Back to Community

Pair Trade with Cointegration and Mean-Reversion Tests

Justin Lent posted Jul 29, 2015

mean reversion pairs trading Show Q tools and tips

Attached is a pair trading algo that allows the user to toggle on/off different tests for cointegration/mean-reversion of the pair's spread prior to taking any trades. If you choose to turn on one of the tests, the value from the test is recorded as a timeseries viewable from the backtest results page.

The pair being traded in this algo is the oil and gold ETFs (USO and GLD), but you can modify these as you wish.

The 3 different tests are:

- Augmented Dickey-Fuller test p-value
  -- Effectively, this is a unit-root test for determining whether the spread is cointegrated
  -- As well, a function is included showing how to use the critical-values from the ADF-test instead of p-value
- Half-life of mean-reversion, computed from an Ornstein–Uhlenbeck process
  -- This is the theoretically computed time, based on a historical window of data, that it will take for the spread to mean-revert half of its distance after having diverged from the mean of the spread
- Hurst exponent
  -- Effectively this returns a value between 0 and 1 that tells you whether a time-series is trending or mean-reverting. The closer the value is to 0.5 means the more "random" the time-series has behaved historically. Values below 0.5 imply the time-series is mean-reverting, and above 0.5 imply trending. The closer the value is to 0 implies greater levels of mean-reversion.
  -- Trading literature is conflicted as to the usefulness of Hurst exponent, but I included it nonetheless, and have set the default switch to False in the algo.

  The backtest results below incorporate two of these tests:

- ADF-test p-value, computed over a 63-day (e.g. 3-months) lookback window, with a required minimum p-value of 0.20

- Half-life days, computed over 126-day (6-month) lookback window, with a requirement of the half-life being between 1 and 42-days (2-months)

To modify the parameter values of the tests just look in the initialize function, for blocks of code that look like this. Here is how the ADF-test p-value parameters are defined:

```
context.stat_filter = {}
context.stat_filter['adf_p_value'] = {}

context.stat_filter['adf_p_value']['use'] = True
context.stat_filter['adf_p_value']['lookback'] = 63
context.stat_filter['adf_p_value']['function'] = adf_p_value
context.stat_filter['adf_p_value']['test_condition_min'] = 0.0
context.stat_filter['adf_p_value']['test_condition_max'] = 0.20
```

Here you see how there is a dictionary defined called 'stat_filter' which you can use to store the parameters of each test. First I create another dictionary inside of 'stat_filter' named 'adf_p_value' and then I load in all of the parameter values relevent to the ADF-test that I want to define when it is acceptable to enter a trade. These exact 5 parameters (e.g. keys of the dictionary) will be defined for all of the tests, as you'll see if you look at the algo code, and notice the adf_critical_value, half_life, hurst_exponent ones are defined following it. The 5 parameters are:

- 'use': Boolean, True if you want the algo to use this test
- 'lookback': Integer, value of how many lookback periods of the timeseries to be used in running the computation
- 'function': Function, this is the name of the function that will be called and return a value back to be compared to the _min and _max conditions below
- 'test_condition_min': Integer or Float, the minimum value returned by 'function' to determine if a trade can be triggered
- 'test_condition_max': Integer or Float, the maximum returned by 'function' to determine if a trade can be triggered

Support for Intraday Frequency
(Let me know if you run into issues with this, as I haven't done as much testing with it as I have with just daily freq)

You can configure this algo to be run on intraday minutely data as well. E.g. construct a pair spread using 15-min bar closing prices.
First, change the variable 'context.trade_freq' from 'daily' to 'intraday':

```
context.trade_freq = 'daily' # 'daily' or 'intraday'
```

Then, look for this code block below in the initialize() function, and specify the 'intraday_freq' value for the frequency of closing prices to use (E.g. 15 minute bars). Then, set 'run_trading_logic' to be how frequently you want the logic to be applied to market data. I chose 60 which means, run this logic every 60-minutes, but if you wish, change it to 1, and the logic will be run every single minute (beware though, as this will result in really long backtest times).
The variable 'check_exit_every_minute' can be set to True if you want the logic to be run *every minute if-and-only-if* you are currently in a trade. E.g. it checks to see whether you need to exit the trade every minute rather than waiting to the next N periods (e.g. 60 minutes, as specified in the 'run_trading_logic_freq' variable)

```
### START: INTRADAY FREQUENCY PARAMETERS
    context.intraday_freq = 15                   # only used if context.trade_freq='intraday'
    context.run_trading_logic_freq = 60    # only used if context.trade_freq='intraday'
    context.check_exit_every_minute = False    # if True, and if in a trade, will check every minute whether to exit
### END: INTRADAY FREQUENCY PARAMETERS
```

