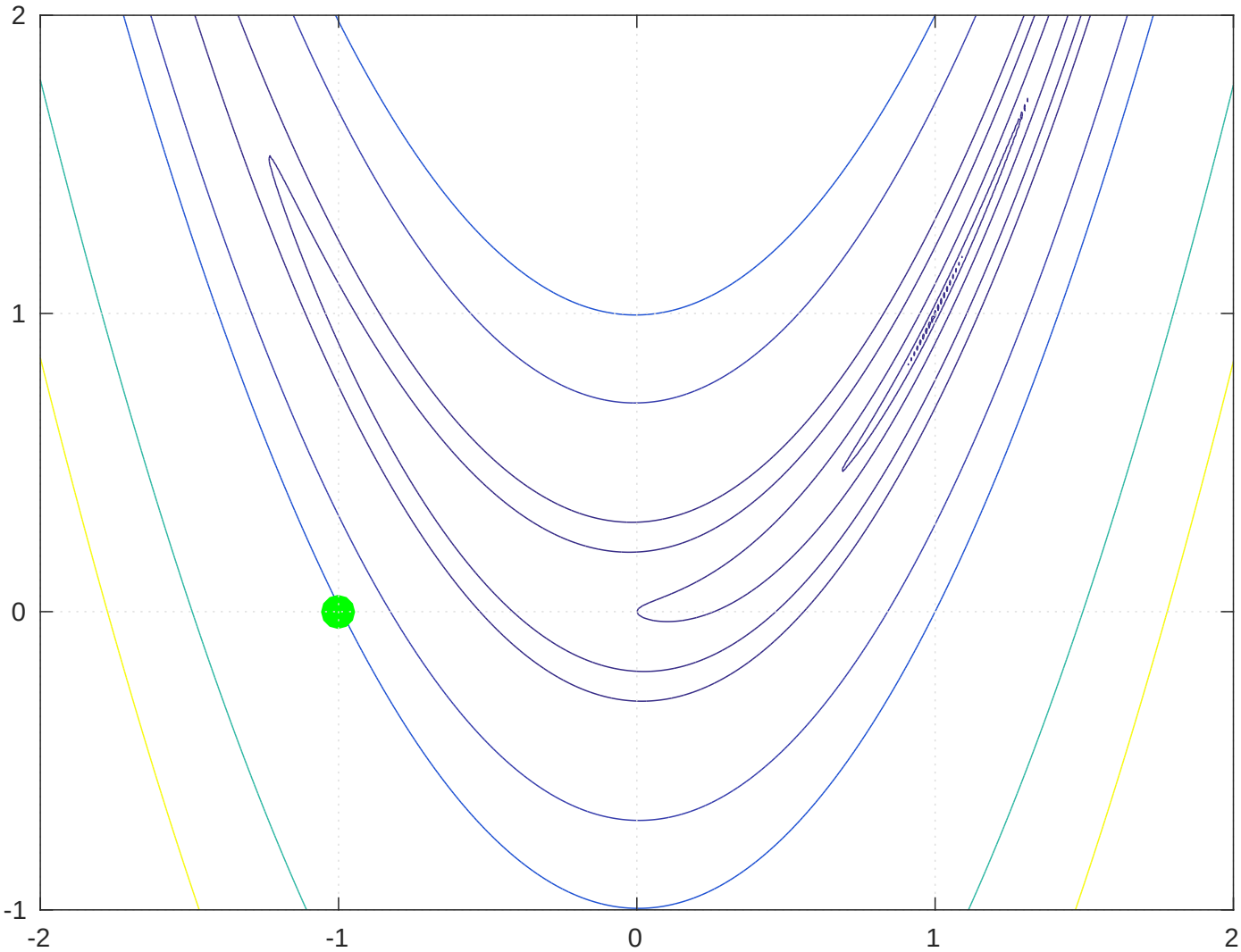
The standard compass-search on Rosenbrock's function



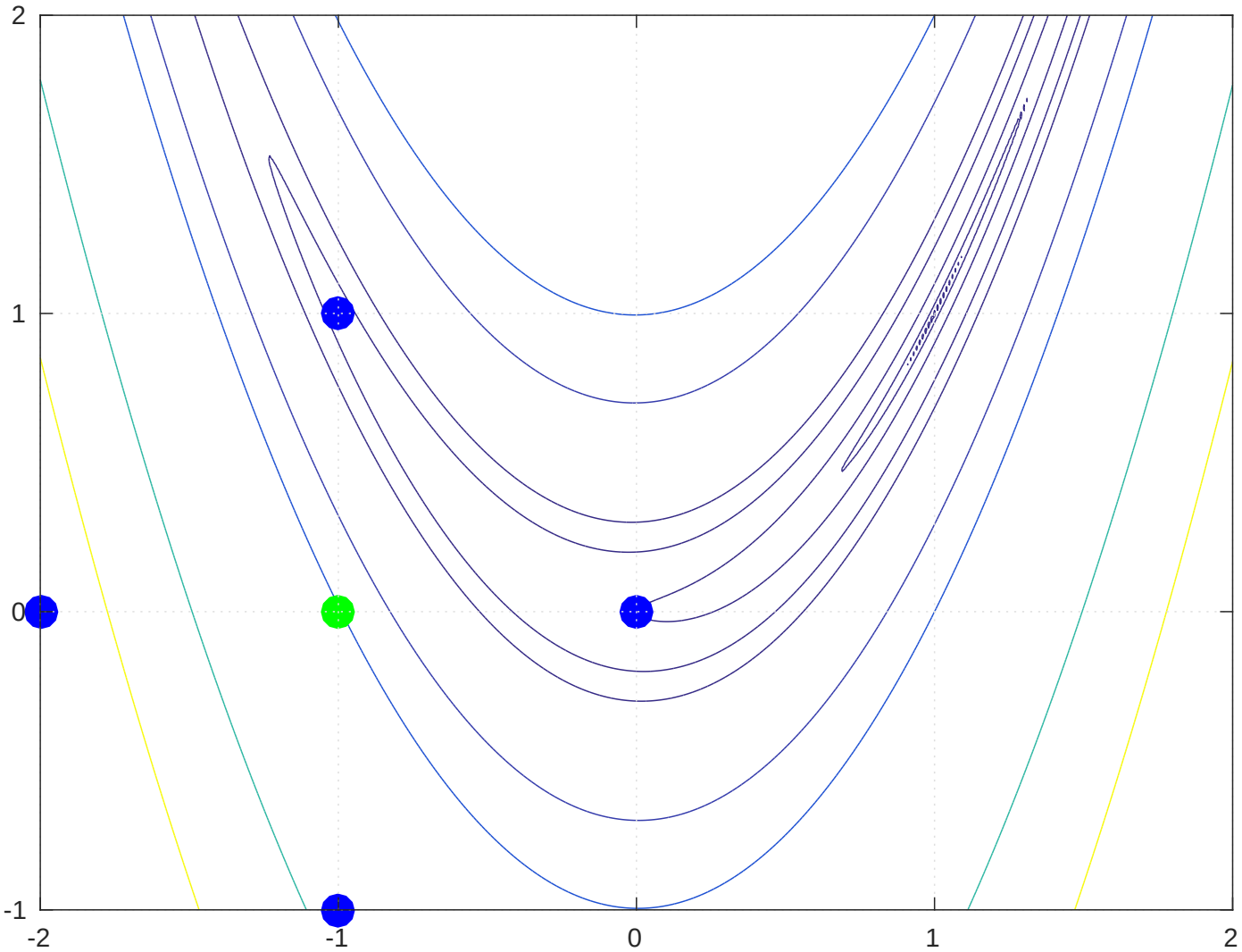
The standard compass-search on Rosenbrock's function



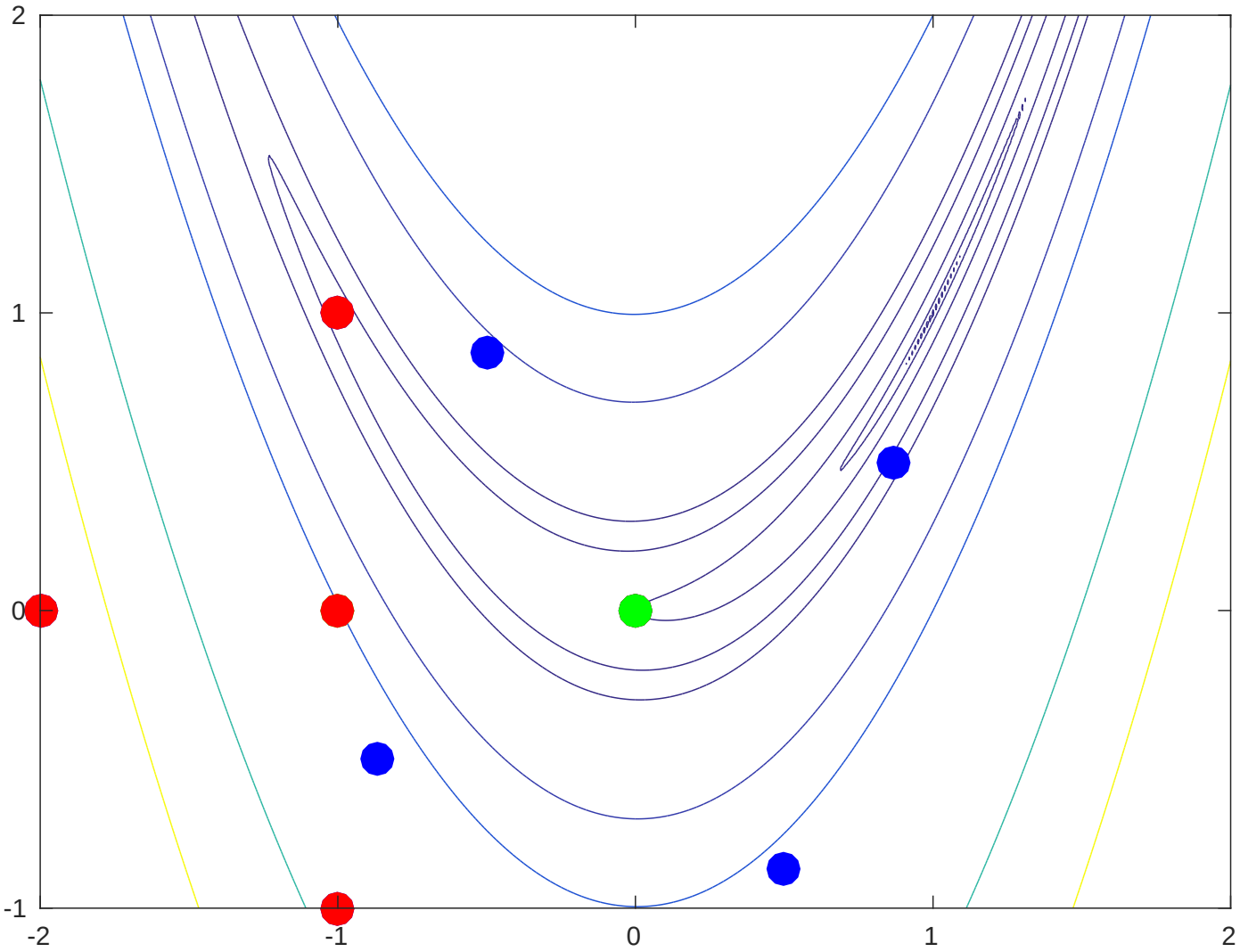
The randomized compass on Rosenbrock's function



