The bbob-constrained COCO Test Suite

This document briefly describes the bbob-constrained test suite implemented in the COCO software [4, 5] and provides some example code.[1] To cite this work, please refer to the following paper:

> Paul Dufossé, Nikolaus Hansen, Dimo Brockhoff, Phillipe R. Sampaio, Asma Atamna, and Anne Auger. Building scalable test problems for benchmarking constrained optimizers. 2022. To be submitted to the *SIAM Journal of Optimization*

Contents

Table of the Functions in bbob-constrained

Number of constraints			1	3	9	$9+\lfloor 3n/4\rfloor$	$9+\lfloor 3n/2\rfloor$	$9+\lfloor 9n/2\rfloor$
Number of active constraints			1	2	6	$6+\lfloor n/2\rfloor$	$6+n$	$6+3n$
Objective	T	c_{scal}	Function IDs					
f_{sphere}	id	10	1	2	3	4	5	6
$f_{\text{ellipsoid}}$	T_{osz}	10^{-4}	7	8	9	10	11	12
f_{linear}	id	10	13	14	15	16	17	18
$f_{\text{elli\_rot}}$	T_{osz}	10^{-4}	19	20	21	22	23	24
f_{discus}	T_{osz}	10^{-4}	25	26	27	28	29	30
$f_{\text{bent\_cigar}}$	$T_{\text{asy}}^{0.5}$	10^{-4}	31	32	33	34	35	36
$f_{\text{diff\_powers}}$	id	10^{-2}	37	38	39	40	41	42
$f_{\text{rastrigin}}$	$T_{\text{asy}}^{0.2}\circ T_{\text{osz}}$	10	43	44	45	46	47	48
$f_{\text{rast\_rot}}$	$T_{\text{asy}}^{0.2}\circ T_{\text{osz}}$	10	49	50	51	52	53	54

Table 1: Identifiers of the bbob-constrained problems, where T is a non-linear search space transformations (described in Non-linear transformations) applied with each objective and $\text{id}(x) = x$. The 2-D contour plots of the first instance of each test problem can be found in the Appendix or by clicking the objective function name. Contour plots of other instances in 2-D are given at this link.

Functions Definitions

The bbob-constrained test suite uses several COCO "raw" functions as previously used in the unconstrained settings. The functions are defined in Table 2.

[1] To know more about the COCO software, go to the GitHub page.

Function name	Formulation	Transformations
Sphere	$f_{\text{sphere}}(x) = z^\top z + f_{\text{uopt}}$	$z = x - x^{\text{uopt}}$
Separable ellipsoid	$f_{\text{ellipsoid}}(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} z_i^2 + f_{\text{uopt}}$	$z = x - x^{\text{uopt}}$
Linear slope	$f_{\text{linear}}(x) = \sum_{i=1}^{n} 5\lvert s_i\rvert - s_i z_i + f_{\text{uopt}}$	$s_i = \text{sign}(x_i^{\text{uopt}})10^{\frac{i-1}{n-1}}$ $z_i = \begin{cases} x_i & \text{if } x_i^{\text{uopt}} x_i < 5^2 \\ x_i^{\text{uopt}} & \text{otherwise} \end{cases}$ for $i = 1, \ldots, n$
Rotated ellipsoid	$f_{\text{elli\_rot}}(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} z_i^2 + f_{\text{uopt}}$	$z = R(x - x^{\text{uopt}})$
Discus	$f_{\text{discus}}(x) = 10^6 z_1^2 + \sum_{i=2}^{n} z_i^2 + f_{\text{uopt}}$	$z = R(x - x^{\text{uopt}})$
Bent cigar	$f_{\text{bent\_cigar}}(x) = z_1^2 + 10^6 \sum_{i=2}^{n} z_i^2 + f_{\text{uopt}}$	$z = R(x - x^{\text{uopt}})$
Different powers	$f_{\text{diff\_powers}}(x) = \sqrt{10^6 \sum_{i=1}^{n} \lvert z_i\rvert^{2+4\frac{i-1}{n-1}}} + f_{\text{uopt}}$	$z = R(x - x^{\text{uopt}})$
Rastrigin	$f_{\text{rastrigin}}(x) = 10\left(n - \sum_{i=1}^{n} \cos(2\pi z_i)\right) + z^\top z + f_{\text{uopt}}$	$z = x - x^{\text{uopt}}$
Rotated Rastrigin	$f_{\text{rast\_rot}}(x) = 10\left(n - \sum_{i=1}^{n} \cos(2\pi z_i)\right) + z^\top z + f_{\text{uopt}}$	$z = R(x - x^{\text{uopt}})$

Table 2: COCO raw function definitions and search space transformations (applied before T from Table 1), where $x^{\text{uopt}} \in \mathbb{R}^n$ is a randomly sampled vector locating the unconstrained minimum of the objective function such that $f(x^{\text{uopt}}) = f_{\text{uopt}} \in \mathbb{R}$. Rotations $R \in \mathbb{R}^{n\times n}$ are randomly sampled orthogonal matrices, fixed for each instance.

Running an Experiment

A Python code example how to benchmark a solver on the COCO constrained test suite is given in Listing 1. Running a full experiment requires, among other things, to attach an observer to the current problem, allow a sufficiently large budget (number of objective and constraints evaluations) and loop over all problems of the test suite.

```
import cocoex, cocopp
from scipy.optimize import fmin_cobyla as fmin
import os, webbrowser  # to open post-processed results in the browser

def get_constraints(problem):
    """return the constraint function from ‘problem‘"""
    def constraint(x):
        """constraints for ‘fmin_cobyla‘ where >= 0 means feasible"""
        return -problem.constraint(x)
    return constraint  # a function

suite = cocoex.Suite('bbob-constrained', '', '')
observer = cocoex.Observer('bbob-constrained', 'result_folder: ' + fmin.__name__)
for problem in suite:  # this for-loop takes a minute or two
    problem.observe_with(observer)
    fmin(problem, problem.initial_solution, get_constraints(problem),
          rhobeg=2, rhoend=1e-8, maxfun=1e2)  # next step: increase maxfun to 1e3...

cocopp.main(observer.result_folder)  # post-processing the data, takes two minutes
webbrowser.open("file://" + os.getcwd() + "/ppdata/index.html")  # browse results
```

Listing 1: Benchmarking SciPy's COBYLA on the bbob-constrained test suite (running the code requires the installation of cocoex and cocopp from the COCO repository). Click here to get the above shown code listing. For a more comprehensive code example for running a full experiment in practice, optionally in batches, see example_experiment2.py.

The interface yields a feasible starting point x^{init} as `problem.initial_solution`. If the experimental design includes restarts, further feasible starting points can be sampled from x^{init} following the procedure described in Algorithm 1.

Algorithm 1 Sampling a new feasible starting point

Require: x^{init} and $\sigma > 0$
- **while** True **do**
  - Sample $z \sim \mathcal{N}(0, \text{id})$ and set $x = x^{\text{init}} + \sigma z$
  - **if** x is feasible **then**
    - return x as the new starting point
  - **else**
    - $\sigma \leftarrow \sigma/2$

General Problem Definition

A constrained test problem is written in the standard form

$$\underset{x \in \mathcal{X}}{\text{minimize}} \quad f(x) \qquad \text{subject to} \quad g_k(x) \leq 0 \quad \text{for } k = 1, \ldots, m \tag{1}$$

where $\mathcal{X}$ may be $[-5, 5]^n$ or $\mathbb{R}^n$, n is the dimension of the search space and m is the number of generally non-linear constraints.

