Trade Optimization

BY COREY HOFFSTEIN / ON AUGUST 27, 2018

Trade optimization is more technical topic than we usually cover in our published research. Therefore, this note will relies heavily on mathematical notation and assumes readers have a basic understanding of optimization. Accompanying the commentary is code written in Python, meant to provide concrete examples of how these ideas can be implemented. The Python code leverages the PuLP optimization library.

Readers not proficient in these areas may still benefit from reading the Introduction and evaluating the example outlined in Section 5.

Summary

- In practice, portfolio managers must account for the real-world implementation costs – both explicit (e.g. commission) and implicit (e.g. bid/ask spread and impact) associated with trading portfolios.
- Managers often implement trade paring constraints that may limit the number of allowed securities, the number of executed trades, the size of a trade, or the size of holdings. These constraints can turn a well-formed convex optimization into a discrete problem.
- In this note, we explore how to formulate trade optimization as a Mixed-Integer Linear Programming ("MILP") problem and implement an example in Python.

0. Initialize Python Libraries

```python
import pandas
import numpy

from pulp import *

import scipy.optimize
```

1. Introduction

In the context of portfolio construction, trade optimization is the process of managing the transactions necessary to move from one set of portfolio weights to another. These optimizations can play an important role both in the cases of rebalancing as well as in the case of a cash infusion or withdrawal. The reason for controlling these trades is to try to minimize the explicit (e.g. commission) and implicit (e.g. bid/ask spread and impact) costs associated with trading.

Two approaches are often taken to trade optimization:

1. Trading costs and constraints are explicitly considered within portfolio construction. For example, a portfolio optimization that seeks to maximize exposure to some alpha source may incorporate explicit measures of transaction costs or constrain the number of trades that are allowed to occur at any given rebalance.
2. Portfolio construction and trade optimization occur in a two step process. For example, a portfolio optimization may take place that creates the "ideal" portfolio, ignoring consideration of trading constraints and costs. Trade optimization would then occur as a second step, seeking to identify the trades that would move the current portfolio "as close as possible" to the target portfolio while minimizing costs or respecting trade constraints.

These two approaches will not necessarily arrive at the same result. At Newfound, we prefer the latter approach, as we believe it creates more transparency in portfolio construction. Combining trade optimization within portfolio optimization can also lead to complicated constraints, leading to infeasible optimizations. Furthermore, the separation of portfolio optimization and trade optimization allows us to target the same model portfolio across all strategy implementations, but vary when and how different portfolios trade depending upon account size and costs.

For example, a highly tactical strategy implemented as a pooled vehicle with a large asset base and penny-per-share commissions can likely afford to execute a much higher number of trades than an investor trying to implement the same strategy with $250,000 and $7.99 ticket charges. While implicit and explicit trading costs will create a fixed drag upon strategy returns, failing to implement each trade as dictated by a hypothetical model will create tracking error.

Ultimately, the goal is to minimize the fixed costs while staying within an acceptable distance (e.g. turnover distance or tracking error) of our target portfolio. Often, this goal is expressed by a portfolio manager with a number of semi-ad-hoc constraints or optimization targets. For example:

- **Asset Paring.** A constraint that specifies the minimum or maximum number of securities that can be held by the portfolio.
- **Trade Paring.** A constraint that specifies the minimum or maximum number of trades that may be executed.
- **Level Paring.** A constraint that establishes a minimum level threshold for securities (e.g. securities must be at least 1% of the portfolio) or trades (e.g. all trades must be larger than 0.5%).

Unfortunately, these constraints often turn the portfolio optimization problem from continuous to discrete, which makes the process of optimization more difficult.

2. The Discreteness Problem

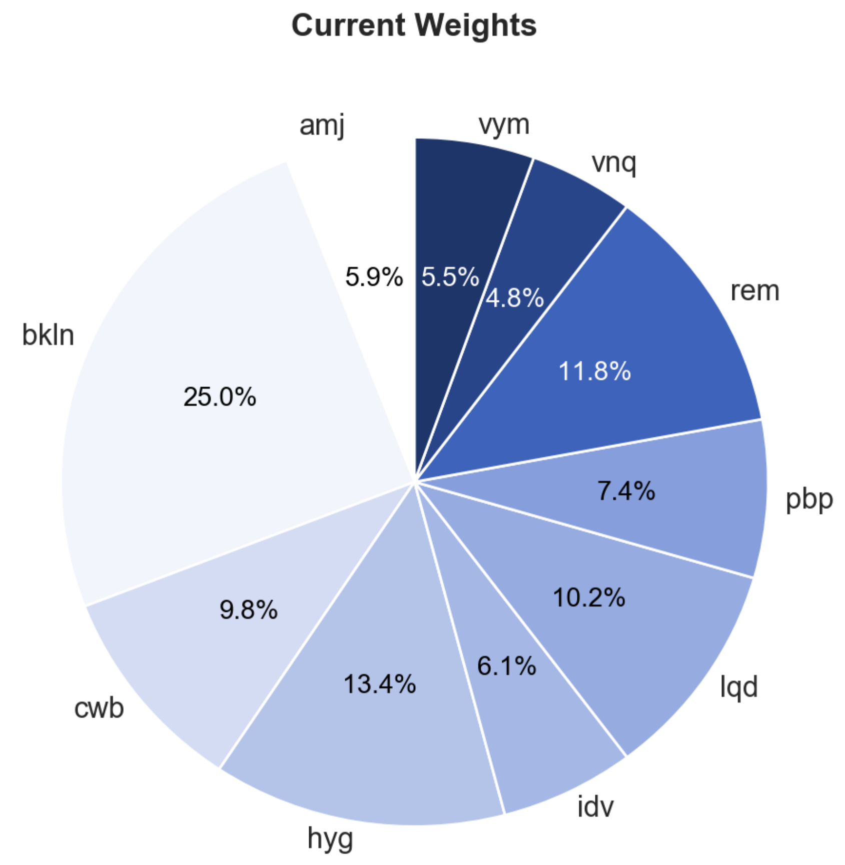
Consider the following simplified scenario. Given our current, drifted portfolio weights w_{old} and a new set of target model weights w_{target}, we want to minimize the number of trades we need to execute to bring our portfolio within some acceptable turnover threshold level, θ. We can define this as the optimization problem:

