0DTE Options Trading - Backtesting System

Overview

This backtesting system allows you to test your 0DTE options trading strategies using saved market data. You can run tests when markets are closed, optimize parameters, and analyze performance without risking real capital.

Components

1. **data_collector.py** - Market Data Collection

Collects and saves market data from Interactive Brokers during market hours.

2. **mock_execution_engine.py** - Simulated Trading

Simulates order execution using saved market data with realistic fill models.

3. **orchestrator_tester.py** - Backtest Orchestrator

Runs complete backtest sessions using saved data and generates performance reports.

Quick Start

Step 1: Collect Market Data (During Market Hours)

```bash
# Basic collection (runs for 6.5 hours with 5-minute intervals)
python data_collector.py

# Custom collection settings
python data_collector.py --days 90 --interval 1 --duration 1.0

# Options:
#   --days: Number of historical days to collect (default: 60)
#   --interval: Minutes between snapshots (default: 5)
#   --duration: Hours to run collection (default: 6.5)
#   --paper/--live: Use paper or live connection (default: paper)
#   --data-dir: Directory to save data (default: "market_data")
```

Step 2: Run Backtest (Anytime)

```bash
```

```
# Basic backtest (last 7 days)
python orchestrator_tester.py

# Custom backtest
python orchestrator_tester.py \
  --start-date 2024-12-01 \
  --end-date 2024-12-15 \
  --capital 25000 \
  --max-trades 3 \
  --fill-model realistic \
  --style balanced

# Options:
#   --start-date: Backtest start date
#   --end-date: Backtest end date
#   --capital: Initial capital (default: 15000)
#   --max-trades: Max trades per day (default: 5)
#   --fill-model: optimistic/realistic/pessimistic (default: realistic)
#   --style: safe/balanced/aggressive (default: balanced)
#   --data-dir: Data directory (default: "market_data")
```

Step 3: Test Mock Execution Engine

```bash
# Test that mock execution works correctly
python mock_execution_engine.py
```

Data Collection Schedule

Recommended Collection Times (US Eastern Time)

- **Pre-market**: 9:00 AM - 9:30 AM (setup and initial data)
- **Morning session**: 9:30 AM - 12:00 PM (active trading)
- **Afternoon session**: 12:00 PM - 4:00 PM (0DTE management)

What Gets Collected

- Historical price data (60 days of daily bars)
- Intraday price data (5-minute bars)
- Complete option chains with Greeks
- VIX data
- Market statistics

- Account snapshots

Directory Structure

```
your_trading_system/
│
├── market_data/              # Collected market data
│   ├── historical/           # Historical price data
│   │   ├── NQ_daily_*.csv
│   │   └── NQ_intraday_*.csv
│   ├── snapshots/            # Market snapshots
│   │   └── snapshot_*.pkl
│   └── options/              # Option chain data
│       └── NQ_options_*.pkl
│
├── backtest_reports/         # Backtest results
│   ├── backtest_result_*.pkl   # Full result object
│   ├── backtest_summary_*.json # JSON summary
│   └── daily_stats_*.csv       # Daily statistics
│
└── logs/                     # System logs
    ├── backtests/            # Backtest logs
    └── collection_*.log      # Data collection logs
```

Fill Models

Optimistic

- Fills at 25% into the spread
- No market impact
- Minimal slippage
- Best-case scenario

Realistic (Default)

- Fills at mid-price
- 5 cents market impact per contract
- Moderate slippage
- Real-world approximation

Pessimistic

- Fills at 75% into the spread
- 10 cents market impact per contract

- Higher slippage
- Worst-case scenario

Trading Styles

Safe

- Strike distance: 1.5x ATR
- Full position size
- Min 70% win probability
- Conservative approach

Balanced (Default)

- Strike distance: 1.0x ATR
- 90% position size
- Min 60% win probability
- Standard approach

Aggressive

- Strike distance: 0.7x ATR
- 60% position size
- Min 55% win probability
- Higher risk/reward

Performance Metrics

The backtest generates comprehensive performance reports including:

- **Returns**: Total return, daily returns, equity curve
- **Risk Metrics**: Sharpe ratio, max drawdown, volatility
- **Trade Statistics**: Win rate, profit factor, average win/loss
- **Position Analysis**: Strategy breakdown, holding times
- **Daily Statistics**: Daily P&L, trades per day, commissions

