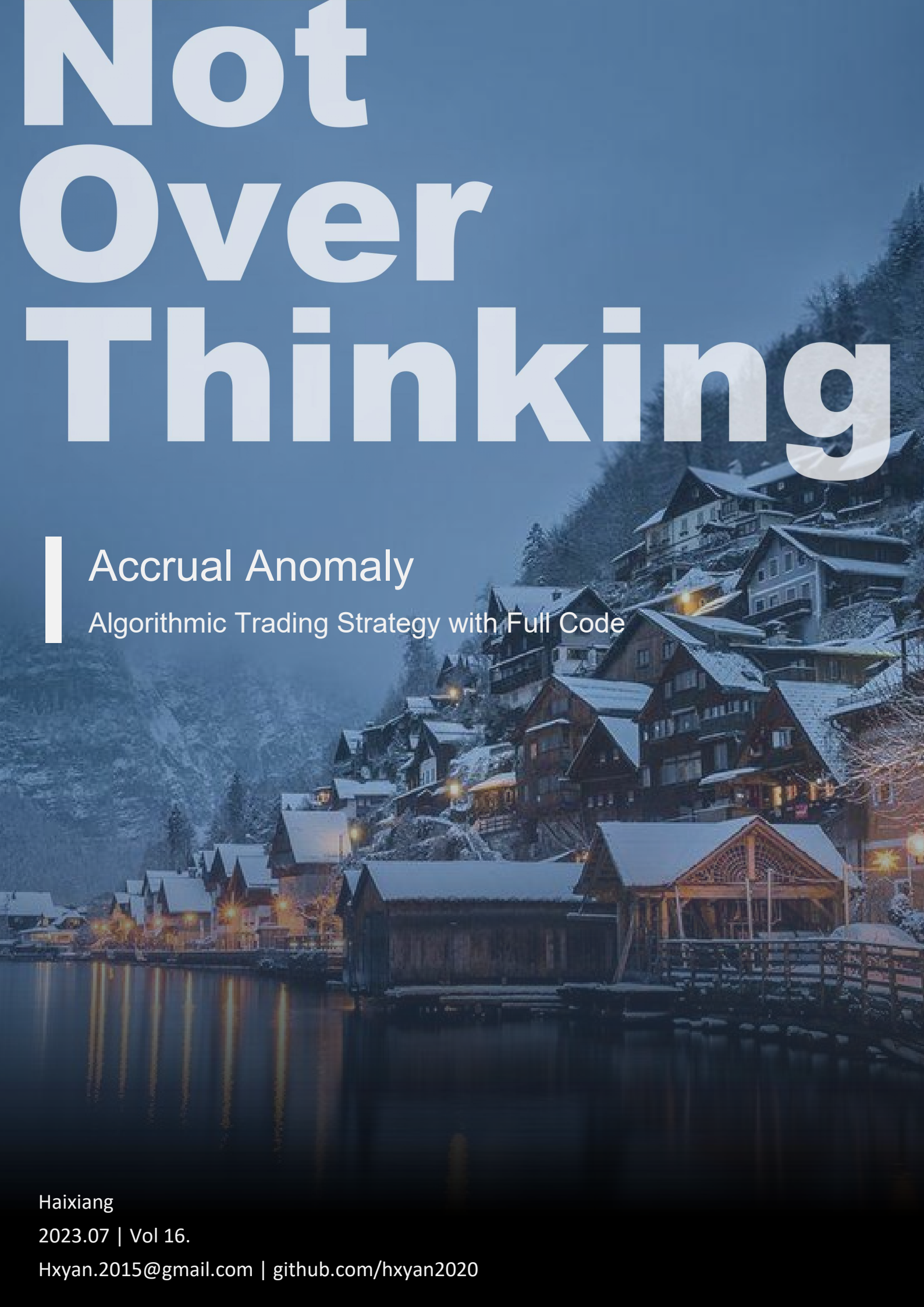
Not Over Thinking



Accrual Anomaly

Algorithmic Trading Strategy with Full Code

Haixiang

2023.07 | Vol 16.

