Financial Documentation

Backtesting Methodology and Best Practices

Overview

TradePulse implements industry-standard backtesting methodologies with emphasis on realistic market simulation and robust performance measurement. The system addresses common backtesting pitfalls through careful design and implementation.

Backtesting Framework

1. Event-Driven Architecture

The system uses an event-driven approach that mirrors real trading:

Market Data → Strategy Logic → Signal Generation → Order Execution → Portfolio Update

Benefits: - Eliminates look-ahead bias - Realistic order timing and execution - Proper handling of market gaps and holidays

2. Realistic Market Simulation

Order Execution Modeling: - **Slippage**: Configurable slippage rates (default: 2 bps for crypto markets) - **Latency**: Realistic execution delays (5-60 seconds configurable) - **Market Impact**: Price impact modeling for large orders

Market Microstructure: - Bid-ask spread simulation - Liquidity constraints modeling - Partial fill simulation for large orders

3. Data Quality and Integrity

Data Validation: - Outlier detection and handling - Missing data interpolation - Corporate action adjustments - Survivorship bias elimination

Multi-Timeframe Consistency: - Proper aggregation from base timeframe - Alignment of different data sources - Handling of timezone differences

Performance Measurement Standards

1. Return Metrics

Total Return: Total Return = (Final Portfolio Value - Initial Capital) / Initial Capital

Annualized Return: Annualized Return = (1 + Total Return)^(252/Trading Days) - 1

Compound Annual Growth Rate (CAGR): cpp double calculate_cagr(double initial_value, double final_value, int days) { return std::pow(final_value / initial_value, 252.0 / days) - 1.0; }

2. Risk-Adjusted Metrics

Sharpe Ratio: Sharpe Ratio = (Portfolio Return - Risk-Free Rate) / Portfolio Volatility

Implementation: cpp double calculate_sharpe_ratio(const std::vector& returns) { double mean_return = std::accumulate(returns.begin(), returns.end(), 0.0) / returns.size(); double variance = 0.0; for (double ret : returns) { variance += (ret - mean_return) * (ret - mean_return); } double std_dev = std::sqrt(variance / returns.size()); return (std_dev == 0.0) ? 0.0 : mean_return / std_dev * std::sqrt(252); }

Sortino Ratio: Sortino Ratio = (Portfolio Return - Risk-Free Rate) / Downside Deviation

Maximum Drawdown: cpp double calculate_max_drawdown(const std::vector& equity_curve) { double peak = equity_curve[0]; double max_dd = 0.0; for (double value : equity_curve) { if (value > peak) peak = value; double drawdown = (peak - value) / peak; max_dd = std::max(max_dd, drawdown); } return max_dd; }

3. Risk Metrics

Value at Risk (VaR): cpp double calculate_var(const std::vector& returns, double confidence = 0.95) { std::vector sorted_returns = returns; std::sort(sorted_returns.begin(), sorted_returns.end()); size_t index = static_cast((1.0 - confidence) * sorted_returns.size()); return -sorted_returns[index]; }

Beta Calculation: cpp double calculate_beta(const std::vector& strategy_returns, const std::vector& market_returns) { // Calculate covariance and market variance double covariance = 0.0, market_variance = 0.0; // ... implementation details return (market_variance != 0.0) ? covariance / market_variance : 0.0; }

Risk Management Framework

1. Position Sizing

Fixed Fractional Method: cpp double calculate_position_size(double account_value, double risk_per_trade, double entry_price, double stop_loss) { double risk_amount = account_value * risk_per_trade; double risk_per_share = std::abs(entry_price - stop_loss); return risk_amount / risk_per_share; }

Kelly Criterion: cpp double kelly_fraction(double win_rate, double avg_win, double avg_loss) { if (avg_loss == 0) return 0; double win_loss_ratio = avg_win / std::abs(avg_loss); return (win_rate * win_loss_ratio - (1 - win_rate)) / win_loss_ratio; }

2. Risk Limits

Portfolio-Level Limits: - Maximum position size: 10-15% of portfolio - Maximum daily loss: 2-5% of portfolio - Maximum drawdown: 5-10% of portfolio - Maximum number of positions: 3-5 concurrent

