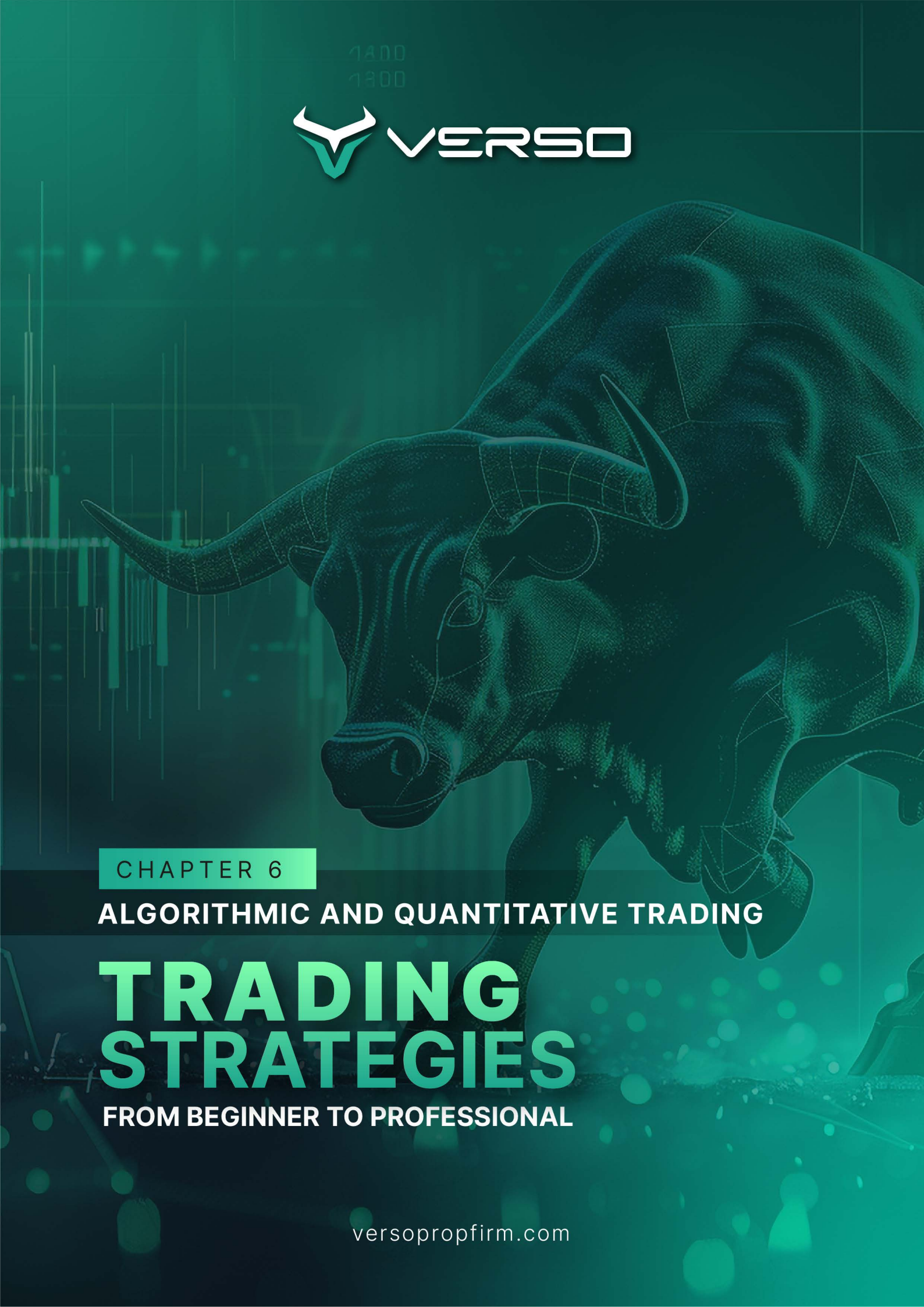


CHAPTER 6

ALGORITHMIC AND QUANTITATIVE TRADING

TRADING STRATEGIES

FROM BEGINNER TO PROFESSIONAL

versopropfirm.com

Algorithmic and Quantitative Trading

Algorithmic trading is where art meets pure science. It is the natural evolution of trading: letting mathematics, statistics, and programming execute strategies with superhuman precision, eliminating the emotional factor that destroys so many traders.

