Section 6Experimental Setup and Benchmarking Protocol

December 25, 2025

1 Experimental Setup and Benchmarking Protocol

This section describes the simulation platform, performance metrics, benchmarking scenarios, and statistical validation methodology used to evaluate the seven SMC variants. All experiments designed for reproducibility and statistical rigor.

1.1 Simulation Platform

Software Environment:

[TABLE - See Markdown version for details]

Hardware Platform:

All simulations executed on standard workstation hardware to demonstrate feasibility for typical research environments: - CPU: Intel Core i7-10700K (8 cores, 3.8-5.1 GHz) or equivalent - RAM: 16 GB DDR4-3200 - Storage: NVMe SSD (for fast I/O during batch simulations) - GPU: Not utilized (CPU-only NumPy for portability)

Operating System: Ubuntu 22.04 LTS / Windows 11 (cross-platform validated)

Simulation Parameters:

[TABLE - See Markdown version for details]

Rationale for Time Step:

The simulation time step $\Delta t = 0.01$ s chosen based on: - Nyquist Criterion: Sample at ¿2x highest system frequency. DIP natural frequencies $\omega n \approx 2\pi \times 5$ rad/s -¿ minimum sample rate 10 Hz. Using 100 Hz provides 10x safety margin. - Control Bandwidth: SMC switching frequency typically 10-50 Hz (Section 7.3). Using 100 Hz control rate captures switching dynamics without aliasing. - Real-Time Feasibility: Control law compute times 18.5-31.6 mus (Section 7.1) ¡¡ 10 ms time step, leaving 99.7-99.8percent CPU headroom. - Numerical Accuracy: Euler integration error $\mathcal{O}(\Delta t^2)$ negligible for $\Delta t = 0.01$ s; validated by comparing to RK45 (adaptive) results (maximum state difference ¡10^{-5}).

Reproducibility Measures:

- Fixed Random Seeds: All stochastic elements seeded with ‘seed=42‘ - Version Pinning: All package versions specified in ‘requirements.txt‘ with exact pinning (e.g., ‘numpy==1.24.3‘) - Configuration Management: Single ‘config.yaml‘ file version-controlled with git - Data Archival: All simulation outputs saved to ‘benchmarks/results/‘ with SHA256 checksums

—

1.2 Performance Metrics

This section defines the 10+ quantitative metrics used to evaluate controller performance across multiple dimensions. Metrics divided into five categories: computational efficiency, transient response, chattering, energy, and robustness.

Category 1: Computational Efficiency

- Control Law Compute Time ($t$compute):

Wall-clock time to execute control law computation (Python ‘time.perf counter()‘ high-resolution timer). Measured per time step, averaged over 1000-step simulation. Reported with 95percent confidence interval via bootstrap.

Physical Interpretation: Determines real-time feasibility. For 10 kHz control loop (100 mus period), $t$compute $<$ 50 mus required (50percent duty cycle budget).

- Memory Usage (Mpeak):

Peak memory consumption during simulation (Python ‘tracemalloc‘ profiler). Relevant for embedded systems with limited RAM (e.g., ARM Cortex-M7 with 512 kB SRAM).

—

Category 2: Transient Response

- Settling Time (ts):

Time for system state to enter and remain within 2percent of equilibrium. 2percent criterion standard in control engineering [ref68]. Lower values indicate faster convergence.

Computation: For each simulation, scan state trajectory forward until $\|\mathbf{x}(t)\| \leq \epsilon\|\mathbf{x}0\|$ satisfied for all remaining time (no re-entry to large-error region). Report mean and standard deviation across Monte Carlo trials.

- Overshoot (OS):

Maximum%age deviation of pendulum angles beyond initial perturbation. Computed separately for $\theta 1, \theta 2$; reported as maximum across both angles. Target: OS ¡ 10percent (standard second-order system spec).

