CHAPTER 13

Backtesting on Synthetic Data

13.1 MOTIVATION

In this chapter we will study an alternative backtesting method, which uses history to generate a synthetic dataset with statistical characteristics estimated from the observed data. This will allow us to backtest a strategy on a large number of unseen, synthetic testing sets, hence reducing the likelihood that the strategy has been fit to a particular set of datapoints.[1] This is a very extensive subject, and in order to reach some depth we will focus on the backtesting of trading rules.

13.2 TRADING RULES

Investment strategies can be defined as algorithms that postulate the existence of a market inefficiency. Some strategies rely on econometric models to predict prices, using macroeconomic variables such as GDP or inflation; other strategies use fundamental and accounting information to price securities, or search for arbitrage-like opportunities in the pricing of derivatives products, etc. For instance, suppose that financial intermediaries tend to sell off-the-run bonds two days before U.S. Treasury auctions, in order to raise the cash needed for buying the new "paper." One could monetize on that knowledge by selling off-the-run bonds three days before auctions. But how? Each investment strategy requires an implementation tactic, often referred to as "trading rules."

There are dozens of hedge fund styles, each running dozens of unique investment strategies. While strategies can be very heterogeneous in nature, tactics are relatively homogeneous. Trading rules provide the algorithm that must be followed to enter and exit a position. For example, a position will be entered when the strategy's signal

[1] I would like to thank Professor Peter Carr (New York University) for his contributions to this chapter.

reaches a certain value. Conditions for exiting a position are often defined through thresholds for profit-taking and stop-losses. These entry and exit rules rely on parameters that are usually calibrated via historical simulations. This practice leads to the problem of *backtest overfitting*, because these parameters target specific observations in-sample, to the point that the investment strategy is so attached to the past that it becomes unfit for the future.

An important clarification is that we are interested in the exit corridor conditions that maximize performance. In other words, the position already exists, and the question is how to exit it optimally. This is the dilemma often faced by execution traders, and it should not be mistaken with the determination of entry and exit thresholds for investing in a security. For a study of that alternative question, see, for example, Bertram [2009].

Bailey et al. [2014, 2017] discuss the problem of backtest overfitting, and provide methods to determine to what extent a simulated performance may be inflated due to overfitting. While assessing the probability of backtest overfitting is a useful tool to discard superfluous investment strategies, it would be better to avoid the risk of overfitting, at least in the context of calibrating a trading rule. In theory this could be accomplished by deriving the optimal parameters for the trading rule directly from the stochastic process that generates the data, rather than engaging in historical simulations. This is the approach we take in this chapter. Using the entire historical sample, we will characterize the stochastic process that generates the observed stream of returns, and derive the optimal values for the trading rule's parameters without requiring a historical simulation.

13.3 THE PROBLEM

Suppose an investment strategy S invests in $i = 1, \dots I$ opportunities or bets. At each opportunity i, S takes a position of m_i units of security X, where $m_i \in (-\infty, \infty)$. The transaction that entered such opportunity was priced at a value $m_i P_{i,0}$, where $P_{i,0}$ is the average price per unit at which the m_i securities were transacted. As other market participants transact security X, we can mark-to-market (MtM) the value of that opportunity i after t observed transactions as $m_i P_{i,t}$. This represents the value of opportunity i if it were liquidated at the price observed in the market after t transactions. Accordingly, we can compute the MtM profit/loss of opportunity i after t transactions as $\pi_{i,t} = m_i(P_{i,t} - P_{i,0})$.

A standard trading rule provides the logic for exiting opportunity i at $t = T_i$. This occurs as soon as one of two conditions is verified:

- $\pi_{i,T_i} \geq \bar{\pi}$, where $\bar{\pi} > 0$ is the profit-taking threshold.
- $\pi_{i,T_i} \leq \underline{\pi}$, where $\underline{\pi} < 0$ is the stop-loss threshold.

These thresholds are equivalent to the horizontal barriers we discussed in the context of meta-labelling (Chapter 3). Because $\underline{\pi} < \bar{\pi}$, one and only one of the two exit conditions can trigger the exit from opportunity i. Assuming that opportunity i can

be exited at T_i, its final profit/loss is π_{i,T_i}. At the onset of each opportunity, the goal is to realize an expected profit $\mathrm{E}_0[\pi_{i,T_i}] = m_i(\mathrm{E}_0[P_{i,T_i}] - P_{i,0})$, where $\mathrm{E}_0[P_{i,T_i}]$ is the forecasted price and $P_{i,0}$ is the entry level of opportunity i.

Definition 1: Trading Rule: A trading rule for strategy S is defined by the set of parameters $R := \{\underline{\pi}, \bar{\pi}\}$.

One way to calibrate (by brute force) the trading rule is to:

1. Define a set of alternative values of R, $\Omega := \{R\}$.
2. Simulate historically (backtest) the performance of S under alternative values of $R \in \Omega$.
3. Select the optimal R^*.

More formally:

$$R^* = \arg\max_{R \in \Omega} \{SR_R\}$$
$$SR_R = \frac{\mathrm{E}[\pi_{i,T_i}|R]}{\sigma[\pi_{i,T_i}|R]} \tag{13.1}$$

where $\mathrm{E}[.]$ and $\sigma[.]$ are respectively the expected value and standard deviation of π_{i,T_i}, conditional on trading rule R, over $i = 1, \ldots I$. In other words, equation (13.1) maximizes the Sharpe ratio of S on I opportunities over the space of alternative trading rules R (see Bailey and López de Prado [2012] for a definition and analysis of the Sharpe ratio). Because we count with two variables to maximize SR_R over a sample of size I, it is easy to overfit R. A trivial overfit occurs when a pair $(\underline{\pi}, \bar{\pi})$ targets a few outliers. Bailey et al. [2017] provide a rigorous definition of backtest overfitting, which can be applied to our study of trading rules as follows.

Definition 2: Overfit Trading Rule: R^* is overfit if $\mathrm{E}\left[\frac{E\left[\pi_{j,T_j}\middle|R^*\right]}{\sigma\left[\pi_{j,T_j}\middle|R^*\right]}\right] < \mathrm{Me}_\Omega\left[\mathrm{E}\left[\frac{E\left[\pi_{j,T_j}\middle|R\right]}{\sigma\left[\pi_{j,T_j}\middle|R\right]}\right]\right]$, where $j = I+1, \ldots J$ and $\mathrm{Me}_\Omega[.]$ is the median.

Intuitively, an optimal in-sample (IS, $i \in [1, I]$) trading rule R^* is overfit when it is expected to underperform the median of alternative trading rules $R \in \Omega$ out-of-sample (OOS, $j \in [I+1, J]$). This is essentially the same definition we used in chapter 11 to derive PBO. Bailey et al. [2014] argue that it is hard not to overfit a backtest, particularly when there are free variables able to target specific observations IS, or the number of elements in Ω is large. A trading rule introduces such free variables,

because R^* can be determined independently from S. The outcome is that the backtest profits from random noise IS, making R^* unfit for OOS opportunities. Those same authors show that overfitting leads to negative performance OOS when $\Delta\pi_{i,t}$ exhibits serial dependence. While PBO provides a useful method to evaluate to what extent a backtest has been overfit, it would be convenient to avoid this problem in the first place.[2] To that aim we dedicate the following section.

13.4 OUR FRAMEWORK

Until now we have not characterized the stochastic process from which observations $\pi_{i,t}$ are drawn. We are interested in finding an optimal trading rule (OTR) for those scenarios where overfitting would be most damaging, such as when $\pi_{i,t}$ exhibits serial correlation. In particular, suppose a discrete Ornstein-Uhlenbeck (O-U) process on prices

$$P_{i,t} = (1-\varphi)\,\mathrm{E}_0[P_{i,T_i}] + \varphi P_{i,t-1} + \sigma\varepsilon_{i,t} \tag{13.2}$$

such that the random shocks are IID distributed $\varepsilon_{i,t} \sim N(0,1)$. The seed value for this process is $P_{i,0}$, the level targeted by opportunity i is $\mathrm{E}_0[P_{i,T_i}]$, and φ determines the speed at which $P_{i,0}$ converges towards $\mathrm{E}_0[P_{i,T_i}]$. Because $\pi_{i,t} = m_i(P_{i,t} - P_{i,0})$, equation (13.2) implies that the performance of opportunity i is characterized by the process

$$\frac{1}{m_i}\pi_{i,t} = (1-\varphi)\mathrm{E}_0[P_{i,T_i}] - P_{i,0} + \varphi P_{i,t-1} + \sigma\varepsilon_{i,t} \tag{13.3}$$

From the proof to Proposition 4 in Bailey and López de Prado [2013], it can be shown that the distribution of the process specified in equation (13.2) is Gaussian with parameters

$$\pi_{i,t} \sim N\left[m_i\left((1-\varphi)\,\mathrm{E}_0[P_{i,T_i}]\sum_{j=0}^{t-1}\varphi^j - P_{i,0}\right), m_i^2\sigma^2\sum_{j=0}^{t-1}\varphi^{2j}\right] \tag{13.4}$$

and a necessary and sufficient condition for its stationarity is that $\varphi \in (-1,1)$. Given a set of input parameters $\{\sigma,\varphi\}$ and initial conditions $\{P_{i,0},\mathrm{E}_0[P_{i,T_i}]\}$ associated with opportunity i, is there an OTR $R^* := (\underline{\pi},\bar{\pi})$? Similarly, should strategy S predict a profit target $\bar{\pi}$, can we compute the optimal stop-loss $\underline{\pi}$ given the input values $\{\sigma,\varphi\}$? If the answer to these questions is affirmative, no backtest would be needed in order to determine R^*, thus avoiding the problem of overfitting the trading rule. In the next section we will show how to answer these questions experimentally.

[2] The strategy may still be the result of backtest overfitting, but at least the trading rule would not have contributed to that problem.

13.5 NUMERICAL DETERMINATION OF OPTIMAL TRADING RULES

In the previous section we used an O-U specification to characterize the stochastic process generating the returns of strategy *S*. In this section we will present a procedure to numerically derive the OTR for any specification in general, and the O-U specification in particular.

13.5.1 The Algorithm

The algorithm consists of five sequential steps.

Step 1: We estimate the input parameters $\{\sigma, \varphi\}$, by linearizing equation (13.2) as:

$$P_{i,t} = \mathrm{E}_0[P_{i,T_i}] + \varphi(P_{i,t-1} - \mathrm{E}_0[P_{i,T_i}]) + \xi_t \tag{13.5}$$

We can then form vectors X and Y by sequencing opportunities:

$$X = \begin{bmatrix} P_{0,0} - \mathrm{E}_0[P_{0,T_0}] \\ P_{0,1} - \mathrm{E}_0[P_{0,T_0}] \\ \cdots \\ P_{0,T-1} - \mathrm{E}_0[P_{0,T_0}] \\ \cdots \\ P_{I,0} - \mathrm{E}_0[P_{I,T_I}] \\ \cdots \\ P_{I,T-1} - \mathrm{E}_0[P_{I,T_I}] \end{bmatrix};\ Y = \begin{bmatrix} P_{0,1} \\ P_{0,2} \\ \cdots \\ P_{0,T} \\ \cdots \\ P_{I,1} \\ \cdots \\ P_{I,T} \end{bmatrix};\ Z = \begin{bmatrix} \mathrm{E}_0[P_{0,T_0}] \\ \mathrm{E}_0[P_{0,T_0}] \\ \cdots \\ \mathrm{E}_0[P_{0,T_0}] \\ \cdots \\ \mathrm{E}_0[P_{I,T_I}] \\ \cdots \\ \mathrm{E}_0[P_{I,T_I}] \end{bmatrix} \tag{13.6}$$

Applying OLS on equation (13.5), we can estimate the original O-U parameters as,

$$\begin{aligned} \hat{\varphi} &= \frac{\mathrm{cov}\,[Y, X]}{\mathrm{cov}\,[X, X]} \\ \hat{\xi}_t &= Y - Z - \hat{\varphi} X \\ \hat{\sigma} &= \sqrt{\mathrm{cov}[\hat{\xi}_t, \hat{\xi}_t]} \end{aligned} \tag{13.7}$$

where $\mathrm{cov}\,[\cdot, \cdot]$ is the covariance operator.

Step 2: We construct a mesh of stop-loss and profit-taking pairs, $(\underline{\pi}, \bar{\pi})$. For example, a Cartesian product of $\underline{\pi} = \{-\frac{1}{2}\sigma, -\sigma, \ldots, -10\sigma\}$ and $\bar{\pi} = \{\frac{1}{2}\sigma, \sigma, \ldots, 10\sigma\}$ give us 20×20 nodes, each constituting an alternative trading rule $R \in \Omega$.

Step 3: We generate a large number of paths (e.g., 100,000) for $\pi_{i,t}$ applying our estimates $\{\hat{\sigma}, \hat{\varphi}\}$. As seed values, we use the observed initial conditions $\{P_{i,0}, \mathrm{E}_0[P_{i,T_i}]\}$ associated with an opportunity i. Because a position cannot be held for an unlimited period of time, we can impose a maximum holding period (e.g., 100 observations) at which point the position is exited even though $\underline{\pi} \leq \pi_{i,100} \leq \bar{\pi}$. This maximum holding period is equivalent to the vertical bar of the triple-barrier method (Chapter 3).[3]

Step 4: We apply the 100,000 paths generated in Step 3 on each node of the 20×20 mesh $(\underline{\pi}, \bar{\pi})$ generated in Step 2. For each node, we apply the stop-loss and profit-taking logic, giving us 100,000 values of π_{i,T_i}. Likewise, for each node we compute the Sharpe ratio associated with that trading rule as described in equation (13.1). See Bailey and López de Prado [2012] for a study of the confidence interval of the Sharpe ratio estimator. This result can be used in three different ways: Step 5a, Step 5b and Step 5c).

Step 5a: We determine the pair $(\underline{\pi}, \bar{\pi})$ within the mesh of trading rules that is optimal, given the input parameters $\{\hat{\sigma}, \hat{\varphi}\}$ and the observed initial conditions $\{P_{i,0}, \mathrm{E}_0[P_{i,T_i}]\}$.

Step 5b: If strategy S provides a profit target $\overline{\pi_i}$ for a particular opportunity i, we can use that information in conjunction with the results in Step 4 to determine the optimal stop-loss, $\underline{\pi_i}$.

Step 5c: If the trader has a maximum stop-loss $\underline{\pi_i}$ imposed by the fund's management for opportunity i, we can use that information in conjunction with the results in Step 4 to determine the optimal profit-taking $\overline{\pi_i}$ within the range of stop-losses $[0, \underline{\pi_i}]$.

Bailey and López de Prado [2013] prove that the half-life of the process in equation (13.2) is $\tau = -\frac{\log[2]}{\log[\varphi]}$, with the requirement that $\varphi \in (0, 1)$. From that result, we can determine the value of φ associated with a certain half-life τ as $\varphi = 2^{-1/\tau}$.

13.5.2 Implementation

Snippet 13.1 provides an implementation in Python of the experiments conducted in this chapter. Function `main` produces a Cartesian product of parameters $(\mathrm{E}_0[P_{i,T_i}], \tau)$, which characterize the stochastic process from equation (13.5). Without loss of generality, in all simulations we have used $\sigma = 1$. Then, for each pair $(\mathrm{E}_0[P_{i,T_i}], \tau)$, function `batch` computes the Sharpe ratios associated with various trading rules.

[3] The trading rule R could be characterized as a function of the three barriers, instead of the horizontal ones. That change would have no impact on the procedure. It would merely add one more dimension to the mesh ($20 \times 20 \times 20$). In this chapter we do not consider that setting, because it would make the visualization of the method less intuitive.

SNIPPET 13.1 PYTHON CODE FOR THE DETERMINATION OF OPTIMAL TRADING RULES

```
import numpy as np
from random import gauss
from itertools import product
#---------------------------------------
def main():
    rPT=rSLm=np.linspace(0,10,21)
    count=0
    for prod_ in product([10,5,0,-5,-10],[5,10,25,50,100]):
        count+=1
        coeffs={'forecast':prod_[0],'hl':prod_[1],'sigma':1}
        output=batch(coeffs,nIter=1e5,maxHP=100,rPT=rPT,rSLm=rSLm)
    return output
```

Snippet 13.2 computes a 20×20 mesh of Sharpe ratios, one for each trading rule $(\underline{\pi}, \bar{\pi})$, given a pair of parameters $(\mathrm{E}_0[P_{i,T_i}], \tau)$. There is a vertical barrier, as the maximum holding period is set at 100 (`maxHP = 100`). We have fixed $P_{i,0} = 0$, since it is the distance $(P_{i,t-1} - \mathrm{E}_0[P_{i,T_i}])$ in equation (13.5) that drives the convergence, not particular absolute price levels. Once the first out of three barriers is touched, the exit price is stored, and the next iteration starts. After all iterations are completed (1E5), the Sharpe ratio can be computed for that pair $(\underline{\pi}, \bar{\pi})$, and the algorithm moves to the next pair. When all pairs of trading rules have been processed, results are reported back to `main`. This algorithm can be parallelized, similar to what we did for the triple-barrier method in Chapter 3. We leave that task as an exercise.

SNIPPET 13.2 PYTHON CODE FOR THE DETERMINATION OF OPTIMAL TRADING RULES

```
def batch(coeffs,nIter=1e5,maxHP=100,rPT=np.linspace(.5,10,20),
    rSLm=np.linspace(.5,10,20),seed=0):
    phi,output1=2**(-1./coeffs['hl']),[]
    for comb_ in product(rPT,rSLm):
        output2=[]
        for iter_ in range(int(nIter)):
            p,hp,count=seed,0,0
            while True:
                p=(1-phi)*coeffs['forecast']+phi*p+coeffs['sigma']*gauss(0,1)
                cP=p-seed;hp+=1
                if cP>comb_[0] or cP<-comb_[1] or hp>maxHP:
                    output2.append(cP)
                    break
```

```
    mean,std=np.mean(output2),np.std(output2)
    print comb_[0],comb_[1],mean,std,mean/std
    output1.append((comb_[0],comb_[1],mean,std,mean/std))
return output1
```

13.6 EXPERIMENTAL RESULTS

Table 13.1 lists the combinations analyzed in this study. Although different values for these input parameters would render different numerical results, the combinations applied allow us to analyze the most general cases. Column "Forecast" refers to $E_0[P_{i,T_i}]$; column "Half-Life" refers to τ; column "Sigma" refers to σ; column "maxHP" stands for maximum holding period.

In the following figures, we have plotted the non-annualized Sharpe ratios that result from various combinations of profit-taking and stop-loss exit conditions. We have omitted the negative sign in the y-axis (stop-losses) for simplicity. Sharpe ratios

TABLE 13.1 Input Parameter Combinations Used in the Simulations

Figure	Forecast	Half-Life	Sigma	maxHP
16.1	0	5	1	100
16.2	0	10	1	100
16.3	0	25	1	100
16.4	0	50	1	100
16.5	0	100	1	100
16.6	5	5	1	100
16.7	5	10	1	100
16.8	5	25	1	100
16.9	5	50	1	100
16.10	5	100	1	100
16.11	10	5	1	100
16.12	10	10	1	100
16.13	10	25	1	100
16.14	10	50	1	100
16.15	10	100	1	100
16.16	−5	5	1	100
16.17	−5	10	1	100
16.18	−5	25	1	100
16.19	−5	50	1	100
16.20	−5	100	1	100
16.21	−10	5	1	100
16.22	−10	10	1	100
16.23	−10	25	1	100
16.24	−10	50	1	100
16.25	−10	100	1	100

are represented in grayscale (lighter indicating better performance; darker indicating worse performance), in a format known as a heat-map. Performance (π_{i,T_i}) is computed per unit held ($m_i = 1$), since other values of m_i would simply re-scale performance, with no impact on the Sharpe ratio. Transaction costs can be easily added, but for educational purposes it is better to plot results without them, so that you can appreciate the symmetry of the functions.

13.6.1 Cases with Zero Long-Run Equilibrium

Cases with zero long-run equilibrium are consistent with the business of market-makers, who provide liquidity under the assumption that price deviations from current levels will correct themselves over time. The smaller τ, the smaller is the autoregressive coefficient ($\varphi = 2^{-1/\tau}$). A small autoregressive coefficient in conjunction with a zero expected profit has the effect that most of the pairs ($\underline{\pi_i}, \overline{\pi_i}$) deliver a zero performance.

Figure 13.1 shows the heat-map for the parameter combination $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{0, 5, 1\}$. The half-life is so small that performance is maximized in a narrow range of combinations of small profit-taking with large stop-losses. In other words, the optimal trading rule is to hold an inventory long enough until a small profit arises, even at the expense of experiencing some 5-fold or 7-fold unrealized losses. Sharpe ratios are

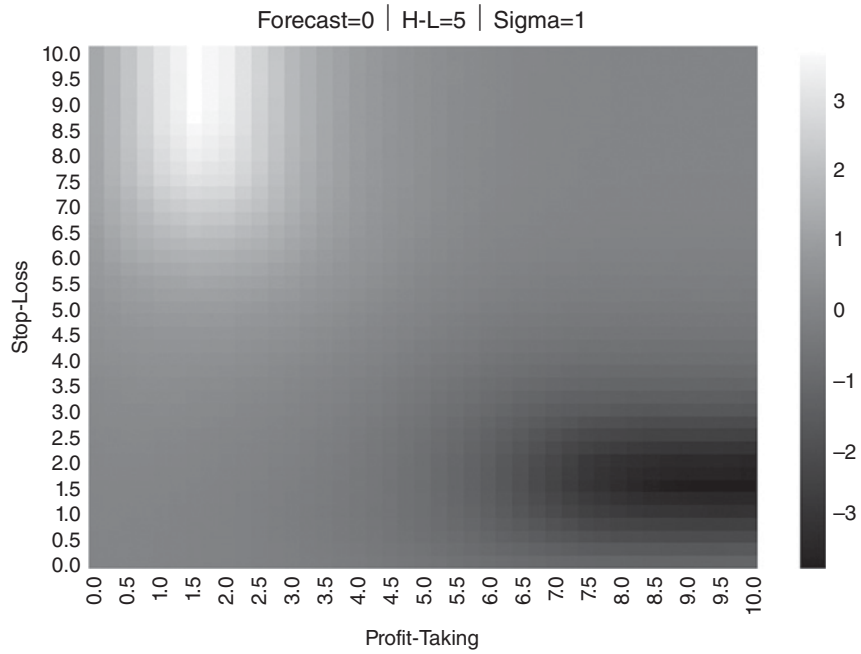


FIGURE 13.1 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{0, 5, 1\}$

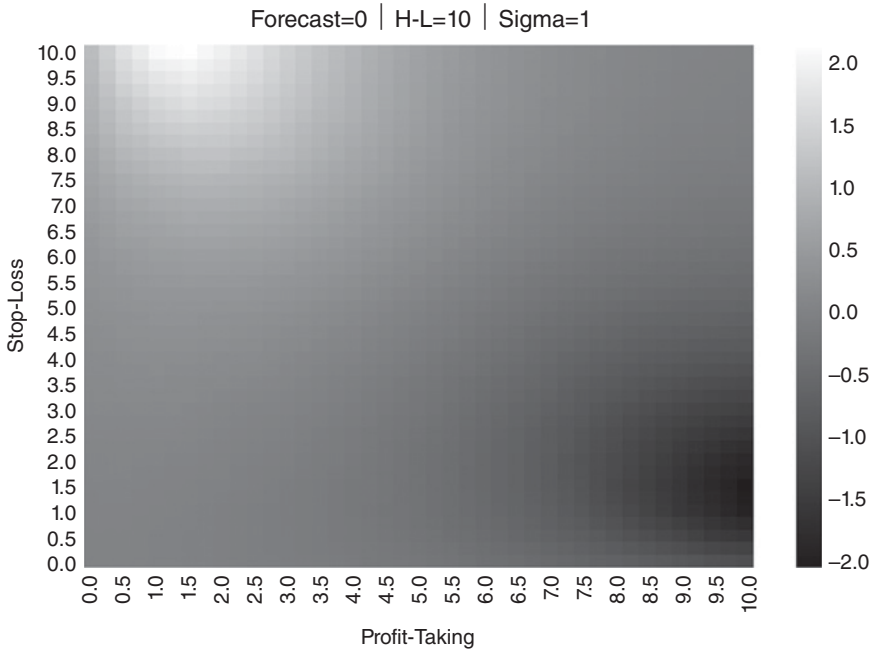


FIGURE 13.2 Heat-map for $\{E_0[P_{i,T_i}], \tau, \sigma\} = \{0, 10, 1\}$

high, reaching levels of around 3.2. This is in fact what many market-makers do in practice, and is consistent with the "asymmetric payoff dilemma" described in Easley et al. [2011]. The worst possible trading rule in this setting would be to combine a short stop-loss with a large profit-taking threshold, a situation that market-makers avoid in practice. Performance is closest to neutral in the diagonal of the mesh, where profit-taking and stop-losses are symmetric. You should keep this result in mind when labeling observations using the triple-barrier method (Chapter 3).

Figure 13.2 shows that, if we increase τ from 5 to 10, the areas of highest and lowest performance spread over the mesh of pairs $(\underline{\pi}, \bar{\pi})$, while the Sharpe ratios decrease. This is because, as the half-life increases, so does the magnitude of the autoregressive coefficient (recall that $\varphi = 2^{-1/\tau}$), thus bringing the process closer to a random walk.

In Figure 13.3, $\tau = 25$, which again spreads the areas of highest and lowest performance while reducing the Sharpe ratio. Figure 13.4 ($\tau = 50$) and Figure 13.5 ($\tau = 100$) continue that progression. Eventually, as $\varphi \to 1$, there are no recognizable areas where performance can be maximized.

Calibrating a trading rule on a random walk through historical simulations would lead to backtest overfitting, because one random combination of profit-taking and stop-loss that happened to maximize Sharpe ratio would be selected. This is why backtesting of synthetic data is so important: to avoid choosing a strategy because some statistical fluke took place in the past (a single random path). Our procedure

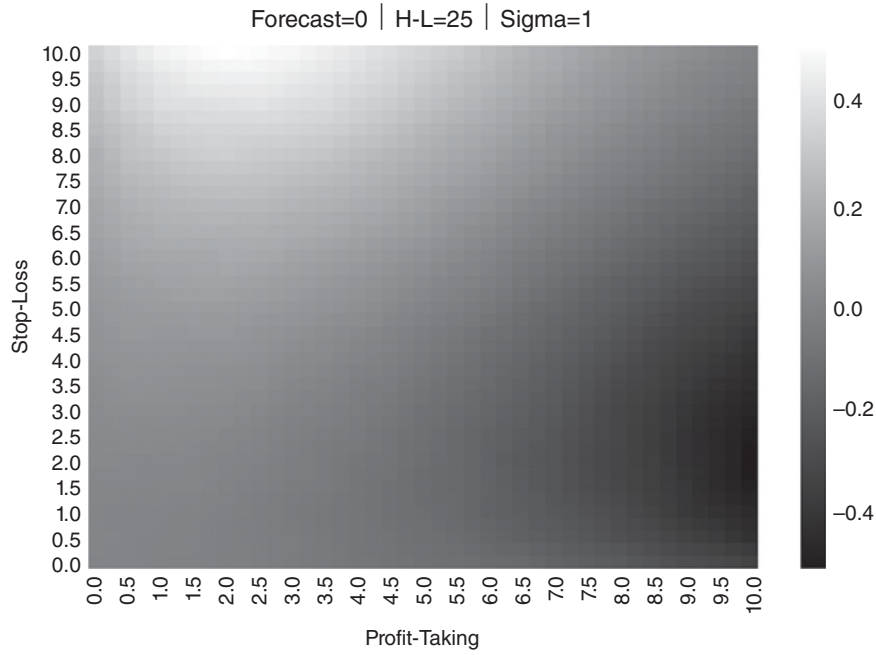


FIGURE 13.3 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{0, 25, 1\}$

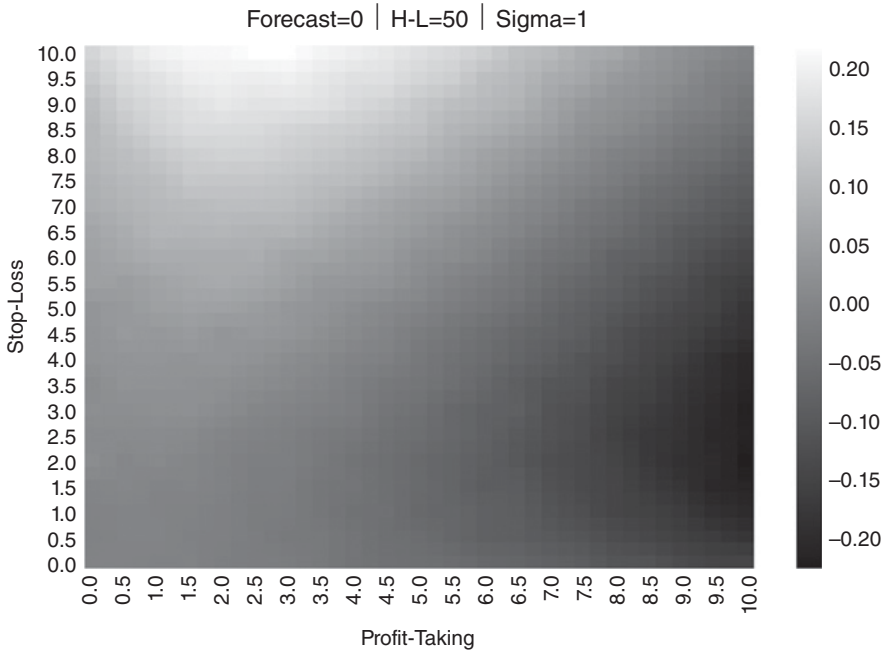


FIGURE 13.4 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{0, 50, 1\}$

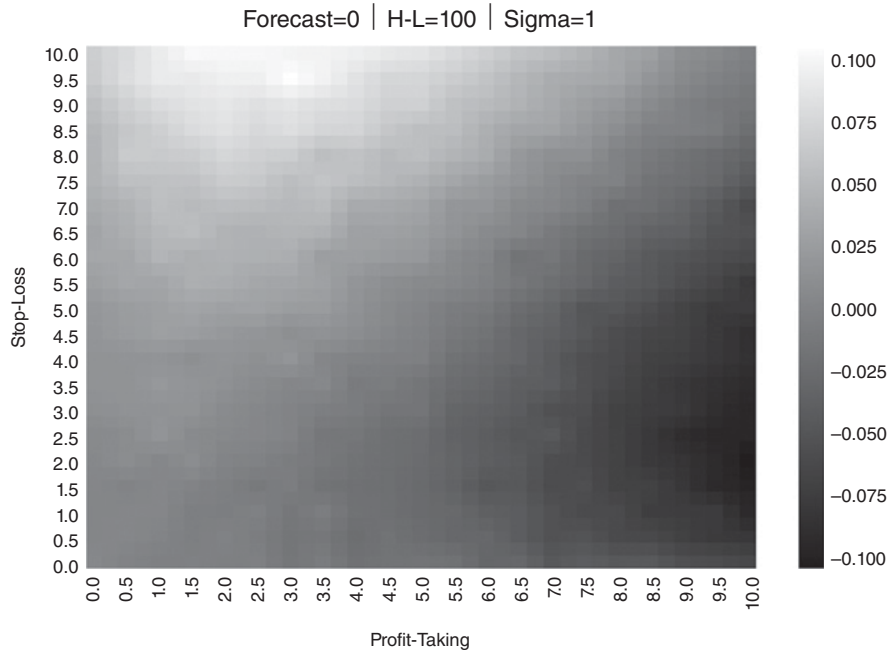


FIGURE 13.5 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{0, 100, 1\}$

prevents overfitting by recognizing that performance exhibits no consistent pattern, indicating that there is no optimal trading rule.

13.6.2 Cases with Positive Long-Run Equilibrium

Cases with positive long-run equilibrium are consistent with the business of a position-taker, such as a hedge-fund or asset manager. Figure 13.6 shows the results for the parameter combination $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{5, 5, 1\}$. Because positions tend to make money, the optimal profit-taking is higher than in the previous cases, centered around 6, with stop-losses that range between 4 and 10. The region of the optimal trading rule takes a characteristic rectangular shape, as a result of combining a wide stop-loss range with a narrower profit-taking range. Performance is highest across all experiments, with Sharpe ratios of around 12.

In Figure 13.7, we have increased the half-life from $\tau = 5$ to $\tau = 10$. Now the optimal performance is achieved at a profit-taking centered around 5, with stop-losses that range between 7 and 10. The range of optimal profit-taking is wider, while the range of optimal stop-losses narrows, shaping the former rectangular area closer to a square. Again, a larger half-life brings the process closer to a random walk, and therefore performance is now relatively lower than before, with Sharpe ratios of around 9.

In Figure 13.8, we have made $\tau = 25$. The optimal profit-taking is now centered around 3, while the optimal stop-losses range between 9 and 10. The previous squared

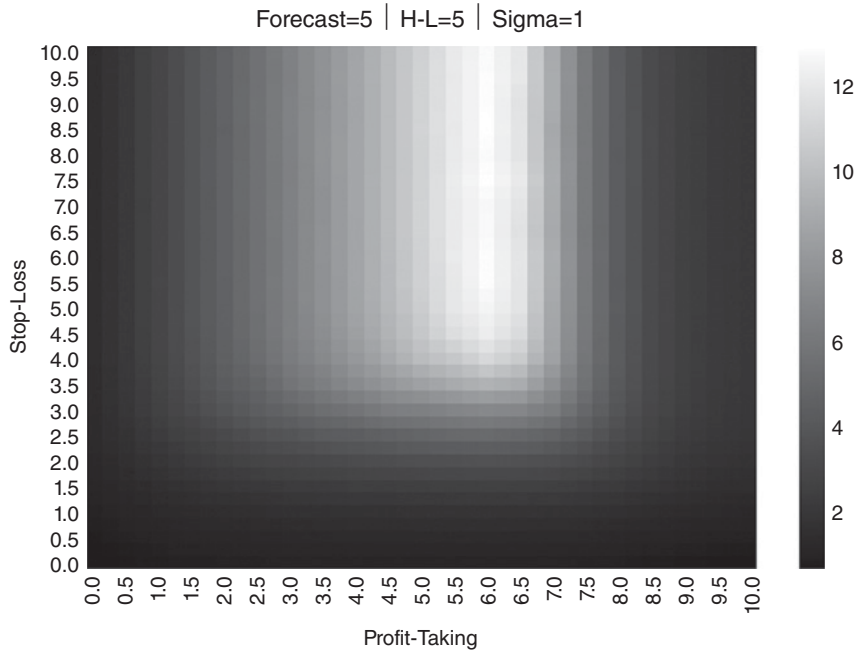


FIGURE 13.6 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{5, 5, 1\}$

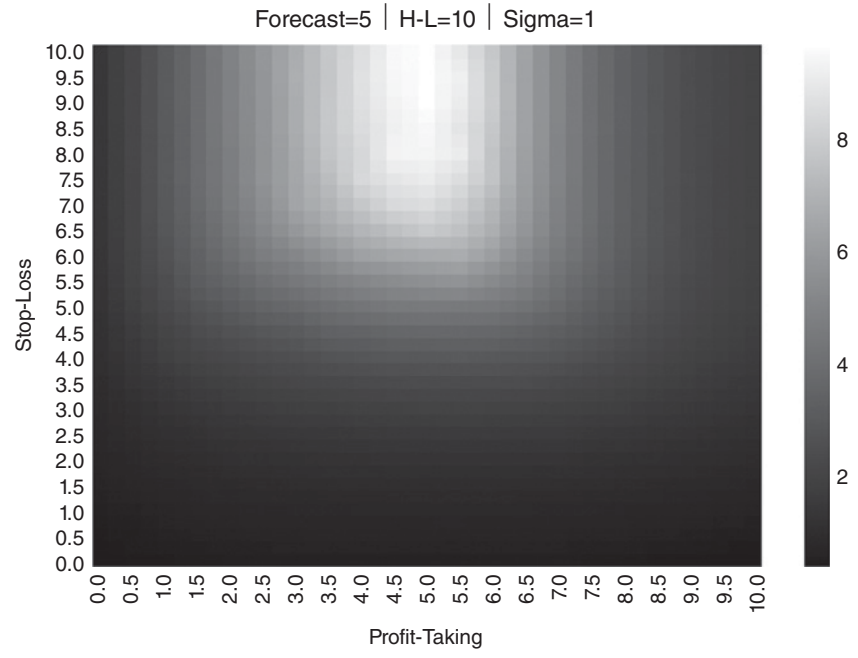


FIGURE 13.7 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{5, 10, 1\}$

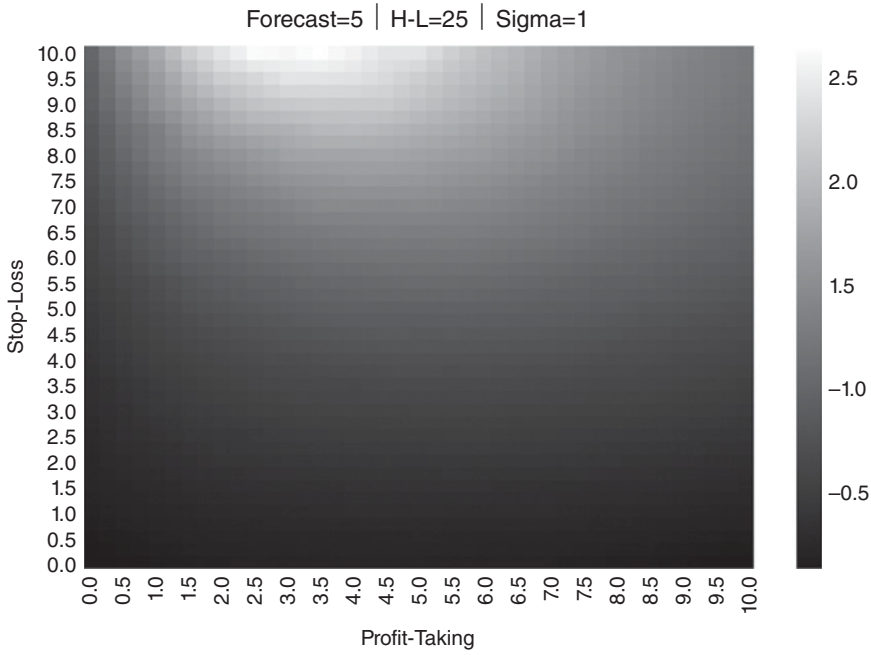


FIGURE 13.8 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{5, 25, 1\}$

area of optimal performance has given way to a semi-circle of small profit-taking with large stop-loss thresholds. Again we see a deterioration of performance, with Sharpe ratios of 2.7.

