📁 Package Structure

This document explains the organization of the Constrained Intelligence Constants codebase.

Directory Tree

```
constrained-intelligence-constants/
├── constrained_intelligence/          # Main package source code
│   ├── __init__.py                    # Package initialization and exports
│   ├── core.py                        # Core measurement and optimization classes
│   ├── discovery.py                   # Constant discovery algorithms
│   └── constants.py                   # Fundamental constants definitions
│
├── tests/                             # Test suite
│   ├── __init__.py
│   ├── test_core.py                   # Tests for core module
│   ├── test_discovery.py              # Tests for discovery module
│   └── test_constants.py              # Tests for constants module
│
├── examples/                          # Usage examples
│   ├── basic_usage.py                 # Simple examples for getting started
│   ├── advanced_examples.py           # Complex use cases
│   └── notebook.ipynb                 # Interactive Jupyter notebook
│
├── validation/                        # Experimental validation
│   └── experimental_validation.py     # Validation experiments and benchmarks
│
├── docs/                              # Documentation (Sphinx)
│   ├── conf.py                        # Sphinx configuration
│   ├── index.rst                      # Documentation index
│   └── api/                           # API reference docs
│
├── .github/                           # GitHub-specific files
│   └── workflows/                     # GitHub Actions workflows
│       └── ci.yml                     # Continuous integration pipeline
│
├── README.md                          # Main project README
├── QUICKSTART.md                      # Quick start guide
├── THEORY.md                          # Mathematical theory and proofs
├── CONTRIBUTING.md                    # Contribution guidelines
├── PACKAGE_STRUCTURE.md               # This file
├── CODE_OF_CONDUCT.md                 # Code of conduct
├── CHANGELOG.md                       # Version history
├── LICENSE                            # MIT License
│
├── setup.py                           # Package installation configuration
├── requirements.txt                   # Runtime dependencies
├── requirements-dev.txt               # Development dependencies
├── MANIFEST.in                        # Additional files for distribution
│
├── Dockerfile                         # Docker container definition
├── .dockerignore                      # Docker ignore patterns
├── .gitignore                         # Git ignore patterns
└── .flake8                            # Flake8 linter configuration
```

Core Modules

`constrained_intelligence/__init__.py`

Purpose: Package entry point, defines public API

Exports:
- Main classes: `ConstantsMeasurement`, `OptimizationEngine`, `BoundedSystemAnalyzer`
- Discovery tools: `ConstantDiscovery`, `DiscoveryMethods`
- All constants from `constants.py`

Usage:

```
from constrained_intelligence import ConstantsMeasurement, GOLDEN_RATIO
```

`constrained_intelligence/core.py`

Purpose: Core functionality for measurement, optimization, and analysis

Classes:

1. **`MeasurementResult`** (dataclass)
   - Stores results of constant measurements
   - Attributes: constant_value, confidence, bounds, empirical_evidence, theoretical_basis

2. **`ConstantsMeasurement`**
   - Measures constants in bounded systems
   - Methods:
     - `measure_resource_allocation()`: Find optimal resource split
     - `measure_learning_convergence()`: Predict convergence timing
     - `measure_optimization_efficiency()`: Analyze efficiency bounds

3. **`OptimizationEngine`**
   - Optimization algorithms using mathematical constants
   - Methods:
     - `golden_ratio_optimization()`: Golden section search
     - `exponential_decay_schedule()`: Generate decay schedules
     - `adaptive_step_size()`: Calculate adaptive learning rates

4. **`BoundedSystemAnalyzer`**
   - Analyze constraints and system boundaries
   - Methods:
     - `analyze_constraint_boundaries()`: Identify optimal boundaries
     - `detect_emergent_patterns()`: Find patterns in time series

Dependencies: `numpy`, `scipy`, `math`

`constrained_intelligence/discovery.py`

Purpose: Discover mathematical constants from empirical data

Classes:

1. `DiscoveryMethods` (Enum)
   - Enumeration of discovery methods
   - Values: OPTIMIZATION_BASED, CONVERGENCE_ANALYSIS, PERIODICITY_DETECTION, etc.

2. `DiscoveryResult` (dataclass)
   - Stores discovery results
   - Includes confidence metrics and validation data

3. `ConstantDiscovery`
   - Main discovery engine
   - Methods:
     - `discover_from_optimization()` : Find constants in optimization trajectories
     - `detect_convergence_constants()` : Discover from convergence patterns
     - `discover_from_ratios()` : Analyze ratio sequences
     - `discover_from_boundaries()` : Find constants from boundary behavior
     - `validate_discovery()` : Validate discovered constants

Dependencies: `numpy` , `scipy.stats` , `math`

`constrained_intelligence/constants.py`

Purpose: Define fundamental mathematical constants

Constants Defined:

Fundamental:
- `GOLDEN_RATIO` : φ ≈ 1.618
- `EULER_NUMBER` : e ≈ 2.718
- `PI` : π ≈ 3.14159

Derived:
- `OPTIMAL_RESOURCE_SPLIT` : 1/φ ≈ 0.618
- `CONVERGENCE_THRESHOLD_FACTOR` : 1/e ≈ 0.368
- `LEARNING_RATE_BOUNDARY` : 1/(2π) ≈ 0.159

System Boundaries:
- `MAX_EFFICIENCY_RATIO` : 0.886
- `MINIMAL_COMPLEXITY_CONSTANT` : e^(1/e) ≈ 1.444
- `INFORMATION_DENSITY_LIMIT` : 2·ln(2) ≈ 1.386

