Risk-Averse "Wobbly Rosenbrock": CVaR-Minimization with Parameter Noise in a

October 21, 2025

Abstract

We formulate a minimal yet expressive risk-averse optimization problem based on the Rosenbrock function with uncertainty in the parameter a. The objective is to minimize the Conditional Value-at-Risk (CVaR) of the perturbed loss, using a scenario approximation that is directly solvable as a nonlinear program (NLP) in CasADi/IPOPT. This toy model serves as a compact entry point towards risk-averse optimal control, where parameter uncertainties (rather than decision perturbations) are the natural driver.

1 Model

The deterministic Rosenbrock function with parameters $a \in \mathbb{R}$, $b > 0$ is

$$\phi(x; a, b) \;=\; (a - x_1)^2 \;+\; b\,(x_2 - x_1^2)^2, \qquad x = (x_1, x_2)^\top. \tag{1}$$

We model uncertainty ξ_a in the parameter a ("wobbly a"). For a given decision x, the random loss is

$$f(x, \xi_a) \;=\; \phi(x; a + \xi_a,\, b). \tag{2}$$

Typical choices are $a = 1$, $b = 100$, with a simple zero-mean distribution for ξ_a, e.g. $\xi_a \sim \mathcal{N}(0, \sigma_a^2)$. A compact feasible set is imposed, e.g. $X = [-2, 2]^2$.

Risk measure. For confidence level $\alpha \in (0, 1)$, we minimize the CVaR of $f(x, \xi_a)$:

$$\min_{x \in X} \; \mathrm{CVaR}_\alpha\big[f(x, \xi_a)\big]. \tag{3}$$

Using the Rockafellar–Uryasev representation with an auxiliary threshold $t \in \mathbb{R}$,

$$\mathrm{CVaR}_\alpha[f] \;=\; \min_{t \in \mathbb{R}} \Big\{\, t \;+\; \tfrac{1}{1-\alpha}\,\mathbb{E}\big[(f - t)_+\big] \Big\}, \tag{4}$$

where $(\cdot)_+ = \max\{0, \cdot\}$.

2 Scenario Approximation (Sample Average Approximation)

Draw N i.i.d. samples $\{\xi_a^{(i)}\}_{i=1}^N$. The sample-based CVaR problem reads

$$\begin{aligned} \min_{x \in X,\, t \in \mathbb{R},\, u \in \mathbb{R}^N} \quad & t \;+\; \frac{1}{(1-\alpha)N} \sum_{i=1}^{N} u_i \\ \text{s.t.} \quad & u_i \;\geq\; \phi(x; a + \xi_a^{(i)},\, b) - t, \quad i = 1, \ldots, N, \\ & u_i \;\geq\; 0, \quad i = 1, \ldots, N, \\ & x \in X. \end{aligned} \tag{5}$$

This is convex in (t, u), but nonconvex in x due to the Rosenbrock term. It is nevertheless small-scale and well suited to standard NLP solvers (e.g. IPOPT).

Defaults for reproducibility. We recommend $a = 1$, $b = 100$, $X = [-2, 2]^2$, $\alpha \in \{0.9, 0.95, 0.99\}$, $N \in [100, 500]$, and $\sigma_a \in [0.02, 0.2]$. Use a fixed RNG seed for comparability. Compare the risk-averse solution to a risk-neutral baseline (minimize the sample mean $\frac{1}{N}\sum_i f(x, \xi_a^{(i)})$).