Physical Significance: Large overshoot risks: - Violating linearization assumptions ($|\theta i| > 0.1$ rad invalidates small-angle approximation) - Actuator saturation (large corrective forces during overshoot) - Reduced stability margins

- Rise Time (tr):

Time for system to traverse from 10percent to 90percent of steady-state value. Characterizes initial response speed (distinct from settling time, which includes oscillations).

—

Category 3: Chattering Characteristics

- Chattering Index (CI):

Root-mean-square control derivative (control slew rate). Higher values indicate more rapid control switching (chattering). Units: N/s (force rate for DIP actuator).

Interpretation: - CI $<$ 50 N/s: Low chattering (smooth control, minimal actuator wear) - 50 $\leq$ CI $<$ 200 N/s: Moderate chattering (acceptable for industrial actuators) - CI $\geq$ 200 N/s: High chattering (risk of actuator damage, acoustic noise)

- Peak Chattering Frequency ($f$chatter):

Dominant frequency in control signal above 10 Hz threshold (FFT analysis). Identifies switching frequency characteristic of boundary layer or sign function approximation.

Computation: Apply FFT to control signal $u(t)$, compute single-sided magnitude spectrum, find peak in range [10 Hz, Nyquist frequency = 50 Hz]. Report frequency and amplitude of peak.

- High-Frequency Energy Fraction (EHF):

Percentage of control signal energy at frequencies ¿10 Hz. Complements peak frequency metric by quantifying total high-frequency content.

—

Category 4: Energy Efficiency

- Total Control Energy (Ectrl):

Integrated squared control effort. Proportional to electrical energy consumed by actuator (assuming $P = u^2/R$ for resistive load). Lower values indicate more efficient control.

Typical Values for DIP System: - Optimal (STA SMC): 11.8 J - Moderate (**Classical SMC**): 12.4 J (+5percent) - High (**Adaptive SMC**): 13.6 J (+15percent)

- Peak Control Power (Ppeak):

Maximum instantaneous control force. Determines actuator sizing requirements. Constraint: Ppeak $\leq u$max $= 20$ N (actuator limit from Section 2).

—

Category 5: Robustness (Additional Metrics)

- Model Uncertainty Tolerance (Δtol):

Maximum%age parameter perturbation before instability (bisection search). Evaluated for masses, lengths, inertias. Higher values indicate better robustness (Section 8.1).

- Disturbance Attenuation Ratio (Adist):

Percentage reduction in maximum state deviation under sinusoidal disturbances. Target: Adist $>$ 80 for robust control (Section 8.2).

—

Metric Summary Table:

[TABLE - See Markdown version for details]

—

1.3 Benchmarking Scenarios

Monte Carlo Statistical Framework:

All controllers evaluated using Monte Carlo simulations to quantify performance variability and enable statistical comparison. Each benchmark scenario consists of Ntrials independent simulations with randomized initial conditions.

Scenario 1: Nominal Performance Benchmark (QW-2 Task)

Purpose: Establish baseline performance under small perturbations representative of measurement noise or minor disturbances.

Initial Conditions: Random uniform sampling within bounds

Number of Trials: Ntrials = 400 (100 per controller x 4 controllers)

Rationale: 400 trials provides: - 95percent confidence interval width $\approx 0.1\sigma$ (standard error $\sigma/\sqrt{400} = 0.05\sigma$) - Statistical power ¿0.8 for detecting 20percent effect size differences (power analysis via GPower) - Sufficient samples for non-parametric tests (bootstrap, permutation)

Scenario 2: Large Perturbation Stress Test (MT-7 Task)

Purpose: Evaluate controller robustness to realistic disturbances (6x larger than nominal).

Initial Conditions:

Number of Trials: Ntrials = 500 (50 per controller x 10 random seeds for seed sensitivity analysis)

Outcome: Severe generalization failure for PSO-tuned controllers (Section 8.3). Highlighted critical need for multi-scenario optimization.

Scenario 3: Model Uncertainty Sweep (LT-6 Task - Partial)

Purpose: Assess robustness to parametric uncertainty in physics model.