Asset-Level Limits: - Correlation limits between positions - Sector/geography exposure limits - Liquidity requirements

3. Dynamic Risk Management

Volatility-Based Position Sizing: cpp double volatility_adjusted_size(double base_size, double current_vol, double target_vol) { return base_size * (target_vol / current_vol); }

Drawdown-Based Scaling: cpp double drawdown_scaling(double base_size, double current_dd, double max_dd) { if (current_dd >= max_dd * 0.8) return base_size * 0.5; // Reduce size return base_size; }

Strategy Development Best Practices

1. Avoiding Common Pitfalls

Look-Ahead Bias Prevention: - All indicators use only historical data - No future information in decision making - Proper handling of data alignment

Survivorship Bias Mitigation: - Include delisted/failed assets in universe - Account for changing market composition - Use point-in-time data for universe selection

Overfitting Prevention: - Out-of-sample testing mandatory - Walk-forward analysis implementation - Parameter stability testing

2. Robust Strategy Design

Signal Generation: cpp // Example: Robust trend following with multiple confirmations bool generate_buy_signal(const MarketData& data) { bool trend_up = data.sma_short > data.sma_long; bool momentum_positive = data.rsi > 50 && data.rsi < 70; bool volume_confirmation = data.volume > data.avg_volume * 1.2;

```
return trend_up && momentum_positive && volume_confirmation;
```

}

Risk Controls: cpp bool risk_check_passed(const Position& position, const RiskLimits& limits) { if (position.size > limits.max_position_size) return false; if (position.correlation > limits.max_correlation) return false; if (portfolio.drawdown > limits.max_drawdown) return false; return true; }

3. Performance Attribution

Factor-Based Attribution: - Market beta exposure - Size factor (small vs large cap) - Value factor (value vs growth) - Momentum factor - Volatility factor

Trade-Level Analysis: cpp struct TradeAnalysis { double entry_timing_alpha; // Alpha from entry timing double exit_timing_alpha; // Alpha from exit timing double selection_alpha; // Alpha from asset selection double market_beta_return; // Return from market exposure };

Benchmark Comparison

1. Benchmark Selection

Appropriate Benchmarks:

- Market indices (S&P 500, NASDAQ for equities)

- Crypto indices (Bitcoin, Ethereum for crypto strategies)

- Risk-free rate (Treasury bills)

- Peer strategy performance

2. Statistical Significance Testing

T-Test for Alpha: cpp double calculate_alpha_t_stat(const std::vector& excess_returns) { double mean_excess = std::accumulate(excess_returns.begin(), excess_returns.end(), 0.0) / excess_returns.size(); double std_error = calculate_standard_error(excess_returns); return mean_excess / std_error; }

Information Ratio: cpp double information_ratio(const std::vector& strategy_returns, const std::vector& benchmark_returns) { std::vector excess_returns; for (size_t i = 0; i < strategy_returns.size(); ++i) { excess_returns.push_back(strategy_returns[i] - benchmark_returns[i]); }

```
double mean_excess = std::accumulate(excess_returns.begin(),
excess_returns.end(), 0.0)
                    / excess_returns.size();
double tracking_error = calculate_standard_deviation(excess_returns);
return (tracking_error != 0.0) ? mean_excess / tracking_error : 0.0;}
```

Regulatory and Compliance Considerations

1. Record Keeping

Trade Records: - Complete audit trail of all decisions - Timestamp accuracy to millisecond precision - Strategy version and parameter tracking - Market data version control

Performance Reporting: - GIPS-compliant performance calculation - Proper handling of cash flows - Composite construction methodology - Risk disclosure requirements

2. Risk Disclosure

Key Risk Factors: - Market risk and volatility - Liquidity risk in stressed markets - Model risk and parameter sensitivity - Technology and operational risks - Regulatory and compliance risks

Performance Disclaimers: - Past performance not indicative of future results - Backtesting limitations and assumptions - Model risk and parameter sensitivity - Market condition dependencies