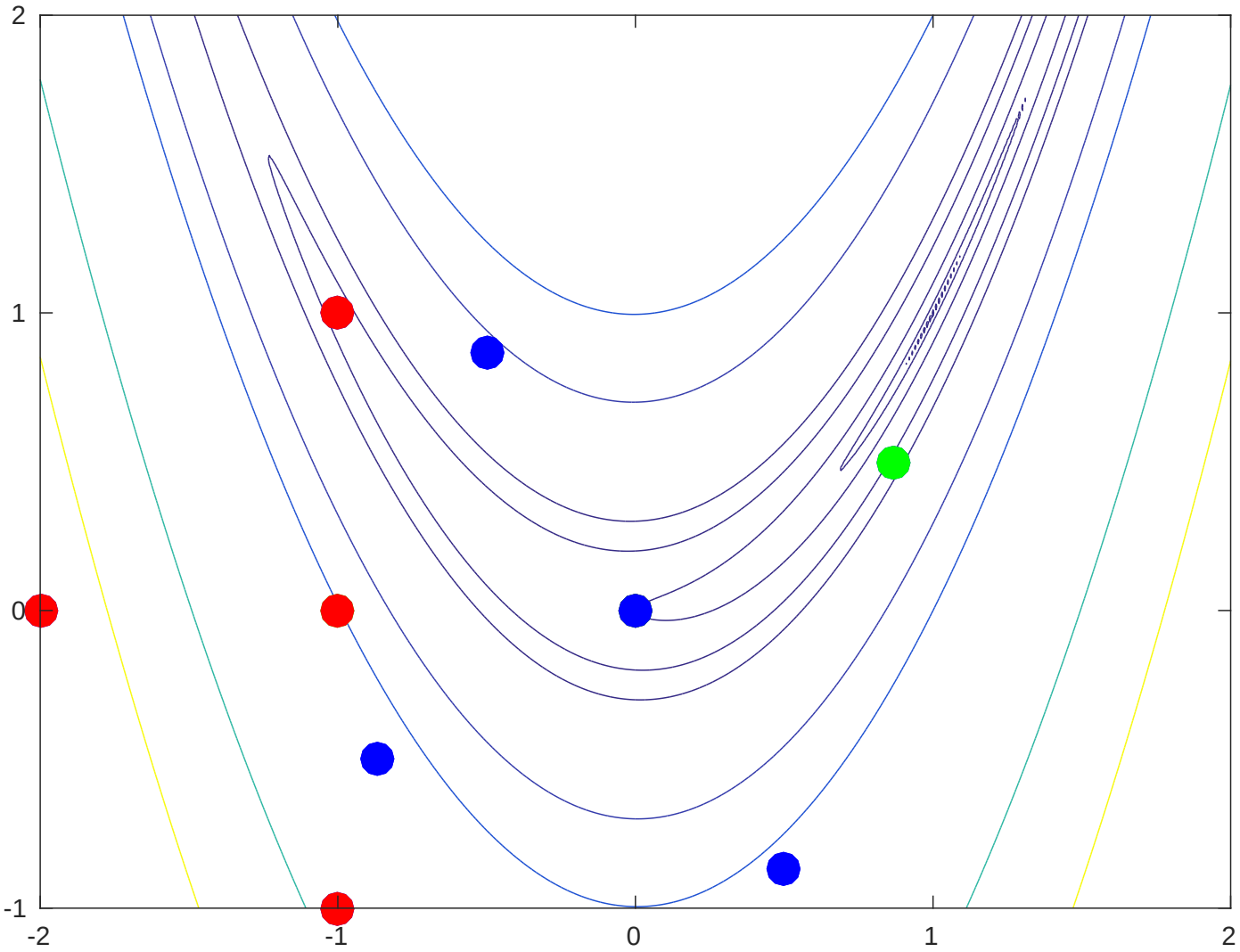
The randomized compass on Rosenbrock's function



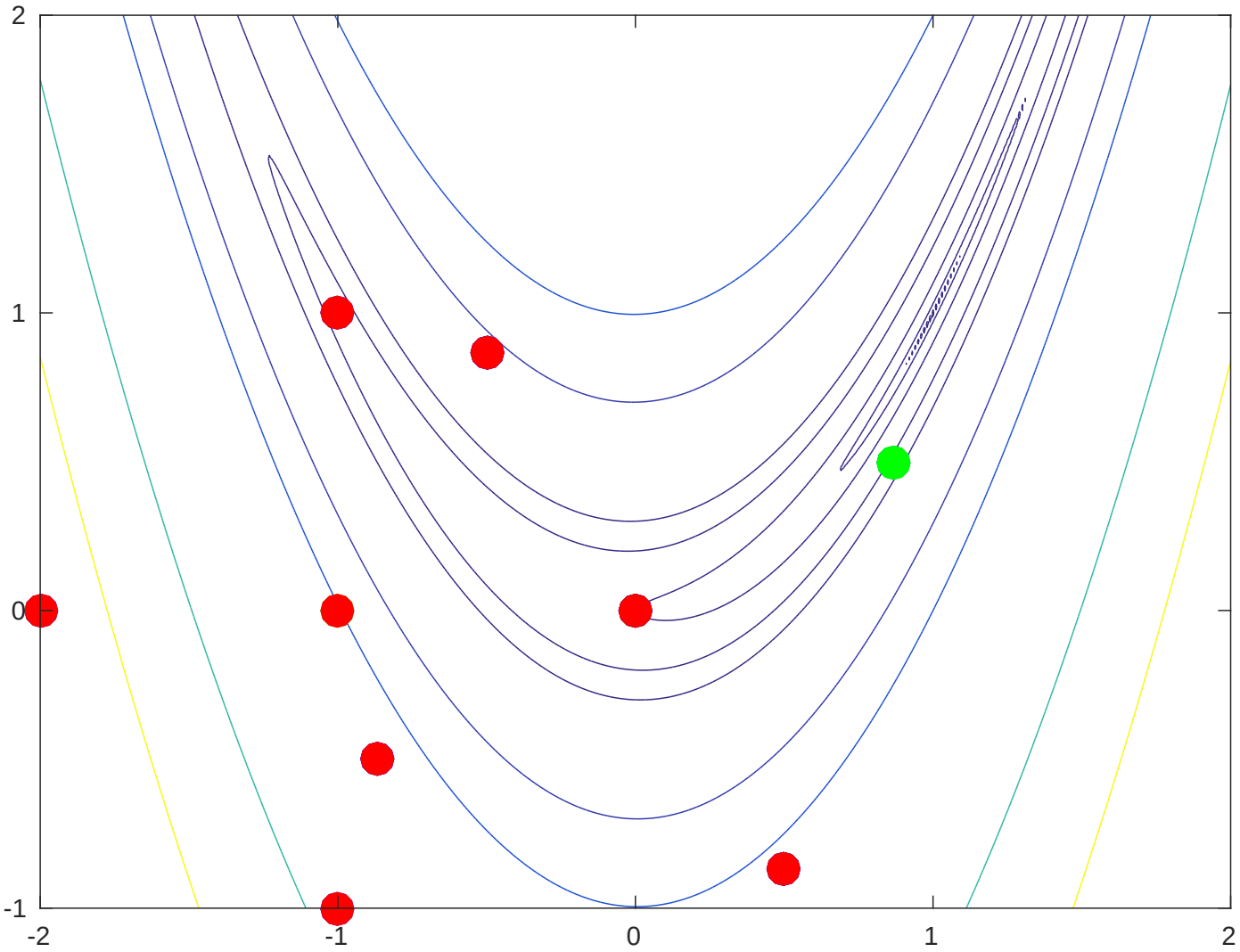
The randomized compass on Rosenbrock's function