Parameter Perturbations: Each mass, length, inertia perturbed by ± 10 and ± 20:

Combinations: Full factorial sweep (5 perturbation levels x 8 parameters = $5^8 \approx$ 390,625 combinations, reduced via Latin Hypercube Sampling to 1000 samples)

Status: Blocked - Default gains produce 0percent convergence even at nominal parameters. Requires PSO tuning prerequisite (Section 8.1).

Scenario 4: Sinusoidal Disturbance Rejection (MT-8 Task - Partial)

Purpose: Evaluate active disturbance rejection capability.

Disturbance Model:

Initial Conditions: Nominal small perturbations (± 0.05 rad)

Trials per Frequency: 100 (total 400 per controller)

Metric: Disturbance attenuation ratio $A\text{dist}$ (Metric 12)

—

Statistical Sampling Strategy:

Random Number Generation: - Global seed: ‘seed=42‘ for NumPy default RNG - Independent draws: Each Monte Carlo trial uses independent random draw from $\mathcal{U}(-\theta\text{max}, +\theta\text{max})$ - Quasi-random sequences (optional): Sobol sequences for uniform space-filling in high-dimensional parameter sweeps (LT-6 scenario)

Sample Size Justification:

Power analysis (GPower 3.1): - Effect size: Cohen's $d = 0.5$ (medium effect, 10percent performance difference) - Significance level: $\alpha = 0.05$ (95percent confidence) - Desired power: $1 - \beta = 0.8$ (80percent probability of detecting true effect) - Required sample size: $n = 64$ per group (Welch's t-test, two-tailed)

Chosen sample sizes (100-500) exceed minimum requirements by 1.5-8x, ensuring robust conclusions.

—

1.4 Validation Methodology

Statistical Hypothesis Testing:

All performance comparisons validated using rigorous statistical tests with pre-specified significance level $\alpha = 0.05$ (95percent confidence).

Primary Test: Welch's t-test (Two-Sample Unequal Variance)

where $(\bar{X}i, si, ni)$ are sample mean, standard deviation, and size for group i.

Rationale: - Welch's t-test more robust than Student's t-test when variances unequal ($s1^2 \neq s2^2$) - Does not assume equal sample sizes ($n1 \neq n2$ permitted) - Approximately normal for $n \geq 30$ (Central Limit Theorem applies for our sample sizes 100-500)

Decision Rule: - Reject null hypothesis $H0 : \mu1 = \mu2$ if $p < 0.05$ - Interpret as: "Controller 1 and Controller 2 have statistically different performance"

Multiple Comparisons Correction:

When comparing $k = 4$ controllers (all pairwise comparisons: $\binom{4}{2} = 6$ tests), apply Bonferroni correction:

Reject $H0$ only if $p < 0.0083$. Controls family-wise error rate (FWER) at 5percent.

Effect Size Analysis (Cohen's d):

Statistical significance ($p < 0.05$) does not imply practical significance. Always report effect size:

Interpretation (Cohen's conventions): - $|d| < 0.2$: Negligible effect (not practically significant) - $0.2 \leq |d| < 0.5$: Small effect - $0.5 \leq |d| < 0.8$: Medium effect - $|d| \geq 0.8$: Large effect (practically significant)

Example from Results: STA vs **Classical SMC** settling time comparison: - $\bar{t}s, \text{STA} = 1.82$ s, $\bar{t}s, \text{Classical} = 2.15$ s - $p < 0.001$ (highly significant) - $d = 2.14$ (very large effect, 2.1 standard deviations apart)

Confidence Intervals (Bootstrap Method):

For each performance metric, compute 95percent confidence interval via bias-corrected accelerated (BCa) bootstrap:

- Resample with replacement: Generate $B = 10{,}000$ bootstrap samples from original data - Compute metric for each bootstrap sample: $\{\hat{\theta}1, \ldots, \hat{\theta}B\}$ - Sort bootstrap distribution and extract 2.5th and 97.5th%iles - Apply bias correction (BCa adjustment for skewed distributions)