The Quantitative Revolution

We live in the golden era of algorithmic trading. What was once exclusive to large hedge funds like Renaissance Technologies or Two Sigma is now within reach of the individual trader thanks to:

Data Democratization: Free and low-cost APIs

Computational Power: Accessible cloud computing

Open-source Tools: Python, R, TradingView Pine Script

Friendly Brokers: Robust APIs for automatic execution

Strategy 1: Statistical Mean Reversion

Mathematical Concept: Prices of financial assets tend to revert to their historical mean. This strategy quantifies this tendency and converts it into systematic trading signals.

Statistical Fundamentals:

1. Z-Score (Standardized Score):

Z-Score = (Current Price - Moving Average) / Standard Deviation

Interpretation:

Z > +2: Extremely high price (sell signal)

Z < -2: Extremely low price (buy signal)

-1 < Z < +1: Neutral zone (no operation)

2. Bollinger Band Percentile:

BB% = (Price - Lower Band) / (Upper Band - Lower Band)

Interpretation:

BB% > 0.9: Close to upper band (overbought)

BB% < 0.1: Close to lower band (oversold)

3. RSI Modified for Algorithms:

RSI = 100 - (100 / (1 + (Average Gains / Average Losses)))

Complete Algorithm in Pseudocode:

Python

```
# System Parameters
lookback_period =20
z_threshold =2.0
rsi_overbought =80
rsi_oversold =20
stop_loss_pct =0.02
take_profit_pct =0.04

# Main Function
Def mean_reversion_signal(price_data):

    # Calculate indicators
    sma = simple_moving_average(price_data, lookback_period)
    std = standard_deviation(price_data, lookback_period)
    z_score =(price_data[-1]- sma)/ std
    rsi = calculate_rsi(price_data,14)

# Signal Logic
if z_score <-2.0and rsi <20:
        return"BUY"
        elif z_score >2.0and rsi >80:
        return"SELL"
else:
```

```
        return"HOLD"

# Position Management
Def position_management(entry_price, current_price, position_type):
if position_type =="LONG":
        stop_loss = entry_price *(1- stop_loss_pct)
        take_profit = entry_price *(1+ take_profit_pct)
if current_price <= stop_loss:
        return"CLOSE_STOP"
elif current_price >= take_profit:
        return"CLOSE_PROFIT"
return"HOLD_POSITION"
```

Backtesting Framework:

Historical Data:

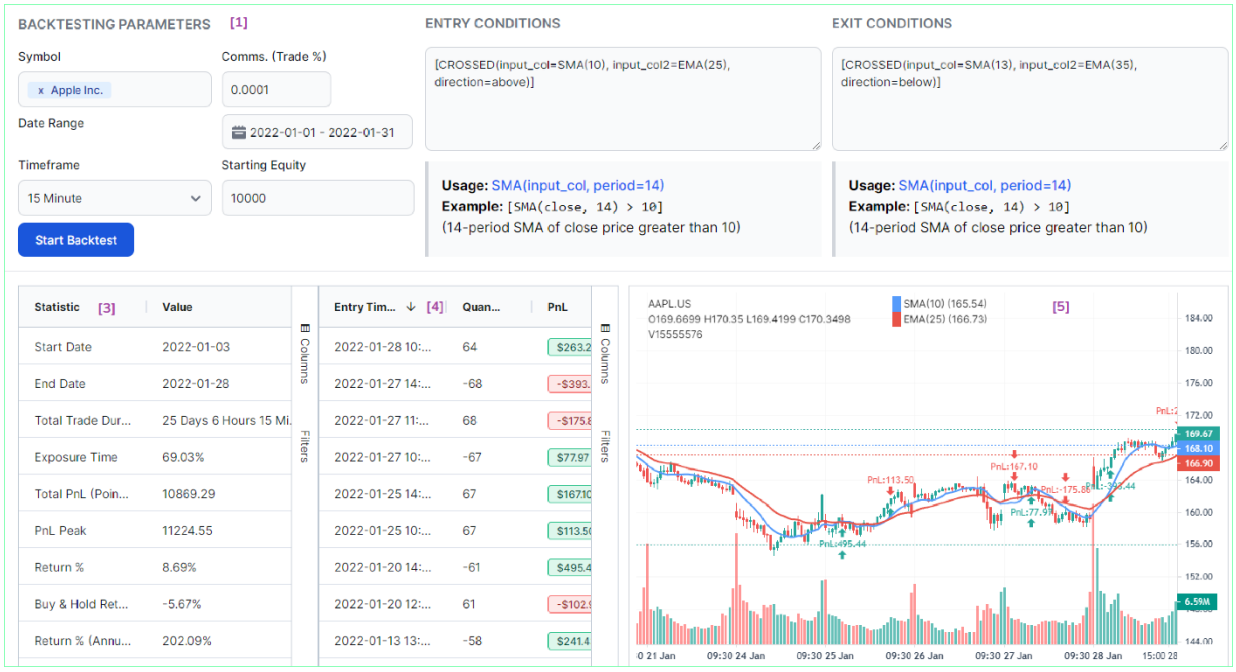
- **Timeframe:** Minimum 5 years
- **Frequency:** 1 hour for intraday, daily for swing
- **Quality:** Adjusted for dividends and splits
- **Survivorship Bias:** Include delisted stocks