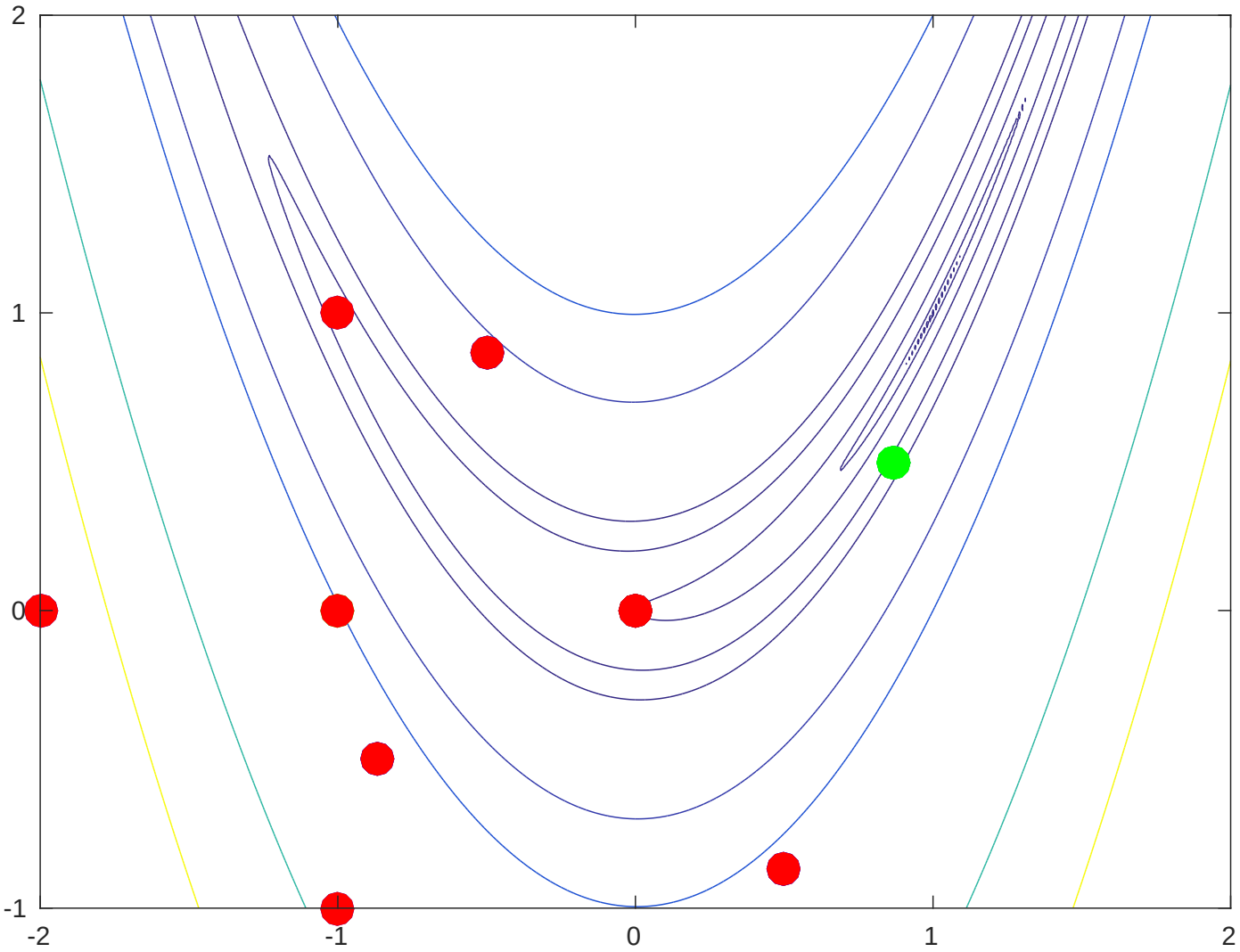
The randomized compass on Rosenbrock's function



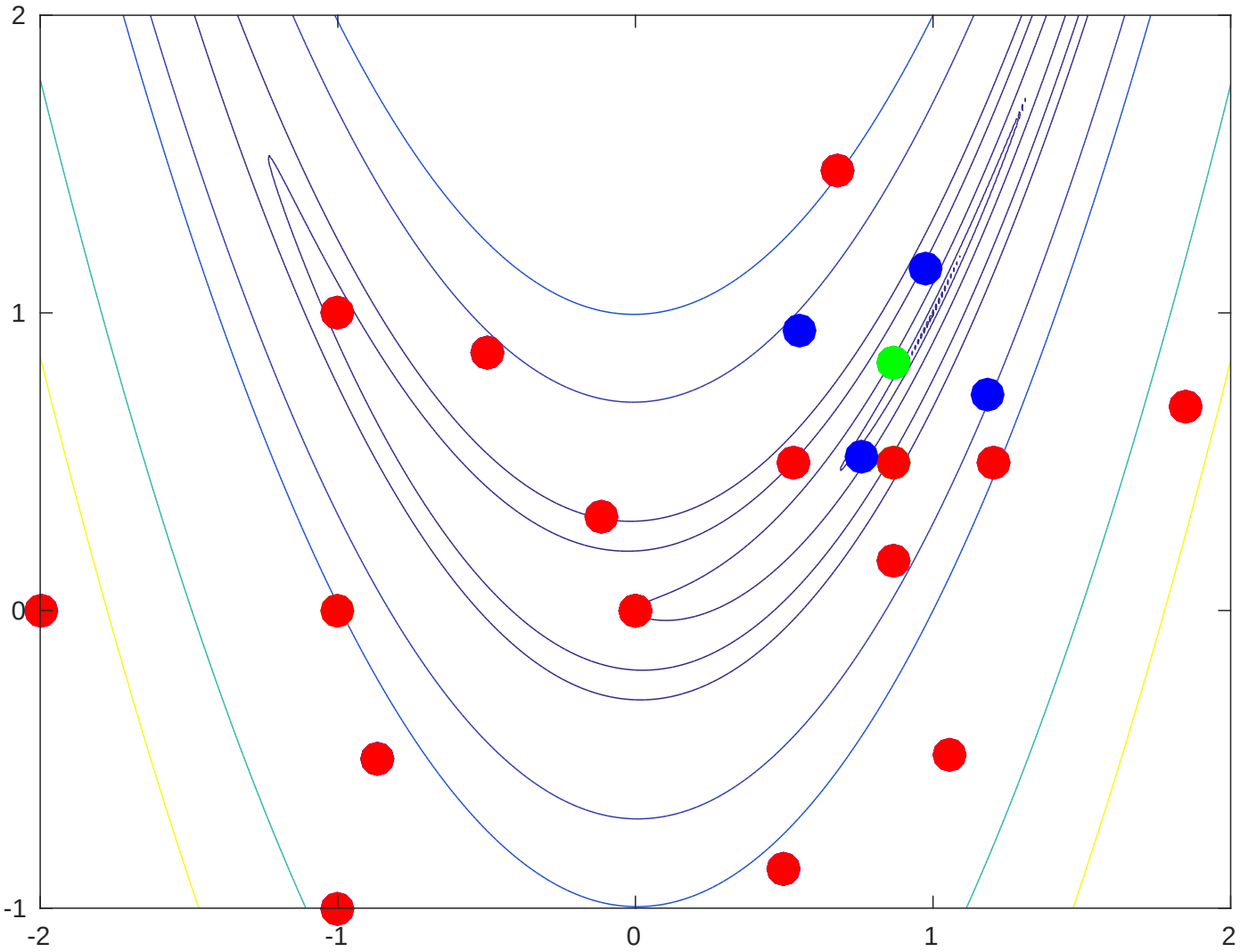
The randomized compass on Rosenbrock's function



The randomized compass on Rosenbrock's function



The randomized compass on Rosenbrock's function



Bound constraints, integer or lattice variables

Bound constraints

- detect which bounds are nearly active
- force their normals to belong to the set of poll directions
- include one-sided or truncated poll search

Integer or lattice variables

- align the initial grid with the integer grid
- avoid shrinking and rotations
- recursively explore a local tree of discrete subspaces
- keep track of record value in each such subspace to avoid re-exploration
- same thing if variables on a user-specified lattice

Additional algorithmic features:

- optional call of user-defined search step using all available objective function values
- accumulate successful descent directions (exploiting "inertia")
- optional user-defined variables' scaling
- provision for multilevel optimization

$$\min_x \max_y \min_z f(x, y, z)$$

with level-dependent bounds (equilibrium/game theory computations)

- incomplete function evaluations (crucial for training)
- flexible termination rules (including objective-function target)
- BFGS finish (for smooth problems)
- allows randomized termination test

Additional implementation features

- check-pointing at user-specified frequency
- allows objective functions with user-defined parameters
- very flexible keyword-based calling sequence
- MATLAB code (single file)
- direct CUTEst interface (for those interested)

Algorithmic parameters

User may specify (amongst others):

- grid shrinking/expansion factors
- inertia for defining progress directions
- initial scale in continuous variables
- local tree-search strategy (depth-first vs breadth-first)

(7 algorithmic parameters in total)

BFO self tuning

BFO has been self-tuned!

- on a large set of test problems (CUTEst) with continuous and mixed-integer variables
- using both the average and robust tuning strategies
- for all 7 algorithmic parameters

Outcome :

- robust strategy slightly better
- gains in performance of
  - 30% for continuous problems
  - 19% for mixed-integer problems

  compared with "intuitively reasonable values"
- *very* competitive with NOMAD (state-of-the-art pattern search algo)

And then...

