# Homework 2

**Collaboration disclosure**: This assignment was discussed with Viswanadha Akella but written independently.

**Overall Summary**: In this report we investigate 8 different trading strategies. The first 5 of these are technical rules based ideas while the last three are pairs trading based strategies. We first optimize the strategies in-sample and then evaluate them on the basis of out-of-sample performance and robustness.

The main results is pretty clear - **simpler models yield robust results**. We see that in-sample sophisticated strategies like the ones with more parameters and estimation factors give better result. On the other hand, we see that out of sample all the sophisticated models backfired. This may be due to **overfitting** in-sample and use of lot of parameters. **Rule 2 stands out as the most simplest rule** (minimum code as an evidence) and is robust out of sample as well. Though other rules were good in-sample they have not performed that well out of sample. Hence **we will recommend rule 2 - Momentum with cross-over as the strategy to trade** - for the given cross-section and for the market conditions similar to the case we have investigated. Other models should be investigated further to see if they can be generalized further to produce better out of sample results. Larger cross-sections can be investigates as well. Never the less, the exercise does show how different rules are implemented and the ideas behind them.

**Some assumptions**: This assignment has been done by writing code primarily in Python. Some analysis was done in R as a check. All the code used is included, but if something has been missed out, it can be made available on request.

To investigate strategies we first give the basic setup of the backtesting code. The following code is used to run the backtest once the corresponding **calc_signal** is written. We use this code for the rest of the assignment without mentioning it again. This can be run in loops with various values of parameters to get various backtests results, which can be aggregated later. The primary data object in Python is Pandas DataFrame, designed especially for time series operations. Notice that under **calc_portfolio**, signal is shifted by 1 day to account for the fact that return on signal based on today's price can only be produced the next day. The trades and the trading cost is realized on the same day, when the trade happens. We present two different allocation schemes. The first, **equal_initial**, assume equal allocation to start with for the three assets. This is used for the first two problems as instructed. The second scheme, **pairs**, is used for problem 3 and 4 where we maintain pairs ratios. Also we do all calculations for an initial portfolio value of $1 (except for problem 3 where we put $1 each in the three pairs, resulting in an initial investment of $3). It is important to note that there are other schemes as well which can be used, we just chose one of the simpler schemes to implement.

Further, we have used the following exit schemes in the problems. When the signal is short it exits if it crosses the mean (average value) from above. If the signal is long the exit is triggered when the mean is breached from below. These means depend on the problem, can be constant or moving dynamically.

Also notice that at some places unlike typical day by day calculations we utilize vectorized calculations which speed up the backtesting considerably. But they are essentially similar to step by step calculation. We also do problem 1 to 4 on data till end of 2013. This is the training set. We use the rest of the data for evaluating the strategy later on an out-of-sample performance basis (see question 5).

```
__author__ = 'manish'
from pandas import *
import datetime as dt
import numpy as np
from matplotlib.pyplot import *

def dtparse(st):
    # parse the dates to datetime object for indexing
    a = str(st)
    return dt.datetime.strptime(a if len(a)==8 else '0'+a, '%m%d%Y')

def get_data(start_date=None, end_date='2013/12/31'):
    # get the data till to_date
    main_dir = '/home/manish/Desktop/S242_AlgoTrading/HW2/'
    df = read_csv(main_dir+'BankData.csv')
    df.index = df['Date'].apply(dtparse)
    df = df[[i for i in df.columns if 'Date' not in i]]
    start_date = df.index[0] if start_date is None else
     dt.datetime.strptime(start_date,'%Y/%m/%d')
    end_date = df.index[-1] if end_date is None else
     dt.datetime.strptime(end_date,'%Y/%m/%d')
    idx = df.index[(df.index>=start_date) & (df.index<=end_date)]
    return df.ix[idx]

def calc_portfolio(sig, ret, prc, tc, alloc='equal_initial'):
    LAG = 1
    if alloc is 'equal_initial':
        # EQUAL INITIAL INVESTMENT WITH SEPARATE REINVESTMENT
        sig = sig.apply(np.sign)
        trd = sig.diff()
        asset_ret_simple = sig.shift(LAG)*ret - tc*trd.shift(LAG-1).apply(np.abs)
        asset_daily_val = np.exp(expanding_sum(np.log(1+asset_ret_simple)))
```

```
        asset_daily_profit = asset_daily_val.pct_change()
        port_daily_profit = (asset_daily_profit*asset_daily_val.shift(LAG)).sum(axis=1)/\
                            asset_daily_val.shift(LAG).sum(axis=1)
    elif alloc is 'pairs':
        # NO ADJUSTMENT FOR PAIRS TRADE TO MAINTAIN RATIO
        trd = sig.diff()
        asset_ret_simple = sig.shift(LAG)*ret - tc*trd.shift(LAG-1).apply(np.abs)
        asset_daily_profit = (np.exp(expanding_sum(np.log(1+asset_ret_simple)))-1).diff(1)
        port_daily_profit = asset_daily_profit.mean(axis=1)
    return DataFrame(port_daily_profit, columns=['portfolio']), asset_daily_profit


def get_beta(df, mkt, rfr):
    # calculate beta wrt mkt
    val = DataFrame(columns=['alpha','beta','std'], index=df.columns)
    for item in df.columns:
        model = ols(x=mkt-rfr/250, y=df[item]-rfr/250)
        val['alpha'][item],val['beta'][item],val['std'][item] = \
            model.beta['intercept'], model.beta['x'], model.resid.std()
    return val

def get_dstd(df):
    # calculate downside std
    dstd = df.std()
    for item in dstd.index:
        dstd[item] = df[item][df[item]<0].std()
    return dstd

def get_maxdd(df):
    # calcualte maximum drawdown
    return (df.cumsum()-expanding_max(df.cumsum())).min()

def calc_performance(pnl, mkt, rfr):
    # cacluate ratios and stats - all annualized
    apnl = pnl-rfr/250 # abnormal returns over risk free rate
    om = get_beta(pnl, mkt, rfr)
    alpha, beta, epstd = om['alpha'], om['beta'], om['std']
    dstd = get_dstd(pnl)
    maxdd = np.abs(get_maxdd(pnl))
    nyrs = (pnl.last_valid_index()-pnl.first_valid_index()).days/365.
    aret = (np.prod(pnl+1)-1)/nyrs
    return {
```

```
        'sharpe': np.sqrt(250)*apnl.mean()/pnl.std(),
        'Treynor': np.sqrt(250)*apnl.mean()/beta,
        'Sortino': np.sqrt(250)*apnl.mean()/dstd,
        'calmar': aret/maxdd,
        'IR': np.sqrt(250)*alpha/epstd,
        'PNL': pnl.cumsum().ix[-1,'portfolio']
    }

def run_bt(param, calc_signal, tc=0, rfr=0.03, alloc='equal_initial',showf=False):
    # backtest run: assume initial captial of $1
    prc = get_data()
    ret = prc.pct_change().fillna(method='ffill')
    sig = calc_signal(prc, param)
    # get returns for signal based portfolio
    pnl,ar = calc_portfolio(sig, ret, prc, tc)
    # get returns for long only portfolio
    long,al = calc_portfolio(sig*0.+1., ret, prc, tc, alloc)
    if showf:
        plot_equity(ar, pnl, long)
    stats = DataFrame(calc_performance(pnl, ret['SPY'], rfr))
    return {'prc': prc, 'ret': ret, 'sig': sig,
            'pnl': pnl, 'ar': ar, 'stats': stats}

def plot_equity(ar, pnl, long):
    # plot nice equity curve along with component contribution
    fig = figure()
    ax1 = fig.add_subplot(311)
    (1+DataFrame({'port': pnl['portfolio'], 'long':long['portfolio']}).\
        cumsum()).plot(ax=ax1, title='portfolio and Long returns', grid=True)
    ax2 = fig.add_subplot(312)
    dd = pnl.cumsum()-expanding_max(pnl.cumsum())
    dd.plot(ax=ax2, title='drawdown', grid=True)
    ax3 = fig.add_subplot(313)
    (1+ar.cumsum()).plot(ax=ax3, title='asset returns', grid=True)

def heat_map(df, title=None):
    # plots the heatmap with nice formatting
    fig, ax = subplots()
    pt = ax.pcolor(df.fillna(0), cmap=cm.RdYlGn, edgecolors='w')
    ax.axis([0, len(df.columns), 0, len(df.index)])
    ax.set_yticks(np.arange(0,len(df.index))+0.5)
    ax.set_xticks(np.arange(0,len(df.columns))+0.5)
```

```
    ax.set_xticklabels(df.columns,minor=False,fontsize=15)
    ax.set_yticklabels(df.index,minor=False,fontsize=15)
    xlabel(df.columns.name)
    ylabel(df.index.name)
    fig.colorbar(pt)
    suptitle(title)
    show()
```

## Problem 1

This problem deals with analysis of 4 different technical rules. The initial portfolio consists of equally weighted shares of WFC (Wells-Fargo), JPM (JP Morgan Chase), and SPY (S&P 500 ETF Trust). Equal amount is invested in each share at the entry point using the scheme **equal_initial**.

## Problem 1a Rule 1

We implement the signal for Moving average trading Rule 1 here as follows:

```
def calc_signal(prc, param, ex=0):
    # we use info on day t-1 to put signal on day t
    # 1a Moving average tarading rules - Rule 1
    ma = rolling_mean(prc, param['L'])
    sig = 0*ma.fillna(0).copy()
    for com in sig.columns:
        for i in range(1,len(sig)):
            tp,t = sig.index[i-1], sig.index[i]
            sig[com][t] = sig[com][tp] # copy over previous state
            if sig[com][tp] == 0:
                # evaluate entry
                if prc[com][tp]>param['w']*ma[com][tp]: # open short
                    sig[com][t] = -1
                elif prc[com][tp]<ma[com][tp]/param['w']: # open long
                    sig[com][t] = 1
            else:
                # evaluate exit
                if sig[com][tp] == -1:
                    if prc[com][tp]<ma[com][tp]: # close short
                        sig[com][t] = 0
                if sig[com][t] == 1:
                    if prc[com][tp]>ma[com][tp]: # close long
                        sig[com][t] = 0
    return sig
```

**Evaluation of Rule 1**: The first thing to note is that Rule 1 is a contrarian strategy. This is only valid when $\omega \geq 1$. When $\omega < 1$ this signal does not make sense and hence we do not explore that region. Under these constraints whenever the price is some factor above the moving average we expect it to come back and go short, and whenever the price is below some factor of moving average we go long. Our entry is dictated by the values of $\omega$ and our exit happens when the price crosses back the moving average. An illustration is shown in fig.1. In these figures the PnL denotes the cumulative profit when starting with \$1 initially.

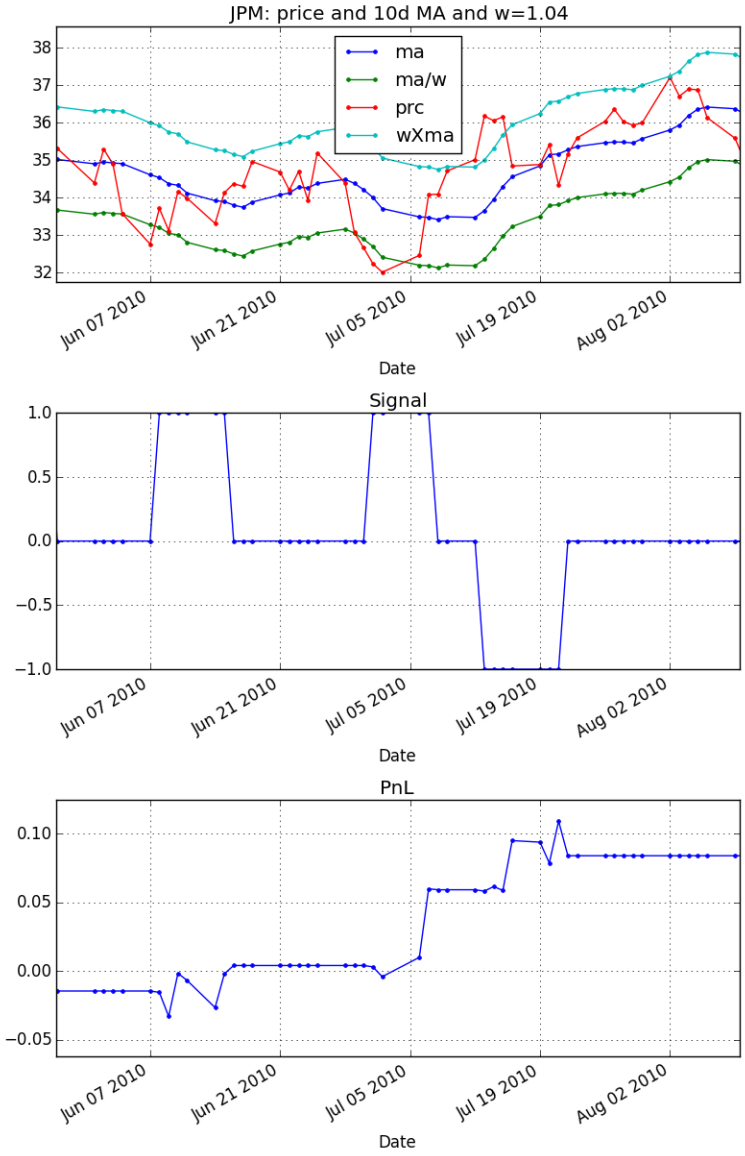


Figure 1: Entry exit rules illustrated for Q1(a) Rule 1 backtest with trading cost 10 bps. $L$ =10 and $\omega$ =1.04 are used.

```
# plot signal and trade entry exit
com = 'JPM'
fig = figure()
ax1 = fig.add_subplot(311)
prc = bt_tc['prc'][com]
ma = rolling_mean(prc,10)
```

```
DataFrame({'prc':prc,
           'ma':ma,
           'wXma':ma*1.04,
           'ma/w':ma/1.04}).\
    plot(ax=ax1,style='.-',grid=True, title=com+': price and 10d MA and w=1.04')
ax2 = fig.add_subplot(312,sharex=ax1)
bt_tc['sig'][com].plot(ax=ax2,style='.-', grid=True, title='Signal')
ax3 = fig.add_subplot(313,sharex=ax1)
bt_tc['ar'][com].cumsum().plot(ax=ax3, style='.-',grid=True, title='PnL')
```

**Qualitative performance** - We start with a moving window of $L = 10$ days and the scaling parameter $\omega = 1.04$. The top plot in fig.2 shows the equity curve (net of 10bps cost) in green along with the long only cumulative returns in blue. The strategy does not outperforms the long only portfolio, but only marginally. The second plot shows the drawdown curve. There is one big drawdowns of almost 14%, with very long durations. Finally, we also show the performance of the three assets which comprised the portfolio with equal weight. WFC contributed to the positive performance while JPM and SPY contributed marginally.

**Quantitative performance** - To quantify the performance let's look at Sharpe, Treynor, Sortino, Calmar and Information ratios (all annualized). We also look at the PNL with and without transaction cost of 10 basis points. Risk free rate is 3%.

| | 0bps | 10bps |
|---|---|---|
| IR | 0.1777988 | 0.06748703 |
| PNL | 0.3179399 | 0.2583195 |
| Sortino | 0.3515821 | 0.2606025 |
| Treynor | 0.0148362 | 0.0105209 |
| calmar | 0.5147889 | 0.3891256 |
| sharpe | 0.3663044 | 0.2603163 |

Table 1: Performance numbers (annualized) for Rule 1 for 0 and 10 bps trading cost.

Table 1 shows the various ratios. We see that applying a trading cost of 10 bps drags down each and every ratio as expected. The net Sharpe ratio is meagre 0.26, while IR is also low at 0.07. Other ratios are low as well. Dollar 1 invested in the strategy had yielded a profit of \$0.26 in the time period of the anlaysis.

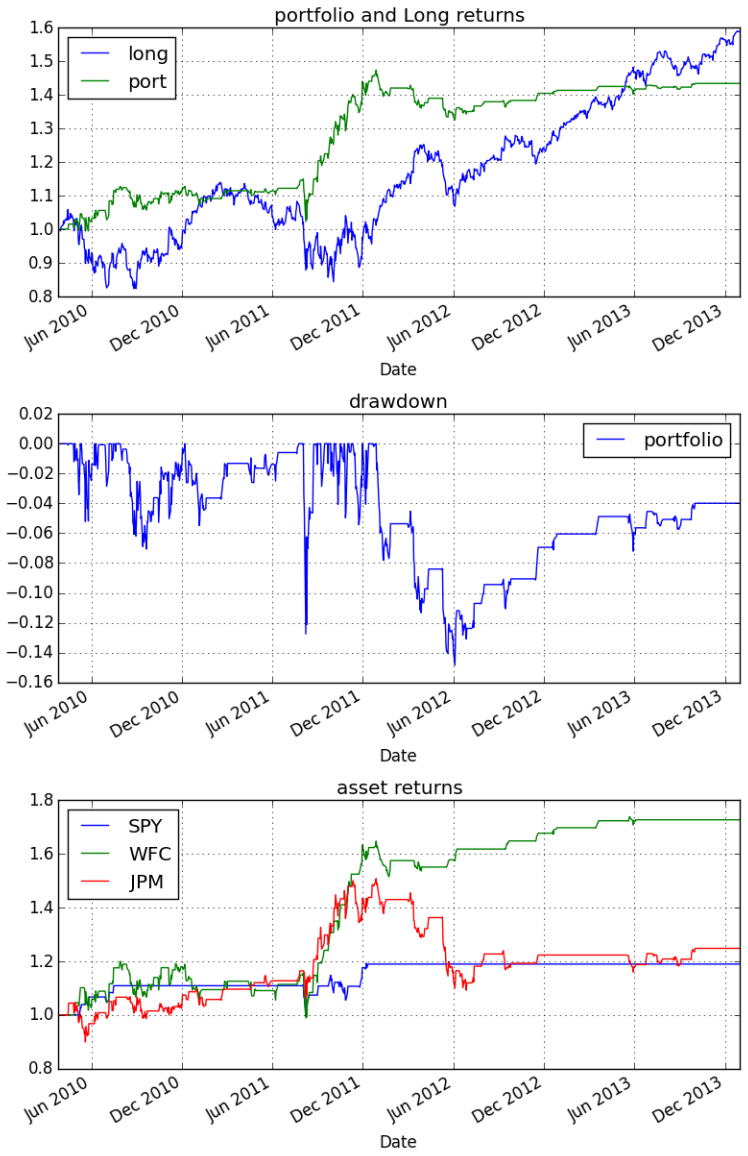


Figure 2: Equity curve analysis for Q1(a) Rule 1 backtest with trading cost 10 bps. $L$ =10 and $\omega$ =1.04 are used.