To create the constrained problems, each one of the 9 objective functions is composed with a non-linear transformation of the search space defined previously and is combined with $m' \in \{1, 2, 6, 6 + \lfloor n/2 \rfloor, 6 + n, 6 + 3n\}$ active constraints and $\lfloor m'/2 \rfloor$ inactive constraints, having overall $m = \lfloor 3m'/2 \rfloor$ constraints. This results in 54 constrained test problems, f_i, $i = 1, \ldots, 54$, summarized in Table 1. In practice, x^{opt} is sampled randomly to define different instances of the same constrained problem. Additionally, the objective functions are scaled down by a factor $c_{\text{scal}} > 0$ which is specific to each function. The final optimization problem writes:

$$\underset{x \in \mathcal{X}}{\text{minimize}} \quad c_{\text{scal}} f(v) \qquad \text{subject to} \quad g_i(v) \leq 0 \quad \text{for } i = 1, \ldots, m \tag{2}$$

where $v = T(x - x^{\text{opt}})$ and T is the non-linear transformation reported in Table 1 along with c_{scal} and the number of constraints associated to each problem. The considered dimensions are the same as for the `bbob` test suite, that is $n \in \{2, 3, 5, 10, 20, 40\}$.

Principles of the Construction

The theory allows to control the location of the constrained optimum $y^\star \in \mathbb{R}^n$, provided that its gradient is non-zero. In the construction, f is an already shifted BBOB function, hence taking $y^\star = 0$ generally provides $\nabla f(y^\star) \neq 0$ (in some cases, e.g. the Rastrigin function, some additional conditions on $y^\star$ are needed [2]). From this point, following the gradient direction, an initial solution is determined

$$x^{\text{init}} = y^\star + \rho \nabla f(y^\star) + x^{\text{opt}} \tag{3}$$

where $x^{\text{opt}} \in \mathcal{X}$ defines another (random) translation such that the global minimum of the constrained problem is not in $y^\star = 0$ and $\rho > 0$ is a scaling factor ensuring that $x^{\text{init}} \in \mathcal{X}$.

Performance Assessment

In order to assess the performance of optimization algorithms, we record the number of objective and constraint evaluations to reach a certain target value.

The target definition encompasses objective minimization and constraint satisfaction together using the merit function

$$\tilde{f}(x) := \max(f_{\text{opt}}, f(x)) + \sum_{k=0}^{m} \max(0, g_k(x)) \ , \tag{4}$$

where $f_{\text{opt}} = f(x^{\text{opt}})$. It is easy to see that $\min \tilde{f} = f_{\text{opt}}$.

The estimated Expected Runtime (ERT) [3] is computed from feasible points only and for 7 target values $f_{\text{target}} = f_{\text{opt}} + 10^i$, with $i \in \{1, 0, -1, -2, -3, -5, -6\}$.

Data points for the ECDFS [3] are collected for 41 target values $f_{\text{target}} = f_{\text{opt}} + 10^i$, with i evenly distributed in $[-6, 2]$.

Supplementary Material

Construction of the feasible region

The construction of any problem of the test suite is detailed in the full paper [2] and relies on Theorem 4.2.16 in [1, p. 195] stating that, if f is *pseudo-convex* at $y^\star$, and g_k is differentiable and *quasi-convex* at $y^\star$ for all $k = 1, \dots, m$, then the KKT conditions are sufficient to ensure that $y^\star$ is a global minimum of the constrained optimization problem. It is easy to show that, if f is *strictly* pseudo-convex at $y^\star$, then the global minimum is also unique [2]. We refer to the textbook of Bazaraa et al. [1] for the definitions and properties of pseudo- and quasi-convexity.

The algorithm to generate an instance of a constrained problem first builds a set of m *linear* constraints, of the form $g_k(x) = \alpha_k a_k^\top (x - y^\star) + b_k$ by setting $a_k \in \mathbb{R}^n$ where $\alpha_k > 0, b_k \geq 0 \in \mathbb{R}$ are input parameters. The first constraint is chosen such that $a_1 = -\alpha_1 \nabla f(y^\star)/\|\nabla f(y^\star)\|$ and $b_1 = 0$ hence $g_1(y^\star) = 0$. If $m = 1$, this is the only choice to satisfy the stationarity condition in the KKT equations. The remaining constraints are randomly sampled from an isotropic multivariate normal distribution. If $g_k(y^\star + \rho \nabla f(y^\star)) > 0$, the sign of a_k is flipped to make sure that $y^\star + \rho \nabla f(y^\star)$ is feasible. A constraint is inactive at the constructed optimum if $b_k > 0$.

The Lagrange multipliers associated to $y^\star$ when the constraints are generated according to the aforementioned procedure are $[1, 0, \dots, 0] \in \mathbb{R}^m$. A second step of the algorithm modifies the set of constraints gradients in a randomized manner such that the first constraint is not any more aligned with the gradient direction at the optimum. The constraints' indices are also shuffled.

Non-linear transformations

Let $T : \mathbb{R}^n \to \mathbb{R}^n$ be a bijective transformation of the search space with inverse T^{-1}. Then $T^{-1}(y^\star)$ is the global optimum of the problem

$$\underset{x \in \mathcal{X}}{\text{minimize}} \quad f(T(x)) \qquad \text{subject to} \quad g_i(T(x)) \leq 0 \quad i = 1, \dots, m \tag{5}$$

if $y^\star$ is the global optimum of the original problem as described in Equation (1). This allows to apply non-linear transformations of the search space to the set of constructed linear constraints. For practical reasons, we consider in this work bijective transformations T that have $y^\star$ as fixed point, i.e. $T(y^\star) = y^\star$. As a consequence, the optimum of the transformed problem in Equation (5) remains $y^\star$. This way, we avoid computing the inverse transformation at $y^\star$. The non-linear transformations implemented in the test suite are coordinate-wise, defined as T_{asy}^β, where $\beta > 0$, and T_{osz}, defined as follows:

$$T_{\text{asy}}^\beta(x) = \left(\begin{cases} x_i^{1+\beta \frac{i-1}{n-1}\sqrt{x_i}} & \text{if } x_i > 0 \\ x_i & \text{otherwise} \end{cases} \right)_{i=1,\dots,n}, \tag{6}$$

$$T_{\text{osz}}(x) = \left(\operatorname{sign}(x_i) \exp(\hat{x}_i + 0.049(\sin(c_1 \hat{x}_i) + \sin(c_2 \hat{x}_i))) \right)_{i=1,\dots,n}, \tag{7}$$

with $\hat{x}_i = \log(|x_i|) \mathbb{1}_{\{x_i \neq 0\}}$ and $\operatorname{sign}(x_i)$ is the sign function with convention $\operatorname{sign}(0) = 0$. The value for the constants c_1 and c_2 are $c_1 = \begin{cases} 10 & \text{if } x_i > 0 \\ 5.5 & \text{otherwise} \end{cases}$, and $c_2 = \begin{cases} 7.9 & \text{if } x_i > 0 \\ 3.1 & \text{otherwise} \end{cases}$. The idea is that T_{asy}^β introduces asymmetry, as it only changes positive coordinates, and $T_{\text{osz}} : \mathbb{R}^n \to \mathbb{R}^n$ oscillates around the origin. Both transformations are bijective, continuous, sign-preserving hence have fixed point 0.

References

[1] Mokhtar S. Bazaraa, C. M. Shetty, and Hanif D. Sherali. *Nonlinear Programming.* Wiley, 2006.