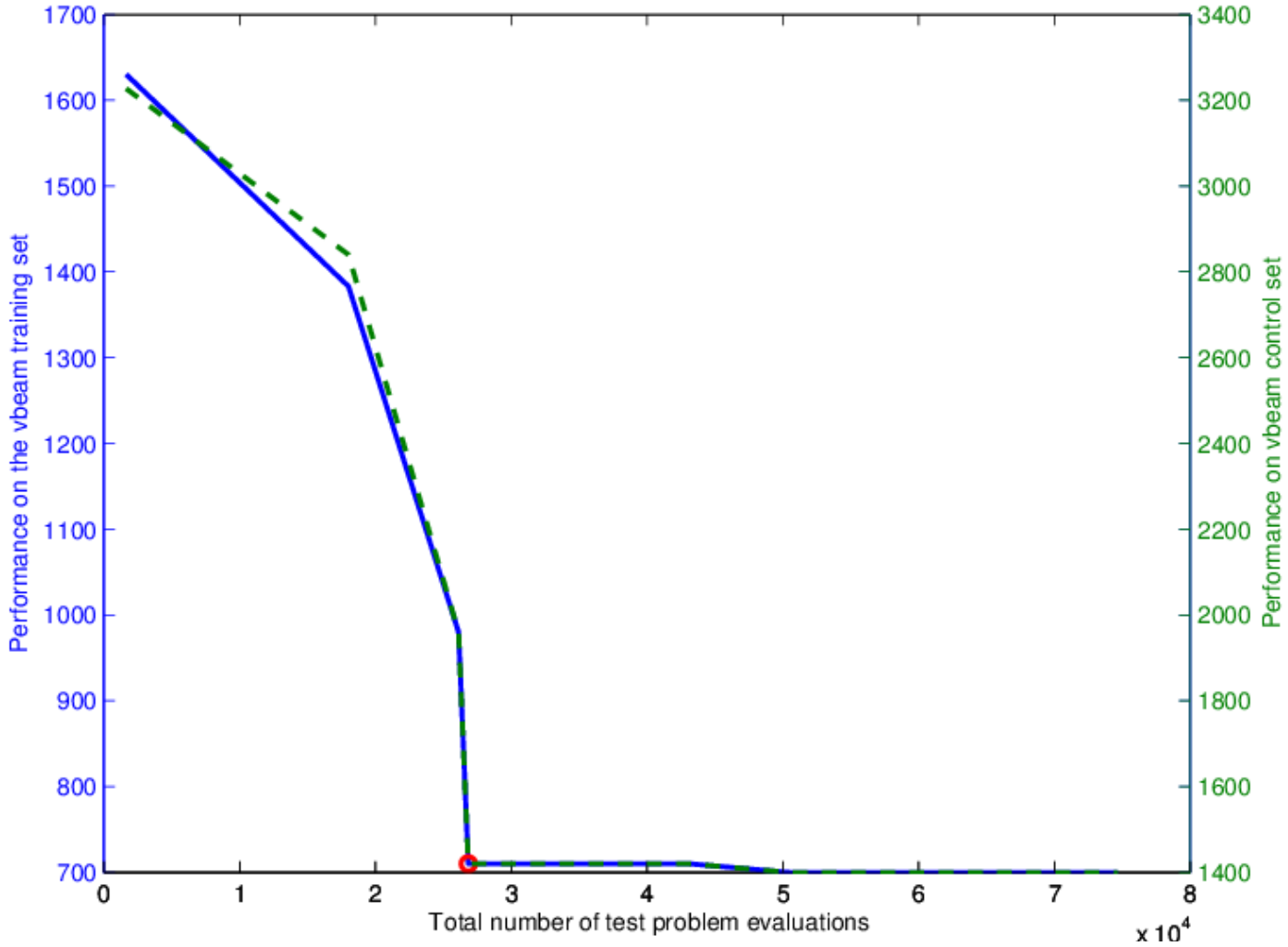
... the algorithm designer is (hopefully) happy !
But what about the user (with his/her own specific problems)?

BFO allows training by the user for specific problem classes

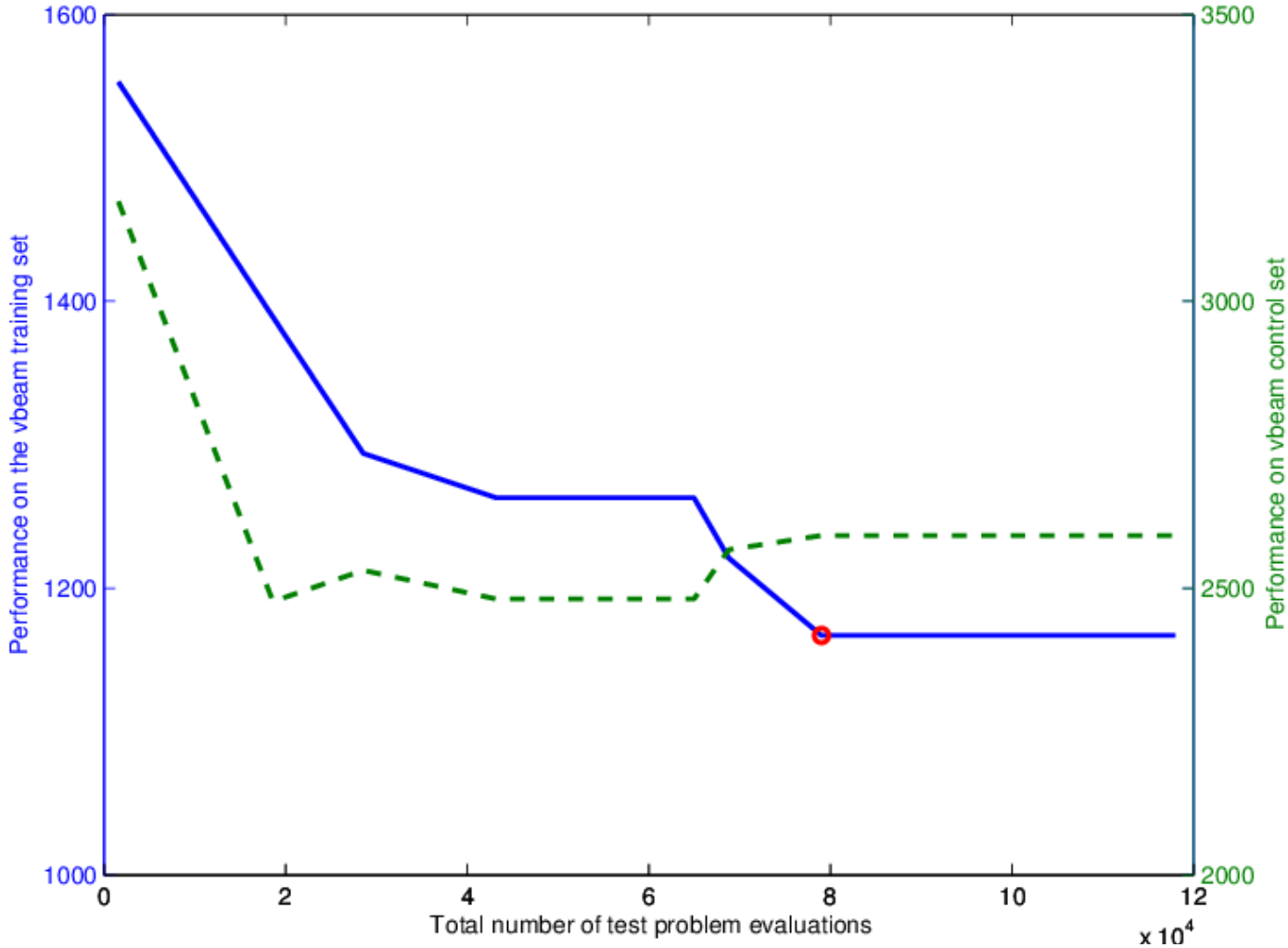
Does this work? Experiment on 2 specific classes of (minimization) problems

- nonlinear nonconvex trajectory tracking least-squares
- nonconvex regularized cubic models