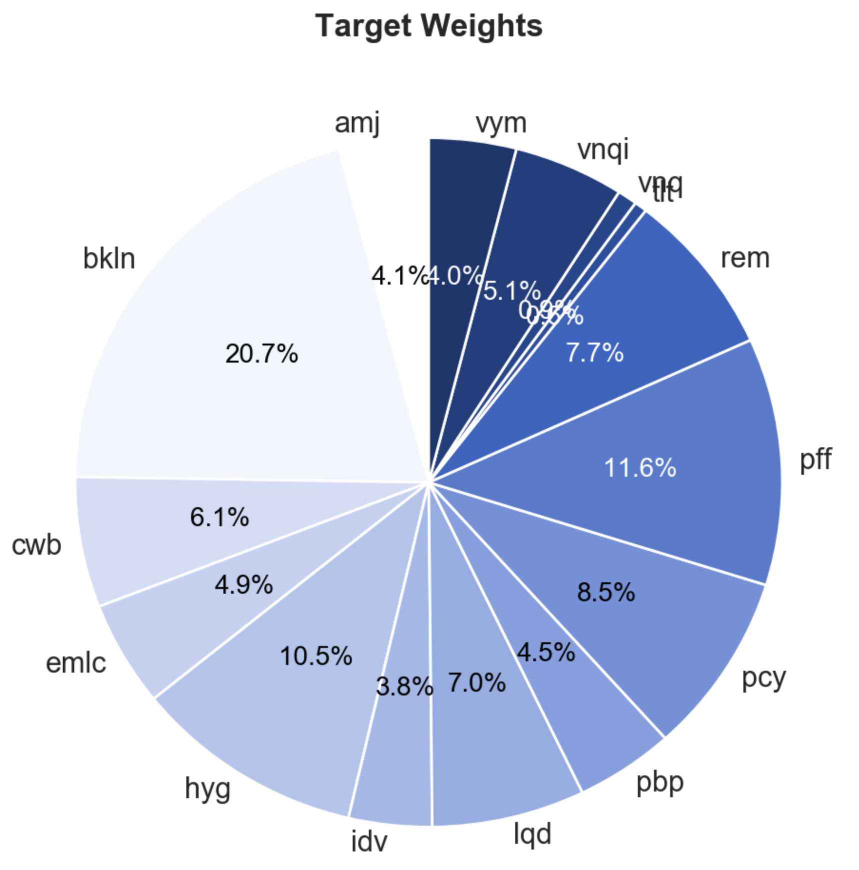
$$\begin{array}{ll}
\text{minimize} & \sum_i 1_{|t_i|} > 0 \\
\text{subject to} & \sum_i |w_{target,i} - (w_{old,i} + t_i)| \leq 2 * \theta \\
 & \sum_i t_i = 0 \\
\text{and} & t_i \geq -w_{old,i}
\end{array}$$

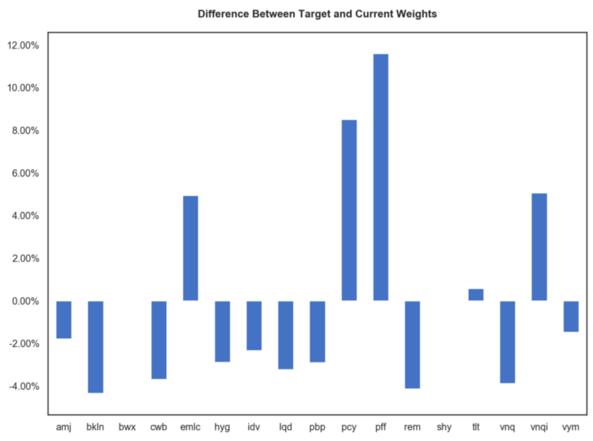
Unfortunately, as we will see below, simply trying to throw this problem into an off-the-shelf convex optimizer, as is, will lead to some potentially odd results. And we have not even introduced any complex paring constraints!

2.1 Example Data

```
# setup some sample data
tickers = "amj bkln bwx cwb emlc hyg idv lqd \
           pbp pcy pff rem shy tlt vnq vnqi vym".split()

w_target = pandas.Series([float(x) for x in "0.04095391 0.206519656 0 \
                      0.061190655 0.049414401 0.105442705 0.038080766 \
                      0.07004622 0.045115708 0.08508047 0.115974239 \
                      0.076953702 0 0.005797291 0.008955226 0.050530852 \
                      0.0399442".split()], index = tickers)

w_old = pandas.Series([float(x) for x in \
                   "0.058788745 0.25 0 0.098132817 \
                    0 0.134293993 0.06144967 0.102295438 \
                    0.074200473 0 0 0.118318536 0 0 \
                    0.04774768 0 0.054772649".split()], \
                      index = tickers)

n = len(tickers)

w_diff = w_target - w_old
```


Current Weights
amj
5.9%
vym
5.5%
vnq
4.8%
rem
11.8%
pbp
7.4%
lqd
10.2%
idv
6.1%
hyg
13.4%
cwb
9.8%
bkln
25.0%

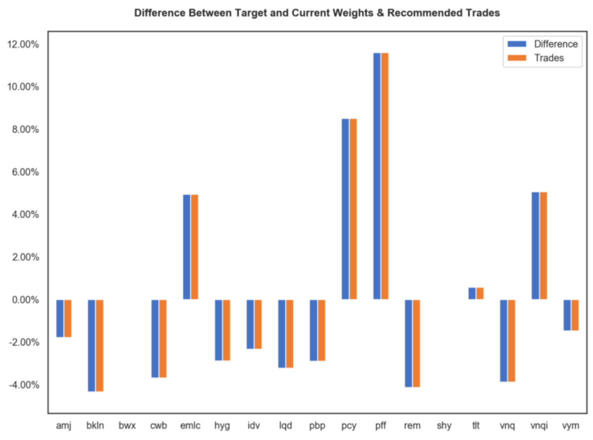

Target Weights
amj
4.1%
vym
4.0%
vnqi
5.1%
vnq
tlt
rem
7.7%
pff
11.6%
pcy
8.5%
pbp
4.5%
lqd
7.0%
idv
3.8%
hyg
10.5%
emlc
4.9%
cwb
6.1%
bkln
20.7%



2.2 Applying a Naive Convex Optimizer

The example below demonstrates the numerical issues associated with attempting to solve discrete problems with traditional convex optimizers. Using the portfolio and target weights established above, we run a naive optimization that seeks to minimize the number of trades necessary to bring our holdings within a 5% turnover threshold from the target weights.

```python
# Try a naive optimization with SLSQP

theta = 0.05
theta_hat = theta + w_diff.abs().sum() / 2.

def _fmin(t):
    return numpy.sum(numpy.abs(t) > 1e-8)

def _distance_constraint(t):
    return theta_hat - numpy.sum(numpy.abs(t)) / 2.

def _sums_to_zero(t):
    return numpy.sum(numpy.square(t))

t0 = w_diff.copy()

bounds = [(-w_old[i], 1) for i in range(0, n)]

result = scipy.optimize.fmin_slsqp(_fmin, t0, bounds = bounds, \
                                   eqcons = [_sums_to_zero], \
                                   ieqcons = [_distance_constraint], \
                                   disp = -1)

result =  pandas.Series(result, index = tickers)
```



Note that the trades we received are simply $w_{target} - w_{old}$, which was our initial guess for the optimization. The optimizer didn't optimize.

