

2025-11-01



## subsection 0.0 What You'll Learn

- **Coverage Tiers:** 85% overall, 95% critical, 100% safety-critical
- **Test Organization:** Peer files (test\_\*.py for every \*.py)
- **Validation Strategies:** Theoretical properties, edge cases, integration tests
- **Automation:** Pre-commit hooks, CI/CD integration, benchmark regression

## subsection 0.0 Why This Matters

**Problem:** Uncaught edge cases (e.g., singular matrices, division by zero) cause silent failures in production.

**Solution:** Enforce 95% coverage on critical paths (controllers, dynamics) with theoretical property validation (Lyapunov stability, boundedness).

**Impact:** Zero critical bugs reported in Phase 5 research (11 tasks, 46 hours) after achieving coverage standards.

## section 0 Coverage Tiers

### subsection 0.0 Three-Tier System

Tier	Coverage Target	Examples
Safety-Critical	100%	SMC control law, saturation
Critical Paths	$\geq 95\%$	Controllers, dynamics, PSO
Overall Codebase	$\geq 85\%$	Utils, visualization, CLI

## subsection 0.0 Rationale

- **100% Safety-Critical:** Code that can cause physical damage (control saturation, state constraints)
- **95% Critical Paths:** Core algorithms (SMC variants, PSO optimizer, dynamics models)
- **85% Overall:** Acceptable trade-off (diminishing returns above 85% for utilities)

## subsection 0.0 Current Status (January 2025)

- **Overall Coverage:** 87% (target: 85%, status: PASS)
- **Critical Paths:** 96% controllers, 94% dynamics (target: 95%, status: PASS)
- **Safety-Critical:** 100% (saturation, state validators, status: PASS)
- **Test Count:** 200+ pytest tests, 15 benchmark tests

## section 0 Test Organization

### subsection 0.0 Peer File Structure

**Rule:** Every \*.py file has a corresponding test\_\*.py peer.

```
lstnumbersrc/
lstnumber  controllers/
lstnumber    classical_smc.py
lstnumber    sta_smc.py
lstnumber    adaptive_smc.py
lstnumber  core/
lstnumber    simulation_runner.py
lstnumber  optimizer/
lstnumber    pso_optimizer.py
lstnumber
lstnumbertests/
lstnumber  test_controllers/
lstnumber    test_classical_smc.py      # Peer for classical_smc.py
lstnumber    test_sto_smc.py          # Peer for sta_smc.py
```

```

lstnumber    test_adaptive_smc.py      # Peer for adaptive_smc.py
lstnumber    test_core/
lstnumber    test_simulation_runner.py # Peer for simulation_runner.py
lstnumber    test_optimizer/
lstnumber    test_pso_optimizer.py     # Peer for pso_optimizer.py

```

## subsection 0.0 Test Discovery

```

lstnumber# Run All Tests
lstnumberpython -m pytest tests/ -v
lstnumber
lstnumber# Run Specific Module
lstnumberpython -m pytest tests/test_controllers/ -v
lstnumber
lstnumber# Run Single File
lstnumberpython -m pytest tests/test_controllers/test_classical_smc.py -v
lstnumber
lstnumber# Run with Coverage Report
lstnumberpython -m pytest tests/ --cov=src --cov-report=html

```

## subsection 0.0 Peer File Validation

```

lstnumber# Check for Missing Peer Files
lstnumberpython scripts/architecture/find_untested.py
lstnumber
lstnumber# Output Example
lstnumber[WARNING] Missing test peers:
lstnumber  src/utils/plotting/advanced_plots.py
lstnumber  src/core/batch_simulator.py
lstnumber
lstnumber[INFO] Action: Create tests/test_utils/test_plotting/test_advanced_plots.py
lstnumber[INFO] Action: Create tests/test_core/test_batch_simulator.py