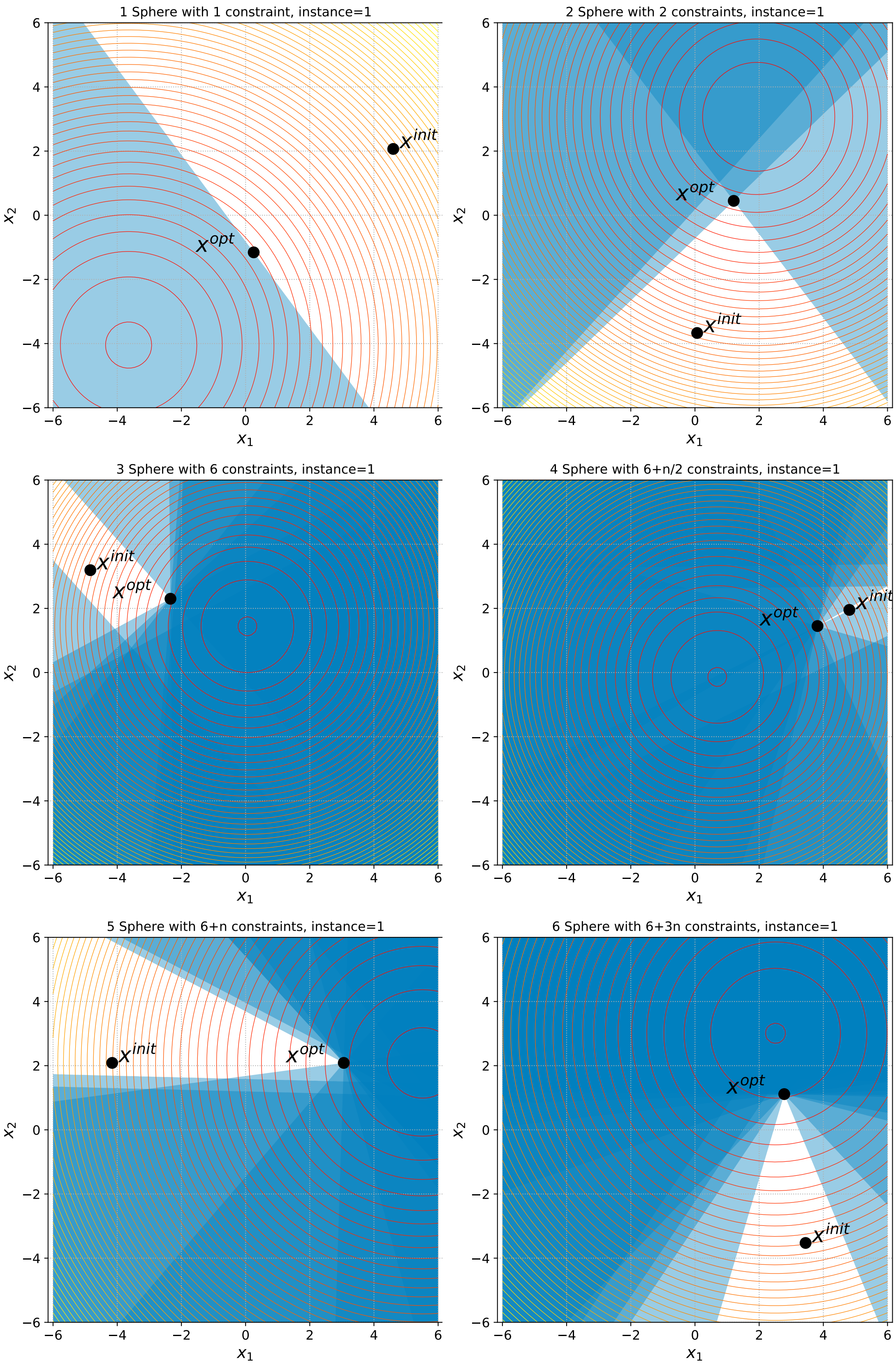
[2] Paul Dufossé, Nikolaus Hansen, Dimo Brockhoff, Phillipe R. Sampaio, Asma Atamna, and Anne Auger. Building scalable test problems for benchmarking constrained optimizers. 2022. To be submitted to the *SIAM Journal of Optimization.*

[3] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar. COCO: performance assessment. *CoRR*, 2016.

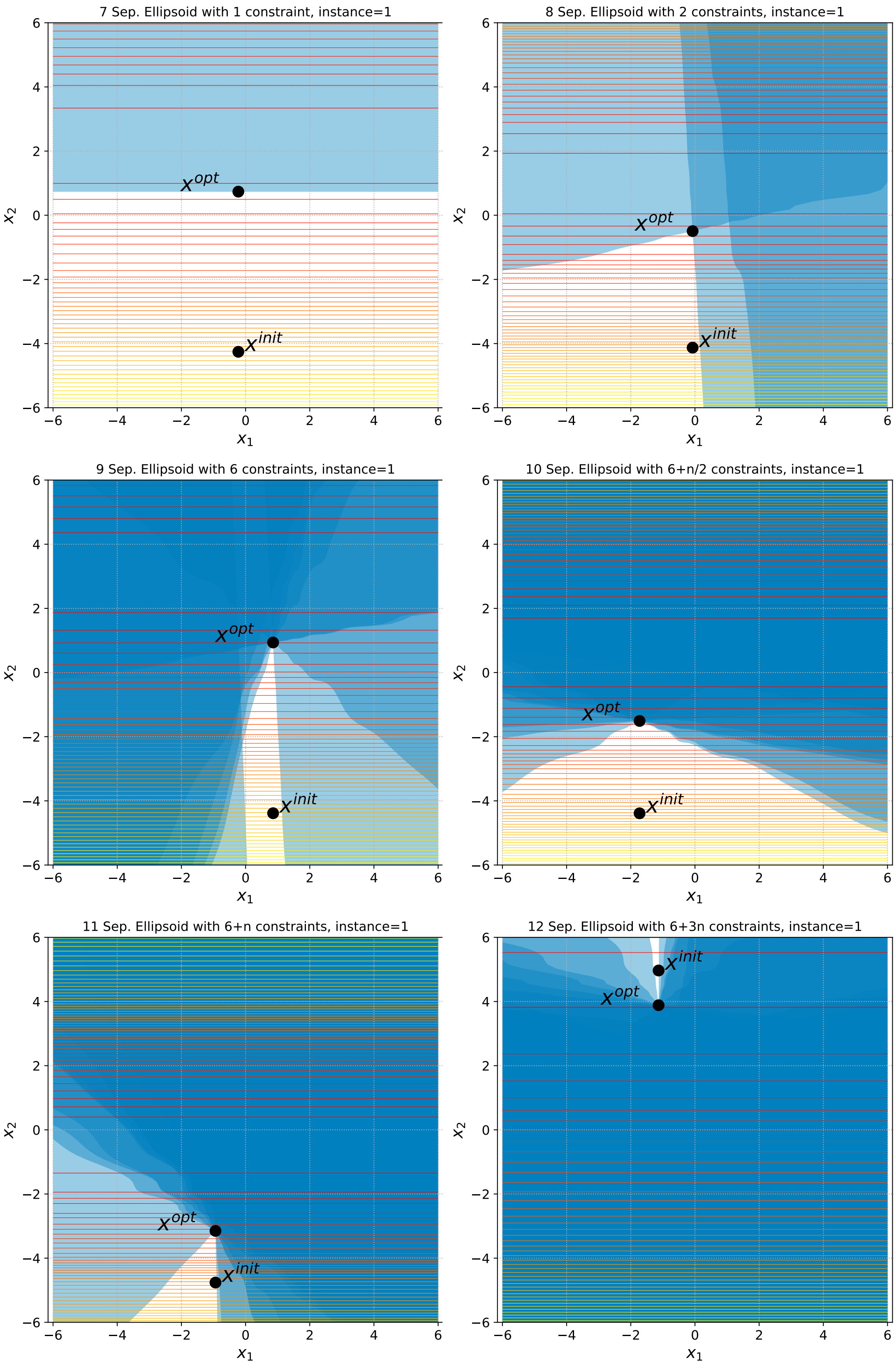
[4] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[5] Nikolaus Hansen, Dimo Brockhoff, Olaf Mersmann, Tea Tusar, Dejan Tusar, Ouassim Ait ElHara, Phillipe R. Sampaio, Asma Atamna, Konstantinos Varelas, Umut Batu, Duc Manh Nguyen, Filip Matzner, and Anne Auger. COmparing Continuous Optimizers: numbbo/COCO on Github, March 2019.