What's going on? Well, many off-the-shelf optimizers – such as the Sequential Least Squares Programming (SLSQP) approach applied here – will attempt to solve this problem by first estimating the gradient of the problem to decide which direction to move in search of the optimal solution. To achieve this numerically, small perturbations are made to the input vector and their influence on the resulting output is calculated.

In this case, small changes are unlikely to create an influence in the problem we are trying to minimize. Whether the trade is 5% or 5.0001% will have no influence on the *number* of trades executed. So the first derivative will appear to be zero and the optimizer will exit.

Fortunately, with a bit of elbow grease, we can turn this problem into a mixed integer linear programming problem ("MILP"), which have their own set of efficient optimization tools (in this article, we will use the PuLP library for the Python programming language). A MILP is a category of optimization problems that take the standard form:

$$\begin{array}{ll} \text{minimize} & c^T x + h^T y \\ \text{subject to} & Ax + Gy \le b \\ \text{and} & x \in \mathbb{Z}^n \end{array}$$

Here b is a vector and A and G are matrices. Don't worry too much about the form.

The important takeaway is that we need: (1) to express our minimization problem as a linear function and (2) express our constraints as a set of linear inequalities.

But first, for us to take advantage of linear programming tools, we need to eliminate our absolute values and indicator functions and somehow transform them into linear constraints.

3. Linear Programming Transformation Techniques

3.1 Absolute Values

Consider an optimization of the form:

$$\begin{array}{ll} \text{minimize} & \sum_i |x_i| \\ \text{subject to} & \ldots \end{array}$$

To get rid of the absolute value function, we can rewrite the function as a minimization of a new variable, ψ.

$$\begin{array}{ll} \text{minimize} & \sum_i \psi_i \\ \text{subject to} & \psi_i \ge x_i \\ & \psi_i \ge -x_i \\ \text{and} & \ldots \end{array}$$

The combination of constraints makes it such that $\psi_i \ge |x_i|$. When x_i is positive, ψ_i is constrained by the first constraint and when x_i is negative, it is constrained by the latter. Since the optimization seeks to minimize the sum of each ψ_i, and we know ψ_i will be positive, the optimizer will reduce ψ_i to equal $|x_i|$, which is it's minimum possible value.

Below is an example of this trick in action. Our goal is to minimize the absolute value of some variables x_i. We apply bounds on each x_i to allow the problem to converge on a solution.

```
lp_problem = LpProblem("Absolute Values", LpMinimize)

x_vars = []
psi_vars = []

bounds = [[1, 7], [-10, 0], [-9, -1], [-1, 5], [6, 9]]

print "Bounds for x: "
print pandas.DataFrame(bounds, columns = ["Left", "Right"])

for i in range(5):
    x_i = LpVariable("x_" + str(i), None, None)
    x_vars.append(x_i)

    psi_i = LpVariable("psi_i" + str(i), None, None)
    psi_vars.append(psi_i)

lp_problem += lpSum(psi_vars), "Objective"

for i in range(5):
    lp_problem += psi_vars[i] >= -x_vars[i]
    lp_problem += psi_vars[i] >= x_vars[i]

    lp_problem += x_vars[i] >= bounds[i][0]
    lp_problem += x_vars[i] <= bounds[i][1]

lp_problem.solve()

print "\nx variables"
print pandas.Series([x_i.value() for x_i in x_vars])

print "\npsi Variables (|x|):"
print pandas.Series([psi_i.value() for psi_i in psi_vars])
```

```
Bounds for x:
   Left  Right
0     1      7
1   -10      0
2    -9     -1
3    -1      5
4     6      9

x variables
```

```
0    1.0
1    0.0
2   -1.0
3    0.0
4    6.0
dtype: float64

psi Variables (|x|):
0    1.0
1    0.0
2    1.0
3    0.0
4    6.0
dtype: float64
```

3.2 Indicator Functions

Consider an optimization problem of the form:

$$\begin{aligned} \text{minimize} \quad & \sum_i 1_{x_i > 0} \\ \text{subject to} \quad & \ldots \end{aligned}$$

We can re-write this problem by introducing a new variable, y_i, and adding a set of linear constraints:

$$\begin{aligned} \text{minimize} \quad & \sum_i y_i \\ \text{subject to} \quad & x_i \leq A * y_i \\ & y_i \geq 0 \\ & y_i \leq 1 \\ & y_i \in \mathbb{Z} \\ \text{and} \quad & \ldots \end{aligned}$$

Note that the last three constraints, when taken together, tell us that $y_i \in \{0, 1\}$. The new variable A should be a large constant, bigger than any value of x_i. Let's assume $A = max(x) + 1$.

Let's first consider what happens when $x_i \leq 0$. In such a case, y_i can be set to zero without violating any constraints. When x_i is positive, however, for $x_i \leq A * y_i$ to be true, it must be the case that $y_i = 1$.

What prevents y_i from equalling 1 in the case where $x_i \leq 0$ is the goal of minimizing the sum of y_i, which will force y_i to be 0 whenever possible.