In Figure 13.9, the half-life is raised to $\tau = 50$. As a result, the region of optimal performance spreads, while Sharpe ratios continue to fall to 0.8. This is the same effect we observed in the case of zero long-run equilibrium (Section 13.6.1), with the difference that because now $\mathrm{E}_0[P_{i,T_i}] > 0$, there is no symmetric area of worst performance.

In Figure 13.10, we appreciate that $\tau = 100$ leads to the natural conclusion of the trend described above. The process is now so close to a random walk that the maximum Sharpe ratio is a mere 0.32.

We can observe a similar pattern in Figures 13.11 through 13.15, where $\mathrm{E}_0[P_{i,T_i}] = 10$ and τ is progressively increased from 5 to 10, 25, 50, and 100, respectively.

13.6.3 Cases with Negative Long-Run Equilibrium

A rational market participant would not initiate a position under the assumption that a loss is the expected outcome. However, if a trader recognizes that losses are the expected outcome of a pre-existing position, she still needs a strategy to stop-out that position while minimizing such losses.

We have obtained Figure 13.16 as a result of applying parameters $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-5, 5, 1\}$. If we compare Figure 13.16 with Figure 13.6, it

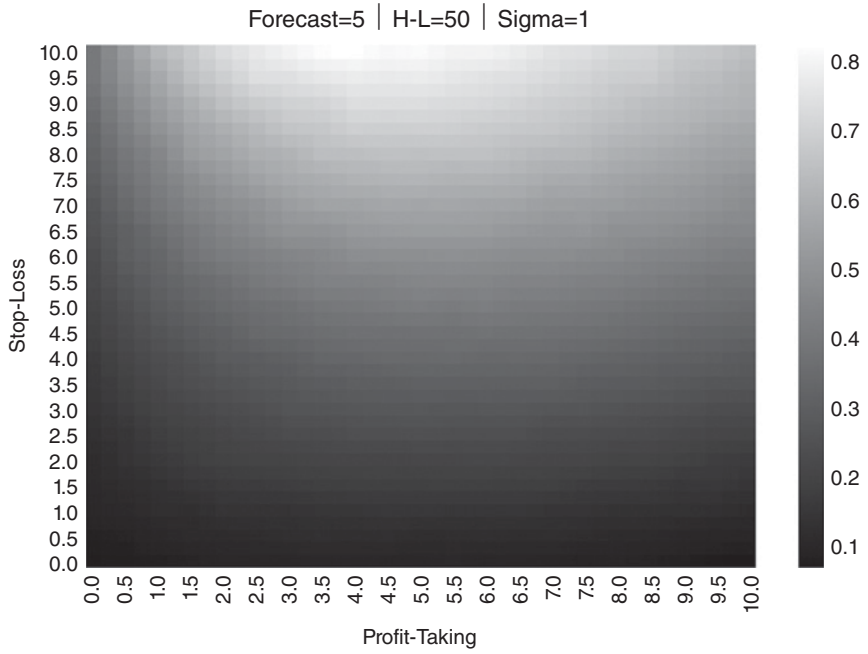


FIGURE 13.9 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{5, 50, 1\}$

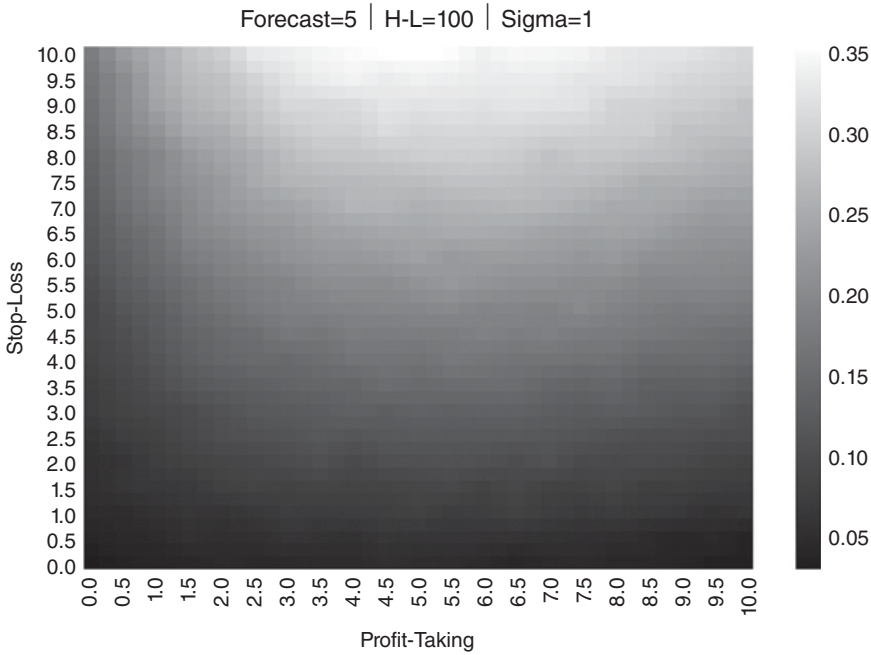


FIGURE 13.10 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{5,100,1\}$

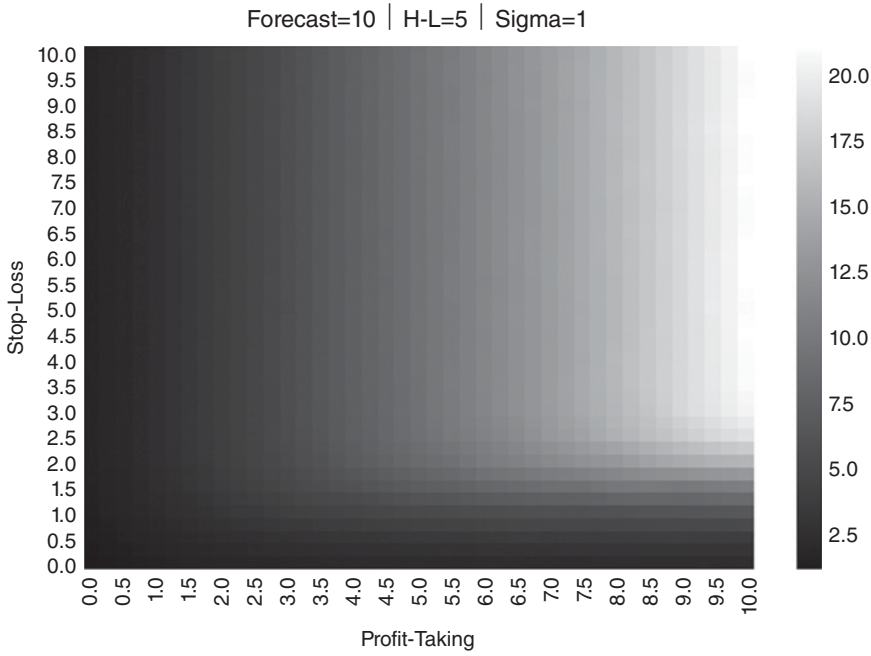


FIGURE 13.11 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{10, 5, 1\}$

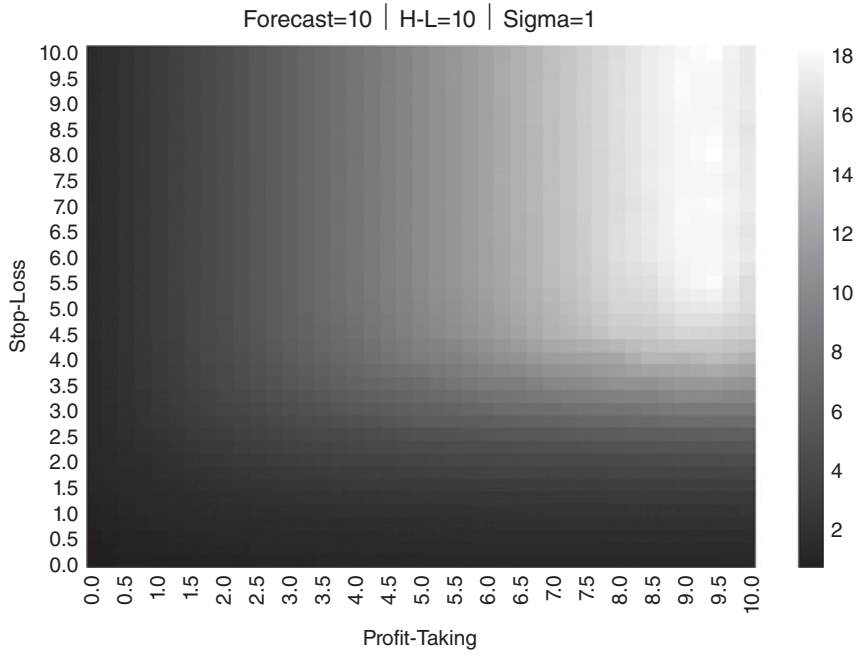


FIGURE 13.12 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{10, 10, 1\}$

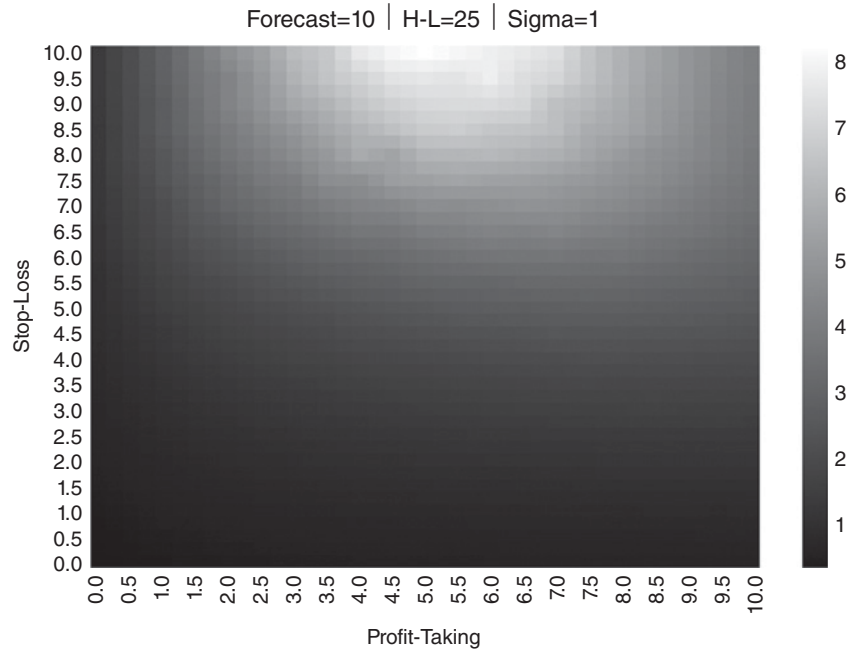


FIGURE 13.13 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{10, 25, 1\}$

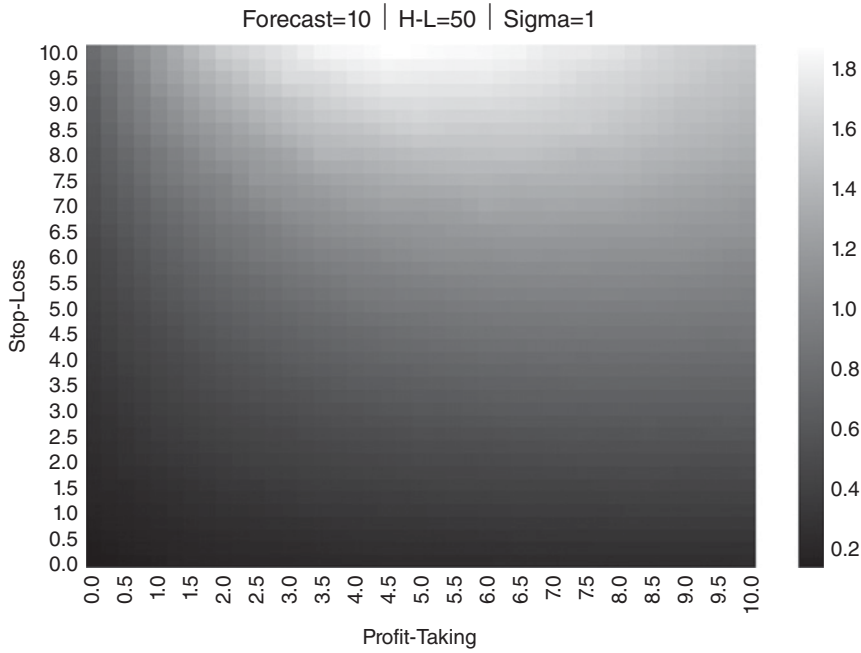


FIGURE 13.14 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{10, 50, 1\}$

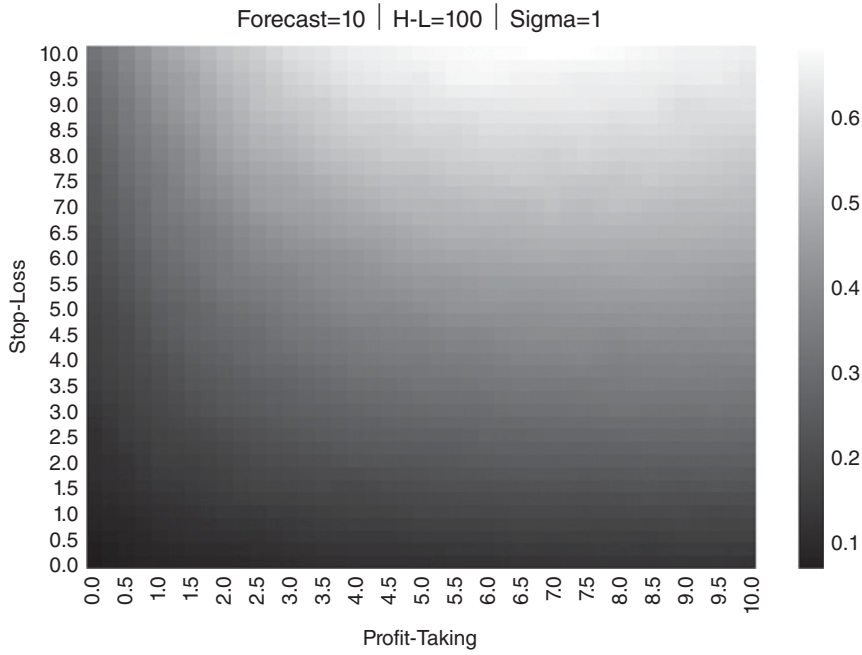


FIGURE 13.15 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \boldsymbol{\sigma}\} = \{10, 100, 1\}$

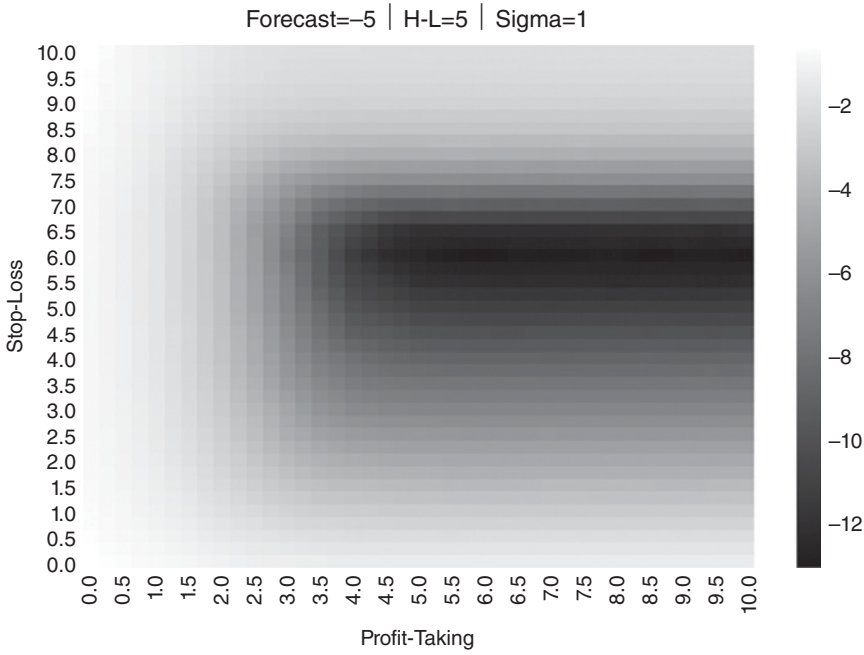


FIGURE 13.16 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-5, 5, 1\}$

appears as if one is a rotated complementary of the other. Figure 13.6 resembles a rotated photographic negative of Figure 13.16. The reason is that the profit in Figure 13.6 is translated into a loss in Figure 13.16, and the loss in Figure 13.6 is translated into a profit in Figure 13.16. One case is a reverse image of the other, just as a gambler's loss is the house's gain.

As expected, Sharpe ratios are negative, with a worst performance region centered around the stop-loss of 6, and profit-taking thresholds that range between 4 and 10. Now the rectangular shape does not correspond to a region of best performance, but to a region of worst performance, with Sharpe ratios of around -12.

In Figure 13.17, $\tau = 10$, and now the proximity to a random walk plays in our favor. The region of worst performance spreads out, and the rectangular area becomes a square. Performance becomes less negative, with Sharpe ratios of about -9.

This familiar progression can be appreciated in Figures 13.18, 13.19, and 13.20, as τ is raised to 25, 50, and 100. Again, as the process approaches a random walk, performance flattens and optimizing the trading rule becomes a backtest-overfitting exercise.

Figures 13.21 through 13.25 repeat the same process for $\mathrm{E}_0[P_{i,T_i}] = -10$ and τ that is progressively increased from 5 to 10, 25, 50, and 100. The same pattern, a rotated complementary to the case of positive long-run equilibrium, arises.

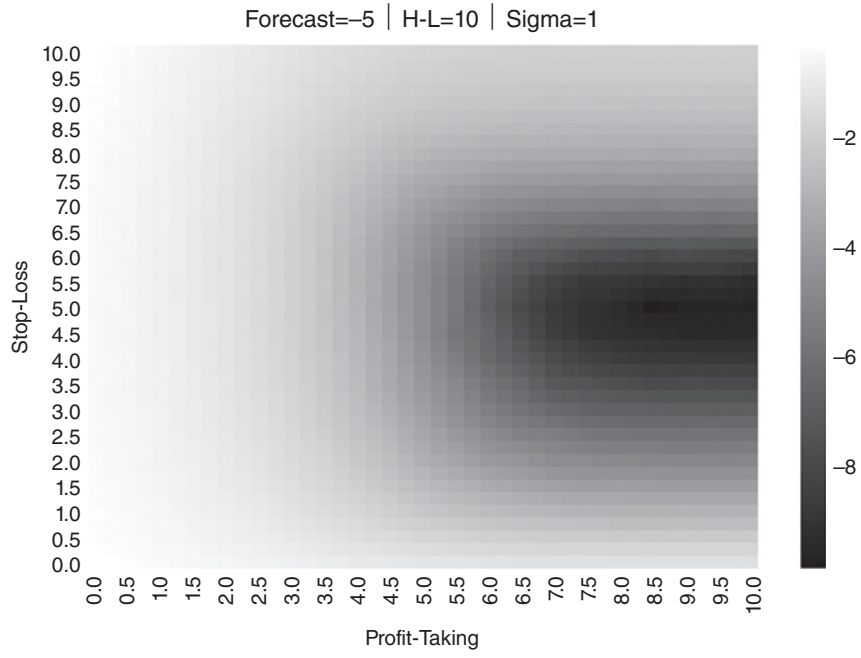


FIGURE 13.17 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-5, 10, 1\}$

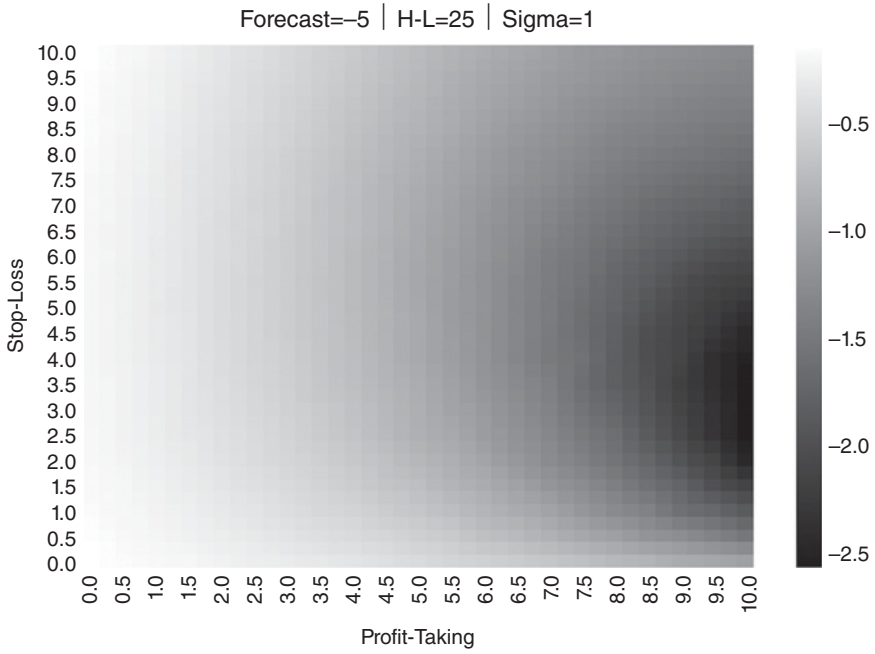


FIGURE 13.18 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-5, 25, 1\}$

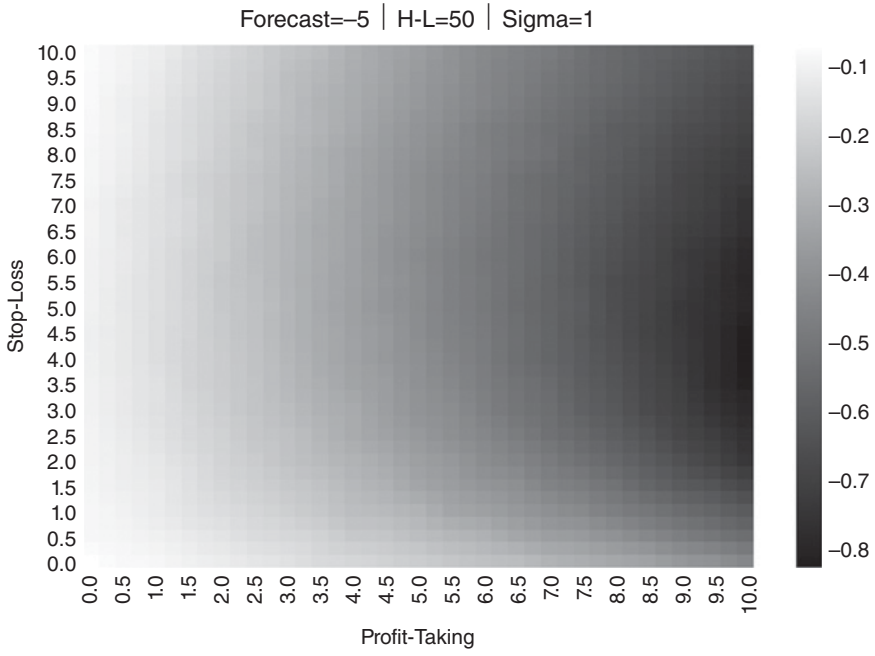


FIGURE 13.19 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-5, 50, 1\}$

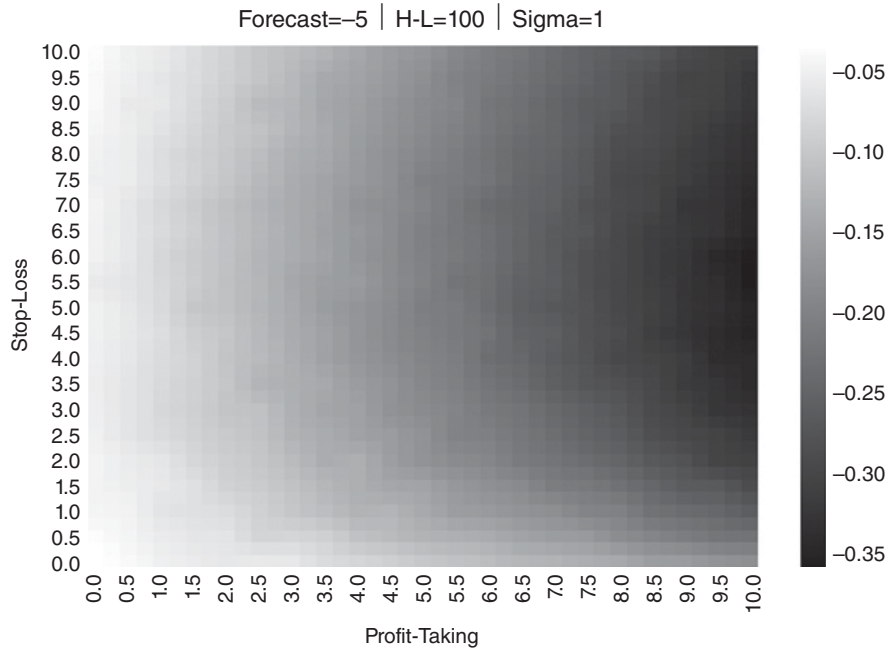


FIGURE 13.20 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-5, 100, 1\}$

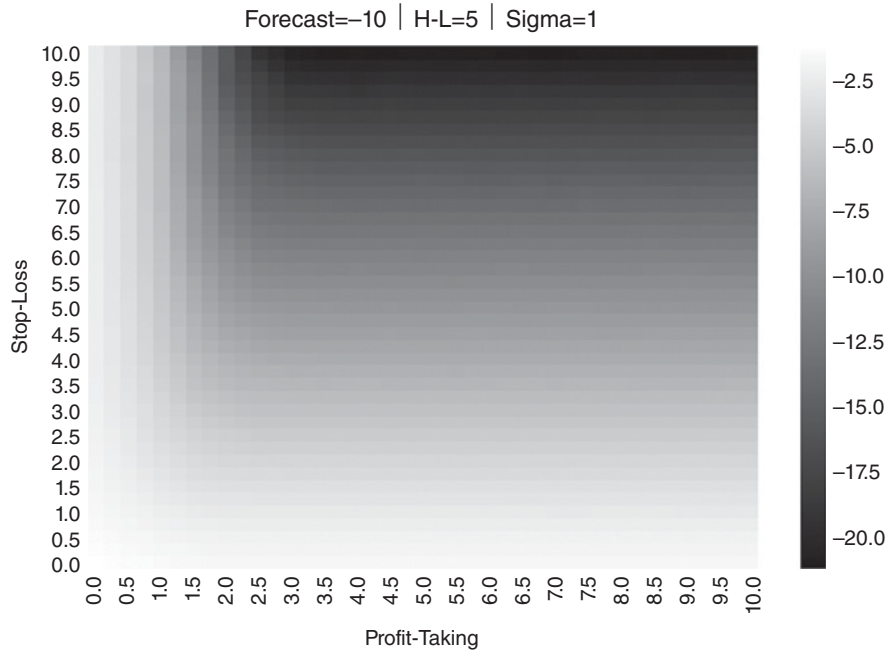


FIGURE 13.21 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-10, 5, 1\}$

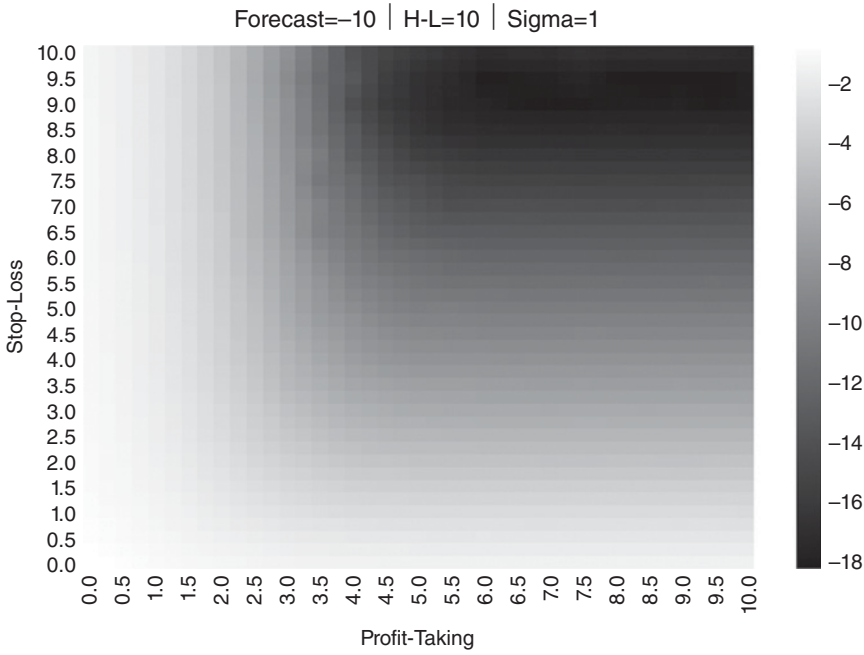


FIGURE 13.22 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-10, 10, 1\}$

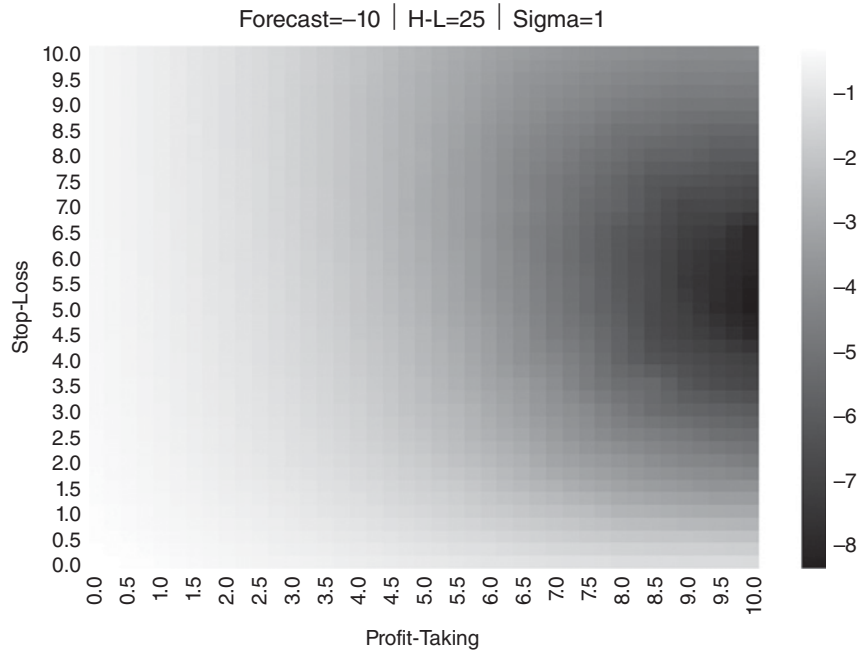


FIGURE 13.23 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-10, 25, 1\}$

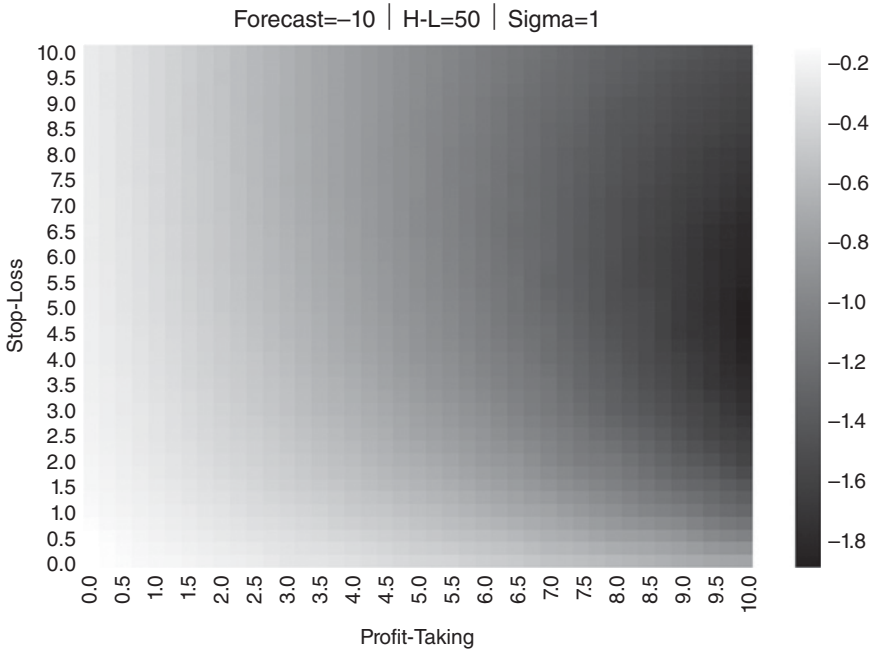


FIGURE 13.24 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-10, 50, 1\}$

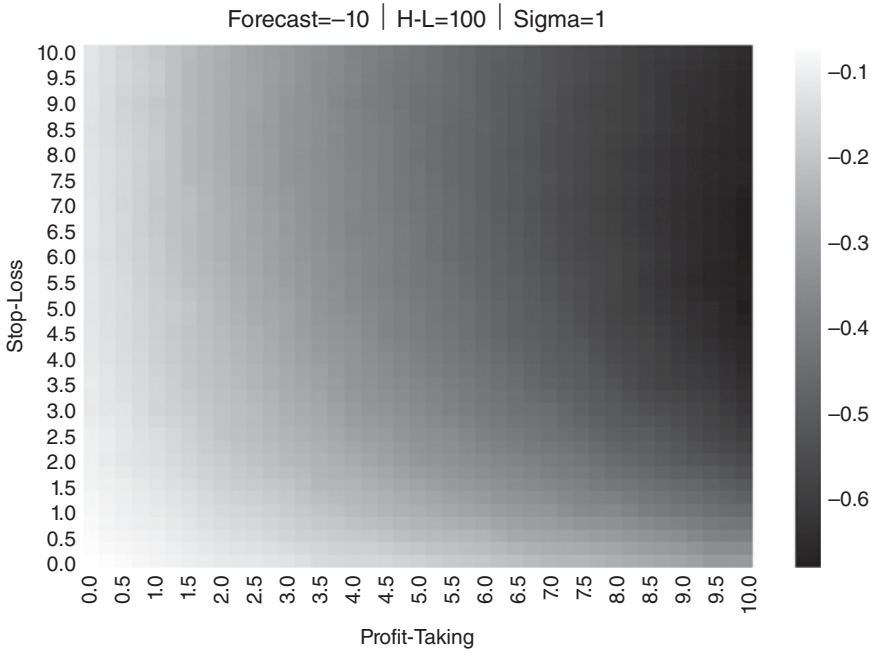


FIGURE 13.25 Heat-map for $\{\mathrm{E}_0[P_{i,T_i}], \tau, \sigma\} = \{-10, 100, 1\}$

13.7 CONCLUSION

In this chapter we have shown how to determine experimentally the optimal trading strategy associated with prices following a discrete O-U process. Because the derivation of such trading strategy is not the result of a historical simulation, our procedure avoids the risks associated with overfitting the backtest to a single path. Instead, the optimal trading rule is derived from the characteristics of the underlying stochastic process that drives prices. The same approach can be applied to processes other than O-U, and we have focused on this particular process only for educational purposes.

While we do not derive the closed-form solution to the optimal trading strategies problem in this chapter, our experimental results seem to support the following OTR conjecture:

> **Conjecture:** Given a financial instrument's price characterized by a discrete O-U process, there is a unique optimal trading rule in terms of a combination of profit-taking and stop-loss that maximizes the rule's Sharpe ratio.

Given that these optimal trading rules can be derived numerically within a few seconds, there is little practical incentive to obtain a closed-form solution. As it is becoming more common in mathematical research, the experimental analysis of a conjecture can help us achieve a goal even in the absence of a proof. It could take years if not decades to prove the above conjecture, and yet all experiments conducted so far confirm it empirically. Let me put it this way: The probability that this conjecture is false is negligible relative to the probability that you will overfit your trading rule by disregarding the conjecture. Hence, the rational course of action is to assume that the conjecture is right, and determine the OTR through synthetic data. In the worst case, the trading rule will be suboptimal, but still it will almost surely outperform an overfit trading rule.

EXERCISES

13.1 Suppose you are an execution trader. A client calls you with an order to cover a short position she entered at a price of 100. She gives you two exit conditions: profit-taking at 90 and stop-loss at 105.

- **(a)** Assuming the client believes the price follows an O-U process, are these levels reasonable? For what parameters?
- **(b)** Can you think of an alternative stochastic process under which these levels make sense?

13.2 Fit the time series of dollar bars of E-mini S&P 500 futures to an O-U process. Given those parameters:

- **(a)** Produce a heat-map of Sharpe ratios for various profit-taking and stop-loss levels.
- **(b)** What is the OTR?

13.3 Repeat exercise 2, this time on a time series of dollar bars of
- **(a)** 10-year U.S. Treasure Notes futures
- **(b)** WTI Crude Oil futures
- **(c)** Are the results significantly different? Does this justify having execution traders specialized by product?

13.4 Repeat exercise 2 after splitting the time series into two parts:
- **(a)** The first time series ends on 3/15/2009.
- **(b)** The second time series starts on 3/16/2009.
- **(c)** Are the OTRs significantly different?