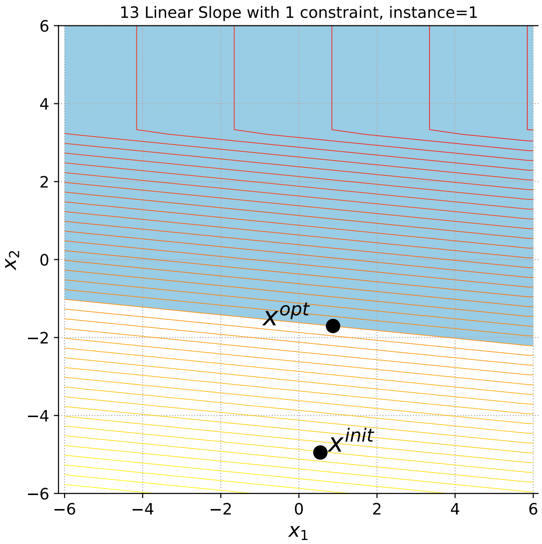
Sphere

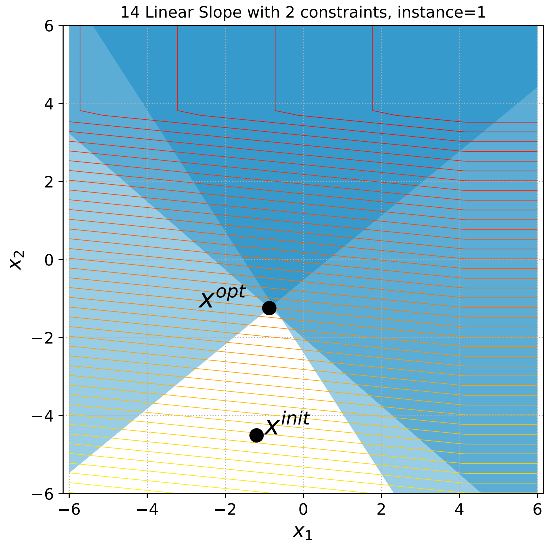

1 Sphere with 1 constraint, instance=1
2 Sphere with 2 constraints, instance=1
3 Sphere with 6 constraints, instance=1
4 Sphere with 6+n/2 constraints, instance=1
5 Sphere with 6+n constraints, instance=1
6 Sphere with 6+3n constraints, instance=1
x^{init}
x^{opt}
x_1
x_2

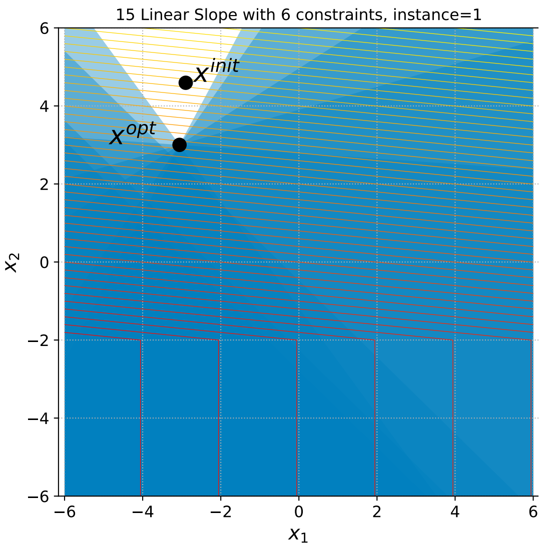
Ellipsoid

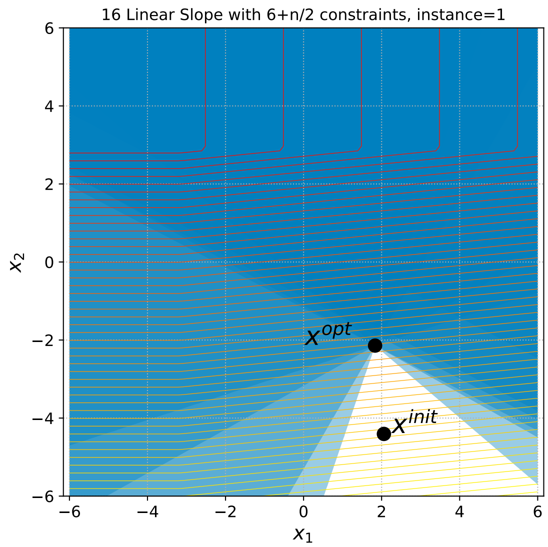

7 Sep. Ellipsoid with 1 constraint, instance=1
8 Sep. Ellipsoid with 2 constraints, instance=1
9 Sep. Ellipsoid with 6 constraints, instance=1
10 Sep. Ellipsoid with 6+n/2 constraints, instance=1
11 Sep. Ellipsoid with 6+n constraints, instance=1
12 Sep. Ellipsoid with 6+3n constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