Below is a sample problem demonstrating this trick, similar to the example described in the prior section.

```python
lp_problem = LpProblem("Indicator Function", LpMinimize)

x_vars = []
y_vars = []

bounds = [[-4, 1], [-3, 5], [-6, 1], [1, 7], [-5, 9]]

A = 11

print "Bounds for x: "
print pandas.DataFrame(bounds, columns = ["Left", "Right"])

for i in range(5):
    x_i = LpVariable("x_" + str(i), None, None)
    x_vars.append(x_i)

    y_i = LpVariable("ind_" + str(i), 0, 1, LpInteger)
    y_vars.append(y_i)

lp_problem += lpSum(y_vars), "Objective"

for i in range(5):
    lp_problem += x_vars[i] >= bounds[i][0]
    lp_problem += x_vars[i] <= bounds[i][1]

    lp_problem += x_vars[i] <= A * y_vars[i]

lp_problem.solve()

print "\nx variables"
print pandas.Series([x_i.value() for x_i in x_vars])

print "\ny Variables (Indicator):"
print pandas.Series([y_i.value() for y_i in y_vars])
```

```
Bounds for x:
   Left  Right
0    -4      1
1    -3      5
2    -6      1
3     1      7
```

```
4    -5      9

x variables
0   -4.0
1   -3.0
2   -6.0
3    1.0
4   -5.0
dtype: float64

y Variables (Indicator):
0    0.0
1    0.0
2    0.0
3    1.0
4    0.0
dtype: float64
```

3.3 Tying the Tricks Together

A problem arises when we try to tie these two tricks together, as both tricks rely upon the minimization function itself. The ψ_i are dragged to the absolute value of x_i because we minimize over them. Similarly, y_i is dragged to zero when the indicator should be off because we are minimizing over it.

What happens, however, if we want to solve a problem of the form:

$$\begin{array}{ll} \text{minimize} & \sum_i 1_{|x_i|>0} \\ \text{subject to} & \ldots \end{array}$$

One way of trying to solve this problem is by using our tricks and then combining the objectives into a single sum.

$$\begin{array}{ll} \text{minimize} & \sum_i y_i + \psi_i \\ \text{subject to} & \psi_i \geq x_i \\ & \psi_i \geq -x_i \\ & x_i \leq A * y_i \\ & y_i \geq 0 \\ & y_i \leq 1 \\ & y_i \in \mathbb{Z} \\ \text{and} & .. \end{array}$$

By minimizing over the sum of both variables, ψ_i is forced towards $|x_i|$ and y_i is forced to zero when $\psi_i = 0$.

Below is an example demonstrating this solution, again similar to the examples discussed in prior sections.

```
lp_problem = LpProblem("Absolute Values", LpMinimize)

x_vars = []
psi_vars = []
y_vars = []

bounds = [[-7, 3], [7, 8], [5, 9], [1, 4], [-6, 2]]

A = 11

print "Bounds for x: "
print pandas.DataFrame(bounds, columns = ["Left", "Right"])

for i in range(5):
    x_i = LpVariable("x_" + str(i), None, None)
    x_vars.append(x_i)

    psi_i = LpVariable("psi_i" + str(i), None, None)
    psi_vars.append(psi_i)

    y_i = LpVariable("ind_" + str(i), 0, 1, LpInteger)
    y_vars.append(y_i)


lp_problem += lpSum(y_vars) + lpSum(psi_vars), "Objective"

for i in range(5):
    lp_problem += x_vars[i] >= bounds[i][0]
    lp_problem += x_vars[i] <= bounds[i][1]

for i in range(5):
    lp_problem += psi_vars[i] >= -x_vars[i]
    lp_problem += psi_vars[i] >= x_vars[i]

    lp_problem += psi_vars[i] <= A * y_vars[i]

lp_problem.solve()

print "\nx variables"
print pandas.Series([x_i.value() for x_i in x_vars])

print "\npsi Variables (|x|):"
print pandas.Series([psi_i.value() for psi_i in psi_vars])

print "\ny Variables (Indicator):"
print pandas.Series([y_i.value() for y_i in y_vars])
```

```
Bounds for x:
   Left  Right
0    -7      3
1     7      8
2     5      9
3     1      4
4    -6      2

x variables
0    0.0
1    7.0
2    5.0
3    1.0
4    0.0
dtype: float64

psi Variables (|x|):
0    0.0
1    7.0
2    5.0
3    1.0
4    0.0
dtype: float64

y Variables (Indicator):
0    0.0
1    1.0
2    1.0
3    1.0
4    0.0
dtype: float64
```

4. Building a Trade Minimization Model

Returning to our original problem,

$$\begin{aligned} \text{minimize} \quad & \sum_i 1_{|t_i|>0} \\ \text{subject to} \quad & \sum_i |w_{target,i} - (w_{old,i} + t_i)| \le 2 * \theta \\ & \sum_i t_i = 0 \\ \text{and} \quad & t_i \ge -w_{old,i} \end{aligned}$$

We can now use the tricks we have established above to re-write this problem as:

$$\begin{aligned} \text{minimize} \quad & \sum_i (\phi_i + \psi_i + y_i) \\ \text{subject to} \quad & \psi_i \ge t_i \\ & \psi_i \ge -t_i \\ & \psi_i \le A * y_i \\ & \phi_i \ge (w_{target,i} - (w_{old,i} + t_i)) \\ & \phi_i \ge -(w_{target,i} - (w_{old,i} + t_i)) \\ & \sum_i \phi_i \le 2 * \theta \\ & \sum_i t_i = 0 \\ \text{and} \quad & t_i \ge -w_{old,i} \end{aligned}$$

While there are a large number of constraints present, in reality there are just a few key steps going on. First, our key variable in question is t_i. We then use our absolute value trick to create $\psi_i = |t_i|$. Next, we use the indicator function trick to create y_i, which tells us whether each position is traded or not. Ultimately, this is the variable we are trying to minimize.

Next, we have to deal with our turnover constraint. Again, we invoke the absolute value trick to create ϕ_i, and replace our turnover constraint as a sum of ϕ's.

Et voila?

As it turns out, not quite.

Consider a simple two-asset portfolio. The current weights are [0.25, 0.75] and we want to get these weights within 0.05 of [0.5, 0.5] (using the L^1 norm – i.e. the sum of absolute values – as our definition of “distance”).

Let's consider the solution [0.475, 0.525]. At this point, $\phi = [0.025, 0.025]$ and $\psi = [0.225, 0.225]$. Is this solution "better" than [0.5, 0.5]? At [0.5, 0.5], $\phi = [0.0, 0.0]$ and $\psi = [0.25, 0.25]$. From the optimizer's viewpoint, these are equivalent solutions. Within this region, there are an infinite number of possible solutions.

Yet if we are willing to let some of our tricks "fail," we can find a solution. If we want to get as close as possible, we effectively want to minimize the sum of ψ's. The infinite solutions problem arises when we simultaneously try to minimize the sum of ψ's and ϕ's, which offset each other.