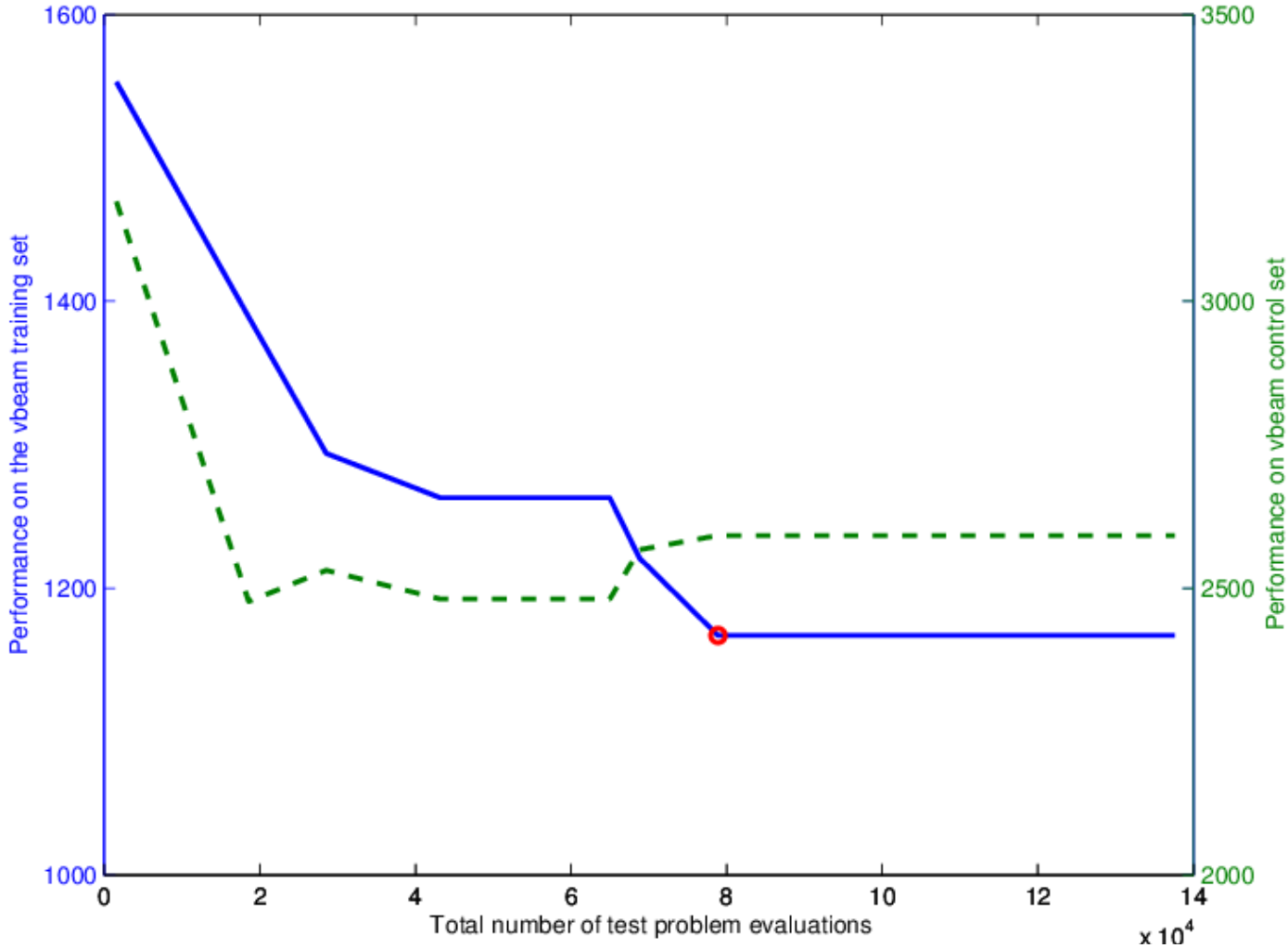
Trajectory tracking:
AO training, medium-low error deviation, low accuracy



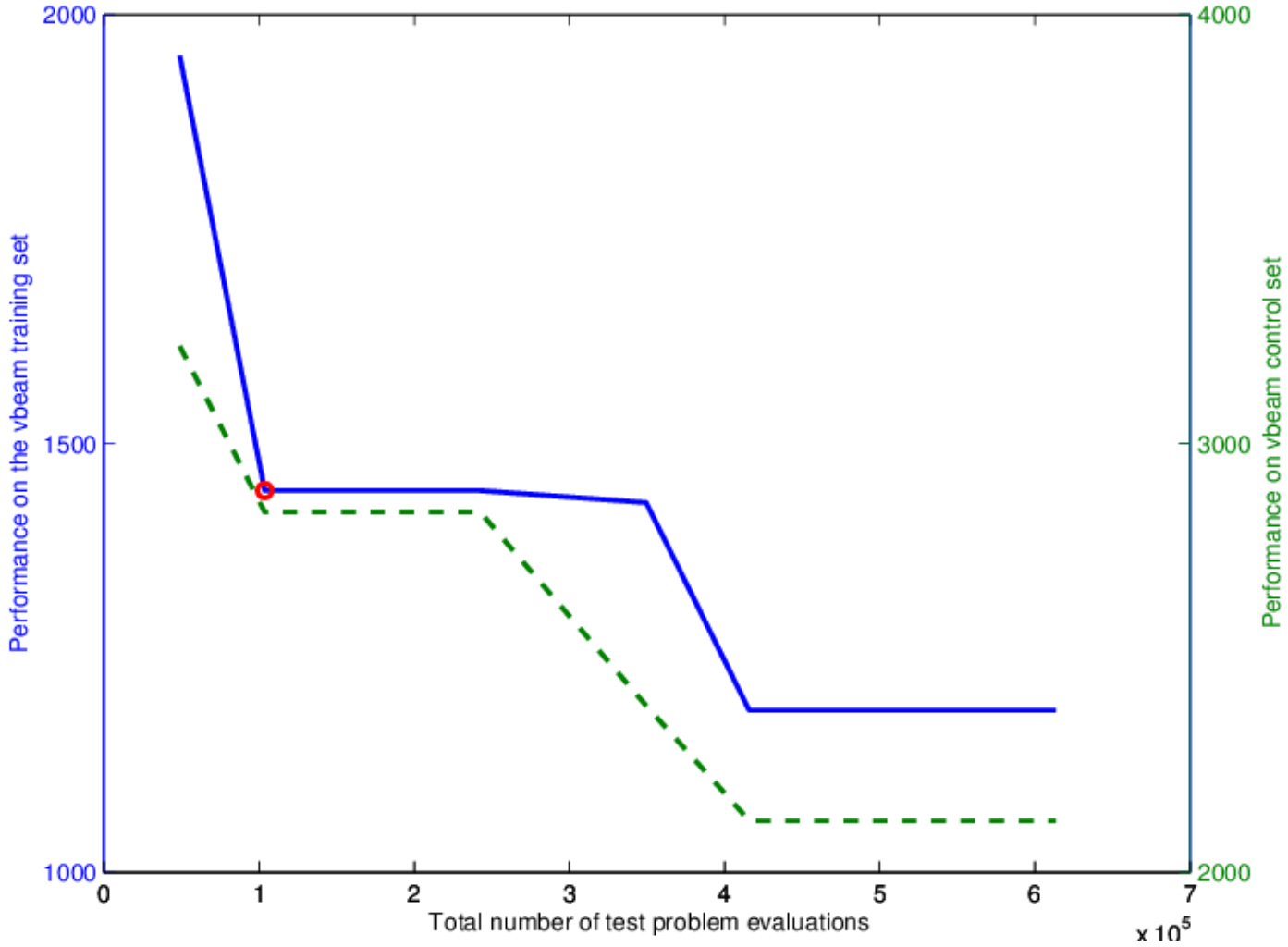
Trajectory tracking:
AO training, high error deviation, low accuracy



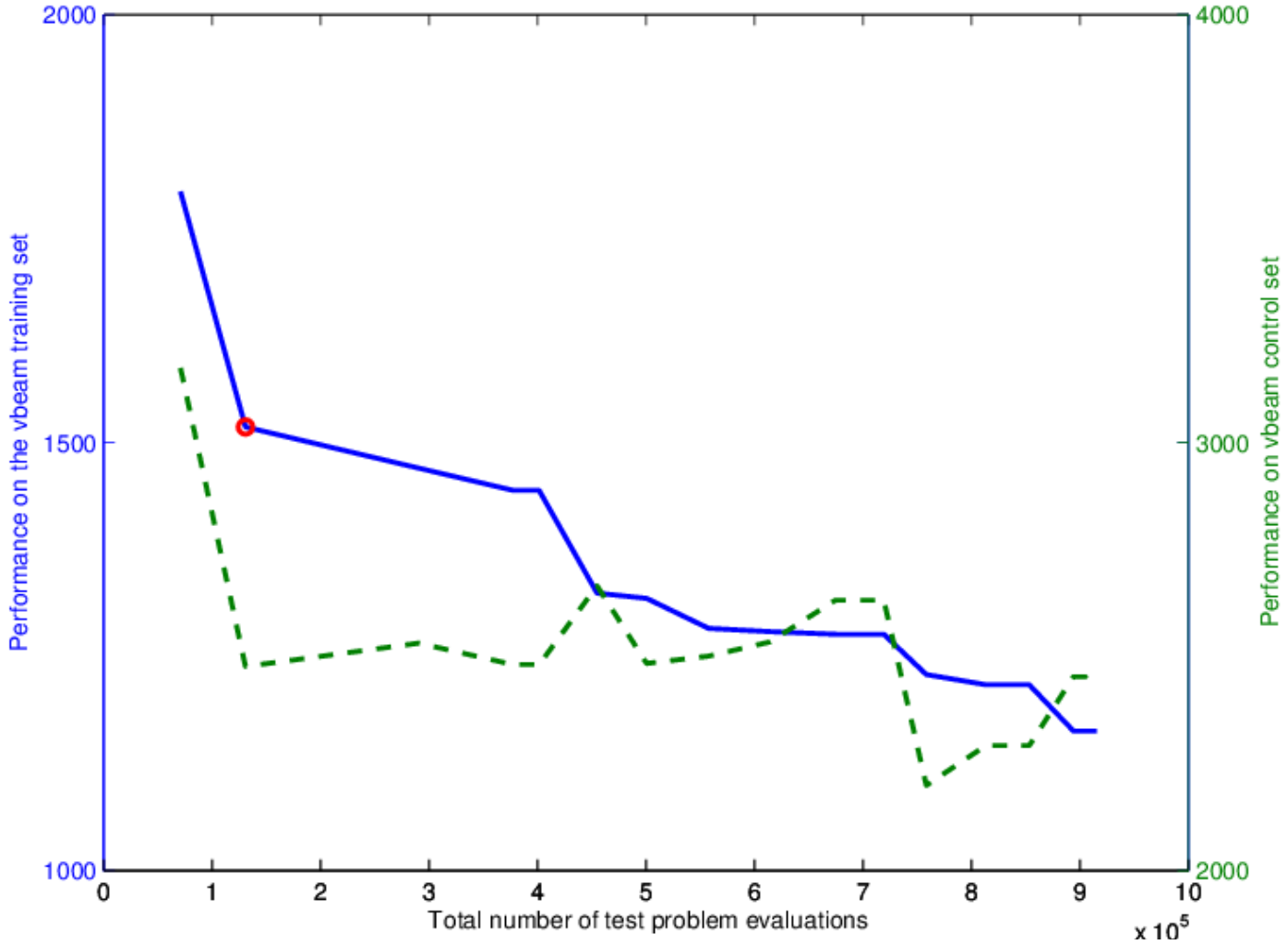
Trajectory tracking:
AO training, high error deviation, high accuracy