Performance Source Code Migrated Source Code

```
# Backtest ID: 55b982b8bedc2f0c743a86d9
import numpy as np
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts
import pandas as pd
from zipline.utils import tradingcalendar
import pytz


def initialize(context):
    '''
    START: USER CONFIGURABLE PARAMETERS FOR THE BACKTEST
    '''
    # Quantopian backtester specific variables
    set_slippage(slippage.VolumeShareSlippage(volume_limit=0.025, price_impact=0.1))
    set_commission(commission.PerShare(cost=0.0075, min_trade_cost=0.0))
    #set_slippage(slippage.VolumeShareSlippage(volume_limit=0.99, price_impact=0.0))
    #set_commission(commission.PerShare(cost=0.00, min_trade_cost=0.0))

    # the ticker symbols for your pair
    context.y = symbol('USO')
    context.x = symbol('GLD')

    # strategy specific variables
    context.lookback = 20     # used for regression
    context.z_window = 20     # used for zscore calculation, must be <= lookback
    context.entry_z = 0.5     # trade entry triggered when spread is + or - entryZ
    context.exit_z = 0.0      # trade exit triggered when spread is + or - entryZ
    context.momentum_or_mean_reversion = 'mean-reversion'     # 'momentum' or 'mean-reversion'
    # This defines whether to use a hedge ratio computed N periods ago.
    # Rationale for this is that since the algo is trading off of mean reversion
    # that a hedge ratio that excludes N days of recency, e.g. when severe divergences
    # could have occured, and which this algo hopes to exploit,  may be more aligned
    # to the economic historical relationship befitting of the stock pair
    context.use_hedge_ratio_lag = True
    context.hedge_ratio_lag = 2


    ####### START: TRADING FREQUENCY DEFINTIONS #####
    context.trade_freq = 'daily'             # 'daily' or 'intraday'

    ### START: DAILY FREQUENCY PARAMETERS
    # if context.trade_freq == 'intraday' then specify these inputs
    # only run the trading logic at 3:30 Eastern time, which 30-minutes before market closes
    context.daily_run_logic_hours = 15          # only used if context.trade_freq='daily'
    context.daily_run_logic_minutes = 30        # only used if context.trade_freq='daily'
    ### END: DAILY FREQUENCY PARAMETERS

    ### START: INTRADAY FREQUENCY PARAMETERS
    # if context.trade_freq == 'intraday' then specify these inputs
    context.intraday_freq = 15             # only used if context.trade_freq='intraday'
    context.run_trading_logic_freq = 60    # only used if context.trade_freq='intraday'
    context.check_exit_every_minute = False    # if True, and if in a trade, will check every minute whether to exit
    ### END: INTRADAY FREQUENCY PARAMETERS

    ####### END: TRADING FREQUENCY DEFINTIONS #####


    ####### START: STATISTICAL FILTERS DEFINITIONS #####
    # Specify 'statistical filters' (e.g. cointegration, half-life of mean reversion, etc)
    # that the pair's spread must pass before before it will be considered for a trade
    context.stat_filter = {}
    context.stat_filter['adf_p_value'] = {}
    context.stat_filter['adf_critical_value'] = {}
    context.stat_filter['half_life_days'] = {}
    context.stat_filter['hurst_exponent'] = {}

    context.stat_filter['adf_p_value']['use'] = True
    context.stat_filter['adf_p_value']['lookback'] = 63
    context.stat_filter['adf_p_value']['function'] = adf_p_value    # function defined below
    context.stat_filter['adf_p_value']['test_condition_min'] = 0.0
    context.stat_filter['adf_p_value']['test_condition_max'] = 0.20

    context.stat_filter['adf_critical_value']['use'] = False
    context.stat_filter['adf_critical_value']['lookback'] = 63
    context.stat_filter['adf_critical_value']['function'] = adf_critical_value_test    # function defined below
    context.stat_filter['adf_critical_value']['test_condition_min'] = int(0)           # see function description for return val
    context.stat_filter['adf_critical_value']['test_condition_max'] = int(10)          # see function description for return val

    context.stat_filter['half_life_days']['use'] = True
    context.stat_filter['half_life_days']['lookback'] = 126
    context.stat_filter['half_life_days']['function'] = half_life    # function defined below
    context.stat_filter['half_life_days']['test_condition_min'] = 1.0
    context.stat_filter['half_life_days']['test_condition_max'] = 42.0

    context.stat_filter['hurst_exponent']['use'] = False
    context.stat_filter['hurst_exponent']['lookback'] = 126
    context.stat_filter['hurst_exponent']['function'] = hurst        # function defined below
    context.stat_filter['hurst_exponent']['test_condition_min'] = 0.0
