Hybrid Metaheuristic for Parameter Estimation and Optimal Experimental Design

Methods and Step-by-Step Algorithmic Description

Your Project

October 16, 2025

Abstract

This document describes the optimization algorithm used in the repository, step by step. It covers the agent representation, parameter discretization, initialization, mutation and recombination operators, acceptance rules, scoring functions, and termination diagnostics for both parameter estimation and optimal experimental design (OED). Where helpful, we include pseudo-code that mirrors the implementation in `ParameterOptimizationModelSets.py` and `OptimizeExperiment.py`, together with utility routines from `OptimizationOperations.py` and `ExperimentOptimizationOperations.py`.

Contents

1 Problem Abstraction

The framework solves two related problems:

1. **Parameter Estimation:** pick parameter values θ that make simulations match data across one or multiple experiments ("measurements").
2. **Optimal Experimental Design:** pick time-varying control inputs $u(t)$ on a grid to maximize an information metric (e.g., D-optimal Fisher information) or an alternate utility.

Both are handled by a common *agent-based*, population-oriented heuristic over a discrete index space.

2 Representation: Discrete Index Space

Every optimizable quantity (parameter or control value) is mapped to a *grid of indices* $I = \{0, \ldots, K\}$. The numeric value is a lookup of an index in a pre-built grid (often logarithmic). The state of an *agent* is therefore a coordinate vector of indices:

$$\mathbf{i} = (i_1, i_2, \ldots, i_d) \in I^d \tag{1}$$

and a derived numeric vector $\theta(\mathbf{i})$. For OED, the representation extends to *time-blocked coordinates* — each control has a sequence of index assignments over non-overlapping windows:

$$\text{coordinates} = \{(t_0, t_1) \mapsto i,\ (t_1, t_2) \mapsto i, \ldots\}\,. \tag{2}$$

3 Scoring (Fitness) Functions

Let $\mathcal{S}(\theta)$ denote simulation of the model(s). A *scoring function* $J(\cdot)$ compares simulations to targets (for fitting) or computes an information metric (for OED). Implemented options include least-squares, standard-deviation-based aggregates, and Fisher/D-optimality (and ESS) for OED. The optimizer is *monotone improving*: a proposal is accepted only if it *strictly* improves J.

4 Population and Agent Structure

An **agent** stores:

- current coordinates and the mapped numeric vector;
- the current score and best-so-far score;
- a cached simulation (`simdata`) to avoid recomputation;
- a flag for sensitivity-aware moves (if forward sensitivities are available);
- for OED, a pulse sequence and a *pulsespace* (grid of control values).

5 Initialization (Multi-start)

1. Build a combined search space across all models/measurements, respecting shared parameters and any `fixed`/`include` masks.
2. Generate N initial samples (`startsamples`); either Monte Carlo or Sobol sampling (if enabled).

3. Select the top-A samples (where A is the agent count) by evaluating J and instantiate agents there.

Algorithm 1 Initialization

1: Build parameter grids $\{\mathcal{G}_k\}_{k=1}^{d}$
2: Sample $\{\mathbf{i}^{(s)}\}_{s=1}^{S}$ ▷ Monte Carlo or Sobol over indices
3: **for** $s = 1 \dots S$ **do**
4: Evaluate $J(\mathbf{i}^{(s)})$ via simulation
5: Pick A best samples, create agents $\{\mathcal{A}_a\}_{a=1}^{A}$ with those coordinates

6 Iteration Loop

Each generation performs mutation and occasional recombination on every agent, using *accept-if-better* replacement. A convergence monitor tracks best score and population diversity.

6.1 Mutation Operators

Mutations work in the discrete index space:

- **Local step:** pick a coordinate k and shift $i_k \leftarrow i_k + \delta$ with $\delta \in \{-r, \dots, +r\}$.
- **Global jump:** reassign i_k uniformly from $\{0, \dots, K\}$.
- **Directed step (optional):** bias the sign/magnitude of δ using forward sensitivities when available.
- **OED block mutation:** select a time window (or a run of windows) and move their indices left/right along the pulse grid; occasional reindexing for exploration.

Mutation counts per move are random in $[1, m_{\max}]$, and attributes to mutate are chosen from the allowed `include` set. A small global-jump probability (e.g., 0.25) maintains exploration.

6.2 Recombination

Periodically, an agent takes a *recombined* step towards the current fittest agent by mixing coordinates componentwise (centroidal/parental crossover in index space). This gently pulls underperformers into promising basins without collapsing diversity.