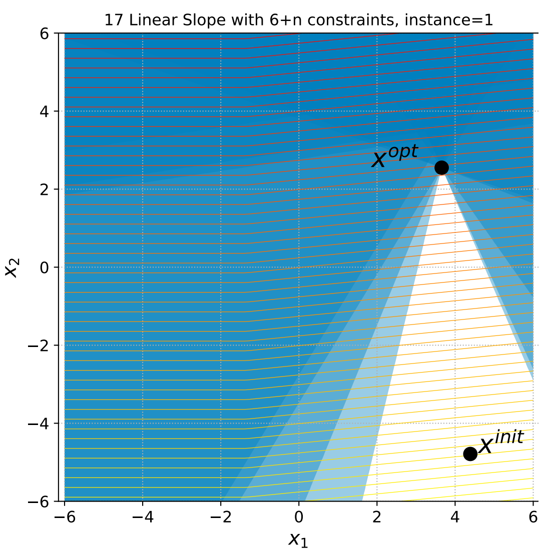
Linear Slope

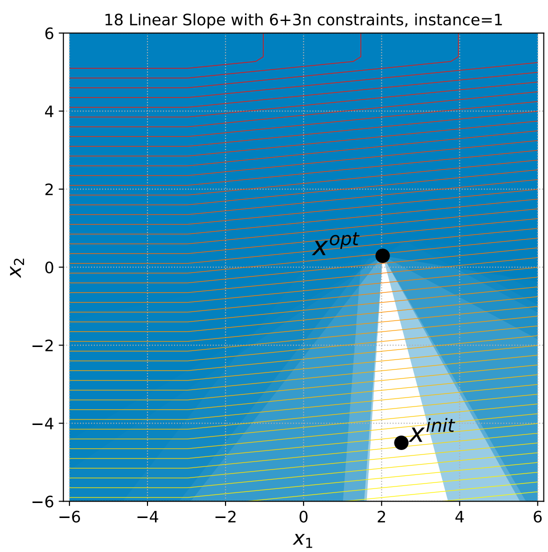






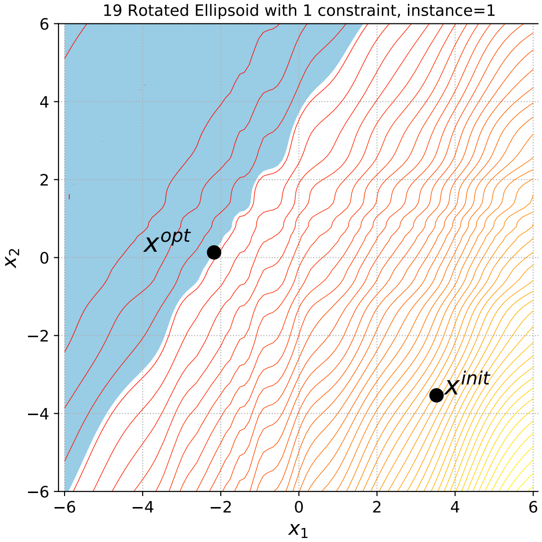




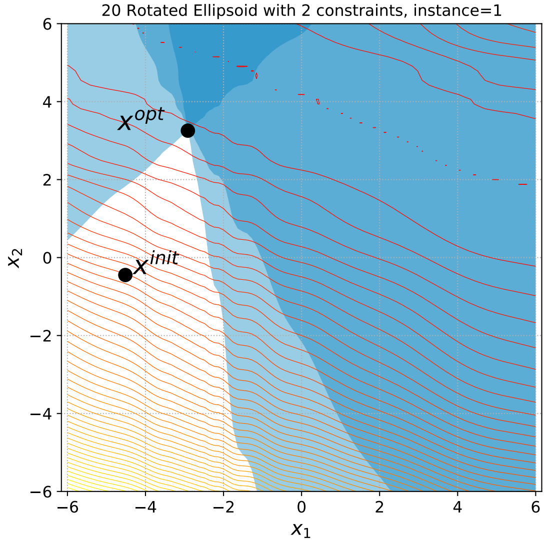


Rotated Ellipsoid

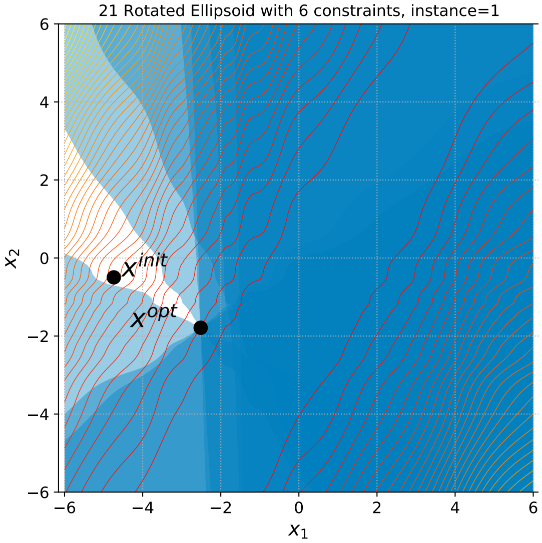
19 Rotated Ellipsoid with 1 constraint, instance=1



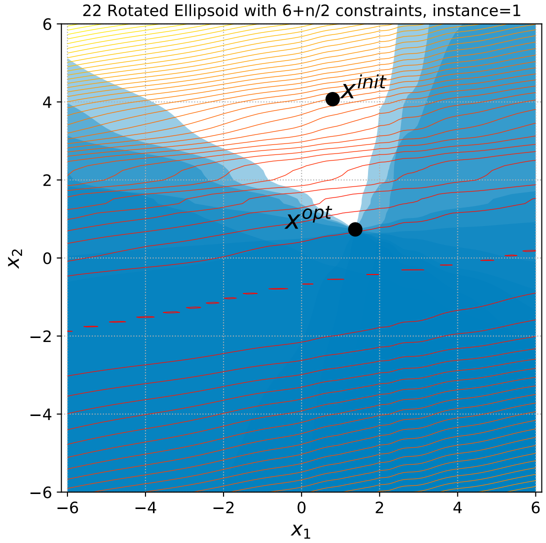
20 Rotated Ellipsoid with 2 constraints, instance=1



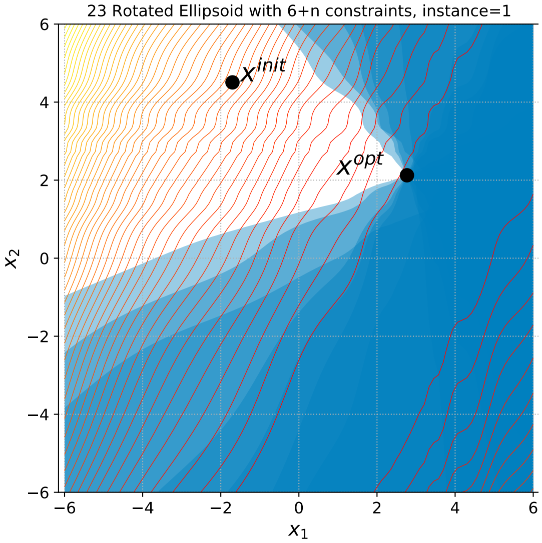
21 Rotated Ellipsoid with 6 constraints, instance=1



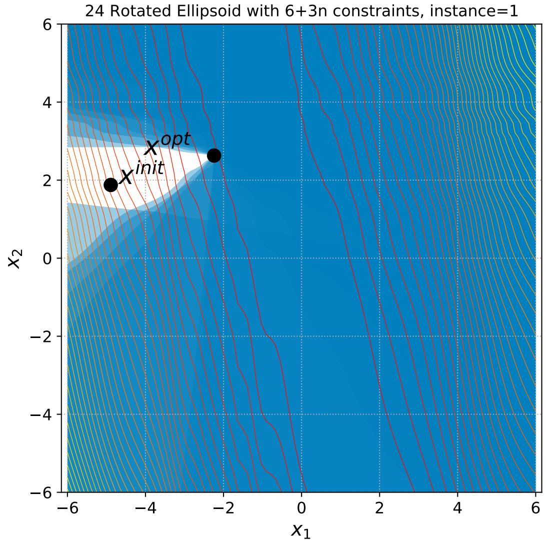
22 Rotated Ellipsoid with 6+n/2 constraints, instance=1



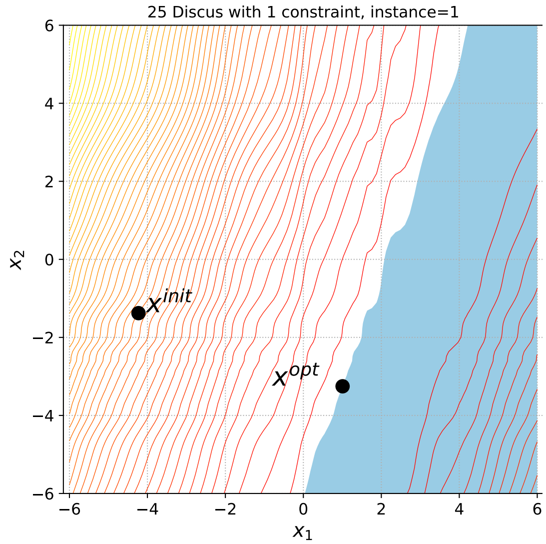
23 Rotated Ellipsoid with 6+n constraints, instance=1

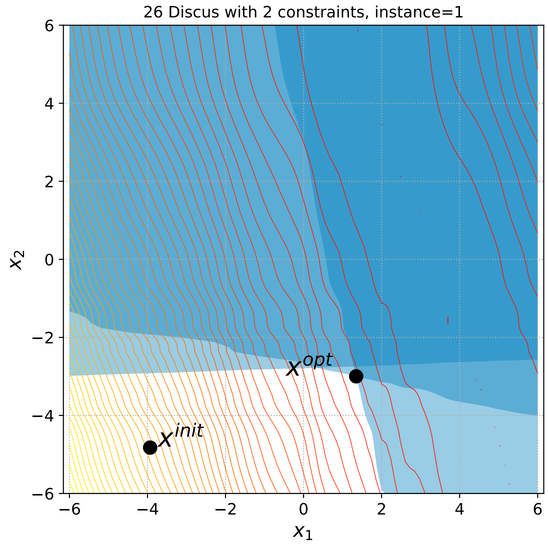


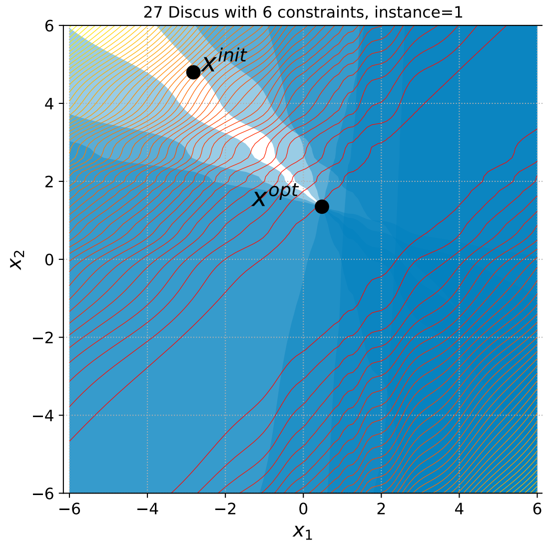
24 Rotated Ellipsoid with 6+3n constraints, instance=1