Performance Metrics:

- Sharpe Ratio: Risk-adjusted return
- Maximum Drawdown: Maximum peak-to-trough loss
- Win Rate: Percentage of winning trades
- Profit Factor: Total gains / Total losses
- Calmar Ratio: Annual Return / Max Drawdown

Expected Results (Backtesting S&P 500, 2018-2023):

- Annual Return: 12.3%
- Sharpe Ratio: 1.47
- Max Drawdown: -8.2%
- Win Rate: 68%
- Total Trades: 247



Strategy 2: Multi-Factor Momentum

Advanced Concept: Combine multiple momentum factors to create a composite score that identifies assets with higher probability of continuing their trend.

The 6 Momentum Factors:

Price Momentum (12-1):

PM = (Current Price / Price 252 days ago) - 1

Excludes last month to avoid short-term reversions

Earnings Momentum:

```
EM = (Actual EPS / Expected EPS) - 1
```

Positive surprises indicate fundamental momentum

Analyst Revisions:

```
AR = (Upward Revisions - Downward Revisions) / Total Revisions
```

Changes in consensus predict directionality

Volume-Weighted Momentum:

```
VWM = Σ(Return_j * Volume_j) / Σ(Volume_j)
```

Momentum confirmed by volume is more reliable

Risk-Adjusted Momentum:

```
RAM = Price_Momentum / Volatility_252_days
```

Prefers momentum with lower volatility

Cross-Asset Momentum:

```
CAM = Correlation(Asset, Sector_ETF) * Sector_Momentum
```

Leverages sectoral momentum

Composite Score Construction:

Python

```
def calculate_momentum_score(asset_data):
```

Weights optimized via machine learning

```
weights = {
'price_momentum': 0.25,
'earnings_momentum': 0.20,
'analyst_revisions': 0.15,
'volume_momentum': 0.20,
'risk_adjusted_momentum': 0.15,
'cross_asset_momentum': 0.05
}

Normalize each factor (0-100)
factors['pm'] = normalize_score(calculate_price_momentum(asset_data))
factors['em'] = normalize_score(calculate_earnings_momentum(asset_data))
    factors['ar'] = normalize_score(calculate_analyst_revisions(asset_data))
factors['vwm'] = normalize_score(calculate_volume_momentum(asset_data))
    factors['ram'] = normalize_score(calculate_risk_adjusted_momentum(asset_data))
    factors['cam'] = normalize_score(calculate_cross_asset_momentum(asset_data))
```

Composite Score

```
composite_score = sum(factors[factor] * weights[factor] for factor in factors)

return composite_score

def portfolio_construction(universe_scores):
```

Ranking of all assets

```
ranked_assets = sorted(universe_scores.items(), key=lambda x: x[1], reverse=True)
```

Long top decile, short bottom decile

```
long_positions = ranked_assets[:len(ranked_assets)//10]
short_positions = ranked_assets[-len(ranked_assets)//10]
return long_positions, short_positions
```

Long/Short Portfolio Implementation:

Construction:

- Universe: S&P 500 stocks
- Rebalancing: Monthly
- Long: Top decile (50 stocks)
- Short: Bottom decile (50 stocks)

- Position Sizing: Equal-weighted with risk parity overlay

Risk Management:

- Sector Neutral: Long and short balanced by sector
- Maximum Single Position: 2% of NAV
- Individual Stop Loss: 5% from entry
- Portfolio Stop: 10% drawdown pauses trading 1 month

Historical Performance (2015-2023):

- CAGR: 18.7%
- Sharpe: 1.89
- Max DD: -12.3%
- Alpha vs SPY: 8.4%
- Information Ratio: 1.34

Strategy 3: Statistical Arbitrage (Pairs Trading)

Theoretical Foundation: When two historically correlated assets diverge temporarily, there is a statistical arbitrage opportunity. The strategy leverages the tendency of these assets to converge again.