**Optimal in-sample values** - To get the optimal value of $L$ and $\omega$ we run a grid search and produce the fig.3 using the following code:

```
# optimization
hm = {}
for L in np.arange(1,21,1):
    sr = {}
```

```
    for w in np.arange(1.0,1.10,0.01):
        print L,w
        sr[w] = run_bt({'L':L, 'w':w}, calc_signal, tc=0.001,
        alloc='equal_initial')['stats']['sharpe']['portfolio']
    hm[L] = Series(sr)
hm = DataFrame(hm).replace(np.inf,np.nan).replace(-np.inf,np.nan).fillna(0)
hm.columns.name='L'
hm.index.name='w'
heat_map(hm, title='heatmap of performance sharpe; Q1 MA Rule 1')
```

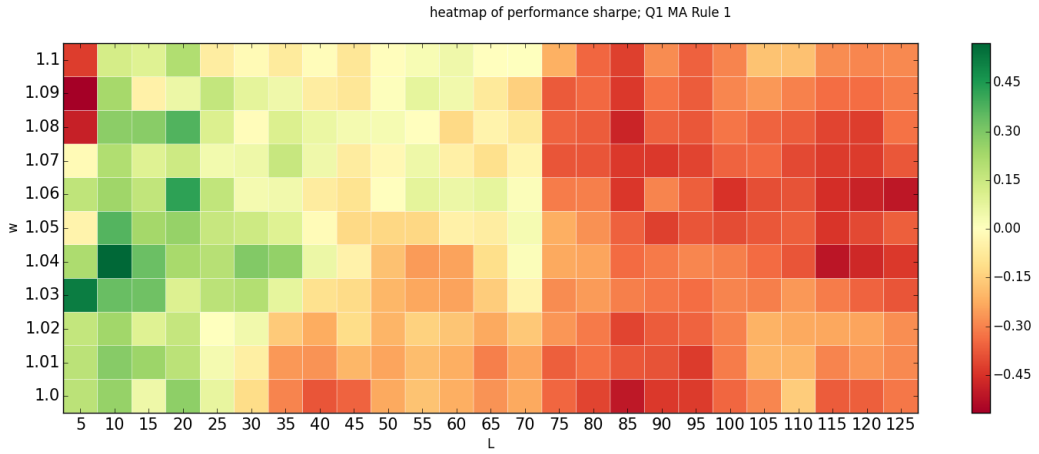


Figure 3: Heatmap for Q1(a) Rule 1 backtest with trading cost 10 bps.

The heat map plots Sharpe Ratio numbers, i.e. **we choose Sharpe as the criteria to choose the optimal value**. Other ratios could also be fairly used as a selection criteria. We see that the optimal heat map of $L$ vs $\omega$ is not very stable. A optimal and stable region looks like **(10,1.04)**. We choose it as the optimal signal for later comparison. The long term moving averages clearly don't work.

## Problem 1a Rule 2

We implement the signal for Momentum trading Rule 2 here as follows:

```
# overwriting the default signal
def calc_signal(prc, param):
    # 1a Moving average tarading rules - Rule 2
    ma_s = rolling_mean(prc, param['s'])
    ma_l = rolling_mean(prc, param['l'])
    sig = (ma_l-ma_s).apply(np.sign).shift(1)
    # shifted by a day so that t-1 info is used on day t
    return sig

# signal run
bt_0 = run_bt({'s':4, 'l':10}, calc_signal, tc=0.000,
showf=False, alloc='equal_initial')
bt_tc = run_bt({'s':4, 'l':10}, calc_signal, tc=0.001,
showf=True, alloc='equal_initial')
df = DataFrame({'0bps': bt_0['stats'].T['portfolio'],
'10bps': bt_tc['stats'].T['portfolio']})
print df.to_latex()
```

**Evaluation of Rule 2**: This rule is a simple crossing of moving averages rule, where the signal is either 1 or -1 depending on if the short MA is smaller than Long MA or vice versa. The Exits are hence automatically defined in this strategy. The stand out feature of this strategy is its simplicity, being parsimonious which is reflected in the 4 line code we needed to code up the signal.

**Qualitative performance**: We try the Short window of 4 days and long window of 10 days. The plot in fig.5 shows the equity curve along with drawdown curve and the performance of the three assets. The strategy performance in the tested period is similar to long only portfolio, but looks uncorrelated. There are massive drawdowns of up to -20%. All three assets contribute to the positive performance.

| | 0bps | 10bps |
|---|---|---|
| IR | 0.6734056 | 0.4106331 |
| PNL | 0.7647831 | 0.5564986 |
| Sortino | 1.127134 | 0.7728579 |
| Treynor | 0.04124897 | 0.02815835 |
| calmar | 1.372893 | 0.7666981 |
| sharpe | 0.8078717 | 0.5508252 |

Table 2: Performance numbers (annualized) for Rule 2 for 0 and 10 bps trading cost.

**Quantitative performance**: Table 2 shows the performance with and without trading cost. This is an improvement over the previous strategy with an a Sharpe of 0.55 and and IR of 0.41. $1 investment in this strategy earned a profit of $0.56 in the backtest period. An illustration of the trading rule for the moving average process is shown in fig.4.

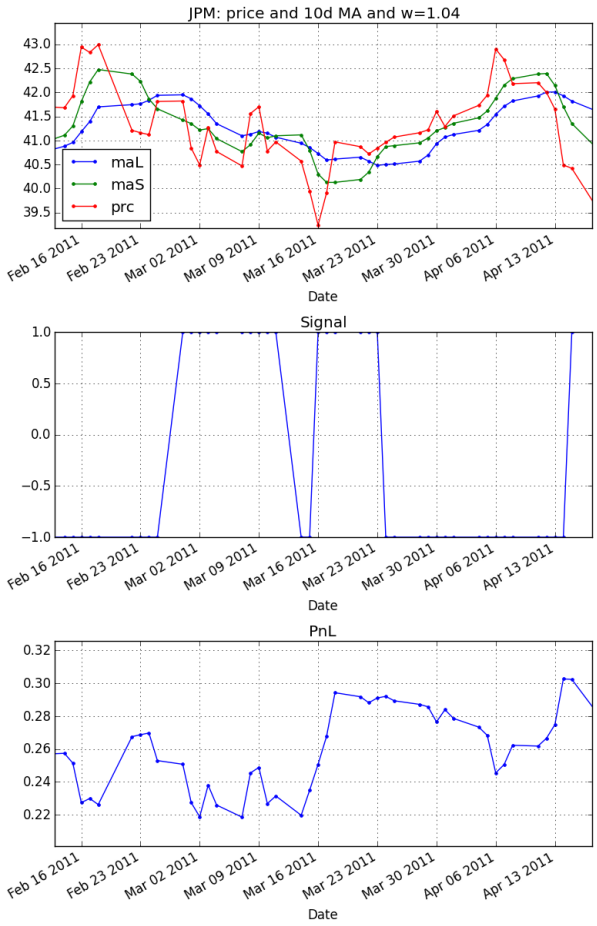


Figure 4: Entry exit rules illustrated for Q1(a) Rule 3 backtest with trading cost 10 bps. $s$ =5 and $l$ =10 are used.

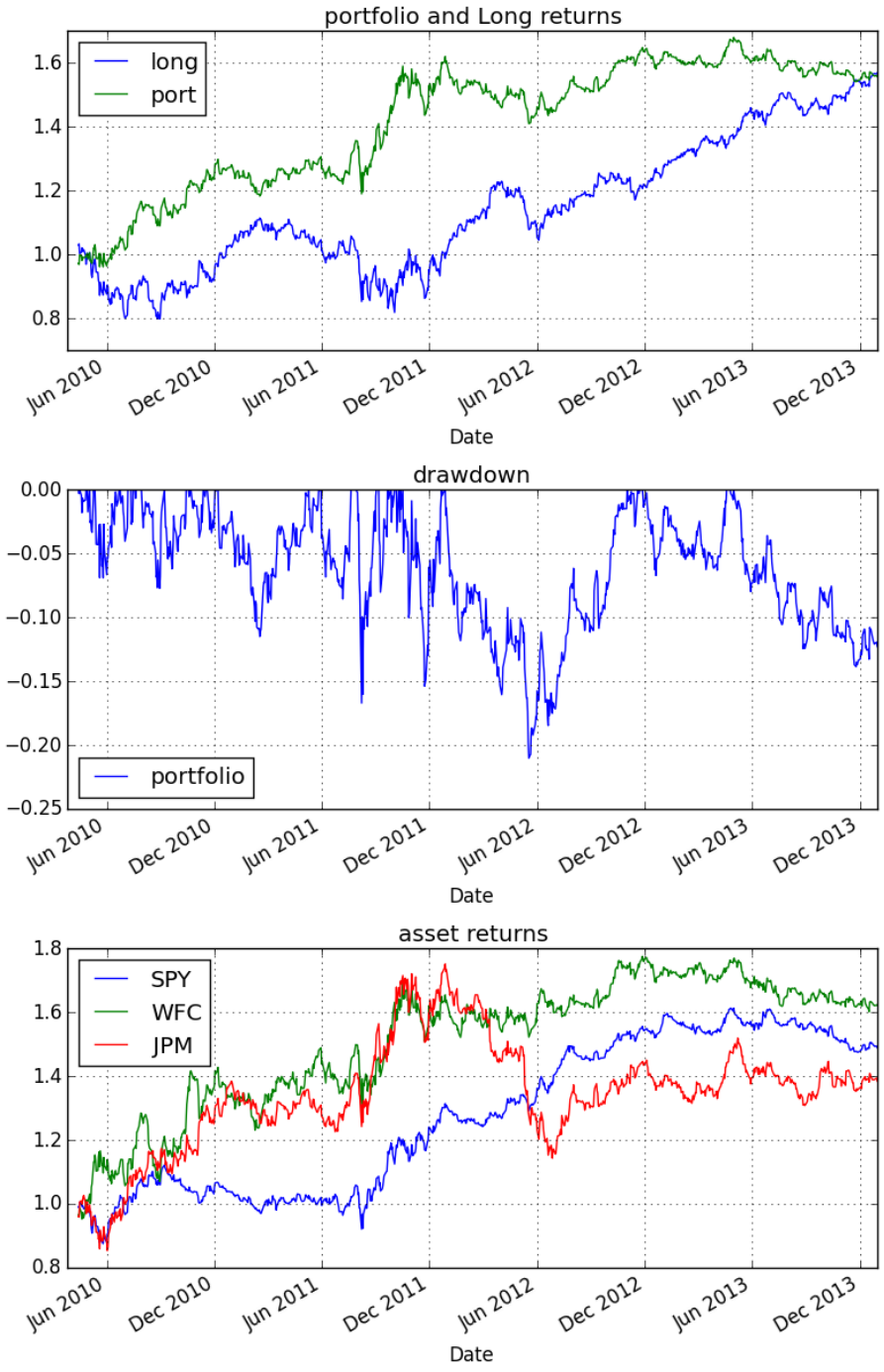


Figure 5: Equity curve analysis for Q1(a) Rule 2 backtest with trading cost 10 bps. $s$ =4 and $l$ =10 days are used.

**Optimal in-sample values** - We concentrate on the short term and use the following code to find the optimal parameters:

```
# optimization
hm = {}
for l in np.arange(1,21,1):
    sr = {}
    for s in np.arange(1,21,1):
        print l,s
        if l>s:
            sr[s] = run_bt({'l':l, 's':s}, calc_signal, tc=0.001)
            ['stats']['sharpe']['portfolio']
        else:
            sr[s] = np.nan
    hm[l] = Series(sr)
hm = DataFrame(hm)
hm.columns.name='l'
hm.index.name='s'
heat_map(hm, title='heatmap of performance sharpe; Q1 MA Rule 2')
```

We get fig.6

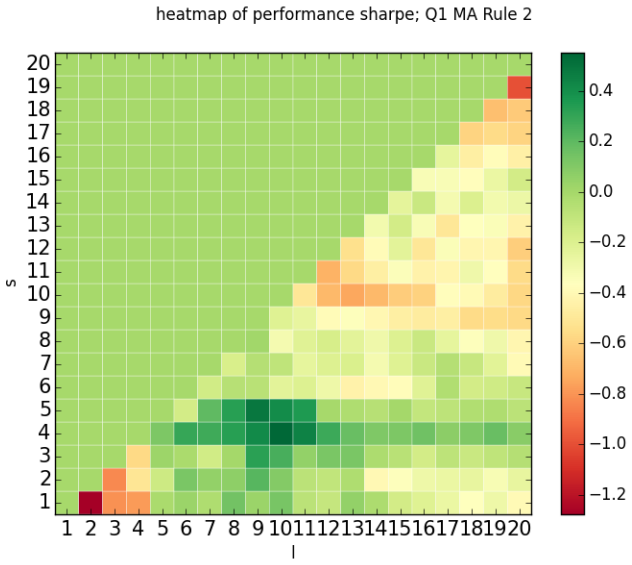


Figure 6: Heatmap for Q1(a) Rule 2 backtest with trading cost 10 bps.

We see that the optimal parameter values for short and long is around **(4,10)**. Long term heatmap does not improve the performance, so we don't show it here.

## Problem 1a Rule 3

We implement the signal for Bollinger band trading Rule 3 here as follows:

```
def calc_signal(prc, param):
    # 1a Moving average tarading rules - Rule 3
    L = param['L']
    p_ma = rolling_mean(prc, L)
    p_wt = DataFrame(columns=p_ma.columns, index=p_ma.index)
    for item in p_wt.columns:
        p_wt[item] = np.convolve(p_ma[item], np.arange(L, 0, -1))[:-L+1]/(L*(L+1)/2)
    p_sd = rolling_std(prc, L)
    sig = 0*p_ma.fillna(0)
    for com in sig.columns:
        for i in range(1,len(sig)):
            tp,t = sig.index[i-1], sig.index[i]
            sig[com][t] = sig[com][tp] # copy over previous state
            if sig[com][tp] == 0:
                # evaluate entry
                if prc[com][tp]>p_wt[com][tp]+2*p_sd[com][tp]: # open short
                    sig[com][t] = -1
                elif prc[com][tp]<p_wt[com][tp]-2*p_sd[com][tp]: # open long
                    sig[com][t] = 1
            else:
                # evaluate exit
                if sig[com][tp] == -1:
                    if prc[com][tp]<p_wt[com][tp]: # close short
                        sig[com][t] = 0
                if sig[com][t] == 1:
                    if prc[com][tp]>p_wt[com][tp]: # close long
                        sig[com][t] = 0
    return sig

# signal run
bt_0 = run_bt({'L':6}, calc_signal, tc=0.000, showf=False, alloc='equal_initial')
bt_tc = run_bt({'L':6}, calc_signal, tc=0.001, showf=True, alloc='equal_initial')
df = DataFrame({'0bps': bt_0['stats'].T['portfolio'],
'10bps': bt_tc['stats'].T['portfolio']})
print df.to_latex()
```

**Evaluation of Rule 3**: This rule also is a contrarian strategy with going short when the price go beyond the positive bollinger band and going long when they go below the lower bollinger band.

**Qualitative performance**: We try a window of 6 days. The plot in fig.7 shows the equity

curve along with drawdown curve and the performance of the three assets. The strategy does not outperform long only portfolio, in fact it is a very mediocre strategy. There are large drawdowns upto 14% and only WFC contribute to the positive performance.

**Quantitative performance**: Table 3 shows the performance with and without trading cost. This is clearly a very weak performance with a Sharpe ratio of 0.08.