Hxyan.2015@gmail.com | github.com/hxyan2020

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of all stocks on NYSE, AMEX, and NASDAQ. Balance sheet based accruals (the non-cash component of earnings) are calculated as:

$$BS\_ACC = (\Delta CA - \Delta Cash) - (\Delta CL - \Delta STD - \Delta ITP) - Dep$$

Where:
ΔCA = annual change in current assets
ΔCash = change in cash and cash equivalents
ΔCL = change in current liabilities
ΔSTD = change in debt included in current liabilities
ΔITP = change in income taxes payable
Dep = annual depreciation and amortization expense

BUY	SELL
The stocks with lowest accruals	The stocks with highest accruals

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equities
FINANCIAL INSTRUMENTS	Stocks
REGION	United States
PERIOD OF REBALANCING	Yearly
NO. OF TRADED INSTRUMENTS	1,000
LONG/SHORT	Long and Short

ALGORITHM

```
from AlgorithmImports import *

class AccrualAnomaly(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2006, 1, 1)
        self.SetCash(100000)

        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol

        self.coarse_count = 1000

        self.long = []
        self.short = []

        # Latest accruals data.
        self.accrual_data = {}

        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
        self.Schedule.On(self.DateRules.MonthEnd(self.symbol), self.TimeRules.AfterMarketOpen(s
elf.symbol), self.Selection)
```

```
    def OnSecuritiesChanged(self, changes):
        for security in changes.AddedSecurities:
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(5)

        for security in changes.RemovedSecurities:
            symbol = security.Symbol
            if symbol in self.accrual_data:
                del self.accrual_data[symbol]

    def CoarseSelectionFunction(self, coarse):
        if not self.selection_flag:
            return Universe.Unchanged

        # selected = [x.Symbol for x in coarse if x.HasFundamentalData and x.Market == 'usa']
        selected = [x.Symbol
            for x in sorted([x for x in coarse if x.HasFundamentalData and x.Market == 'usa'],
                key = lambda x: x.DollarVolume, reverse = True)[:self.coarse_count]]

        return selected

    def FineSelectionFunction(self, fine):
        fine = [x for x in fine if (float(x.FinancialStatements.BalanceSheet.CurrentAssets.Twel
veMonths) != 0)
                                   and (float(x.FinancialStatements.BalanceSheet.CashAndCashEquiva
lents.TwelveMonths) != 0)
                                   and (float(x.FinancialStatements.BalanceSheet.CurrentLiabilitie
s.TwelveMonths) != 0)
                                   and (float(x.FinancialStatements.BalanceSheet.CurrentDebt.Twelv
eMonths) != 0)
                                   and (float(x.FinancialStatements.BalanceSheet.IncomeTaxPayable.
TwelveMonths) != 0)
                                   and (float(x.FinancialStatements.IncomeStatement.DepreciationAn
dAmortization.TwelveMonths) != 0)]

        if len(fine) > self.coarse_count:
            sorted_by_market_cap = sorted(fine, key = lambda x: x.MarketCap, reverse=True)
            top_by_market_cap = sorted_by_market_cap[:self.coarse_count]
        else:
            top_by_market_cap = fine

        accruals = {}
        for stock in top_by_market_cap:
            symbol = stock.Symbol

            if symbol not in self.accrual_data:
                self.accrual_data[symbol] = None

            # Accrual calc.
            current_accruals_data = AccrualsData(stock.FinancialStatements.BalanceSheet.Current
Assets.TwelveMonths, stock.FinancialStatements.BalanceSheet.CashAndCashEquivalents.TwelveMonths,
                                                 stock.FinancialStatements.BalanceSheet.CurrentL
iabilities.TwelveMonths, stock.FinancialStatements.BalanceSheet.CurrentDebt.TwelveMonths, stock.
FinancialStatements.BalanceSheet.IncomeTaxPayable.TwelveMonths,