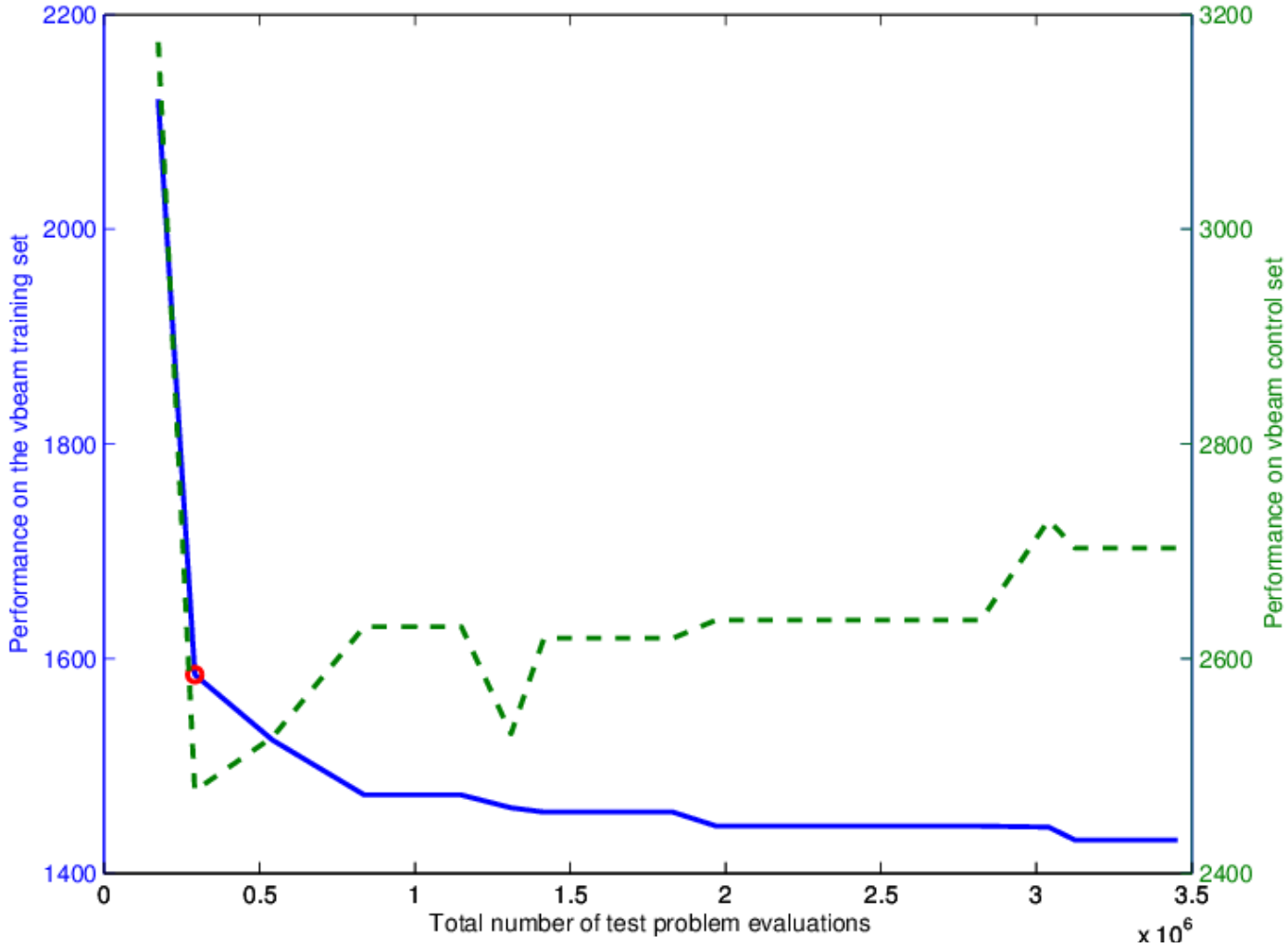
Trajectory tracking:
RO training, medium-low error deviation, low accuracy



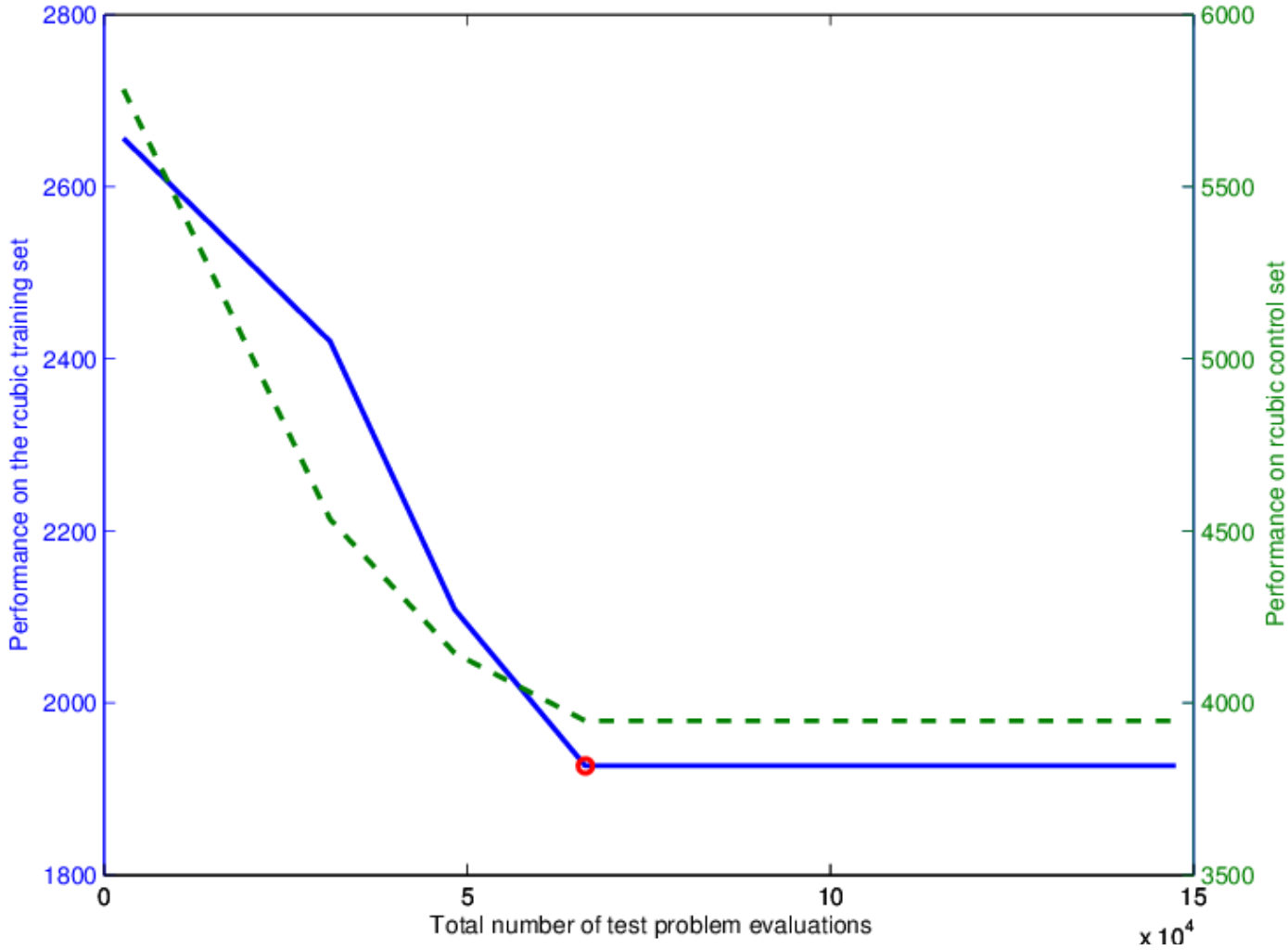
Trajectory tracking:
RO training, high error deviation, low accuracy