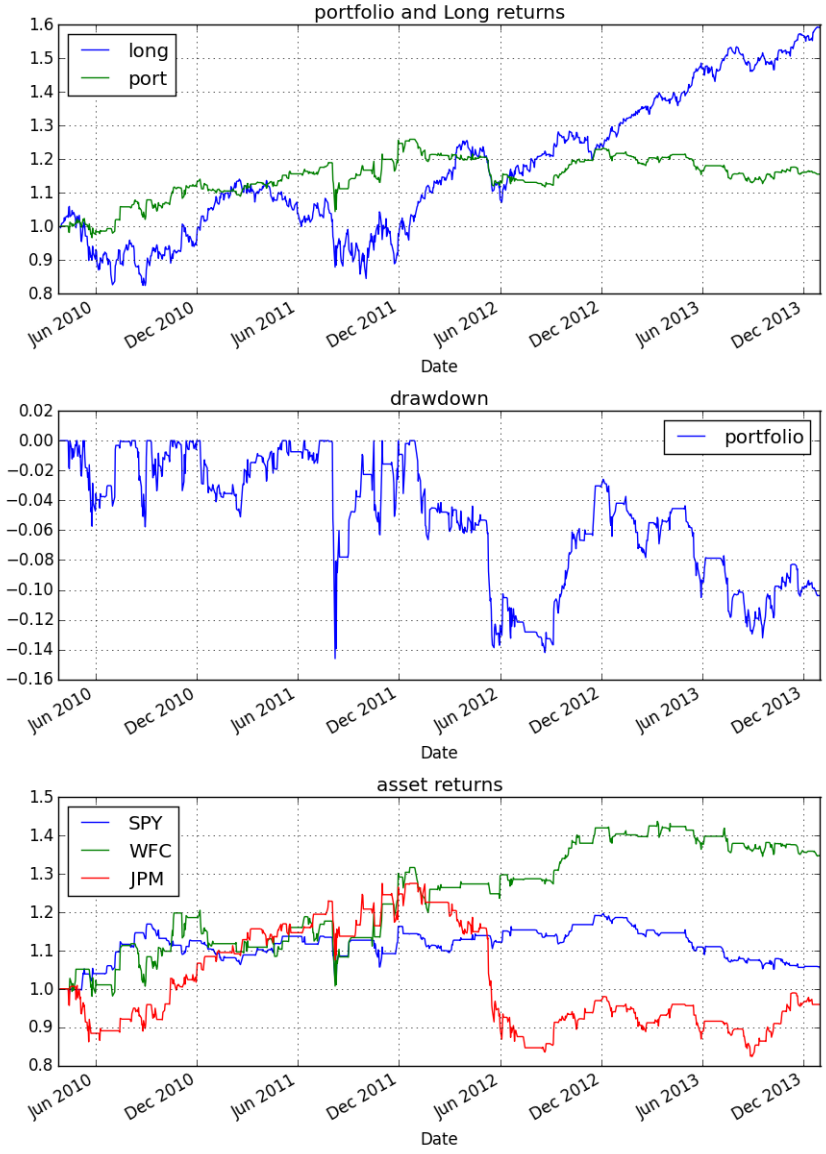


Figure 7: Equity curve analysis for Q1(a) Rule 3 backtest with trading cost 10 bps. $L$ =6 days is used.

| | 0bps | 10bps |
|---|---|---|
| IR | 0.1544971 | -0.08162516 |
| PNL | 0.2757012 | 0.1543328 |
| Sortino | 0.3292552 | 0.08765961 |
| Treynor | 0.01543566 | 0.00388769 |
| calmar | 0.4972689 | 0.2280463 |
| sharpe | 0.3087509 | 0.07745397 |

Table 3: Performance numbers (annualized) for Rule 3 for 0 and 10 bps trading cost.

**Optimal in-sample values** - We concentrate on the short term and use the following code to find the optimal parameters:

```
hm = {}
for l in list(np.arange(5,21,1))+list(np.arange(25,126,5)):
    print l
    hm[l] = run_bt({'L':l}, calc_signal, tc=0.001, alloc='equal_initial')
    ['stats']['sharpe']['portfolio']
hm = Series(hm)
hm.index.name='L'
hm.plot(style='k.-',title='heatmap of performance sharpe; Q1 MA Rule 3',grid=True)
```

We get fig.8. We see that the optimal parameter values for $L$ is **6 days**. The performance is still negative for most of the parameter range.

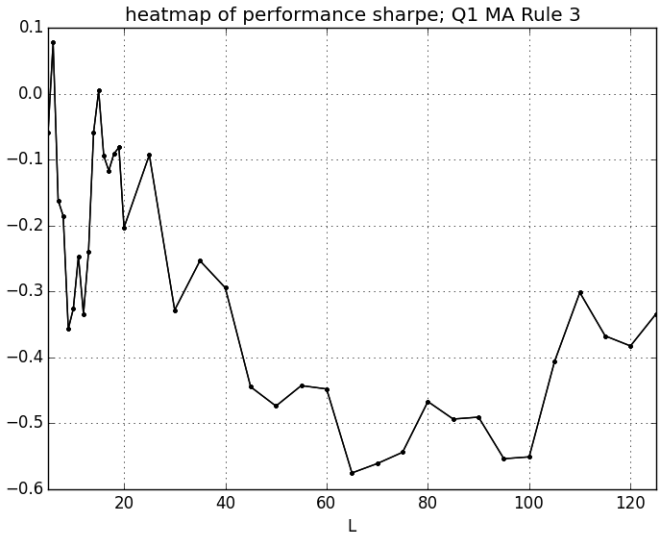


Figure 8: Sharpe for Q1(a) Rule 3 backtest with trading cost 10 bps.

To get an illustration of how this rule works we can look at the close up of entry exit shown for JPM for this rule in fig.9.

```
# plot signal and trade entry exit
com = 'JPM'
L = 6
fig = figure()
ax1 = fig.add_subplot(311)
prc = bt_tc['prc']
p_ma = rolling_mean(prc, L)
p_wt = DataFrame(columns=p_ma.columns, index=p_ma.index)
for item in p_wt.columns:
    p_wt[item] = np.convolve(p_ma[item], np.arange(L, 0, -1))[:-L+1]/(L*(L+1)/2)
p_sd = rolling_std(prc, L)
DataFrame({'prc':prc[com],
           'wma':p_wt[com],
           'wma+2std':p_wt[com]+2*p_sd[com],
           'wma-2std':p_wt[com]-2*p_sd[com]}).\
    plot(ax=ax1,style='.-',grid=True, title=com+': price and 10d MA and w=1.04')
ax2 = fig.add_subplot(312,sharex=ax1)
bt_tc['sig'][com].plot(ax=ax2,style='.-', grid=True, title='Signal')
ax3 = fig.add_subplot(313,sharex=ax1)
bt_tc['ar'][com].cumsum().plot(ax=ax3, style='.-',grid=True, title='PnL')
```

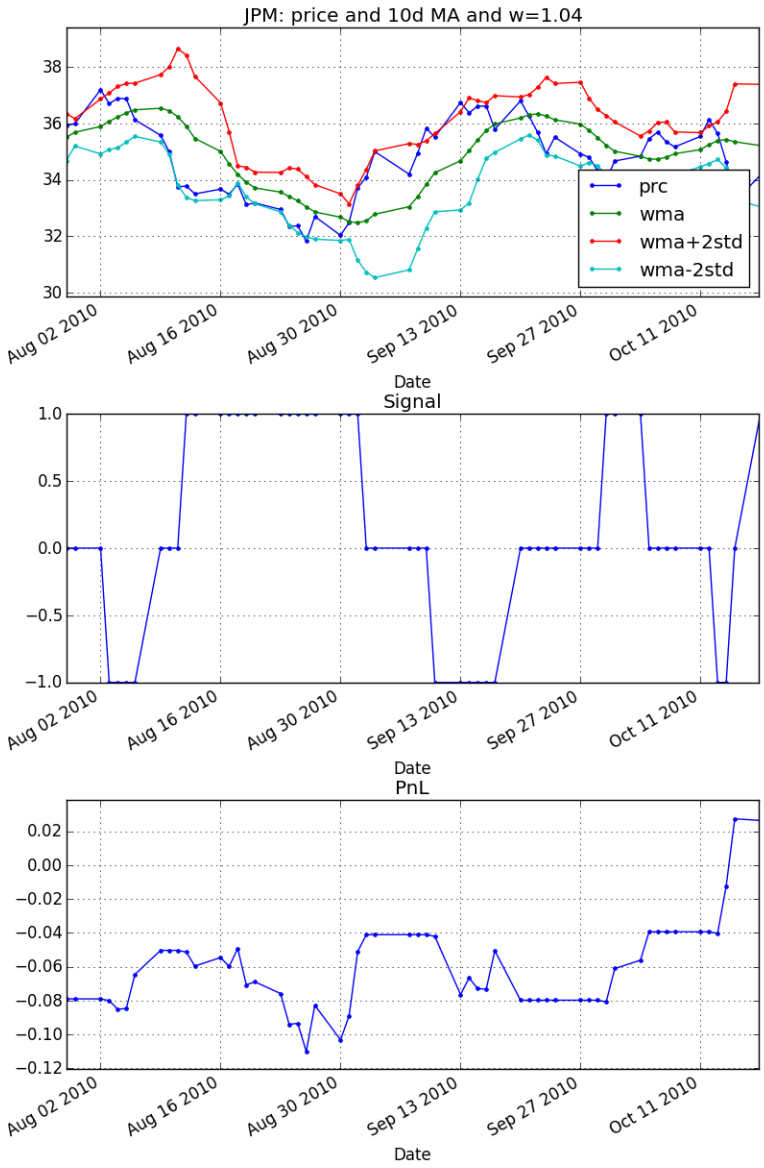


Figure 9: Entry exit rules illustrated for Q1(a) Rule 3 backtest with trading cost 10 bps. $L$ =6.

## Problem 1b Rule 4

We implement the signal for Oscillator trading Rule 4 here as follows:

```
def calc_signal(prc, param):
    # 1b RSI rule - Rule 4
    L = param['L']
    ret = prc.pct_change()
    up = ret.apply(np.sign)
    up[up<0] = 0
    down = -ret.apply(np.sign)
    down[down<0] = 0
    ut = rolling_sum(up,L)
    dt = rolling_sum(down,L)
    rsi = 100*ut/(ut+dt)
    sig = 0*ret.fillna(0)
    for com in sig.columns:
        for i in range(1,len(sig)):
            tp,t = sig.index[i-1], sig.index[i]
            sig[com][t] = sig[com][tp] # copy over previous state
            if sig[com][tp] == 0:
                # evaluate entry
                if rsi[com][tp]>70: # open short
                    sig[com][t] = -1
                elif rsi[com][tp]<30: # open long
                    sig[com][t] = 1
            else:
                # evaluate exit
                if sig[com][tp] == -1:
                    if rsi[com][tp]<50: # close short
                        sig[com][t] = 0
                if sig[com][t] == 1:
                    if rsi[com][tp]>50: # close long
                        sig[com][t] = 0
    return sig

# signal run
bt_0 = run_bt({'L':4}, calc_signal, tc=0.000, showf=False, alloc='equal_initial')
bt_tc = run_bt({'L':4}, calc_signal, tc=0.001, showf=True, alloc='equal_initial')
df = DataFrame({'0bps': bt_0['stats'].T['portfolio'],
'10bps': bt_tc['stats'].T['portfolio']})
print df.to_latex()
```

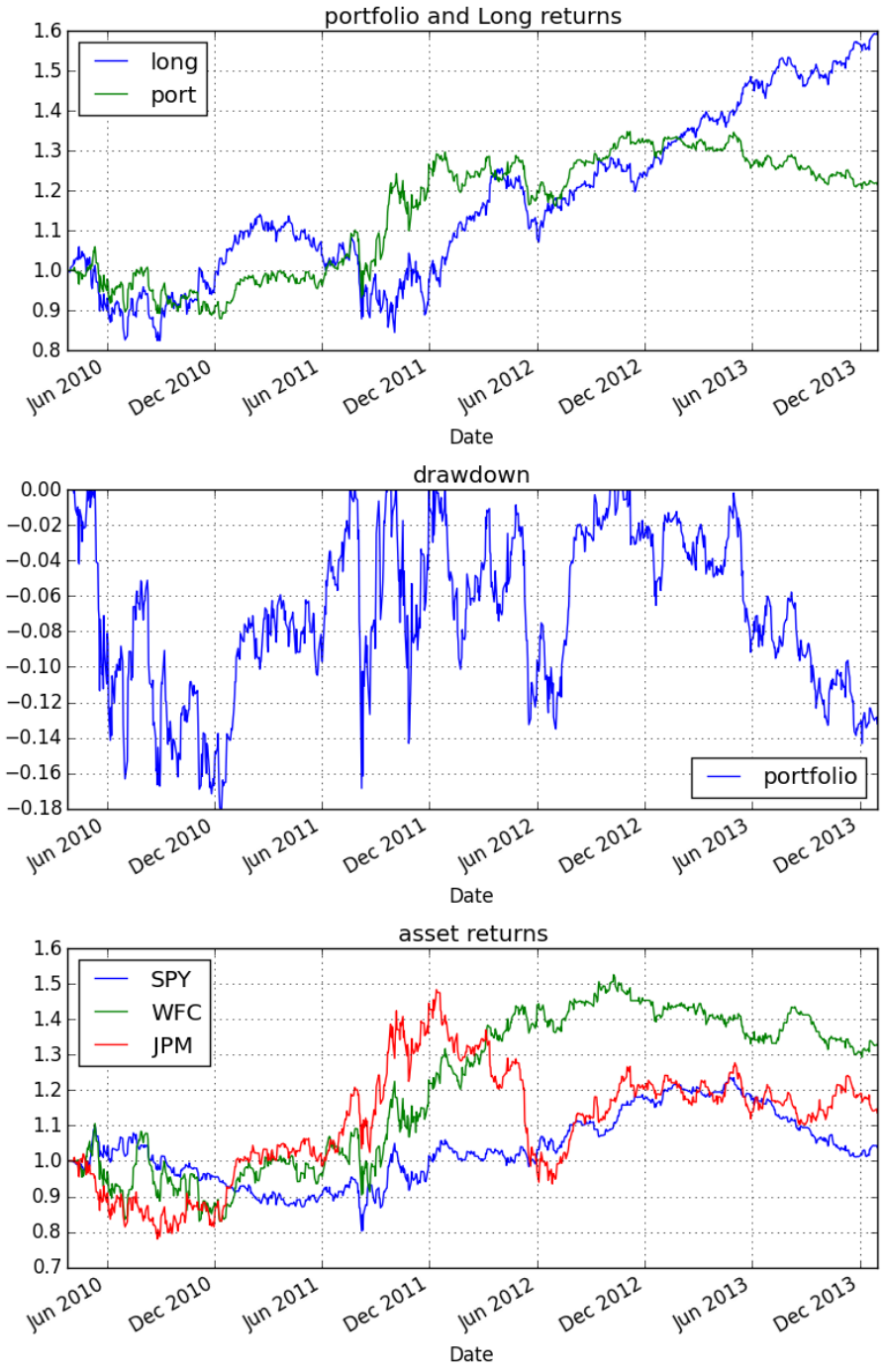


Figure 10: Equity curve analysis for Q1(b) Rule 4 backtest with trading cost 10 bps. $L$ =5 days is used.

**Evaluation of Rule 4**: This rule also is a contrarian strategy with going short when the RSI go beyond 70 and goigng long when they go below 30.

**Qualitative performance**: We try a window of 4 days. The plot in fig.10 shows the equity curve along with drawdown curve and the performance of the three assets. The strategy does not outperform long only portfolio. There are big drawdowns and none of the assets contribute to the positive performance significantly, in the last year.

**Quantitative performance**: Table 4 shows the performance with and without trading cost. This is clearly a very weak performance with a Sharpe ratio of 0.14.

| | 0bps | 10bps |
|---|---|---|
| IR | 0.245105 | -0.02132705 |
| PNL | 0.4086183 | 0.2154051 |
| Sortino | 0.5248648 | 0.1843482 |
| Treynor | 0.01983634 | 0.006850414 |
| calmar | 0.6312346 | 0.227044 |
| sharpe | 0.3975438 | 0.1371907 |

Table 4: Performance numbers (annualized) for Rule 4 for 0 and 10 bps trading cost.

**Optimal in-sample values** - We concentrate on the short term and use the following code to find the optimal parameters:

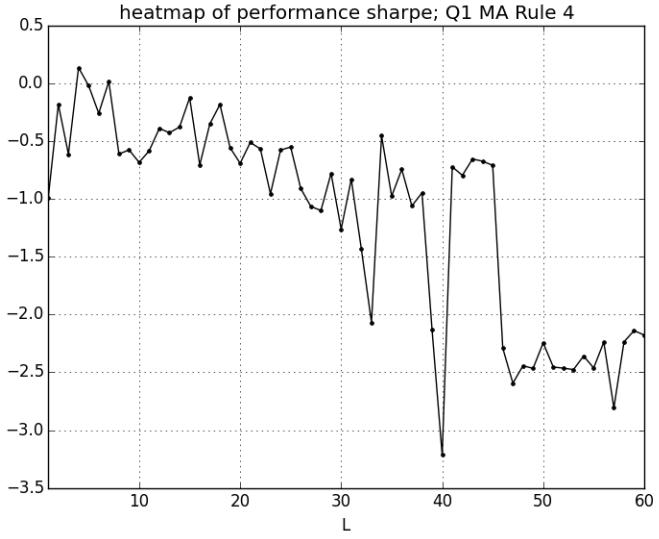


Figure 11: Sharpe for Q1(b) Rule 4 backtest with trading cost 10 bps.

```
# optimization
hm = {}
for l in np.arange(1,61,1):
    print l
    hm[l] = run_bt({'L':l}, calc_signal, tc=0.001,
     alloc='equal_initial')['stats']['sharpe']['portfolio']
hm = Series(hm)
hm.index.name='L'
hm.plot(style='k.-',title='heatmap of performance sharpe; Q1 MA Rule 4',grid=True)
```

We get fig.11. We see that the optimal parameter values for $L$ is **4 days**. The performance is slightly positive, but in general not very good.