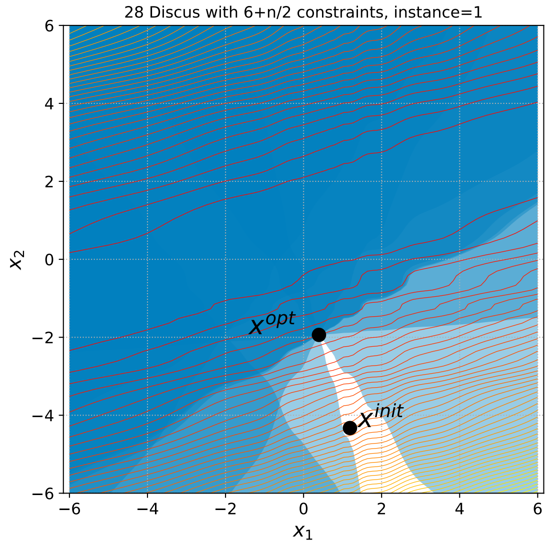


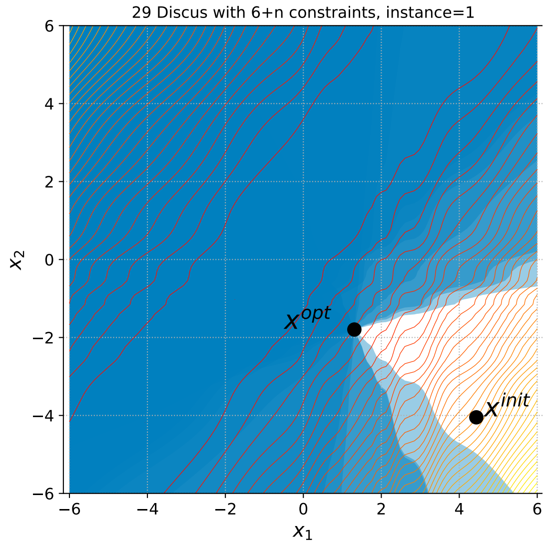
Discus

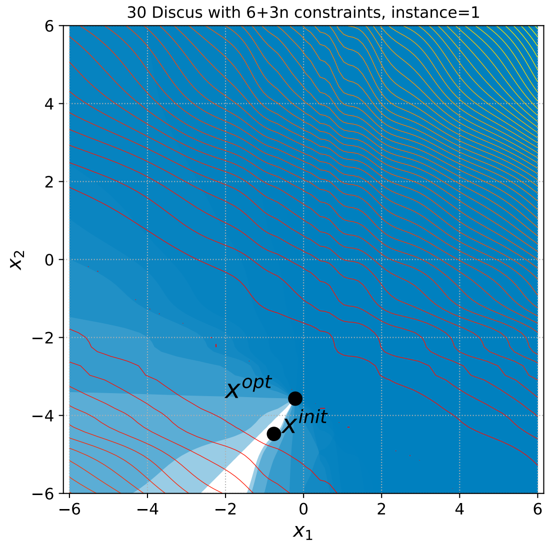

25 Discus with 1 constraint, instance=1
x^{init}
x^{opt}
x_1
x_2


26 Discus with 2 constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

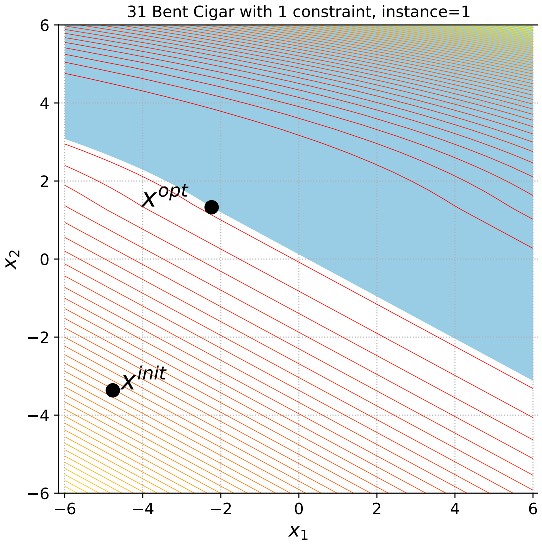

27 Discus with 6 constraints, instance=1
x^{init}
x^{opt}
x_1
x_2

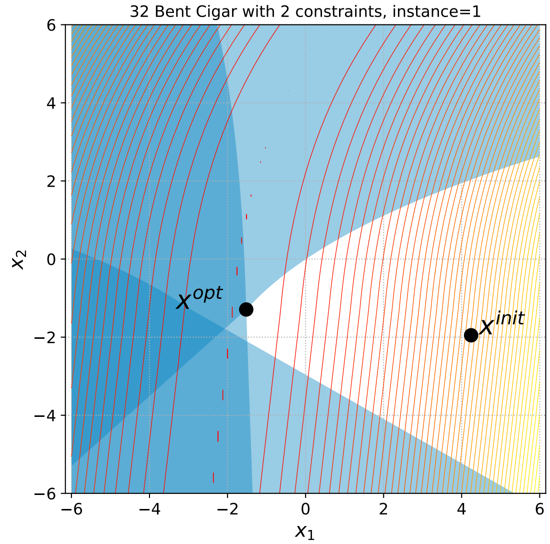

28 Discus with 6+n/2 constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

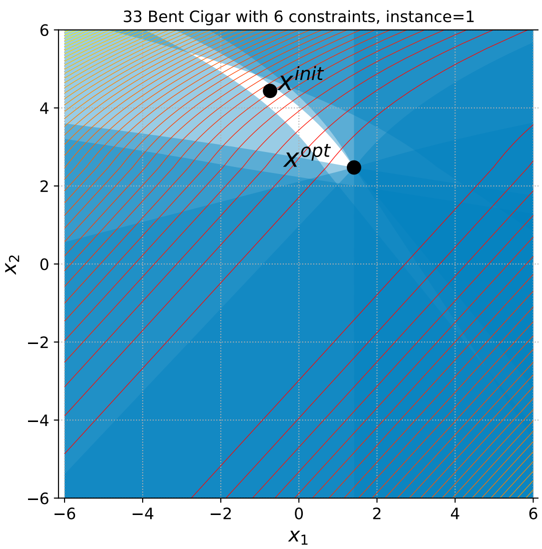

29 Discus with 6+n constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

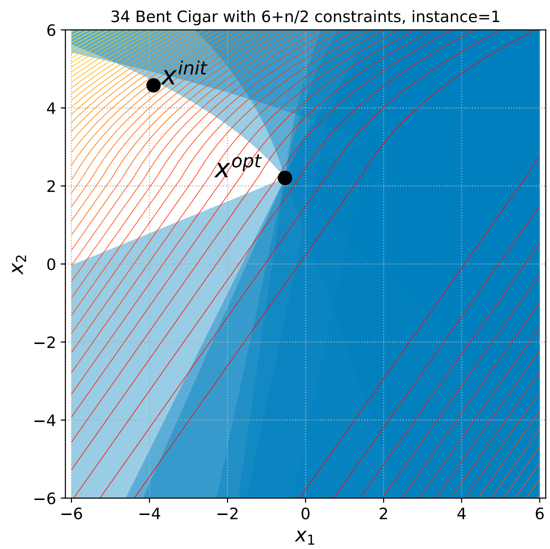

30 Discus with 6+3n constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