13.5 How long do you estimate it would take to derive OTRs on the 100 most liquid futures contracts worldwide? Considering the results from exercise 4, how often do you think you may have to re-calibrate the OTRs? Does it make sense to pre-compute this data?

13.6 Parallelize Snippets 13.1 and 13.2 using the `mpEngine` module described in Chapter 20.

REFERENCES

Bailey, D. and M. López de Prado (2012): "The Sharpe ratio efficient frontier." *Journal of Risk*, Vol. 15, No. 2, pp. 3–44. Available at http://ssrn.com/abstract=1821643.

Bailey, D. and M. López de Prado (2013): "Drawdown-based stop-outs and the triple penance rule." *Journal of Risk*, Vol. 18, No. 2, pp. 61–93. Available at http://ssrn.com/abstract=2201302.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2014): "Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance." *Notices of the American Mathematical Society*, 61(5), pp. 458–471. Available at http://ssrn.com/abstract=2308659.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2017): "The probability of backtest overfitting." *Journal of Computational Finance*, Vol. 20, No. 4, pp. 39–70. Available at http://ssrn.com/abstract=2326253.

Bertram, W. (2009): "Analytic solutions for optimal statistical arbitrage trading." Working paper. Available at http://ssrn.com/abstract=1505073.

Easley, D., M. Lopez de Prado, and M. O'Hara (2011): "The exchange of flow-toxicity." *Journal of Trading*, Vol. 6, No. 2, pp. 8–13. Available at http://ssrn.com/abstract=1748633.

CHAPTER 14

Backtest Statistics

14.1 MOTIVATION

In the previous chapters, we have studied three backtesting paradigms: First, historical simulations (the walk-forward method, Chapters 11 and 12). Second, scenario simulations (CV and CPCV methods, Chapter 12). Third, simulations on synthetic data (Chapter 13). Regardless of the backtesting paradigm you choose, you need to report the results according to a series of statistics that investors will use to compare and judge your strategy against competitors. In this chapter we will discuss some of the most commonly used performance evaluation statistics. Some of these statistics are included in the Global Investment Performance Standards (GIPS),[1] however a comprehensive analysis of performance requires metrics specific to the ML strategies under scrutiny.

14.2 TYPES OF BACKTEST STATISTICS

Backtest statistics comprise metrics used by investors to assess and compare various investment strategies. They should help us uncover potentially problematic aspects of the strategy, such as substantial asymmetric risks or low capacity. Overall, they can be categorized into general characteristics, performance, runs/drawdowns, implementation shortfall, return/risk efficiency, classification scores, and attribution.

[1] For further details, visit https://www.gipsstandards.org.

14.3 GENERAL CHARACTERISTICS

The following statistics inform us about the general characteristics of the backtest:

- **Time range:** Time range specifies the start and end dates. The period used to test the strategy should be sufficiently long to include a comprehensive number of regimes (Bailey and López de Prado [2012]).
- **Average AUM:** This is the average dollar value of the assets under management. For the purpose of computing this average, the dollar value of long and short positions is considered to be a positive real number.
- **Capacity:** A strategy's capacity can be measured as the highest AUM that delivers a target risk-adjusted performance. A minimum AUM is needed to ensure proper bet sizing (Chapter 10) and risk diversification (Chapter 16). Beyond that minimum AUM, performance will decay as AUM increases, due to higher transaction costs and lower turnover.
- **Leverage:** Leverage measures the amount of borrowing needed to achieve the reported performance. If leverage takes place, costs must be assigned to it. One way to measure leverage is as the ratio of average dollar position size to average AUM.
- **Maximum dollar position size:** Maximum dollar position size informs us whether the strategy at times took dollar positions that greatly exceeded the average AUM. In general we will prefer strategies that take maximum dollar positions close to the average AUM, indicating that they do not rely on the occurrence of extreme events (possibly outliers).
- **Ratio of longs:** The ratio of longs show what proportion of the bets involved long positions. In long-short, market neutral strategies, ideally this value is close to 0.5. If not, the strategy may have a position bias, or the backtested period may be too short and unrepresentative of future market conditions.
- **Frequency of bets:** The frequency of bets is the number of bets per year in the backtest. A sequence of positions on the same side is considered part of the same bet. A bet ends when the position is flattened or flipped to the opposite side. The number of bets is always smaller than the number of trades. A trade count would overestimate the number of independent opportunities discovered by the strategy.
- **Average holding period:** The average holding period is the average number of days a bet is held. High-frequency strategies may hold a position for a fraction of seconds, whereas low frequency strategies may hold a position for months or even years. Short holding periods may limit the capacity of the strategy. The holding period is related but different to the frequency of bets. For example, a strategy may place bets on a monthly basis, around the release of nonfarm payrolls data, where each bet is held for only a few minutes.
- **Annualized turnover:** Annualized turnover measures the ratio of the average dollar amount traded per year to the average annual AUM. High turnover may occur even with a low number of bets, as the strategy may require constant tuning of the position. High turnover may also occur with a low number of

trades, if every trade involves flipping the position between maximum long and maximum short.

- **Correlation to underlying:** This is the correlation between strategy returns and the returns of the underlying investment universe. When the correlation is significantly positive or negative, the strategy is essentially holding or short-selling the investment universe, without adding much value.

Snippet 14.1 lists an algorithm that derives the timestamps of flattening or flipping trades from a pandas series of target positions (`tPos`). This gives us the number of bets that have taken place.

SNIPPET 14.1 DERIVING THE TIMING OF BETS FROM A SERIES OF TARGET POSITIONS

```
# A bet takes place between flat positions or position flips
df0=tPos[tPos==0].index
df1=tPos.shift(1);df1=df1[df1!=0].index
bets=df0.intersection(df1) # flattening
df0=tPos.iloc[1:]*tPos.iloc[:-1].values
bets=bets.union(df0[df0<0].index).sort_values() # tPos flips
if tPos.index[-1] not in bets:bets=bets.append(tPos.index[-1:]) # last bet
```

Snippet 14.2 illustrates the implementation of an algorithm that estimates the average holding period of a strategy, given a pandas series of target positions (`tPos`).

SNIPPET 14.2 IMPLEMENTATION OF A HOLDING PERIOD ESTIMATOR

```
def getHoldingPeriod(tPos):
    # Derive avg holding period (in days) using avg entry time pairing algo
    hp,tEntry=pd.DataFrame(columns=['dT','w']),0.
    pDiff,tDiff=tPos.diff(),(tPos.index-tPos.index[0])/np.timedelta64(1,'D')
    for i in xrange(1,tPos.shape[0]):
        if pDiff.iloc[i]*tPos.iloc[i-1]>=0: # increased or unchanged
            if tPos.iloc[i]!=0:
                tEntry=(tEntry*tPos.iloc[i-1]+tDiff[i]*pDiff.iloc[i])/tPos.iloc[i]
        else: # decreased
            if tPos.iloc[i]*tPos.iloc[i-1]<0: # flip
                hp.loc[tPos.index[i],['dT','w']]=(tDiff[i]-tEntry,abs(tPos.iloc[i-1]))
                tEntry=tDiff[i] # reset entry time
            else:
                hp.loc[tPos.index[i],['dT','w']]=(tDiff[i]-tEntry,abs(pDiff.iloc[i]))
    if hp['w'].sum()>0:hp=(hp['dT']*hp['w']).sum()/hp['w'].sum()
    else:hp=np.nan
    return hp
```

14.4 PERFORMANCE

Performance statistics are dollar and returns numbers without risk adjustments. Some useful performance measurements include:

- **PnL:** The total amount of dollars (or the equivalent in the currency of denomination) generated over the entirety of the backtest, including liquidation costs from the terminal position.
- **PnL from long positions:** The portion of the PnL dollars that was generated exclusively by long positions. This is an interesting value for assessing the bias of long-short, market neutral strategies.
- **Annualized rate of return:** The time-weighted average annual rate of total return, including dividends, coupons, costs, etc.
- **Hit ratio:** The fraction of bets that resulted in a positive PnL.
- **Average return from hits:** The average return from bets that generated a profit.
- **Average return from misses:** The average return from bets that generated a loss.

14.4.1 Time-Weighted Rate of Return

Total return is the rate of return from realized and unrealized gains and losses, including accrued interest, paid coupons, and dividends for the measurement period. GIPS rules calculate time-weighted rate of returns (TWRR), adjusted for external cash flows (CFA Institute [2010]). Periodic and sub-periodic returns are geometrically linked. For periods beginning on or after January 1, 2005, GIPS rules mandate calculating portfolio returns that adjust for daily-weighted external cash flows.

We can compute the TWRR by determining the value of the portfolio at the time of each external cash flow.[2] The TWRR for portfolio i between subperiods $[t-1, t]$ is denoted $r_{i,t}$, with equations

$$
\begin{aligned}
r_{i,t} &= \frac{\pi_{i,t}}{K_{i,t}} \\
\pi_{i,t} &= \sum_{j=1}^{J} [(\Delta P_{j,t} + A_{j,t})\theta_{i,j,t-1} + \Delta\theta_{i,j,t}(P_{j,t} - \overline{P}_{j,t-1})] \\
K_{i,t} &= \sum_{j=1}^{J} \tilde{P}_{j,t-1}\theta_{i,j,t-1} + \max\left\{0, \sum_{j=1}^{J} \overline{\tilde{P}}_{j,t}\Delta\theta_{i,j,t}\right\}
\end{aligned}
$$

[2] External cash flows are assets (cash or investments) that enter or exit a portfolio. Dividend and interest income payments, for example, are not considered external cash flows.

where

- $\pi_{i,t}$ is the mark-to-market (MtM) profit or loss for portfolio i at time t.
- $K_{i,t}$ is the market value of the assets under management by portfolio i through subperiod t. The purpose of including the max $\{.\}$ term is to fund additional purchases (ramp-up).
- $A_{j,t}$ is the interest accrued or dividend paid by one unit of instrument j at time t.
- $P_{j,t}$ is the clean price of security j at time t.
- $\theta_{i,j,t}$ are the holdings of portfolio i on security j at time t.
- $\tilde{P}_{j,t}$ is the dirty price of security j at time t.
- $\overline{P}_{j,t}$ is the average transacted clean price of portfolio i on security j over subperiod t.
- $\overline{\tilde{P}}_{j,t}$ is the average transacted dirty price of portfolio i on security j over subperiod t.

Cash inflows are assumed to occur at the beginning of the day, and cash outflows are assumed to occur at the end of the day. These sub-period returns are then linked geometrically as

$$\varphi_{i,T} = \prod_{t=1}^{T}(1 + r_{i,t})$$

The variable $\varphi_{i,T}$ can be understood as the performance of one dollar invested in portfolio i over its entire life, $t = 1, \ldots, T$. Finally, the annualized rate of return of portfolio i is

$$R_i = (\varphi_{i,T})^{-y_i} - 1$$

where y_i is the number of years elapsed between $r_{i,1}$ and $r_{i,T}$.

14.5 RUNS

Investment strategies rarely generate returns drawn from an IID process. In the absence of this property, strategy returns series exhibit frequent runs. Runs are uninterrupted sequences of returns of the same sign. Consequently, runs increase downside risk, which needs to be evaluated with proper metrics.

14.5.1 Returns Concentration

Given a time series of returns from bets, $\{r_t\}_{t=1,\ldots,T}$, we compute two weight series, w^- and w^+:

$$r^+ = \{r_t | r_t \geq 0\}_{t=1,\ldots,T}$$
$$r^- = \{r_t | r_t < 0\}_{t=1,\ldots,T}$$

$$w^+ = \left\{ r_t^+ \left(\sum_t r_t^+ \right)^{-1} \right\}_{t=1,\ldots,T}$$

$$w^- = \left\{ r_t^- \left(\sum_t r_t^- \right)^{-1} \right\}_{t=1,\ldots,T}$$

Inspired by the Herfindahl-Hirschman Index (HHI), for $||w^+|| > 1$, where $||.||$ is the size of the vector, we define the concentration of positive returns as

$$h^+ \equiv \frac{\sum_t \left(w_t^+\right)^2 - ||w^+||^{-1}}{1 - ||w^+||^{-1}} = \left(\frac{\mathrm{E}\left[\left(r_t^+\right)^2\right]}{\mathrm{E}\left[r_t^+\right]^2} - 1 \right) \left(||r^+|| - 1 \right)^{-1}$$

and the equivalent for concentration of negative returns, for $||w^-|| > 1$, as

$$h^- \equiv \frac{\sum_t \left(w_t^-\right)^2 - ||w^-||^{-1}}{1 - ||w^-||^{-1}} = \left(\frac{\mathrm{E}\left[\left(r_t^-\right)^2\right]}{\mathrm{E}\left[r_t^-\right]^2} - 1 \right) (||r^-|| - 1)^{-1}$$

From Jensen's inequality, we know that $\mathrm{E}[r_t^+]^2 \leq \mathrm{E}[(r_t^+)^2]$. And because $\frac{\mathrm{E}[(r_t^+)^2]}{\mathrm{E}[r_t^+]^2} \leq ||r^+||$, we deduce that $\mathrm{E}[r_t^+]^2 \leq \mathrm{E}[(r_t^+)^2] \leq \mathrm{E}[r_t^+]^2 ||r^+||$, with an equivalent boundary on negative bet returns. These definitions have a few interesting properties:

1. $0 \leq h^+ \leq 1$
2. $h^+ = 0 \Leftrightarrow w_t^+ = ||w^+||^{-1}, \forall t$ (uniform returns)
3. $h^+ = 1 \Leftrightarrow \exists i | w_i^+ = \sum_t w_t^+$ (only one non-zero return)

It is easy to derive a similar expression for the concentration of bets across months, $h[t]$. Snippet 14.3 implements these concepts. Ideally, we are interested in strategies where *bets*' returns exhibit:

- high Sharpe ratio
- high number of bets per year, $||r^+|| + ||r^-|| = T$
- high hit ratio (relatively low $||r^-||$)
- low h^+ (no right fat-tail)
- low h^- (no left fat-tail)
- low $h[t]$ (bets are not concentrated in time)

SNIPPET 14.3 ALGORITHM FOR DERIVING HHI CONCENTRATION

```
rHHIPos=getHHI(ret[ret>=0]) # concentration of positive returns per bet
rHHINeg=getHHI(ret[ret<0]) # concentration of negative returns per bet
tHHI=getHHI(ret.groupby(pd.TimeGrouper(freq='M')).count()) # concentr. bets/month
#---------------------------------------
def getHHI(betRet):
    if betRet.shape[0]<=2:return np.nan
    wght=betRet/betRet.sum()
    hhi=(wght**2).sum()
    hhi=(hhi-betRet.shape[0]**-1)/(1.-betRet.shape[0]**-1)
    return hhi
```

14.5.2 Drawdown and Time under Water

Intuitively, a drawdown (DD) is the maximum loss suffered by an investment between two consecutive high-watermarks (HWMs). The time under water (TuW) is the time elapsed between an HWM and the moment the PnL exceeds the previous maximum PnL. These concepts are best understood by reading Snippet 14.4. This code derives both DD and TuW series from either (1) the series of returns (`dollars=False`) or; (2) the series of dollar performance (`dollar=True`). Figure 14.1 provides an example of DD and TuW.

SNIPPET 14.4 DERIVING THE SEQUENCE OF DD AND TuW

```
def computeDD_TuW(series,dollars=False):
    # compute series of drawdowns and the time under water associated with them
    df0=series.to_frame('pnl')
    df0['hwm']=series.expanding().max()
    df1=df0.groupby('hwm').min().reset_index()
    df1.columns=['hwm','min']
    df1.index=df0['hwm'].drop_duplicates(keep='first').index # time of hwm
    df1=df1[df1['hwm']>df1['min']] # hwm followed by a drawdown
    if dollars:dd=df1['hwm']-df1['min']
    else:dd=1-df1['min']/df1['hwm']
    tuw=((df1.index[1:]-df1.index[:-1])/np.timedelta64(1,'Y')).values# in years
    tuw=pd.Series(tuw,index=df1.index[:-1])
    return dd,tuw
```

14.5.3 Runs Statistics for Performance Evaluation

Some useful measurements of runs statistics include:

- **HHI index on positive returns:** This is `getHHI(ret[ret>=0])` in Snippet 14.3.

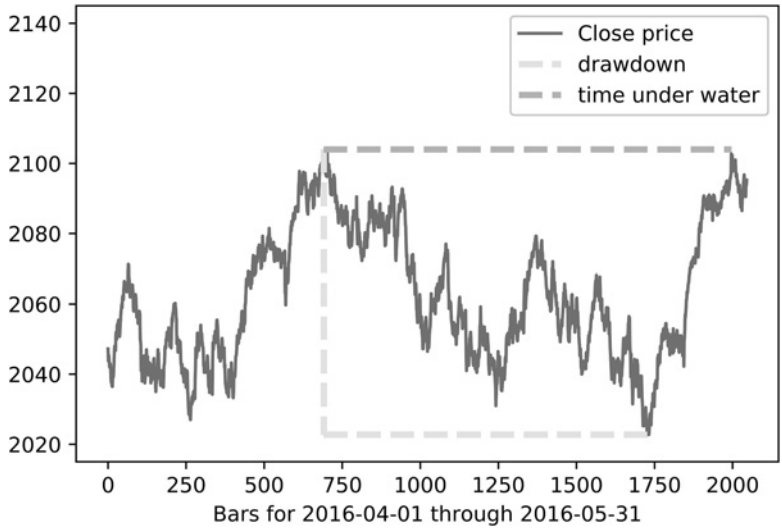


FIGURE 14.1 Examples of drawdown (DD) and time under water + (TuW)

- **HHI index on negative returns:** This is `getHHI(ret[ret < 0])` in Snippet 14.3.
- **HHI index on time between bets:** This is `getHHI(ret.groupby (pd.TimeGrouper (freq='M')).count())` in Snippet 14.3.
- **95-percentile DD:** This is the 95th percentile of the DD series derived by Snippet 14.4.
- **95-percentile TuW:** This is the 95th percentile of the TuW series derived by Snippet 14.4.

14.6 IMPLEMENTATION SHORTFALL

Investment strategies often fail due to wrong assumptions regarding execution costs. Some important measurements of this include:

- **Broker fees per turnover:** These are the fees paid to the broker for turning the portfolio over, including exchange fees.
- **Average slippage per turnover:** These are execution costs, excluding broker fees, involved in one portfolio turnover. For example, it includes the loss caused by buying a security at a fill-price higher than the mid-price at the moment the order was sent to the execution broker.
- **Dollar performance per turnover:** This is the ratio between dollar performance (including brokerage fees and slippage costs) and total portfolio turnovers. It signifies how much costlier the execution could become before the strategy breaks even.

- **Return on execution costs:** This is the ratio between dollar performance (including brokerage fees and slippage costs) and total execution costs. It should be a large multiple, to ensure that the strategy will survive worse-than-expected execution.

14.7 EFFICIENCY

Until now, all performance statistics considered profits, losses, and costs. In this section, we account for the risks involved in achieving those results.

14.7.1 The Sharpe Ratio

Suppose that a strategy's excess returns (in excess of the risk-free rate), $\{r_t\}_{t=1,\ldots,T}$, are IID Gaussian with mean μ and variance σ^2. The Sharpe ratio (SR) is defined as

$$SR = \frac{\mu}{\sigma}$$

The purpose of SR is to evaluate the skills of a particular strategy or investor. Since μ, σ are usually unknown, the true SR value cannot be known for certain. The inevitable consequence is that Sharpe ratio calculations may be the subject of substantial estimation errors.

14.7.2 The Probabilistic Sharpe Ratio

The probabilistic Sharpe ratio (PSR) provides an adjusted estimate of SR, by removing the inflationary effect caused by short series with skewed and/or fat-tailed returns. Given a user-defined benchmark[3] Sharpe ratio (SR^*) and an observed Sharpe ratio $\widehat{SR}$, PSR estimates the probability that $\widehat{SR}$ is greater than a hypothetical SR^*. Following Bailey and López de Prado [2012], PSR can be estimated as

$$\widehat{PSR}\left[SR^*\right] = Z\left[\frac{\left(\widehat{SR} - SR^*\right)\sqrt{T-1}}{\sqrt{1 - \hat{\gamma}_3\widehat{SR} + \frac{\hat{\gamma}_4 - 1}{4}\widehat{SR}^2}}\right]$$

where $Z[.]$ is the cumulative distribution function (CDF) of the standard Normal distribution, T is the number of observed returns, $\hat{\gamma}_3$ is the skewness of the returns, and $\hat{\gamma}_4$ is the kurtosis of the returns ($\hat{\gamma}_4 = 3$ for Gaussian returns). For a given SR^*, $\widehat{PSR}$ increases with greater $\widehat{SR}$ (in the original sampling frequency, i.e. non-annualized), or longer track records (T), or positively skewed returns ($\hat{\gamma}_3$), but it decreases with fatter

[3] This could be set to a default value of zero (i.e., comparing against no investment skill).

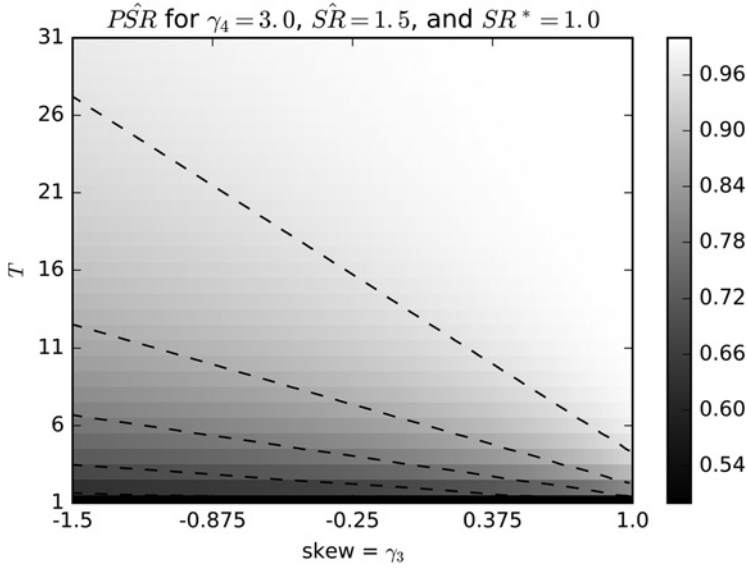


FIGURE 14.2 PSR as a function of skewness and sample length

tails ($\hat{\gamma}_4$). Figure 14.2 plots $\widehat{PSR}$ for $\hat{\gamma}_4 = 3$, $\widehat{SR} = 1.5$ and $SR^* = 1.0$ as a function of $\hat{\gamma}_3$ and T.

14.7.3 The Deflated Sharpe Ratio

The deflated Sharpe ratio (DSR) is a PSR where the rejection threshold is adjusted to reflect the multiplicity of trials. Following Bailey and López de Prado [2014], DSR can be estimated as $\widehat{PSR}[SR^*]$, where the benchmark Sharpe ratio, SR^*, is no longer user-defined. Instead, SR^* is estimated as

$$SR^* = \sqrt{V\left[\left\{\widehat{SR}_n\right\}\right]}\left((1-\gamma)Z^{-1}\left[1-\frac{1}{N}\right]+\gamma Z^{-1}\left[1-\frac{1}{N}e^{-1}\right]\right)$$

where $V[\{\widehat{SR}_n\}]$ is the variance across the trials' estimated SR, N is the number of independent trials, $Z[.]$ is the CDF of the standard Normal distribution, γ is the Euler-Mascheroni constant, and $n = 1, \ldots, N$. Figure 14.3 plots SR^* as a function of $V[\{\widehat{SR}_n\}]$ and N.

The rationale behind DSR is the following: Given a set of SR estimates, $\{\widehat{SR}_n\}$, its expected maximum is greater than zero, even if the true SR is zero. Under the null hypothesis that the actual Sharpe ratio is zero, $H_0 : SR = 0$, we know that the expected maximum $\widehat{SR}$ can be estimated as the SR^*. Indeed, SR^* increases quickly as more independent trials are attempted (N), or the trials involve a greater variance ($V[\{\widehat{SR}_n\}]$). From this knowledge we derive the third law of backtesting.

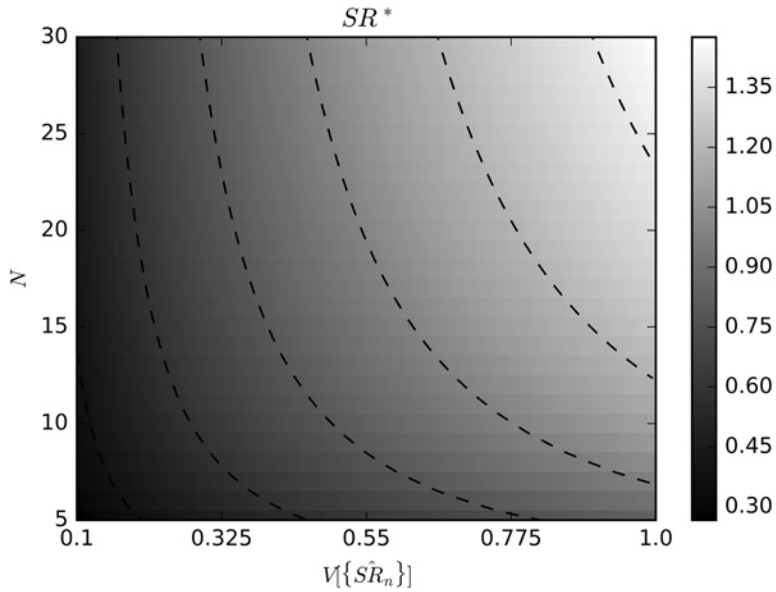


FIGURE 14.3 SR^* as a function of $V[\{\widehat{SR}_n\}]$ and N

SNIPPET 14.5 MARCOS' THIRD LAW OF BACKTESTING. MOST DISCOVERIES IN FINANCE ARE FALSE BECAUSE OF ITS VIOLATION

> "Every backtest result must be reported in conjunction with all the trials involved in its production. Absent that information, it is impossible to assess the backtest's 'false discovery' probability."
>
> —Marcos López de Prado
> *Advances in Financial Machine Learning* (2018)

14.7.4 Efficiency Statistics

Useful efficiency statistics include:

- **Annualized Sharpe ratio:** This is the SR value, annualized by a factor $\sqrt{a}$, where a is the average number of returns observed per year. This common annualization method relies on the assumption that returns are IID.
- **Information ratio:** This is the SR equivalent of a portfolio that measures its performance relative to a benchmark. It is the annualized ratio between the average excess return and the tracking error. The excess return is measured as the portfolio's return in excess of the benchmark's return. The tracking error is estimated as the standard deviation of the excess returns.
- **Probabilistic Sharpe ratio:** PSR corrects SR for inflationary effects caused by non-Normal returns or track record length. It should exceed 0.95, for the

standard significance level of 5%. It can be computed on absolute or relative returns.

- **Deflated Sharpe ratio:** DSR corrects SR for inflationary effects caused by non-Normal returns, track record length, and multiple testing/selection bias. It should exceed 0.95, for the standard significance level of 5%. It can be computed on absolute or relative returns.

14.8 CLASSIFICATION SCORES

In the context of meta-labeling strategies (Chapter 3, Section 3.6), it is useful to understand the performance of the ML overlay algorithm in isolation. Remember that the primary algorithm identifies opportunities, and the secondary (overlay) algorithm decides whether to pursue them or pass. A few useful statistics include:

- **Accuracy:** Accuracy is the fraction of opportunities correctly labeled by the overlay algorithm,

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.
- **Precision:** Precision is the fraction of true positives among the predicted positives,

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** Recall is the fraction of true positives among the positives,

$$recall = \frac{TP}{TP + FN}$$

- **F1**: Accuracy may not be an adequate classification score for meta-labeling applications. Suppose that, after you apply meta-labeling, there are many more negative cases (label '0') than positive cases (label '1'). Under that scenario, a classifier that predicts every case to be negative will achieve high accuracy, even though recall=0 and precision is undefined. The F1 score corrects for that flaw, by assessing the classifier in terms of the (equally weighted) harmonic mean of precision and recall,

$$F_1 = 2\frac{precision \cdot recall}{precision + recall}$$

  As a side note, consider the unusual scenario where, after applying meta-labeling, there are many more positive cases than negative cases. A classifier that predicts all cases to be positive will achieve TN=0 and FN=0, hence accuracy=precision and recall=1. Accuracy will be high, and F1 will not be smaller than accuracy, even though the classifier is not able to discriminate between the observed samples. One solution would be to switch the definitions

of positive and negative cases, so that negative cases are predominant, and then score with F1.

- **Negative log-loss:** Negative log-loss was introduced in Chapter 9, Section 9.4, in the context of hyper-parameter tuning. Please refer to that section for details. The key conceptual difference between accuracy and negative log-loss is that negative log-loss takes into account not only whether our predictions were correct or not, but the probability of those predictions as well.

See Chapter 3, Section 3.7 for a visual representation of precision, recall, and accuracy. Table 14.1 characterizes the four degenerate cases of binary classification. As you can see, the F1 score is not defined in two of those cases. For this reason, when Scikit-learn is asked to compute F1 on a sample with no observed 1s or with no predicted 1s, it will print a warning (`UndefinedMetricWarning`), and set the F1 value to 0.

TABLE 14.1 The Four Degenerate Cases of Binary Classification

Condition	Collapse	Accuracy	Precision	Recall	F1
Observed all 1s	TN=FP=0	=recall	1	[0,1]	[0,1]
Observed all 0s	TP=FN=0	[0,1]	0	NaN	NaN
Predicted all 1s	TN=FN=0	=precision	[0,1]	1	[0,1]
Predicted all 0s	TP=FP=0	[0,1]	NaN	0	NaN

When all observed values are positive (label '1'), there are no true negatives or false positives, thus precision is 1, recall is a positive real number between 0 and 1 (inclusive), and accuracy equals recall. Then, $F_1 = 2\frac{recall}{1+recall} \geq recall$.

When all predicted values are positive (label '1'), there are no true negatives or false negatives, thus precision is a positive real number between 0 and 1 (inclusive), recall is 1, and accuracy equals precision. Then, $F_1 = 2\frac{precision}{1+precision} \geq precision$.

14.9 ATTRIBUTION

The purpose of performance attribution is to decompose the PnL in terms of risk classes. For example, a corporate bond portfolio manager typically wants to understand how much of its performance comes from his exposure to the following risks classes: duration, credit, liquidity, economic sector, currency, sovereign, issuer, etc. Did his duration bets pay off? What credit segments does he excel at? Or should he focus on his issuer selection skills?

These risks are not orthogonal, so there is always an overlap between them. For example, highly liquid bonds tend to have short durations and high credit rating, and are normally issued by large entities with large amounts outstanding, in U.S. dollars. As a result, the sum of the attributed PnLs will not match the total PnL, but at least we will be able to compute the Sharpe ratio (or information ratio) per risk class. Perhaps the most popular example of this approach is Barra's multi-factor method. See Barra [1998, 2013] and Zhang and Rachev [2004] for details.

Of equal interest is to attribute PnL across categories within each class. For example, the duration class could be split between short duration (less than 5 years), medium duration (between 5 and 10 years), and long duration (in excess of 10 years). This PnL attribution can be accomplished as follows: First, to avoid the overlapping problem we referred to earlier, we need to make sure that each member of the investment universe belongs to one and only one category of each risk class at any point in time. In other words, for each risk class, we split the entire investment universe into disjoint partitions. Second, for each risk class, we form one index per risk category. For example, we will compute the performance of an index of short duration bonds, another index of medium duration bonds, and another index of long duration bonds. The weightings for each index are the re-scaled weights of our investment portfolio, so that each index's weightings add up to one. Third, we repeat the second step, but this time we form those risk category indices using the weights from the investment universe (e.g., Markit iBoxx Investment Grade), again re-scaled so that each index's weightings add up to one. Fourth, we compute the performance metrics we discussed earlier in the chapter on each of these indices' returns and excess returns. For the sake of clarity, in this context the excess return of a short duration index is the return using (re-scaled) portfolio weightings (step 2) minus the return using (re-scaled) universe weightings (step 3).

EXERCISES

14.1 A strategy exhibits a high turnover, high leverage, and high number of bets, with a short holding period, low return on execution costs, and a high Sharpe ratio. Is it likely to have large capacity? What kind of strategy do you think it is?

14.2 On the dollar bars dataset for E-mini S&P 500 futures, compute

- **(a)** HHI index on positive returns.
- **(b)** HHI index on negative returns.
- **(c)** HHI index on time between bars.
- **(d)** The 95-percentile DD.
- **(e)** The 95-percentile TuW.
- **(f)** Annualized average return.
- **(g)** Average returns from hits (positive returns).
- **(h)** Average return from misses (negative returns).
- **(i)** Annualized SR.
- **(j)** Information ratio, where the benchmark is the risk-free rate.
- **(k)** PSR.
- **(l)** DSR, where we assume there were 100 trials, and the variance of the trials' SR was 0.5.

14.3 Consider a strategy that is long one futures contract on even years, and is short one futures contract on odd years.

- **(a)** Repeat the calculations from exercise 2.
- **(b)** What is the correlation to the underlying?

14.4 The results from a 2-year backtest are that monthly returns have a mean of 3.6%, and a standard deviation of 0.079%.

(a) What is the SR?

(b) What is the annualized SR?

14.5 Following on exercise 1:

(a) The returns have a skewness of 0 and a kurtosis of 3. What is the PSR?

(b) The returns have a skewness of -2.448 and a kurtosis of 10.164. What is the PSR?

14.6 What would be the PSR from 2.b, if the backtest had been for a length of 3 years?

14.7 A 5-year backtest has an annualized SR of 2.5, computed on daily returns. The skewness is -3 and the kurtosis is 10.

(a) What is the PSR?

(b) In order to find that best result, 100 trials were conducted. The variance of the Sharpe ratios on those trials is 0.5. What is the DSR?

REFERENCES

Bailey, D. and M. López de Prado (2012): "The Sharpe ratio efficient frontier." *Journal of Risk*, Vol. 15, No. 2, pp. 3–44.

Bailey, D. and M. López de Prado (2014): "The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting and non-normality." *Journal of Portfolio Management*, Vol. 40, No. 5. Available at https://ssrn.com/abstract=2460551.

Barra (1998): *Risk Model Handbook: U.S. Equities*, 1st ed. Barra. Available at http://www.alacra.com/alacra/help/barra_handbook_US.pdf.

Barra (2013): *MSCI BARRA Factor Indexes Methodology*, 1st ed. MSCI Barra. Available at https://www.msci.com/eqb/methodology/meth_docs/MSCI_Barra_Factor%20Indices_Methodology_Nov13.pdf.

CFA Institute (2010): "Global investment performance standards." CFA Institute, Vol. 2010, No. 4, February. Available at https://www.gipsstandards.org/.

Zhang, Y. and S. Rachev (2004): "Risk attribution and portfolio performance measurement—An overview." Working paper, University of California, Santa Barbara. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.318.7169.

BIBLIOGRAPHY

American Statistical Society (1999): "Ethical guidelines for statistical practice." Available at http://www.amstat.org/committees/ethics/index.html.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2014): "Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance." *Notices of the American Mathematical Society*, Vol. 61, No. 5. Available at http://ssrn.com/abstract=2308659.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2017): "The probability of backtest overfitting." *Journal of Computational Finance*, Vol. 20, No. 4, pp. 39–70. Available at http://ssrn.com/abstract=2326253.

Bailey, D. and M. López de Prado (2012): "Balanced baskets: A new approach to trading and hedging risks." *Journal of Investment Strategies (Risk Journals)*, Vol. 1, No. 4, pp. 21–62.

Beddall, M. and K. Land (2013): "The hypothetical performance of CTAs." Working paper, Winton Capital Management.

Benjamini, Y. and Y. Hochberg (1995): "Controlling the false discovery rate: A practical and powerful approach to multiple testing." *Journal of the Royal Statistical Society, Series B (Methodological)*, Vol. 57, No. 1, pp. 289–300.

Bennet, C., A. Baird, M. Miller, and G. Wolford (2010): "Neural correlates of interspecies perspective taking in the post-mortem Atlantic salmon: An argument for proper multiple comparisons correction." *Journal of Serendipitous and Unexpected Results*, Vol. 1, No. 1, pp. 1–5.

Bruss, F. (1984): "A unified approach to a class of best choice problems with an unknown number of options." *Annals of Probability*, Vol. 12, No. 3, pp. 882–891.

Dmitrienko, A., A.C. Tamhane, and F. Bretz (2010): *Multiple Testing Problems in Pharmaceutical Statistics*, 1st ed. CRC Press.

Dudoit, S. and M.J. van der Laan (2008): *Multiple Testing Procedures with Applications to Genomics*, 1st ed. Springer.

Fisher, R.A. (1915): "Frequency distribution of the values of the correlation coefficient in samples of an indefinitely large population." *Biometrika (Biometrika Trust)*, Vol. 10, No. 4, pp. 507–521.

Hand, D. J. (2014): *The Improbability Principle*, 1st ed. Scientific American/Farrar, Straus and Giroux.

Harvey, C., Y. Liu, and H. Zhu (2013): ". . . And the cross-section of expected returns." Working paper, Duke University. Available at http://ssrn.com/abstract=2249314.

Harvey, C. and Y. Liu (2014): "Backtesting." Working paper, Duke University. Available at http://ssrn.com/abstract=2345489.

Hochberg Y. and A. Tamhane (1987): *Multiple Comparison Procedures*, 1st ed. John Wiley and Sons.

Holm, S. (1979): "A simple sequentially rejective multiple test procedure." *Scandinavian Journal of Statistics*, Vol. 6, pp. 65–70.

Ioannidis, J.P.A. (2005): "Why most published research findings are false." *PloS Medicine*, Vol. 2, No. 8, pp. 696–701.

Ingersoll, J., M. Spiegel, W. Goetzmann, and I. Welch (2007): "Portfolio performance manipulation and manipulation-proof performance measures." *Review of Financial Studies*, Vol. 20, No. 5, pp. 1504–1546.