    context.stat_filter['hurst_exponent']['test_condition_max'] = 0.4
    ####### END: STATISTICAL FILTERS DEFINITIONS #####

    '''
    END: USER CONFIGURABLE PARAMETERS
    '''

    # define a few more global variables based on the inputs above
    context.intraday_history_lookback =  context.lookback * context.intraday_freq + 10
    context.spread = np.array([])
    context.hedge_ratio_history = np.array([])
    context.in_long = False
    context.in_short = False

    if not context.use_hedge_ratio_lag:
        # a lag of 1 means to include the most recent price in the hedge_ratio calculation
        # specificlly, this is used for np.array[-1] indexing
        context.hedge_ratio_lag = 1

# Will be called on every trade event for the securities you specify.
def handle_data(context, data):
    if get_open_orders():
        return
    now = get_datetime()
    exchange_time = now.astimezone(pytz.timezone('US/Eastern'))

    # Only trade N-minutes before market close
    if context.trade_freq == 'daily':
        if not (exchange_time.hour == context.daily_run_logic_hours and exchange_time.minute == context.daily_run_logic_minutes)
            return

    if context.trade_freq == 'intraday':
        # Only run trading logic every N minutes
        if exchange_time.minute % context.run_trading_logic_freq > 0:
            return

    if context.trade_freq == 'daily':
        prices = history(context.lookback, '1d', 'price').iloc[-context.lookback::]

    if context.trade_freq == 'intraday':
        prices = history(context.intraday_history_lookback, '1m', 'price')
        indexes_at_freq = range(prices.shape[0]-1, 0, -context.intraday_freq)
        indexes_at_freq.sort()
        prices = prices.ix[ indexes_at_freq, :]

    y = prices[context.y]
    x = prices[context.x]

    try:
        hedge = hedge_ratio(y, x, add_const=True)
    except ValueError as e:
        log.debug(e)
        return

    context.hedge_ratio_history = np.append(context.hedge_ratio_history, hedge)
    # Calculate the current day's spread and add it to the running tally
    if context.hedge_ratio_history.size < context.hedge_ratio_lag:
        return
    # Grab the previous day's hedge ratio
    hedge = context.hedge_ratio_history[-context.hedge_ratio_lag]
    context.spread = np.append(context.spread, y[-1] - hedge * x[-1])
    spread_length = context.spread.size
    #record(spd=context.spread[-1])

    # apply all the statistical filters to 'context.spread', if specified in initialize()
    # only test for trading triggers if the spread passes all specified criteria
    for stat_name in context.stat_filter:
        if context.stat_filter[stat_name]['use']:
            test_lookback = context.stat_filter[stat_name]['lookback']
            if spread_length < test_lookback:
                return
            test_spread = context.spread[-test_lookback:]
            test_func = context.stat_filter[stat_name]['function']
            test_value = test_func(test_spread)
            test_min = context.stat_filter[stat_name]['test_condition_min']
            test_max = context.stat_filter[stat_name]['test_condition_max']

            if stat_name == 'adf_p_value':
                record(adf_p=test_value)
            if stat_name == 'adf_critical_value':
                record(adf_cv=test_value)
            if stat_name == 'half_life_days':
                record(half_life=test_value)
            if stat_name == 'hurst_exponent':
                record(hurst=test_value)
            if (test_value < test_min) or (test_value > test_max):
                return

    if context.spread.size > context.z_window:
        # Keep only the z-score lookback period
        spreads = context.spread[-context.z_window:]

        zscore = (spreads[-1] - spreads.mean()) / spreads.std()
        record(Z=zscore)
        #record(gr_lev=context.account.leverage, net_lev=context.account.net_leverage)

        if context.in_short and zscore < context.exit_z:
            order_target(context.y, 0)
            order_target(context.x, 0)
            context.in_short = False
            context.in_long = False
            #record(in_trade=0)
            #record(stock_Y_pct=0, stock_X_pct=0)
            return

        if context.in_long and zscore > context.exit_z:
            order_target(context.y, 0)
            order_target(context.x, 0)
            context.in_short = False
            context.in_long = False
            #record(in_trade=0)
            #record(stock_Y_pct=0, stock_X_pct=0)
            return



        # only check for new trades every N minutes, but can exit existing trades _each_ minute
        # which is why this check is here. Trade exits are handled above.