Tips for Effective Backtesting

1. Collect Diverse Market Data

- Collect data across different market conditions
- Include high volatility days (VIX > 20)

- Capture trend days and range days
- Get FOMC and earnings days

2. Test Multiple Scenarios

```bash
# Test different market conditions
for style in safe balanced aggressive; do
    python orchestrator_tester.py --style $style
done
```

3. Analyze Drawdowns

- Pay attention to max drawdown
- Check consecutive losses
- Monitor position concentration

4. Validate Results

- Compare backtest results with paper trading
- Check for unrealistic fills
- Verify commission calculations

5. Parameter Optimization

- Test different wing widths
- Vary profit targets
- Adjust stop losses
- Try different times of day

Advanced Usage

Custom Market Scenarios

You can create specific market scenarios for testing:

```python
```

```
# In your test script
from data_collector import MarketSnapshot
import pickle

# Load a snapshot
with open("market_data/snapshots/snapshot_20241201_100000.pkl", "rb") as f:
    snapshot = pickle.load(f)

# Modify for testing
snapshot.vix_data["last"] = 30.0  # High VIX scenario
snapshot.underlying_price *= 0.98  # 2% gap down

# Save modified snapshot
with open("market_data/snapshots/test_scenario.pkl", "wb") as f:
    pickle.dump(snapshot, f)
```

Batch Backtesting

Create a script to run multiple backtests:

```python
```

```python
#!/usr/bin/env python3
import asyncio
from datetime import datetime, timedelta
from orchestrator_tester import OrchestratorTester, BacktestConfig

async def run_batch_tests():
    """Run multiple backtests with different parameters."""

    results = []

    # Test different styles
    for style in ["safe", "balanced", "aggressive"]:
        config = BacktestConfig(
            start_date=datetime.now() - timedelta(days=30),
            end_date=datetime.now(),
            style=style,
        )

        tester = OrchestratorTester(config)
        await tester.initialize()
        result = await tester.run_backtest()
        results.append((style, result))

    # Compare results
    for style, result in results:
        print(f"\n{style.upper()} Strategy:")
        print(f"  Return: {result.performance['total_return']:.2f}%")
        print(f"  Sharpe: {result.performance['sharpe_ratio']:.2f}")
        print(f"  Max DD: {result.performance['max_drawdown']:.2f}%")

if __name__ == "__main__":
    asyncio.run(run_batch_tests())
```

Troubleshooting

No snapshots found

- Ensure data_collector.py ran successfully
- Check the market_data/snapshots/ directory
- Verify date range matches collected data

Connection errors

- Ensure TWS/IB Gateway is running (for data collection)
- Check port settings (7497 for paper, 7496 for live)

- Verify API permissions in TWS

Import errors

```bash
# Install required packages
pip install ib_insync pandas numpy click rich pytz
```

Memory issues with large datasets

- Reduce the number of days in backtest
- Use fewer snapshots (increase interval)
- Process data in batches

Best Practices

1. **Always collect data during different market conditions**
2. **Test with realistic commission and slippage**
3. **Validate against paper trading results**
4. **Don't over-optimize on historical data**
5. **Consider market regime changes**
6. **Account for early assignment risk in 0DTE**
7. **Test position management rules thoroughly**

Support Files

requirements.txt

```txt
ib_insync>=0.9.86
pandas>=2.1.4
numpy>=1.26.2
click>=8.1.7
rich>=13.7.0
pytz>=2023.3
nest_asyncio>=1.5.8
```

Install all requirements

```bash
pip install -r requirements.txt
```

Next Steps

1. **Collect initial data**: Run data_collector.py for at least one full trading day
2. **Run first backtest**: Use default settings to verify everything works
3. **Analyze results**: Review the reports in backtest_reports/
4. **Optimize parameters**: Test different configurations
5. **Validate with paper trading**: Compare backtest results with live paper trading

Questions?

Check the logs in the `logs/` directory for detailed debugging information. Each component creates detailed logs that can help troubleshoot issues.

Remember: Past performance does not guarantee future results. Always validate backtest results with paper trading before using real capital.