Lo, A. (2002): "The statistics of Sharpe ratios." *Financial Analysts Journal*, Vol. 58, No. 4 (July/August), pp. 36–52.

López de Prado M., and A. Peijan (2004): "Measuring loss potential of hedge fund strategies." *Journal of Alternative Investments*, Vol. 7, No. 1 (Summer), pp. 7–31. Available at http://ssrn.com/abstract=641702.

Mertens, E. (2002): "Variance of the IID estimator in Lo (2002)." Working paper, University of Basel.

Roulston, M. and D. Hand (2013): "Blinded by optimism." Working paper, Winton Capital Management.

Schorfheide, F. and K. Wolpin (2012): "On the use of holdout samples for model selection." *American Economic Review*, Vol. 102, No. 3, pp. 477–481.

Sharpe, W. (1966): "Mutual fund performance." *Journal of Business*, Vol. 39, No. 1, pp. 119–138.

Sharpe, W. (1975): "Adjusting for risk in portfolio performance measurement." *Journal of Portfolio Management*, Vol. 1, No. 2 (Winter), pp. 29–34.

Sharpe, W. (1994): "The Sharpe ratio." *Journal of Portfolio Management*, Vol. 21, No. 1 (Fall), pp. 49–58.

Studený M. and Vejnarová J. (1999): "The multiinformation function as a tool for measuring stochastic dependence," in M. I. Jordan, ed., *Learning in Graphical Models*. MIT Press, pp. 261–296.

Wasserstein R., and Lazar N. (2016) "The ASA's statement on p-values: Context, process, and purpose." *American Statistician*, Vol. 70, No. 2, pp. 129–133. DOI: 10.1080/00031305.2016.1154108.

Watanabe S. (1960): "Information theoretical analysis of multivariate correlation." *IBM Journal of Research and Development*, Vol. 4, pp. 66–82.

CHAPTER 15

Understanding Strategy Risk

15.1 MOTIVATION

As we saw in Chapters 3 and 13, investment strategies are often implemented in terms of positions held until one of two conditions are met: (1) a condition to exit the position with profits (profit-taking), or (2) a condition to exit the position with losses (stop-loss). Even when a strategy does not explicitly declare a stop-loss, there is always an implicit stop-loss limit, at which the investor can no longer finance her position (margin call) or bear the pain caused by an increasing unrealized loss. Because most strategies have (implicitly or explicitly) these two exit conditions, it makes sense to model the distribution of outcomes through a binomial process. This in turn will help us understand what combinations of betting frequency, odds, and payouts are uneconomic. The goal of this chapter is to help you evaluate when a strategy is vulnerable to small changes in any of these variables.

15.2 SYMMETRIC PAYOUTS

Consider a strategy that produces n IID bets per year, where the outcome X_i of a bet $i \in [1, n]$ is a profit $\pi > 0$ with probability $\mathrm{P}[X_i = \pi] = p$, and a loss $-\pi$ with probability $\mathrm{P}[X_i = -\pi] = 1 - p$. You can think of p as the precision of a binary classifier where a positive means betting on an opportunity, and a negative means passing on an opportunity: True positives are rewarded, false positives are punished, and negatives (whether true or false) have no payout. Since the betting outcomes $\{X_i\}_{i=1,\ldots,n}$ are independent, we will compute the expected moments per bet. The expected profit from one bet is $\mathrm{E}[X_i] = \pi p + (-\pi)(1 - p) = \pi(2p - 1)$. The variance is $\mathrm{V}[X_i] = \mathrm{E}[X_i^2] - \mathrm{E}[X_i]^2$, where $\mathrm{E}[X_i^2] = \pi^2 p + (-\pi)^2(1 - p) = \pi^2$, thus

$\mathrm{V}[X_i] = \pi^2 - \pi^2(2p-1)^2 = \pi^2[1-(2p-1)^2] = 4\pi^2 p(1-p)$. For n IID bets per year, the annualized Sharpe ratio (θ) is

$$\theta[p,n] = \frac{n\mathrm{E}[X_i]}{\sqrt{n\mathrm{V}[X_i]}} = \underbrace{\frac{2p-1}{2\sqrt{p(1-p)}}}_{\substack{\text{t-value of } p \\ \text{under } H_0 : p = \frac{1}{2}}} \sqrt{n}$$

Note how π cancels out of the above equation, because the payouts are symmetric. Just as in the Gaussian case, $\theta[p,n]$ can be understood as a re-scaled t-value. This illustrates the point that, even for a small $p > \frac{1}{2}$, the Sharpe ratio can be made high for a sufficiently large n. This is the economic basis for high-frequency trading, where p can be barely above .5, and the key to a successful business is to increase n. The Sharpe ratio is a function of precision rather than accuracy, because passing on an opportunity (a negative) is not rewarded or punished directly (although too many negatives may lead to a small n, which will depress the Sharpe ratio toward zero).

For example, for $p = .55$, $\frac{2p-1}{2\sqrt{p(1-p)}} = 0.1005$, and achieving an annualized Sharpe ratio of 2 requires 396 bets per year. Snippet 15.1 verifies this result experimentally. Figure 15.1 plots the Sharpe ratio as a function of precision, for various betting frequencies.

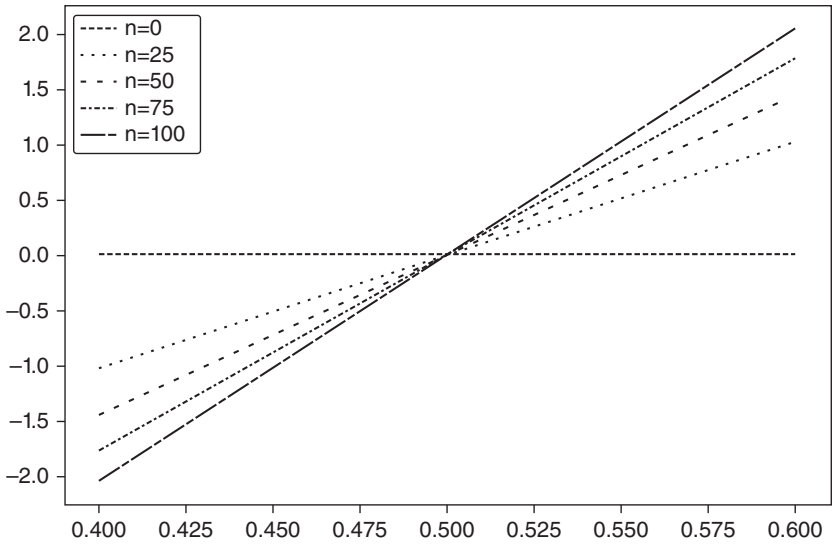


FIGURE 15.1 The relation between precision (x-axis) and sharpe ratio (y-axis) for various bet frequencies (n)

SNIPPET 15.1 TARGETING A SHARPE RATIO AS A FUNCTION OF THE NUMBER OF BETS

```
out,p=[],.55
for i in xrange(1000000):
    rnd=np.random.binomial(n=1,p=p)
    x=(1 if rnd==1 else -1)
    out.append(x)
print np.mean(out),np.std(out),np.mean(out)/np.std(out)
```

Solving for $0 \leq p \leq 1$, we obtain $-4p^2 + 4p - \frac{n}{\theta^2+n} = 0$, with solution

$$p = \frac{1}{2}\left(1 + \sqrt{1 - \frac{n}{\theta^2 + n}}\right)$$

This equation makes explicit the trade-off between precision (p) and frequency (n) for a given Sharpe ratio (θ). For example, a strategy that only produces weekly bets ($n = 52$) will need a fairly high precision of $p = 0.6336$ to deliver an annualized Sharpe of 2.

15.3 ASYMMETRIC PAYOUTS

Consider a strategy that produces n IID bets per year, where the outcome X_i of a bet $i \in [1, n]$ is π_+ with probability $\mathrm{P}[X_i = \pi_+] = p$, and an outcome π_-, $\pi_- < \pi_+$ occurs with probability $\mathrm{P}[X_i = \pi_-] = 1 - p$. The expected profit from one bet is $\mathrm{E}[X_i] = p\pi_+ + (1-p)\pi_- = (\pi_+ - \pi_-)p + \pi_-$. The variance is $\mathrm{V}[X_i] = \mathrm{E}[X_i^2] - \mathrm{E}[X_i]^2$, where $\mathrm{E}[X_i^2] = p\pi_+^2 + (1-p)\pi_-^2 = (\pi_+^2 - \pi_-^2)p + \pi_-^2$, thus $\mathrm{V}[X_i] = (\pi_+ - \pi_-)^2 p(1-p)$. For n IID bets per year, the annualized Sharpe ratio (θ) is

$$\theta[p, n, \pi_-, \pi_+] = \frac{n\mathrm{E}[X_i]}{\sqrt{n\mathrm{V}[X_i]}} = \frac{(\pi_+ - \pi_-)p + \pi_-}{(\pi_+ - \pi_-)\sqrt{p(1-p)}}\sqrt{n}$$

And for $\pi_- = -\pi_+$ we can see that this equation reduces to the symmetric case: $\theta[p, n, -\pi_+, \pi_+] = \frac{2\pi_+ p + \pi_+}{2\pi_+\sqrt{p(1-p)}}\sqrt{n} = \frac{2p-1}{2\sqrt{p(1-p)}}\sqrt{n} = \theta[p, n]$. For example, for $n = 260, \pi_- = -.01, \pi_+ = .005, p = .7$, we get $\theta = 1.173$.

Finally, we can solve the previous equation for $0 \leq p \leq 1$, to obtain

$$p = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

where:

- $a = (n + \theta^2)(\pi_+ - \pi_-)^2$
- $b = [2n\pi_- - \theta^2(\pi_+ - \pi_-)]\left(\pi_+ - \pi_-\right)$
- $c = n\pi_-^2$

As a side note, Snippet 15.2 verifies these symbolic operations using SymPy Live: http://live.sympy.org/.

SNIPPET 15.2 USING THE SymPy LIBRARY FOR SYMBOLIC OPERATIONS

```
>>> from sympy import *
>>> init_printing(use_unicode=False,wrap_line=False,no_global=True)
>>> p,u,d=symbols('p u d')
>>> m2=p*u**2+(1-p)*d**2
>>> m1=p*u+(1-p)*d
>>> v=m2-m1**2
>>> factor(v)
```

The above equation answers the following question: Given a trading rule characterized by parameters $\{\pi_-, \pi_+, n\}$, what is the precision rate p required to achieve a Sharpe ratio of θ^*? For example, for $n = 260, \pi_- = -.01, \pi_+ = .005$, in order to get $\theta = 2$ we require a $p = .72$. Thanks to the large number of bets, a very small change in p (from $p = .7$ to $p = .72$) has propelled the Sharpe ratio from $\theta = 1.173$ to $\theta = 2$. On the other hand, this also tells us that the strategy is vulnerable to small changes in p. Snippet 15.3 implements the derivation of the implied precision. Figure 15.2 displays the implied precision as a function of n and π_-, where $\pi_+ = 0.1$ and $\theta^* = 1.5$. As π_- becomes more negative for a given n, a higher p is required to achieve θ^* for a given π_+. As n becomes smaller for a given π_-, a higher p is required to achieve θ^* for a given π_+.

SNIPPET 15.3 COMPUTING THE IMPLIED PRECISION

```
def binHR(sl,pt,freq,tSR):
    '''
    Given a trading rule characterized by the parameters {sl,pt,freq},
    what's the min precision p required to achieve a Sharpe ratio tSR?
    1) Inputs
    sl: stop loss threshold
    pt: profit taking threshold
    freq: number of bets per year
```

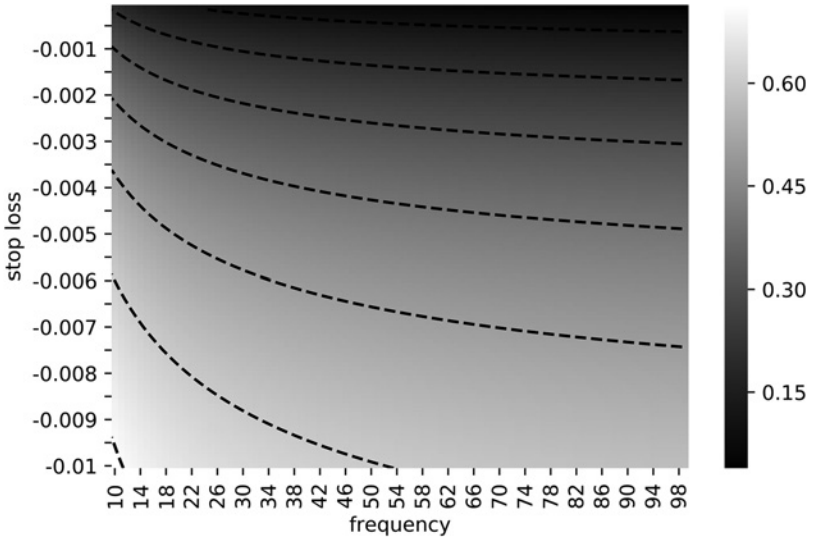


FIGURE 15.2 Heat-map of the implied precision as a function of n and π_-, with $\pi_+ = 0.1$ and $\theta^* = 1.5$

```
    tSR: target annual Sharpe ratio
    2) Output
    p: the min precision rate p required to achieve tSR
    '''
    a=(freq+tSR**2)*(pt-sl)**2
    b=(2*freq*sl-tSR**2*(pt-sl))*(pt-sl)
    c=freq*sl**2
    p=(-b+(b**2-4*a*c)**.5)/(2.*a)
    return p
```

Snippet 15.4 solves $\theta[p, n, \pi_-, \pi_+]$ for the implied betting frequency, n. Figure 15.3 plots the implied frequency as a function of p and π_-, where $\pi_+ = 0.1$ and $\theta^* = 1.5$. As π_- becomes more negative for a given p, a higher n is required to achieve θ^* for a given π_+. As p becomes smaller for a given π_-, a higher n is required to achieve θ^* for a given π_+.

SNIPPET 15.4 COMPUTING THE IMPLIED BETTING FREQUENCY

```
def binFreq(sl,pt,p,tSR):
    '''
    Given a trading rule characterized by the parameters {sl,pt,freq},
    what's the number of bets/year needed to achieve a Sharpe ratio
    tSR with precision rate p?
    Note: Equation with radicals, check for extraneous solution.
```

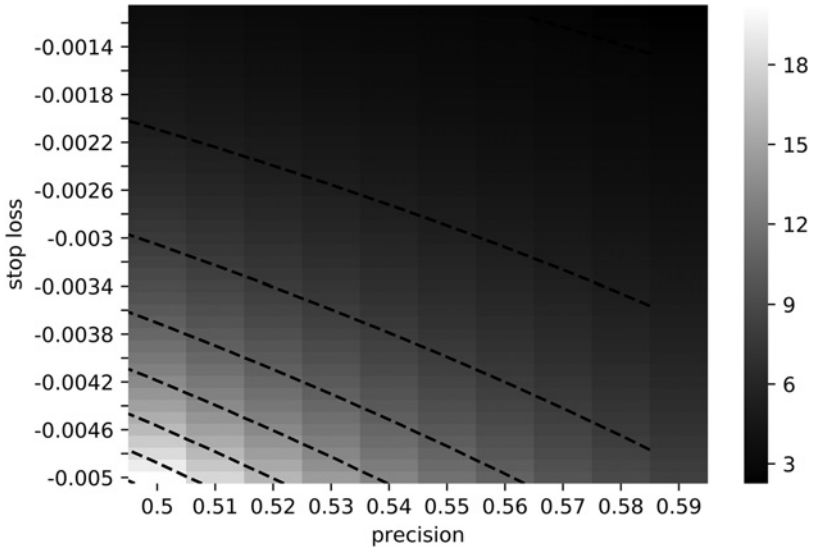


FIGURE 15.3 Implied frequency as a function of p and, with = 0.1 and = 1.5

```
1) Inputs
sl: stop loss threshold
pt: profit taking threshold
p: precision rate p
tSR: target annual Sharpe ratio
2) Output
freq: number of bets per year needed
'''
freq=(tSR*(pt-sl))**2*p*(1-p)/((pt-sl)*p+sl)**2 # possible extraneous
if not np.isclose(binSR(sl,pt,freq,p),tSR):return
return freq
```

15.4 THE PROBABILITY OF STRATEGY FAILURE

In the example above, parameters $\pi_- = -.01, \pi_+ = .005$ are set by the portfolio manager, and passed to the traders with the execution orders. Parameter $n = 260$ is also set by the portfolio manager, as she decides what constitutes an opportunity worth betting on. The two parameters that are not under the control of the portfolio manager are p (determined by the market) and θ^* (the objective set by the investor). Because p is unknown, we can model it as a random variable, with expected value $\mathrm{E}[p]$. Let us define p_{θ^*} as the value of p below which the strategy will underperform a target Sharpe ratio θ^*, that is, $p_{\theta^*} = \max\{p|\theta \leq \theta^*\}$. We can use the equations above (or the `binHR` function) to conclude that for $p_{\theta^*=0} = \frac{2}{3}$, $p < p_{\theta^*=0} \Rightarrow \theta \leq 0$. This highlights

the risks involved in this strategy, because a relatively small drop in p (from $p = .7$ to $p = .67$) will wipe out all the profits. The strategy is intrinsically risky, even if the holdings are not. That is the critical difference we wish to establish with this chapter: *Strategy risk* should not be confused with *portfolio risk.*

Most firms and investors compute, monitor, and report portfolio risk without realizing that this tells us nothing about the risk of the strategy itself. Strategy risk is not the risk of the underlying portfolio, as computed by the chief risk officer. Strategy risk is the risk that the investment strategy will fail to succeed over time, a question of far greater relevance to the chief investment officer. The answer to the question "What is the probability that this strategy will fail?" is equivalent to computing $\mathrm{P}[p < p_{\theta^*}]$. The following algorithm will help us compute the strategy risk.

15.4.1 Algorithm

In this section we will describe a procedure to compute $\mathrm{P}[p < p_{\theta^*}]$. Given a time series of bet outcomes $\{\pi_t\}_{t=1,\dots,T}$, first we estimate $\pi_- = \mathrm{E}[\{\pi_t | \pi_t \leq 0\}_{t=1,\dots,T}]$, and $\pi_+ = \mathrm{E}[\{\pi_t | \pi_t > 0\}_{t=1,\dots,T}]$. Alternatively, $\{\pi_-, \pi_+\}$ could be derived from fitting a mixture of two Gaussians, using the EF3M algorithm (López de Prado and Foreman [2014]). Second, the annual frequency n is given by $n = \frac{T}{y}$, where y is the number of years elapsed between $t = 1$ and $t = T$. Third, we bootstrap the distribution of p as follows:

1. For iterations $i = 1, \dots, I$:
   (a) Draw $\lfloor nk \rfloor$ samples from $\{\pi_t\}_{t=1,\dots,T}$ with replacement, where k is the number of years used by investors to assess a strategy (e.g., 2 years). We denote the set of these drawn samples as $\{\pi_j^{(i)}\}_{j=1,\dots,\lfloor nk \rfloor}$.
   (b) Derive the observed precision from iteration i as $p_i = \frac{1}{\lfloor nk \rfloor} \| \{\pi_j^{(i)} | \pi_j^{(i)} > 0\}_{j=1,\dots,\lfloor nk \rfloor} \|$.
2. Fit the PDF of p, denoted $f[p]$, by applying a Kernel Density Estimator (KDE) on $\{p_i\}_{i=1,\dots,I}$.

For a sufficiently large k, we can approximate this third step as $f[p] \sim N[\bar{p}, \bar{p}(1-\bar{p})]$, where $\bar{p} = \mathrm{E}[p] = \frac{1}{T} \| \{\pi_t^{(i)} | \pi_t^{(i)} > 0\}_{t=1,\dots,T} \|$. Fourth, given a threshold θ^* (the Sharpe ratio that separates failure from success), derive p_{θ^*} (see Section 15.4). Fifth, the strategy risk is computed as $\mathrm{P}[p < p_{\theta^*}] = \int_{-\infty}^{p_{\theta^*}} f[p] dp$.

15.4.2 Implementation

Snippet 15.5 lists one possible implementation of this algorithm. Typically we would disregard strategies where $\mathrm{P}[p < p_{\theta^*}] > .05$ as too risky, even if they invest in low volatility instruments. The reason is that even if they do not lose much money, the probability that they will fail to achieve their target is too high. In order to be deployed, the strategy developer must find a way to reduce p_{θ^*}.

SNIPPET 15.5 CALCULATING THE STRATEGY RISK IN PRACTICE

```
import numpy as np,scipy.stats as ss
#———————————————————————————————————
def mixGaussians(mu1,mu2,sigma1,sigma2,prob1,nObs):
    # Random draws from a mixture of gaussians
    ret1=np.random.normal(mu1,sigma1,size=int(nObs*prob1))
    ret2=np.random.normal(mu2,sigma2,size=int(nObs)-ret1.shape[0])
    ret=np.append(ret1,ret2,axis=0)
    np.random.shuffle(ret)
    return ret
#———————————————————————————————————
def probFailure(ret,freq,tSR):
    # Derive probability that strategy may fail
    rPos,rNeg=ret[ret>0].mean(),ret[ret<=0].mean()
    p=ret[ret>0].shape[0]/float(ret.shape[0])
    thresP=binHR(rNeg,rPos,freq,tSR)
    risk=ss.norm.cdf(thresP,p,p*(1-p)) # approximation to bootstrap
    return risk
#———————————————————————————————————
def main():
    #1) Parameters
    mu1,mu2,sigma1,sigma2,prob1,nObs=.05,-.1,.05,.1,.75,2600
    tSR,freq=2.,260
    #2) Generate sample from mixture
    ret=mixGaussians(mu1,mu2,sigma1,sigma2,prob1,nObs)
    #3) Compute prob failure
    probF=probFailure(ret,freq,tSR)
    print 'Prob strategy will fail',probF
    return
#———————————————————————————————————
if __name__=='__main__':main()
```

This approach shares some similarities with PSR (see Chapter 14, and Bailey and López de Prado [2012, 2014]). PSR derives the probability that the true Sharpe ratio exceeds a given threshold under non-Gaussian returns. Similarly, the method introduced in this chapter derives the strategy's probability of failure based on asymmetric binary outcomes. The key difference is that, while PSR does not distinguish between parameters under or outside the portfolio manager's control, the method discussed here allows the portfolio manager to study the viability of the strategy subject to the parameters under her control: $\{\pi_-, \pi_+, n\}$. This is useful when designing or assessing the viability of a trading strategy.

EXERCISES

15.1 A portfolio manager intends to launch a strategy that targets an annualized SR of 2. Bets have a precision rate of 60%, with weekly frequency. The exit conditions are 2% for profit-taking, and –2% for stop-loss.

(a) Is this strategy viable?

(b) *Ceteris paribus*, what is the required precision rate that would make the strategy profitable?

(c) For what betting frequency is the target achievable?

(d) For what profit-taking threshold is the target achievable?

(e) What would be an alternative stop-loss?

15.2 Following up on the strategy from exercise 1.

(a) What is the sensitivity of SR to a 1% change in each parameter?

(b) Given these sensitivities, and assuming that all parameters are equally hard to improve, which one offers the lowest hanging fruit?

(c) Does changing any of the parameters in exercise 1 impact the others? For example, does changing the betting frequency modify the precision rate, etc.?

15.3 Suppose a strategy that generates monthly bets over two years, with returns following a mixture of two Gaussian distributions. The first distribution has a mean of –0.1 and a standard deviation of 0.12. The second distribution has a mean of 0.06 and a standard deviation of 0.03. The probability that a draw comes from the first distribution is 0.15.

(a) Following López de Prado and Peijan [2004] and López de Prado and Foreman [2014], derive the first four moments for the mixture's returns.

(b) What is the annualized SR?

(c) Using those moments, compute PSR[1] (see Chapter 14). At a 95% confidence level, would you discard this strategy?

15.4 Using Snippet 15.5, compute $\mathrm{P}[p < p_{\theta^*=1}]$ for the strategy described in exercise 3. At a significance level of 0.05, would you discard this strategy? Is this result consistent with $PSR[\theta^*]$?

15.5 In general, what result do you expect to be more accurate, $PSR[\theta^*]$ or $\mathrm{P}[p < p_{\theta^*=1}]$? How are these two methods complementary?

15.6 Re-examine the results from Chapter 13, in light of what you have learned in this chapter.

(a) Does the asymmetry between profit taking and stop-loss thresholds in OTRs make sense?

(b) What is the range of p implied by Figure 13.1, for a daily betting frequency?

(c) What is the range of p implied by Figure 13.5, for a weekly betting frequency?

REFERENCES

Bailey, D. and M. López de Prado (2014): "The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting and non-normality." *Journal of Portfolio Management*, Vol. 40, No. 5. Available at https://ssrn.com/abstract=2460551.

Bailey, D. and M. López de Prado (2012): "The Sharpe ratio efficient frontier." *Journal of Risk*, Vol. 15, No. 2, pp. 3–44. Available at https://ssrn.com/abstract=1821643.

López de Prado, M. and M. Foreman (2014): "A mixture of Gaussians approach to mathematical portfolio oversight: The EF3M algorithm." *Quantitative Finance*, Vol. 14, No. 5, pp. 913–930. Available at https://ssrn.com/abstract=1931734.

López de Prado, M. and A. Peijan (2004): "Measuring loss potential of hedge fund strategies." *Journal of Alternative Investments*, Vol. 7, No. 1 (Summer), pp. 7–31. Available at http://ssrn.com/abstract=641702.

CHAPTER 16

Machine Learning Asset Allocation

16.1 MOTIVATION

This chapter introduces the Hierarchical Risk Parity (HRP) approach.[1] HRP portfolios address three major concerns of quadratic optimizers in general and Markowitz's Critical Line Algorithm (CLA) in particular: instability, concentration, and underperformance. HRP applies modern mathematics (graph theory and machine learning techniques) to build a diversified portfolio based on the information contained in the covariance matrix. However, unlike quadratic optimizers, HRP does not require the invertibility of the covariance matrix. In fact, HRP can compute a portfolio on an ill-degenerated or even a singular covariance matrix, an impossible feat for quadratic optimizers. Monte Carlo experiments show that HRP delivers lower out-of-sample variance than CLA, even though minimum-variance is CLA's optimization objective. HRP produces less risky portfolios out-of-sample compared to traditional risk parity methods. Historical analyses have also shown that HRP would have performed better than standard approaches (Kolanovic et al. [2017], Raffinot [2017]). A practical application of HRP is to determine allocations across multiple machine learning (ML) strategies.

16.2 THE PROBLEM WITH CONVEX PORTFOLIO OPTIMIZATION

Portfolio construction is perhaps the most recurrent financial problem. On a daily basis, investment managers must build portfolios that incorporate their views and forecasts on risks and returns. This is the primordial question that 24-year-old Harry Markowitz attempted to answer more than six decades ago. His monumental insight

[1] A short version of this chapter appeared in the *Journal of Portfolio Management,* Vo1. 42, No. 4, pp. 59–69, Summer of 2016.

was to recognize that various levels of risk are associated with different optimal portfolios in terms of risk-adjusted returns, hence the notion of "efficient frontier" (Markowitz [1952]). One implication is that it is rarely optimal to allocate all assets to the investments with highest expected returns. Instead, we should take into account the correlations across alternative investments in order to build a diversified portfolio.

Before earning his PhD in 1954, Markowitz left academia to work for the RAND Corporation, where he developed the Critical Line Algorithm. CLA is a quadratic optimization procedure specifically designed for inequality-constrained portfolio optimization problems. This algorithm is notable in that it guarantees that the exact solution is found after a known number of iterations, and that it ingeniously circumvents the Karush-Kuhn-Tucker conditions (Kuhn and Tucker [1951]). A description and open-source implementation of this algorithm can be found in Bailey and López de Prado [2013]. Surprisingly, most financial practitioners still seem unaware of CLA, as they often rely on generic-purpose quadratic programming methods that do not guarantee the correct solution or a stopping time.

Despite of the brilliance of Markowitz's theory, a number of practical problems make CLA solutions somewhat unreliable. A major caveat is that small deviations in the forecasted returns will cause CLA to produce very different portfolios (Michaud [1998]). Given that returns can rarely be forecasted with sufficient accuracy, many authors have opted for dropping them altogether and focusing on the covariance matrix. This has led to risk-based asset allocation approaches, of which "risk parity" is a prominent example (Jurczenko [2015]). Dropping the forecasts on returns improves but does not prevent the instability issues. The reason is that quadratic programming methods require the inversion of a positive-definite covariance matrix (all eigenvalues must be positive). This inversion is prone to large errors when the covariance matrix is numerically ill-conditioned, that is, when it has a high condition number (Bailey and López de Prado [2012]).

16.3 MARKOWITZ'S CURSE

The condition number of a covariance, correlation (or normal, thus diagonalizable) matrix is the absolute value of the ratio between its maximal and minimal (by moduli) eigenvalues. Figure 16.1 plots the sorted eigenvalues of several correlation matrices, where the condition number is the ratio between the first and last values of each line. This number is lowest for a diagonal correlation matrix, which is its own inverse. As we add correlated (multicollinear) investments, the condition number grows. At some point, the condition number is so high that numerical errors make the inverse matrix too unstable: A small change on any entry will lead to a very different inverse. This is Markowitz's curse: The more correlated the investments, the greater the need for diversification, and yet the more likely we will receive unstable solutions. The benefits of diversification often are more than offset by estimation errors.

Increasing the size of the covariance matrix will only make matters worse, as each covariance coefficient is estimated with fewer degrees of freedom. In general, we need at least $\frac{1}{2}N(N+1)$ independent and identically distributed (IID) observations in order

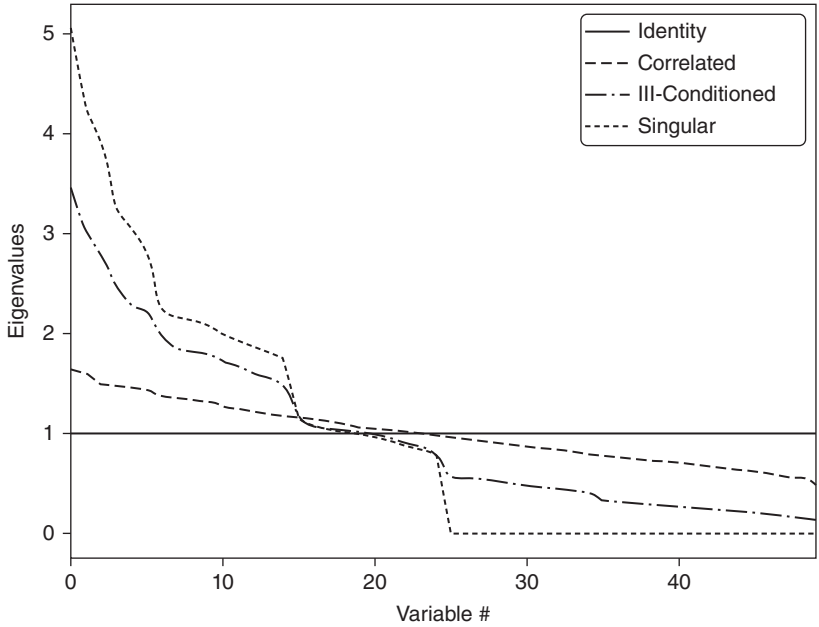


FIGURE 16.1 Visualization of Markowitz's curse
A diagonal correlation matrix has the lowest condition number. As we add correlated investments, the maximum eigenvalue is greater and the minimum eigenvalue is lower. The condition number rises quickly, leading to unstable inverse correlation matrices. At some point, the benefits of diversification are more than offset by estimation errors.

to estimate a covariance matrix of size N that is not singular. For example, estimating an invertible covariance matrix of size 50 requires, at the very least, 5 years of daily IID data. As most investors know, correlation structures do not remain invariant over such long periods by any reasonable confidence level. The severity of these challenges is epitomized by the fact that even naïve (equally-weighted) portfolios have been shown to beat mean-variance and risk-based optimization out-of-sample (De Miguel et al. [2009]).

16.4 FROM GEOMETRIC TO HIERARCHICAL RELATIONSHIPS

These instability concerns have received substantial attention in recent years, as Kolm et al. [2014] have carefully documented. Most alternatives attempt to achieve robustness by incorporating additional constraints (Clarke et al. [2002]), introducing Bayesian priors (Black and Litterman [1992]), or improving the numerical stability of the covariance matrix's inverse (Ledoit and Wolf [2003]).

All the methods discussed so far, although published in recent years, are derived from (very) classical areas of mathematics: geometry, linear algebra, and calculus. A correlation matrix is a linear algebra object that measures the cosines of the angles

between any two vectors in the vector space formed by the returns series (see Calkin and López de Prado [2014a, 2015b]). One reason for the instability of quadratic optimizers is that the vector space is modelled as a complete (fully connected) graph, where every node is a potential candidate to substitute another. In algorithmic terms, inverting the matrix means evaluating the partial correlations across the complete graph. Figure 16.2(a) visualizes the relationships implied by a covariance matrix of 50×50, that is 50 nodes and 1225 edges. This complex structure magnifies small estimation errors, leading to incorrect solutions. Intuitively, it would be desirable to drop unnecessary edges.

Let us consider for a moment the practical implications of such a topological structure. Suppose that an investor wishes to build a diversified portfolio of securities, including hundreds of stocks, bonds, hedge funds, real estate, private placements, etc. Some investments seem closer substitutes of one another, and other investments seem complementary to one another. For example, stocks could be grouped in terms of liquidity, size, industry, and region, where stocks within a given group compete for allocations. In deciding the allocation to a large publicly traded U.S. financial stock like J. P. Morgan, we will consider adding or reducing the allocation to another large publicly traded U.S. bank like Goldman Sachs, rather than a small community bank in Switzerland, or a real estate holding in the Caribbean. Yet, to a correlation matrix, all investments are potential substitutes to one another. In other words, correlation matrices lack the notion of *hierarchy.* This lack of hierarchical structure allows weights to vary freely in unintended ways, which is a root cause of CLA's instability. Figure 16.2(b) visualizes a hierarchical structure known as a tree. A tree structure introduces two desirable features: (1) It has only $N - 1$ edges to connect N nodes, so the weights only rebalance among peers at various hierarchical levels; and (2) the weights are distributed top-down, consistent with how many asset managers build their portfolios (e.g., from asset class to sectors to individual securities). For these reasons, hierarchical structures are better designed to give not only stable but also intuitive results.

In this chapter we will study a new portfolio construction method that addresses CLA's pitfalls using modern mathematics: graph theory and machine learning. This Hierarchical Risk Parity method uses the information contained in the covariance matrix without requiring its inversion or positive-definitiveness. HRP can even compute a portfolio based on a singular covariance matrix. The algorithm operates in three stages: tree clustering, quasi-diagonalization, and recursive bisection.

16.4.1 Tree Clustering

Consider a *TxN* matrix of observations X, such as returns series of N variables over T periods. We would like to combine these N column-vectors into a hierarchical structure of clusters, so that allocations can flow downstream through a tree graph.