The trading behavior can be seen from the illustration in fig.12

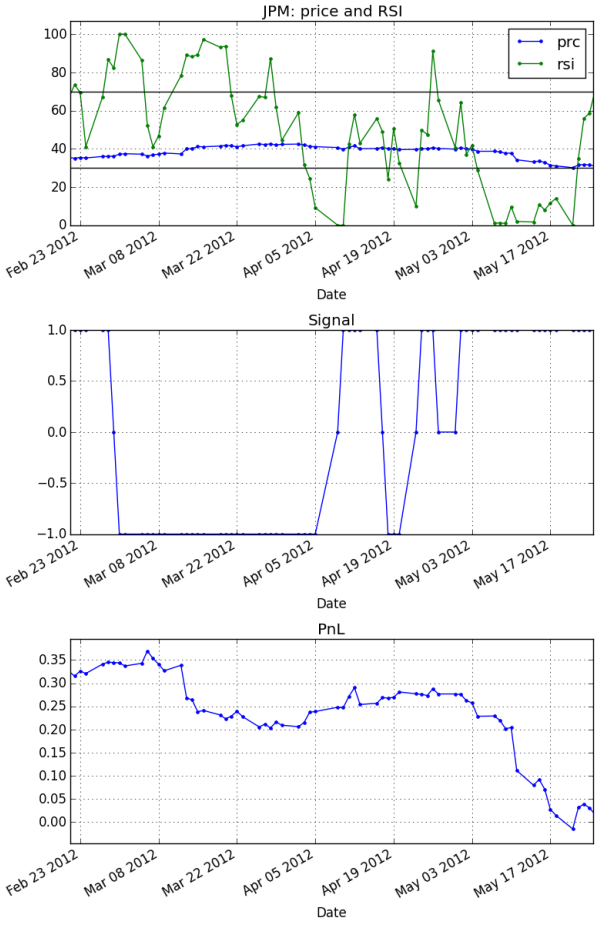


Figure 12: Entry exit rules illustrated for Q1(b) Rule 4 backtest with trading cost 10 bps.

## Problem 2

The method of Anatolyev and Gospodeinov (2010) of decomposing the returns into sign and magnitude is based on the idea that predicting them separately is more worth while and introduces nonlinear effects as well, which improves the predictability. The decomposition is

$$r_t = c + |r_t - c| sign(r_t - c) = c + |r_t - c|(2\mathbb{I}[r_t > c] - 1)$$

We saw in HW1 that autocorrelations in returns are hardly there (for daily data) but for returns square we do see significant autocorrealtion (and hence predictability). The absolute value of returns is similar to it's square and shows similar autocorrealtion and hence predicatability. Hence if we try to predict absolute value of returns, than simply returns, we are expected to do better!! Absolute returns $|r_t - c|$ is a positively valued variable and is modelled using multiplicative error framework of Engle (2002) $|r_t - c| = \psi_t \eta_t$, where $\psi_t = E[|r_t - c||F_{t-1}]$ and $\eta_t$ is a positive multiplicative error with $E[\eta_t|F_{t-1}] = 1$ and conditional distribution $\mathbb{D}$. $\psi_t$ can be modelled using logarithmic autoregressive conditional duration (LACD) as

$$ln\psi_t = \omega_v + \beta_v ln\psi_{t-1} + \gamma_l n|r_{t-1} - c| + \rho_v \mathbb{I}[r_{t-1} > c] + \boldsymbol{x}_{t-1}^T \boldsymbol{\delta}_v.$$

The second last term allows for regime-specific volatility dependence while the last term represents macroeconomic predictors of volatility. $\mathbb{D}$ can be modelled as constant parameter Weibull distribution (or others distributions with the shape parameter vector $\varsigma$ a function of the past).

Further, Christoffersen and Diebold (2006) show a remarkable result that suggests that time-varying volatility can generate sign predictability which would arise from time variability in second and higher-order moments. This leads us to parametrize $p_t$ as a dynamic logit model:

$$p_t = \frac{e^{\theta_t}}{1 + e^{\theta_t}} \quad \text{with} \quad \theta_t = \omega_d + \phi_d \mathbb{I}[t_{t-1} > c] + \boldsymbol{y}_{t-1}^T \boldsymbol{\delta}_d,$$

where the last term denotes macroeconomic variables (valuation ratios, interest rate) and realized measure (variance, bipower vriation, realized third and fourth moment of returns).

Lastly, we need a Copula model to connect the two to get joint distribution. To construct the bivariate conditional distribution of $R_t = [|r_t - c|, \mathbb{I}[r_t > c]]^T$ copula theory is used. In particular,

$$F_{R_t}(u, v) = C(F_{|r_t - c|}(u), F_{\mathbb{I}[r_t > c]}(v))$$

where $F$ denotes the CDF and $C(u, v)$ is a copula. Most common choices are Frank, Clayton or Farlie-Gumbel-Morgenstern copulas. Once the three ingredients of the joint distribution of $R_t$, i.e. the volatility model, the direction model and the copula are specified, the parameter vector can be estimated by maximum likelihood.

The main interest is the mean forecast of returns

$$E[r_t|F_{t-1}] = c - E[|r_t - c||F_{t-1}] + 2E[|r_t - c|\mathbb{I}[r_t > c]|F_{t-1}]$$

In terms of inference

$$\hat{r}_t = c - \hat{\psi}_t + 2\hat{\xi}_t$$

Under conditional independence or if conditional dependence is weak we have

$$\xi_t = E[|r_t - c||F_{t-1}]E[\mathbb{I}[r_t > c]|F_{t-1}] = \psi_t p_t.$$

so,

$$\hat{r}_t = c + (2\hat{p}_t - 1)\hat{\psi}_t.$$

Under the general case of dependence, the copula estimation is essential.

Hence, under the condition of independence or weak dependence, and only estimating the mean, it is possible to do this modelling without the copula and only estimating $\psi_t$ and $p_t$ from their regression equation. We can also ignore any macroeconomic variable for predictability and focus on lagged returns and volatility. Moreover, instead of using Engle's multiplicative error framework we simply use the autoregressive model for the magnitude equation. Specifically, we do the following prediction of returns on day t as

$$\hat{r}_t = c + (2\hat{p}_t - 1)\hat{\psi}_t,$$

where

$$\hat{p}_t = \frac{e^{\theta_t}}{1 + e^{\theta_t}} \quad \theta_t = \alpha_d + \beta_d \psi_{t-1} + \gamma_d \mathbb{I}[r_{t-1} > c]$$

and

$$ln\hat{\psi}_t = \alpha_v + \beta_v ln\psi_{t-1} + \gamma_v \mathbb{I}[r_{t-1} > c],$$

where

$$|r_t - c| = \psi_t$$

The way the code is implemented is such that on each day $t$ we use information till day $t-1$ to predict the returns and trade it that same day and realize the returns the next day. The signal is simply the sign of the predicted returns for the 'equal_initial allocation. The regression is done in a moving window with parameter $w$. Finally the signal is smoothed by taking a $m$ day moving window. We optimize the strategy for these two parameters.

We implement the signal for Decomposition rule here as follows:

```
from pandas import *
from stats242.HW2.bt import *
import numpy as np
from sklearn import linear_model
############### Decomposition ##########################
# overwriting the default signal

def calc_signal(prc, param):
    # decomposition rule
```

```
    ret = prc.pct_change().fillna(method='ffill')
    c = param['tc']
    w = param['win']
    m = param['m']
    # prepare regressors
    regs = Panel({
        'psi': (ret-c).apply(np.abs),
        'Lpsi': (ret-c).apply(np.abs).apply(np.log),
        'ind': (ret>c).astype(int)
    }).replace([np.inf, -np.inf], np.nan)
    pred = {}
    for com in ret.columns:
        # Magnitude regression
        x = regs.ix[['Lpsi','ind'],:,com].shift(1).dropna()
        y = regs.ix['Lpsi',:,com].reindex(x.index).dropna()
        x = x.reindex(y.index)
        model_Lpsi = ols(y=y,x=x,intercept=True, window=w)
        pred_Lpsi = Series(model_Lpsi.y_fitted, index=y.index)
        # Sign regression
        pt_pred={}
        for t in np.arange(w,len(ret)):
            sidx,eidx=ret.index[t-w],ret.index[t-1]
            logreg = linear_model.LogisticRegression()
            y = regs.ix['ind',sidx:eidx,com]
            x = regs.ix[['psi','ind'],sidx:eidx,com].shift(1).dropna()
            y = y.reindex(x.index).dropna()
            x = x.reindex(y.index)
            logreg.fit(x,y)
            pt_pred[ret.index[t]] = \
                logreg.predict(regs.ix[['psi','ind'],ret.index[t],com])[0]
        pt_pred = Series(pt_pred)
        pred[com] = c+(2*pt_pred-1)*pred_Lpsi.apply(np.exp)
    sig = DataFrame(pred).reindex(ret.index).fillna(method='ffill')
    return rolling_mean(sig,m)


# signal run
bt_0 = run_bt({'tc':0., 'win':35, 'm':8}, calc_signal,
tc=0.000, alloc='equal_initial', showf=True)
bt_tc = run_bt({'tc':0.001, 'win':35, 'm':8}, calc_signal,
tc=0.001, alloc='equal_initial', showf=True)
df = DataFrame({'0bps': bt_0['stats'].T['portfolio'],
```

```
                '10bps': bt_tc['stats'].T['portfolio']})
print df[['0bps','10bps']].to_latex()
```

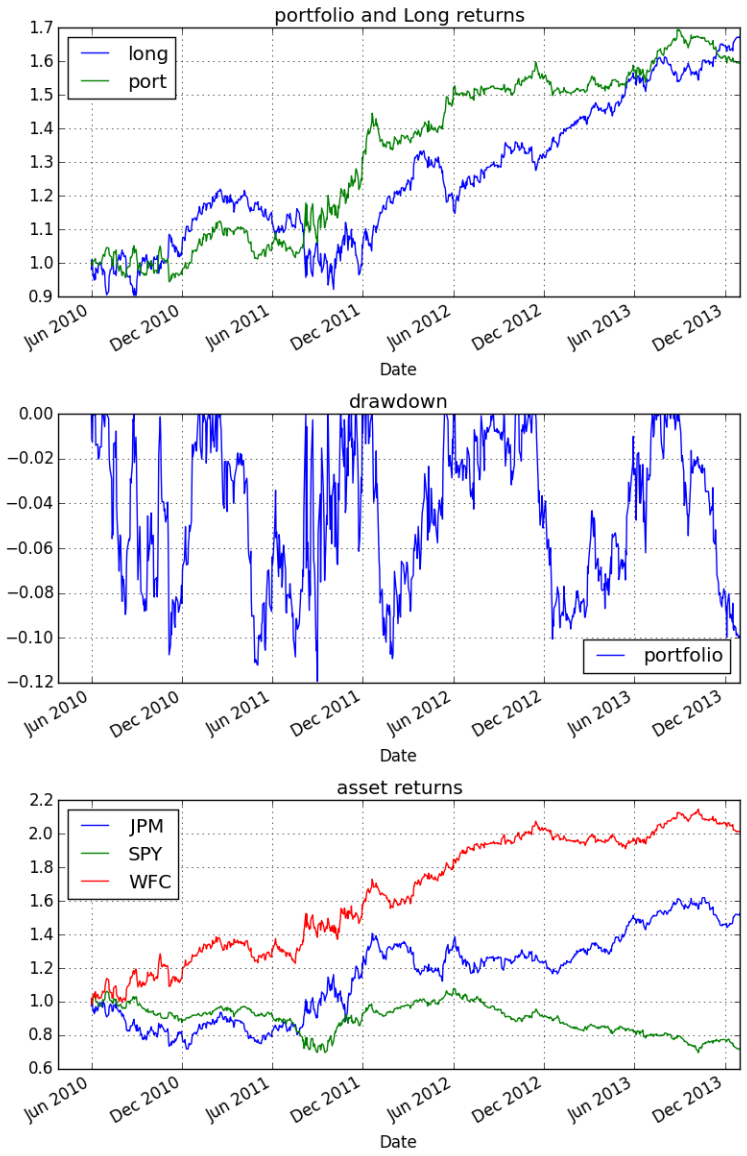


Figure 13: Equity curve analysis for Q2 Decomposition Rule backtest with trading cost 10 bps. $w$ =35 days and $m$ =8 days are used.

**Evaluation of Decomposition Rule**: This rule has a flavor of momentum strategy (due to the smoothing). The predicted probability when averaged over a moving window of 8 days improves the performance. Also, the signal depend on the trading cost here, so they are not directly

comparable in the 'natural' sense of trading cost analysis.

**Qualitative performance**: We try a regression window of 35 days and a smoothing window of 8 days. The plot in fig.13 shows the equity curve along with drawdown curve and the performance of the three assets. The strategy does not outperforms long only portfolio. There are medium drawdowns or around 10% and the performance comes form JPM and WFC at the expense of SPY.

**Quantitative performance**: Table 5 shows the performance with and without trading cost. This is a strong performance, specially in terms of IR (1.1), as expected in literature for the decomposition model, vs the other models tried out in question 1. The net Sharpe ratio of 0.70.

| | 0bps | 10bps |
|---|---|---|
| IR | 1.151764 | 1.102852 |
| PNL | 0.7161898 | 0.5952357 |
| Sortino | 1.260342 | 1.002231 |
| Treynor | -0.04212204 | -0.02020603 |
| calmar | 1.812145 | 1.624701 |
| sharpe | 0.9153902 | 0.6995933 |

Table 5: Performance numbers (annualized) for Decomposition Rule for 0 and 10 bps trading cost.

**Optimal in-sample values** - We concentrate on the short term and use the following code to find the optimal parameters:

```
# optimization
hm = {}
for l in np.arange(10,61,5):
    s = {}
    for m in np.arange(2,21,2):
        print l,m
        s[m] = run_bt({'win':l, 'tc':0.001, 'm':m},
                      calc_signal, tc=0.001,
                      alloc='equal_initial')[
            'stats']['sharpe']['portfolio']
    hm[l] = Series(s)
hm = DataFrame(hm)
hm.columns.name='l'
hm.index.name='m'
heat_map(hm, title='heatmap of performance sharpe; Q2 decomposition rule')
```

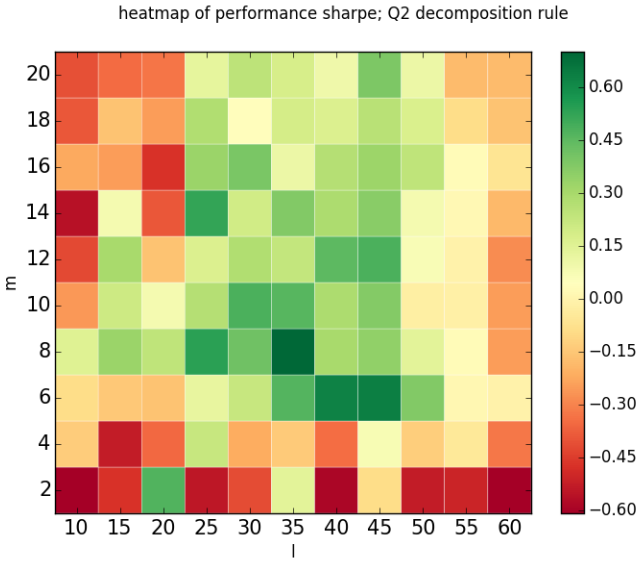


Figure 14: Sharpe for Q2 Decomposition Rule backtest with trading cost 10 bps.

We get fig.14. We see that the optimal parameter values for $m$ and $L$ is **(8,35) days**, where $m$ is the smoothing window and $L$ is the regression window. The performance is much improved than what we saw in problem 1, with highest Sharpe and impressive IR. We will evaluate the out of sample performance in Problem 5.

## Problem 3a

This problem deals with analysis of Pairs trading via distance rule. The distance is measured as the square of difference between normalized prices of the pairs. At the last trading day of the month, we look back w days and calculate pair distances for all the three pairs. At the end of next month we reset all the positions and start over. The entry rule is if the distance square crosses the entry threshold and exit occurs when the exit threshold is crossed. We optimize the strategy for these three parameters. Though the problem prescribe 60 days window we include it as a special case of the general analysis we do here. The initial portfolio consists of equally weighted three pairs using the scheme **pairs**.