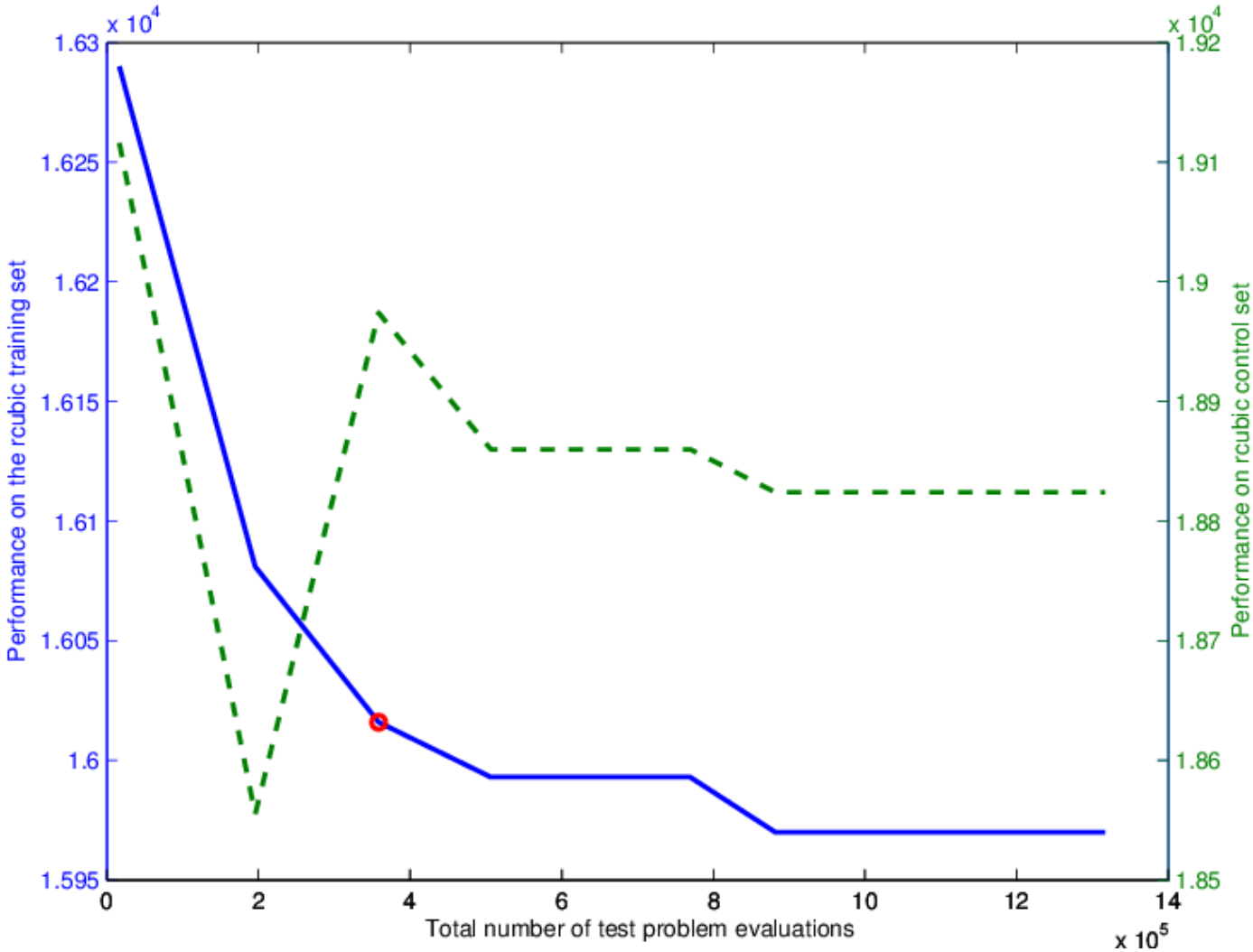
Trajectory tracking:
RO training, high error deviation, high accuracy



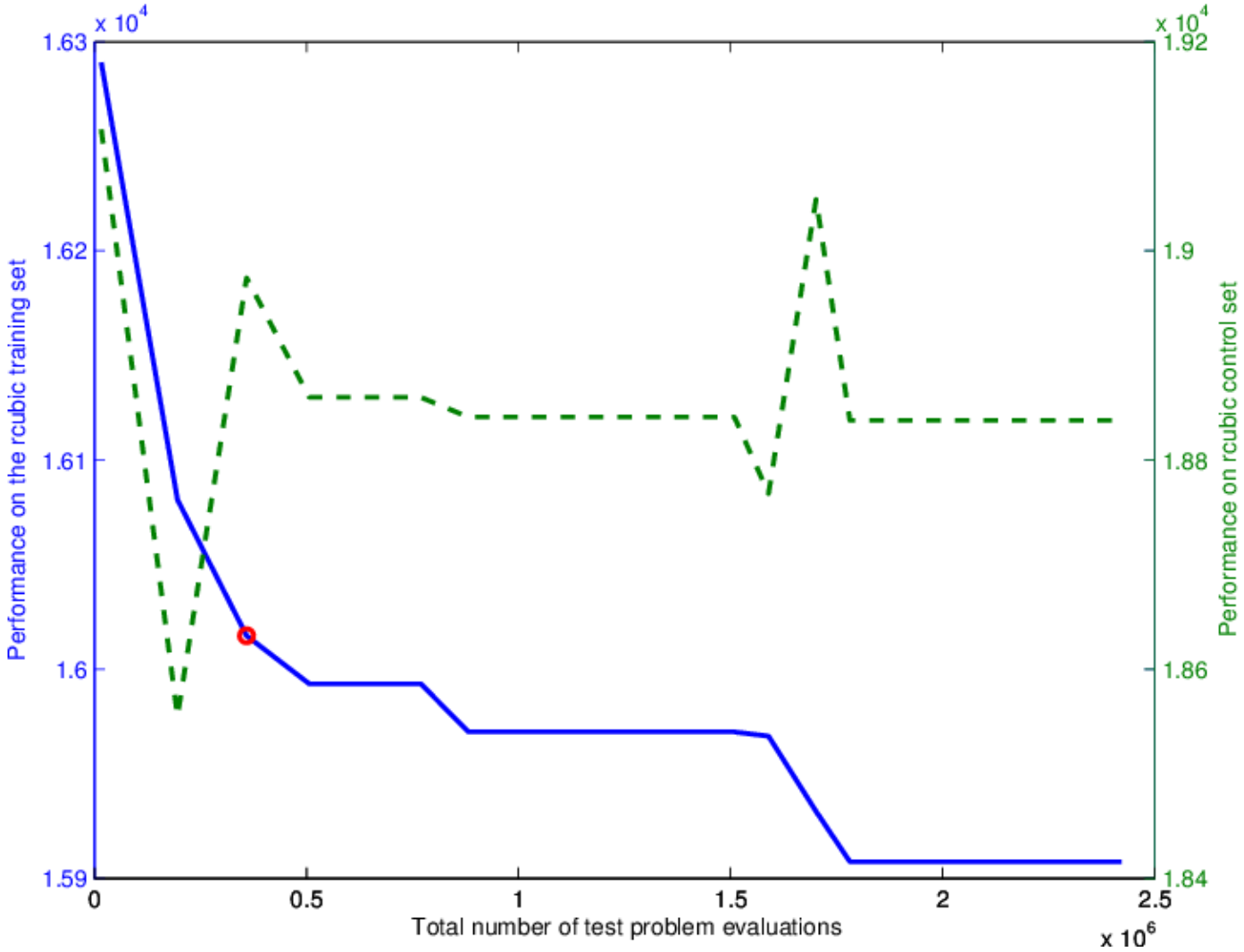
Regularized cubics: AO training, medium-low error deviation, low accuracy



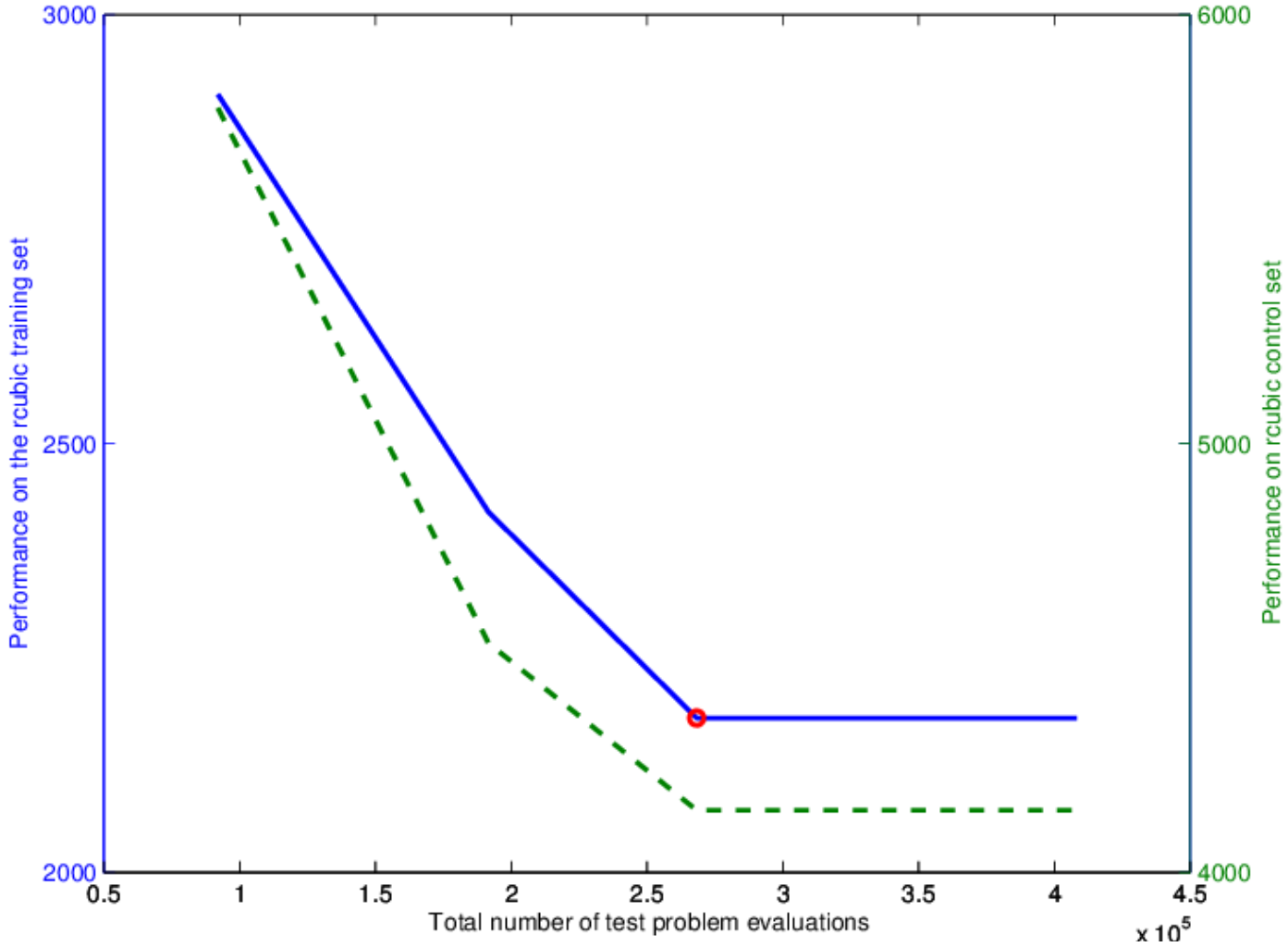
Regularized cubics:
AO training, high error deviation, low accuracy