Do we actually need the values of ψ to be correct? As it turns out: no. All we really need is that ψ_i is positive when t_i is non-zero, which will then force y_i to be 1. By minimizing on y_i, ψ_i will still be forced to 0 when $t_i = 0$.

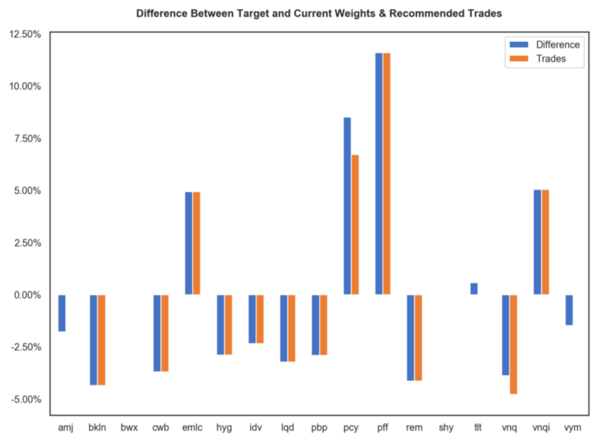
So if we simply remove ψ_i from the minimization, we will end up reducing the number of trades as far as possible and then reducing the distance to the target model as much as possible given that trade level.

$$
\begin{aligned}
\text{minimize} \quad & \sum_i (\phi_i + y_i) \\
\text{subject to} \quad & \psi_i \geq t_i \\
& \psi_i \geq -t_i \\
& \psi_i \leq A * y_i \\
& \phi_i \geq (w_{target,i} - (w_{old,i} + t_i)) \\
& \phi_i \geq -(w_{target,i} - (w_{old,i} + t_i)) \\
& \sum_i \phi_i \leq 2 * \theta \\
& \sum_i t_i = 0 \\
\text{and} \quad & t_i \geq -w_{old,i}
\end{aligned}
$$

As a side note, because the sum of ϕ's will at most equal 2 and the sum of y's can equal the number of assets in the portfolio, the optimizer will get more minimization bang for its buck by focusing on reducing the number of trades first before reducing the distance to the target model. This priority can be adjusted by multiplying ϕ_i by a sufficiently large scaler in our objective.

```python
theta = 0.05

trading_model = LpProblem("Trade Minimization Problem", LpMinimize)

t_vars = []
psi_vars = []
phi_vars = []
y_vars = []

A = 2

for i in range(n):
    t = LpVariable("t_" + str(i), -w_old[i], 1 - w_old[i])
    t_vars.append(t)

    psi = LpVariable("psi_" + str(i), None, None)
    psi_vars.append(psi)

    phi = LpVariable("phi_" + str(i), None, None)
    phi_vars.append(phi)

    y = LpVariable("y_" + str(i), 0, 1, LpInteger) #set y in {0, 1}
    y_vars.append(y)


# add our objective to minimize y, which is the number of trades
trading_model += lpSum(phi_vars) + lpSum(y_vars), "Objective"

for i in range(n):
    trading_model += psi_vars[i] >= -t_vars[i]
    trading_model += psi_vars[i] >= t_vars[i]
    trading_model += psi_vars[i] <= A * y_vars[i]

for i in range(n):
    trading_model += phi_vars[i] >= -(w_diff[i] - t_vars[i])
    trading_model += phi_vars[i] >= (w_diff[i] - t_vars[i])

# Make sure our trades sum to zero
trading_model += (lpSum(t_vars) == 0)

# Set our trade bounds
trading_model += (lpSum(phi_vars) / 2. <= theta)

trading_model.solve()

results = pandas.Series([t_i.value() for t_i in t_vars], index = tickers)

print "Number of trades: " + str(sum([y_i.value() for y_i in y_vars]))

print "Turnover distance: " + str((w_target - (w_old + results)).abs().sum() / 2.)
```

```
Number of trades: 12.0
Turnover distance: 0.032663284500000014
```



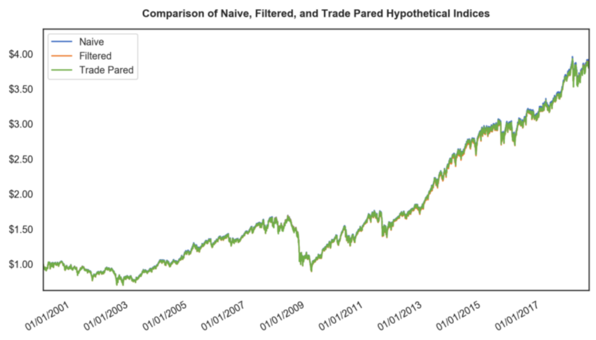
5. A Sector Rotation Example

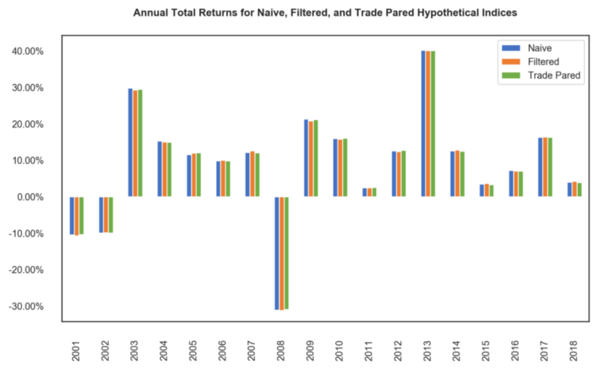
As an example of applying trade paring, we construct a sample sector rotation strategy. The investment universe consists of nine sector ETFs (XLB, XLE, XLF, XLI, XLK, XLU, XLV and XLY). The sectors are ranked by their 12-1 month total returns and the portfolio holds the four top-ranking ETFs in equal weight. To reduce timing luck, we apply a four-week tranching process.

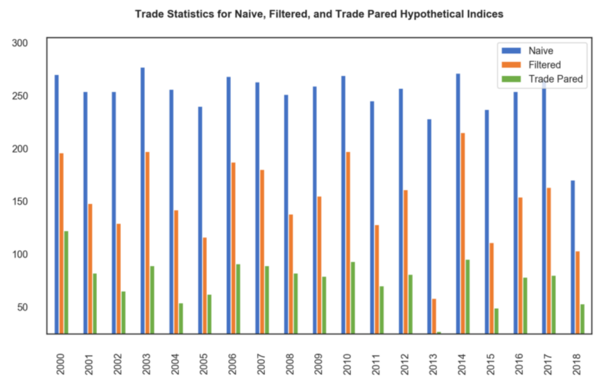
We construct three versions of the strategy.