First, we compute an *NxN* correlation matrix with entries $\rho = \{\rho_{i,j}\}_{i,j=1,\ldots,N}$, where $\rho_{i,j} = \rho[X_i, X_j]$. We define the distance measure $d : (X_i, X_j) \subset B \to \mathbb{R} \in [0, 1]$, $d_{i,j} = d[X_i, X_j] = \sqrt{\frac{1}{2}(1 - \rho_{i,j})}$, where B is the Cartesian product of items

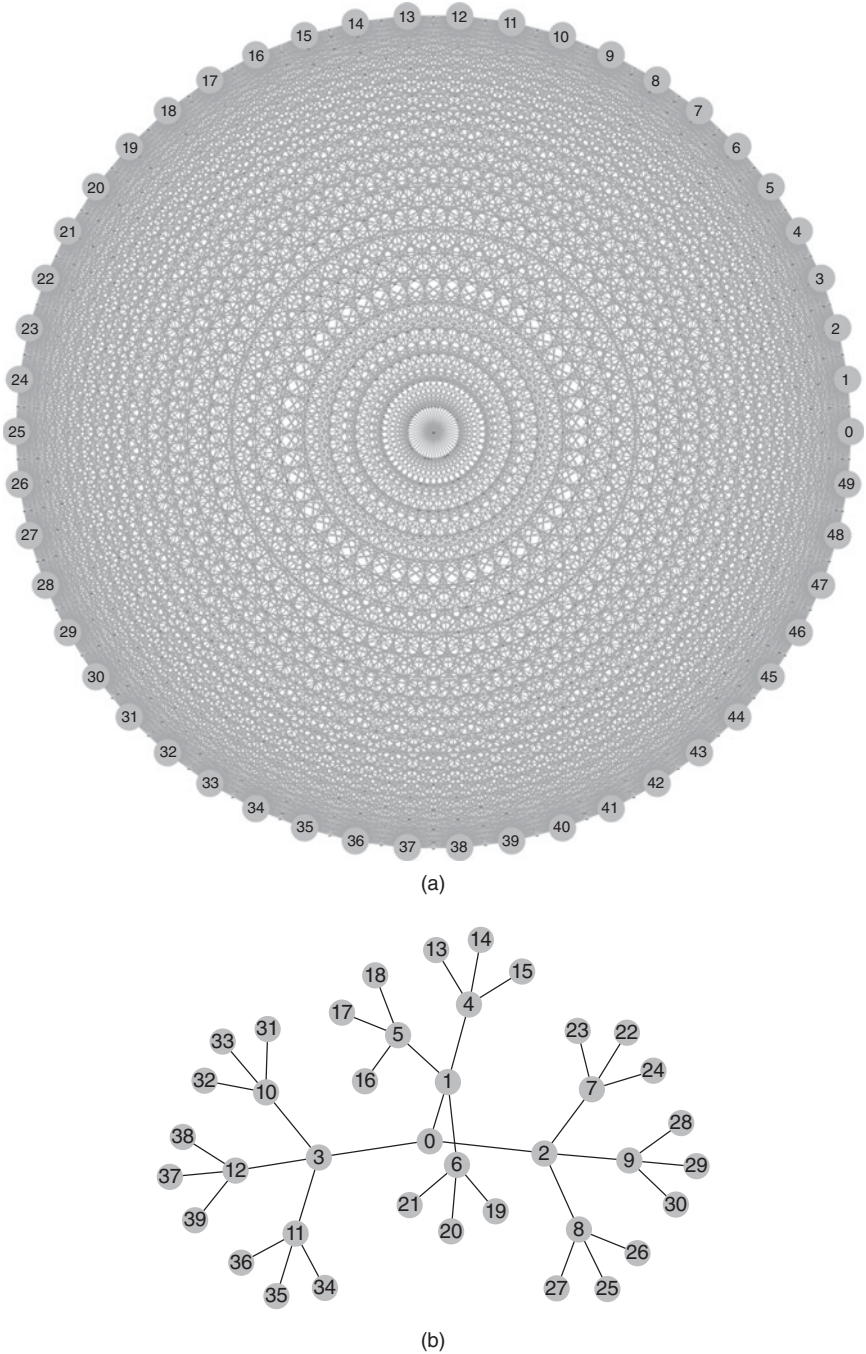


FIGURE 16.2 The complete-graph (top) and the tree-graph (bottom) structures
Correlation matrices can be represented as complete graphs, which lack the notion of hierarchy: Each investment is substitutable with another. In contrast, tree structures incorporate hierarchical relationships.

in $\{1,\ldots,i,\ldots,N\}$. This allows us to compute an NxN distance matrix $D = \{d_{i,j}\}_{i,j=1,\ldots,N}$. Matrix D is a proper metric space (see Appendix 16.A.1 for a proof), in the sense that $d[x,y] \geq 0$ (non-negativity), $d[x,y] = 0 \Leftrightarrow X = Y$ (coincidence), $d[x,y] = d[Y,X]$ (symmetry), and $d[X,Z] \leq d[x,y] + d[Y,Z]$ (sub-additivity). See Example 16.1.

$$\{\rho_{i,j}\} = \begin{bmatrix} 1 & .7 & .2 \\ .7 & 1 & -.2 \\ .2 & -.2 & 1 \end{bmatrix} \rightarrow \{d_{i,j}\} = \begin{bmatrix} 0 & .3873 & .6325 \\ .3873 & 0 & .7746 \\ .6325 & .7746 & 0 \end{bmatrix}$$

Example 16.1 Encoding a correlation matrix ρ as a distance matrix D

Second, we compute the Euclidean distance between any two column-vectors of D, $\tilde{d} : (D_i, D_j) \subset B \rightarrow \mathbb{R} \in [0, \sqrt{N}]$, $\tilde{d}_{i,j} = \tilde{d}[D_i, D_j] = \sqrt{\sum_{n=1}^{N} (d_{n,i} - d_{n,j})^2}$. Note the difference between distance metrics $d_{i,j}$ and $\tilde{d}_{i,j}$. Whereas $d_{i,j}$ is defined on column-vectors of X, $\tilde{d}_{i,j}$ is defined on column-vectors of D (a distance of distances). Therefore, $\tilde{d}$ is a distance defined over the entire metric space D, as each $\tilde{d}_{i,j}$ is a function of the entire correlation matrix (rather than a particular cross-correlation pair). See Example 16.2.

$$\{d_{i,j}\} = \begin{bmatrix} 0 & .3873 & .6325 \\ .3873 & 0 & .7746 \\ .6325 & .7746 & 0 \end{bmatrix} \rightarrow \{\tilde{d}_{i,j}\}_{i,j=\{1,2,3\}} = \begin{bmatrix} 0 & .5659 & .9747 \\ .5659 & 0 & 1.1225 \\ .9747 & 1.1225 & 0 \end{bmatrix}$$

Example 16.2 Euclidean distance of correlation distances

Third, we cluster together the pair of columns (i^*, j^*) such that $(i^*, j^*) = \text{argmin}(i,j)_{i \neq j}\{\tilde{d}_{i,j}\}$, and denote this cluster as $u[1]$. See Example 16.3.

$$\{\tilde{d}_{i,j}\}_{i,j=\{1,2,3\}} = \begin{bmatrix} 0 & .5659 & .9747 \\ .5659 & 0 & 1.1225 \\ .9747 & 1.1225 & 0 \end{bmatrix} \rightarrow u[1] = (1,2)$$

Example 16.3 Clustering items

Fourth, we need to define the distance between a newly formed cluster $u[1]$ and the single (unclustered) items, so that $\{\tilde{d}_{i,j}\}$ may be updated. In hierarchical clustering analysis, this is known as the "linkage criterion." For example, we can define the

distance between an item i of $\tilde{d}$ and the new cluster $u[1]$ as $\dot{d}_{i,u[1]} = \min[\{\tilde{d}_{i,j}\}_{j \in u[1]}]$ (the nearest point algorithm). See Example 16.4.

$$u[1] = (1,2) \rightarrow \{\dot{d}_{i,u[1]}\} = \begin{bmatrix} \min[0, .5659] \\ \min[.5659, 0] \\ \min[.9747, 1.1225] \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ .9747 \end{bmatrix}$$

Example 16.4 Updating matrix $\{\tilde{d}_{i,j}\}$ with the new cluster u

Fifth, matrix $\{\tilde{d}_{i,j}\}$ is updated by appending $\dot{d}_{i,u[1]}$ and dropping the clustered columns and rows $j \in u[1]$. See Example 16.5.

$$\{\tilde{d}_{i,j}\}_{i,j=\{1,2,3,4\}} = \begin{bmatrix} 0 & .5659 & .9747 & 0 \\ .5659 & 0 & 1.1225 & 0 \\ .9747 & 1.1225 & 0 & .9747 \\ 0 & 0 & .9747 & 0 \end{bmatrix}$$

$$\{\tilde{d}_{i,j}\}_{i,j=\{3,4\}} = \begin{bmatrix} 0 & .9747 \\ .9747 & 0 \end{bmatrix}$$

Example 16.5 Updating matrix $\{\tilde{d}_{i,j}\}$ with the new cluster u

Sixth, applied recursively, steps 3, 4, and 5 allow us to append $N - 1$ such clusters to matrix D, at which point the final cluster contains all of the original items, and the clustering algorithm stops. See Example 16.6.

$$\{\tilde{d}_{i,j}\}_{i,j=\{3,4\}} = \begin{bmatrix} 0 & .9747 \\ .9747 & 0 \end{bmatrix} \rightarrow u[2] = (3,4) \rightarrow \text{Stop}$$

Example 16.6 Recursion in search of remaining clusters

Figure 16.3 displays the clusters formed at each iteration for this example, as well as the distances $\tilde{d}_{i^* j^*}$ that triggered every cluster (third step). This procedure can be applied to a wide array of distance metrics $d_{i,j}$, $\tilde{d}_{i,j}$ and $\dot{d}_{i,u}$, beyond those illustrated in this chapter. See Rokach and Maimon [2005] for alternative metrics, the discussion on Fiedler's vector and Stewart's spectral clustering method in Brualdi [2010], as

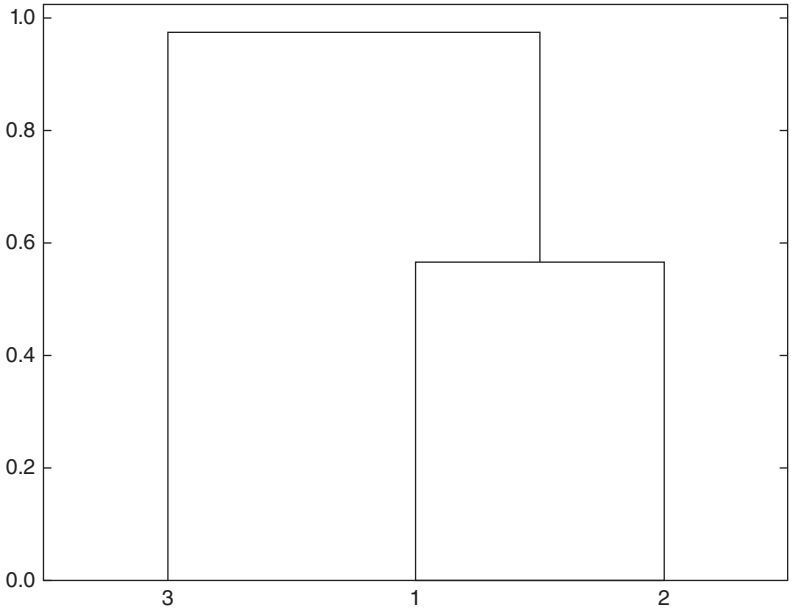


FIGURE 16.3 Sequence of cluster formation
A tree structure derived from our numerical example, here plotted as a dendogram. The y-axis measures the distance between the two merging leaves.

well as algorithms in the scipy library.[2] Snippet 16.1 provides an example of tree clustering using scipy functionality.

SNIPPET 16.1 TREE CLUSTERING USING SCIPY FUNCTIONALITY

```
import scipy.cluster.hierarchy as sch
import numpy as np
import pandas as pd
cov,corr=x.cov(),x.corr()
dist=((1-corr)/2.)**.5 # distance matrix
link=sch.linkage(dist,'single') # linkage matrix
```

This stage allows us to define a linkage matrix as an $(N-1)x4$ matrix with structure $Y = \{(y_{m,1}, y_{m,2}, y_{m,3}, y_{m,4})\}_{m=1,\ldots,N-1}$ (i.e., with one 4-tuple per cluster). Items $(y_{m,1}, y_{m,2})$ report the constituents. Item $y_{m,3}$ reports the distance between $y_{m,1}$ and

[2] For additional metrics see:

http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html

http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.cluster.hierarchy.linkage.html

$y_{m,2}$, that is $y_{m,3} = \tilde{d}_{y_{m,1},y_{m,2}}$. Item $y_{m,4} \leq N$ reports the number of original items included in cluster m.

16.4.2 Quasi-Diagonalization

This stage reorganizes the rows and columns of the covariance matrix, so that the largest values lie along the diagonal. This quasi-diagonalization of the covariance matrix (without requiring a change of basis) renders a useful property: Similar investments are placed together, and dissimilar investments are placed far apart (see Figures 16.5 and 16.6 for an example). The algorithm works as follows: We know that each row of the linkage matrix merges two branches into one. We replace clusters in $(y_{N-1,1}, y_{N-1,2})$ with their constituents recursively, until no clusters remain. These replacements preserve the order of the clustering. The output is a sorted list of original (unclustered) items. This logic is implemented in Snippet 16.2.

SNIPPET 16.2 QUASI-DIAGONALIZATION

```
def getQuasiDiag(link):
    # Sort clustered items by distance
    link=link.astype(int)
    sortIx=pd.Series([link[-1,0],link[-1,1]])
    numItems=link[-1,3] # number of original items
    while sortIx.max()>=numItems:
        sortIx.index=range(0,sortIx.shape[0]*2,2) # make space
        df0=sortIx[sortIx>=numItems] # find clusters
        i=df0.index;j=df0.values-numItems
        sortIx[i]=link[j,0] # item 1
        df0=pd.Series(link[j,1],index=i+1)
        sortIx=sortIx.append(df0) # item 2
        sortIx=sortIx.sort_index() # re-sort
        sortIx.index=range(sortIx.shape[0]) # re-index
    return sortIx.tolist()
```

16.4.3 Recursive Bisection

Stage 2 has delivered a quasi-diagonal matrix. The inverse-variance allocation is optimal for a diagonal covariance matrix (see Appendix 16.A.2 for a proof). We can take advantage of these facts in two different ways: (1) bottom-up, to define the variance of a contiguous subset as the variance of an inverse-variance allocation; or (2) top-down, to split allocations between adjacent subsets in inverse proportion to their aggregated variances. The following algorithm formalizes this idea:

1. The algorithm is initialized by:
   (a) setting the list of items: $L = \{L_0\}$, with $L_0 = \{n\}_{n=1,\ldots,N}$
   (b) assigning a unit weight to all items: $w_n = 1, \ \forall n = 1, \ldots, N$

2. If $|L_i| = 1,\ \forall L_i \in L$, then stop.
3. For each $L_i \in L$ such that $|L_i| > 1$:
   (a) bisect L_i into two subsets, $L_i^{(1)} \cup L_i^{(2)} = L_i$, where $|L_i^{(1)}| = \text{int}[\frac{1}{2}|L_i|]$, and the order is preserved
   (b) define the variance of $L_i^{(j)},\ j = 1, 2$, as the quadratic form $\tilde{V}_i^{(j)} \equiv \tilde{w}_i^{(j)\prime} V_i^{(j)} \tilde{w}_i^{(j)}$, where $V_i^{(j)}$ is the covariance matrix between the constituents of the $L_i^{(j)}$ bisection, and $\tilde{w}_i^{(j)} = \text{diag}[V_i^{(j)}]^{-1} \frac{1}{\text{tr}[\text{diag}[V_i^{(j)}]^{-1}]}$, where diag[.] and tr[.] are the diagonal and trace operators
   (c) compute the split factor: $\alpha_i = 1 - \frac{\tilde{V}_i^{(1)}}{\tilde{V}_i^{(1)} + V_i^{(2)}}$, so that $0 \le \alpha_i \le 1$
   (d) re-scale allocations w_n by a factor of $\alpha_i, \forall n \in L_i^{(1)}$
   (e) re-scale allocations w_n by a factor of $(1 - \alpha_i), \forall n \in L_i^{(2)}$
4. Loop to step 2

Step 3b takes advantage of the quasi-diagonalization bottom-up, because it defines the variance of the partition $L_i^{(j)}$ using inverse-variance weightings $\tilde{w}_i^{(j)}$. Step 3c takes advantage of the quasi-diagonalization top-down, because it splits the weight in inverse proportion to the cluster's variance. This algorithm guarantees that $0 \le w_i \le 1, \forall i = 1, \ldots, N$, and $\sum_{i=1}^{N} w_i = 1$, because at each iteration we are splitting the weights received from higher hierarchical levels. Constraints can be easily introduced in this stage, by replacing the equations in steps 3c, 3d, and 3e according to the user's preferences. Stage 3 is implemented in Snippet 16.3.

SNIPPET 16.3 RECURSIVE BISECTION

```
def getRecBipart(cov,sortIx):
    # Compute HRP alloc
    w=pd.Series(1,index=sortIx)
    cItems=[sortIx] # initialize all items in one cluster
    while len(cItems)>0:
        cItems=[i[j:k] for i in cItems for j,k in ((0,len(i)/2),\
            (len(i)/2,len(i))) if len(i)>1] # bi-section
        for i in xrange(0,len(cItems),2): # parse in pairs
            cItems0=cItems[i] # cluster 1
            cItems1=cItems[i+1] # cluster 2
            cVar0=getClusterVar(cov,cItems0)
            cVar1=getClusterVar(cov,cItems1)
            alpha=1-cVar0/(cVar0+cVar1)
            w[cItems0]*=alpha # weight 1
            w[cItems1]*=1-alpha # weight 2
    return w
```

This concludes a first description of the HRP algorithm, which solves the allocation problem in best-case deterministic logarithmic time, $T(n) = \mathcal{O}\left(\log_2[n]\right)$, and worst-case deterministic linear time, $T(n) = \mathcal{O}(n)$. Next, we will put to practice what we have learned, and evaluate the method's accuracy out-of-sample.

16.5 A NUMERICAL EXAMPLE

We begin by simulating a matrix of observations X, of order $(10000x10)$. The correlation matrix is visualized in Figure 16.4 as a heatmap. Figure 16.5 displays the dendogram of the resulting clusters (stage 1). Figure 16.6 shows the same correlation matrix, reorganized in blocks according to the identified clusters (stage 2). Appendix 16.A.3 provides the code used to generate this numerical example.

On this random data, we compute HRP's allocations (stage 3), and compare them to the allocations from two competing methodologies: (1) Quadratic optimization, as represented by CLA's minimum-variance portfolio (the only portfolio of the efficient frontier that does not depend on returns' means); and (2) traditional risk parity, exemplified by the Inverse-Variance Portfolio (IVP). See Bailey and López de Prado [2013] for a comprehensive implementation of CLA, and Appendix 16.A.2 for a

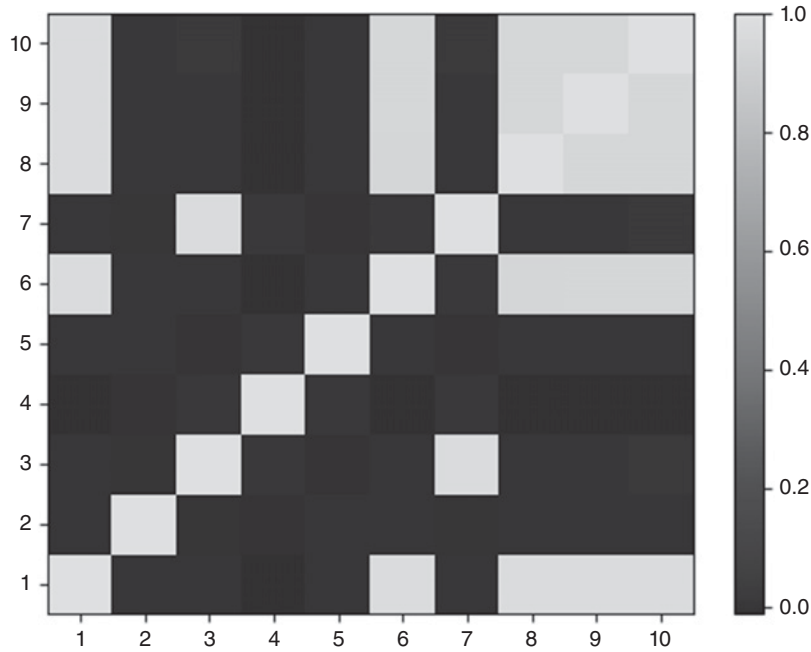


FIGURE 16.4 Heat-map of original covariance matrix
This correlation matrix has been computed using function `generateData` from snippet 16.4 (see Section 16.A.3). The last five columns are partially correlated to some of the first five series.

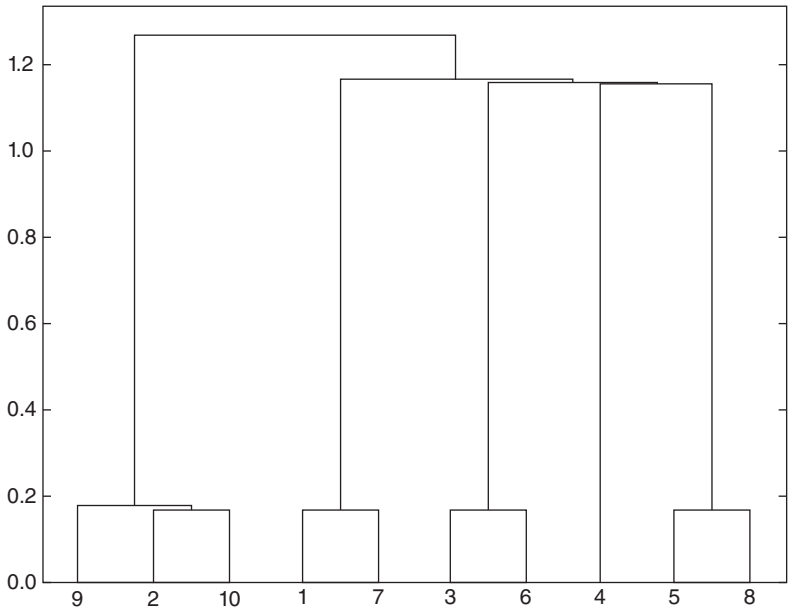


FIGURE 16.5 Dendogram of cluster formation
The clustering procedure has correctly identified that series 9 and 10 were perturbations of series 2, hence $(9, 2, 10)$ are clustered together. Similarly, 7 is a perturbation of 1, 6 is a perturbation of 3, and 8 is a perturbation of 5. The only original item that was not perturbated is 4, and that is the one item for which the clustering algorithm found no similarity.

derivation of IVP. We apply the standard constraints that $0 \leq w_i \leq 1$ (non-negativity), $\forall i = 1, \ldots, N$, and $\sum_{i=1}^{N} w_i = 1$ (full investment). Incidentally, the condition number for the covariance matrix in this example is only 150.9324, not particularly high and therefore not unfavorable to CLA.

From the allocations in Table 16.1, we can appreciate a few stylized features: First, CLA concentrates 92.66% of the allocation on the top-5 holdings, while HRP concentrates only 62.57%. Second, CLA assigns zero weight to 3 investments (without the $0 \leq w_i$ constraint, the allocation would have been negative). Third, HRP seems to find a compromise between CLA's concentrated solution and traditional risk parity's IVP allocation. The reader can use the code in Appendix 16.A.3 to verify that these findings generally hold for alternative random covariance matrices.

What drives CLA's extreme concentration is its goal of minimizing the portfolio's risk. And yet both portfolios have a very similar standard deviation ($\sigma_{HRP} = 0.4640$, $\sigma_{CLA} = 0.4486$). So CLA has discarded half of the investment universe in favor of a minor risk reduction. The reality of course is that CLA's portfolio is deceitfully diversified, because any distress situation affecting the top-5 allocations will have a much greater negative impact on CLA's than on HRP's portfolio.

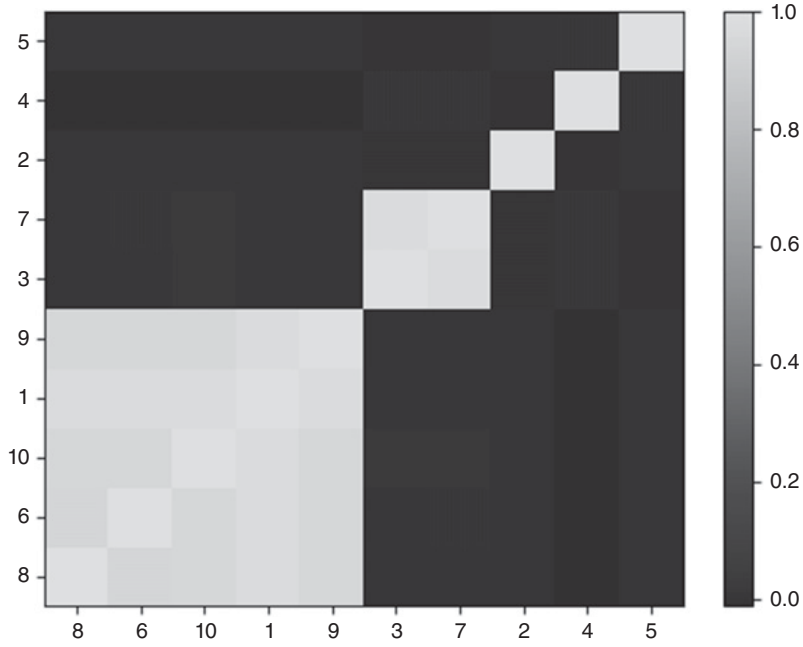


FIGURE 16.6 Clustered covariance matrix
Stage 2 quasi-diagonalizes the correlation matrix, in the sense that the largest values lie along the diagonal. However, unlike PCA or similar procedures, HRP does not require a change of basis. HRP solves the allocation problem robustly, while working with the original investments.

TABLE 16.1 A Comparison of Three Allocations

Weight #	CLA	HRP	IVP
1	14.44%	7.00%	10.36%
2	19.93%	7.59%	10.28%
3	19.73%	10.84%	10.36%
4	19.87%	19.03%	10.25%
5	18.68%	9.72%	10.31%
6	0.00%	10.19%	9.74%
7	5.86%	6.62%	9.80%
8	1.49%	9.10%	9.65%
9	0.00%	7.12%	9.64%
10	0.00%	12.79%	9.61%

A characteristic outcome of the three methods studied: CLA concentrates weights on a few investments, hence becoming exposed to idiosyncratic shocks. IVP evenly spreads weights through all investments, ignoring the correlation structure. This makes it vulnerable to systemic shocks. HRP finds a compromise between diversifying across all investments and diversifying across cluster, which makes it more resilient against both types of shocks.

16.6 OUT-OF-SAMPLE MONTE CARLO SIMULATIONS

In our numerical example, CLA's portfolio has lower risk than HRP's in-sample. However, the portfolio with minimum variance in-sample is not necessarily the one with minimum variance out-of-sample. It would be all too easy for us to pick a particular historical dataset where HRP outperforms CLA and IVP (see Bailey and López de Prado [2014], and recall our discussion of selection bias in Chapter 11). Instead, in this section we follow the backtesting paradigm explained in Chapter 13, and evaluate via Monte Carlo the performance out-of-sample of HRP against CLA's minimum-variance and traditional risk parity's IVP allocations. This will also help us understand what features make a method preferable to the rest, regardless of anecdotal counter-examples.

First, we generate 10 series of random Gaussian returns (520 observations, equivalent to 2 years of daily history), with 0 mean and an arbitrary standard deviation of 10%. Real prices exhibit frequent jumps (Merton [1976]) and returns are not cross-sectionally independent, so we must add random shocks and a random correlation structure to our generated data. Second, we compute HRP, CLA, and IVP portfolios by looking back at 260 observations (a year of daily history). These portfolios are re-estimated and rebalanced every 22 observations (equivalent to a monthly frequency). Third, we compute the out-of-sample returns associated with those three portfolios. This procedure is repeated 10,000 times.

All mean portfolio returns out-of-sample are essentially 0, as expected. The critical difference comes from the variance of the out-of-sample portfolio returns: $\sigma^2_{CLA} = 0.1157$, $\sigma^2_{IVP} = 0.0928$, and $\sigma^2_{HRP} = 0.0671$. Although CLA's goal is to deliver the lowest variance (that is the objective of its optimization program), its performance happens to exhibit the highest variance out-of-sample, and 72.47% greater variance than HRP's. This experimental finding is consistent with the historical evidence in De Miguel et al. [2009]. In other words, HRP would improve the out-of-sample Sharpe ratio of a CLA strategy by about 31.3%, a rather significant boost. Assuming that the covariance matrix is diagonal brings some stability to the IVP; however, its variance is still 38.24% greater than HRP's. This variance reduction out-of-sample is critically important to risk parity investors, given their use of substantial leverage. See Bailey et al. [2014] for a broader discussion of in-sample vs. out-of-sample performance.

The mathematical proof for HRP's outperformance over Markowitz's CLA and traditional risk parity's IVP is somewhat involved and beyond the scope of this chapter. In intuitive terms, we can understand the above empirical results as follows: Shocks affecting a specific investment penalize CLA's concentration. Shocks involving several correlated investments penalize IVP's ignorance of the correlation structure. HRP provides better protection against both common and idiosyncratic shocks by finding a compromise between diversification across all investments and diversification across clusters of investments at multiple hierarchical levels. Figure 16.7 plots the time series of allocations for the first of the 10,000 runs.

Appendix 16.A.4 provides the Python code that implements the above study. The reader can experiment with different parameter configurations and reach similar conclusions. In particular, HRP's out-of-sample outperformance becomes even

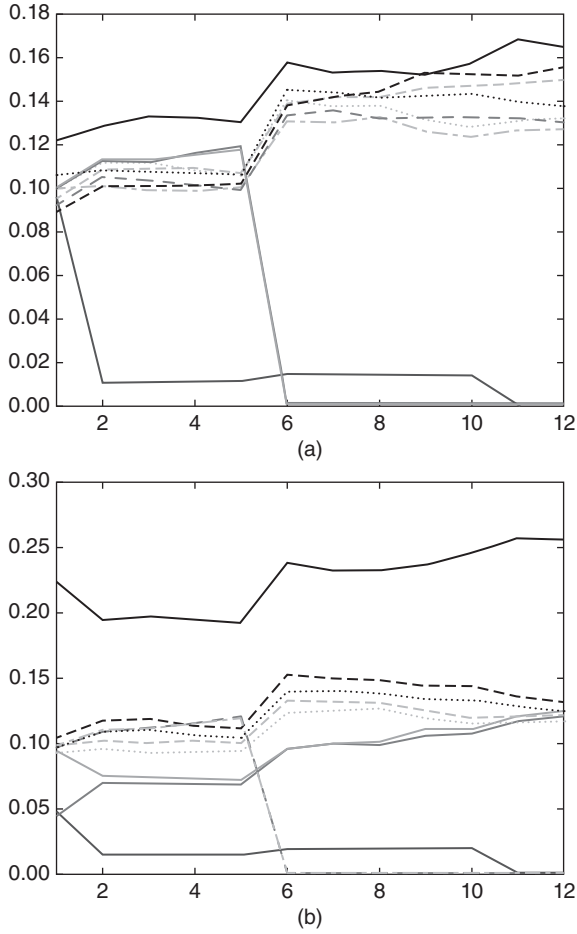


FIGURE 16.7 (a) Time series of allocations for IVP.
Between the first and second rebalance, one investment receives an idiosyncratic shock, which increases its variance. IVP's response is to reduce the allocation to that investment, and spread that former exposure across all other investments. Between the fifth and sixth rebalance, two investments are affected by a common shock. IVP's response is the same. As a result, allocations among the seven unaffected investments grow over time, regardless of their correlation.
(b) Time series of allocations for HRP
HRP's response to the idiosyncratic shock is to reduce the allocation to the affected investment, and use that reduced amount to increase the allocation to a correlated investment that was unaffected. As a response to the common shock, HRP reduces allocation to the affected investments and increases allocation to uncorrelated ones (with lower variance).
(c) Time series of allocations for CLA
CLA allocations respond erratically to idiosyncratic and common shocks. If we had taken into account rebalancing costs, CLA's performance would have been very negative.

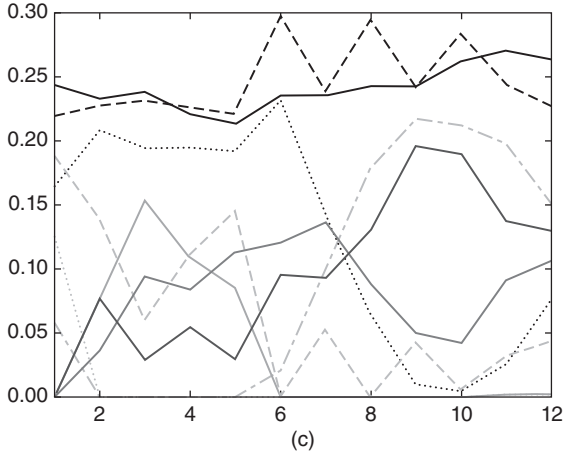


FIGURE 16.7 (*Continued*)

more substantial for larger investment universes, or when more shocks are added, or a stronger correlation structure is considered, or rebalancing costs are taken into account. Each of these CLA rebalances incurs transaction costs that can accumulate into prohibitive losses over time.

16.7 FURTHER RESEARCH

The methodology introduced in this chapter is flexible, scalable and admits multiple variations of the same ideas. Using the code provided, readers can research and evaluate what HRP configurations work best for their particular problem. For example, at stage 1 they can apply alternative definitions of $d_{i,j}$, $\tilde{d}_{i,j}$ and $\dot{d}_{i,u}$, or different clustering algorithms, like biclustering; at stage 3, they can use different functions for $\tilde{w}_m$ and α, or alternative allocation constraints. Instead of carrying out a recursive bisection, stage 3 could also split allocations top-down using the clusters from stage 1.

It is relatively straightforward to incorporate forecasted returns, Ledoit-Wolf shrinkage, and Black-Litterman–style views to this hierarchical approach. In fact, the inquisitive reader may have realized that, at its core, HRP is essentially a robust procedure to avoid matrix inversions, and the same ideas underlying HRP can be used to replace many econometric regression methods, notorious for their unstable outputs (like VAR or VECM). Figure 16.8 displays (a) a large correlation matrix of fixed income securities before and (b) after clustering, with over 2.1 million entries. Traditional optimization or econometric methods fail to recognize the hierarchical structure of financial Big Data, where the numerical instabilities defeat the benefits of the analysis, resulting in unreliable and detrimental outcomes.

Kolanovic et al. [2017] conducted a lengthy study of HRP, concluding that "HRP delivers superior risk-adjusted returns. Whilst both the HRP and the MV portfolios deliver the highest returns, the HRP portfolios match with volatility targets much better than MV portfolios. We also run simulation studies to confirm the robustness

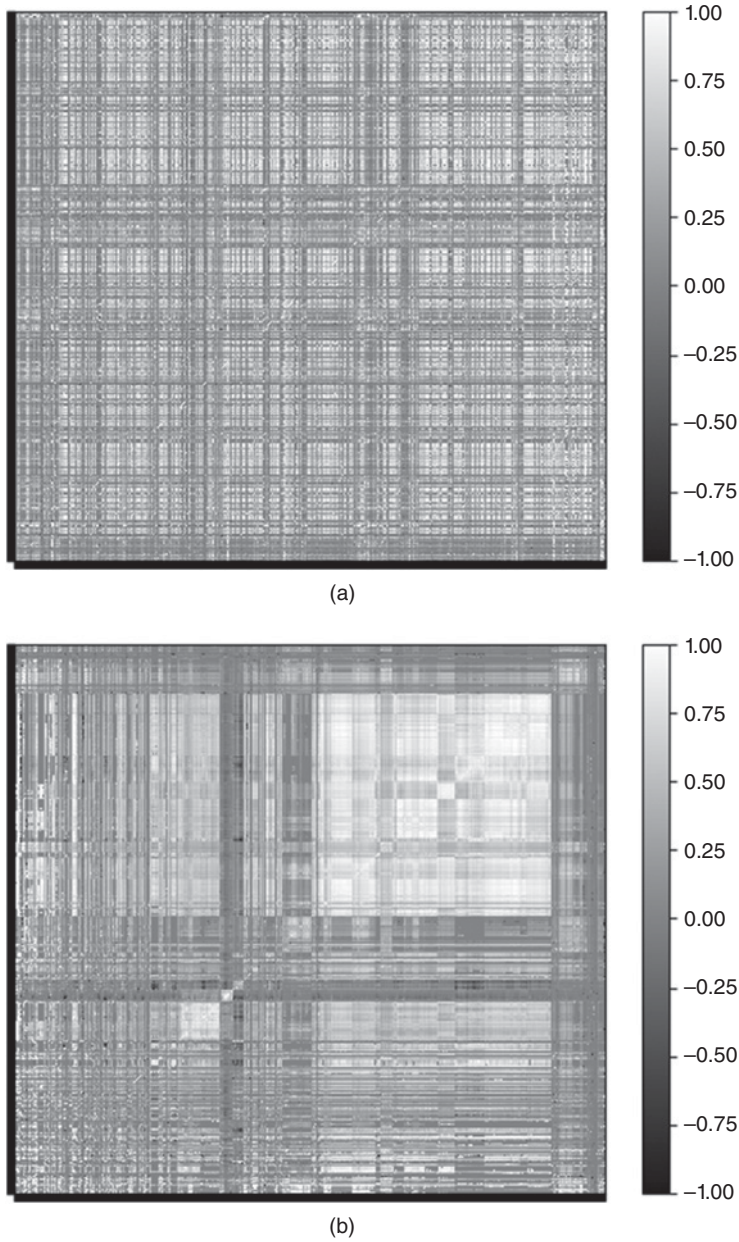


FIGURE 16.8 Correlation matrix before and after clustering
The methodology described in this chapter can be applied to problems beyond optimization. For example, a PCA analysis of a large fixed income universe suffers the same drawbacks we described for CLA. Small-data techniques developed decades and centuries ago (factor models, regression analysis, econometrics) fail to recognize the hierarchical nature of financial big data.

of our findings, in which HRP consistently deliver a superior performance over MV and other risk-based strategies [...] HRP portfolios are truly diversified with a higher number of uncorrelated exposures, and less extreme weights and risk allocations."

Raffinot [2017] concludes that "empirical results indicate that hierarchical clustering based portfolios are robust, truly diversified and achieve statistically better risk-adjusted performances than commonly used portfolio optimization techniques."

16.8 CONCLUSION

Exact analytical solutions can perform much worse than approximate ML solutions. Although mathematically correct, quadratic optimizers in general, and Markowitz's CLA in particular, are known to deliver generally unreliable solutions due to their instability, concentration, and underperformance. The root cause for these issues is that quadratic optimizers require the inversion of a covariance matrix. Markowitz's curse is that the more correlated investments are, the greater is the need for a diversified portfolio, and yet the greater are that portfolio's estimation errors.

In this chapter, we have exposed a major source of quadratic optimizers' instability: A matrix of size N is associated with a complete graph with $\frac{1}{2}N(N-1)$ edges. With so many edges connecting the nodes of the graph, weights are allowed to rebalance with complete freedom. This lack of hierarchical structure means that small estimation errors will lead to entirely different solutions. HRP replaces the covariance structure with a tree structure, accomplishing three goals: (1) Unlike traditional risk parity methods, it fully utilizes the information contained in the covariance matrix, (2) weights' stability is recovered and (3) the solution is intuitive by construction. The algorithm converges in deterministic logarithmic (best case) or linear (worst case) time.

HRP is robust, visual, and flexible, allowing the user to introduce constraints or manipulate the tree structure without compromising the algorithm's search. These properties are derived from the fact that HRP does not require covariance invertibility. Indeed, HRP can compute a portfolio on an ill-degenerated or even a singular covariance matrix.

This chapter focuses on a portfolio construction application; however, the reader will find other practical uses for making decisions under uncertainty, particularly in the presence of a nearly singular covariance matrix: capital allocation to portfolio managers, allocations across algorithmic strategies, bagging and boosting of machine learning signals, forecasts from random forests, replacement to unstable econometric models (VAR, VECM), etc.

Of course, quadratic optimizers like CLA produce the minimum-variance portfolio in-sample (that is its objective function). Monte Carlo experiments show that HRP delivers lower out-of-sample variance than CLA or traditional risk parity methods (IVP). Since Bridgewater pioneered risk parity in the 1990s, some of the largest asset managers have launched funds that follow this approach, for combined assets in excess of $500 billion. Given their extensive use of leverage, these funds should benefit from adopting a more stable risk parity allocation method, thus achieving superior risk-adjusted returns and lower rebalance costs.

