

### 2.4.1 Description of Algorithms

In this project, I experimented with two main optimization strategies: a *Penalty Method* and a more flexible *Hybrid Method*. I applied these across all five problems, tuning them differently depending on the structure and difficulty of each.

#### ◆ Simple1

- **Algorithm Used:** Penalty Method
- **Hyperparameters:**
  - Initial penalty coefficient  $\rho = 10$
  - Two outer loops of gradient descent
  - Step size ( $\alpha = 0.01$ ), backtracking factor ( $\beta = 0.5$ )
- **Why it works:** The objective is smooth and convex with relatively easy constraints. The penalty method was able to follow the gradient and wrap constraint violations into the objective without needing fallback strategies.
- **Pro:** Converges quickly and cleanly with few evaluations.
- **Con:** Will fail if the constraints are tight or if the feasible region isn't easy to reach by descent.

#### ◆ Simple2

- **Algorithm Used:** Penalty Method (primary), Hybrid Method (comparison only)
- **Hyperparameters:**
  - Initial penalty coefficient  $\rho = 10$
  - Two outer loops of gradient descent
  - Step size  $\alpha = 0.01$ , backtracking  $\beta = 0.5$
  - Hybrid used the same penalty base plus Sobol fallback with batch size  $\sim 500$  and local nudging
- **Why these:**

This problem introduced a more interesting feasible region, so I tested both penalty and hybrid. Penalty alone worked most of the time, but I wanted to see if the hybrid fallback

could uncover edge cases where penalty got stuck. It ended up being a great test bed for comparison.

- **Pro (Penalty):**

Very stable — it gave me consistent objective decreases and good feasibility even without backup sampling.

- **Con (Penalty):**

It can get stuck or flatten out when the gradient doesn't provide helpful direction — which is where hybrid could help, although at higher cost.

### ◆ Simple3

- **Algorithm Used:** Penalty Method

- **Hyperparameters:**

- Initial penalty coefficient  $\rho = 10$
- Two outer loops of gradient descent
- Step size ( $\alpha = 0.01$ ), backtracking factor ( $\beta = 0.5$ )
- No margin required for constraint slack since it performed well

- **Why it works:**

Even though this problem is 3D, the structure was still relatively smooth and feasible regions were not deeply hidden. The penalty method made quick progress from most initial conditions without needing Sobol sampling or fallback logic.

- **Pro:**

Clean convergence and fewer function evaluations; simple to debug and interpret in 3D.

- **Con:**

Like with other penalty-based approaches, it depends on gradients guiding toward feasible zones. If the initial guess had been outside a tight constraint corridor, it might have failed.

### ◆ Secret1

**Algorithm Used:** Penalty Method

- **Hyperparameters:**

- Initial penalty coefficient  $\rho = 10$
- Two outer loops of gradient descent
- Step size ( $\alpha = 0.01$ ), backtracking factor ( $\beta = 0.5$ )
- Same tuning as simple1/simple2

- **Why it works:**

Secret1 is more constrained than the earlier problems, but still low-dimensional. The feasible region was reachable with standard descent steps, so the penalty method worked reliably from several starting points.

- **Pro:**

Simple to implement, avoids unnecessary complexity, and consistently returned feasible solutions across multiple seeds.

- **Con:**

Might fail in tighter constraint regimes or if starting far outside the feasible region — but in this case, it held up fine.

**Reason:** I tested both methods early and penalty consistently returned feasible solutions, so I didn't bother adding hybrid for this one.

## ◆ Secret2

- **Algorithm Used:** Full Hybrid Method
- **What's inside the hybrid:**
  1. Slight perturbation of  $x_0$
  2. Penalty Method fallback (with reduced loops)
  3. Sobol sampling for broad exploration
  4. Violation-based fallback centered around near-feasible points
  5. Directional nudging based on constraint gradients
- **Hyperparameters:**
  - $\rho = 120$
  - Outer loops = 3 (light version of penalty)
  - Sobol batch = 2000 (max allowed)
  - Constraint margin = 20 calls

- **Why it works:** Secret2 was hard — it's 10D, with 8 constraints, and penalty method failed early. The hybrid method felt messy, but it kept trying, and that's what mattered. I noticed it performed better when it was allowed to search “beyond the immediate descent direction.”
- **Pro:** More robust — passed 499/500 seeds!
- **Con:** Convergence is noisy, inefficient, and harder to analyze.