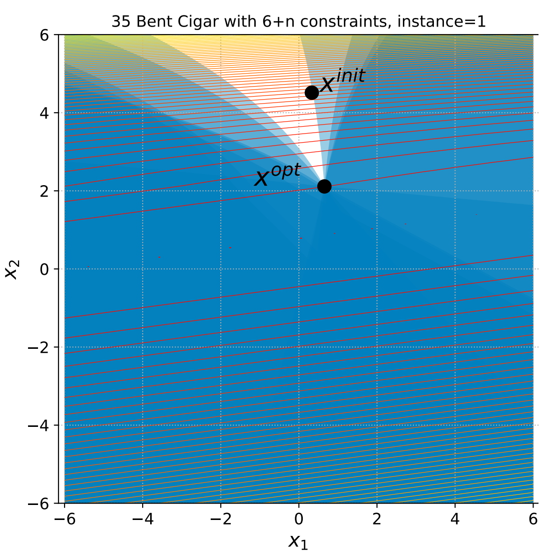
Bent Cigar

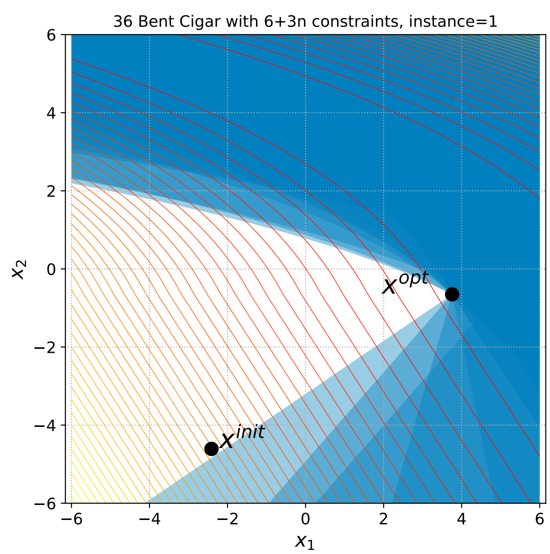

31 Bent Cigar with 1 constraint, instance=1
x^{opt}
x^{init}
x_1
x_2


32 Bent Cigar with 2 constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

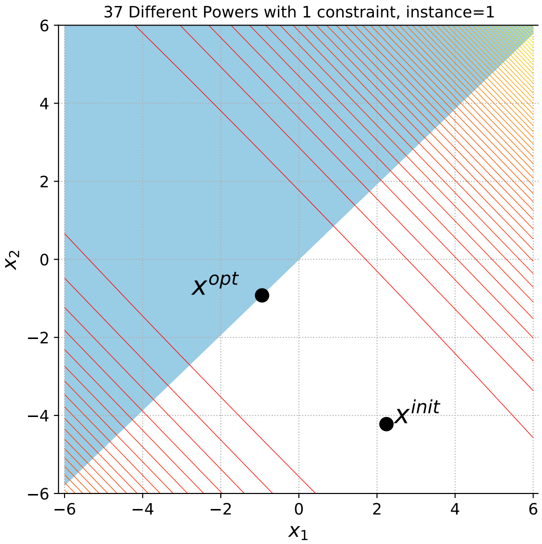

33 Bent Cigar with 6 constraints, instance=1
x^{init}
x^{opt}
x_1
x_2

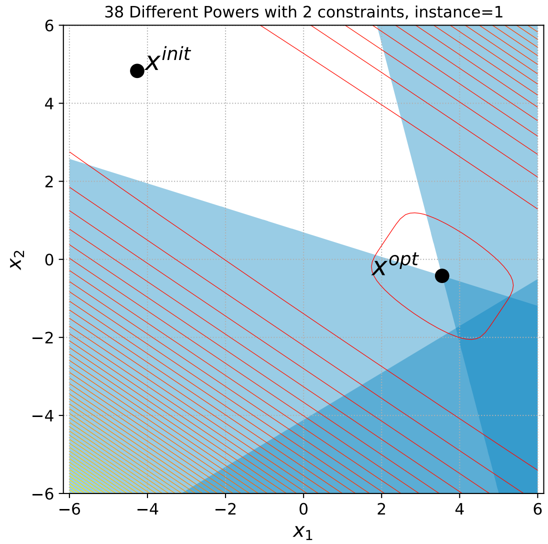

34 Bent Cigar with 6+n/2 constraints, instance=1
x^{init}
x^{opt}
x_1
x_2

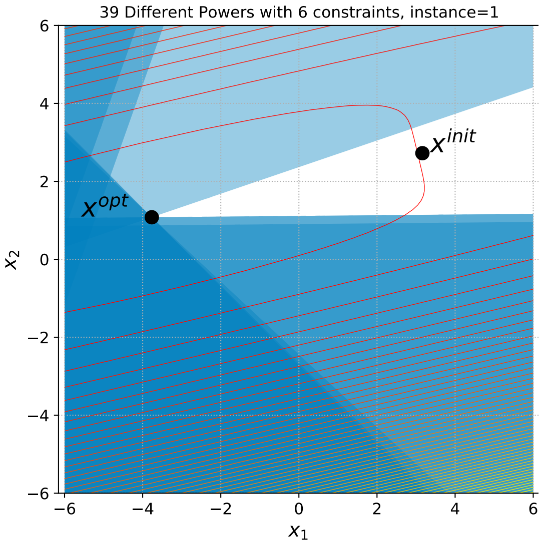

35 Bent Cigar with 6+n constraints, instance=1
x^{init}
x^{opt}
x_1
x_2

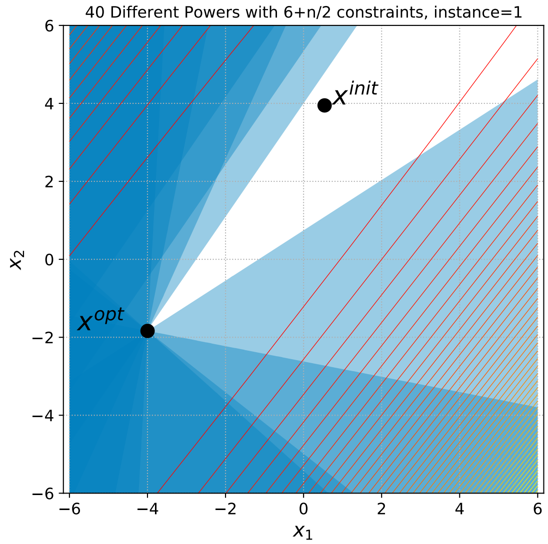

36 Bent Cigar with 6+3n constraints, instance=1
x^{opt}
x^{init}
x_1
x_2

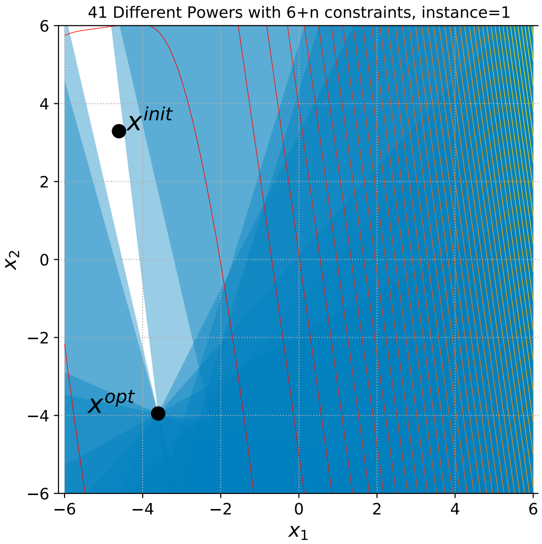