                                                 stock.FinancialStatements.IncomeStatement.Depre
ciationAndAmortization.TwelveMonths, stock.FinancialStatements.BalanceSheet.TotalAssets.TwelveM
onths)

            # There is not previous accrual data.
            if not self.accrual_data[symbol]:
                self.accrual_data[symbol] = current_accruals_data
                continue

            # Accruals and market cap calc.
```

```
            acc = self.CalculateAccruals(current_accruals_data, self.accrual_data[symbol])
            accruals[symbol] = acc

            # Update accruals data.
            self.accrual_data[symbol] = current_accruals_data

        # Accruals sorting.
        sorted_by_accruals = sorted(accruals.items(), key = lambda x: x[1], reverse = True)
        decile = int(len(sorted_by_accruals) / 10)
        self.long = [x[0] for x in sorted_by_accruals[-decile:]]
        self.short = [x[0] for x in sorted_by_accruals[:decile]]

        return self.long + self.short

    def OnData(self, data):
        if not self.selection_flag:
            return
        self.selection_flag = False

        # Trade execution.
        stocks_invested = [x.Key for x in self.Portfolio if x.Value.Invested]
        for symbol in stocks_invested:
            if symbol not in self.long:
                self.Liquidate(symbol)

        for symbol in self.long:
            self.SetHoldings(symbol, 1 / len(self.long))
        for symbol in self.short:
            self.SetHoldings(symbol, -1 / len(self.short))

        self.long.clear()
        self.short.clear()

    def Selection(self):
        if self.Time.month == 4:
            self.selection_flag = True

    def CalculateAccruals(self, current_accrual_data, prev_accrual_data):
        delta_assets = current_accrual_data.CurrentAssets - prev_accrual_data.CurrentAssets
        delta_cash = current_accrual_data.CashAndCashEquivalents - prev_accrual_data.CashAndCas
hEquivalents
        delta_liabilities = current_accrual_data.CurrentLiabilities - prev_accrual_data.Current
Liabilities
        delta_debt = current_accrual_data.CurrentDebt - prev_accrual_data.CurrentDebt
        delta_tax = current_accrual_data.IncomeTaxPayable - prev_accrual_data.IncomeTaxPayable
        dep = current_accrual_data.DepreciationAndAmortization
        avg_total = (current_accrual_data.TotalAssets + prev_accrual_data.TotalAssets) / 2

        bs_acc = ((delta_assets - delta_cash) - (delta_liabilities - delta_debt - delta_tax) -
dep) / avg_total
        return bs_acc

class AccrualsData():
    def __init__(self, current_assets, cash_and_cash_equivalents, current_liabilities, current_
debt, income_tax_payable, depreciation_and_amortization, total_assets):
        self.CurrentAssets = current_assets
        self.CashAndCashEquivalents = cash_and_cash_equivalents
        self.CurrentLiabilities = current_liabilities
        self.CurrentDebt = current_debt
        self.IncomeTaxPayable = income_tax_payable
        self.DepreciationAndAmortization = depreciation_and_amortization
        self.TotalAssets = total_assets

# Custom fee model.
```

```
class CustomFeeModel(FeeModel):
    def GetOrderFee(self, parameters):
        fee = parameters.Security.Price * parameters.Order.AbsoluteQuantity * 0.00005
        return OrderFee(CashAmount(fee, "USD"))
```

BACKTESTING PERFORMANCE

Fig 1. Overall Performance

PSR	0.000%	Sharpe Ratio	-0.25
Total Trades	1264	Average Win	1.42%
Average Loss	-1.57%	Compounding Annual Return	-6.463%
Drawdown	77.800%	Expectancy	-0.070
Net Profit	-68.187%	Loss Rate	51%
Win Rate	49%	Profit-Loss Ratio	0.90
Alpha	-0.029	Beta	-0.083
Annual Standard Deviation	0.142	Annual Variance	0.02
Information Ratio	-0.498	Tracking Error	0.227
Treynor Ratio	0.425	Total Fees	$154.87
Estimated Strategy Capacity	$15000000.00	Lowest Capacity Asset	FELE R735QTJ8XC9X