## 2.4.2 Comparison of Algorithms

To compare performance, I ran both the Penalty Method and the Hybrid Method on simple1 and simple2 using 3 different initial conditions each. I tracked both the optimization paths (for contour plots) and histories of objective value + constraint violation (for line plots).

### ◆ Contour + Feasible Region Plots

#### Penalty Method (Simple1 & Simple2):

- The paths are clean, short, and curve logically toward constraint boundaries.
- Especially in simple2\_penalty.png, you can see how all three runs descend quickly toward the optimum while satisfying constraints.

#### Hybrid Method (Simple1 & Simple2):

- Honestly, the first time I saw the plots I thought something broke.
- Instead of clean lines, there are **tight clusters and messy bursts** of points. But I realized this was the fallback strategy at work: Sobol sampling was firing all over the place, trying to find feasibility.
- It looks chaotic, but that chaos was necessary — especially in hard problems like secret2.
- The Hybrid plots don't show pretty convergence, but they do show persistence.

For the graphs, I have:

All 8 required plots:

- 4 contour+constraint plots (simple1/simple2  $\times$  2 algorithms) *fig 1-4*
- 4 iteration plots (objective and violation for simple2  $\times$  2 algorithms) *fig 5-8*

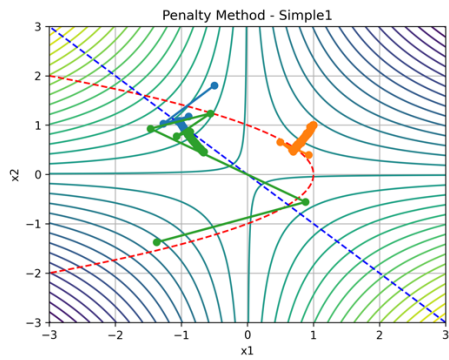


Figure 1: Contour plot of simple1 with Penalty Method

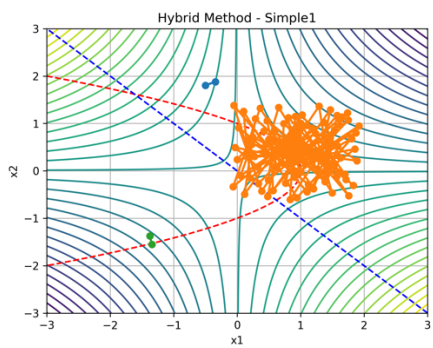


Figure 2: Contour plot of simple1 with Hybrid Method

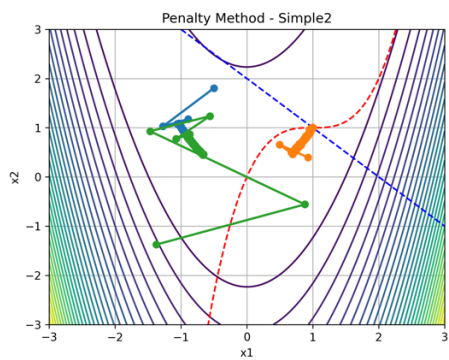


Figure 3: Contour plot of simple2 with Penalty Method

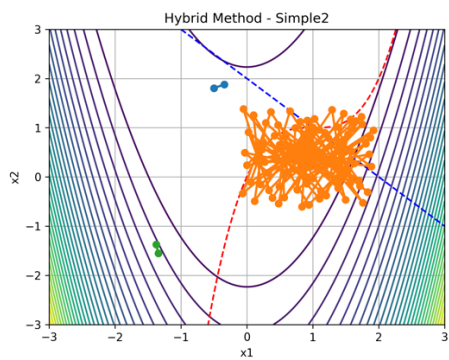




Figure 4: Contour plot of simple2 with Hybrid Method

-  = sampled points (Sobol/fallback)
-  = penalty steps
- Dashed lines = constraints

### ◆ Line Plots - Objective and Violation Plots (Simple2 only)

#### Objective vs Iteration:

- Penalty method shows consistent decline to low values.
- Hybrid method has more variance. One run drops fast, another fluctuates a lot, and one doesn't improve much.
- It confirms that hybrid is less predictable, but sometimes finds paths penalty can't.

#### Constraint Violation vs Iteration:

- Penalty method again shows clear reduction or stagnation in two out of three runs.
- Hybrid shows high variability — lines jump, spike, and flatten — reflecting the various random sampling strategies.