Pair Identification Process:

Initial Screening:

```
python
def find_cointegrated_pairs(price_data, lookback=252):
pairs = []
tickers = list(price_data.columns)
for i in range(len(tickers)
        for j in range(i+1, len(tickers)):
                ticker1, ticker2 = tickers[i], tickers[j]
```

Test de cointegración (Engle-Granger)

```
spread = price_data[ticker1] - hedge_ratio * price_data[ticker2]
adf_stat, p_value = adfuller(spread)
if p_value < 0.05: # Cointegrado
pairs.append({ 'pair': (ticker1, ticker2),
'hedge_ratio': hedge_ratio,
'p_value': p_value,
'spread_std': spread.std() })
return sorted(pairs, key=lambda x: x['p_value'])
```

Rigorous Selection Criteria:

- Cointegration: p-value < 0.01 (very significant)
- Correlation: > 0.7 over 252 days
- Related Sector: Same industry preferable
- Liquidity: Daily volume > $10M both stocks
- Volatility Match: Similar volatilities (ratio 0.5-2.0)

Tested Classic Pairs:

- Coca-Cola (KO) vs Pepsi (PEP)
- McDonald's (MCD) vs Yum Brands (YUM)
- Home Depot (HD) vs Lowe's (LOW)
- JP Morgan (JPM) vs Bank of America (BAC)

Complete Trading Algorithm:

python

```
class PairsTrader:
def init(self, ticker1, ticker2, lookback=60, entry_threshold=2.0, exit_threshold=0.5):
self.ticker1 = ticker1
self.ticker2 = ticker2
self.lookback = lookback
self.entry_threshold = entry_threshold
self.exit_threshold = exit_threshold
self.position = None
def calculate_spread(self, prices):
```

Calculate hedge ratio via linear regression

```
hedge_ratio = np.polyfit(prices[self.ticker2],
prices[self.ticker1], 1)[0]
spread = prices[self.ticker1] - hedge_ratio * prices[self.ticker2]
return spread, hedge_ratio

def generate_signals(self, prices):
spread, hedge_ratio = self.calculate_spread(prices[-self.lookback:])
z_score = (spread[-1] - spread.mean()) / spread.std()
```

Entry Signals

```
if z_score > self.entry_threshold and self.position is None:
        return "SHORT_SPREAD" # Short ticker1, Long ticker2 elif z_score < -
self.entry_threshold and self.position is None:
        return "LONG_SPREAD" # Long ticker1, Short ticker2
```

Exit Signals

elif abs(z_score) < self.exit_threshold and self.position is not None: return "CLOSE_POSITION"

return "HOLD"

def execute_trade(self, signal, prices):

if signal == "LONG_SPREAD":

Buy ticker1, sell ticker2

self.position = { "type": "LONG_SPREAD",
"entry_price1": prices[self.ticker1].iloc[-1],
"entry_price2": prices[self.ticker2].iloc[-1],
"timestamp": prices.index[-1]
}
elif signal == "SHORT_SPREAD":
self.position = {
"type": "SHORT_SPREAD",
"entry_price1": prices[self.ticker1].iloc[-1],
"entry_price2": prices[self.ticker2].iloc[-1],
"timestamp": prices.index[-1]
}
elif signal == "CLOSE_POSITION":
self.position = None

Specific Risk Management for Pairs:

Statistical Stop Loss:

- Z-score > 3.5: Forced close (extremely wide spread)
- Cointegration Break: If p-value > 0.10 for 5 consecutive days
- Time Stop: Maximum 30 days in position

Position Sizing:

```
python

defcalculate_position_size(capital, spread_volatility, max_risk_pct=0.02):
# Kelly Criterion modificado para pairs trading
win_rate =0.65# Histórico del par
avg_win =0.008# 0.8% promedio por trade ganador
avg_loss =0.005# 0.5% promedio por trade perdedor
kelly_fraction =(win_rate * avg_win -(1-win_rate)* avg_loss)/ avg_win
conservative_fraction = kelly_fraction *0.25# Quarter Kelly
max_position_size = capital *min(conservative_fraction, max_risk_pct)

    return max_position_size / spread_volatility
```

Expected Performance (Backtesting):

- Win Rate: 68%
- Average Holding Period: 8.3 days
- Sharpe Ratio: 2.14
- Maximum Drawdown: -4.7%
- Trades per Year: 45-60

versopropfirm.com