Fig 2. Performance Metrics

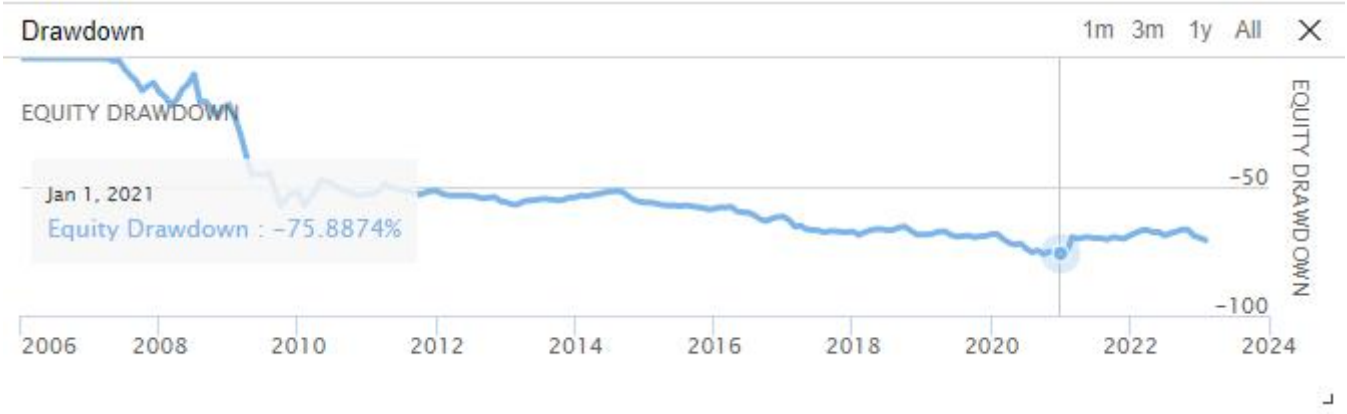


Fig 3. Drawdown

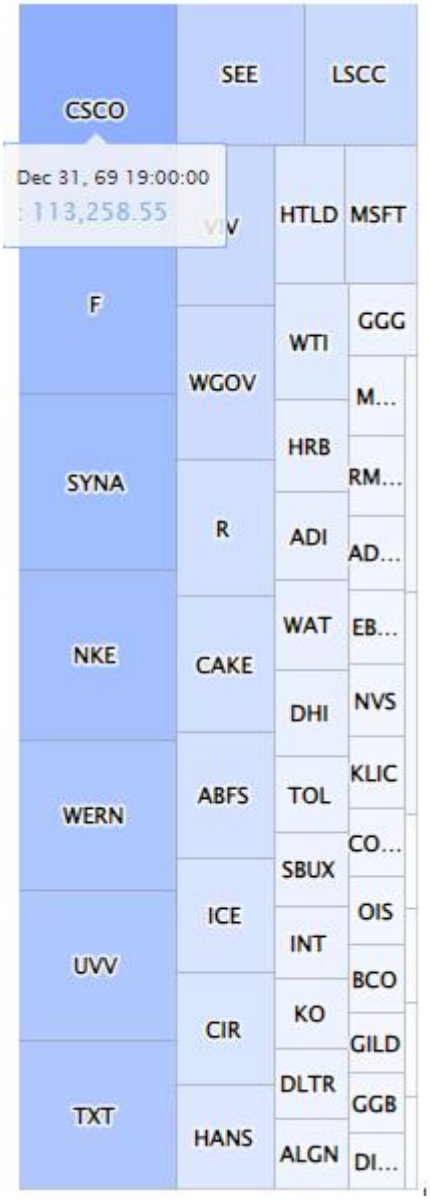


Fig 4. Assets Sales Volume