We implement the signal as follows:

```
from pandas import *
from stats242.HW2.bt import *
import numpy as np

############### Pairs: distance ##########################

def find_pairs_list(com):
    plist = []
    for com1 in com:
        for com2 in com:
            if com1!=com2:
                if ((com1,com2) not in plist) & ((com2,com1) not in plist):
                    plist.append((com1, com2))
    return plist

def pairs_converged(item, ptrade, pr, pairs_mean, pairs_std, tt, ex):
    com1, com2 = item
    if ptrade[item][tt]>0:
        if (pr[com1][tt]-pr[com2][tt])**2 < (pairs_mean[item]+ex*pairs_std[item]):
            return True

def pairs_diverged(item, ptrade, pr, pairs_mean, pairs_std, tt, sl):
    com1, com2 = item
    if ptrade[item][tt]>0:
        if (pr[com1][tt]-pr[com2][tt])**2 > (pairs_mean[item]+sl*pairs_std[item]):
            return True

def calc_signal(prc, param, printf=True):
    en, ex, sl, win = param['entry'], param['exit'], param['stoploss'], param['win']
    # make the pairs trade matrix initialized to zero
```

```
    # we store trade for com1, com2 trade is just the opposite
    pairs_list = find_pairs_list(prc.columns)
    ptrade,pairs_mean,pairs_std,pm,ps,pd={},{},{},{},{},{}
    for com1, com2 in pairs_list:
        ptrade[(com1,com2)] = (prc[com1]*0).astype(int)
        pm[(com1,com2)] = (prc[com1]*0)
        ps[(com1,com2)] = (prc[com1]*0)
        pd[(com1,com2)] = (prc[com1]*0)
        pairs_mean[(com1, com2)] = 0
        pairs_std[(com1, com2)] = 0
    init_flag = False
    for t in range(win,len(prc.index)-2):
        # prediction for t+1 in this step
        tt = prc.index[t]
        ttn = prc.index[t+1]
        for item in pairs_list:
            com1,com2 = item
            # default carry last position
            ptrade[item][ttn] = ptrade[item][tt]
            pm[item][ttn] = pairs_mean[item]
            ps[item][ttn] = pairs_std[item]
            try:
                pd[item][ttn] = (pr[com1][tt]-pr[com2][tt])**2
            except:
                pass
            if tt.month!=ttn.month: # if month end
                init_flag = True
                t_idx = prc.index[t+1]           # last day before prediction + 1
                m3_t_idx = prc.index[t-win]     # start day of last 60 days
                # normalize price so that start value is 1
                pr = prc.ix[m3_t_idx:,:]/prc.ix[m3_t_idx,:]
                # find distance and std of distance: based on past w days data
                # first close the existing pair
                ptrade[item][ttn] = 0
                # construct condition for next month
                dis_sqr = (pr[com1][:t_idx]-pr[com2][:t_idx])**2
                pairs_mean[item] = dis_sqr.mean()
                pairs_std[item] = dis_sqr.std()
            elif init_flag:
                # check exit rule
                if ptrade[item][tt]!=0:
                     if pairs_converged(item, ptrade, pr, pairs_mean, pairs_std, tt, ex) \
```

```
                    or pairs_diverged(item, ptrade, pr, pairs_mean, pairs_std, tt, sl):
                    ptrade[item][ttn] = 0
            # check entry rules
            elif (pr[com1][tt]-pr[com2][tt])**2>(pairs_mean[item]+en*pairs_std[item]):
                # buy cheap and sell expensive
                ptrade[item][ttn] = -1 if pr[com1][tt]>pr[com2][tt] else 1
    # construct the final signal dataframe
    sig = prc*0.
    for item in ptrade.keys():
        com1,com2=item
        sig[com1] += ptrade[item]
        sig[com2] -= ptrade[item]
        if printf:
            DataFrame({
                'dist': pd[item],
                'mean': pm[item],
                'mean+1*std': pm[item]+1*ps[item],
                'mean+2*std': pm[item]+2*ps[item],
                'mean+5*std': pm[item]+5*ps[item],
            }).plot(title=item,grid=True)
    if printf:
        DataFrame(ptrade).plot(style='.-', grid=True)
    return sig

# signal run
bt_0 = run_bt({'entry':5, 'exit':2, 'stoploss':10, 'win':10},
 calc_signal, tc=0.00, alloc='pairs', showf=True)

bt_tc = run_bt({'entry':5, 'exit':2, 'stoploss':10, 'win':10},
 calc_signal, tc=0.001,alloc='pairs', showf=True)
df = DataFrame({'0bps': bt_0['stats'].T['portfolio'],
'10bps': bt_tc['stats'].T['portfolio']})
print df.to_latex()
```

**Evaluation of pairs via distance**: The basic mechanism of the trading is shown in fig.15. The distance squared is always positive and we can see clearly that it keeps returning back to zero. This is what the pairs trading is betting on. We choose various entry and exit conditions and but we show later that none of them result in good performance on a net basis. The way the allocation work is that we don't alter the relative values of the pair and aggregate the portfolio by simply summing up the components.

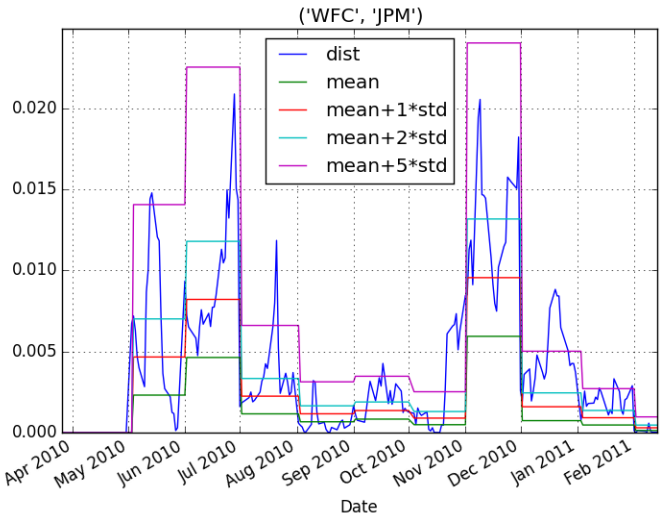


Figure 15: Entry exit rules illustrated for pairs trading via distance method. Moving window of 10 days used.

**Qualitative performance** - We start with a moving window of $L = 10$ days, entry of 5 standard deviation from mean and exit at two standard deviation from mean. We put a stop-loss at 10 standard deviation from the mean. We see in fig.16 that the rule is not able to time the entry exits profitably. The gross returns are positive but the trading cost puts it in the negative territory.

**Quantitative performance** - To quantify the performance let's look at Sharpe, Treynor, Sortino, Calmar and Information ratios (all annualized). We also look at the PNL with and without transaction cost of 10 basis points. Risk free rate is 3%.

| | 0bps | 10bps |
|---|---|---|
| IR | 0.7453647 | 0.004251147 |
| PNL | 0.7913759 | 0.0417287 |
| Sortino | 0.687988 | -0.08537529 |
| Treynor | -0.06639743 | 0.007481289 |
| calmar | 0.8279625 | -0.02870346 |
| sharpe | 0.6623903 | -0.07722153 |

Table 6: Performance numbers (annualized) for distance based pairs for 0 and 10 bps trading cost.

Table 6 shows the various ratios. We see that applying a trading cost of 10 bps drags down each and every ratio as expected. The net Sharpe ratio becomes negative to -0.08, while IR is also low at 0.004. Other ratios are low as well. Dollar 1 invested in the strategy had yielded a profit of

$0.04 in the time period of the analysis. Clearly this does not make the cut.

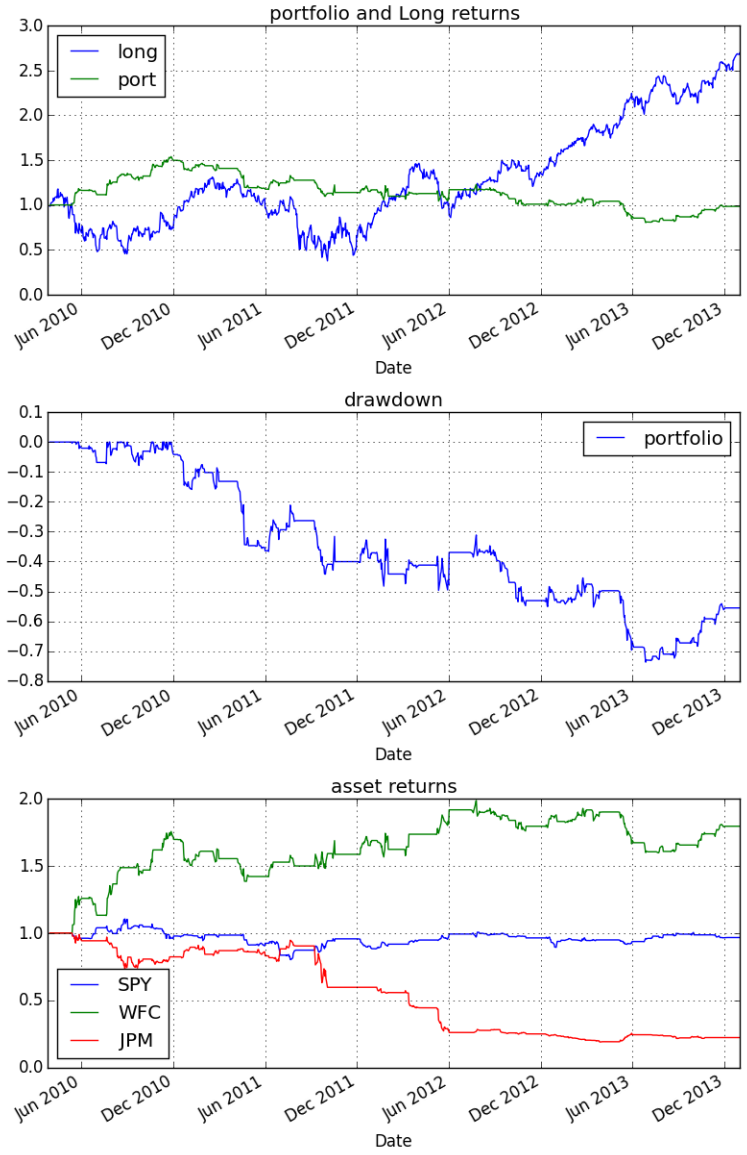


Figure 16: Equity curve analysis for pairs based on distance backtest with trading cost 10 bps

**Optimal in-sample values** - To get the optimal value of trading window entry and exit we run two grid searches and produce the fig.17 using the following code:

```
# optimization
param={
```

```
    #'win': 10,
    'en': np.arange(0.5,6.1,0.5),
    'ex': np.arange(0,4,0.25),
}
hm = {}
for i in param[param.keys()[0]]:
    s = {}
    for j in param[param.keys()[1]]:
        print i,j
        s[j] = run_bt({'entry':i, 'exit':j, 'stoploss':10, 'win':10},
                      calc_signal, tc=0.001,
                      alloc='pairs')[
            'stats']['sharpe']['portfolio']
    hm[i] = Series(s)
hm = DataFrame(hm)
hm.columns.name=param.keys()[0]
hm.index.name=param.keys()[1]
heat_map(hm, title='heatmap of performance sharpe; Q3a Distance Pair rule')
```

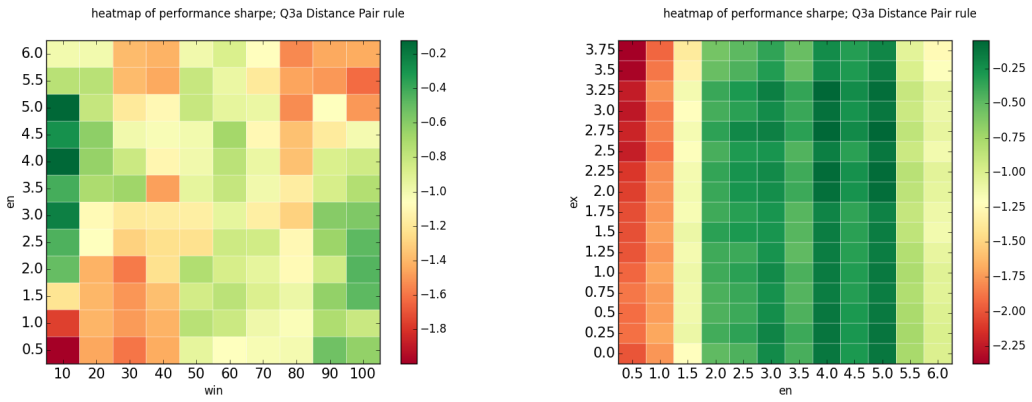


Figure 17: Heatmap for distance based pairs backtest with trading cost 10 bps. Left: entry vs moving window, Right: entry vs exit.

The heat map plots Sharpe Ratio numbers, i.e. **we chose Sharpe as the criteria to choose the optimal value like in previous questions**. Other ratios could also be fairly used as a selection criteria. We see that the performance is negative everywhere. A optimal and stable region looks like **(10,5,2)** for moving window, entry and exit. We choose it as the optimal signal for later comparison. We fail to find a good signal here, but the mechanics of entry exit is as expected.

## Problem 3b

We start by showing the calculations for cointegration relationship between JPY and WFC, to develop the mathematics for the backtest. We are following **Engle and Granger approach** to finding pairs, instead of log of prices we simply use prices. For cointegration based approach see problem 4. We are not establishing the cointegration relationship here as such (though we see weak relationship here) but only showing how to do the calculations which will be used in a moving window fashion later in the strategy code. For a bivariate case we can simply use regression to get the beta between the two price series and construct the the mean reverting relationship. We show below the steps of the process.

```
mod = pd.ols(x=prc['WFC'],y=prc['JPM'],intercept=True)

In[86]: mod
Out[86]:
-------------------------Summary of Regression Analysis-------------------------
Formula: Y ~ <x> + <intercept>
Number of Observations:         946
Number of Degrees of Freedom:   2
R-squared:         0.8115
Adj R-squared:     0.8113
Rmse:              3.1107
F-stat (1, 944):  4063.9132, p-value:     0.0000

Degrees of Freedom: model 1, resid 944
-----------------------Summary of Estimated Coefficients-----------------------
      Variable       Coef    Std Err     t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
             x     1.0761     0.0169      63.75     0.0000     1.0430     1.1091
     intercept     7.1405     0.5189      13.76     0.0000     6.1235     8.1575
---------------------------------End of Summary---------------------------------
```

We see that regressing the price of WFC on the price of JPM gives a hightly significant $\beta$ of 1.0761. The plot of ACF and PACF of residual in fig.18 shows that there is significant autocorrelation. This is same as doing Dublin-Watson test or using Dickey-Fuller test to establish the null hypothesis if unit root presence.

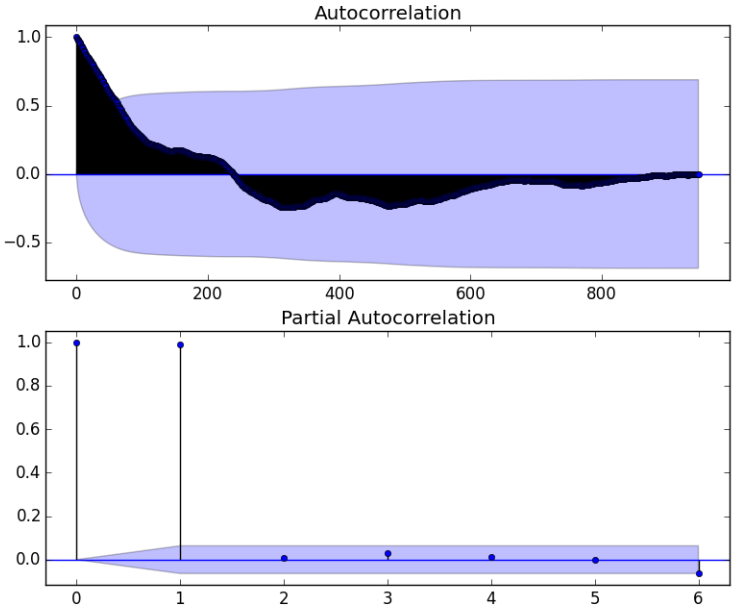


Figure 18: ACF and PACF of residual series of WFC regressed on JPY prices.

```
res = mod.resid
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
fig = figure()
ax1 = fig.add_subplot(211)
plot_acf(res, ax=ax1)
ax2 = fig.add_subplot(212)
plot_pacf(res,ax=ax2, lags=6)

fig = figure()
ax1=fig.add_subplot(211)
prc[['WFC','JPM']].plot(ax=ax1,title='P1t and P2t',grid=True)
ax2=fig.add_subplot(212)
res.plot(ax=ax2,title='wt',grid=True)
```

We further look at the plots of the price series and the $w = p_1 - \beta p_2$ series (demeaned) in fig.19. The strategy will bet on the mean reversion of $w$.

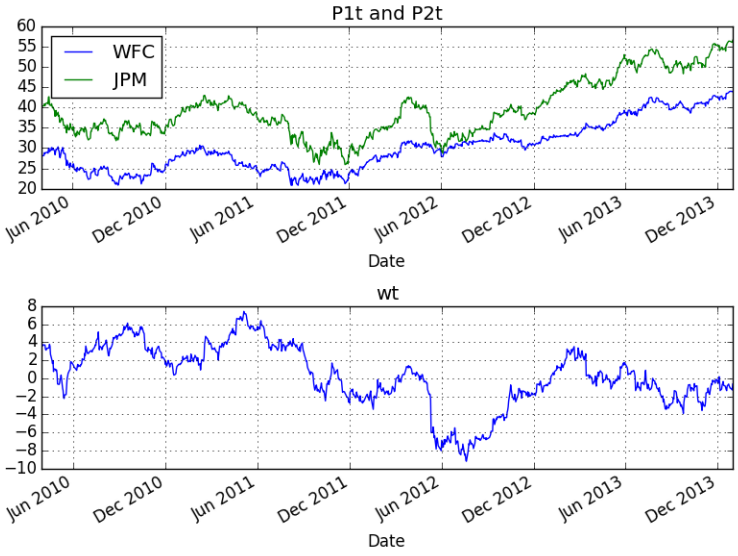


Figure 19: Time series of the prices and the residual.