APPENDICES

16.A.1 CORRELATION-BASED METRIC

Consider two real-valued vectors X, Y of size T, and a correlation variable $\rho[x, y]$, with the only requirement that $\sigma[x, y] = \rho[x, y]\sigma[X]\sigma[Y]$, where $\sigma[x, y]$ is the covariance between the two vectors, and $\sigma[.]$ is the standard deviation. Note that Pearson's is not the only correlation to satisfy these requirements.

Let us prove that $d[x, y] = \sqrt{\frac{1}{2}(1 - \rho[x, y])}$ is a true metric. First, the Euclidean distance between the two vectors is $d[x, y] = \sqrt{\sum_{t=1}^{T} (X_t - Y_t)^2}$. Second, we z-standardize those vectors as $x = \frac{X - \bar{X}}{\sigma[X]}$, $y = \frac{Y - \bar{Y}}{\sigma[Y]}$. Consequently, $0 \leq \rho[x, y] = \rho[x, y]$. Third, we derive the Euclidean distance $d[x, y]$ as,

$$d[x, y] = \sqrt{\sum_{t=1}^{T} (x_t - y_t)^2} = \sqrt{\sum_{t=1}^{T} x_t^2 + \sum_{t=1}^{T} y_t^2 - 2\sum_{t=1}^{T} x_t y_t}$$

$$= \sqrt{T + T - 2T\sigma[x, y]} = \sqrt{2T\left(1 - \underbrace{\rho[x, y]}_{=\rho[x,y]}\right)} = \sqrt{4T}d[x, y]$$

In other words, the distance $d[x, y]$ is a linear multiple of the Euclidean distance between the vectors $\{X, Y\}$ after z-standardization, hence it inherits the true-metric properties of the Euclidean distance.

Similarly, we can prove that $d[x, y] = \sqrt{1 - |\rho[x, y]|}$ descends to a true metric on the $\mathbb{Z}/2\mathbb{Z}$ quotient. In order to do that, we redefine $y = \frac{Y - \bar{Y}}{\sigma[Y]} \text{sgn}[\rho[x, y]]$, where sgn $[.]$ is the sign operator, so that $0 \leq \rho[x, y] = |\rho[x, y]|$. Then,

$$d[x, y] = \sqrt{2T\left(1 - \underbrace{\rho[x, y]}_{=|\rho[x,y]|}\right)} = \sqrt{2T}d[x, y]$$

16.A.2 INVERSE VARIANCE ALLOCATION

Stage 3 (see Section 16.4.3) splits a weight in inverse proportion to the subset's variance. We now prove that such allocation is optimal when the covariance matrix is diagonal. Consider the standard quadratic optimization problem of size N,

$$\min_{\omega} \omega' V \omega$$
$$\text{s.t. : } \omega' a = 1_I$$

with solution $\omega = \frac{V^{-1}a}{a'V^{-1}a}$. For the characteristic vector $a = 1_N$, the solution is the minimum variance portfolio. If V is diagonal, $\omega_n = \frac{V_{n,n}^{-1}}{\sum_{i=1}^{N} V_{i,i}^{-1}}$. In the particular case of $N = 2$, $\omega_1 = \frac{\frac{1}{V_{1,1}}}{\frac{1}{V_{1,1}}+\frac{1}{V_{2,2}}} = 1 - \frac{V_{1,1}}{V_{1,1}+V_{2,2}}$, which is how stage 3 splits a weight between two bisections of a subset.

16.A.3 REPRODUCING THE NUMERICAL EXAMPLE

Snippet 16.4 can be used to reproduce our results and simulate additional numerical examples. Function `generateData` produces a matrix of time series where a number `size0` of vectors are uncorrelated, and a number `size1` of vectors are correlated. The reader can change the `np.random.seed` in `generateData` to run alternative examples and gain an intuition of how HRP works. Scipy's function `linkage` can be used to perform stage 1 (Section 16.4.1), function `getQuasiDiag` performs stage 2 (Section 16.4.2), and function `getRecBipart` carries out stage 3 (Section 16.4.3).

SNIPPET 16.4 FULL IMPLEMENTATION OF THE HRP ALGORITHM

```
import matplotlib.pyplot as mpl
import scipy.cluster.hierarchy as sch,random,numpy as np,pandas as pd
#------------------------------------------------
def getIVP(cov,**kargs):
    # Compute the inverse-variance portfolio
    ivp=1./np.diag(cov)
    ivp/=ivp.sum()
    return ivp
#------------------------------------------------
def getClusterVar(cov,cItems):
    # Compute variance per cluster
    cov_=cov.loc[cItems,cItems] # matrix slice
    w_=getIVP(cov_).reshape(-1,1)
    cVar=np.dot(np.dot(w_.T,cov_),w_)[0,0]
    return cVar
#------------------------------------------------
def getQuasiDiag(link):
    # Sort clustered items by distance
    link=link.astype(int)
    sortIx=pd.Series([link[-1,0],link[-1,1]])
    numItems=link[-1,3] # number of original items
    while sortIx.max()>=numItems:
        sortIx.index=range(0,sortIx.shape[0]*2,2) # make space
        df0=sortIx[sortIx>=numItems] # find clusters
        i=df0.index;j=df0.values-numItems
        sortIx[i]=link[j,0] # item 1
```

```
        df0=pd.Series(link[j,1],index=i+1)
        sortIx=sortIx.append(df0) # item 2
        sortIx=sortIx.sort_index() # re-sort
        sortIx.index=range(sortIx.shape[0]) # re-index
    return sortIx.tolist()
#-------------------------------------------------
def getRecBipart(cov,sortIx):
    # Compute HRP alloc
    w=pd.Series(1,index=sortIx)
    cItems=[sortIx] # initialize all items in one cluster
    while len(cItems)>0:
        cItems=[i[j:k] for i in cItems for j,k in ((0,len(i)/2), \
            (len(i)/2,len(i))) if len(i)>1] # bi-section
        for i in xrange(0,len(cItems),2): # parse in pairs
            cItems0=cItems[i] # cluster 1
            cItems1=cItems[i+1] # cluster 2
            cVar0=getClusterVar(cov,cItems0)
            cVar1=getClusterVar(cov,cItems1)
            alpha=1-cVar0/(cVar0+cVar1)
            w[cItems0]*=alpha # weight 1
            w[cItems1]*=1-alpha # weight 2
    return w
#-------------------------------------------------
def correlDist(corr):
    # A distance matrix based on correlation, where 0<=d[i,j]<=1
    # This is a proper distance metric
    dist=((1-corr)/2.)**.5 # distance matrix
    return dist
#-------------------------------------------------
def plotCorrMatrix(path,corr,labels=None):
    # Heatmap of the correlation matrix
    if labels is None:labels=[]
    mpl.pcolor(corr)
    mpl.colorbar()
    mpl.yticks(np.arange(.5,corr.shape[0]+.5),labels)
    mpl.xticks(np.arange(.5,corr.shape[0]+.5),labels)
    mpl.savefig(path)
    mpl.clf();mpl.close() # reset pylab
    return
#-------------------------------------------------
def generateData(nObs,size0,size1,sigma1):
    # Time series of correlated variables
    #1) generating some uncorrelated data
    np.random.seed(seed=12345);random.seed(12345)
    x=np.random.normal(0,1,size=(nObs,size0)) # each row is a variable
    #2) creating correlation between the variables
    cols=[random.randint(0,size0-1) for i in xrange(size1)]
    y=x[:,cols]+np.random.normal(0,sigma1,size=(nObs,len(cols)))
    x=np.append(x,y,axis=1)
```

```
    x=pd.DataFrame(x,columns=range(1,x.shape[1]+1))
    return x,cols
#-----------------------------------------
def main():
    #1) Generate correlated data
    nObs,size0,size1,sigma1=10000,5,5,.25
    x,cols=generateData(nObs,size0,size1,sigma1)
    print [(j+1,size0+i) for i,j in enumerate(cols,1)]
    cov,corr=x.cov(),x.corr()
    #2) compute and plot correl matrix
    plotCorrMatrix('HRP3_corr0.png',corr,labels=corr.columns)
    #3) cluster
    dist=correlDist(corr)
    link=sch.linkage(dist,'single')
    sortIx=getQuasiDiag(link)
    sortIx=corr.index[sortIx].tolist() # recover labels
    df0=corr.loc[sortIx,sortIx] # reorder
    plotCorrMatrix('HRP3_corr1.png',df0,labels=df0.columns)
    #4) Capital allocation
    hrp=getRecBipart(cov,sortIx)
    print hrp
    return
#-----------------------------------------
if __name__=='__main__':main()
```

16.A.4 REPRODUCING THE MONTE CARLO EXPERIMENT

Snippet 16.5 implements Monte Carlo experiments on three allocation methods: HRP, CLA, and IVP. All libraries are standard except for HRP, which is provided in Appendix 16.A.3, and CLA, which can be found in Bailey and López de Prado [2013]. The subroutine `generateData` simulates the correlated data, with two types of random shocks: common to various investments and specific to a single investment. There are two shocks of each type, one positive and one negative. The variables for the experiments are set as arguments of `hrpMC`. They were chosen arbitrarily, and the user can experiment with alternative combinations.

SNIPPET 16.5 MONTE CARLO EXPERIMENT ON HRP OUT-OF-SAMPLE PERFORMANCE

```
import scipy.cluster.hierarchy as sch,random,numpy as np,pandas as pd,CLA
from HRP import correlDist,getIVP,getQuasiDiag,getRecBipart
#-----------------------------------------
def generateData(nObs,sLength,size0,size1,mu0,sigma0,sigma1F):
    # Time series of correlated variables
    #1) generate random uncorrelated data
```

```
    x=np.random.normal(mu0,sigma0,size=(nObs,size0))
    #2) create correlation between the variables
    cols=[random.randint(0,size0-1) for i in xrange(size1)]
    y=x[:,cols]+np.random.normal(0,sigma0*sigma1F,size=(nObs,len(cols)))
    x=np.append(x,y,axis=1)
    #3) add common random shock
    point=np.random.randint(sLength,nObs-1,size=2)
    x[np.ix_(point,[cols[0],size0])]=np.array([[-.5,-.5],[2,2]])
    #4) add specific random shock
    point=np.random.randint(sLength,nObs-1,size=2)
    x[point,cols[-1]]=np.array([-.5,2])
    return x,cols
    #-----------------------------------------
def getHRP(cov,corr):
    # Construct a hierarchical portfolio
    corr,cov=pd.DataFrame(corr),pd.DataFrame(cov)
    dist=correlDist(corr)
    link=sch.linkage(dist,'single')
    sortIx=getQuasiDiag(link)
    sortIx=corr.index[sortIx].tolist() # recover labels
    hrp=getRecBipart(cov,sortIx)
    return hrp.sort_index()
    #-----------------------------------------
def getCLA(cov,**kargs):
    # Compute CLA's minimum variance portfolio
    mean=np.arange(cov.shape[0]).reshape(-1,1) # Not used by C portf
    lB=np.zeros(mean.shape)
    uB=np.ones(mean.shape)
    cla=CLA.CLA(mean,cov,lB,uB)
    cla.solve()
    return cla.w[-1].flatten()
#-------------------------------------------
def hrpMC(numIters=1e4,nObs=520,size0=5,size1=5,mu0=0,sigma0=1e-2, \
    sigma1F=.25,sLength=260,rebal=22):
    # Monte Carlo experiment on HRP
    methods=[getIVP,getHRP,getCLA]
    stats,numIter={i.__name__:pd.Series() for i in methods},0
    pointers=range(sLength,nObs,rebal)
    while numIter<numIters:
        print numIter
        #1) Prepare data for one experiment
        x,cols=generateData(nObs,sLength,size0,size1,mu0,sigma0,sigma1F)
        r={i.__name__:pd.Series() for i in methods}
        #2) Compute portfolios in-sample
        for pointer in pointers:
            x_=x[pointer-sLength:pointer]
            cov_,corr_=np.cov(x_,rowvar=0),np.corrcoef(x_,rowvar=0)
            #3) Compute performance out-of-sample
            x_=x[pointer:pointer+rebal]
```

```
            for func in methods:
                w_=func(cov=cov_,corr=corr_) # callback
                r_=pd.Series(np.dot(x_,w_))
                r[func.__name__]=r[func.__name__].append(r_)
        #4) Evaluate and store results
        for func in methods:
            r_=r[func.__name__].reset_index(drop=True)
            p_=(1+r_).cumprod()
            stats[func.__name__].loc[numIter]=p_.iloc[-1]-1
        numIter+=1
    #5) Report results
    stats=pd.DataFrame.from_dict(stats,orient='columns')
    stats.to_csv('stats.csv')
    df0,df1=stats.std(),stats.var()
    print pd.concat([df0,df1,df1/df1['getHRP']-1],axis=1)
    return
#---------------------------------------------
if __name__=='__main__':hrpMC()
```

EXERCISES

16.1 Given the PnL series on N investment strategies:

- **(a)** Align them to the average frequency of their bets (e.g., weekly observations for strategies that trade on a weekly basis). Hint: This kind of data alignment is sometimes called "downsampling."
- **(b)** Compute the covariance of their returns, V.
- **(c)** Identify the hierarchical clusters among the N strategies.
- **(d)** Plot the clustered correlation matrix of the N strategies.

16.2 Using the clustered covariance matrix V from exercise 1:

- **(a)** Compute the HRP allocations.
- **(b)** Compute the CLA allocations.
- **(c)** Compute the IVP allocations.

16.3 Using the covariance matrix V from exercise 1:

- **(a)** Perform a spectral decomposition: $VW = W\Lambda$.
- **(b)** Form an array ε by drawing N random numbers from a $U[0, 1]$ distribution.
- **(c)** Form an NxN matrix $\tilde{\Lambda}$, where $\tilde{\Lambda}_{n,n} = N\varepsilon_n \Lambda_{n,n} (\sum_{n=1}^{N} \varepsilon_n)^{-1}$, $n = 1, \ldots, N$.
- **(d)** Compute $\tilde{V} = W\tilde{\Lambda}W^{-1}$.
- **(e)** Repeat exercise 2, this time using $\tilde{V}$ as covariance matrix. What allocation method has been most impacted by the re-scaling of spectral variances?

16.4 How would you modify the HRP algorithm to produce allocations that add up to 0, where $|w_n| \leq 1, \ \forall n = 1, \ldots, N$?

16.5 Can you think of an easy way to incorporate expected returns in the HRP allocations?

REFERENCES

Bailey, D. and M. López de Prado (2012): "Balanced baskets: A new approach to trading and hedging risks." *Journal of Investment Strategies*, Vol. 1, No. 4, pp. 21–62. Available at http://ssrn.com/abstract=2066170.

Bailey, D. and M. López de Prado (2013): "An open-source implementation of the critical-line algorithm for portfolio optimization." *Algorithms*, Vol. 6, No. 1, pp. 169–196. Available at http://ssrn.com/abstract=2197616.

Bailey, D., J. Borwein, M. López de Prado, and J. Zhu (2014) "Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance." *Notices of the American Mathematical Society*, Vol. 61, No. 5, pp. 458–471. Available at http://ssrn.com/abstract=2308659.

Bailey, D. and M. López de Prado (2014): "The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting and non-normality." *Journal of Portfolio Management*, Vol. 40, No. 5, pp. 94–107.

Black, F. and R. Litterman (1992): "Global portfolio optimization." *Financial Analysts Journal*, Vol. 48, pp. 28–43.

Brualdi, R. (2010): "The mutually beneficial relationship of graphs and matrices." Conference Board of the Mathematical Sciences, Regional Conference Series in Mathematics, Nr. 115.

Calkin, N. and M. López de Prado (2014): "Stochastic flow diagrams." *Algorithmic Finance*, Vol. 3, No. 1, pp. 21–42. Availble at http://ssrn.com/abstract=2379314.

Calkin, N. and M. López de Prado (2014): "The topology of macro financial flows: An application of stochastic flow diagrams." *Algorithmic Finance*, Vol. 3, No. 1, pp. 43–85. Available at http://ssrn.com/abstract=2379319.

Clarke, R., H. De Silva, and S. Thorley (2002): "Portfolio constraints and the fundamental law of active management." *Financial Analysts Journal*, Vol. 58, pp. 48–66.

De Miguel, V., L. Garlappi, and R. Uppal (2009): "Optimal versus naive diversification: How inefficient is the 1/N portfolio strategy?" *Review of Financial Studies*, Vol. 22, pp. 1915–1953.

Jurczenko, E. (2015): *Risk-Based and Factor Investing*, 1st ed. Elsevier Science.

Kolanovic, M., A. Lau, T. Lee, and R. Krishnamachari (2017): "Cross asset portfolios of tradable risk premia indices. Hierarchical risk parity: Enhancing returns at target volatility." White paper, Global Quantitative & Derivatives Strategy. J.P. Morgan, April 26.

Kolm, P., R. Tutuncu and F. Fabozzi (2014): "60 years of portfolio optimization." *European Journal of Operational Research*, Vol. 234, No. 2, pp. 356–371.

Kuhn, H. W. and A. W. Tucker (1951): "Nonlinear programming." Proceedings of 2nd Berkeley Symposium. Berkeley, University of California Press, pp. 481–492.

Markowitz, H. (1952): "Portfolio selection." *Journal of Finance*, Vol. 7, pp. 77–91.

Merton, R. (1976): "Option pricing when underlying stock returns are discontinuous." *Journal of Financial Economics*, Vol. 3, pp. 125–144.

Michaud, R. (1998): *Efficient Asset Allocation: A Practical Guide to Stock Portfolio Optimization and Asset Allocation*, 1st ed. Harvard Business School Press.

Ledoit, O. and M. Wolf (2003): "Improved estimation of the covariance matrix of stock returns with an application to portfolio selection." *Journal of Empirical Finance*, Vol. 10, No. 5, pp. 603–621.

Raffinot, T. (2017): "Hierarchical clustering based asset allocation." *Journal of Portfolio Management*, forthcoming.

Rokach, L. and O. Maimon (2005): "Clustering methods," in Rokach, L. and O. Maimon, eds., *Data Mining and Knowledge Discovery Handbook*. Springer, pp. 321–352.

PART 4

Useful Financial Features

CHAPTER 17

Structural Breaks

17.1 MOTIVATION

In developing an ML-based investment strategy, we typically wish to bet when there is a confluence of factors whose predicted outcome offers a favorable risk-adjusted return. Structural breaks, like the transition from one market regime to another, is one example of such a confluence that is of particular interest. For instance, a mean-reverting pattern may give way to a momentum pattern. As this transition takes place, most market participants are caught off guard, and they will make costly mistakes. This sort of errors is the basis for many profitable strategies, because the actors on the losing side will typically become aware of their mistake once it is too late. Before they accept their losses, they will act irrationally, try to hold the position, and hope for a comeback. Sometimes they will even increase a losing position, in desperation. Eventually they will be forced to stop loss or stop out. Structural breaks offer some of the best risk/rewards. In this chapter, we will review some methods that measure the likelihood of structural breaks, so that informative features can be built upon them.

17.2 TYPES OF STRUCTURAL BREAK TESTS

We can classify structural break tests in two general categories:

- **CUSUM tests:** These test whether the cumulative forecasting errors significantly deviate from white noise.
- **Explosiveness tests:** Beyond deviation from white noise, these test whether the process exhibits exponential growth or collapse, as this is inconsistent with a random walk or stationary process, and it is unsustainable in the long run.

- **Right-tail unit-root tests:** These tests evaluate the presence of exponential growth or collapse, while assuming an autoregressive specification.
- **Sub/super-martingale tests:** These tests evaluate the presence of exponential growth or collapse under a variety of functional forms.

17.3 CUSUM TESTS

In Chapter 2 we introduced the CUSUM filter, which we applied in the context of event-based sampling of bars. The idea was to sample a bar whenever some variable, like cumulative prediction errors, exceeded a predefined threshold. This concept can be further extended to test for structural breaks.

17.3.1 Brown-Durbin-Evans CUSUM Test on Recursive Residuals

This test was proposed by Brown, Durbin and Evans [1975]. Let us assume that at every observation $t = 1, \dots, T$, we count with an array of features x_t predictive of a value y_t. Matrix X_t is composed of the time series of features $t \leq T$, $\{x_i\}_{i=1,\dots,t}$. These authors propose that we compute recursive least squares (RLS) estimates of β, based on the specification

$$y_t = \beta_t' x_t + \varepsilon_t$$

which is fit on subsamples $([1, k+1], [1, k+2], \dots, [1, T])$, giving $T - k$ least squares estimates $(\hat{\beta}_{k+1}, \dots, \hat{\beta}_T)$. We can compute the standardized 1-step ahead recursive residuals as

$$\hat{\omega}_t = \frac{y_t - \hat{\beta}_{t-1}' x_t}{\sqrt{f_t}}$$

$$f_t = \hat{\sigma}_\varepsilon^2 \left[1 + x_t' \left(X_t' X_t\right)^{-1} x_t\right]$$

The CUSUM statistic is defined as

$$S_t = \sum_{j=k+1}^{t} \frac{\hat{\omega}_j}{\hat{\sigma}_\omega}$$

$$\hat{\sigma}_\omega^2 = \frac{1}{T-k} \sum_{t=k}^{T} (\hat{\omega}_t - \mathrm{E}[\hat{\omega}_t])^2$$

Under the null hypothesis that β is some constant value, $H_0 : \beta_t = \beta$, then $S_t \sim N[0, t - k - 1]$. One caveat of this procedure is that the starting point is chosen arbitrarily, and results may be inconsistent due to that.

17.3.2 Chu-Stinchcombe-White CUSUM Test on Levels

This test follows Homm and Breitung [2012]. It simplifies the previous method by dropping $\{x_t\}_{t=1,\dots,T}$, and assuming that $H_0 : \beta_t = 0$, that is, we forecast no change ($\mathrm{E}_{t-1}[\Delta y_t] = 0$). This will allow us to work directly with y_t levels, hence reducing the computational burden. We compute the standardized departure of log-price y_t relative to the log-price at y_n, $t > n$, as

$$S_{n,t} = (y_t - y_n)\left(\hat{\sigma}_t \sqrt{t-n}\right)^{-1}$$

$$\hat{\sigma}_t^2 = (t-1)^{-1} \sum_{i=2}^{t} (\Delta y_i)^2$$

Under the null hypothesis $H_0 : \beta_t = 0$, then $S_{n,t} \sim N[0,1]$. The time-dependent critical value for the *one-sided test* is

$$c_\alpha[n,t] = \sqrt{b_\alpha + \log[t-n]}$$

These authors derived via Monte Carlo that $b_{0.05} = 4.6$. One disadvantage of this method is that the reference level y_n is set somewhat arbitrarily. To overcome this pitfall, we could estimate $S_{n,t}$ on a series of backward-shifting windows $n \in [1,t]$, and pick $S_t = \sup_{n \in [1,t]} \{S_{n,t}\}$.

17.4 EXPLOSIVENESS TESTS

Explosiveness tests can be generally divided between those that test for one bubble and those that test for multiple bubbles. In this context, bubbles are not limited to price rallies, but they also include sell-offs. Tests that allow for multiple bubbles are more robust in the sense that a cycle of bubble-burst-bubble will make the series appear to be stationary to single-bubble tests. Maddala and Kim [1998], and Breitung [2014] offer good overviews of the literature.

17.4.1 Chow-Type Dickey-Fuller Test

A family of explosiveness tests was inspired by the work of Gregory Chow, starting with Chow [1960]. Consider the first order autoregressive process

$$y_t = \rho y_{t-1} + \varepsilon_t$$

where ε_t is white noise. The null hypothesis is that y_t follows a random walk, H_0: $\rho = 1$, and the alternative hypothesis is that y_t starts as a random walk but changes at time $\tau^* T$, where $\tau^* \in (0,1)$, into an explosive process:

$$H_1 : y_t = \begin{cases} y_{t-1} + \varepsilon_t \text{ for } t = 1, \dots, \tau^* T \\ \rho y_{t-1} + \varepsilon_t \text{ for } t = \tau^* T + 1, \dots, T, \text{ with } \rho > 1 \end{cases}$$

At time T we can test for a switch (from random walk to explosive process) having taken place at time $\tau^* T$ (break date). In order to test this hypothesis, we fit the following specification,

$$\Delta y_t = \delta y_{t-1} D_t[\tau^*] + \varepsilon_t$$

where $D_t[\tau^*]$ is a dummy variable that takes zero value if $t < \tau^* T$, and takes the value one if $t \geq \tau^* T$. Then, the null hypothesis $H_0 : \delta = 0$ is tested against the (one-sided) alternative $H_1 : \delta > 1$:

$$DFC_{\tau^*} = \frac{\hat{\delta}}{\hat{\sigma}_\delta}$$

The main drawback of this method is that τ^* is unknown. To address this issue, Andrews [1993] proposed a new test where all possible τ^* are tried, within some interval $\tau^* \in [\tau_0, 1 - \tau_0]$. As Breitung [2014] explains, we should leave out some of the possible τ^* at the beginning and end of the sample, to ensure that either regime is fitted with enough observations (there must be enough zeros and enough ones in $D_t[\tau^*]$). The test statistic for an unknown τ^* is the maximum of all $T(1 - 2\tau_0)$ values of DFC_{τ^*}.

$$SDFC = \sup_{\tau^* \in [\tau_0, 1-\tau_0]} \{DFC_{\tau^*}\}$$

Another drawback of Chow's approach is that it assumes that there is only one break date $\tau^* T$, and that the bubble runs up to the end of the sample (there is no switch back to a random walk). For situations where three or more regimes (random walk → bubble → random walk …) exist, we need to discuss the Supremum Augmented Dickey-Fuler (SADF) test.

17.4.2 Supremum Augmented Dickey-Fuller

In the words of Phillips, Wu and Yu [2011], "standard unit root and cointegration tests are inappropriate tools for detecting bubble behavior because they cannot effectively distinguish between a stationary process and a periodically collapsing bubble model. Patterns of periodically collapsing bubbles in the data look more like data generated from a unit root or stationary autoregression than a potentially explosive process." To address this flaw, these authors propose fitting the regression specification

$$\Delta y_t = \alpha + \beta y_{t-1} + \sum_{l=1}^{L} \gamma_l \Delta y_{t-l} + \varepsilon_t$$

where we test for $H_0 : \beta \leq 0$, $H_1 : \beta > 0$. Inspired by Andrews [1993], Phillips and Yu [2011] and Phillips, Wu and Yu [2011] proposed the Supremum Augmented

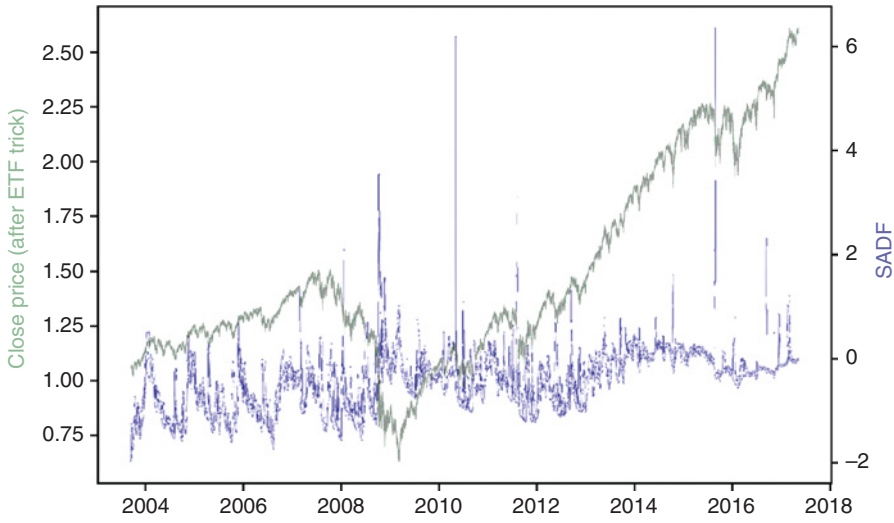


FIGURE 17.1 Prices (left y-axis) and SADF (right y-axis) over time

Dickey-Fuller test (SADF). SADF fits the above regression at each end point t with backwards expanding start points, then computes

$$SADF_t = \sup_{t_0 \in [1, t-\tau]} \{ADF_{t_0,t}\} = \sup_{t_0 \in [1, t-\tau]} \left\{ \frac{\hat{\beta}_{t_0,t}}{\hat{\sigma}_{\beta_{t_0,t}}} \right\}$$

where $\hat{\beta}_{t_0,t}$ is estimated on a sample that starts at t_0 and ends at t, τ is the minimum sample length used in the analysis, t_0 is the left bound of the backwards expanding window, and $t = \tau, \ldots, T$. For the estimation of $SADF_t$, the right side of the window is fixed at t. The standard ADF test is a special case of $SADF_t$, where $\tau = t - 1$.

There are two critical differences between $SADF_t$ and SDFC: First, $SADF_t$ is computed at each $t \in [\tau, T]$, whereas SDFC is computed only at T. Second, instead of introducing a dummy variable, SADF recursively expands the beginning of the sample ($t_0 \in [1, t - \tau]$). By trying all combinations of a nested double loop on (t_0, t), SADF does not assume a known number of regime switches or break dates. Figure 17.1 displays the series of E-mini S&P 500 futures prices after applying the ETF trick (Chapter 2, Section 2.4.1), as well as the SADF derived from that price series. The SADF line spikes when prices exhibit a bubble-like behavior, and returns to low levels when the bubble bursts. In the following sections, we will discuss some enhancements to Phillips' original SADF method.

17.4.2.1 Raw vs. Log Prices

It is common to find in the literature studies that carry out structural break tests on raw prices. In this section we will explore why log prices should be preferred, particularly when working with long time series involving bubbles and bursts.

For raw prices $\{y_t\}$, if ADF's null hypotesis is rejected, it means that prices are stationary, with finite variance. The implication is that returns $\frac{y_t}{y_{t-1}} - 1$ are not time invariant, for returns' volatility must decrease as prices rise and increase as prices fall in order to keep the price variance constant. When we run ADF on raw prices, we assume that returns' variance is not invariant to price levels. If returns variance happens to be invariant to price levels, the model will be structurally heteroscedastic.

In contrast, if we work with log prices, the ADF specification will state that

$$\Delta \log[y_t] \propto \log[y_{t-1}]$$

Let us make a change of variable, $x_t = ky_t$. Now, $\log[x_t] = \log[k] + \log[y_t]$, and the ADF specification will state that

$$\Delta \log[x_t] \propto \log[x_{t-1}] \propto \log[y_{t-1}]$$

Under this alternative specification based on log prices, price levels condition returns' mean, not returns' volatility. The difference may not matter in practice for small samples, where $k \approx 1$, but SADF runs regressions across decades and bubbles produce levels that are significantly different between regimes ($k \neq 1$).

17.4.2.2 Computational Complexity

The algorithm runs in $\mathcal{O}(n^2)$, as the number of ADF tests that SADF requires for a total sample length T is

$$\sum_{t=\tau}^{T} t - \tau + 1 = \frac{1}{2}(T - \tau + 2)(T - \tau + 1) = \binom{T - \tau + 2}{2}$$

Consider a matrix representation of the ADF specification, where $X \in \mathbb{R}^{TxN}$ and $y \in \mathbb{R}^{Tx1}$. Solving a single ADF regression involves the floating point operations (FLOPs) listed in Table 17.1.

This gives a total of $f(N,T) = N^3 + N^2(2T+3) + N(4T-1) + 2T + 2$ FLOPs per ADF estimate. A single SADF update requires $g(N,T,\tau) = \sum_{t=\tau}^{T} f(N,t) + T - \tau$ FLOPs ($T - \tau$ operations to find the maximum ADF stat), and the estimation of a full SADF series requires $\sum_{t=\tau}^{T} g(N,T,\tau)$.

Consider a dollar bar series on E-mini S&P 500 futures. For $(T,N) = (356631,3)$, an ADF estimate requires 11,412,245 FLOPs, and a SADF update requires 2,034,979,648,799 operations (roughly 2.035 TFLOPs). A full SADF time series requires 241,910,974,617,448,672 operations (roughly 242 PFLOPs). This number will increase quickly, as the T continues to grow. And this estimate excludes notoriously expensive operations like alignment, pre-processing of data, I/O jobs, etc. Needless to say, this algorithm's double loop requires a large number of operations. An HPC cluster running an efficiently parallelized implementation of the algorithm may be needed to estimate the SADF series within a reasonable amount of time. Chapter 20 will present some parallelization strategies useful in these situations.

TABLE 17.1 FLOPs per ADF Estimate

Matrix Operation	FLOPs
$o_1 = X'y$	$(2T-1)N$
$o_2 = X'X$	$(2T-1)N^2$
$o_3 = o_2^{-1}$	$N^3 + N^2 + N$
$o_4 = o_3 o_1$	$2N^2 - N$
$o_5 = y - Xo_4$	$T + (2N-1)T$
$o_6 = o_5' o_5$	$2T - 1$
$o_7 = o_3 o_6 \frac{1}{T-N}$	$2 + N^2$
$o_8 = \frac{o_4[0,0]}{\sqrt{o_7[0,0]}}$	1

17.4.2.3 Conditions for Exponential Behavior

Consider the zero-lag specification on log prices, $\Delta\log[y_t] = \alpha + \beta\log[y_{t-1}] + \varepsilon_t$. This can be rewritten as $\log[\tilde{y}_t] = (1+\beta)\log[\tilde{y}_{t-1}] + \varepsilon_t$, where $\log[\tilde{y}_t] = \log[y_t] + \frac{\alpha}{\beta}$. Rolling back t discrete steps, we obtain $\mathrm{E}[\log[\tilde{y}_t]] = (1+\beta)^t \log[\tilde{y}_0]$, or $\mathrm{E}[\log[y_t]] = -\frac{\alpha}{\beta} + (1+\beta)^t (l\log[y_0] + \frac{\alpha}{\beta})$. The index t can be reset at a given time, to project the future trajectory of $y_0 \to y_t$ after the next t steps. This reveals the conditions that characterize the three states for this dynamic system:

- Steady: $\beta < 0 \Rightarrow \lim_{t\to\infty} \mathrm{E}[\log[y_t]] = -\frac{\alpha}{\beta}$.
  - The disequilibrium is $\log[y_t] - (-\frac{\alpha}{\beta}) = \log[\tilde{y}_t]$.
  - Then $\frac{\mathrm{E}[\log[\tilde{y}_t]]}{\log[\tilde{y}_0]} = (1+\beta)^t = \frac{1}{2}$ at $t = -\frac{\log[2]}{\log[1+\beta]}$ (half-life).
- Unit-root: $\beta = 0$, where the system is non-stationary, and behaves as a martingale.
- Explosive: $\beta > 0$, where $\lim_{t\to\infty} \mathrm{E}[\log[y_t]] = \begin{cases} -\infty, & \text{if } \log[y_0] < \frac{\alpha}{\beta} \\ +\infty, & \text{if } \log[y_0] > \frac{\alpha}{\beta} \end{cases}$.

17.4.2.4 Quantile ADF

SADF takes the supremum of a series on t-values, $SADF_t = \sup_{t_0 \in [1, t-\tau]} \{ADF_{t_0,t}\}$. Selecting the extreme value introduces some robustness problems, where SADF estimates could vary significantly depending on the sampling frequency and the specific timestamps of the samples. A more robust estimator of ADF extrema would be the following: First, let $s_t = \{ADF_{t_0,t}\}_{t_0 \in [0, t_1 - \tau]}$. Second, we define $Q_{t,q} = Q[s_t, q]$ the q quantile of s_t, as a measure of centrality of high ADF values, where $q \in [0,1]$. Third, we define $\dot{Q}_{t,q,v} = Q_{t,q+v} - Q_{t,q-v}$, with $0 < v \leq \min\{q, 1-q\}$, as a measure of dispersion of high ADF values. For example, we could set $q = 0.95$ and $v = 0.025$. Note

that SADF is merely a particular case of QADF, where $SADF_t = Q_{t,1}$ and $\dot{Q}_{t,q,v}$ is not defined because $q = 1$.

17.4.2.5 Conditional ADF

Alternatively, we can address concerns on SADF robustness by computing conditional moments. Let $f[x]$ be the probability distribution function of $s_t = \{ADF_{t_0,t}\}_{t_0 \in [1, t_1 - \tau]}$, with $x \in s_t$. Then, we define $C_{t,q} = K^{-1} \int_{Q_{t,q}}^{\infty} xf[x]dx$ as a measure of centrality of high ADF values, and $\dot{C}_{t,q} = \sqrt{K^{-1} \int_{Q_{t,q}}^{\infty} (x - C_{t,q})^2 f[x]dx}$ as a measure of dispersion of high ADF values, with regularization constant $K = \int_{Q_{t,q}}^{\infty} f[x]dx$. For example, we could use $q = 0.95$.

By construction, $C_{t,q} \leq SADF_t$. A scatter plot of $SADF_t$ against $C_{t,q}$ shows that lower boundary, as an ascending line with approximately unit gradient (see Figure 17.2). When SADF grows beyond -1.5, we can appreciate some horizontal trajectories, consistent with a sudden widening of the right fat tail in s_t. In other words, $(SADF_t - C_{t,q})/\dot{C}_{t,q}$ can reach significantly large values even if $C_{t,q}$ is relatively small, because $SADF_t$ is sensitive to outliers.

Figure 17.3(a) plots $(SADF_t - C_{t,q})/\dot{C}_{t,q}$ for the E-mini S&P 500 futures prices over time. Figure 17.3(b) is the scatter-plot of $(SADF_t - C_{t,q})/\dot{C}_{t,q}$ against $SADF_t$, computed on the E-mini S&P 500 futures prices. It shows evidence that outliers in s_t bias $SADF_t$ upwards.