Different Powers

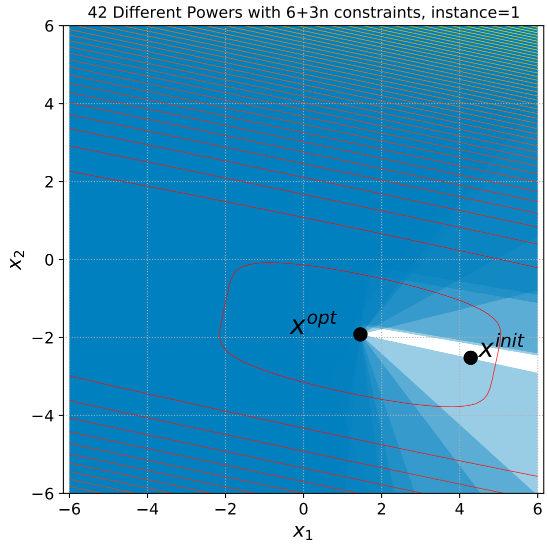




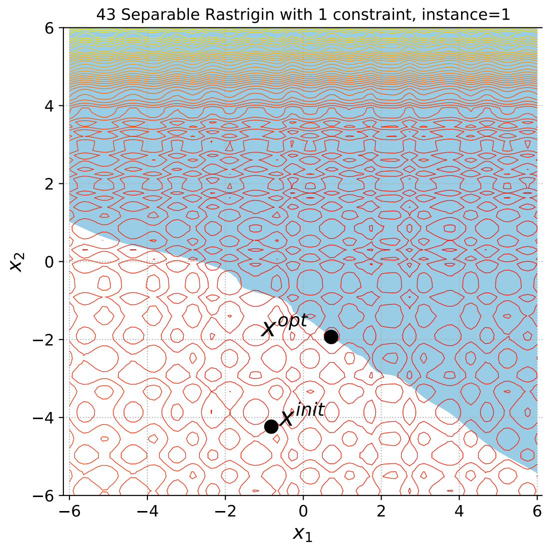


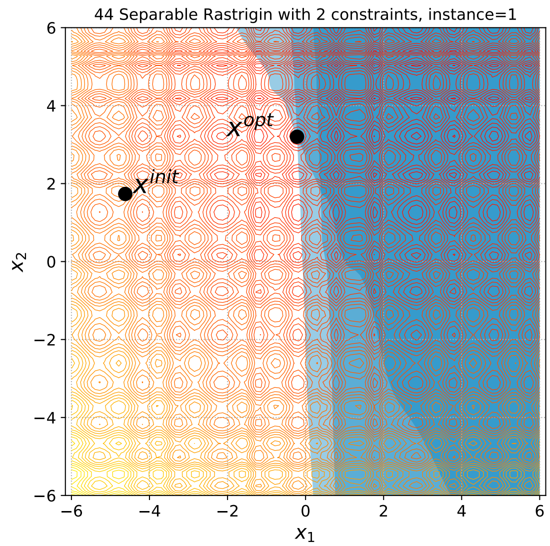


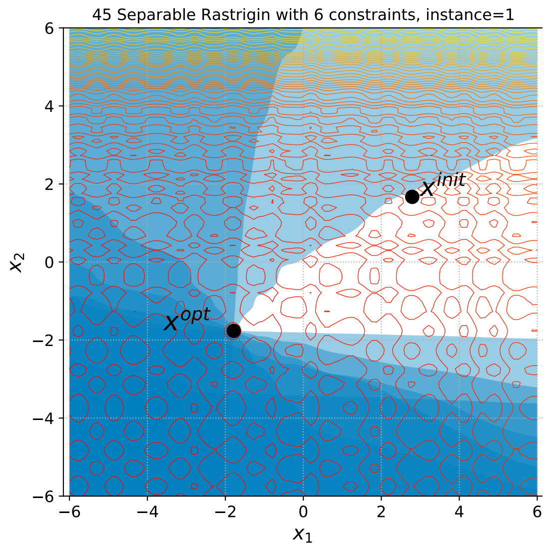


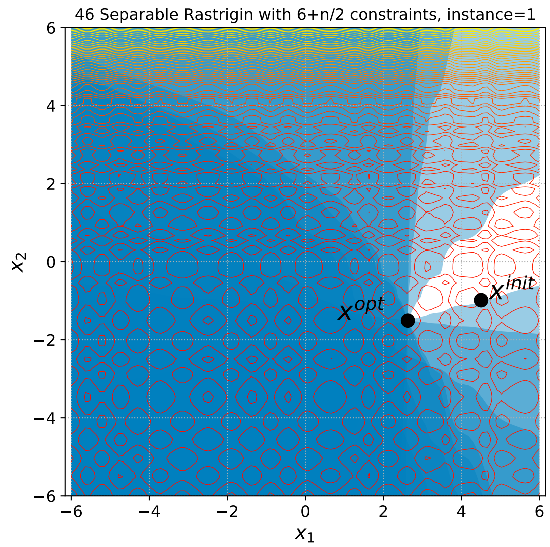


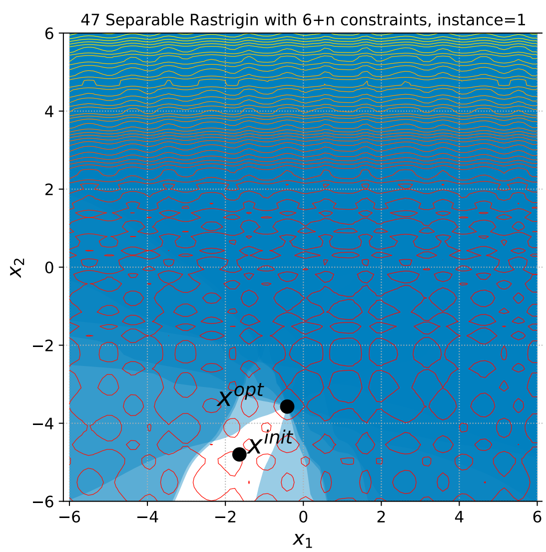
Rastrigin

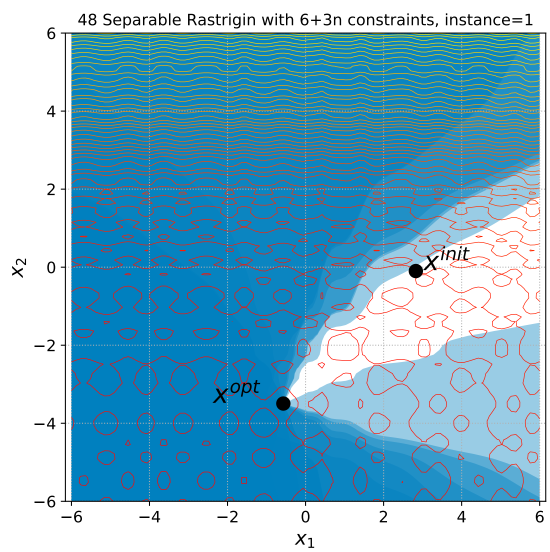

43 Separable Rastrigin with 1 constraint, instance=1
x_1
x_2


44 Separable Rastrigin with 2 constraints, instance=1
x_1
x_2


45 Separable Rastrigin with 6 constraints, instance=1
x_1
x_2


46 Separable Rastrigin with 6+n/2 constraints, instance=1
x_1
x_2


47 Separable Rastrigin with 6+n constraints, instance=1
x_1
x_2


48 Separable Rastrigin with 6+3n constraints, instance=1
x_1
x_2

Rotated Rastrigin