Thresholds:
- `HIGH_CONFIDENCE_THRESHOLD` : 0.9
- `MEDIUM_CONFIDENCE_THRESHOLD` : 0.7
- `VALIDATION_SIGNIFICANCE_LEVEL` : 0.05

Helper Functions:
- `get_constant_info()` : Get details about a constant
- `list_all_constants()` : List all defined constants

Test Structure

`tests/test_core.py`

Tests for `core.py` module:
- Test `ConstantsMeasurement` methods
- Test `OptimizationEngine` algorithms
- Test `BoundedSystemAnalyzer` analysis functions
- Validate against known mathematical results

`tests/test_discovery.py`

Tests for `discovery.py` module:
- Test all discovery methods
- Validate discovered constants against theoretical values
- Test statistical validation
- Edge cases and error handling

`tests/test_constants.py`

Tests for `constants.py` module:
- Verify constant values
- Test helper functions
- Ensure mathematical relationships hold

Examples

`examples/basic_usage.py`

Simple, self-contained examples:
- Resource allocation
- Golden ratio optimization
- Learning rate schedules
- Convergence detection

`examples/advanced_examples.py`

Complex use cases:
- Multi-objective optimization
- Adaptive learning systems
- Real-world applications
- Performance comparisons

`examples/notebook.ipynb`

Interactive Jupyter notebook:
- Visualizations
- Step-by-step tutorials
- Experimental playground

Validation

`validation/experimental_validation.py`

Comprehensive validation suite:
- Validate against known mathematical results
- Empirical performance benchmarks
- Statistical significance tests
- Comparison with baseline methods

Functions:
- `validate_golden_ratio()` : Validate φ-based methods
- `validate_exponential_convergence()` : Test e-based predictions
- `run_all_validations()` : Execute full validation suite

Configuration Files

`setup.py`

Package configuration for PyPI:
- Package metadata (name, version, author)
- Dependencies
- Entry points
- Classifiers

`requirements.txt`

Runtime dependencies:

```
numpy>=1.19.0
scipy>=1.5.0
```

`requirements-dev.txt`

Development dependencies:

```
pytest>=6.0.0
pytest-cov>=2.10.0
black>=21.0
flake8>=3.8.0
mypy>=0.800
jupyter>=1.0.0
matplotlib>=3.3.0
```

`MANIFEST.in`

Additional files to include in distribution:

```
include README.md
include LICENSE
include requirements.txt
recursive-include examples *.py *.ipynb
recursive-include tests *.py
```

Docker

`Dockerfile`

Container definition:
- Base image: Python 3.8+
- Install package and dependencies
- Set up working directory
- Entry point for running examples/tests

`.dockerignore`

Exclude from Docker build:
- `__pycache__`
- `*.pyc`
- `.git`
- Virtual environments

CI/CD

`.github/workflows/ci.yml`

Automated testing and deployment:
- Run tests on push/PR
- Check code style
- Generate coverage reports
- Build and publish to PyPI (on release)

Jobs:
1. **test**: Run pytest on multiple Python versions
2. **lint**: Check code style with black and flake8
3. **type-check**: Run mypy type checker
4. **deploy**: Publish to PyPI (on tags)

Documentation

Markdown Files

- **README.md**: Main project documentation
- **QUICKSTART.md**: 5-minute getting started guide
- **THEORY.md**: Mathematical foundations
- **CONTRIBUTING.md**: Contribution guidelines
- **PACKAGE_STRUCTURE.md**: This file

Sphinx Documentation (Optional)

For generating HTML/PDF documentation:

```
cd docs/
make html
```

Development Workflow

1. Clone Repository

```
git clone https://github.com/yourusername/constrained-intelligence-constants.git
cd constrained-intelligence-constants
```

2. Install Development Environment

```
pip install -e ".[dev]"
```

3. Run Tests

```
pytest tests/
```

4. Run Examples

```
python examples/basic_usage.py
```

5. Run Validation

```
python validation/experimental_validation.py
```

Building and Distribution

Build Package

```
python setup.py sdist bdist_wheel
```

Install Locally

```
pip install -e .
```

Upload to PyPI

```
twine upload dist/*
```

Code Organization Principles

1. **Separation of Concerns**: Each module has a clear, single responsibility
2. **Modularity**: Components can be used independently
3. **Testability**: All functions/classes have corresponding tests
4. **Documentation**: Every public API is documented
5. **Examples**: Usage examples for all major features

Import Hierarchy

```
constants.py          # No internal imports (only stdlib/numpy)
    ↓
core.py              # Imports from constants.py
    ↓
discovery.py         # Imports from core.py and constants.py
    ↓
__init__.py          # Imports from all modules, defines public API
```

This prevents circular dependencies and maintains clean architecture.

File Naming Conventions

- **Modules**: `lowercase_with_underscores.py`
- **Classes**: `CamelCase`
- **Functions**: `lowercase_with_underscores()`
- **Constants**: `UPPERCASE_WITH_UNDERSCORES`
- **Tests**: `test_*.py`
- **Examples**: `*_example.py` or `*_usage.py`

Questions?

If you have questions about the package structure:
- Check this document
- Read the inline code comments
- Open a GitHub Discussion (https://github.com/yourusername/constrained-intelligence-constants/discussions)

Maintained by the Constrained Intelligence Research Team