```

# section 0 Validation Strategies

## subsection 0.0 1. Unit Tests (Isolated Components)

**Goal:** Verify single function/class behavior in isolation.

```

lstnumberdef test_classical_smc_zero_error():
    """Test SMC returns zero control when error is zero."""
    controller = ClassicalSMC(lambda1=10, lambda2=5, phi1=2, phi2=1)
    state = np.array([0, 0, 0, 0]) # Zero error
    control = controller.compute_control(state)
    assert np.allclose(control, 0.0, atol=1e-6)

lstnumberdef test_pso_converges():
    """Test PSO reduces cost over iterations."""
    optimizer = PSOTuner(swarm_size=10, iterations=20)
    initial_cost = optimizer.best_cost
    optimizer.optimize(objective_function)
    final_cost = optimizer.best_cost
    assert final_cost < initial_cost # Cost must decrease

```

## subsection 0.0 2. Integration Tests (Multi-Component)

**Goal:** Verify components work together correctly.

```

lstnumberdef test_simulation_end_to_end():
    """Test full simulation pipeline (controller + dynamics)."""
    controller = create_controller("classical_smc", config)
    dynamics = SimplifiedDynamics(config)
    runner = SimulationRunner(controller, dynamics, config)

    result = runner.run()

    # Verify convergence
    assert result.settling_time < 5.0 # Settles in <5s

```

```

lstnumber     assert np.max(np.abs(result.states[-1])) < 0.1 # Final error <0.1
lstnumber
lstnumberdef test_pso_tunes_controller():
lstnumber     """Test PSO optimizer improves controller performance."""
lstnumber     baseline_iae = simulate_with_manual_gains()
lstnumber     pso = PSOTuner(swarm_size=30, iterations=50)
lstnumber     optimized_gains = pso.optimize(controller_objective)
lstnumber
lstnumber     optimized_iae = simulate_with_gains(optimized_gains)
lstnumber     assert optimized_iae < baseline_iae # PSO improves performance

```

### subsection 0.0 3. Property-Based Tests (Theoretical Validation)

**Goal:** Verify mathematical properties hold for ALL inputs.

```

lstnumberfrom hypothesis import given, strategies as st
lstnumber
lstnumber@given(st.floats(min_value=-10, max_value=10, allow_nan=False))
lstnumberdef test_saturation_bounds(control_input):
lstnumber     """Test saturation enforces bounds for ALL inputs."""
lstnumber     saturated = saturate(control_input, u_max=5.0)
lstnumber     assert -5.0 <= saturated <= 5.0 # Always bounded
lstnumber
lstnumber@given(st.lists(st.floats(), min_size=4, max_size=4))
lstnumberdef test_lyapunov_decreasing(state):
lstnumber     """Test Lyapunov function decreases for ALL states."""
lstnumber     V_current = compute_lyapunov(state)
lstnumber     state_next = simulate_step(state)
lstnumber     V_next = compute_lyapunov(state_next)
lstnumber     assert V_next < V_current # Lyapunov always decreases

```

### subsection 0.0 4. Edge Case Tests

**Goal:** Verify behavior at boundaries and degenerate cases.

```

lstnumberdef test_controller_singular_matrix():
lstnumber     """Test controller handles singular inertia matrix."""
lstnumber     dynamics = SimplifiedDynamics(inertia=np.zeros((4,4))) # Singular
lstnumber     with pytest.raises(SingularMatrixError):
lstnumber         simulate(dynamics)
lstnumber
lstnumberdef test_pso_empty_search_space():
lstnumber     """Test PSO handles zero-width search bounds."""
lstnumber     bounds = [(1.0, 1.0), (2.0, 2.0)] # Zero width
lstnumber     with pytest.raises(ValueError, match="SearchSpace has zero volume"):
lstnumber         PSOTuner(bounds=bounds)
lstnumber
lstnumberdef test_controller_inf_nan_inputs():
lstnumber     """Test controller rejects inf/nan inputs gracefully."""
lstnumber     controller = ClassicalSMC(lambda1=10, lambda2=5)
lstnumber     state = np.array([np.inf, np.nan, 0, 0])
lstnumber     with pytest.raises(ValueError, match="State contains inf/nan"):
lstnumber         controller.compute_control(state)

