Pairs Trading with Optimize

這邊第一部份，匯入套件庫

```
import numpy as np
import statsmodels.api as sm
import pandas as pd

import quantopian.optimize as opt
import quantopian.algorithm as algo
```

首先是initialize的部份

```
def initialize(context):
    # Quantopian backtester specific variables
    set_slippage(slippage.FixedSlippage(spread=0))
    set_commission(commission.PerTrade(cost=1))
    set_symbol_lookup_date('2014-01-01')

    context.stock_pairs = [(symbol('ABGB'), symbol('FSLR')),
                           (symbol('CSUN'), symbol('ASTI'))]

    context.stocks = symbols('ABGB', 'FSLR', 'CSUN', 'ASTI')

    context.num_pairs = len(context.stock_pairs)
    # strategy specific variables
    context.lookback = 20 # used for regression
    context.z_window = 20 # used for zscore calculation, must be <= lookback

    context.target_weights = pd.Series(index=context.stocks, data=0.25)

    context.spread = np.ndarray((context.num_pairs, 0))
    context.inLong = [False] * context.num_pairs
    context.inShort = [False] * context.num_pairs

    # Only do work 30 minutes before close
    schedule_function(func=check_pair_status, date_rule=date_rules.every_day(),
time_rule=time_rules.market_close(minutes=30))

# Will be called on every trade event for the securities you specify.
```

我們在一開始先設定一些回測環境參數，首先是設定slippage model的部份，這邊slippage是要估計我們的訂單對於市價的影響的設定，我們這邊使用的slippage model為Fixed Slippage，這個設定代表我們的訂單不影響我們的交易價格，這邊的spread參數代表，當我們丟出買單後，一半的spread將會被加到價格中，而賣單則是相反。

再來我們設定手續費的部份，這邊應該是設定每次交易要付一元的手續費，在這邊我們也設定look up date為2014-01-01，這邊的設定是代表，我們用這天當我們股票代號的參考日，因為這些代號可能會隨時間指稱不同的證券。

```
# Quantopian backtester specific variables
    set_slippage(slippage.FixedSlippage(spread=0))
    set_commission(commission.PerTrade(cost=1))
    set_symbol_lookup_date('2014-01-01')
```

接下來我們分別設定一些變數，我們將我們股票代號以及長度存到context變數之下

```
 context.stock_pairs = [(symbol('ABGB'), symbol('FSLR')),
                        (symbol('CSUN'), symbol('ASTI'))]

 context.stocks = symbols('ABGB', 'FSLR', 'CSUN', 'ASTI')

 context.num_pairs = len(context.stock_pairs)
```

接下來跟basic pair trading的例子一樣，我們設定相關的變數。

```
 # strategy specific variables
    context.lookback = 20 # used for regression
    context.z_window = 20 # used for zscore calculation, must be <= lookback

    context.target_weights = pd.Series(index=context.stocks, data=0.25)

    context.spread = np.ndarray((context.num_pairs, 0))
    context.inLong = [False] * context.num_pairs
    context.inShort = [False] * context.num_pairs
```

最後設定行程函數

```
 # Only do work 30 minutes before close
    schedule_function(func=check_pair_status, date_rule=date_rules.every_day(),
time_rule=time_rules.market_close(minutes=30))
```

在來看一下我們的下單邏輯

```
def allocate(context, data):
    # Set objective to match target weights as closely as possible, given constraints
    objective = opt.TargetWeights(context.target_weights)

    # Define constraints
    constraints = []
    constraints.append(opt.MaxGrossExposure(1.0))


    algo.order_optimal_portfolio(
        objective=objective,
        constraints=constraints,
    )
```

基本上與basic pair trading的例子一致

我們接下看程式的主邏輯

```
    prices = data.history(context.stocks, 'price', 35, '1d').iloc[-context.lookback::]

    new_spreads = np.ndarray((context.num_pairs, 1))
```

我們將四個目標股票的前二十天股價抓出，然後造出一個2×1的零矩陣。接下來我們對這兩對pair進行操作

```
for i in range(context.num_pairs):

        (stock_y, stock_x) = context.stock_pairs[i]

        Y = prices[stock_y]
        X = prices[stock_x]

        # Comment explaining try block
        try:
            hedge = hedge_ratio(Y, X, add_const=True)
        except ValueError as e:
            log.debug(e)
            return
```