Advantages over parametric CIs: - No distributional assumptions (robust to non-normality) - Accurate for skewed metrics (e.g., chattering index, which is bounded at zero) - Accounts for sampling uncertainty

Reporting Format: Mean ± SD [95percent CI] - Example: $ts = 1.82 \pm 0.15$ [1.78, 1.87] s

Non-Parametric Tests (Robustness Checks):

When data violate normality assumptions (Shapiro-Wilk test $p < 0.05$), use non-parametric alternatives: - Mann-Whitney U test: Non-parametric equivalent of t-test (ranks-based) - Kruskal-Wallis H test: Non-parametric ANOVA for ¿2 groups - Permutation tests: Exact significance via random permutations (computationally intensive, used when $n < 30$)

Reproducibility and Data Archival:

All statistical analyses satisfy FAIR principles (Findable, Accessible, Interoperable, Reusable):

- Raw Data: All simulation outputs saved to 'benchmarks/results/¡task id¿/raw data.csv' with SHA256 checksums - Analysis Scripts: Statistical analysis code version-controlled in 'src/analysis/validation/statist tests.py' - Figures: All plots generated programmatically via 'matplotlib' scripts in 'src/analysis/visualization/' - Configuration: Single source of truth: 'config.yaml' specifying all simulation parameters - Environment: Docker container or Conda environment file ('environment.yml') for exact package version replication

Open Science Commitment:

Upon publication, full dataset and analysis code will be released under MIT license on GitHub repository [GITHUB LINK]. This enables independent verification, extension, and replication by other researchers.

—

1.5 Disturbance Rejection Protocol

Real-world control systems must maintain performance under external disturbances (e.g., wind gusts, payload variations, sensor noise). This subsection describes the disturbance rejection testing protocol used to evaluate SMC robustness beyond nominal performance.

Motivation:

Standard benchmarking (Section 6.3) evaluates controllers under ideal conditions (no external forces). However, practical deployment requires: - Transient Disturbances: Step changes, impulses (e.g., collisions, actuator faults) - Periodic Disturbances: Sinusoidal forces (e.g., vibration, harmonic excitation) - Stochastic Disturbances: Random noise (e.g., sensor errors, environmental uncertainty)

Failure to test under disturbances can lead to catastrophic performance degradation in deployment [69, 70].

Disturbance Model:

External disturbances modeled as additive forces to the cart control input:

where $u$nominal(t) is the controller output and $d(t)$ is the external disturbance force (N). This models physical scenarios like: - Wind gusts: Step or sinusoidal forces - Payload drops: Impulse forces - Ground vibration: Random Gaussian noise

Disturbance Scenarios:

Primary Test Set (Robust PSO Optimization):

Used to optimize controller gains for disturbance rejection via Particle Swarm Optimization (Section 5):

- Step Disturbance: $d(t) = 10.0$ N for $t \geq 2.0$ s (constant force after t=2s) - Impulse Disturbance: $d(t) = 30.0$ N for $t \in [2.0, 2.1]$ s (brief spike)

Robust Fitness Function:

where: - Jnominal: Cost under nominal conditions (no disturbance) - Jdisturbed: Average cost under step and impulse disturbances - Cost function: $J = w1ts + w2\text{OS} + w3E$control (Section 6.2)

Rationale: Balancing nominal and disturbed performance prevents over-fitting to either scenario. Pure nominal optimization yields controllers that fail under disturbances (Section 8.4).

Extended Test Set (Generalization Validation):

To evaluate generalization beyond the PSO fitness function, additional disturbance types tested:

- Sinusoidal Low: $d(t) = 5.0 \sin(2\pi \cdot 0.5 \cdot (t-1))$ N for $t \geq 1.0$ s (0.5 Hz, sub-resonant) - Sinusoidal Resonant: $d(t) = 8.0 \sin(2\pi \cdot 2.0 \cdot (t-1))$ N for $t \geq 1.0$ s (2 Hz, near-resonant) - Sinusoidal High: $d(t) = 3.0 \sin(2\pi \cdot 5.0 \cdot (t-1))$ N for $t \geq 1.0$ s (5 Hz, super-resonant) - Random Gaussian (Low): $d(t) \sim \mathcal{N}(0, 2.0^2)$ N for $t \geq 1.0$ s - Random Gaussian (Mid): $d(t) \sim \mathcal{N}(0, 3.0^2)$ N for $t \geq 1.0$ s - Random Gaussian (High): $d(t) \sim \mathcal{N}(0, 5.0^2)$ N for $t \geq 1.0$ s

Critical Observation: Extended scenarios (3-8) were NOT included in PSO fitness. This tests whether robust gains generalize to unseen disturbance types.

Test Protocol:

- Baseline Testing: Evaluate default gains (Section 5.3) under all 8 disturbance scenarios - Robust PSO Optimization: Optimize gains using Jrobust fitness (scenarios 1-2 only) - Validation Testing: Re-evaluate optimized gains under all 8 scenarios - Generalization Analysis: Compare performance on seen (1-2) vs unseen (3-8) scenarios

Performance Metrics (Disturbance-Specific):

- Settling Time (ts): Time to stabilize after disturbance onset (Section 6.2) - Max Overshoot (OSmax): Peak angle deviation after disturbance - Convergence Rate ($p$conv): Fraction of trials achieving $||\theta|| < 5deg$ within 9 seconds - Robustness Score: $R = p\text{conv} \times (1 - \text{OSmax}/180deg)$ (higher is better)

Statistical Validation:

- Monte Carlo Trials: 50 trials per scenario per controller (random seeds 0-49) - Confidence Intervals: 95percent CI via bootstrap (10,000 resamples) - Significance Testing: Welch's t-test for pairwise comparisons ($\alpha = 0.01$)

Implementation:

All disturbance scenarios implemented using ‘DisturbanceGenerator‘ class (‘src/utils/disturbances.py‘): - Step: ‘add step disturbance(magnitude=10.0, start time=2.0)‘ - Impulse: ‘add impulse disturbance(magnitude=30.0, start time=2.0, duration=0.1)‘ - Sinusoidal: ‘add sinusoidal disturbance(magnitude=A, frequency=f, start time=1.0)‘ - Random: ‘add random disturbance(std dev=sigma, start time=1.0)‘ with seeded RNG

Scripts: - ‘scripts/mt8 robust pso.py‘ - Robust PSO optimization (4 controllers, 70 min runtime) - ‘scripts/mt8 extended validation.py‘ - Generalization testing (6 scenarios, 50 trials) - ‘benchmarks/MT8 COMPLETE REPORT.md‘ - Full analysis and results

Key Finding (Preview):

Robust PSO optimization achieved 21.4percent improvement for Hybrid Adaptive STA SMC on step/impulse scenarios, but 0percent convergence on sinusoidal/random scenarios. This demonstrates limited generalization and highlights the critical importance of comprehensive disturbance coverage in fitness functions. Detailed results in Section 8.4.

1.6 Reproducibility Checklist

This section provides a step-by-step guide for independent researchers to replicate the experimental results presented in this paper.

—

Step-by-Step Replication Guide

Step 1: Environment Setup

- Install Python 3.9 or later:
- Clone repository and install dependencies:
- Verify package versions: Expected output: ‘NumPy: 1.24.x, SciPy: 1.10.x, PySwarms: 1.3.x‘
- Verify numerical backend (optional, Linux only):
- Test installation:

Checkpoint 1: All package versions match ‘requirements.txt‘ specifications

—

Step 2: Configuration Validation

- Copy reference configuration:
- Check random seed configuration:
- Verify file paths:

Checkpoint 2: Configuration file matches reference settings, seed=42 confirmed

—

Step 3: Baseline Test (Single Simulation)

- Run single simulation with **Classical SMC**:
- Compare trajectory to reference output: Expected: Maximum state difference ¡ $10^{-}5(bitwiseidenticalonsamep$
- Verify performance metrics: Expected: Settling time 1.8-2.0s, Overshoot ¡5percent