```

## section 0 Benchmark Tests

### subsection 0.0 Purpose

- **Performance Regression Detection:** Alert if simulation slows by >5%
- **Comparative Analysis:** Benchmark 7 controllers head-to-head
- **Optimization Validation:** Verify PSO reduces IAE by 40%+

### subsection 0.0 Benchmark Organization

```

lstnumbertests/test_benchmarks/
lstnumber  test_controller_benchmarks.py    # Time per step, IAE, chattering

```

```
lstnumber    test_pso_benchmarks.py          # Convergence speed, final cost
lstnumber    test_simulation_benchmarks.py  # End-to-end runtime
```

## subsection 0.0 Example Benchmark Test

```
lstnumberimport pytest
lstnumber
lstnumber@ pytest.mark.benchmark(group="controllers"):
lstnumberdef test_classical_smclBenchmark(benchmark):
lstnumber    """Benchmark classical SMC compute_control() speed."""
lstnumber    controller = ClassicalSMC(lambda1=10, lambda2=5, phi1=2, phi2=1)
lstnumber    state = np.array([0.1, 0.2, 0.0, 0.0])
lstnumber
lstnumber    result = benchmark(controller.compute_control, state)
lstnumber
lstnumber    # Verify performance target (10ms max)
lstnumber    assert result.stats.mean < 0.01  # <10ms average
lstnumber
lstnumber@ pytest.mark.benchmark(group="optimization"):
lstnumberdef test_pso_benchmark(benchmark):
lstnumber    """Benchmark PSO convergence time."""
lstnumber    pso = PSOTuner(swarm_size=30, iterations=50)
lstnumber
lstnumber    result = benchmark(pso.optimize, rosenbrock_function)
lstnumber
lstnumber    # Verify convergence time (<2 seconds)
lstnumber    assert result.stats.mean < 2.0
```

## subsection 0.0 Running Benchmarks

```
lstnumber# Run All Benchmarks
lstnumberpython -m pytest tests/test_benchmarks/ --benchmark-only
lstnumber
lstnumber# Compare Against Baseline
lstnumberpython -m pytest tests/test_benchmarks/ --benchmark-compare=baseline.json
lstnumber
lstnumber# Save New Baseline
lstnumberpython -m pytest tests/test_benchmarks/ --benchmark-save=new_baseline
```

# section 0 Pre-commit Hooks

## subsection 0.0 Automated Test Enforcement

File: .pre-commit-config.yaml

```
lstnumberrepos:
lstnumber  - repo: local
lstnumber    hooks:
lstnumber      # Run Pytest on Changed Files
lstnumber      - id: pytest
lstnumber        name: Run Tests
lstnumber        entry: python -m pytest tests/ -v
lstnumber        language: system
lstnumber        pass_filenames: false
lstnumber        stages: [commit]
lstnumber
lstnumber      # Check Coverage Thresholds
lstnumber      - id: coverage
lstnumber        name: Check Coverage
lstnumber        entry: python -m pytest --cov=src --cov-report=term --cov-fail-under=85
lstnumber        language: system
lstnumber        pass_filenames: false
lstnumber        stages: [commit]
lstnumber
lstnumber      # Find Untested Files
lstnumber      - id: untested
lstnumber        name: Check for Untested Files
lstnumber        entry: python scripts/architecture/find_untested.py
```

```
lstnumber      language: system
lstnumber      stages: [commit]
```

## subsection 0.0 Commit Workflow

```
lstnumber# Attempt Commit (triggers pre-commit hooks)
lstnumbergit commit -m "feat: Add adaptive SMC controller"
lstnumber
lstnumber# Pre-commit Hook Output
lstnumber[INFO] Running pytest tests/
lstnumber===== 200 passed in 12.3s =====
lstnumber
lstnumber[INFO] Checking coverage thresholds
lstnumberCoverage: 87% (target: 85%) PASS
lstnumber
lstnumber[INFO] Checking for untested files
lstnumberAll files have test peers. PASS
lstnumber
lstnumber[OK] Pre-commit checks passed. Commit created.
```