We then go ahead and calculate the $\alpha$s as 0.0734 and -0.7105. They are opposite in signs suggesting tradability.

```
In[106]: mod1 = ols(x=res, y=ret['JPM'],intercept=True)
print mod1.summary
-------------------------Summary of Regression Analysis-------------------------
Formula: Y ~ <x> + <intercept>
Number of Observations:         946
Number of Degrees of Freedom:   2
R-squared:         0.0012
Adj R-squared:     0.0002
Rmse:              7.1603
F-stat (1, 944):     1.1455, p-value:     0.2848

Degrees of Freedom: model 1, resid 944
-----------------------Summary of Estimated Coefficients------------------------
      Variable       Coef    Std Err     t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
             x     0.0734     0.0686       1.07     0.2848    -0.0610     0.2078
     intercept    39.5650     0.2335     169.44     0.0000    39.1073    40.0227
---------------------------------End of Summary---------------------------------
```

```
In[107]: mod2 = ols(x=res, y=ret['WFC'],intercept=True)
print mod2.summary
-------------------------Summary of Regression Analysis-------------------------
Formula: Y ~ <x> + <intercept>
Number of Observations:         946
Number of Degrees of Freedom:   2
R-squared:         0.1620
Adj R-squared:     0.1612
Rmse:              5.4906
F-stat (1, 944):   182.5473, p-value:     0.0000

Degrees of Freedom: model 1, resid 944
-----------------------Summary of Estimated Coefficients-----------------------
      Variable       Coef    Std Err     t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
             x    -0.7105     0.0526     -13.51     0.0000    -0.8136    -0.6074
     intercept    30.3386     0.1791     169.44     0.0000    29.9877    30.6896
---------------------------------End of Summary---------------------------------
```

We can now use this setup in a rolling fashion to check for cointegrated pairs and design a strategy around it. A more complicated analysis will check for the statistical genuinity of the cointegration relationship. We will implement only a very simple model for illustration. We construct the 'mean-reverting' series for all the three pairs and apply entry exit rule on them in the rolling fashion, without checking for significance.

We implement the signal as follows:

```
from pandas import *
from stats242.HW2.bt import *
import numpy as np

############### Pairs: Conintegration ##########################
# overwriting the default signal
def find_pairs_list(com):
    plist = []
    for com1 in com:
        for com2 in com:
            if com1!=com2:
                if ((com1,com2) not in plist) & ((com2,com1) not in plist):
                    plist.append((com1, com2))
    return plist

def pairs_converged(item, ptrade, prc, pairs_mean, pairs_std, tt, ex, beta):
```

```
    com1, com2 = item
    if ptrade[item][tt]==1:
        # looking for exit from top
        if (prc[com2][tt]-beta*prc[com1][tt]) < (pairs_mean[item]+ex*pairs_std[item]):
            return True
    elif ptrade[item][tt]==-1:
        # looking for exit from botton
        if (prc[com2][tt]-beta*prc[com1][tt]) > (pairs_mean[item]-ex*pairs_std[item]):
            return True


def pairs_diverged(item, ptrade, prc, pairs_mean, pairs_std, tt, sl, beta):
    com1, com2 = item
    if ptrade[item][tt]==1:
        # looking for going to >sl
        if (prc[com2][tt]-beta*prc[com1][tt]) > (pairs_mean[item]+sl*pairs_std[item]):
            return True
    elif ptrade[item][tt]==-1:
        # looking for going below -sl
        if (prc[com2][tt]-beta*prc[com1][tt]) < (pairs_mean[item]-sl*pairs_std[item]):
            return True

def calc_signal(prc, param, printf=True):
    en, ex, sl, win = param['entry'], param['exit'], param['stoploss'], param['win']
    # make the pairs trade matrix initialized to zero
    # we store trade for com1, com2 trade is just the opposite
    pairs_list = find_pairs_list(prc.columns)
    ptrade,pairs_mean,pairs_std,pairs_beta = {},{},{},{}
    pm,ps,pb = {},{},{}
    for com1, com2 in pairs_list:
        ptrade[(com1,com2)] = (prc[com1]*0).astype(int)
        pm[(com1,com2)] = (prc[com1]*0)
        ps[(com1,com2)] = (prc[com1]*0)
        pb[(com1,com2)] = (prc[com1]*0)
        pairs_mean[(com1, com2)] = 0
        pairs_std[(com1, com2)] = 0
        pairs_beta[(com1,com2)] = 0
    init_flag = False
    for t in range(win,len(prc.index)-2):
        # prediction for t+1 in this step
        tt = prc.index[t]
        ttn = prc.index[t+1]
```

```
        for item in pairs_list:
            com1,com2 = item
            # default carry last position
            ptrade[item][ttn] = ptrade[item][tt]
            try:
                pm[item][ttn] = pairs_mean[item]
                ps[item][ttn] = pairs_std[item]
                pb[item][ttn] = pairs_beta[item]
            except:
                pass
            if tt.month!=ttn.month: # if month end
                init_flag = True
                t_idx = prc.index[t+1]           # last day before prediction + 1
                m3_t_idx = prc.index[t-win]     # start day of last 60 days
                # find distance and std of distance: based on past w days data
                # first close the existing pair
                ptrade[item][ttn] = 0
                # construct condition for next month
                # get the beta
                x = prc[com1][m3_t_idx:t_idx].dropna()
                y = prc[com2][m3_t_idx:t_idx].dropna()
                x = x.reindex(y.index).dropna()
                y = y.reindex(x.index)
                mod = ols(x=x,y=y,intercept=True)
                # this is the ratio of com1 for 1 unit of com2
                pairs_beta[(com1,com2)] = mod.beta['x']
                pairs_mean[(com1,com2)] = mod.beta['intercept']
                pairs_std[(com1,com2)] = mod.resid.std()
            elif init_flag:
                # check exit rule
                if ptrade[item][tt]!=0:
                     if pairs_converged(item, ptrade, prc, pairs_mean,
                                        pairs_std, tt, ex, pairs_beta[item]) \
                        or pairs_diverged(item, ptrade, prc, pairs_mean,
                                          pairs_std, tt, sl, pairs_beta[item]):
                        ptrade[item][ttn] = 0
                # check entry rules
                elif (prc[com2][tt]-pairs_beta[item]*prc[com1][tt])>\
                        (pairs_mean[item]+en*pairs_std[item]):
                    # buy cheap and sell expensive
                    ptrade[item][tt] = -1
                elif (prc[com2][tt]-pairs_beta[item]*prc[com1][tt])<\
```

```
                    (pairs_mean[item]-en*pairs_std[item]):
                # buy cheap and sell expensive
                ptrade[item][tt] = 1
    # construct the final signal dataframe
    sig = prc*0.
    for item in ptrade.keys():
        com1,com2=item
        sig[com1] -= ptrade[item]*pb[item]
        sig[com2] += ptrade[item]
        if printf:
            DataFrame({
                'p1-b*p2': prc[com2]-pb[item]*prc[com1],
                'mean': pm[item],
                'mean+1*std': pm[item]+1*ps[item],
                'mean-1*std': pm[item]-1*ps[item],
            }).plot(title=item,grid=True)
    if printf:
        DataFrame(ptrade).plot(style='.-', grid=True)
        DataFrame(pb).plot(grid=True)
    return sig
```

**Evaluation of pairs via Cointegration**: The basic mechanism of the trading is shown in fig.20. The top plot shows the different levels and standard deviation limits for the 10 day moving window. The lower plots shows a particular 10 day period in detail where the stationary linear combination of the prices of two assets oscillates around the mean and results in handsome trades.

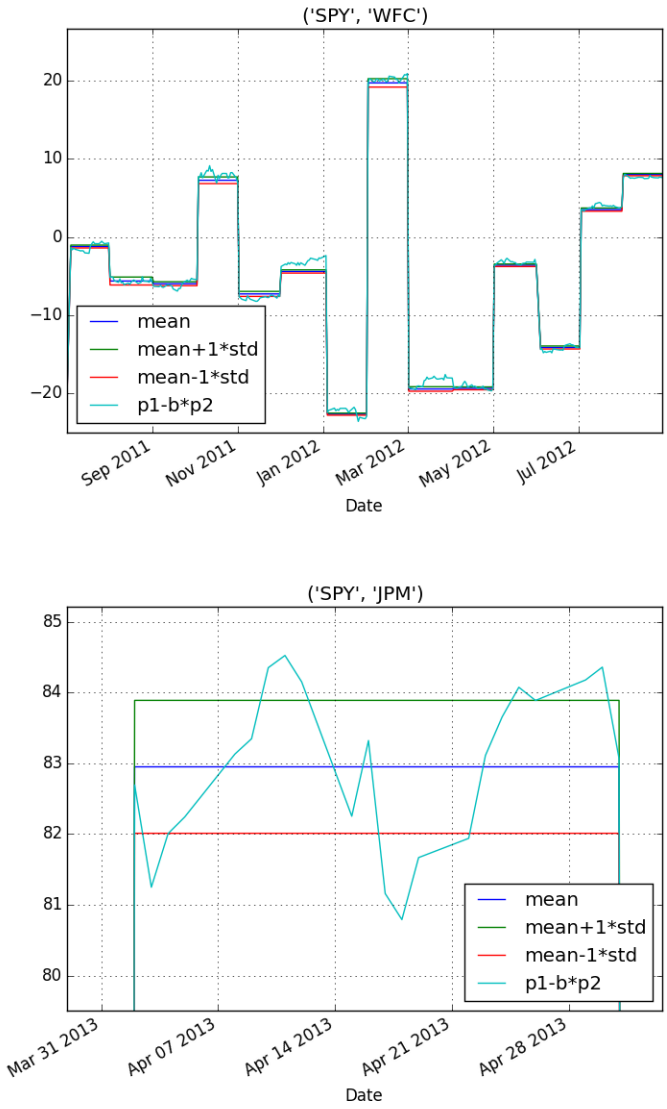


Figure 20: Entry exit rules illustrated for pairs trading via Cointegration. Moving window of 10 days used.

**Qualitative performance** - We start with a moving window of $L = 10$ days, entry of 1 standard deviation from mean and exit at -1 standard deviation from mean. We put a stop-loss at 10 standard deviation from the mean. We see in fig.22 that the rule is able to time the entry exits profitably. There is a deep drawdown but it recovers and turns out to be a very good strategy. The moving window beta for the three pairs are also stable as seen from fig.21, resulting in good performance.

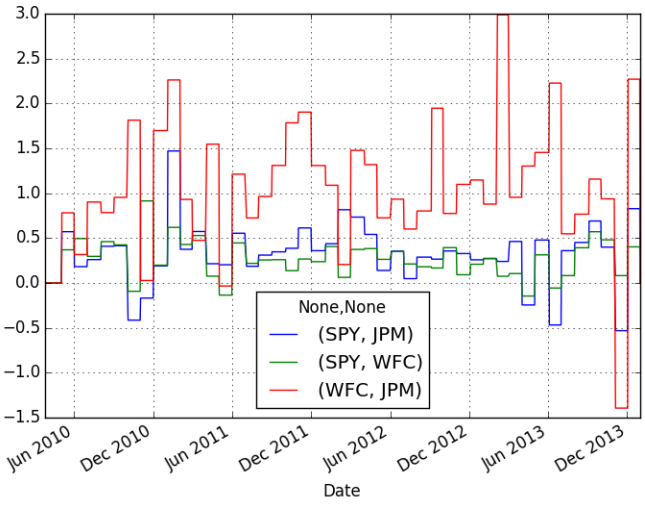


Figure 21: Moving window beta for the the three pairs.

**Quantitative performance** - To quantify the performance let's look at Sharpe, Treynor, Sortino, Calmar and Information ratios (all annualized). We also look at the PNL with and without transaction cost of 10 basis points. Risk free rate is 3%.

| | 0bps | 10bps |
|---|---|---|
| IR | 1.275598 | 0.7114834 |
| PNL | 4.478921 | 1.845663 |
| Sortino | 1.849087 | 1.002543 |
| Treynor | -0.2105101 | -0.0998686 |
| calmar | 4.900844 | 0.4907142 |
| sharpe | 1.227119 | 0.6577468 |

Table 7: Performance numbers (annualized) for Cointegration based pairs for 0 and 10 bps trading cost.

Table 7 shows the various ratios. We see that even after applying a trading cost of 10 bps the performance is pretty good. The net Sharpe ratio is 0.66, while IR is 0.71. Other ratios are reasonable as well. Dollar 3 invested in the strategy (as we start with one dollar for each pair) had yielded a profit of $1.84 in the time period of the analysis, but with high volatility. Clearly this is one of the better strategies.

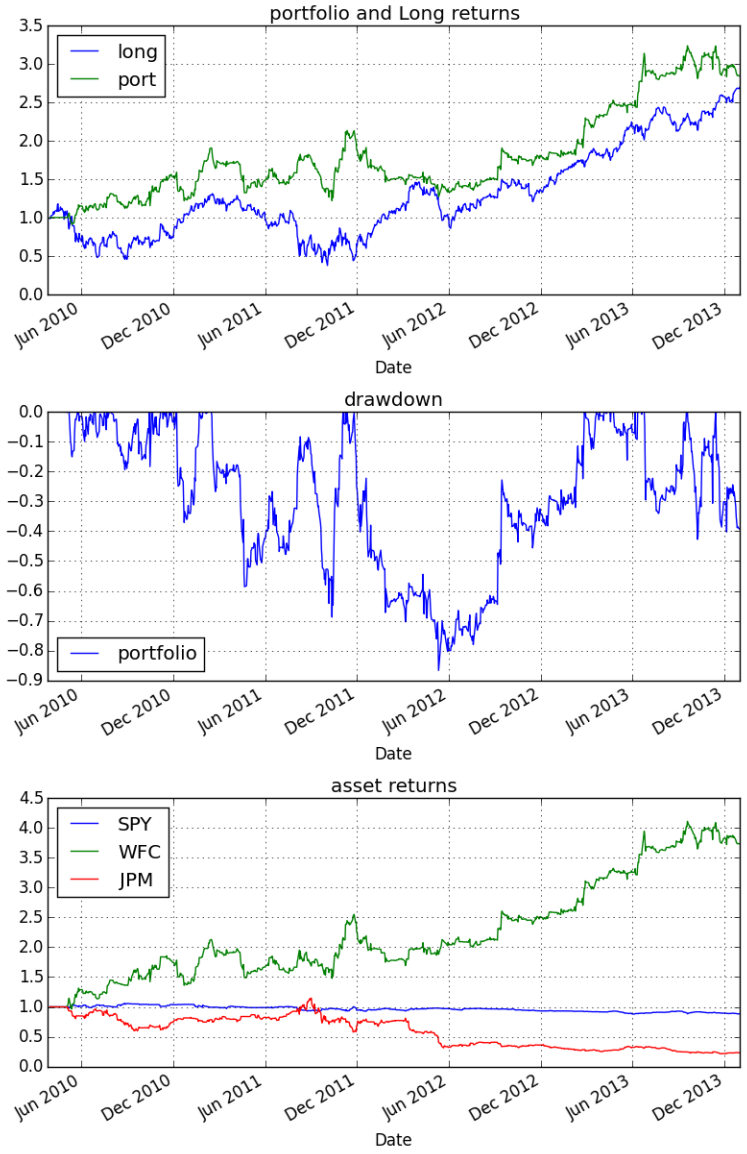


Figure 22: Equity curve analysis for pairs based on cointegration backtest with trading cost 10 bps

**Optimal in-sample values** - To get the optimal value of trading window entry and exit we run two grid searches and produce the fig.23 using the following code:

```
# optimization
param={
    #'win': np.arange(10,101,10),
    'en': np.arange(0.5,3,0.25),
    'ex': np.arange(-1,1,0.25),
```

```
}
hm = {}
for i in param[param.keys()[0]]:
    s = {}
    for j in param[param.keys()[1]]:
        print i,j
        s[j] = run_bt({'entry':i, 'exit':j, 'stoploss':5, 'win':10},
                      calc_signal, tc=0.001,
                      alloc='pairs')[
            'stats']['sharpe']['portfolio']
    hm[i] = Series(s)
hm = DataFrame(hm)
hm.columns.name=param.keys()[0]
hm.index.name=param.keys()[1]
heat_map(hm, title='heatmap of performance sharpe; Q3b Cointegration Pair rule')
```

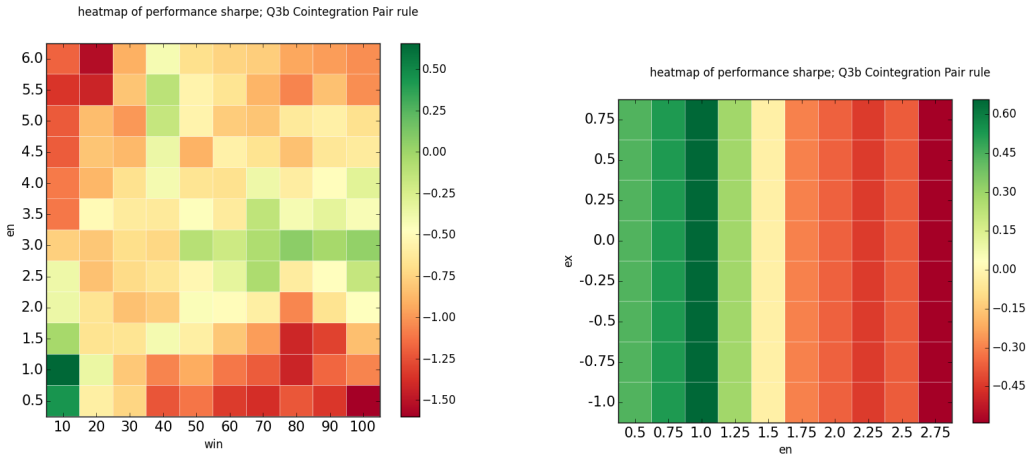


Figure 23: Heatmap for Cointegration based pairs backtest with trading cost 10 bps. Left: entry vs moving window, Right: entry vs exit.