        #if exchange_time.minute % context.run_trading_logic_freq > 0:
        #    return

        if zscore < -context.entry_z and (not context.in_long):
            # Only trade if NOT already in a trade
            y_target_shares = 1
            x_target_shares = -hedge
            context.in_long = True
            context.in_short = False

            (y_target_pct, x_target_pct) = compute_holdings_pct(y_target_shares,
                                                                 x_target_shares,
                                                                 y[-1], x[-1] )

            if context.momentum_or_mean_reversion == 'momentum':
                y_target_pct = -1.0 * y_target_pct
                x_target_pct = -1.0 * x_target_pct

            order_target_percent(context.y, y_target_pct)
            order_target_percent(context.x, x_target_pct)
            #record(in_trade=1)
            #record(stock_Y_pct=y_target_pct, stock_X_pct=x_target_pct)
            return

        if zscore > context.entry_z and (not context.in_short):
            # Only trade if NOT already in a trade
            y_target_shares = -1
            x_target_shares = hedge
            context.in_short = True
            context.in_long = False

            (y_target_pct, x_target_pct) = compute_holdings_pct(y_target_shares,
                                                                 x_target_shares,
                                                                 y[-1], x[-1] )

            if context.momentum_or_mean_reversion == 'momentum':
                y_target_pct = -1.0 * y_target_pct
                x_target_pct = -1.0 * x_target_pct

            order_target_percent(context.y, y_target_pct)
            order_target_percent(context.x, x_target_pct)
            #record(in_trade=1)
            #record(stock_Y_pct=y_target_pct, stock_X_pct=x_target_pct)

def is_market_close(dt):
    ref = tradingcalendar.canonicalize_datetime(dt)
    return dt == tradingcalendar.open_and_closes.T[ref]['market_close']

def hedge_ratio(y, x, add_const=True):
    if add_const:
        x = sm.add_constant(x)
        model = sm.OLS(y, x).fit()
        return model.params[1]
    model = sm.OLS(y, x).fit()
    return model.params.values

def compute_holdings_pct(y_shares, x_shares, y_price, x_price):
    y_dollars = y_shares * y_price
    x_dollars = x_shares * x_price
    notional_dollars =  abs(y_dollars) + abs(x_dollars)
    y_target_pct = y_dollars / notional_dollars
    x_target_pct = x_dollars / notional_dollars
    return (y_target_pct, x_target_pct)

def adf_p_value(input_ts):
    # returns p-value from Augmented Dickey-Fullet test for cointegration, with lag=1
    return ts.adfuller(input_ts, 1)[1]

def adf_critical_value_test(input_ts):
    # returns 1: if the t-stat of the ADF-test is less than the 1% critical value
    # returns 5: if the t-stat of the ADF-test is less than the 5% critical value (but greater than the 1% level)
    # returns 10: if the t-stat of the ADF-test is less than the 10% critical value (but greater than the 1% and 5% levels)
    # return 99: if the t-stat of the ADF-test is greater than the 10% critical value
    adf_test = ts.adfuller(input_ts, 1)
    t_stat = adf_test[0]
    critical_values = adf_test[4]
    if t_stat < critical_values['1%']:
        return int(1)
    if t_stat < critical_values['5%']:
        return int(5)
    if t_stat < critical_values['10%']:
        return int(10)
    return int(99)

def half_life(input_ts):
    # returns the [theoretical, based on OU-process equations] number of periods to expect
    # to have to wait for the spread to mean-revert half the distance to its mean
    price = pd.Series(input_ts)
    lagged_price = price.shift(1).fillna(method="bfill")
    delta = price - lagged_price
    beta = np.polyfit(lagged_price, delta, 1)[0]
    half_life = (-1*np.log(2)/beta)
    return half_life

def hurst(input_ts, lags_to_test=20):
    # interpretation of return value
    # hurst < 0.5 - input_ts is mean reverting
    # hurst = 0.5 - input_ts is effectively random/geometric brownian motion
    # hurst > 0.5 - input_ts is trending
    tau = []
    lagvec = []
    #  Step through the different lags
    for lag in range(2, lags_to_test):
        #  produce price difference with lag
        pp = np.subtract(input_ts[lag:], input_ts[:-lag])
        #  Write the different lags into a vector
        lagvec.append(lag)
        #  Calculate the variance of the differnce vector
        tau.append(np.sqrt(np.std(pp)))
    #  linear fit to double-log graph (gives power)
    m = np.polyfit(np.log10(lagvec), np.log10(tau), 1)
    # calculate hurst
    hurst = m[0]*2
    return hurst
```