## section 0 CI/CD Integration

---

### subsection 0.0 GitHub Actions Workflow

File: .github/workflows/tests.yml

```
lstnumbername: Test Suite
lstnumberon: [push, pull_request]
lstnumber
lstnumberjobs:
lstnumber  test:
lstnumber    runs-on: ubuntu-latest
lstnumber    steps:
lstnumber      - uses: actions/checkout@v3
lstnumber      - uses: actions/setup-python@v4
lstnumber      with:
lstnumber        python-version: '3.9'
lstnumber
lstnumber      - name: Install Dependencies
lstnumber        run: pip install -r requirements.txt
lstnumber
lstnumber      - name: Run Tests
lstnumber        run: python -m pytest tests/ -v --cov=src --cov-report=xml
lstnumber
lstnumber      - name: Upload Coverage
lstnumber        uses: codecov/codecov-action@v3
lstnumber        with:
lstnumber          file: ./coverage.xml
lstnumber          fail_ci_if_error: true
lstnumber
lstnumber      - name: Check Coverage Thresholds
lstnumber        run: |
lstnumber          python -m pytest --cov=src --cov-fail-under=85
lstnumber          python scripts/quality/check_critical_coverage.py # Verify 95% critical
lstnumber
lstnumber      - name: Run Benchmarks
lstnumber        run: python -m pytest tests/test_benchmarks/ --benchmark-only
```

## subsection 0.0 Pull Request Checks

- **Test Pass Rate:** 100% required (all 200+ tests must pass)
- **Coverage Thresholds:** Overall  $\geq 85\%$ , critical  $\geq 95\%$
- **Benchmark Regression:** No slowdowns  $>5\%$  vs. baseline
- **Linting:** Ruff + MyPy strict mode, zero errors

## section 0 Testing Anti-Patterns

---

### subsection 0.0 Anti-Pattern 1: Trivial Tests

**Problem:** Tests that only verify imports or type checks.

```
lstnumber# BAD: Trivial test (no value)
lstnumberdef test_controller_imports():
lstnumber    from src.controllers.classical_sm import ClassicalSMC
lstnumber    assert ClassicalSMC is not None # Useless assertion
lstnumber
lstnumber# GOOD: Functional test
lstnumberdef test_controller_convergence():
lstnumber    controller = ClassicalSMC(...)
lstnumber    result = simulate(controller, initial_state=[1, 0, 0, 0])
lstnumber    assert result.settling_time < 5.0 # Meaningful property
```

### subsection 0.0 Anti-Pattern 2: Non-Deterministic Tests

**Problem:** Tests that fail randomly due to uncontrolled randomness.

```
lstnumber# BAD: Random seed not fixed
lstnumberdef test_pso_converges():
lstnumber    pso = PSOTuner() # No seed
lstnumber    result = pso.optimize(objective)
lstnumber    assert result.cost < 0.1 # Fails 10% of time
lstnumber
lstnumber# GOOD: Fixed seed for reproducibility
lstnumberdef test_pso_converges():
lstnumber    pso = PSOTuner(seed=42) # Deterministic
lstnumber    result = pso.optimize(objective)
lstnumber    assert result.cost < 0.1 # Always passes
```

### subsection 0.0 Anti-Pattern 3: Monolithic Tests

**Problem:** Single test verifies 10+ behaviors (hard to debug).

```
lstnumber# BAD: Test does too much
lstnumberdef test_entire_system():
lstnumber    # 50 lines testing controller + dynamics + PSO + visualization
lstnumber    assert everything_works # Unclear what failed
lstnumber
lstnumber# GOOD: Atomic tests
lstnumberdef test_controller_compute():
lstnumber    # Test only controller.compute_control()
lstnumber
lstnumberdef test_dynamics_step():
lstnumber    # Test only dynamics.step()
lstnumber
lstnumberdef test_pso_optimize():
lstnumber    # Test only pso.optimize()
```

## section 0 Coverage Measurement

---

### subsection 0.0 Line Coverage

**Tool:** pytest-cov

```
lstnumber# Generate HTML Report
lstnumberpython -m pytest --cov=src --cov-report=html
lstnumber
lstnumber# View Report
lstnumberopen htmlcov/index.html # (Mac/Linux)
lstnumberstart htmlcov/index.html # (Windows)
```