6.3 Evaluation and Acceptance

Every proposal is mapped to numeric values, simulated, and scored. The agent adopts the new coordinates *iff* the score improves. The best-so-far member (*fittest*) is tracked across the generation.

Algorithm 2 One Generation (per agent)

for agent $a \in \{1, \ldots, A\}$ **do**
  if rand() $< p_{\text{recomb}}$ **then**
    $\mathbf{i} \leftarrow \text{Recombine}(\mathbf{i}, \mathbf{i}_{\text{best}})$
  if rand() $< p_{\text{mut}}$ **then**
    $\mathbf{i} \leftarrow \text{Mutate}(\mathbf{i})$
  Evaluate $J(\mathbf{i})$
  if $J(\mathbf{i})$ improved **then**
    Accept and update caches
  else
    Revert
Update population best; update convergence statistics

7 OED-Specific Mechanics

OED extends the above with time-blocked coordinates for each control:

1. Build a *pulsespace*: the set of allowed control values (index grid) per control parameter.
2. Construct a pulse *time sequence* (fixed window length; configurable start time).
3. The agent's coordinates are a dict: `{control_name: {(t0,t1): idx, ...}}`.
4. Mutations operate on contiguous runs of windows, shifting them left or right, or randomizing their indices.
5. The simulator stitches segments, applying each window's control value.

8 Convergence and Termination

A convergence monitor keeps (i) the best score series, (ii) improvement plateaus, and (iii) optional diversity checks. Termination triggers include: maximum generations, a plateau over a fixed window, or diversity collapse (all agents nearly identical).

9 Sensitivity-Aware Moves

If forward sensitivities $\partial \mathbf{y}/\partial \theta$ are available from the solver, mutation ranges can be biased to parameters with larger influence on the score. This is realized by (i) weighting attribute selection and (ii) biasing step signs along the local gradient estimate in index space.

10 Practical Defaults and Hyperparameters

- **Agents (A):** 5–20 for light models; higher for rugged landscapes.
- **Generations (G):** 50–250 typical; use more when data are noisy or models stiff.
- **Mutation radius (r):** 1–3 index steps for local moves; global jump probability ≈ 0.1–0.3.
- **Recombination rate:** ≈ 0.2–0.3 encourages funneling without premature convergence.
- **Multi-start (S):** 50–1000 depending on dimensionality; Sobol for low-discrepancy coverage.

11 Complexity and Caching

The dominant cost is simulation; the method is *evaluation-efficient* through:

1. caching the current best simulation within each agent;
2. accepting only improvements (no extra simulations for rejected random walks);
3. sharing compiled models across agents.

12 Failure Modes and Remedies

- **Stagnation:** increase global-jump probability, increase agents/generations, widen bounds.
- **Over-exploration:** reduce global-jump rate, increase acceptance strictness (keep as "improve-only").
- **Oscillations (OED):** increase pulse window length or add mild smoothing to the pulse index grid.

13 Pseudocode Summary

Algorithm 3 Hybrid Optimization Loop

Initialize grids and build multi-start seeds
Select top-A seeds and create agents
for $g = 1 \dots G$ **do**
    for each agent a **do**
        if rand() $< p_{\text{recomb}}$ **then**
            $\mathbf{i}_a \leftarrow \text{Recombine}(\mathbf{i}_a, \mathbf{i}_\star)$
        if rand() $< p_{\text{mut}}$ **then**
            $\mathbf{i}_a \leftarrow \text{Mutate}(\mathbf{i}_a)$
        Evaluate $J(\mathbf{i}_a)$
        if improved **then**
            Accept; update a's caches
    Update best agent $\mathbf{i}_\star$; check convergence and stop if triggered
return best coordinates, score, and simulation

14 Extensibility Points

- **New scoring rules** (plug into the scoring module).
- **Custom mutations** (extend the agent's `mutate` / `random_mutation` hooks).
- **Alternate recombination** (replace the coordinate mixer function).
- **Constraint handling** (project after mutation or add penalty to J).

Notes on Implementation

This description corresponds to the classes and functions in:

- `ParameterOptimizationModelSets.Optimization` (population loop, recombination/mutation, acceptance);
- `OptimizeExperiment.ExperimentAgent` and `OptimizeExperiment.OptimizeInformation` (OED agent & driver);
- `OptimizationOperations.py` (mutation ranges, selection, weighted choices);
- `ExperimentOptimizationOperations.py` (pulse recombination and mutation utilities).