- **Naive:** A version which rebalances back to hypothetical model weights on a weekly basis.
- **Filtered:** A version that rebalances back to hypothetical model weights when drifted portfolio weights exceed a 5% turnover distance from target weights.
- **Trade Pared:** A version that applies trade paring to rebalance back to within a 1% turnover distance from target weights when drifted weights exceed a 5% turnover distance from target weights.

The equity curves and per-year trade counts are plotted for each version below. Note that the equity curves do not account for any implicit or explicit trading costs.







Data Source: CSI. Calculations by Newfound Research. Past performance does not guarantee future results. All returns are hypothetical index returns. You cannot invest directly in an index and unmanaged index returns do not reflect any fees, expenses, sales charges, or trading expenses. Index returns include the reinvestment of dividends. No index is meant to measure any strategy that is or ever has been managed by Newfound Research. The indices were constructed by Newfound in August 2018 for purposes of this analysis and are therefore entirely backtested and not investment strategies that are currently managed and offered by Newfound.

For the reporting period covering full years (2001 – 2017), the trade filtering process alone reduced the average number of annual trades by 40.6% (from 255.7 to 151.7). The added trade paring process reduced the number of trades another 50.9% (from 151.7 to 74.5), for a total reduction of 70.9%.

6. Possible Extensions & Limitations

There are a number of extensions that can be made to this model, including:

- **Accounting for trading costs.** Instead of minimizing the number of trades, we could minimize the total cost of trading by multiplying each trade against an estimate of cost (including bid/ask spread, commission, and impact).
- **Forcing accuracy.** There may be positions for which more greater drift can be permitted and others where drift is less desired. This can be achieved by adding specific constraints to our ϕ_i variables.

Unfortunately, there are also a number of limitations. The first set is due to the fact we are formulating our optimization as a linear program. This means that quadratic constraints or objectives, such as tracking error constraints, are forbidden. The second set is due to the complexity of the optimization problem. While the problem may be technically solvable, problems containing a large number of securities and constraints may be time infeasible.

6.1 Non-Linear Constraints

In the former case, we can choose to move to a mixed integer quadratic programming framework. Or, we can also employ multi-step heuristic methods to find feasible, though potentially non-optimal, solutions.

For example, consider the case where we wish our optimized portfolio to fall within a certain tracking error constraint of our target portfolio. Prior to optimization, the marginal contribution to tracking error can be calculated for each asset and the total current tracking error can be calculated. A constraint can then be added such that the current tracking error minus the sum of weighted marginal contributions must be less than the tracking error target. After the optimization is complete, we can determine whether our solution meets the tracking error constraint.

If it does not, we can use our solution as our new w_{old}, re-calculate our tracking error and marginal contribution figures, and re-optimize. This iterative approach approximates a gradient descent approach.