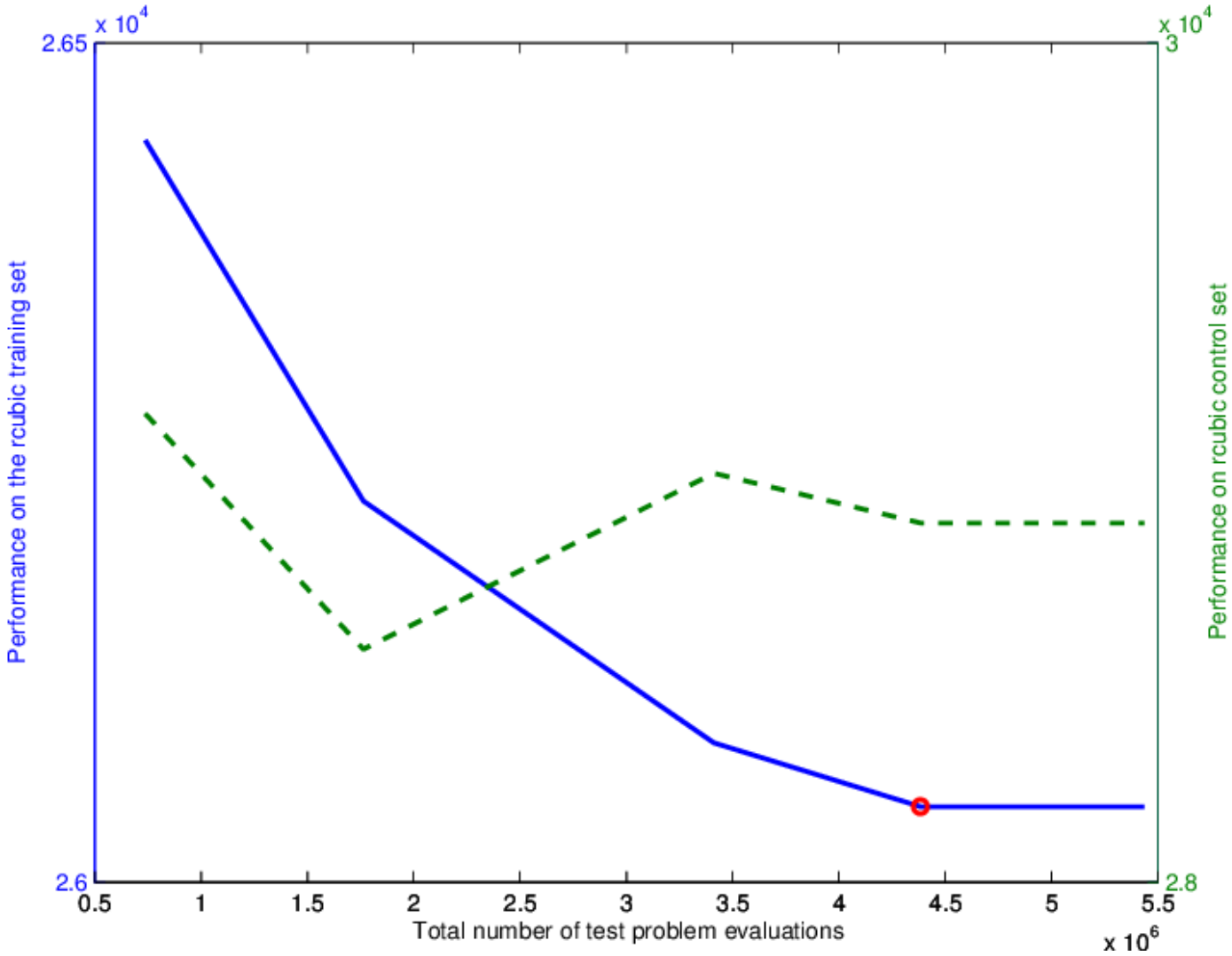
Regularized cubics:
AO training, high error deviation, high accuracy



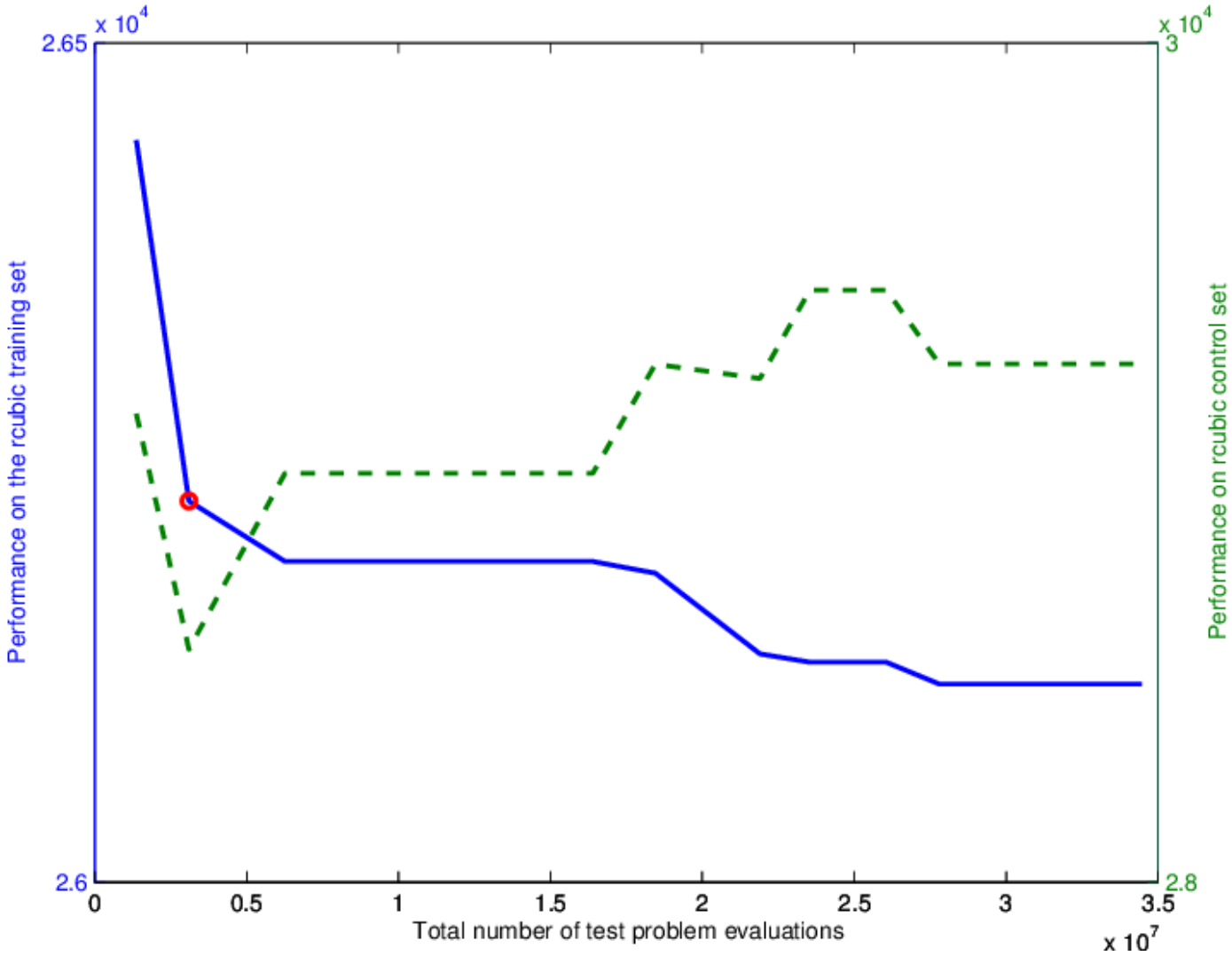
Regularized cubics:
RO training, medium-low error deviation, low accuracy



Regularized cubics:
RO training, high error deviation, low accuracy



Regularized cubics:
RO training, high error deviation, high accuracy



Examples of calls

- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ] )`
- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ], 'xtype', 'ic' )`
- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ], 'xlower', 0, 'epsilon',0.01)`
- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ] , ...`
  `'save-freq',10,'restart-file','bfo.rst')`
- `[ x, fx ] = bfo( @banana, [ -1.2, 1 ] , ...`
  `'training-mode', 'train', ...`
  `'training-parameters', 'fruity', ...`
  `'training-problems', {@banana,@apple},...`
  `'training-problems-data', {@fruit_data} )`
- `[ x, fx ] = bfo( @robust_training, [ 0, -1, 0, 1 ] , ...`
  `'xlevel', [ 1 1 2 2 ], ...`
  `'max-or-min', [ 'min', 'max'] )`

And now...

Some conclusions

*** Use BFO! ***

*** Use BFO to tune your algorithm! ***

(you can even tune BFO to tune your own algorithms)

More user-tunable codes?

Perspectives for the immediate future

- more complicated constraints
- use the original interpolation DFO algorithm in the search step!
- ...

Many thanks for your attention!

Further reading...

M. Porcelli and Ph. L. Toint,
"BFO, a trainable derivative-free Brute Force Optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables",

naXys Tech Report naXys-06-2015,
Namur Center for Complex Systems, University of Namur (Belgium), 2015.

available from http://perso.unamur.be/~phtoint/toint.html