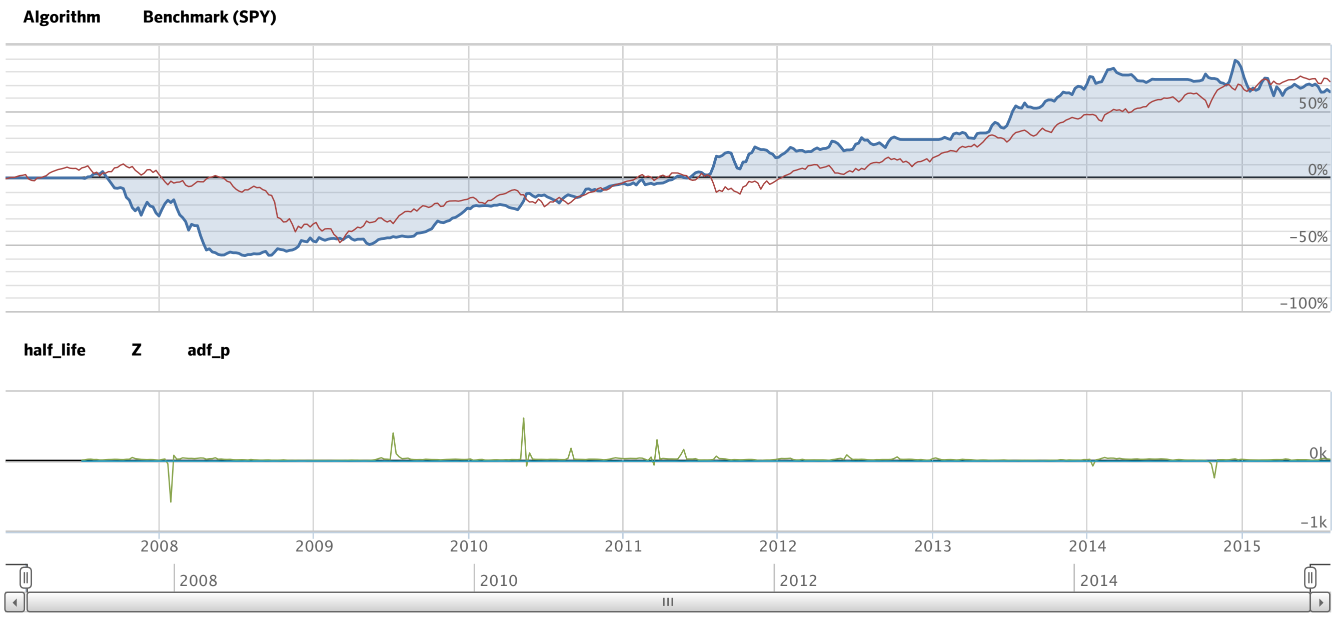
17 responses

Vladimir Yevtushenko Jul 30, 2015

Same algo just start 9 month earlier.