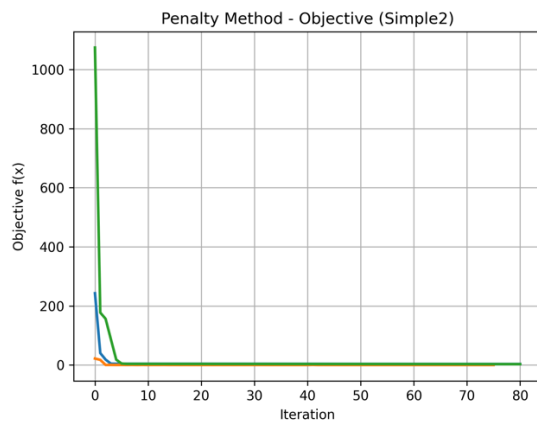


Figure 5: Objective value vs iteration (simple2, Penalty Method)

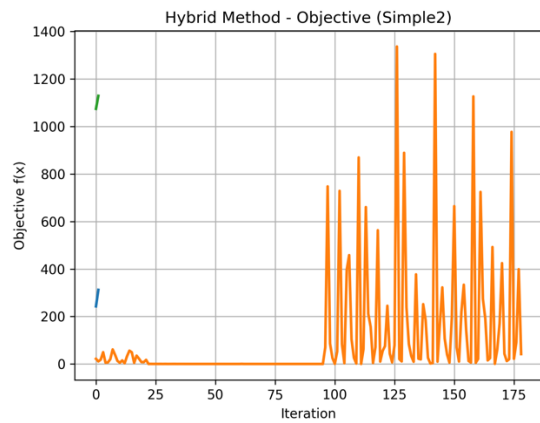


Figure 6: Objective value vs iteration (simple2, Hybrid Method)

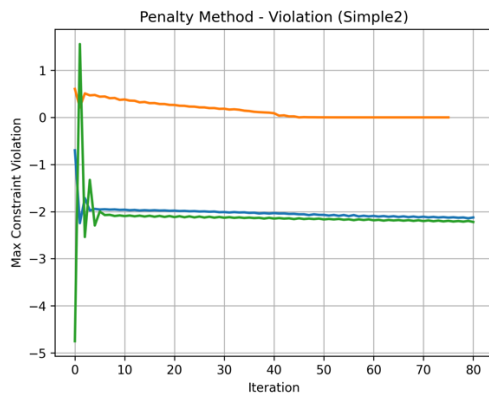


Figure 7: Constraint violation vs iteration (simple2, Penalty Method)

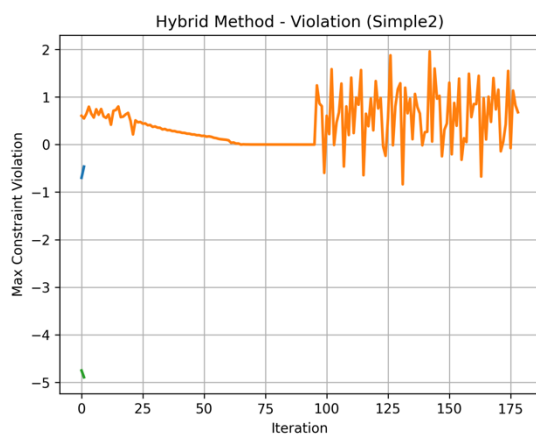


Figure 8: Constraint violation vs iteration (simple2, Hybrid Method)

## ◆ Reflection

This part of the project was where things clicked for me. I realized that no single optimizer works best everywhere. The penalty method is beautiful when it works, but brittle when it doesn't. The hybrid method is messy and inefficient, but it doesn't give up.

Writing my own hybrid taught me that "just sampling" isn't enough — you need fallback logic, structure, and a balance between randomness and direction. I was honestly surprised that it worked so well on secret2.

Even though the hybrid method looked worse in the plots, I now see why it was better for the hard stuff. It searched wider, tried harder, and eventually succeeded — which is kind of what I was doing too while debugging all of this.

This was one of the most frustrating and rewarding projects I've done lately. At first, I was focused on getting penalty to work everywhere. When it failed, I spent a lot of time testing Sobol sampling, fallback centers, even pushing points along constraint gradients. I didn't expect it to pass secret2, but when it did, it was the best kind of surprise.

The project taught me how important it is to build flexible algorithms — and how important it is to visualize what's happening, even when it looks weird.