3 CasADi/IPOPT Reference Script

The following Python script sets up (5) with parameter noise in a, solves it with IPOPT, and prints the optimal decision and objective value. It also solves a risk-neutral baseline for comparison.

```python
# Requires: casadi>=3.6, numpy
import casadi as ca
import numpy as np

# --- Settings (feel free to edit) ---
a, b = 1.0, 100.0
alpha   = 0.95        # CVaR level
N       = 300         # number of scenarios
sigma_a = 0.05        # std-dev of noise in 'a'
rng     = np.random.default_rng(0)

# Feasible box for x = (x1, x2)
lbx = np.array([-2.0, -2.0])
ubx = np.array([ 2.0,  2.0])
x0  = np.array([-1.2, 1.0])  # classical Rosenbrock start

# --- Samples for parameter noise in 'a' ---
Xi = rng.normal(0.0, sigma_a, size=N)  # shape (N,)
# Xi = rng.standard_t(df=3, size=N) * sigma_a # for heavy tails

# --- Define Rosenbrock ---
def phi(x, a_eff, b):
    x1, x2 = x[0], x[1]
    return (a_eff - x1)**2 + b*(x2 - x1**2)**2

# --- Build CVaR NLP ---
x = ca.MX.sym('x', 2)
t = ca.MX.sym('t')
u = ca.MX.sym('u', N)

g = []  # constraints
for i in range(N):
    a_eff = a + Xi[i]
    fi = phi(x, a_eff, b)
    g.append(u[i] - (fi - t))  # u_i >= f - t
    g.append(u[i])             # u_i >= 0

obj = t + (1.0/((1.0 - alpha)*N))*ca.sum1(u)

# Pack variables and bounds
w  = ca.vertcat(x, t, u)
lbg = [0.0]*len(g)
ubg = [ca.inf]*len(g)
lbw = list(lbx) + [-ca.inf] + [0.0]*N
ubw = list(ubx) + [ ca.inf] + [ca.inf]*N
w0  = np.r_[x0, 1.0, np.zeros(N)]

```

```
nlp = {'x': w, 'f': obj, 'g': ca.vertcat(*g)}
solver = ca.nlpsol('s', 'ipopt', nlp, {
    'ipopt.print_level': 0,
    'print_time': 0,
})

sol = solver(x0=w0, lbx=lbw, ubx=ubw, lbg=lbg, ubg=ubg)
w_opt = np.array(sol['x']).squeeze()
x_cvar = w_opt[:2]
t_cvar = w_opt[2]
obj_cvar = float(sol['f'])

print("CVaR solution @ alpha=%.2f:" % alpha)
print("  x* =", x_cvar)
print("  t* =", t_cvar)
print("  CVaR objective =", obj_cvar)

# --- Risk-neutral baseline: minimize mean loss ---
x_rn = ca.MX.sym('x', 2)
mean_loss = 0
for i in range(N):
    a_eff = a + Xi[i]
    mean_loss = mean_loss + phi(x_rn, a_eff, b)
mean_loss = mean_loss / N

nlp_rn = {'x': x_rn, 'f': mean_loss}
solver_rn = ca.nlpsol('srn', 'ipopt', nlp_rn, {
    'ipopt.print_level': 0,
    'print_time': 0,
})
sol_rn = solver_rn(x0=x0, lbx=lbx, ubx=ubx)
x_rn_opt = np.array(sol_rn['x']).squeeze()
obj_rn = float(sol_rn['f'])

print("\nRisk-neutral baseline (mean loss):")
print("  x_RN =", x_rn_opt)
print("  mean loss =", obj_rn)

# --- Simple out-of-sample check ---
M = 5000
Xi_out = rng.normal(0.0, sigma_a, size=M)

def eval_stats(xval, Xi_samples):
    vals = np.array([phi(xval, a + z, b) for z in Xi_samples])
    return float(vals.mean()), float(np.quantile(vals, 0.99))

m_cvar, q99_cvar = eval_stats(x_cvar, Xi_out)
m_rn,   q99_rn   = eval_stats(x_rn_opt, Xi_out)

print("\nOut-of-sample (M=%d) stats:" % M)
print("  CVaR soln:  mean=%.6f,  q99=%.6f" % (m_cvar, q99_cvar))
print("  RN   soln:  mean=%.6f,  q99=%.6f" % (m_rn,   q99_rn))
```

Listing 1: CVaR-minimization for wobbly Rosenbrock with parameter noise in a.

This is the output

```
> python3 wobbly_rosenbrock_cvar_param_noise.py
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************

Nominal solution (deterministic Rosenbrock):
  x_nom = [0.99999999 0.99999999]   f(x_nom; a,b) = 2.9895642187051186e-17
  OOS (M=5000) mean=0.002474  CVaR@alpha=0.95 = 0.013252

CVaR-optimized solution (train N=300, alpha=0.95):
  x_cvar = [0.99513433 0.99029234]   t* = 0.009209538130616965   training objective (RU) = 0
  OOS (M=5000) mean=0.002497  CVaR@alpha=0.95 = 0.013466

Comparison (OOS): CVaR@alpha=0.95
  CVaR(x_cvar)  vs  CVaR(x_nom)  -->  0.013466  vs  0.013252
  Mean(x_cvar)  vs  Mean(x_nom)  -->  0.002497  vs  0.002474
```