Performance Source Code

Backtest from **2007-01-02** to **2015-07-28** with **$100,000** initial capital



Jamie Lunn Sep 13, 2015

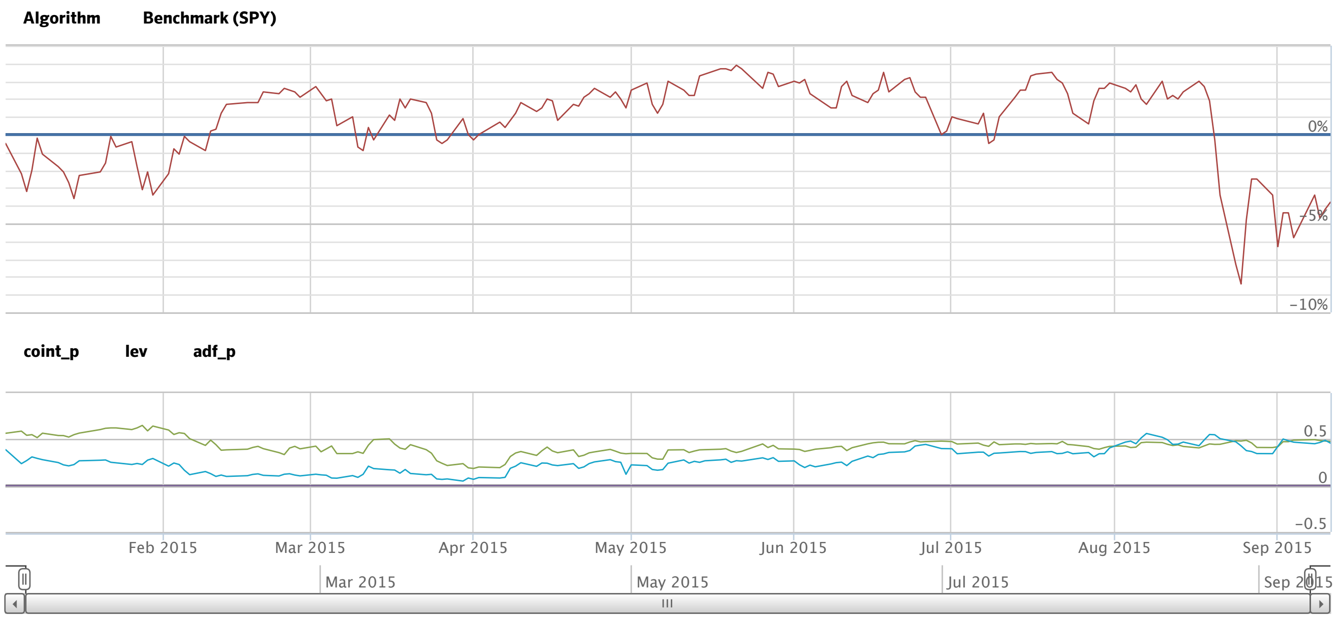
Hey Justin,

Thanks for the share. I've noticed that there is a coint function in statsmodels.tsa.stattools. Is there a significant difference between the coint function and the ADF test? Any sense in using both?

I've attached a backtest below that attempts to find the pvalue of both tests for each pair, every day. Disclaimer: what I often think is happening in python is actually not.

Performance | Source Code

Backtest from **2015-01-01** to **2015-09-11** with **$100,000** initial capital



Justin Lent Sep 25, 2015

@Jamie,
I haven't tried the coint function in stattools yet, though I imagine it's very similar. I just took a quick glimpse at the code, and it's effectively running a regression of the lagged version of the input timeseries versus the unlagged version which is quite similar to ADF. The difference may lie in how the critical values are computed.

The Engle-Granger test is also sometimes used to test for co-integration, but I haven't looked at that implementation yet.

DISCLAIMER

Adeel Imtiaz Sep 26, 2015

Thumbs up (y)

Johy Reaper Sep 26, 2015

Great Algo. Its amazing. Very Helpful

Adam Preston Sep 26, 2015

Hello Justin / All
Could you suggest how I can run this algo on multiple pairs, rather than just one pair?

Thanks!
Preston

James Christopher Sep 26, 2015

Try making a pairs trading class that keeps track of all the bookkeeping for each given pair. See David's generalized Kalman filters pair trading algo for a great example of class based pairs trading

Adeel Imtiaz Sep 27, 2015

Thanks for sharing info

Adam Preston Sep 27, 2015

Hi all,
I cloned Justin's algo, however when I run a backtest, the performance remains at 0% for the entirety of the backtest window.
I made no changes to the original source code.

Any ideas why this would be occurring?

Thanks!

Vladimir Yevtushenko Sep 27, 2015

Adam,

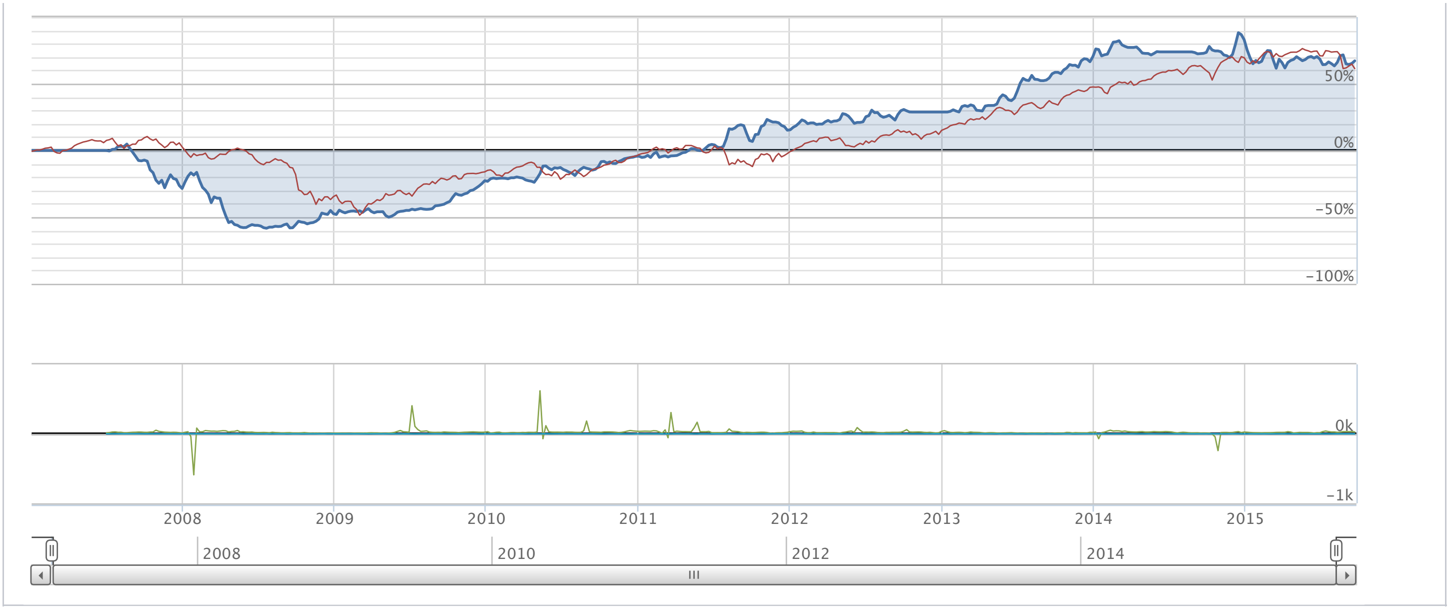
You probably run algo in daily mode and it only work in minute mode.
Here is my latest backtest of original Justin's Lent algo started just 9 month earlier.