對每對資產，我們將兩個價格抓出，分別存為Y與X，然後我們計算所謂的hedge ratio，這邊計算的邏輯如下

```
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
```

這邊的hedge_ratio其實就是把Y下去跟X跑回歸，然後跑回來的估計係數。

然後我們用以下的邏輯得到目前的資產權重

```
def get_current_portfolio_weights(context, data):
    positions = context.portfolio.positions
    positions_index = pd.Index(positions)
    share_counts = pd.Series(
        index=positions_index,
        data=[positions[asset].amount for asset in positions]
    )

    current_prices = data.current(positions_index, 'price')
    current_weights = share_counts * current_prices / context.portfolio.portfolio_value
    return current_weights.reindex(positions_index.union(context.stocks), fill_value=0.0)
```

我們取得目前的資產權重並計算出回歸計算出的價差

```
        context.target_weights = get_current_portfolio_weights(context, data)

        new_spreads[i, :] = Y[-1] - hedge * X[-1]
```

接下來我們計算z score

```
if context.spread.shape[1] > context.z_window:
            # Keep only the z-score lookback period
            spreads = context.spread[i, -context.z_window:]

            zscore = (spreads[-1] - spreads.mean()) / spreads.std()
```

接下來我們討論出場邏輯，如果目前再做空這組價差，則當z score又掉回0以下後，我們出場，將部位權重都換回0，並且將context.inShort換回False，另外一個方向也是類似。

```
            if context.inShort[i] and zscore < 0.0:
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0

                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            if context.inLong[i] and zscore > 0.0:
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0


                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return
```

接下來討論進場邏輯，首先，當zscore小於-1時，我們y_target_share為1(買y)做空x -hege的量，並且把context.inLong改成True，這時候我們寫一個函數來計算y跟x的權重

```
def computeHoldingsPct(yShares, xShares, yPrice, xPrice):
    yDol = yShares * yPrice
    xDol = xShares * xPrice
    notionalDol =  abs(yDol) + abs(xDol)
    y_target_pct = yDol / notionalDol
    x_target_pct = xDol / notionalDol
    return (y_target_pct, x_target_pct)
```

這邊我們先計算出y與x的幣值，然後根據y幣值/總幣值來計算y與x比例。因為我們有兩對資產，所以我們最終的權重還要除以context.num_pairs。

```
            if zscore < -1.0 and (not context.inLong[i]):
                # Only trade if NOT already in a trade
                y_target_shares = 1
                X_target_shares = -hedge
                context.inLong[i] = True
                context.inShort[i] = False

                (y_target_pct, x_target_pct) =
computeHoldingsPct(y_target_shares,X_target_shares, Y[-1], X[-1])

                context.target_weights[stock_y] = y_target_pct * (1.0/context.num_pairs)
                context.target_weights[stock_x] = x_target_pct * (1.0/context.num_pairs)

                record(Y_pct=y_target_pct, X_pct=x_target_pct)
                allocate(context, data)
                return


            if zscore > 1.0 and (not context.inShort[i]):
                # Only trade if NOT already in a trade
                y_target_shares = -1
                X_target_shares = hedge
                context.inShort[i] = True
                context.inLong[i] = False

                (y_target_pct, x_target_pct) = computeHoldingsPct( y_target_shares,
X_target_shares, Y[-1], X[-1] )

                context.target_weights[stock_y] = y_target_pct * (1.0/context.num_pairs)
                context.target_weights[stock_x] = x_target_pct * (1.0/context.num_pairs)

                record(Y_pct=y_target_pct, X_pct=x_target_pct)
                allocate(context, data)
                return

    context.spread = np.hstack([context.spread, new_spreads])
```

