Algorithmic Trading and Strategy Design

Theory, Implementation, and Best Practices

Chapter 1: Introduction to Algorithmic Trading

1.1 Definition and Context

Algorithmic trading refers to the automated execution of trading orders using predefined rules, models, or programs. This discipline combines finance, mathematics, statistics, and computer science to systematically generate, test, and deploy trading strategies in markets.

1.2 Components of an Algorithmic Trading System

- **Signal generation:** Identifies trading opportunities (price patterns, mean reversion, trend-following)
- **Order execution:** Sends buy/sell orders to the market (market, limit, stop orders)
- **Risk management:** Controls exposure, prevents catastrophic losses (stop-loss, position sizing)

Chapter 2: Core Strategy Types

2.1 Trend-Following

- Follows prevailing price trends using moving averages or breakout rules
- **Signal example:** Simple 50-200 SMA crossover
- **Implementation:**

```
import pandas as pd
ma_fast = prices.rolling(window=50).mean()
ma_slow = prices.rolling(window=200).mean()
signal = (ma_fast &gt; ma_slow).astype(int)
```

2.2 Mean Reversion

- Assumes price will revert to mean
- **Pairs trading:** Long one asset/short another when spreads deviate from historical mean

2.3 Statistical Arbitrage

- Seeks to exploit price inefficiencies using statistical models
- **Example:** Z-score for standardized mean reversion signals

2.4 Machine Learning Strategies

- Use ML models to predict returns, classify signals, or optimize strategy parameters
- **Popular models:** Logistic regression, Random Forest, XGBoost, SVM, Neural Networks
- **Implementation:**

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
X_train, y_train = ... # Feature matrix/labels from historical data
rf.fit(X_train, y_train)
pred_signal = rf.predict(X_test)
```

2.5 Reinforcement Learning in Trading

- RL agent learns optimal actions (buy/hold/sell) via rewards (PnL)
- **Q-Learning:** Off-policy RL for discrete action spaces

Chapter 3: Backtesting and Evaluation

3.1 Backtesting Principles

- Simulate strategy on historical data without look-ahead bias
- Record trades, returns, drawdown, and risk metrics

3.2 Performance Metrics

- Sharpe ratio, Sortino ratio, max drawdown, win rate
- **Implementation:**

```
strategy_returns = ... # From trade log
sharpe = strategy_returns.mean() / strategy_returns.std() * np.sqrt(252)
```

Chapter 4: Implementation Examples

4.1 Tooling

- **backtrader, zipline:** Python backtesting frameworks
- **pandas, numpy, statsmodels:** Data wrangling and statistical analysis

4.2 Example: Simple Mean Reversion

```
spread = asset1 - asset2
zscore = (spread - spread.mean()) / spread.std()
entry, exit = 2, 0.5
longs = (zscore &lt; -entry)
shorts = (zscore &gt; entry)
close_long = (zscore &gt; -exit)
close_short = (zscore &lt; exit)
```

4.3 Example: ML-Based Classifier

```
features = ... # e.g. lagged returns, volatility, volume
labels = (future_returns &gt; 0).astype(int)
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(features, labels)
pred_proba = clf.predict_proba(X_test)
```

Chapter 5: Strategy Robustness and Risk Controls

- Out-of-sample testing
- Walk-forward validation
- Transaction costs and slippage analysis
- Kelly criterion and position sizing

Chapter 6: Project Blueprints

- Build and backtest a moving average crossover strategy on S&P 500 data
- Implement pairs trading using cointegration tests
- Train an RL Q-Learning agent for simple trend-following

Additional Reading

- Marcos Lopez de Prado: Advances in Financial Machine Learning
- Narang: Inside the Black Box
- Ernest Chan: Algorithmic Trading (open summaries)

This textbook covers comprehensive theory, coding, and real-world practice for algorithmic trading system design, making it ideal for both self-study and project implementation.