Performance Source Code

Backtest from **2007-01-02** to **2015-09-25** with **$100,000** initial capital

Algorithm **Benchmark (SPY)**

half_life **Z** **adf_p**



Justin Lent Sep 27, 2015

All,
It's worth noting that when I post backtests, code, and research notebooks, the intent is to illustrate a methodology, and provide some code templates to spur the creative thought process of the community and save folks some time by providing cut-and-paste code fragments that can be integrated into their own code. By no means am I posting something that has been fully vetted, and immediately investable in it's exact form, by any stretch of the imagination. I often bias for simpler, rather than overly complex, examples as well, so as to benefit a broader spectrum of readers.

Vladimir,
I see you've recognized that the backtest I posted above seems to fail pretty badly over a different timeframe. We see this a lot with strategies we look at, many of which are overfit to just the 2 year period in the contests we run. We try to work with the algo owner and provide advice as to why it may have broken down over the different timeframes. Perhaps you can extend your analysis to provide me some advice as to improving this strategy? Maybe you have some recommendations as to how to incorporate a regime switching model which is very likely to help a strategy such as this given the time frame it seems to fail (the financial/commodity futures crises that occurred in late 2008). Perhaps a stochastic volatility regime switching model might help significantly. If you have experience in this area I'm sure the community would find it a solid addition to incorporate into strategies such as these to make them more robust. I know I would.

DISCLAIMER

Adam Preston Sep 27, 2015

Justin,
Why did you choose the pair USO and GLD? I guess a broader question is can you suggest a process for scanning through a basket of stocks and determining if there are tradable pairs? I'm assuming tests for cointegration would be one method such as ADF as you used. It would be nice if there could be an algo to run through a basket of stocks and auto determine which would make "good" pairs.

Just a thought.
Adam

Justin Lent Sep 27, 2015

Hey Adam,
I just chose USO/GLD in order to replicate this example that uses those same tickers, from this book: http://www.amazon.com/Quantitative-Trading-Build-Algorithmic-Business/dp/0470284889/

That book is a really good intro to stat arb pair trading (as well as his other books). All the code in the book is in Matlab, so my algo was an attempt to implement it in Python, in our backtester, and incorporate some of the other statistical techniques described throughout the book.

You are correct, that screening a bunch of potential pairs is a reasonable research idea, but you should be cognizant of simply datamining. You first want to determine a sensible economic basis for which the pairs of stocks should be tied (e.g. pairs of stocks in the same sector would be reasonable pairs of stocks to search across). Writing an algo in our backtester to accomplish this would be fairly straightforward: First you can use our Morningstar fundamentals database to grab all stocks in the Energy sector, perhaps even filtering down to stocks of companies of a certain band of marketcap (e.g. only mid-cap energy stocks), then in before_trading_starts(), you loop over each stock pair computing the ADF p-value (or other cointegration stat), keep all the stock pairs that meet your criteria, and then in handle_data() you just run the ones that meet the criteria through an algo similar to the one I shared to enter/exit the trades.

Myself or someone on our team here at Q can try to develop a template for this and share it.

As well you can look at this forum post that shows how to develop a single algo that trades a portfolio of multiple pairs:
It's the algo backtest in the first comment from David Edwards, here:
https://www.quantopian.com/posts/quantopian-lecture-series-this-time-youre-more-wrong?c=1

DISCLAIMER

Adam Preston Apr 6, 2016

Justin,
I noticed in the blog section you have a notebook on using a Bayesian optimizer...would you know how i can pull it into Q? its currently on github..thanks!