這邊為完整的程式碼

```
import numpy as np
import statsmodels.api as sm
import pandas as pd

import quantopian.optimize as opt
import quantopian.algorithm as algo

def initialize(context):
    # Quantopian backtester specific variables
```

```
    set_slippage(slippage.FixedSlippage(spread=0))
    set_commission(commission.PerTrade(cost=1))
    set_symbol_lookup_date('2014-01-01')

    context.stock_pairs = [(symbol('ABGB'), symbol('FSLR')),
                           (symbol('CSUN'), symbol('ASTI'))]

    context.stocks = symbols('ABGB', 'FSLR', 'CSUN', 'ASTI')

    context.num_pairs = len(context.stock_pairs)
    # strategy specific variables
    context.lookback = 20 # used for regression
    context.z_window = 20 # used for zscore calculation, must be <= lookback

    context.target_weights = pd.Series(index=context.stocks, data=0.25)

    context.spread = np.ndarray((context.num_pairs, 0))
    context.inLong = [False] * context.num_pairs
    context.inShort = [False] * context.num_pairs

    # Only do work 30 minutes before close
    schedule_function(func=check_pair_status, date_rule=date_rules.every_day(),
time_rule=time_rules.market_close(minutes=30))

# Will be called on every trade event for the securities you specify.
def handle_data(context, data):
    # Our work is now scheduled in check_pair_status
    pass

def check_pair_status(context, data):

    prices = data.history(context.stocks, 'price', 35, '1d').iloc[-context.lookback::]

    new_spreads = np.ndarray((context.num_pairs, 1))

    for i in range(context.num_pairs):

        (stock_y, stock_x) = context.stock_pairs[i]

        Y = prices[stock_y]
        X = prices[stock_x]

        # Comment explaining try block
        try:
            hedge = hedge_ratio(Y, X, add_const=True)
        except ValueError as e:
            log.debug(e)
            return

        context.target_weights = get_current_portfolio_weights(context, data)

        new_spreads[i, :] = Y[-1] - hedge * X[-1]
```

```
        if context.spread.shape[1] > context.z_window:
            # Keep only the z-score lookback period
            spreads = context.spread[i, -context.z_window:]

            zscore = (spreads[-1] - spreads.mean()) / spreads.std()

            if context.inShort[i] and zscore < 0.0:
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0

                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            if context.inLong[i] and zscore > 0.0:
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0


                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            if zscore < -1.0 and (not context.inLong[i]):
                # Only trade if NOT already in a trade
                y_target_shares = 1
                X_target_shares = -hedge
                context.inLong[i] = True
                context.inShort[i] = False

                (y_target_pct, x_target_pct) =
computeHoldingsPct(y_target_shares,X_target_shares, Y[-1], X[-1])

                context.target_weights[stock_y] = y_target_pct * (1.0/context.num_pairs)
                context.target_weights[stock_x] = x_target_pct * (1.0/context.num_pairs)

                record(Y_pct=y_target_pct, X_pct=x_target_pct)
                allocate(context, data)
                return


            if zscore > 1.0 and (not context.inShort[i]):
                # Only trade if NOT already in a trade
                y_target_shares = -1
                X_target_shares = hedge
                context.inShort[i] = True
                context.inLong[i] = False
```

```
                (y_target_pct, x_target_pct) = computeHoldingsPct( y_target_shares,
X_target_shares, Y[-1], X[-1] )

                context.target_weights[stock_y] = y_target_pct * (1.0/context.num_pairs)
                context.target_weights[stock_x] = x_target_pct * (1.0/context.num_pairs)

                record(Y_pct=y_target_pct, X_pct=x_target_pct)
                allocate(context, data)
                return

    context.spread = np.hstack([context.spread, new_spreads])

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 computeHoldingsPct(yShares, xShares, yPrice, xPrice):
    yDol = yShares * yPrice
    xDol = xShares * xPrice
    notionalDol =  abs(yDol) + abs(xDol)
    y_target_pct = yDol / notionalDol
    x_target_pct = xDol / notionalDol
    return (y_target_pct, x_target_pct)

def get_current_portfolio_weights(context, data):
    positions = context.portfolio.positions
    positions_index = pd.Index(positions)
    share_counts = pd.Series(
        index=positions_index,
        data=[positions[asset].amount for asset in positions]
    )

    current_prices = data.current(positions_index, 'price')
    current_weights = share_counts * current_prices / context.portfolio.portfolio_value
    return current_weights.reindex(positions_index.union(context.stocks), fill_value=0.0)


def allocate(context, data):
    # Set objective to match target weights as closely as possible, given constraints
    objective = opt.TargetWeights(context.target_weights)

    # Define constraints
    constraints = []
    constraints.append(opt.MaxGrossExposure(1.0))


    algo.order_optimal_portfolio(
        objective=objective,
```

```
        constraints=constraints,
    )
```