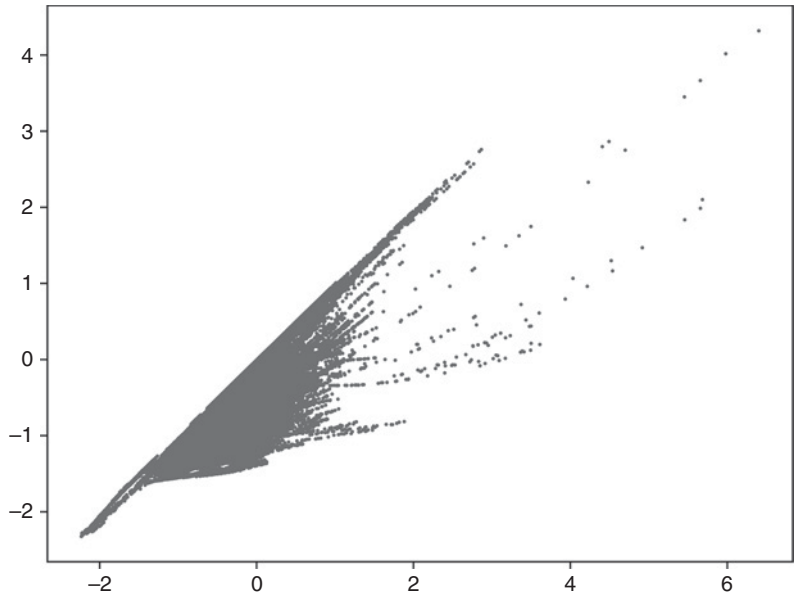


FIGURE 17.2 SADF (x-axis) vs CADF (y-axis)

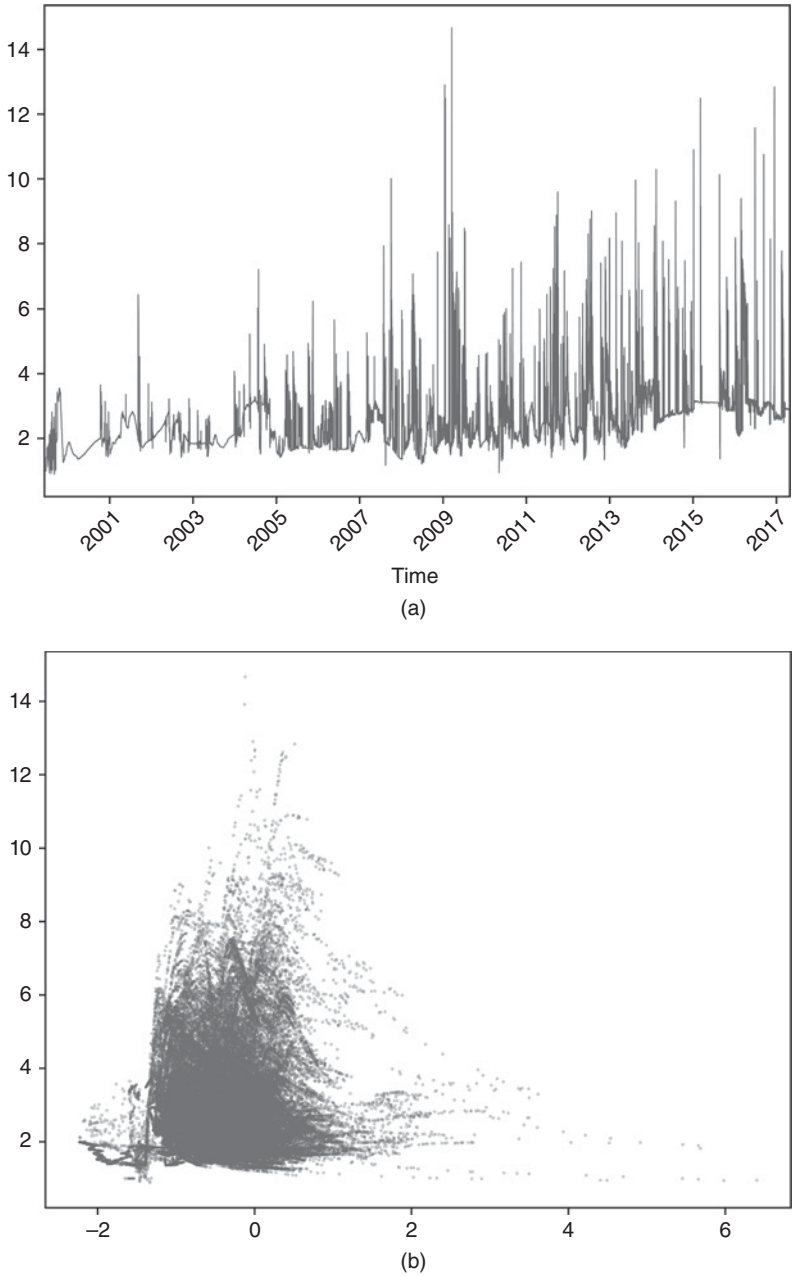


FIGURE 17.3 (a) $(SADF_t - C_{t,q})/\dot{C}_{t,q}$ over time (b) $(SADF_t - C_{t,q})/\dot{C}_{t,q}$ (y-axis) as a function of $SADF_t$ (x-axis)

17.4.2.6 Implementation of SADF

This section presents an implementation of the SADF algorithm. The purpose of this code is not to estimate SADF quickly, but to clarify the steps involved in its estimation. Snippet 17.1 lists SADF's inner loop. That is the part that estimates $SADF_t = \sup_{t_0 \in [1, t-\tau]} \{\frac{\hat{\beta}_{t_0,t}}{\hat{\sigma}_{\beta_{t_0,t}}}\}$, which is the backshifting component of the algorithm. The outer loop (not shown here) repeats this calculation for an advancing t, $\{SADF_t\}_{t=1,\ldots,T}$. The arguments are:

- `logP`: a pandas series containing log-prices
- `minSL`: the minimum sample length (τ), used by the final regression
- `constant`: the regression's time trend component
  - `'nc'`: no time trend, only a constant
  - `'ct'`: a constant plus a linear time trend
  - `'ctt'`: a constant plus a second-degree polynomial time trend
- `lags`: the number of lags used in the ADF specification

SNIPPET 17.1 SADF'S INNER LOOP

```
def get_bsadf(logP,minSL,constant,lags):
    y,x=getYX(logP,constant=constant,lags=lags)
    startPoints,bsadf,allADF=range(0,y.shape[0]+lags-minSL+1),None,[]
    for start in startPoints:
        y_,x_=y[start:],x[start:]
        bMean_,bStd_=getBetas(y_,x_)
        bMean_,bStd_=bMean_[0,0],bStd_[0,0]**.5
        allADF.append(bMean_/bStd_)
        if allADF[-1]>bsadf:bsadf=allADF[-1]
    out={'Time':logP.index[-1],'gsadf':bsadf}
    return out
```

Snippet 17.2 lists function `getXY`, which prepares the numpy objects needed to conduct the recursive tests.

SNIPPET 17.2 PREPARING THE DATASETS

```
def getYX(series,constant,lags):
    series_=series.diff().dropna()
    x=lagDF(series_,lags).dropna()
    x.iloc[:,0]=series.values[-x.shape[0]-1:-1,0] # lagged level
    y=series_.iloc[-x.shape[0]:].values
```

```
    if constant!='nc':
        x=np.append(x,np.ones((x.shape[0],1)),axis=1)
        if constant[:2]=='ct':
            trend=np.arange(x.shape[0]).reshape(-1,1)
            x=np.append(x,trend,axis=1)
        if constant=='ctt':
            x=np.append(x,trend**2,axis=1)
    return y,x
```

Snippet 17.3 lists function `lagDF`, which applies to a dataframe the lags specified in its argument `lags`.

SNIPPET 17.3 APPLY LAGS TO DATAFRAME

```
def lagDF(df0,lags):
    df1=pd.DataFrame()
    if isinstance(lags,int):lags=range(lags+1)
    else:lags=[int(lag) for lag in lags]
    for lag in lags:
        df_=df0.shift(lag).copy(deep=True)
        df_.columns=[str(i)+'_'+str(lag) for i in df_.columns]
        df1=df1.join(df_,how='outer')
    return df1
```

Finally, Snippet 17.4 lists function `getBetas`, which carries out the actual regressions.

SNIPPET 17.4 FITTING THE ADF SPECIFICATION

```
def getBetas(y,x):
    xy=np.dot(x.T,y)
    xx=np.dot(x.T,x)
    xxinv=np.linalg.inv(xx)
    bMean=np.dot(xxinv,xy)
    err=y-np.dot(x,bMean)
    bVar=np.dot(err.T,err)/(x.shape[0]-x.shape[1])*xxinv
    return bMean,bVar
```

17.4.3 Sub- and Super-Martingale Tests

In this section we will introduce explosiveness tests that do not rely on the standard ADF specification. Consider a process that is either a sub- or super-martingale. Given

some observations $\{y_t\}$, we would like to test for the existence of an explosive time trend, $H_0 : \beta = 0$, $H_1 : \beta \neq 0$, under alternative specifications:

- Polynomial trend (SM-Poly1):

$$y_t = \alpha + \gamma t + \beta t^2 + \varepsilon_t$$

- Polynomial trend (SM-Poly2):

$$\log[y_t] = \alpha + \gamma t + \beta t^2 + \varepsilon_t$$

- Exponential trend (SM-Exp):

$$y_t = \alpha e^{\beta t} + \varepsilon_t \Rightarrow \log[y_t] = \log[\alpha] + \beta t + \xi_t$$

- Power trend (SM-Power):

$$y_t = \alpha t^{\beta} + \varepsilon_t \Rightarrow \log[y_t] = \log[\alpha] + \beta \log[t] + \xi_t$$

Similar to SADF, we fit any of these specifications to each end point $t = \tau, \ldots, T$, with backwards expanding start points, then compute

$$SMT_t = \sup_{t_0 \in [1, t-\tau]} \left\{ \frac{|\hat{\beta}_{t_0,t}|}{\hat{\sigma}_{\beta_{t_0,t}}} \right\}$$

The reason for the absolute value is that we are equally interested in explosive growth and collapse. In the simple regression case (Greene [2008], p. 48), the variance of β is $\hat{\sigma}_{\beta}^2 = \frac{\hat{\sigma}_{\varepsilon}^2}{\hat{\sigma}_{xx}^2 (t - t_0)}$, hence $\lim_{t \to \infty} \hat{\sigma}_{\beta_{t_0,t}} = 0$. The same result is generalizable to the multivariate linear regression case (Greene [2008], pp. 51–52). The $\hat{\sigma}_{\beta}^2$ of a weak long-run bubble may be smaller than the $\hat{\sigma}_{\beta}^2$ of a strong short-run bubble, hence biasing the method towards long-run bubbles. To correct for this bias, we can penalize large sample lengths by determining the coefficient $\varphi \in [0, 1]$ that yields best explosiveness signals.

$$SMT_t = \sup_{t_0 \in [1, t-\tau]} \left\{ \frac{|\hat{\beta}_{t_0,t}|}{\hat{\sigma}_{\beta_{t_0,t}} (t - t_0)^{\varphi}} \right\}$$

For instance, when $\varphi = 0.5$, we compensate for the lower $\hat{\sigma}_{\beta_{t_0,t}}$ associated with longer sample lengths, in the simple regression case. For $\varphi \to 0$, SMT_t will exhibit longer trends, as that compensation wanes and long-run bubbles mask short-run bubbles. For $\varphi \to 1$, SMT_t becomes noisier, because more short-run bubbles are selected over long-run bubbles. Consequently, this is a natural way to adjust the explosiveness

signal, so that it filters opportunities targeting a particular holding period. The features used by the ML algorithm may include SMT_t estimated from a wide range of φ values.

EXERCISES

17.1 On a dollar bar series on E-mini S&P 500 futures,

(a) Apply the Brown-Durbin-Evans method. Does it recognize the dot-com bubble?

(b) Apply the Chu-Stinchcombe-White method. Does it find a bubble in 2007–2008?

17.2 On a dollar bar series on E-mini S&P 500 futures,

(a) Compute the *SDFC* (Chow-type) explosiveness test. What break date does this method select? Is this what you expected?

(b) Compute and plot the SADF values for this series. Do you observe extreme spikes around the dot-com bubble and before the Great Recession? Did the bursts also cause spikes?

17.3 Following on exercise 2,

(a) Determine the periods where the series exhibited

(i) Steady conditions

(ii) Unit-Root conditions

(iii) Explosive conditions

(b) Compute QADF.

(c) Compute CADF.

17.4 On a dollar bar series on E-mini S&P 500 futures,

(a) Compute SMT for SM-Poly1 and SM-Poly 2, where $\varphi = 1$. What is their correlation?

(b) Compute SMT for SM-Exp, where $\varphi = 1$ and $\varphi = 0.5$. What is their correlation?

(c) Compute SMT for SM-Power, where $\varphi = 1$ and $\varphi = 0.5$. What is their correlation?

17.5 If you compute the reciprocal of each price, the series $\{y_t^{-1}\}$ turns bubbles into bursts and bursts into bubbles.

(a) Is this transformation needed, to identify bursts?

(b) What methods in this chapter can identify bursts without requiring this transformation?

REFERENCES

Andrews, D. (1993): "Tests for parameter instability and structural change with unknown change point." *Econometrics*, Vol. 61, No. 4 (July), pp. 821–856.

Breitung, J. and R. Kruse (2013): "When Bubbles Burst: Econometric Tests Based on Structural Breaks." *Statistical Papers*, Vol. 54, pp. 911–930.

Breitung, J. (2014): "Econometric tests for speculative bubbles." *Bonn Journal of Economics*, Vol. 3, No. 1, pp. 113–127.

Brown, R.L., J. Durbin, and J.M. Evans (1975): "Techniques for Testing the Constancy of Regression Relationships over Time." *Journal of the Royal Statistical Society, Series B*, Vol. 35, pp. 149–192.

Chow, G. (1960). "Tests of equality between sets of coefficients in two linear regressions." *Econometrica*, Vol. 28, No. 3, pp. 591–605.

Greene, W. (2008): *Econometric Analysis*, 6th ed. Pearson Prentice Hall.

Homm, U. and J. Breitung (2012): "Testing for speculative bubbles in stock markets: A comparison of alternative methods." *Journal of Financial Econometrics*, Vol. 10, No. 1, 198–231.

Maddala, G. and I. Kim (1998): *Unit Roots, Cointegration and Structural Change*, 1st ed. Cambridge University Press.

Phillips, P., Y. Wu, and J. Yu (2011): "Explosive behavior in the 1990s Nasdaq: When did exuberance escalate asset values?" *International Economic Review*, Vol. 52, pp. 201–226.

Phillips, P. and J. Yu (2011): "Dating the timeline of financial bubbles during the subprime crisis." *Quantitative Economics*, Vol. 2, pp. 455–491.

Phillips, P., S. Shi, and J. Yu (2013): "Testing for multiple bubbles 1: Historical episodes of exuberance and collapse in the S&P 500." Working paper 8–2013, Singapore Management University.

CHAPTER 18

Entropy Features

18.1 MOTIVATION

Price series convey information about demand and supply forces. In perfect markets, prices are unpredictable, because each observation transmits everything that is known about a product or service. When markets are not perfect, prices are formed with partial information, and as some agents know more than others, they can exploit that informational asymmetry. It would be helpful to estimate the informational content of price series, and form features on which ML algorithms can learn the likely outcomes. For example, the ML algorithm may find that momentum bets are more profitable when prices carry little information, and that mean-reversion bets are more profitable when prices carry a lot of information. In this chapter, we will explore ways to determine the amount of information contained in a price series.

18.2 SHANNON'S ENTROPY

In this section we will review a few concepts from information theory that will be useful in the remainder of the chapter. The reader can find a complete exposition in MacKay [2003]. The father of information theory, Claude Shannon, defined entropy as the average amount of information (over long messages) produced by a stationary source of data. It is the smallest number of bits per character required to describe the message in a uniquely decodable way. Mathematically, Shannon [1948] defined the entropy of a discrete random variable X with possible values $x \in A$ as

$$H[X] \equiv -\sum_{x \in A} p[x] \log_2 p[x]$$

with $0 \leq H[X] \leq \log_2[\|A\|]$ where: $p[x]$ is the probability of x; $H[X] = 0 \Leftrightarrow \exists x|p[x] = 1$; $H[X] = \log_2[\|A\|] \Leftrightarrow p[x] = \frac{1}{\|A\|}$ for all x; and $\|A\|$ is the size of the set A. This can be interpreted as the probability weighted average of informational content in X, where the bits of information are measured as $\log_2 \frac{1}{p[x]}$. The rationale for measuring information as $\log_2 \frac{1}{p[x]}$ comes from the observation that low-probability outcomes reveal more information than high-probability outcomes. In other words, we learn when something unexpected happens. Similarly, redundancy is defined as

$$R[X] \equiv 1 - \frac{H[X]}{\log_2[\|A\|]}$$

with $0 \leq R[X] \leq 1$. Kolmogorov [1965] formalized the connection between redundancy and complexity of a Markov information source. The mutual information between two variables is defined as the Kullback-Leibler divergence from the joint probability density to the product of the marginal probability densities.

$$MI[X, Y] = \mathrm{E}_{f[x,y]}\left[\log \frac{f[x, y]}{f[x]f[y]}\right] = H[X] + H[Y] - H[X, Y]$$

The mutual information (MI) is always non-negative, symmetric, and equals zero if and only if X and Y are independent. For normally distributed variables, the mutual information is closely related to the familiar Pearson correlation, ρ.

$$MI[X, Y] = -\frac{1}{2}\log[1 - \rho^2]$$

Therefore, mutual information is a natural measure of the association between variables, regardless of whether they are linear or nonlinear in nature (Hausser and Strimmer [2009]). The normalized variation of information is a metric derived from mutual information. For several entropy estimators, see:

- In R: http://cran.r-project.org/web/packages/entropy/entropy.pdf
- In Python: https://code.google.com/archive/p/pyentropy/

18.3 THE PLUG-IN (OR MAXIMUM LIKELIHOOD) ESTIMATOR

In this section we will follow the exposition of entropy's maximum likelihood estimator in Gao et al. [2008]. The nomenclature may seem a bit peculiar at first (no pun intended), but once you become familiar with it you will find it convenient. Given a data sequence x_1^n, comprising the string of values starting in position 1 and ending in position n, we can form a dictionary of all words of length $w < n$ in that sequence, A^w. Consider an arbitrary word $y_1^w \in A^w$ of length w. We denote $\hat{p}_w[y_1^w]$ the empirical probability of the word y_1^w in x_1^n, which means that $\hat{p}_w[y_1^w]$ is the frequency with which

y_1^w appears in x_1^n. Assuming that the data is generated by a stationary and ergodic process, then the law of large numbers guarantees that, for a fixed w and large n, the empirical distribution $\hat{p}_w$ will be close to the true distribution p_w. Under these circumstances, a natural estimator for the entropy rate (i.e., average entropy per bit) is

$$\hat{H}_{n,w} = -\frac{1}{w} \sum_{y_1^w \in A^w} \hat{p}_w \left[y_1^w\right] \log_2 \hat{p}_w \left[y_1^w\right]$$

Since the empirical distribution is also the maximum likelihood estimate of the true distribution, this is also often referred to as the maximum likelihood entropy estimator. The value w should be large enough for $\hat{H}_{n,w}$ to be acceptably close to the true entropy H. The value of n needs to be much larger than w, so that the empirical distribution of order w is close to the true distribution. Snippet 18.1 implements the plug-in entropy estimator.

SNIPPET 18.1 PLUG-IN ENTROPY ESTIMATOR

```
import time,numpy as np
#-------------------------------------------
def plugIn(msg,w):
    # Compute plug-in (ML) entropy rate
    pmf=pmf1(msg,w)
    out=-sum([pmf[i]*np.log2(pmf[i]) for i in pmf])/w
    return out,pmf
#-------------------------------------------
def pmf1(msg,w):
    # Compute the prob mass function for a one-dim discrete rv
    # len(msg)-w occurrences
    lib={}
    if not isinstance(msg,str):msg=''.join(map(str,msg))
    for i in xrange(w,len(msg)):
        msg_=msg[i-w:i]
        if msg_ not in lib:lib[msg_]=[i-w]
        else:lib[msg_]=lib[msg_]+[i-w]
    pmf=float(len(msg)-w)
    pmf={i:len(lib[i])/pmf for i in lib}
    return pmf
```

18.4 LEMPEL-ZIV ESTIMATORS

Entropy can be interpreted as a measure of complexity. A complex sequence contains more information than a regular (predictable) sequence. The Lempel-Ziv (LZ)

algorithm efficiently decomposes a message into non-redundant substrings (Ziv and Lempel [1978]). We can estimate the compression rate of a message as a function of the number of items in a Lempel-Ziv dictionary relative to the length of the message. The intuition here is that complex messages have high entropy, which will require large dictionaries relative to the length of the string to be transmitted. Snippet 18.2 shows an implementation of the LZ compression algorithm.

SNIPPET 18.2 A LIBRARY BUILT USING THE LZ ALGORITHM

```
def lempelZiv_lib(msg):
    i,lib=1,[msg[0]]
    while i<len(msg):
        for j in xrange(i,len(msg)):
            msg_=msg[i:j+1]
            if msg_ not in lib:
                lib.append(msg_)
                break
        i=j+1
    return lib
```

Kontoyiannis [1998] attempts to make a more efficient use of the information available in a message. What follows is a faithful summary of the exposition in Gao et al. [2008]. We will reproduce the steps in that paper, while complementing them with code snippets that implement their ideas. Let us define L_i^n as 1 plus the length of the longest match found in the n bits prior to i,

$$L_i^n = 1 + \max\left\{l \middle| x_i^{i+l} = x_j^{j+l} \text{ for some } i-n \leq j \leq i-1, l \in [0,n]\right\}$$

Snippet 18.3 implements the algorithm that determines the length of the longest match. A few notes worth mentioning:

- The value n is constant for a sliding window, and $n = i$ for an expanding window.
- Computing L_i^n requires data x_{i-n}^{i+n-1}. In other words, index i must be at the center of the window. This is important in order to guarantee that both matching strings are of the same length. If they are not of the same length, l will have a limited range and its maximum will be underestimated.
- Some overlap between the two substrings is allowed, although obviously both cannot start at i.

SNIPPET 18.3 FUNCTION THAT COMPUTES THE LENGTH OF THE LONGEST MATCH

```
def matchLength(msg,i,n):
    # Maximum matched length+1, with overlap.
    # i>=n & len(msg)>=i+n
    subS=''
    for l in xrange(n):
        msg1=msg[i:i+l+1]
        for j in xrange(i-n,i):
            msg0=msg[j:j+l+1]
            if msg1==msg0:
                subS=msg1
                break # search for higher l.
    return len(subS)+1,subS # matched length + 1
```

Ornstein and Weiss [1993] formally established that

$$\lim_{n\to\infty} \frac{L_i^n}{\log_2[n]} = \frac{1}{H}$$

Kontoyiannis uses this result to estimate Shannon's entropy rate. He estimates the average $\frac{L_i^n}{\log_2[n]}$, and uses the reciprocal of that average to estimate H. The general intuition is, as we increase the available history, we expect that messages with high entropy will produce relatively shorter non-redundant substrings. In contrast, messages with low entropy will produce relatively longer non-redundant substrings as we parse through the message. Given a data realization $x_{-\infty}^{\infty}$, a window length $n \geq 1$, and a number of matches $k \geq 1$, the sliding-window LZ estimator $\hat{H}_{n,k} = \hat{H}_{n,k}[x_{-n+1}^{n+k-1}]$ is defined by

$$\hat{H}_{n,k} = \left[\frac{1}{k}\sum_{i=1}^{k}\frac{L_i^n}{\log_2[n]}\right]^{-1}$$

Similarly, the increasing window LZ estimator $\hat{H}_n = \hat{H}_n\left[x_0^{2n-1}\right]$, is defined by

$$\hat{H}_n = \left[\frac{1}{n}\sum_{i=2}^{n}\frac{L_i^i}{\log_2[i]}\right]^{-1}$$

The window size n is constant when computing $\hat{H}_{n,k}$, thus L_i^n. However, when computing $\hat{H}_n$, the window size increases with i, thus L_i^i, with $n = \frac{N}{2}$. In this

expanding window case the length of the message N should be an even number to ensure that all bits are parsed (recall that x_i is at the center, so for an odd-length message the last bit would not be read).

The above expressions have been derived under the assumptions of: stationarity, ergodicity, that the process takes finitely many values, and that the process satisfies the Doeblin condition. Intuitively, this condition requires that, after a finite number of steps r, no matter what has occurred before, anything can happen with positive probability. It turns out that this Doeblin condition can be avoided altogether if we consider a modified version of the above estimators:

$$\tilde{H}_{n,k} = \frac{1}{k}\sum_{i=1}^{k}\frac{\log_2[n]}{L_i^n}$$

$$\tilde{H}_{n} = \frac{1}{n}\sum_{i=2}^{n}\frac{\log_2[i]}{L_i^i}$$

One practical question when estimating $\tilde{H}_{n,k}$ is how to determine the window size n. Gao et al. [2008] argue that $k + n = N$ should be approximately equal to the message length. Considering that the bias of L_i^n is of order $\mathcal{O}[1/\log_2[n]]$ and the variance of L_i^n is order $\mathcal{O}[1/k]$, the bias/variance trade-off is balanced at around $k \approx \mathcal{O}[(\log_2[n])^2]$. That is, n could be chosen such that $N \approx n + (\log_2[n])^2$. For example, for $N = 2^8$, a balanced bias/variance window size would be $n \approx 198$, in which case $k \approx 58$.

Kontoyiannis [1998] proved that $\hat{H}[X]$ converges to Shannon's entropy rate with probability 1 as n approaches infinity. Snippet 18.4 implements the ideas discussed in Gao et al. [2008], which improve on Kontoyiannis [1997] by looking for the maximum redundancy between two substrings of the same size.

SNIPPET 18.4 IMPLEMENTATION OF ALGORITHMS DISCUSSED IN GAO ET AL. [2008]

```
def konto(msg,window=None):
    '''
    * Kontoyiannis' LZ entropy estimate, 2013 version (centered window).
    * Inverse of the avg length of the shortest non-redundant substring.
    * If non-redundant substrings are short, the text is highly entropic.
    * window==None for expanding window, in which case len(msg)%2==0
    * If the end of msg is more relevant, try konto(msg[::-1])
    '''
    out={'num':0,'sum':0,'subS':[]}
    if not isinstance(msg,str):msg=''.join(map(str,msg))
    if window is None:
        points=xrange(1,len(msg)/2+1)
```

```
    else:
        window=min(window,len(msg)/2)
        points=xrange(window,len(msg)-window+1)
    for i in points:
        if window is None:
            l,msg_=matchLength(msg,i,i)
            out['sum']+=np.log2(i+1)/l # to avoid Doeblin condition
        else:
            l,msg_=matchLength(msg,i,window)
            out['sum']+=np.log2(window+1)/l # to avoid Doeblin condition
        out['subS'].append(msg_)
        out['num']+=1
    out['h']=out['sum']/out['num']
    out['r']=1-out['h']/np.log2(len(msg)) # redundancy, 0<=r<=1
    return out
#--------------------------------------
if __name__=='__main__':
    msg='101010'
    print konto(msg*2)
    print konto(msg+msg[::-1])
```

One caveat of this method is that entropy rate is defined in the limit. In the words of Kontoyiannis, "we fix a large integer N as the size of our database." The theorems used by Kontoyiannis' paper prove asymptotic convergence; however, nowhere is a monotonicity property claimed. When a message is short, a solution may be to repeat the same message multiple times.

A second caveat is that, because the window for matching must be symmetric (same length for the dictionary as for the substring being matched), the last bit is only considered for matching if the message's length corresponds to an even number. One solution is to remove the first bit of a message with odd length.

A third caveat is that some final bits will be dismissed when preceded by irregular sequences. This is also a consequence of the symmetric matching window. For example, the entropy rate for "10000111" equals the entropy rate for "10000110," meaning that the final bit is irrelevant due to the unmatchable "11" in the sixth and seventh bit. When the end of the message is particularly relevant, a good solution may be to analyze the entropy of the reversed message. This not only ensures that the final bits (i.e., the initial ones after the reversing) are used, but actually they will be used to potentially match every bit. Following the previous example, the entropy rate of "11100001" is 0.96, while the entropy rate for "01100001' is 0.84.

18.5 ENCODING SCHEMES

Estimating entropy requires the encoding of a message. In this section we will review a few encoding schemes used in the literature, which are based on returns. Although

not discussed in what follows, it is advisable to encode information from fractionally (rather than integer) differentiated series (Chapter 4), as they still contain some memory.

18.5.1 Binary Encoding

Entropy rate estimation requires the discretization of a continuous variable, so that each value can be assigned a code from a finite alphabet. For example, a stream of returns r_t can be encoded according to the sign, 1 for $r_t > 0$, 0 for $r_t < 0$, removing cases where $r_t = 0$. Binary encoding arises naturally in the case of returns series sampled from price bars (i.e., bars that contain prices fluctuating between two symmetric horizontal barriers, centered around the start price), because $|r_t|$ is approximately constant.

When $|r_t|$ can adopt a wide range of outcomes, binary encoding discards potentially useful information. That is particularly the case when working with intraday time bars, which are affected by the heteroscedasticity that results from the inhomogeneous nature of tick data. One way to partially address this heteroscedasticity is to sample prices according to a subordinated stochastic process. Examples of that are trade bars and volume bars, which contain a fixed number of trades or trades for a fixed amount of volume (see Chapter 2). By operating in this non-chronological, market-driven clock, we sample more frequently during highly active periods, and less frequently during periods of less activity, hence regularizing the distribution of $|r_t|$ and reducing the need for a large alphabet.

18.5.2 Quantile Encoding

Unless price bars are used, it is likely that more than two codes will be needed. One approach consists in assigning a code to each r_t according to the quantile it belongs to. The quantile boundaries are determined using an in-sample period (training set). There will be the same number of observations assigned to each letter for the overall in-sample, and close to the same number of observations per letter out-of-sample. When using the method, some codes span a greater fraction of r_t's range than others. This uniform (in-sample) or close to uniform (out-of-sample) distribution of codes tends to increase entropy readings on average.

18.5.3 Sigma Encoding

As an alternative approach, rather than fixing the number of codes, we could let the price stream determine the actual dictionary. Suppose we fix a discretization step, σ. Then, we assign the value 0 to $r_t \in [\min\{r\}, \min\{r\} + \sigma)$, 1 to $r_t \in [\min\{r\} + \sigma, \min\{r\} + 2\sigma)$ and so on until every observation has been encoded with a total of ceil $\left[\frac{\max\{r\} - \min\{r\}}{\sigma}\right]$ codes, where ceil [.] is the ceiling function. Unlike quantile encoding, now each code covers the same fraction of r_t's range. Because codes are not uniformly distributed, entropy readings will tend to be smaller than in quantile

encoding on average; however, the appearance of a "rare" code will cause spikes in entropy readings.

18.6 ENTROPY OF A GAUSSIAN PROCESS

The entropy of an IID Normal random process (see Norwich [2003]) can be derived as

$$H = \frac{1}{2}\log[2\pi e\sigma^2]$$

For the standard Normal, $H \approx 1.42$. There are at least two uses of this result. First, it allows us to benchmark the performance of an entropy estimator. We can draw samples from a standard normal distribution, and find what combination of estimator, message length, and encoding gives us an entropy estimate $\hat{H}$ sufficiently close to the theoretically derived value H. For example, Figure 18.1 plots the bootstrapped distributions of entropy estimates under 10, 7, 5, and 2 letter encodings, on messages of length 100, using Kontoyiannis' method. For alphabets of at least 10 letters, the algorithm in Snippet 18.4 delivers the correct answer. When alphabets are too small, information is discarded and entropy is underestimated.

Second, we can use the above equation to connect entropy with volatility, by noting that $\sigma_H = \frac{e^{H-1/2}}{\sqrt{2\pi}}$. This gives us an entropy-implied volatility estimate, provided that returns are indeed drawn from a Normal distribution.

18.7 ENTROPY AND THE GENERALIZED MEAN

Here is an interesting way of thinking about entropy. Consider a set of real numbers $x = \{x_i\}_{i=1,\ldots,n}$ and weights $p = \{p_i\}_{i=1,\ldots,n}$, such that $0 \leq p_i \leq 1$, $\forall i$ and $\sum_{i=1}^{n} p_i = 1$. The generalized weighted mean of x with weights p on a power $q \neq 0$ is defined as

$$M_q[x, p] = \left(\sum_{i=1}^{n} p_i x_i^q\right)^{1/q}$$

For $q < 0$, we must require that $x_i > 0, \forall i$. The reason this is a generalized mean is that other means can be obtained as special cases:

- Minimum: $\lim_{q\to-\infty} M_q[x, p] = \min_i\{x_i\}$
- Harmonic mean: $M_{-1}[x, p] = \left(\sum_{i=1}^{n} p_i x_i^{-1}\right)^{-1}$
- Geometric mean: $\lim_{q\to 0} M_q[x, p] = e^{\sum_{i=1}^{n} p_i \log[x_i]} = \prod_{i=1}^{n} x_i^{p_i}$
- Arithmetic mean: $M_1[x, \{n^{-1}\}_{i=1,\ldots,n}] = n^{-1}\sum_{i=1}^{n} x_i$

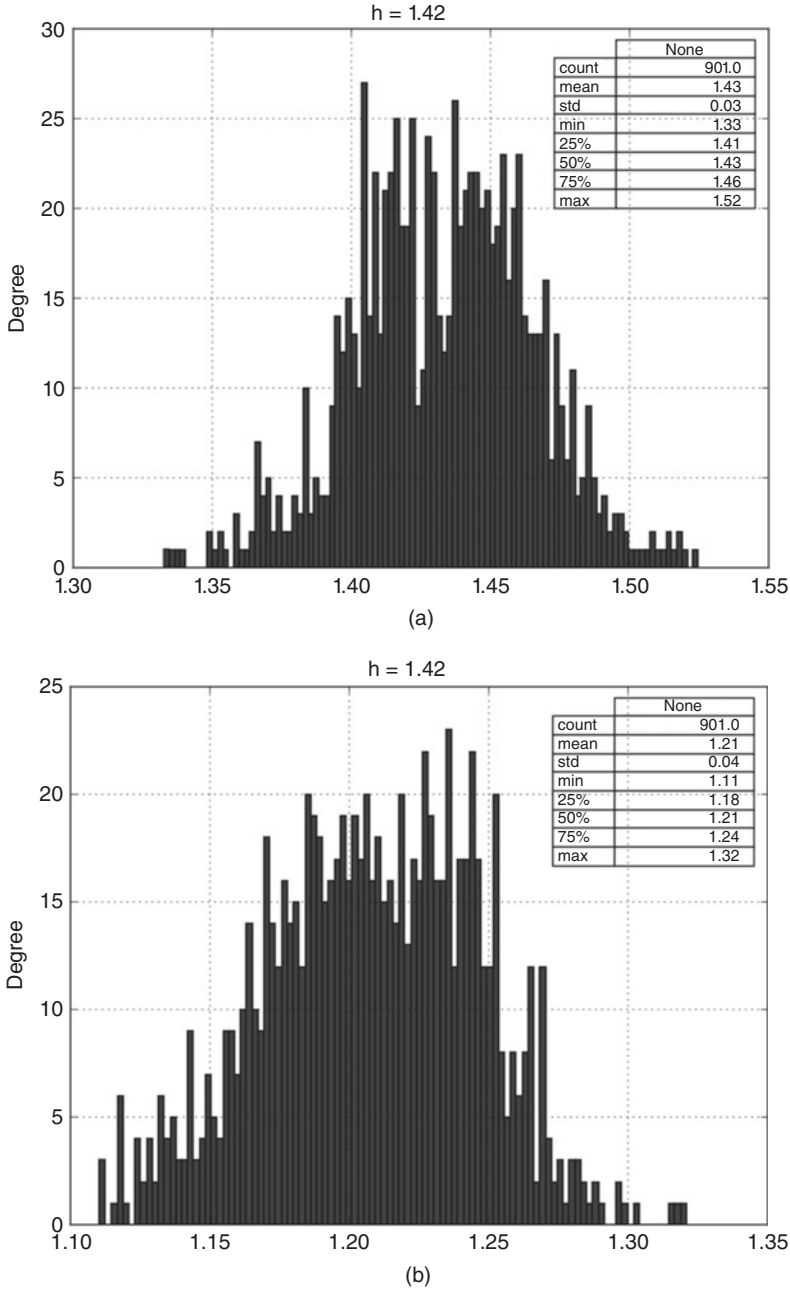


FIGURE 18.1 Distribution of entropy estimates under 10 (top), 7 (bottom), letter encodings, on messages of length 100

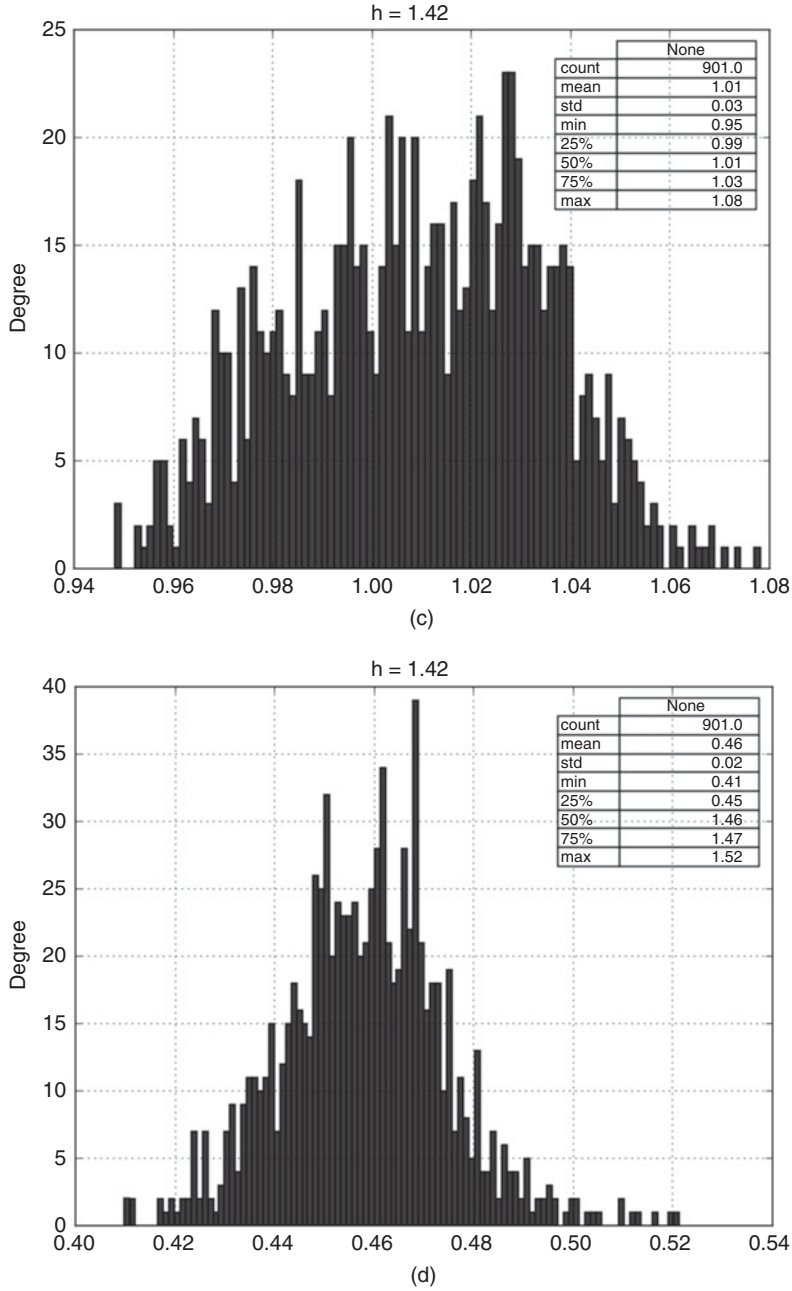


FIGURE 18.1 (*Continued*) Distribution of entropy estimates under 5 (top), and 2 (bottom) letter encodings, on messages of length 100

- Weighted mean: $M_1[x,p] = \sum_{i=1}^{n} p_i x_i$
- Quadratic mean: $M_2[x,p] = \left(\sum_{i=1}^{n} p_i x_i^2\right)^{1/2}$
- Maximum: $\lim_{q\to+\infty} M_q[x,p] = \max_i\{x_i\}$

In the context of information theory, an interesting special case is $x = \{p_i\}_{i=1,\ldots,n}$, hence

$$M_q[p,p] = \left(\sum_{i=1}^{n} p_i p_i^q\right)^{1/q}$$

Let us define the quantity $N_q[p] = \frac{1}{M_{q-1}[p,p]}$, for some $q \neq 1$. Again, for $q < 1$ in $N_q[p]$, we must have $p_i > 0, \forall i$. If $p_i = \frac{1}{k}$ for $k \in [1,n]$ different indices and $p_i = 0$ elsewhere, then the weight is spread evenly across k different items, and $N_q[p] = k$ for $q > 1$. In other words, $N_q[p]$ gives us the *effective number* or *diversity* of items in p, according to some weighting scheme set by q.