The heat map plots Sharpe Ratio numbers. We see that the performance is pretty good for moving window of 10 days. A optimal and stable region looks like **(10,1,-1)** for moving window, entry and exit. We choose it as the optimal signal for later comparison. This turned out to be a significant improvement over the distance method in part a of this problem.

## Problem 4

This problem asks to fit a trivariate model based on cointegration. We are going to use a moving window of $w$ days and come up with the strongest cointegration relationship that exist at that point of time. This implies that we do not necessarily test for the statistical strength of the relationship, but rather simply trade on the existing 'strongest' cointegrated relationship. We present a sample calculation here to motivate the procedure.

Let's start with the trying to fit a VAR model on the 3 dimensional data (primarily to calculate the the significant lags we should feed the johansen's test) as follows:

```
In[66]: from stats242.HW2.bt import *
df = get_data().ix[140:200,:]
# understanding via VAR model
from statsmodels.tsa.api import VAR
Vmodel = VAR(df.values)
results = Vmodel.fit(1)
print results.summary()
#results.plot()
#results.plot_acorr()
Vmodel.select_order(5)
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 24, Jul, 2015
Time:                     18:40:03
--------------------------------------------------------------------
No. of Equations:         3.00000    BIC:                   -4.41249
Nobs:                     59.0000    HQIC:                  -4.67009
Log likelihood:          -96.5184    FPE:                  0.00795134
AIC:                     -4.83504    Det(Omega_mle):       0.00653092
--------------------------------------------------------------------
Results for equation y1
====================================================================
           coefficient       std. error           t-stat            prob
--------------------------------------------------------------------
const         6.918101        10.061377            0.688           0.495
L1.y1         0.967922         0.140332            6.897           0.000
L1.y2         0.239760         0.219747            1.091           0.280
L1.y3        -0.260910         0.141818           -1.840           0.071
====================================================================
```

```
Results for equation y2
===============================================================================
           coefficient       std. error           t-stat            prob
-------------------------------------------------------------------------------
const       -14.432285         5.967556           -2.418           0.019
L1.y1         0.247504         0.083233            2.974           0.004
L1.y2         0.751393         0.130335            5.765           0.000
L1.y3        -0.187841         0.084114           -2.233           0.030
===============================================================================

Results for equation y3
===============================================================================
           coefficient       std. error           t-stat            prob
-------------------------------------------------------------------------------
const       -15.010027         7.415421           -2.024           0.048
L1.y1         0.263103         0.103428            2.544           0.014
L1.y2        -0.114468         0.161958           -0.707           0.483
L1.y3         0.682990         0.104522            6.534           0.000
===============================================================================
Correlation matrix of residuals
            y1        y2        y3
y1    1.000000  0.648955  0.651216
y2    0.648955  1.000000  0.830016
y3    0.651216  0.830016  1.000000

                 VAR Order Selection
=====================================================
           aic          bic          fpe         hqic
-----------------------------------------------------
0      -0.6059      -0.4964       0.5456      -0.5635
1      -5.144*      -4.706*    0.005836*      -4.975*
2       -5.070       -4.303     0.006310       -4.773
3       -4.955       -3.860     0.007136       -4.531
4       -4.818       -3.395     0.008303       -4.268
5       -4.691       -2.939     0.009667       -4.014
=====================================================
* Minimum

e,v = np.linalg.eig(results.coefs-np.eye(3))
In[67]: e
Out[67]:
```

```
array([[-0.42261330+0.j        , -0.08754084+0.05569765j,
        -0.08754084-0.05569765j]])
```

We notice that lag 1 VAR model is sufficient. This fits the following model to the data:

$$\begin{pmatrix} p_{1t} \\ p_{2t} \\ p_{3t} \end{pmatrix} = \begin{pmatrix} 6.92 \\ -14.43 \\ -15.01 \end{pmatrix} + \begin{pmatrix} 0.97 & 0.24 & -0.26 \\ 0.25 & 0.75 & -0.19 \\ 0.26 & -0.11 & 0.68 \end{pmatrix} \begin{pmatrix} p_{1,t-1} \\ p_{2,t-1} \\ p_{3,t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \end{pmatrix}$$

where the correlation and variance of the error term can be seen in the code and output above. The error corrected form becomes:

$$\begin{pmatrix} \Delta p_{1t} \\ \Delta p_{2t} \\ \Delta p_{3t} \end{pmatrix} = \begin{pmatrix} 14.16 \\ 6.01 \\ 6.28 \end{pmatrix} + \begin{pmatrix} -0.03 & 0.24 & -0.26 \\ 0.25 & -0.25 & -0.19 \\ 0.26 & -0.11 & -0.32 \end{pmatrix} \begin{pmatrix} p_{1,t-1} \\ p_{2,t-1} \\ p_{3,t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \end{pmatrix}$$

The real part of the eigen-values of the above coefficient matrix are -0.42, -0.09 and -0.09. We notice that two of the eigen values are close to 0, weakly suggesting deficit in the rank and hence presence of cointegration. This is by no means a certain way of looking at it so we are going to test this out using Johensen's method which does all the calculations for us. We feed the Johansen test with a lag of 1 determined from the VAR analysis. The code used makes some use of code in statsmodel sandbox branch, which is not available in current version of statsmodel library, but available on internet.

```
from statsmodels.tsa.vector_ar.var_model import VAR
from Echan.EChanBook2.johansen_test import coint_johansen
In[68]: def get_johansen(y, p):
    N, l = y.shape
    jres = coint_johansen(y, 0, p)
    trstat = jres.lr1                       # trace statistic
    tsignf = jres.cvt                       # critical values
    r = l
    for i in range(l):
        if trstat[i] > tsignf[i, 1]:     # 0: 90%  1:95% 2: 99%
            r = i + 1
    jres.r = r
    jres.evecr = jres.evec[:, :r]
    return jres

df = get_data().ix[140:200,:]
p=1 # lags to consider in the VAR model
jres=get_johansen(df,p)
print "There are ", jres.r, "cointegration vectors"

-------------------------------------------------
```

```
-->Trace Statistics
     Crit-90%  Crit-95%  Crit-99%  variable statistics
r=0   27.0669   29.7961   35.4628            37.875668
r=1   13.4294   15.4943   19.9349            13.300227
r=2    2.7055    3.8415    6.6349             2.563698
-------------------------------------------------
-->Eigen Statistics
     Crit-90%  Crit-95%  Crit-99%  variable statistics
r=0   18.8928   21.1314   25.8650            24.575440
r=1   12.2971   14.2639   18.5200            10.736529
r=2    2.7055    3.8415    6.6349             2.563698
-------------------------------------------------
-->eigenvectors
[[-1.33492996  0.63570671 -0.63761653]
 [ 2.18702629  0.44018382  1.03859207]
 [ 0.12768837 -1.42173292 -0.59973852]]
-------------------------------------------------
-->eigenvalues
[ 0.34538925  0.16898926  0.04323904]
-------------------------------------------------
There are  1 cointegration vectors

v1=jres.evecr[:,0]
In[77]: print v1/-v1[0]
[-1.          1.51759798  0.26096633]
```

The test yields that the rank of this matrix is 1 and there is only one cointegrating relationship which, after normalizing the first dimension is

$$-p_{1t} + 1.52p_{2t} + 0.26p_{3t}.$$

Figure 24 shows on the left the price series of the three stocks. On the right top it shows the plot of cointegrated relationship. the lower two plots correspond to the other two vectors and seem to indicate some trends in them. This, at least in principle, establishes the validity and procedure of finding cointegrating relationships in the a universe of assets. There are situations, practically in the simualation, when more than 1 eigenvectors become important. There are various ways to tackle it, what we did was to take the highest eigenvalue based vector and trade on the relationship based on that. We understand that, this might be an inefficient approach to choose pairs but for illustration it suffices.

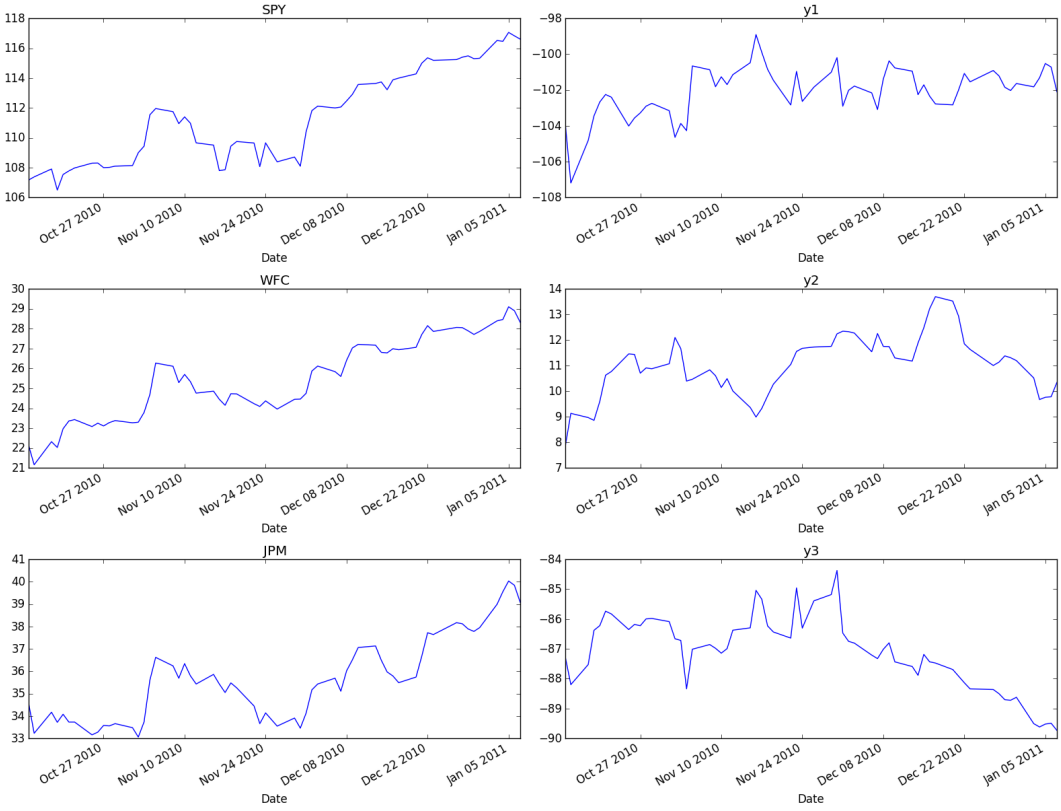


Figure 24: Heatmap for Cointegration based pairs backtest with trading cost 10 bps. Left: entry vs moving window, Right: entry vs exit.

We use the usual entry and exit rule and moving window parameter and optimize on that. We scale the positions such that the initial investment is \$1. We follow the same approach of estimating the relationship using a window of size $w$ at the last day of the month, like in problem 3, for comparability. The code for the simulation is as follows:

```
__author__ = 'manish'
from pandas import *
from stats242.HW2.bt import *
import numpy as np
from Echan.EChanBook2.johansen_test import coint_johansen
tol = 1e-4

############### Pairs: conintegratoin ##########################
# overwriting the default signal

def pairs_converged(ptrade, tt, pairs_beta, prc, pairs_mean, pairs_std,ex):
```

```
    if (ptrade.ix[tt,:].sum()-np.sum(pairs_beta))<tol:
        # looking for exit from top
        if ((prc.ix[tt,:]*pairs_beta).sum()<pairs_mean+ex*pairs_std):
            #print 'pc True'
            return True
    elif (ptrade.ix[tt,:].sum()+np.sum(pairs_beta))<tol:
        # looking for exit from botton
        if ((prc.ix[tt,:]*pairs_beta).sum()>pairs_mean-ex*pairs_std):
            #print 'pc True'
            return True
    #print 'pc False'


def pairs_diverged(ptrade, tt, pairs_beta, prc, pairs_mean, pairs_std,sl):
    if (ptrade.ix[tt,:].sum()-np.sum(pairs_beta))<tol:
        # looking for exit from top
        if ((prc.ix[tt,:]*pairs_beta).sum()>pairs_mean+sl*pairs_std):
            #print 'pd True'
            return True
    elif (ptrade.ix[tt,:].sum()+np.sum(pairs_beta))<tol:
        # looking for exit from bottonm
        if ((prc.ix[tt,:]*pairs_beta).sum()<pairs_mean-sl*pairs_std):
            #print 'pd True'
            return True
    #print 'pd False'

def get_johansen(y, p, print_on_console=False):
    N, l = y.shape
    jres = coint_johansen(y, 0, p, print_on_console)
    trstat = jres.lr1                       # trace statistic
    tsignf = jres.cvt                       # critical values
    r = l
    for i in range(l):
        if trstat[i] > tsignf[i, 1]:     # 0: 90%  1:95% 2: 99%
            r = i + 1
    jres.r = r
    jres.evecr = jres.evec[:, :r]
    return jres

def calc_signal(prc, param, showf=False):
    print param
    en, ex, sl, win = param['entry'], param['exit'], param['stoploss'], param['win']
```

```
    # make the pairs trade matrix initialized to zero
    pairs_mean,pairs_std,pairs_beta=0,0,[0,0,0] # state values
    # time series variables
    ptrade = prc.fillna(0)*0
    pb = prc.fillna(0)*0
    pm,ps = {},{}

    init_flag = False
    for t in range(win,len(prc.index)-2):
        # prediction for t+1 in this step
        tt = prc.index[t]
        ttn = prc.index[t+1]
        #print tt
        # default carry last position
        ptrade.loc[ttn,:] = ptrade.ix[tt,:]
        pm[ttn] = pairs_mean
        ps[ttn] = pairs_std
        pb.loc[ttn,:] = pairs_beta
        if tt.month!=ttn.month: # if month end
            init_flag = True
            t_idx = prc.index[t+1]           # last day before prediction + 1
            m3_t_idx = prc.index[t-win]     # start day of last 60 days
            # find distance and std of distance: based on past w days data
            # first close the existing pair
            ptrade.loc[ttn,:] = 0
            # construct condition for next month
            # get the beta - always assume VAR(1) and take heighest eigen vector
            jres=get_johansen(prc.ix[m3_t_idx:t_idx,:],1)
            biggest_idx = np.where(jres.eig==max(jres.eig))[0][0]
            eigv = jres.evec[:,biggest_idx]
            eigv = eigv/np.sum(np.abs(eigv))
            resid = (prc.ix[m3_t_idx:t_idx,:]*eigv).sum(axis=1)
            pairs_beta = eigv
            pairs_mean = resid.mean()
            pairs_std = resid.std()
            #print pairs_mean, pairs_std
        elif init_flag:
            # check exit rule
            #print 'exit rule ',ptrade.ix[tt,:].apply(np.abs).sum()
            if ptrade.ix[tt,:].apply(np.abs).sum()>0:
                 if pairs_converged(ptrade, tt, pairs_beta, prc, pairs_mean, pairs_std,ex) \
                    or pairs_diverged(ptrade, tt, pairs_beta, prc, pairs_mean, pairs_std,sl):
```

```
                ptrade.loc[ttn,:] = 0
            # check entry rules
            elif ((prc.ix[tt,:]*pairs_beta).sum()>pairs_mean+en*pairs_std):
                ptrade.loc[ttn,:] = pairs_beta
            elif ((prc.ix[tt,:]*pairs_beta).sum()<pairs_mean-en*pairs_std):
                ptrade.loc[ttn,:] = -pairs_beta
            #print 'top en rule ',((prc.ix[tt,:]*pairs_beta).sum()>pairs_mean+en*pairs_std)
            #print 'bottom en rule ',((prc.ix[tt,:]*pairs_beta).sum()<pairs_mean-en*pairs_std)
    # construct the final signal dataframe
    pm = Series(pm).reindex(prc.index)
    ps = Series(ps).reindex(prc.index)
    sig = ptrade/ptrade.apply(np.abs).sum(axis=1)
    sig = sig.replace([-np.inf,np.inf],np.nan).fillna(0)
    if showf:
        DataFrame({
            'p1-b1*p2-b3*p3': (prc*pb).sum(axis=1)-pm,
            'mean+2*std': 2*ps,
            'mean-2*std': -2*ps,
        }).plot(title='Cointegrated Pairs',grid=True)
        ptrade.plot(style='.-', grid=True, title='ptrade')
        sig.plot(style='.-', grid=True, title='signal')
        pb.plot(grid=True, title='beta')
    return sig
```

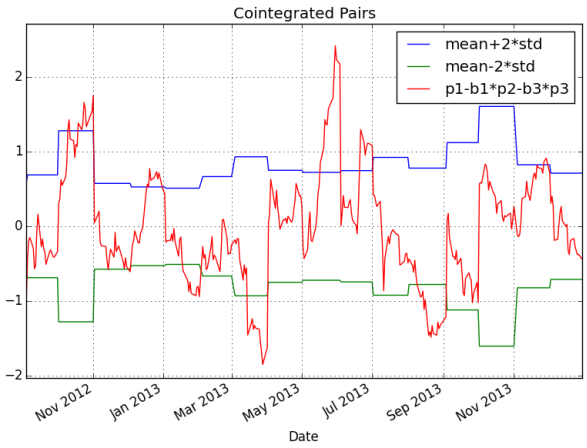


Figure 25: Entry exit rules illustrated for pairs trading via Cointegration. Moving window of 100 days used, updated every 30 days. The series is demeaned to zero.