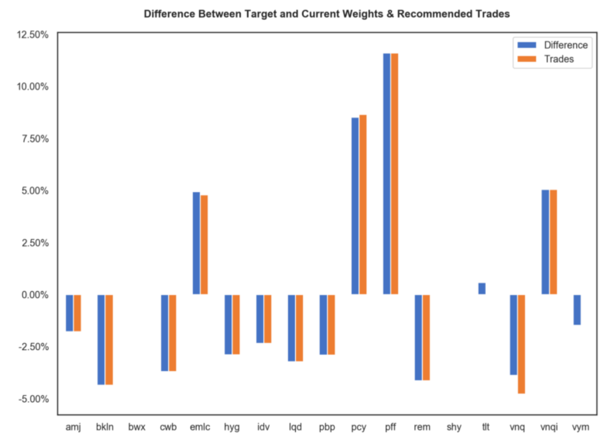
In the example below, we introduce a covariance matrix and seek to target a solution whose tracking error is less than 0.25%.

```
covariance_matrix = [[ 3.62767735e-02,  2.18757921e-03,  2.88389154e-05,
          7.34489308e-03,  1.96701876e-03,  4.42465667e-03,
          1.12579361e-02,  1.65860525e-03,  5.64030644e-03,
          2.76645571e-03,  3.63015800e-04,  3.74241173e-03,
         -1.35199744e-04, -2.19000672e-03,  6.80914121e-03,
          8.41701096e-03,  1.07504229e-02],
        [ 2.18757921e-03,  5.40346050e-04,  5.52196510e-04,
          9.03853792e-04,  1.26047511e-03,  6.54178355e-04,
          1.72005989e-03,  3.60920296e-04,  4.32241813e-04,
          6.55664695e-04,  1.60990263e-04,  6.64729334e-04,
         -1.34505970e-05, -3.61651337e-04,  6.56663689e-04,
          1.55184724e-03,  1.06451898e-03],
        [ 2.88389154e-05,  5.52196510e-04,  4.73857357e-03,
          1.55701811e-03,  6.22138578e-03,  8.13498400e-04,
          3.36654245e-03,  1.54941008e-03,  6.19861236e-05,
          2.93028853e-03,  8.70115005e-04,  4.90113403e-04,
          1.22200026e-04,  2.34074752e-03,  1.39606650e-03,
          5.31970717e-03,  8.86435533e-04],
        [ 7.34489308e-03,  9.03853792e-04,  1.55701811e-03,
          4.70643696e-03,  2.36059044e-03,  1.45119740e-03,
          4.46141908e-03,  8.06488179e-04,  2.09341490e-03,
          1.54107719e-03,  6.99000273e-04,  1.31596059e-03,
         -2.52039718e-05, -5.18390335e-04,  2.41334278e-03,
          5.14806453e-03,  3.76769305e-03],
        [ 1.96701876e-03,  1.26047511e-03,  6.22138578e-03,
          2.36059044e-03,  1.26644146e-02,  2.00358907e-03,
          8.04023724e-03,  2.30076077e-03,  5.70077091e-04,
          5.65049374e-03,  9.76571021e-04,  1.85279450e-03,
          2.56652171e-05,  1.19266940e-03,  5.84713900e-04,
          9.29778319e-03,  2.84300900e-03],
        [ 4.42465667e-03,  6.54178355e-04,  8.13498400e-04,
          1.45119740e-03,  2.00358907e-03,  1.52522064e-03,
          2.91651452e-03,  8.70569737e-04,  1.09752760e-03,
          1.66762294e-03,  5.36854007e-04,  1.75343988e-03,
          1.29714019e-05,  9.11071171e-05,  1.68043070e-03,
          2.42628131e-03,  1.90713194e-03],
        [ 1.12579361e-02,  1.72005989e-03,  3.36654245e-03,
          4.46141908e-03,  8.04023724e-03,  2.91651452e-03,
          1.19931947e-02,  1.61222907e-03,  2.75699780e-03,
          4.16113427e-03,  6.25609018e-04,  2.91008175e-03,
         -1.92908806e-04, -1.57151126e-03,  3.25855486e-03,
          1.06990068e-02,  6.05007409e-03],
        [ 1.65860525e-03,  3.60920296e-04,  1.54941008e-03,
          8.06488179e-04,  2.30076077e-03,  8.70569737e-04,
          1.61222907e-03,  1.90797844e-03,  6.04486114e-04,
          2.47501106e-03,  8.57227194e-04,  2.42587888e-03,
          1.85623409e-04,  2.91479004e-03,  3.33754926e-03,
          2.61280946e-03,  1.16461350e-03],
        [ 5.64030644e-03,  4.32241813e-04,  6.19861236e-05,
          2.09341490e-03,  5.70077091e-04,  1.09752760e-03,
          2.75699780e-03,  6.04486114e-04,  2.53455649e-03,
          9.66091919e-04,  3.91053383e-04,  1.83120456e-03,
         -4.91230334e-05, -5.60316891e-04,  2.28627416e-03,
          2.40776877e-03,  3.15907037e-03],
        [ 2.76645571e-03,  6.55664695e-04,  2.93028853e-03,
          1.54107719e-03,  5.65049374e-03,  1.66762294e-03,
          4.16113427e-03,  2.47501106e-03,  9.66091919e-04,
          4.81734656e-03,  1.14396535e-03,  3.23711266e-03,
          1.69157413e-04,  3.03445975e-03,  3.09323955e-03,
          5.27456576e-03,  2.11317800e-03],
        [ 3.63015800e-04,  1.60990263e-04,  8.70115005e-04,
          6.99000273e-04,  9.76571021e-04,  5.36854007e-04,
          6.25609018e-04,  8.57227194e-04,  3.91053383e-04,
          1.14396535e-03,  1.39905835e-03,  2.01826986e-03,
          1.04811491e-04,  1.67653296e-03,  2.59598793e-03,
          1.01532651e-03,  2.60716967e-04],
        [ 3.74241173e-03,  6.64729334e-04,  4.90113403e-04,
          1.31596059e-03,  1.85279450e-03,  1.75343988e-03,
          2.91008175e-03,  2.42587888e-03,  1.83120456e-03,
          3.23711266e-03,  2.01826986e-03,  1.16861730e-02,
          2.24795908e-04,  3.46679680e-03,  8.38606091e-03,
          3.65575720e-03,  1.80220367e-03],
        [-1.35199744e-04, -1.34505970e-05,  1.22200026e-04,
         -2.52039718e-05,  2.56652171e-05,  1.29714019e-05,
         -1.92908806e-04,  1.85623409e-04, -4.91230334e-05,
          1.69157413e-04,  1.04811491e-04,  2.24795908e-04,
          5.49990619e-05,  5.01897963e-04,  3.74856789e-04,
         -8.63113243e-06, -1.51400879e-04],
        [-2.19000672e-03, -3.61651337e-04,  2.34074752e-03,
         -5.18390335e-04,  1.19266940e-03,  9.11071171e-05,
         -1.57151126e-03,  2.91479004e-03, -5.60316891e-04,
          3.03445975e-03,  1.67653296e-03,  3.46679680e-03,
          5.01897963e-04,  8.74709395e-03,  6.37760454e-03,
          1.74349274e-03, -1.26348683e-03],
        [ 6.80914121e-03,  6.56663689e-04,  1.39606650e-03,
          2.41334278e-03,  5.84713900e-04,  1.68043070e-03,
          3.25855486e-03,  3.33754926e-03,  2.28627416e-03,
          3.09323955e-03,  2.59598793e-03,  8.38606091e-03,
          3.74856789e-04,  6.37760454e-03,  1.55034038e-02,
          5.20888498e-03,  4.17926704e-03],
        [ 8.41701096e-03,  1.55184724e-03,  5.31970717e-03,
          5.14806453e-03,  9.29778319e-03,  2.42628131e-03,
          1.06990068e-02,  2.61280946e-03,  2.40776877e-03,
          5.27456576e-03,  1.01532651e-03,  3.65575720e-03,
         -8.63113243e-06,  1.74349274e-03,  5.20888498e-03,
          1.35424275e-02,  5.49882762e-03],
        [ 1.07504229e-02,  1.06451898e-03,  8.86435533e-04,
          3.76769305e-03,  2.84300900e-03,  1.90713194e-03,
          6.05007409e-03,  1.16461350e-03,  3.15907037e-03,
          2.11317800e-03,  2.60716967e-04,  1.80220367e-03,
         -1.51400879e-04, -1.26348683e-03,  4.17926704e-03,
          5.49882762e-03,  7.08734925e-03]]

covariance_matrix = pandas.DataFrame(covariance_matrix, \
                                     index = tickers, \
                                     columns = tickers)
```

```
theta = 0.05
target_te = 0.0025

w_old_prime = w_old.copy()

# calculate the difference from the target portfolio
# and use this difference to estimate tracking error
# and marginal contribution to tracking error ("mcte")
z = (w_old_prime - w_target)
te = numpy.sqrt(z.dot(covariance_matrix).dot(z))
mcte = (z.dot(covariance_matrix)) / te

while True:
```

```python
    w_diff_prime = w_target - w_old_prime

    trading_model = LpProblem("Trade Minimization Problem", LpMinimize)

    t_vars = []
    psi_vars = []
    phi_vars = []
    y_vars = []

    A = 2

    for i in range(n):
        t = LpVariable("t_" + str(i), -w_old_prime[i], 1 - w_old_prime[i])
        t_vars.append(t)

        psi = LpVariable("psi_" + str(i), None, None)
        psi_vars.append(psi)

        phi = LpVariable("phi_" + str(i), None, None)
        phi_vars.append(phi)

        y = LpVariable("y_" + str(i), 0, 1, LpInteger) #set y in {0, 1}
        y_vars.append(y)


    # add our objective to minimize y, which is the number of trades
    trading_model += lpSum(phi_vars) + lpSum(y_vars), "Objective"

    for i in range(n):
        trading_model += psi_vars[i] >= -t_vars[i]
        trading_model += psi_vars[i] >= t_vars[i]
        trading_model += psi_vars[i] <= A * y_vars[i]

    for i in range(n):
        trading_model += phi_vars[i] >= -(w_diff_prime[i] - t_vars[i])
        trading_model += phi_vars[i] >= (w_diff_prime[i] - t_vars[i])

    # Make sure our trades sum to zero
    trading_model += (lpSum(t_vars) == 0)

    # Set tracking error limit
    #    delta(te) = mcte * delta(z)
    #              = mcte * ((w_old_prime + t - w_target) -
    #                         (w_old_prime - w_target))
    #              = mcte * t
    #    te + delta(te) <= target_te
    #    ==> delta(te) <= target_te - te
    trading_model += (lpSum([mcte.iloc[i] * t_vars[i] for i in range(n)]) \
                              <= (target_te - te))

    # Set our trade bounds
    trading_model += (lpSum(phi_vars) / 2. <= theta)

    trading_model.solve()

    # update our w_old' with the current trades
    results = pandas.Series([t_i.value() for t_i in t_vars], index = tickers)
    w_old_prime = (w_old_prime + results)

    z = (w_old_prime - w_target)
    te = numpy.sqrt(z.dot(covariance_matrix).dot(z))
    mcte = (z.dot(covariance_matrix)) / te

    if te < target_te:
        break

print "Tracking error: " + str(te)

# since w_old' is an iterative update,
# the current trades only reflect the updates from
# the prior w_old'.  Thus, we need to calculate
# the trades by hand
results = (w_old_prime - w_old)
n_trades = (results.abs() > 1e-8).astype(int).sum()

print "Number of trades: " + str(n_trades)

print "Turnover distance: " + str((w_target - (w_old + results)).abs().sum() / 2.)
```

```
Tracking error: 0.0016583319880074485
Number of trades: 13
Turnover distance: 0.01624453350000001
```