Justin Lent Apr 7, 2016

@Adam, At present it's not possible to use the Bayesian optimizer from the blog post in the Q environment. It was more of a proof of concept implementation idea. As you mentioned, the code I used for the blog post is on github and you can sign up for a trial with SigOpt to get a username/API key to work with it in your own python/zipline environment locally. Offering some of these alternative methods of optimization as a service is an interesting concept which we will have to think about as we develop our Q platform in the future. Thanks for the feedback!

DISCLAIMER

Adam Preston Apr 7, 2016

Thanks Justin! Would be neat to be able to do that type of optimization and / or a particle swarm technique in Q. :)

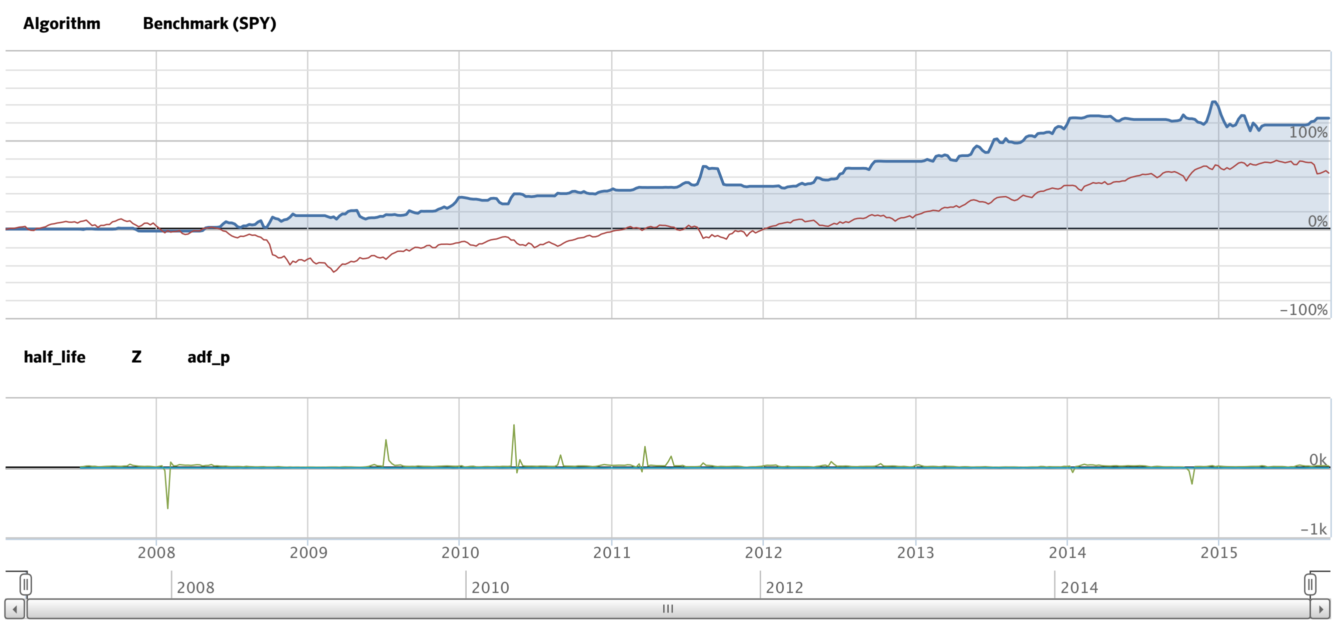
Shawn Emhe II Nov 27, 2016

Hello Justin,
I believe I found a gap in the trading logic. In the statistic filtering section (lines ~155-176) the algorithm immediately exits if a test fails. That prevents new trades from being opened but does nothing to handle existing trades. Open trades stay open until all of the statistical tests pass again and the algorithm reaches its standard exit logic.
By design we should also have a high likelihood of being in a trade when this happens so the impact could be quite high. The problem in detecting this is that if the relationship re-establishes quickly the performance won't suffer. But if we include a time period in which the relationship doesn't return quickly, as Vladimir did, the results are noticeable.
I added a few lines to close any positions that are open when the statistical tests break down. There are probably better ways of handling the exit logic, but this simple change shows the benefit of having it there. The algorithm doesn't do as well during the original test period but the performance improves over the extended period.

(I also made minor change on lines 20 and 21 to use sid() function to set x and y assets rather than symbol(). The rest of the algorithm is unchanged.)

Performance Source Code

Backtest from **2007-01-02** to **2015-09-25** with **$100,000** initial capital



Please sign in or join Quantopian to post a reply.

About Quantopian

Careers

Community

Events

QuantCon

Help

Academia

Lectures

Workshops

Investor Relations

Status

Facebook

Twitter

LinkedIn

Blog

Terms of Use

Privacy Policy