Checkpoint 3: Single simulation produces expected trajectory (max difference ¡ $10^{-}5$)

—

Step 4: Full Benchmark Execution

- Run QW-2 quick benchmark (4 controllers, 100 trials each): Expected runtime: 15-20 minutes on reference hardware (4 controllers x 100 trials x 2-3s/sim)
- Verify completion: Expected output: ‘Total trials: 400‘
- Run MT-7 medium benchmark (10 random seeds, 50 trials each): Expected runtime: 45-60 minutes (10 seeds x 50 trials x 4 controllers x 2-3s/sim)

Checkpoint 4: QW-2 benchmark completes in 15-20 minutes, all 400 trials successful

—

Step 5: Statistical Analysis

- Run validation scripts:
- Verify statistical outputs: Expected: p-value matches reference ($\pm$0.001), Cohen’s d matches reference ($\pm$0.05)
- Generate performance figures:
- Compare figures to reference: Expected: Structural similarity index (SSIM) ¿ 0.95 for all plots

Checkpoint 5: Statistical outputs match reference (p-values $\pm$0.001, Cohen’s d $\pm$0.05)

—

Verification Checkpoints Summary

[TABLE - See Markdown version for details]

All checkpoints must pass () for successful replication.

—

Common Setup Issues and Solutions

[TABLE - See Markdown version for details]

—

Platform-Specific Notes

Windows: - Use ‘python‘ instead of ‘python3‘ (python3.exe does not exist on standard Windows installations) - File paths use backslashes: ‘optimization results2 results.json‘ - PowerShell: Use ‘Get-Content‘ instead of ‘cat‘

Linux: - Verify BLAS backend: ‘ldd $(python - c'importnumpy; print(numpy.file)')|grepblas` - Installsystemdependencies : `sudoaptinstallbuild - essentiallibopenblas - dev$‘

macOS: - Use Homebrew for Python: ‘brew install python@3.9‘ - Install Xcode Command Line Tools: ‘xcode-select –install‘

—

Reproducibility Guarantee

Following this checklist ensures: - Bitwise-identical results on the same platform (CPU architecture, OS, Python version) - Statistically equivalent results across platforms (p-values within ±0.001, effect sizes within ±0.05) - Comparable performance (runtimes within ±20percent on similar hardware)

For questions or issues during replication, consult the GitHub repository issues page or contact the authors.

1.7 Experimental Setup Quick Reference

This table provides a one-page lookup of all critical setup specifications for rapid reference during replication.

Table 6.1: Experimental Setup Quick Reference Card

[TABLE - See Markdown version for details]

—

Usage Guidelines:

- For replication: Use values in ”Value” column exactly as specified - For cross-reference: See ”Reference” column for detailed explanations - For custom experiments: Modify values and document changes in experimental log - For troubleshooting: Compare actual vs expected values from this table

Critical Parameters (DO NOT MODIFY without justification): - Random seed (42) - Required for reproducibility - Integrator tolerances (atol, rtol) - Affects numerical accuracy - Statistical significance (alpha = 0.05) - Standard in control systems literature - PSO hyperparameters (w, c, c) - Validated in Section 5.7

Platform-Specific Adjustments: - CPU speed: If slower than i7-10700K, increase timeout limits proportionally - RAM: If ¡16 GB, reduce batch size or use sequential simulation - Python version: If 3.10+, verify NumPy compatibility (no major issues expected)

1.8 Pre-Flight Validation Protocol

Before running full benchmarks (which may take hours), execute this 5-minute validation protocol to verify experimental setup correctness. This prevents wasting computational resources on misconfigured experiments.