6.2 Time Constraints

For time feasibility, heuristic approaches can be employed in effort to rapidly converge upon a "close enough" solution. For example, Rong and Liu (2011) discuss "build-up" and "pare-down" heuristics.

The basic algorithm of "pare-down" is:

1. Start with a trade list that includes every security
2. Solve the optimization problem in its unconstrained format, allowing trades to occur only for securities in the trade list.
3. If the solution meets the necessary constraints (e.g. maximum number of trades, trade size thresholds, tracking error constraints, etc), terminate the optimization.
4. Eliminate from the trade list a subset of securities based upon some measure of trade utility (e.g. violation of constraints, contribution to tracking error, etc).
5. Go to step 2.

The basic algorithm of "build-up" is:

1. Start with an empty trade list
2. Add a subset of securities to the trade list based upon some measure of trade utility.
3. Solve the optimization problem in its unconstrained format, allowing trades to occur only for securities in the trade list.
4. If the solution meets the necessary constraints (e.g. maximum number of trades, trade size thresholds, tracking error constraints, etc), terminate the optimization.
5. Go to step 2.

These two heuristics can even be combined in an integrated fashion. For example, a binary search approach can be employed, where the initial trade list list is filled with 50% of the tradable securities. Depending upon success or failure of the resulting optimization, a pare-down or build-up approach can be taken to either prune or expand the trade list.

7. Conclusion

In this research note we have explored the practice of trade optimization, which seeks to implement portfolio changes in as few trade as possible. While a rarely discussed detail of portfolio management, trade optimization has the potential to eliminate unnecessary trading costs – both explicit and implicit – that can be a drag on realized investor performance.

Constraints within the practice of trade optimization typically fall into one of three categories: asset paring, trade paring, and level paring. Asset paring restricts the number of securities the portfolio can hold, trade paring restricts the number of trades that can be made, and level paring restricts the size of positions and trades. Introducing these constraints often turns an optimization into a discrete problem, making it much more difficult to solve for traditional convex optimizations.

With this in mind, we introduced mixed-integer linear programming ("MILP") and explore a few techniques that can be utilized to transform non-linear functions into a set of linear constraints. We then combined these transformations to develop a simple trade optimization framework that can be solved using MILP optimizers.

To offer numerical support in the discussion, we created a simple momentum-based sector rotation strategy. We found that naive turnover-filtering helped reduce the number of trades executed by 50%, while explicit trade optimization reduced the number of trades by 70%.

Finally, we explored how our simplified framework could be further extended to account for both non-linear functional constraints (e.g. tracking error) and operational constraints (e.g. managing execution time).

The paring constraints introduced by trade optimization often lead to problems that are difficult to solve. However, when we consider that the cost of trading is a very real drag on the results realized by investors, we believe that the solutions are worth pursuing.

Corey Hoffstein

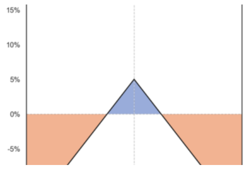
Corey is co-founder and Chief Investment Officer of Newfound Research, a quantitative asset manager offering a suite of separately managed accounts and mutual funds. At Newfound, Corey is responsible for portfolio management, investment research, strategy development, and communication of the firm's views to clients. Prior to offering asset management services, Newfound licensed research from the quantitative investment models developed by Corey. At peak, this research helped steer the tactical allocation decisions for upwards of $10bn. Corey holds a Master of Science in Computational Finance from Carnegie Mellon University and a Bachelor of Science in Computer Science, cum laude, from Cornell University. You can connect with Corey on LinkedIn or Twitter.

Or schedule a time to connect.

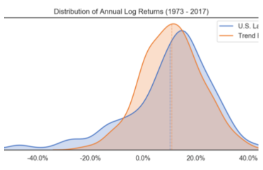
LINEAR PROGRAMMING   MIXED INTEGER LINEAR PROGRAMMINNG   TRADE OPTIMIZATION   TRADE PARING

READ NEXT

Q2 2019 Firm Update


A Trend Equity Primer

Sequence Risk


2018 Highlights – The Top 20 Posts You Might Have Missed

2 PINGBACKS

Quantocracy's Daily Wrap for 08/27/2018 | Quantocracy

Research links: trade optimization – 365bet手机版