### subsection 0.0 Branch Coverage

**Goal:** Verify all conditional branches (if/else) tested.

```

lstnumber# Enable Branch Coverage
lstnumberpython -m pytest --cov=src --cov-branch --cov-report=term
lstnumber
lstnumber# Example Output
lstnumbersrc/controllers/classical_smc.py    96%   (4 branches, 3 covered)
lstnumbersrc/optimizer/pso_optimizer.py        94%   (12 branches, 11 covered)

```

## subsection 0.0 Critical Path Coverage

Script: scripts/quality/check\_critical\_coverage.py

```

lstnumberCRITICAL_PATHS = [
lstnumber    "src/controllers/",
lstnumber    "src/core/dynamics.py",
lstnumber    "src/optimizer/pso_optimizer.py",
lstnumber    "src/utils/control_primitives.py"
lstnumber]
lstnumber
lstnumberdef check_critical_coverage():
lstnumber    coverage_data = parse_coverage_report()
lstnumber    for path in CRITICAL_PATHS:
lstnumber        coverage = coverage_data[path]
lstnumber        if coverage < 0.95: # 95% threshold
lstnumber            raise Exception(f"Critical path {path} only {coverage*100}% covered")
lstnumber    print("[OK] All critical paths meet 95% threshold")

```

## section 0 Maintenance & Continuous Improvement

### subsection 0.0 Monthly Test Audit

```

lstnumber# 1. Find Untested Code
lstnumberpython scripts/architecture/find_untested.py
lstnumber
lstnumber# 2. Identify Low-Coverage Modules
lstnumberpython -m pytest --cov=src --cov-report=term | grep -E '[0-7][0-9]%'
lstnumber
lstnumber# 3. Review Flaky Tests (tests that fail intermittently)
lstnumberpython scripts/quality/detect_flaky_tests.py
lstnumber
lstnumber# 4. Update Benchmark Baselines
lstnumberpython -m pytest tests/test_benchmarks/ --benchmark-save=baseline_jan2025

```

### subsection 0.0 Regression Analysis

- **Track Test Count Growth:** Target 10+ new tests per major feature
- **Monitor Coverage Trends:** Alert if coverage drops below 85%
- **Benchmark Drift:** Flag if any benchmark slows by >5%
- **Flaky Test Detection:** Remove non-deterministic tests

### subsection 0.0 Test Documentation

File: tests/README.md

- Document test organization (peer files, benchmarks, integration)
- List coverage targets (85%/95%/100% tiers)
- Explain how to run specific test suites
- Provide examples of property-based tests
- Link to CI/CD workflow and pre-commit hooks

## section 0 Success Metrics

---

### subsection 0.0 Quantitative Indicators

Metric	Target	Current
Overall coverage	$\geq 85\%$	87%
Critical coverage	$\geq 95\%$	96% (controllers)
Safety-critical coverage	100%	100% (saturation)
Test count	$> 200$	215
Benchmark tests	$> 15$	18
Flaky test rate	$< 2\%$	0.5%

### subsection 0.0 Qualitative Indicators

- Zero critical bugs reported in Phase 5 research (11 tasks, 46 hours)
- All edge cases discovered via property-based tests (Hypothesis)
- Pre-commit hooks prevent untested code from merging
- CI/CD catches regressions before production deployment

## Checklist: Testing Philosophy

---

- Organize:** Create peer test files (test\_\*.py for every \*.py)
- Coverage:** Achieve 85% overall, 95% critical, 100% safety-critical
- Validate:** Write property-based tests for theoretical invariants
- Benchmark:** Track performance regressions (<5% tolerance)
- Pre-commit:** Enable hooks to block untested code
- CI/CD:** Integrate GitHub Actions for PR checks
- Edge Cases:** Test singular matrices, inf/nan, zero-width bounds
- Monitor:** Monthly audits for flaky tests and coverage drift

## Next Steps

---

- **E019:** Production safety - memory management and thread safety
- **E020:** MCP integration - auto-trigger strategy and server orchestration
- **E021:** Maintenance mode and future vision for DIP-SMC-PSO