**Evaluation of Cointegration trade**: The basic mechanism of the trading is shown in fig.25. The plot shows The upper and lower 2 standard deviation limits and the conintegrated series

(adjusted for levels when the monthly switch happens). We can see the mean reverting property of the series quite well which leads to handsome results. We can also see the changing weights on the three assets in fig.26. We see that the strategy is quite dynamic.

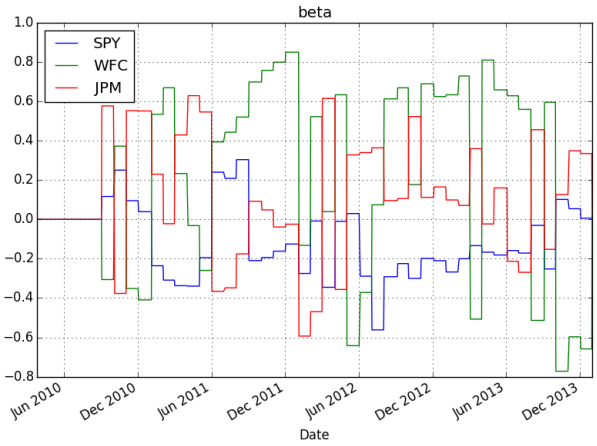


Figure 26: Changing weights on the three assets in the cointegrated vector. This vector is based on the highest eigen value from the Johansen's method.

**Qualitative performance** - We start with a moving window of $L = 100$ days, entry of 2 standard deviation from mean and exit at 1.5 standard deviation from mean. We put a stop-loss at 10 standard deviation from the mean, which has no effect. We see in fig.27 that the rule is able to time the entry exits profitably. The drawdowns are shallow, it recovers and turns out to be a very good strategy. We see that most of the performance comes from WFC and JPM. This is a the best of the all strategies we have tried in terms of drawdowns, shown by high Calmar number next.

**Quantitative performance** - To quantify the performance let's look at Sharpe, Treynor, Sortino, Calmar and Information ratios (all annualized). We also look at the PNL with and without transaction cost of 10 basis points. Risk free rate is 3%.

| | 0bps | 10bps |
|---|---|---|
| IR | 1.294162 | 0.7955172 |
| PNL | 0.446462 | 0.3021917 |
| Sortino | 1.320611 | 0.8579062 |
| Treynor | -0.1211976 | -0.06749025 |
| calmar | 4.795547 | 1.649113 |
| sharpe | 1.208128 | 0.7077495 |

Table 8: Performance numbers (annualized) for Cointegration based pairs for 0 and 10 bps trading cost.

Table 8 shows the various ratios. We see that even after applying a trading cost of 10 bps the performance is pretty good. The net Sharpe ratio is 0.71, while IR is 0.80. Other ratios are very reasonable as well, particularly Calmar, which is high due to low drawdowns. Dollar 1 invested in the strategy (as we start with one dollar for each pair) had yielded a profit of $0.30 in the time period of the analysis, but with very low volatility. Clearly this is one of the better strategies in terms of Sharpe ratio as well.

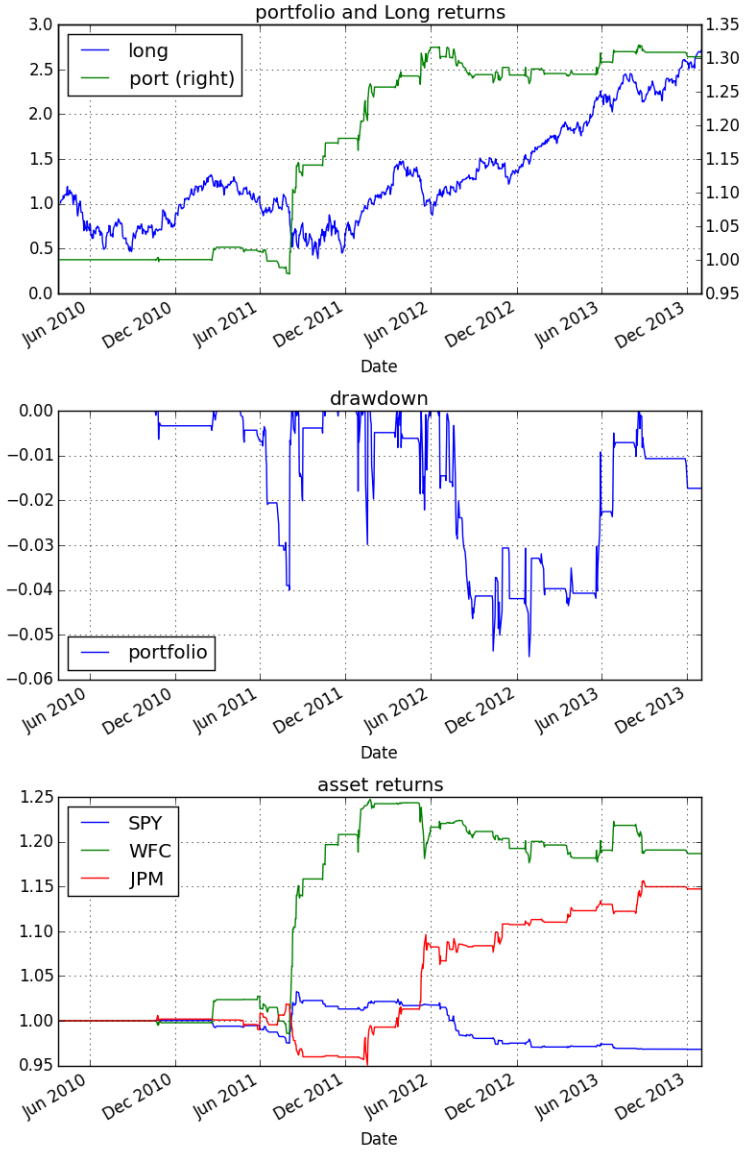


Figure 27: Equity curve analysis for pairs based on cointegration backtest with trading cost 10 bps

**Optimal in-sample values** - To get the optimal value of trading window entry and exit we run two grid searches and produce the fig.28 using the following code:

```
#optimization
param={
    #'win': np.arange(10,101,10),
    'en': np.arange(0.5,4,0.25),
    'ex': np.arange(2,4,0.25),
}
hm = {}
for i in param[param.keys()[0]]:
    s = {}
    for j in param[param.keys()[1]]:
        print i,j
        s[j] = run_bt({'entry':i, 'exit':j, 'stoploss':10, 'win':100},
                      calc_signal, tc=0.001,
                      alloc='pairs')[
            'stats']['sharpe']['portfolio']
    hm[i] = Series(s)
hm = DataFrame(hm)
hm.columns.name=param.keys()[0]
hm.index.name=param.keys()[1]
heat_map(hm, title='heatmap of performance sharpe; Q4 Cointegration rule')
```

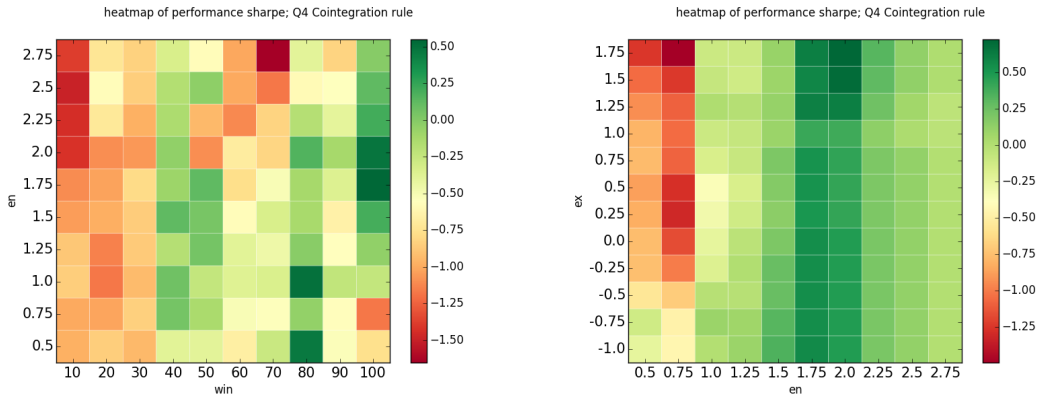


Figure 28: Heatmap for Cointegration based pairs backtest with trading cost 10 bps. Left: entry vs moving window, Right: entry vs exit.

The heat map plots Sharpe Ratio numbers. We see that the performance is pretty good for moving window of 100 days. A optimal and stable region looks like **(100,2,1.5)** for moving window, entry and exit. We choose it as the optimal signal for later comparison. This turned out to be a

significant improvement over the distance method as well as bivariate pairs trading strategy based on OLS.

## Problem 5

**Univariate, bivariate and tri-variate strategies**: In this document we investigated 8 strategies in detail - 5 on technical rules and the 3 on pairs trading. Figure 29 shows the performance of all the strategies at a glance. The period of analysis was from 2010 to 2013.

| strategy | Sharpe | IR | Calmar | Treynor | Sortino | PNL |
|---|---|---|---|---|---|---|
| Rule1 | 0.26 | 0.07 | 0.39 | 0.01 | 0.26 | 0.26 |
| Rule2 | 0.55 | 0.41 | 0.77 | 0.03 | 0.77 | 0.56 |
| Rule3 | 0.08 | -0.08 | 0.23 | 0 | 0.09 | 0.15 |
| Rule4 | 0.14 | -0.02 | 0.23 | 0.01 | 0.18 | 0.22 |
| Decomposition | 0.7 | 1.1 | 1.62 | -0.02 | 1 | 0.6 |
| Pair Distance | -0.08 | 0 | -0.03 | 0.01 | -0.09 | 0.04 |
| Pair Coin – bi | 0.66 | 0.71 | 0.49 | -0.1 | 1 | 0.61 |
| Pairs Coin – tri | 0.71 | 0.8 | 1.65 | -0.07 | 0.86 | 0.3 |

Figure 29: In sample performance of the 8 different strategies.

Further We also see that univariate strategies were easy to construct and code as they stood alone but resulted in medium to low performance. Bivariate and trivariate strategies which use the inter-relationships of the assets tend to do better because of the use of these inter-asset relationships. But, on the flip side, they are more complicated to implement as reflected in the amount of code needed to implement them.

**Reasoning behind the results**: The primary and most significant metric is Sharpe ratio which measures the returns per unit risk taken. Under this criteria - decomposition method comes out the best in the technical rules category and trivariate Pairs based on conintegration comes out the best for pairs.These two hold good standing under other ratios as well. Particularly important is their high Calmar ratio because of their very low drawdowns. The sophistication which went into these strategies goes on to show in their performance. These are results along expected lines as documented in literature.

**Trading recommendations**: In order to recommend a strategy let us look at the out of sample performance of these strategies on the data we kept aside at the start of the exercise. The results are shown in fig.30 and table 31 using the following script:

```
__author__ = 'manish'
from stats242.HW2.bt import *
from stats242.HW2.q1 import calc_signal1,\
    calc_signal2,calc_signal3,calc_signal4
from stats242.HW2.q2 import calc_signal as calc_signal5
```

```
from stats242.HW2.q3a import calc_signal as calc_signal6
from stats242.HW2.q3b import calc_signal as calc_signal7
from stats242.HW2.q4 import calc_signal as calc_signal8
# collecting all the results based on out of sample data

stD = '2014/01/01'
enD = '2015/03/25'
bt = {}
bt[1] = run_bt({'L':10, 'w':1.04}, calc_signal1,
tc=0.001, showf=False, alloc='equal_initial',sd=stD, ed=enD)
bt[2] = run_bt({'s':4, 'l':10}, calc_signal2,
tc=0.001, showf=False, alloc='equal_initial',sd=stD, ed=enD)
bt[3] = run_bt({'L':6}, calc_signal3, tc=0.001,
showf=False, alloc='equal_initial',sd=stD, ed=enD)
bt[4] = run_bt({'L':4}, calc_signal4, tc=0.001,
showf=False, alloc='equal_initial',sd=stD, ed=enD)
bt[5] = run_bt({'tc':0.001, 'win':35, 'm':8},
calc_signal5, tc=0.001, alloc='equal_initial', showf=False,sd=stD, ed=enD)
bt[6] = run_bt({'entry':5, 'exit':2, 'stoploss':10, 'win':10},
calc_signal6, tc=0.001,alloc='pairs', showf=False,sd=stD, ed=enD)
bt[7] = run_bt({'entry':1, 'exit':-1, 'stoploss':10, 'win':10},
calc_signal7, tc=0.001,alloc='pairs', showf=False,sd=stD, ed=enD)
bt[8] = run_bt({'entry':2, 'exit':1.5, 'stoploss':10, 'win':100},
calc_signal8, tc=0.001,alloc='pairs', showf=False,sd=stD, ed=enD)

stat = {}
for i in range(1,9):
    stat[i]= bt[i]['stats'].ix['portfolio']
stat = DataFrame(stat).T

pnl = DataFrame({i:bt[i]['pnl']['portfolio'] for i in range(1,9)})
pnl = 0.1*np.sqrt(250)*pnl/pnl.std() #normalize for comparision
pnl.cumsum().plot(title="Out of Sample performance", grid=True)
```

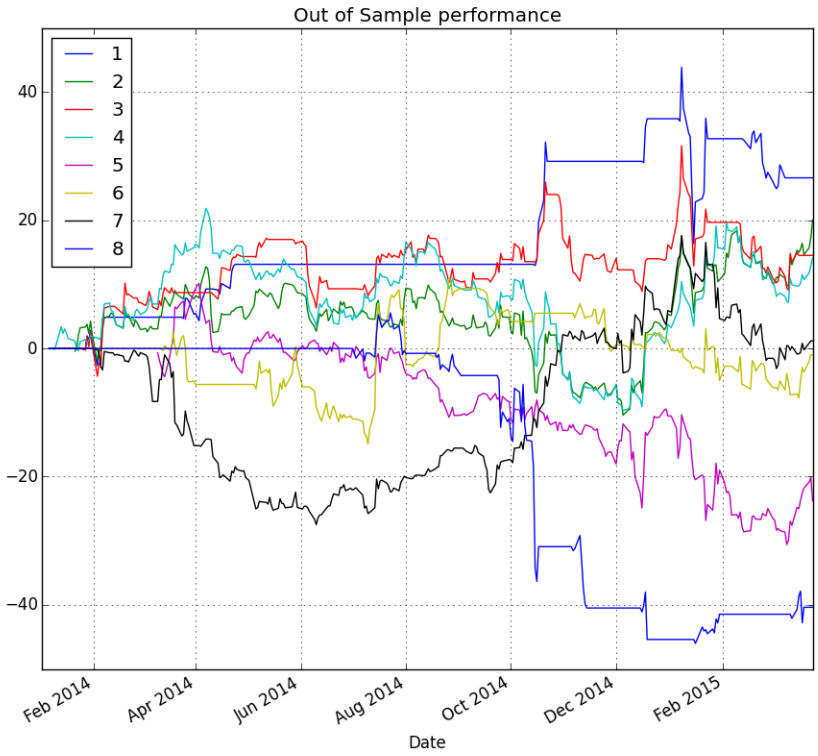


Figure 30: Out of sample performance of the 8 different strategies.

| strategy | Sharpe | IR | Calmar | Treynor | Sortino | PNL |
|---|---|---|---|---|---|---|
| Rule1 | 0.21 | -0.03 | 0.79 | 0.01 | 0.15 | 0.05 |
| Rule2 | 0.44 | 0.32 | 0.7 | 0.02 | 0.69 | 0.1 |
| Rule3 | 0.05 | -0.13 | 0.5 | 0 | 0.07 | 0.04 |
| Rule4 | 0.16 | 0.13 | 0.34 | 0.04 | 0.28 | 0.06 |
| Decomposition | -1.2 | -1.56 | -0.56 | -0.02 | -1.75 | -0.09 |
| Pair Distance | -0.24 | -0.31 | -0.14 | -0.02 | -0.32 | -0.01 |
| Pair Coin – bi | -0.06 | -0.07 | -0.1 | -0.03 | -0.08 | 0.01 |
| Pairs Coin – tri | -2.08 | -2.33 | -0.63 | -0.07 | -1.17 | -0.06 |

Figure 31: Out of sample performance of the 8 different strategies.

The main results is pretty clear - simpler models yield robust results. We see that in out of sample all the sophisticated models backfired. This may be due to overfitting in - sample and use of lot of parameters. Rule 2 stands out as the most simplest rule (minimum code as an evidence) and is robust out of sample as well. Though other rules were good in-sample they have not performed that well out of sample. Hence, **we will recommend rule 2** - Momentum with cross-over as the strategy to trade - for the given cross-section and for the market conditions similar to the case we have investigated. Other models should be investigated further to see if they can be generalized further to produce better out of sample results.