Developer Documentation

Unit Ball Integration Project

Your Name

December 9, 2025

Contents

1 Introduction

This document provides a developer-oriented overview of the **Unit Ball Integration Project**. It explains the mathematical foundations, internal organization, and algorithmic details needed to maintain, extend, or optimize the project.

The current codebase supports:

- Monte–Carlo sampling in arbitrary dimension d,
- direct uniform sampling inside B^d (no rejection),
- rejection sampling inside $[-1, 1]^d$ for validation,
- variance-reduced symmetrized Monte–Carlo using 2^d sign flips,
- deterministic spherical-coordinate integration in 3D,
- automated experiment scripts and diagnostics.

All numerical components are stateless and functional. Reproducibility is controlled using the `set_seed` utility.

2 Architecture Overview

- **src/** — numerical core (integration, sampling, utilities)
- **scripts/** — experiment orchestration and reproducible pipelines
- **tests/** — correctness, consistency, and regression tests
- **outputs/** — generated CSV result tables and plots

3 Mathematical Foundations

3.1 Volume of the d-Ball

The volume of a radius-R d-dimensional Euclidean ball is

$$\mathrm{Vol}(B^d(R)) = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2}+1\right)} R^d.$$

Implemented in:

```
utils.volume_ball(radius=1.0, d=3)
```

3.2 Direct Uniform Sampling in B^d

Implemented in `sample_uniform_ball_direct(N, d)`.

Algorithm:

1. Sample a Gaussian vector $Z \sim \mathcal{N}(0, I_d)$,
2. Normalize $u = Z/\|Z\|$ (uniform on the sphere),
3. Sample $U \sim \text{Uniform}(0, 1)$,
4. Set radius $r = U^{1/d}$,
5. Output $x = ru$ (uniform in B^d).

Advantages:

- rejection-free,
- $O(d)$ computational cost,
- stable even for high dimensions.

3.3 Rejection Sampling

Primarily used as a correctness check.

- Draw $x \sim \text{Uniform}([-1, 1]^d)$,
- Accept if $\|x\| \leq 1$.

Acceptance probability:

$$p_{\text{acc}} = \frac{\text{Vol}(B^d(1))}{2^d},$$

which becomes exponentially small in d. This is practical only for $d \leq 4$.

3.4 Monte–Carlo Estimator

Given samples $X_i \sim \text{Uniform}(B^d)$,

$$\widehat{I}_N = \text{Vol}(B^d) \cdot \frac{1}{N} \sum_{i=1}^{N} f(X_i).$$

$$\mathbb{E}[\widehat{I}_N] = I, \qquad \text{Var}(\widehat{I}_N) = O(N^{-1}).$$

3.5 Symmetrized Monte–Carlo (Variance Reduction)

For each $x \in \mathbb{R}^d$, define

$$Q_s x = (s_1 x_1, \ldots, s_d x_d), \qquad s_i \in \{\pm 1\}.$$

Estimator:

$$\bar{f}(x) = 2^{-d} \sum_{s \in \{\pm 1\}^d} f(Q_s x).$$

Total function evaluations:

$$n_{\text{eval}} = N \cdot 2^d.$$

This cancels all odd components of f exactly, yielding large variance reduction for nonlinear or asymmetric integrands.

For $d > 6$, full 2^d enumeration becomes expensive; the library supports restricting to a subset if desired.

4 Module-Level Documentation

4.1 src/utils.py

- `volume_ball(radius, d)`: Gamma-function formula.
- `f_from_points(X)`: assignment integrand (3D).
- `set_seed(seed)`: RNG reproducibility.
- `spherical_to_cartesian(r,theta,phi)`: coordinate transform.

4.2 src/integration.py

Implements the 3D deterministic integrator using spherical coordinates.

- uniform grids in r, θ, ϕ,
- product-rule (Riemann-sum) integration,
- complexity $O(m^3)$,
- used to compute reference solutions for error analysis.

4.3 src/montecarlo.py

Features:

- unified API:

```
mc_standard_estimator(f, N, d=3, sampling='direct')
symmetrized_estimator(f, N, d=3, sampling='direct')
mc_estimate_with_repeats(f, N_list, repeats, d, sampling)
```

- supports direct or rejection sampling,
- returns both the estimator and total evaluation count (critical for error vs. cost plots),
- includes helper routines to compute empirical variance.

5 Experiment Orchestration

`run_experiment.py` performs:

- deterministic 3D reference integral computation,
- Monte–Carlo experiments for selected N,
- standard vs. symmetrized MC comparisons,
- generation of CSV summaries,
- production of log–log error plots.

6 Diagnostics

`diagnostics.py` computes per-experiment summaries:

- mean estimator,
- absolute and relative errors,
- empirical variance and standard deviation,
- 95% confidence intervals,
- comparison between MC variants.

7 Testing

7.1 test_consistency.py

Ensures:

- MC on linear odd functions gives ≈ 0,
- symmetrized MC gives machine-precision zero,
- sampling invariants (shapes, domains).

7.2 test_general_d_sampling.py

Checks:

- correct shapes and dimensionality,
- $\|x\| \leq 1$ for all samples,
- rejection acceptance rates for $d = 2, 3, 4$.

8 Extending the System

8.1 Deterministic Integration

Potential improvements:

- Gauss–Legendre quadrature in r,
- adaptive local mesh refinement,
- sparse-grid quadrature for dimension reduction.

8.2 Monte–Carlo Enhancements

Future variance-reduction techniques:

- stratified sampling,
- Latin hypercube sampling,
- importance sampling,
- quasi-random Sobol or Halton sequences.

8.3 Parallelization

The system naturally supports:

- multiprocessing,
- `joblib` parallel batching,
- GPU evaluation via CuPy / PyTorch.

9 Conclusion

This document provides a complete developer-level overview of the numerical methods, architecture, and extensibility of the Unit Ball Integration Project. It serves as a reference for continued development and potential research extensions.