



Quick Start Guide

Get started with Constrained Intelligence Constants in 5 minutes!

Installation

```
pip install constrained-intelligence-constants
```

Your First Constant Discovery

Let's discover the golden ratio from optimization data:

```
from constrained_intelligence import ConstantDiscovery

# Create discovery engine
discovery = ConstantDiscovery()

# Simulate optimization data (values decreasing by golden ratio)
optimization_data = [100.0, 61.8, 38.2, 23.6, 14.6, 9.0, 5.6]

# Discover the constant
result = discovery.discover_from_optimization(
    optimization_data,
    method="golden_ratio"
)

# Check results
print(f"\u2708 Discovered constant: {result.discovered_constant:.4f}")
print(f"\u2708 Expected (\u03c6): {result.theoretical_constant:.4f}")
print(f"\u2708 Confidence: {result.confidence:.2%}")
```

Expected Output:

```
 Discovered constant: 1.6180
 Expected ( $\phi$ ): 1.6180
 Confidence: 98.5%
```

Optimize Using Golden Ratio

Use golden ratio search to find function minima:

```

from constrained_intelligence import OptimizationEngine

# Create optimizer
optimizer = OptimizationEngine(constraints={})

# Define objective function
def my_objective(x):
    return (x - 7.5) ** 2

# Optimize!
result = optimizer.golden_ratio_optimization(
    objective_function=my_objective,
    bounds=(0, 15),
    max_iterations=50
)

print(f"\u2022 Optimal x: {result['optimal_x']:.6f}")
print(f"\u2022 Optimal value: {result['optimal_value']:.6f}")
print(f"\u2022 Converged: {result['converged']}")

```

Measure Resource Allocation

Find optimal resource split:

```

from constrained_intelligence import ConstantsMeasurement

# Create measurer
measurer = ConstantsMeasurement(
    system_type="resource_allocation",
    constraints={}
)

# Measure optimal allocation for 1000 units
result = measurer.measure_resource_allocation(resource_budget=1000)

print(f"\u2022 Allocate: {result.empirical_evidence['optimal_allocated']:.1f} units")
print(f"\u2022 Reserve: {result.empirical_evidence['reserved']:.1f} units")
print(f"\u2022 Ratio: {result.empirical_evidence['allocation_ratio']:.4f}")

```

Analyze Learning Convergence

Predict when learning will converge:

```

from constrained_intelligence import ConstantsMeasurement

measurer = ConstantsMeasurement(
    system_type="learning",
    constraints={}
)

# Simulate learning curve
performance = [0.1, 0.25, 0.45, 0.62, 0.74, 0.83, 0.89, 0.92, 0.94]

result = measurer.measure_learning_convergence(
    iterations=100,
    performance_data=performance
)

print(f"\u27e8 Predicted convergence at iteration: {result.empirical_evidence['predicted_convergence_point']:.1f}\u27e9")
print(f"\u27e8 Performance gain: {result.empirical_evidence['actual_performance_gain']:.2f}\u27e9")

```

Detect Constants in Your Data

Discover constants from any numerical sequence:

```

from constrained_intelligence import ConstantDiscovery

discovery = ConstantDiscovery()

# Your data (e.g., from experiments)
my_data = [10.0, 6.18, 3.82, 2.36, 1.46, 0.90]

result = discovery.discover_from_ratios(my_data, order=1)

print(f"\u27e8 Discovered ratio: {result.discovered_constant:.4f}\u27e9")
print(f"\u27e8 Matches known constant: {result.validation_metrics}\u27e9")

```

Use Pre-defined Constants

Access common constants directly:

```

from constrained_intelligence import (
    GOLDEN_RATIO,
    EULER_NUMBER,
    OPTIMAL_RESOURCE_SPLIT,
    CONVERGENCE_THRESHOLD_FACTOR
)

print(f"Golden Ratio (\u03d5): {GOLDEN_RATIO:.6f}")
print(f"Euler's Number (e): {EULER_NUMBER:.6f}")
print(f"Optimal Split (1/\u03d5): {OPTIMAL_RESOURCE_SPLIT:.6f}")
print(f"Convergence Factor (1/e): {CONVERGENCE_THRESHOLD_FACTOR:.6f}")

```

Complete Example: Learning Rate Schedule

Create an exponentially decaying learning rate schedule:

```

from constrained_intelligence import OptimizationEngine
import matplotlib.pyplot as plt

optimizer = OptimizationEngine(constraints={})

# Create learning rate schedule
learning_rates = optimizer.exponential_decay_schedule(
    initial_value=0.1,      # Start at 0.1
    decay_constant=0.05,    # Decay rate
    steps=100                # 100 training steps
)

# Plot the schedule
plt.plot(learning_rates)
plt.xlabel('Training Step')
plt.ylabel('Learning Rate')
plt.title('Exponential Decay Learning Rate Schedule')
plt.grid(True)
plt.savefig('lr_schedule.png')
print("✓ Learning rate schedule created and saved!")

```

Next Steps

- [Full Documentation \(README.md\)](#): Learn about all features
- [Theory \(THEORY.md\)](#): Understand the mathematical foundations
- [Examples \(examples/\)](#): Explore complete examples
- [Contributing \(CONTRIBUTING.md\)](#): Help improve the framework

Common Patterns

Pattern 1: Resource Allocation

```

from constrained_intelligence import OPTIMAL_RESOURCE_SPLIT

total_resources = 1000
active_pool = total_resources * OPTIMAL_RESOURCE_SPLIT  # ≈618
reserve_pool = total_resources - active_pool           # ≈382

```

Pattern 2: Convergence Check

```

from constrained_intelligence import CONVERGENCE_THRESHOLD_FACTOR

max_iterations = 1000
check_convergence_at = int(max_iterations * CONVERGENCE_THRESHOLD_FACTOR)
# Check at iteration ≈368

```

Pattern 3: Optimization

```

from constrained_intelligence import OptimizationEngine

optimizer = OptimizationEngine(constraints={})
result = optimizer.golden_ratio_optimization(objective_fn, (0, 100))

```

Troubleshooting

Import Error

```
# If you get import errors, ensure installation:  
pip install --upgrade constrained-intelligence-constants
```

Insufficient Data

```
# Most discovery methods need at least 3-5 data points  
data = [1, 2, 3, 4, 5] # ✓ Good  
data = [1, 2]           # ✗ Too few
```

Numerical Stability

```
# Avoid division by zero or extreme values  
from constrained_intelligence import NUMERICAL_TOLERANCE  
  
if abs(value) > NUMERICAL_TOLERANCE:  
    result = numerator / value
```

Getting Help

- **Documentation:** Check [README.md](#) (README.md)
- **Examples:** Browse [examples/](#) (examples/)
- **Issues:** [GitHub Issues](#) (<https://github.com/yourusername/constrained-intelligence-constants/issues>)
- **Discussions:** [GitHub Discussions](#) (<https://github.com/yourusername/constrained-intelligence-constants/discussions>)

Ready to dive deeper? Check out the [full documentation](#) (README.md)!