—

Validation Test 1: Package Version Check

Purpose: Ensure all dependencies meet minimum version requirements

Command:

Expected Output:

Pass Criterion: All versions meet or exceed minimum requirements

Failure Actions: - If Python ¡ 3.9: Upgrade Python or use ‘pyenv‘/‘conda‘ - If packages outdated: ‘pip install –upgrade numpy scipy matplotlib pyswarms‘ - If version conflicts: Create fresh virtual environment

—

Validation Test 2: Single Simulation Sanity Check

Purpose: Verify basic simulation functionality and controller stability

Command:

Expected Behavior: - Simulation completes without errors (exit code 0) - Runtime: 0.4-0.6s on reference hardware (i7-10700K) - No warnings about numerical instability - Output file ‘preflight test.json‘ created

Post-Simulation Checks:

Expected Metrics: - Settling time: 1.8-2.2s - Overshoot: ¡10percent - Max cart position: ¡2.0 m (no runway escape) - Crashed: False (no instability)

Pass Criterion: All metrics within expected ranges, no crashes

Failure Actions: - If runtime ¿1.0s: Check CPU load, BLAS backend (see Section 6.6) - If settling time ¿3.0s: Controller gains may be wrong, verify ‘config.yaml‘ - If crashed: Increase boundary layer epsilon or check initial conditions - If NaN values: Reduce integration tolerance (rtol = 10^-2)

—

Validation Test 3: Numerical Accuracy Verification

Purpose: Ensure integration tolerances are appropriate (not too loose, not too tight)

Command:

Expected Output:

Pass Criterion: Maximum state difference ¡ $10^-4(indicatesappropriatetolerances)$

Failure Actions: - If difference ¿ $10^-3 : Tolerancestooloose, decrease`rtol`to10^-4 - Ifdifference < 10^-6 : Tolerancesunnecessarilytight, increase`rtol`to10^-2forspeed - IfRK45fails : Checkforstiffdynamics, c$

—

Validation Test 4: Reproducibility Test

Purpose: Verify random seed functionality for bitwise-identical results

Command:

Expected Output:

Pass Criterion: Trajectories are bitwise identical (diff = 0.0)

Failure Actions: - If diff ¿ 0: Check for ‘np.random.seed()‘ vs ‘random.seed()‘ inconsistency - Verify all randomness sources use seeded generator - Platform-dependent: Some numerical libraries (MKL) may have non-deterministic threading - Solution: Set ‘OMP NUM THREADS=1‘ environment variable for strict reproducibility

—

Validation Test 5: Computational Performance Baseline

Purpose: Verify simulation runtime matches expected performance for resource planning

Command:

Expected Output:

Pass Criterion: Average time 0.4-0.8s on similar hardware (±50percent tolerance for CPU differences)

Failure Actions: - If ¿1.0s: Investigate CPU throttling (‘cpufreq-info‘ on Linux) - Check BLAS backend: ‘python -c ”import numpy; numpy.show config()”‘ - Recommended: OpenBLAS or MKL (not reference BLAS) - If ¡0.2s: Suspiciously fast, verify simulation actually running (check trajectory length)

—

Pre-Flight Validation Summary

[TABLE - See Markdown version for details]

Total Pre-Flight Time: 3 minutes

Overall Pass Criterion: ALL 5 tests must pass () before proceeding to full benchmarks.

—

What to Do If Pre-Flight Fails:

- One test fails: Fix specific issue (see "Failure Actions" for that test), re-run that test - Multiple tests fail: Likely environmental issue (Python version, dependencies, hardware) - Recommended: Fresh virtual environment + reinstall dependencies - All tests fail: Critical setup problem - Verify Python installation: ‘which python‘ (should be 3.9+) - Verify repository clone: ‘git status‘ (should show clean working directory) - Contact authors or open GitHub issue with full error logs

Pre-Flight Success -¿ Proceed to Benchmarks:

Once all 5 tests pass, you can confidently run full benchmarks (QW-2, MT-7) knowing that: - Software environment is correct - Numerical stability is adequate - Reproducibility is guaranteed - Computational performance is acceptable

Estimated Full Benchmark Runtimes (based on Test 5 baseline): - QW-2 (400 trials): 15-20 minutes - MT-7 (500 trials): 45-60 minutes - Full campaign (all scenarios): 2-3 hours

—