Using Jensen's inequality, we can prove that $\frac{\partial M_q[p,p]}{\partial q} \geq 0$, hence $\frac{\partial N_q[p]}{\partial q} \leq 0$. Smaller values of q assign a more uniform weight to elements of the partition, giving relatively more weight to less common elements, and $\lim_{q\to 0} N_q[p]$ is simply the total number of nonzero p_i.

Shannon's entropy is $H[p] = \sum_{i=1}^{n} -p_i \log[p_i] = -\log[\lim_{q\to 0} M_q[p]] = \log[\lim_{q\to 1} N_q[p]]$. This shows that entropy can be interpreted as the logarithm of the *effective number* of items in a list p, where $q \to 1$. Figure 18.2 illustrates how the log effective numbers for a family of randomly generated p arrays converge to

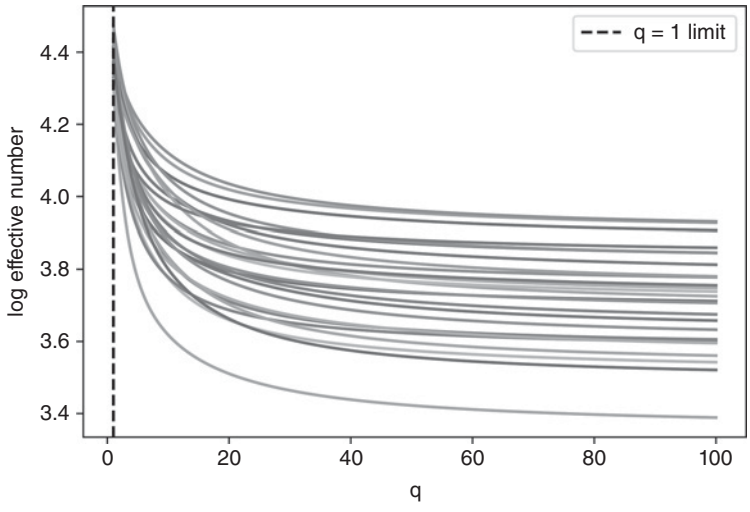


FIGURE 18.2 Log effective numbers for a family of randomly generated p arrays

Shannon's entropy as q approaches 1. Notice, as well, how their behavior stabilizes as q grows large.

Intuitively, entropy measures information as the level of *diversity* contained in a random variable. This intuition is formalized through the notion of generalized mean. The implication is that Shannon's entropy is a special case of a diversity measure (hence its connection with volatility). We can now define and compute alternative measures of diversity, other than entropy, where $q \neq 1$.

18.8 A FEW FINANCIAL APPLICATIONS OF ENTROPY

In this section we will introduce a few applications of entropy to the modelling of financial markets.

18.8.1 Market Efficiency

When arbitrage mechanisms exploit the complete set of opportunities, prices instantaneously reflect the full amount of available information, becoming unpredictable (i.e., a martingale), with no discernable patterns. Conversely, when arbitrage is not perfect, prices contain incomplete amounts of information, which gives rise to predictable patterns. Patterns occur when a string contains redundant information, which enables its compression. The entropy rate of a string determines its optimal compression rate. The higher the entropy, the lower the redundancy and the greater the informational content. Consequently, the entropy of a price string tells us the degree of market efficiency at a given point in time. A "decompressed" market is an efficient market, because price information is non-redundant. A "compressed" market is an inefficient market, because price information is redundant. Bubbles are formed in compressed (low entropy) markets.

18.8.2 Maximum Entropy Generation

In a series of papers, Fiedor [2014a, 2014b, 2014c] proposes to use Kontoyiannis [1997] to estimate the amount of entropy present in a price series. He argues that, out of the possible future outcomes, the one that maximizes entropy may be the most profitable, because it is the one that is least predictable by frequentist statistical models. It is the black swan scenario most likely to trigger stop losses, thus generating a feedback mechanism that will reinforce and exacerbate the move, resulting in runs in the signs of the returns time series.

18.8.3 Portfolio Concentration

Consider an NxN covariance matrix V, computed on returns. First, we compute an eigenvalue decomposition of the matrix, $VW = W\Lambda$. Second, we obtain the factor loadings vector as $f_\omega = W'\omega$, where ω is the vector of allocations, $\sum_{n=1}^{N} \omega_n = 1$.[1]

[1] Alternatively, we could have worked with a vector of holdings, should the covariance matrix had been computed on price changes.

Third, we derive the portion of risk contributed by each principal component (Bailey and López de Prado [2012]) as

$$\theta_i = \frac{[f_\omega]_i^2 \Lambda_{i,i}}{\sum_{n=1}^{N} [f_\omega]_n^2 \Lambda_{n,n}}$$

where $\sum_{i=1}^{N} \theta_i = 1$, and $\theta_i \in [0, 1]$, $\forall i = 1, \ldots, N$. Fourth, Meucci [2009] proposed the following entropy-inspired definition of portfolio concentration,

$$H = 1 - \frac{1}{N} e^{-\sum_{n=1}^{N} \theta_i \log[\theta_i]}$$

At first, this definition of portfolio concentration may sound striking, because θ_i is not a probability. The connection between this notion of concentration and entropy is due to the generalized mean, which we discussed in Chapter 18, Section 18.7.

18.8.4 Market Microstructure

Easley et al. [1996, 1997] showed that, when the odds of good news / bad news are even, the probability of informed trading (PIN) can be derived as

$$PIN = \frac{\alpha\mu}{\alpha\mu + 2\varepsilon}$$

where μ is the rate of arrival of informed traders, ε is the rate of arrival of uninformed traders, and α is the probability of an informational event. PIN can be interpreted as the fraction of orders that arise from informed traders relative to the overall order flow.

Within a volume bar of size V, we can classify ticks as buy or sell according to some algorithm, such as the tick rule or the Lee-Ready algorithm. Let V_τ^B be the sum of the volumes from buy ticks included in volume bar τ, and V_τ^S the sum of the volumes from sell ticks within volume bar τ. Easley et al. [2012a, 2012b] note that $\mathrm{E}[|V_\tau^B - V_\tau^S|] \approx \alpha\mu$ and that the expected total volume is $\mathrm{E}[V_\tau^B + V_\tau^S] = \alpha\mu + 2\varepsilon$. By using a volume clock (Easley et al. [2012c]), we can set the value of $\mathrm{E}[V_\tau^B + V_\tau^S] = \alpha\mu + 2\varepsilon = V$ exogenously. This means that, under a volume clock, PIN reduces to

$$VPIN = \frac{\alpha\mu}{\alpha\mu + 2\varepsilon} = \frac{\alpha\mu}{V} \approx \frac{1}{V}\mathrm{E}\left[\left|2V_\tau^B - V\right|\right] = \mathrm{E}\left[\left|2v_\tau^B - 1\right|\right]$$

where $v_\tau^B = \frac{V_\tau^B}{V}$. Note that $2v_\tau^B - 1$ represents the order flow imbalance, OI_τ, which is a bounded real-valued variable, where $OI_\tau \in [-1, 1]$. The VPIN theory thus provides a formal link between the probability of informed trading (PIN) and the persistency of order flow imbalances under a volume clock. See Chapter 19 for further details on this microstructural theory.

Persistent order flow imbalance is a necessary, non-sufficient condition for adverse selection. For market makers to provide liquidity to informed traders, that order flow imbalance $|OI_\tau|$ must also have been relatively unpredictable. In other words, market makers are not adversely selected when their prediction of order flow imbalance is accurate, even if $|OI_\tau| \gg 0$. In order to determine the probability of adverse selection, we must determine how unpredictable the order flow imbalance is. We can determine this by applying information theory.

Consider a long sequence of symbols. When that sequence contains few redundant patterns, it encompasses a level of complexity that makes it hard to describe and predict. Kolmogorov [1965] formulated this connection between redundancy and complexity. In information theory, lossless compression is the task of perfectly describing a sequence with as few bits as possible. The more redundancies a sequence contains, the greater compression rates can be achieved. Entropy characterizes the redundancy of a source, hence its Kolmogorov complexity and its predictability. We can use this connection between the redundancy of a sequence and its unpredictability (by market makers) to derive the probability of adverse selection.

Here we will discuss one particular procedure that derives the probability of adverse selection as a function of the complexity ingrained in the order flow imbalance. First, given a sequence of volume bars indexed by $\tau = 1, \dots, N$, each bar of size V, we determine the portion of volume classified as buy, $v_\tau^B \in [0, 1]$. Second, we compute the q-quantiles on $\{v_\tau^B\}$ that define a set K of q disjoint subsets, $K = \{K_1, \dots, K_q\}$. Third, we produce a mapping from each v_τ^B to one of the disjoint subsets, $f : v_\tau^B \to \{1, \dots, q\}$, where $f[v_\tau^B] = i \Leftrightarrow v_\tau^B \in K_i, \forall i \in [1, q]$. Fourth, we quantize $\{v_\tau^B\}$ by assigning to each value v_τ^B the index of the subset K it belongs to, $f[v_\tau^B]$. This results in a translation of the set of order imbalances $\{v_\tau^B\}$ into a quantized message $X = [f[v_1^B], f[v_2^B], \dots, f[v_N^B]]$. Fifth, we estimate the entropy $H[X]$ using Kontoyiannis' Lempel-Ziv algorithm. Sixth, we derive the cumulative distribution function, $F[H[X]]$, and use the time series of $\{F[H[X_\tau]]\}_{\tau=1,\dots,N}$ as a feature to predict adverse selection.

EXERCISES

18.1 Form dollar bars on E-mini S&P 500 futures:

- **(a)** Quantize the returns series using the binary method.
- **(b)** Quantize the returns series using the quantile encoding, using 10 letters.
- **(c)** Quantize the returns series using the sigma encoding, where σ is the standard deviation of all bar returns.
- **(d)** Compute the entropy of the three encoded series, using the plug-in method.
- **(e)** Compute the entropy of the three encoded series, using Kontoyiannis' method, with a window size of 100.

18.2 Using the bars from exercise 1:

- **(a)** Compute the returns series, $\{r_t\}$.
- **(b)** Encode the series as follows: 0 if $r_t r_{t-1} < 0$, and 1 if $r_t r_{t-1} \geq 0$.

(c) Partition the series into 1000 non-overlapping subsets of equal size (you may have to drop some observations at the beginning).

(d) Compute the entropy of each of the 1000 encoded subsets, using the plug-in method.

(e) Compute the entropy of each of the 1000 encoded subsets, using the Kontoyiannis method, with a window size of 100.

(f) Compute the correlation between results 2.d and 2.e.

18.3 Draw 1000 observations from a standard Normal distribution:

(a) What is the true entropy of this process?

(b) Label the observations according to 8 quantiles.

(c) Estimate the entropy using the plug-in method.

(d) Estimate the entropy using the Kontoyiannis method:

(i) using a window size of 10.

(ii) using a window size of 100.

18.4 Using the draws from exercise 3, $\{x_t\}_{t=1,\ldots,1000}$:

(a) Compute $y_t = \rho y_{t-1} + x_t$, where $\rho = .5$, $y_0 = 0$.

(b) Label $\{y_t\}$ the observations according to 8 quantiles.

(c) Estimate the entropy using the plug-in method.

(d) Estimate the entropy using the Kontoyiannis method

(i) using a window size of 10.

(ii) using a window size of 100.

18.5 Suppose a portfolio of 10 holdings with equal dollar allocations.

(a) The portion of the total risk contributed by the ith principal component is $\frac{1}{10}$, $i = 1, \ldots, 10$. What is the portfolio's entropy?

(b) The portion of the total risk contributed by the ith principal component is $1 - \frac{i}{55}$, $i = 1, \ldots, 10$. What is the portfolio's entropy?

(c) The portion of the total risk contributed by the ith principal component is $\alpha\frac{1}{10} + (1 - \alpha)(1 - \frac{i}{55})$, $i = 1, \ldots, 10$, $\alpha \in [0, 1]$. Plot the portolio's entropy as a function of α.

REFERENCES

Bailey, D. and M. López de Prado (2012): "Balanced baskets: A new approach to trading and hedging risks." *Journal of Investment Strategies*, Vol. 1, No. 4, pp. 21–62. Available at https://ssrn.com/abstract=2066170.

Easley D., M. Kiefer, M. O'Hara, and J. Paperman (1996): "Liquidity,information, and infrequently traded stocks." *Journal of Finance*, Vol. 51, No. 4, pp. 1405–1436.

Easley D., M. Kiefer and, M. O'Hara (1997): "The information content of the trading process." *Journal of Empirical Finance*, Vol. 4, No. 2, pp. 159–185.

Easley, D., M. López de Prado, and M. O'Hara (2012a): "Flow toxicity and liquidity in a high frequency world." *Review of Financial Studies*, Vol. 25, No. 5, pp. 1547–1493.

Easley, D., M. López de Prado, and M. O'Hara (2012b): "The volume clock: Insights into the high frequency paradigm." *Journal of Portfolio Management*, Vol. 39, No. 1, pp. 19–29.

Gao, Y., I. Kontoyiannis and E. Bienestock (2008): "Estimating the entropy of binary time series: Methodology, some theory and a simulation study." Working paper, arXiv. Available at https://arxiv.org/abs/0802.4363v1.

Fiedor, Pawel (2014a): "Mutual information rate-based networks in financial markets." Working paper, arXiv. Available at https://arxiv.org/abs/1401.2548.

Fiedor, Pawel (2014b): "Information-theoretic approach to lead-lag effect on financial markets." Working paper, arXiv. Available at https://arxiv.org/abs/1402.3820.

Fiedor, Pawel (2014c): "Causal non-linear financial networks." Working paper, arXiv. Available at https://arxiv.org/abs/1407.5020.

Hausser, J. and K. Strimmer (2009): "Entropy inference and the James-Stein estimator, with application to nonlinear gene association networks," *Journal of Machine Learning Research*, Vol. 10, pp. 1469–1484. http://www.jmlr.org/papers/volume10/hausser09a/hausser09a.pdf.

Kolmogorov, A. (1965): "Three approaches to the quantitative definition of information." *Problems in Information Transmission*, Vol. 1, No. 1, pp. 1–7.

Kontoyiannis, I. (1997): "The complexity and entropy of literary styles", *NSF Technical Report* # 97.

Kontoyiannis (1998): "Asymptotically optimal lossy Lempel-Ziv coding," *ISIT*, Cambridge, MA, August 16–August 21.

MacKay, D. (2003): *Information Theory, Inference, and Learning Algorithms, 1st ed*. Cambridge University Press.

Meucci, A. (2009): "Managing diversification." *Risk Magazine*, Vol. 22, pp. 74–79.

Norwich, K. (2003): *Information, Sensation and Perception, 1st ed*. Academic Press.

Ornstein, D.S. and B. Weiss (1993): "Entropy and data compression schemes." *IEEE Transactions on Information Theory*, Vol. 39, pp. 78–83.

Shannon, C. (1948): "A mathematical theory of communication." *Bell System Technical Journal*, Vol. 27, No. 3, pp. 379–423.

Ziv, J. and A. Lempel (1978): "Compression of individual sequences via variable-rate coding." *IEEE Transactions on Information Theory*, Vol. 24, No. 5, pp. 530–536.

BIBLIOGRAPHY

Easley, D., R. Engle, M. O'Hara, and L. Wu (2008): "Time-varying arrival rates of informed and uninformed traders." *Journal of Financial Econometrics*, Vol. 6, No. 2, pp. 171–207.

Easley, D., M. López de Prado, and M. O'Hara (2011): "The microstructure of the flash crash." *Journal of Portfolio Management*, Vol. 37, No. 2, pp. 118–128.

Easley, D., M. López de Prado, and M. O'Hara (2012c): "Optimal execution horizon." *Mathematical Finance*, Vol. 25, No. 3, pp. 640–672.

Gnedenko, B. and I. Yelnik (2016): "Minimum entropy as a measure of effective dimensionality." Working paper. Available at https://ssrn.com/abstract=2767549.

CHAPTER 19

Microstructural Features

19.1 MOTIVATION

Market microstructure studies "the process and outcomes of exchanging assets under explicit trading rules" (O'Hara [1995]). Microstructural datasets include primary information about the auctioning process, like order cancellations, double auction book, queues, partial fills, aggressor side, corrections, replacements, etc. The main source is Financial Information eXchange (FIX) messages, which can be purchased from exchanges. The level of detail contained in FIX messages provides researchers with the ability to understand how market participants conceal and reveal their intentions. That makes microstructural data one of the most important ingredients for building predictive ML features.

19.2 REVIEW OF THE LITERATURE

The depth and complexity of market microstructure theories has evolved over time, as a function of the amount and variety of the data available. The first generation of models used solely price information. The two foundational results from those early days are trade classification models (like the tick rule) and the Roll [1984] model. The second generation of models came after volume datasets started to become available, and researchers shifted their attention to study the impact that volume has on prices. Two examples for this generation of models are Kyle [1985] and Amihud [2002].

The third generation of models came after 1996, when Maureen O'Hara, David Easley, and others published their "probability of informed trading" (PIN) theory (Easley et al. [1996]). This constituted a major breakthrough, because PIN explained the bid-ask spread as the consequence of a sequential strategic decision between liquidity providers (market makers) and position takers (informed traders). Essentially, it illustrated that market makers were sellers of the option to be adversely selected by

informed traders, and the bid-ask spread is the premium they charge for that option. Easley et al. [2012a, 2012b] explain how to estimate VPIN, a high-frequency estimate of PIN under volume-based sampling.

These are the main theoretical frameworks used by the microstructural literature. O'Hara [1995] and Hasbrouck [2007] offer a good compendium of low-frequency microstructural models. Easley et al. [2013] present a modern treatment of high-frequency microstructural models.

19.3 FIRST GENERATION: PRICE SEQUENCES

The first generation of microstructural models concerned themselves with estimating the bid-ask spread and volatility as proxies for illiquidity. They did so with limited data and without imposing a strategic or sequential structure to the trading process.

19.3.1 The Tick Rule

In a double auction book, quotes are placed for selling a security at various price levels (offers) or for buying a security at various price levels (bids). Offer prices always exceed bid prices, because otherwise there would be an instant match. A trade occurs whenever a buyer matches an offer, or a seller matches a bid. Every trade has a buyer and a seller, but only one side initiates the trade.

The tick rule is an algorithm used to determine a trade's aggressor side. A buy-initiated trade is labeled "1", and a sell-initiated trade is labeled "-1", according to this logic:

$$b_t = \begin{cases} 1 & \text{if } \Delta p_t > 0 \\ -1 & \text{if } \Delta p_t < 0 \\ b_{t-1} & \text{if } \Delta p_t = 0 \end{cases}$$

where p_t is the price of the trade indexed by $t = 1, \ldots, T$, and b_0 is arbitrarily set to 1. A number of studies have determined that the tick rule achieves high classification accuracy, despite its relative simplicity (Aitken and Frino [1996]). Competing classification methods include Lee and Ready [1991] and Easley et al. [2016].

Transformations of the $\{b_t\}$ series can result in informative features. Such transformations include: (1) Kalman Filters on its future expected value, $\mathrm{E}_t[b_{t+1}]$; (2) structural breaks on such predictions (Chapter 17), (3) entropy of the $\{b_t\}$ sequence (Chapter 18); (4) t-values from Wald-Wolfowitz's tests of runs on $\{b_t\}$; (5) fractional differentiation of the cumulative $\{b_t\}$ series, $\sum_{i=1}^{t} b_i$ (Chapter 5); etc.

19.3.2 The Roll Model

Roll [1984] was one of the first models to propose an explanation for the effective bid-ask spread at which a security trades. This is useful in that bid-ask spreads are a function of liquidity, hence Roll's model can be seen as an early attempt to measure

the liquidity of a security. Consider a mid-price series $\{m_t\}$, where prices follow a Random Walk with no drift,

$$m_t = m_{t-1} + u_t$$

hence price changes $\Delta m_t = m_t - m_{t-1}$ are independently and identically drawn from a Normal distribution

$$\Delta m_t \sim N\left[0, \sigma_u^2\right]$$

These assumptions are, of course, against all empirical observations, which suggest that financial time series have a drift, they are heteroscedastic, exhibit serial dependency, and their returns distribution is non-Normal. But with a proper sampling procedure, as we saw in Chapter 2, these assumptions may not be too unrealistic. The observed prices, $\{p_t\}$, are the result of sequential trading against the bid-ask spread:

$$p_t = m_t + b_t c$$

where c is half the bid-ask spread, and $b_t \in \{-1, 1\}$ is the aggressor side. The Roll model assumes that buys and sells are equally likely, $P[b_t = 1] = P[b_t = -1] = \frac{1}{2}$, serially independent, $\mathrm{E}[b_t b_{t-1}] = 0$, and independent from the noise, $\mathrm{E}[b_t u_t] = 0$. Given these assumptions, Roll derives the values of c and σ_u^2 as follows:

$$\sigma^2\left[\Delta p_t\right] = \mathrm{E}\left[\left(\Delta p_t\right)^2\right] - \left(\mathrm{E}\left[\left(\Delta p_t\right)\right]\right)^2 = 2c^2 + \sigma_u^2$$

$$\sigma\left[\Delta p_t, \Delta p_{t-1}\right] = -c^2$$

resulting in $c = \sqrt{\max\{0, -\sigma[\Delta p_t, \Delta p_{t-1}]\}}$ and $\sigma_u^2 = \sigma^2[\Delta p_t] + 2\sigma[\Delta p_t, \Delta p_{t-1}]$. In conclusion, the bid-ask spread is a function of the serial covariance of price changes, and the true (unobserved) price's noise, excluding microstructural noise, is a function of the observed noise and the serial covariance of price changes.

The reader may question the need for Roll's model nowadays, when datasets include bid-ask prices at multiple book levels. One reason the Roll model is still in use, despite its limitations, is that it offers a relatively direct way to determine the *effective* bid-ask spread of securities that are either rarely traded, or where the published quotes are not representative of the levels at which market makers' are willing to provide liquidity (e.g., corporate, municipal, and agency bonds). Using Roll's estimates, we can derive informative features regarding the market's liquidity conditions.

19.3.3 High-Low Volatility Estimator

Beckers [1983] shows that volatility estimators based on high-low prices are more accurate than the standard estimators of volatility based on closing prices. Parkinson

[1980] derives that, for continuously observed prices following a geometric Brownian motion,

$$\mathrm{E}\left[\frac{1}{T}\sum_{t=1}^{T}\left(\log\left[\frac{H_t}{L_t}\right]\right)^2\right] = k_1\sigma_{HL}^2$$

$$\mathrm{E}\left[\frac{1}{T}\sum_{t=1}^{T}\left(\log\left[\frac{H_t}{L_t}\right]\right)\right] = k_2\sigma_{HL}$$

where $k_1 = 4\log[2]$, $k_2 = \sqrt{\frac{8}{\pi}}$, H_t is the high price for bar t, and L_t is the low price for bar t. Then the volatility feature σ_{HL} can be robustly estimated based on observed high-low prices.

19.3.4 Corwin and Schultz

Building on the work of Beckers [1983], Corwin and Schultz [2012] introduce a bid-ask spread estimator from high and low prices. The estimator is based on two principles: First, high prices are almost always matched against the offer, and low prices are almost always matched against the bid. The ratio of high-to-low prices reflects fundamental volatility as well as the bid-ask spread. Second, the component of the high-to-low price ratio that is due to volatility increases proportionately with the time elapsed between two observations.

Corwin and Schultz show that the spread, as a percentage of price, can be estimated as

$$S_t = \frac{2\left(e^{\alpha_t} - 1\right)}{1 + e^{\alpha_t}}$$

where

$$\alpha_t = \frac{\sqrt{2\beta_t} - \sqrt{\beta_t}}{3 - 2\sqrt{2}} - \sqrt{\frac{\gamma_t}{3 - 2\sqrt{2}}}$$

$$\beta_t = \mathrm{E}\left[\sum_{j=0}^{1}\left[\log\left(\frac{H_{t-j}}{L_{t-j}}\right)\right]^2\right]$$

$$\gamma_t = \left[\log\left(\frac{H_{t-1,t}}{L_{t-1,t}}\right)\right]^2$$

and $H_{t-1,t}$ is the high price over 2 bars ($t-1$ and t), whereas $L_{t-1,t}$ is the low price over 2 bars ($t-1$ and t). Because $\alpha_t < 0 \Rightarrow S_t < 0$, the authors recommend setting negative

alphas to 0 (see Corwin and Schultz [2012], p. 727). Snippet 19.1 implements this algorithm. The `corwinSchultz` function receives two arguments, a series dataframe with columns (`High`,`Low`), and an integer value `sl` that defines the sample length used to estimate β_t.

SNIPPET 19.1 IMPLEMENTATION OF THE CORWIN-SCHULTZ ALGORITHM

```
def getBeta(series,sl):
    hl=series[['High','Low']].values
    hl=np.log(hl[:,0]/hl[:,1])**2
    hl=pd.Series(hl,index=series.index)
    beta=pd.stats.moments.rolling_sum(hl,window=2)
    beta=pd.stats.moments.rolling_mean(beta,window=sl)
    return beta.dropna()
#———————————————————————————————-
def getGamma(series):
    h2=pd.stats.moments.rolling_max(series['High'],window=2)
    l2=pd.stats.moments.rolling_min(series['Low'],window=2)
    gamma=np.log(h2.values/l2.values)**2
    gamma=pd.Series(gamma,index=h2.index)
    return gamma.dropna()
#———————————————————————————————-
def getAlpha(beta,gamma):
    den=3-2*2**.5
    alpha=(2**.5-1)*(beta**.5)/den
    alpha-=(gamma/den)**.5
    alpha[alpha<0]=0 # set negative alphas to 0 (see p.727 of paper)
    return alpha.dropna()
#———————————————————————————————-
def corwinSchultz(series,sl=1):
    # Note: S<0 iif alpha<0
    beta=getBeta(series,sl)
    gamma=getGamma(series)
    alpha=getAlpha(beta,gamma)
    spread=2*(np.exp(alpha)-1)/(1+np.exp(alpha))
    startTime=pd.Series(series.index[0:spread.shape[0]],index=spread.index)
    spread=pd.concat([spread,startTime],axis=1)
    spread.columns=['Spread','Start_Time'] # 1st loc used to compute beta
    return spread
```

Note that volatility does not appear in the final Corwin-Schultz equations. The reason is that volatility has been replaced by its high/low estimator. As a byproduct of this model, we can derive the Becker-Parkinson volatility as shown in Snippet 19.2.

SNIPPET 19.2 ESTIMATING VOLATILITY FOR HIGH-LOW PRICES

```
def getSigma(beta,gamma):
    k2=(8/np.pi)**.5
    den=3-2*2**.5
    sigma=(2**-.5-1)*beta**.5/(k2*den)
    sigma+=(gamma/(k2**2*den))**.5
    sigma[sigma<0]=0
    return sigma
```

This procedure is particularly helpful in the corporate bond market, where there is no centralized order book, and trades occur through bids wanted in competition (BWIC). The resulting feature, bid-ask spread S, can be estimated recursively over a rolling window, and values can be smoothed using a Kalman filter.

19.4 SECOND GENERATION: STRATEGIC TRADE MODELS

Second generation microstructural models focus on understanding and measuring illiquidity. Illiquidity is an important informative feature in financial ML models, because it is a risk that has an associated premium. These models have a stronger theoretical foundation than first-generation models, in that they explain trading as the strategic interaction between informed and uninformed traders. In doing so, they pay attention to signed volume and order flow imbalance.

Most of these features are estimated through regressions. In practice, I have observed that the t-values associated with these microstructural estimates are more informative than the (mean) estimates themselves. Although the literature does not mention this observation, there is a good argument for preferring features based on t-values over features based on mean values: t-values are re-scaled by the standard deviation of the estimation error, which incorporates another dimension of information absent in mean estimates.

19.4.1 Kyle's Lambda

Kyle [1985] introduced the following strategic trade model. Consider a risky asset with terminal value $v \sim N[p_0, \Sigma_0]$, as well as two traders:

- A noise trader who trades a quantity $u = N[0, \sigma_u^2]$, independent of v.
- An informed trader who knows v and demands a quantity x, through a market order.

The market maker observes the total order flow $y = x + u$, and sets a price p accordingly. In this model, market makers cannot distinguish between orders from noise

traders and informed traders. They adjust prices as a function of the order flow imbalance, as that may indicate the presence of an informed trader. Hence, there is a positive relationship between price change and order flow imbalance, which is called market impact.

The informed trader conjectures that the market maker has a linear price adjustment function, $p = \lambda y + \mu$, where λ is an inverse measure of liquidity. The informed trader's profits are $\pi = (v - p)x$, which are maximized at $x = \frac{v-\mu}{2\lambda}$, with second order condition $\lambda > 0$.

Conversely, the market maker conjectures that the informed trader's demand is a linear function of v: $x = \alpha + \beta v$, which implies $\alpha = -\frac{\mu}{2\lambda}$ and $\beta = \frac{1}{2\lambda}$. Note that lower liquidity means higher λ, which means lower demand from the informed trader.

Kyle argues that the market maker must find an equilibrium between profit maximization and market efficiency, and that under the above linear functions, the only possible solution occurs when

$$\mu = p_0$$

$$\alpha = p_0 \sqrt{\frac{\sigma_u^2}{\Sigma_0}}$$

$$\lambda = \frac{1}{2} \sqrt{\frac{\Sigma_0}{\sigma_u^2}}$$

$$\beta = \sqrt{\frac{\sigma_u^2}{\Sigma_0}}$$

Finally, the informed trader's expected profit can be rewritten as

$$\mathrm{E}[\pi] = \frac{\left(v - p_0\right)^2}{2} \sqrt{\frac{\sigma_u^2}{\Sigma_0}} = \frac{1}{4\lambda} \left(v - p_0\right)^2$$

The implication is that the informed trader has three sources of profit:

- The security's mispricing.
- The variance of the noise trader's net order flow. The higher the noise, the easier the informed trader can conceal his intentions.
- The reciprocal of the terminal security's variance. The lower the volatility, the easier to monetize the mispricing.

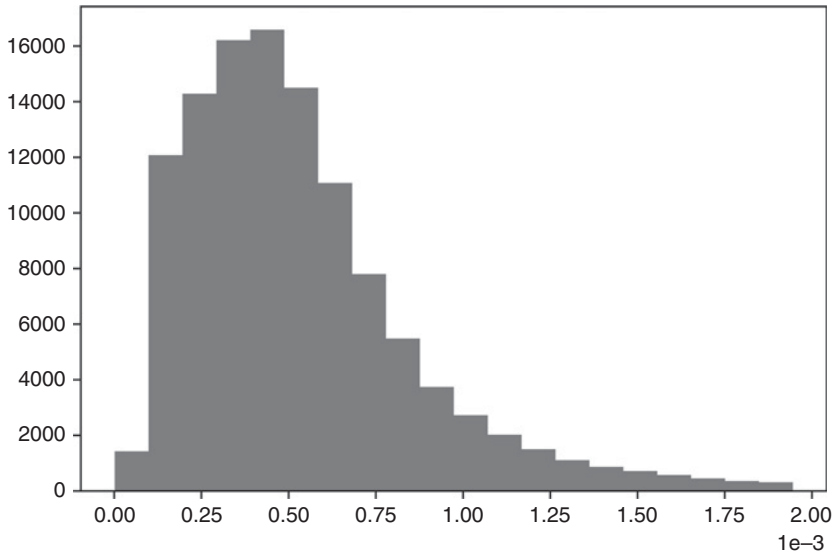


FIGURE 19.1 Kyle's Lambdas Computed on E-mini S&P 500 Futures

In Kyle's model, the variable λ captures price impact. Illiquidity increases with uncertainty about v and decreases with the amount of noise. As a feature, it can be estimated by fitting the regression

$$\Delta p_t = \lambda \left(b_t V_t\right) + \varepsilon_t$$

where $\{p_t\}$ is the time series of prices, $\{b_t\}$ is the time series of aggressor flags, $\{V_t\}$ is the time series of traded volumes, and hence $\{b_t V_t\}$ is the time series of signed volume or net order flow. Figure 19.1 plots the histogram of Kyle's lambdas estimated on the E-mini S&P 500 futures series.

19.4.2 Amihud's Lambda

Amihud [2002] studies the positive relationship between absolute returns and illiquidity. In particular, he computes the daily price response associated with one dollar of trading volume, and argues its value is a proxy of price impact. One possible implementation of this idea is

$$\left|\Delta \log\left[\tilde{p}_\tau\right]\right| = \lambda \sum_{t \in B_\tau} \left(p_t V_t\right) + \varepsilon_\tau$$

where B_τ is the set of trades included in bar τ, $\tilde{p}_\tau$ is the closing price of bar τ, and $p_t V_t$ is the dollar volume involved in trade $t \in B_\tau$. Despite its apparent simplicity, Hasbrouck [2009] found that daily Amihud's lambda estimates exhibit a high rank

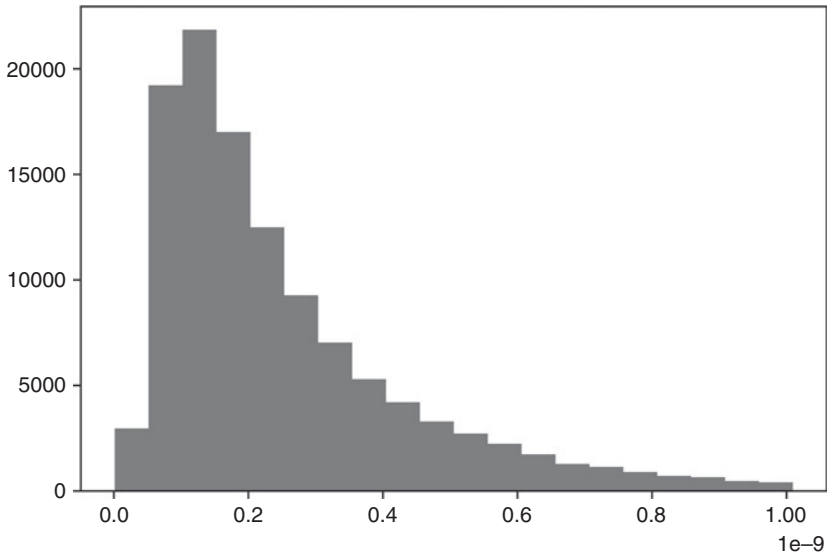


FIGURE 19.2 Amihud's lambdas estimated on E-mini S&P 500 futures

correlation to intraday estimates of effective spread. Figure 19.2 plots the histogram of Amihud's lambdas estimated on the E-mini S&P 500 futures series.

19.4.3 Hasbrouck's Lambda

Hasbrouck [2009] follows up on Kyle's and Amihud's ideas, and applies them to estimating the price impact coefficient based on trade-and-quote (TAQ) data. He uses a Gibbs sampler to produce a Bayesian estimation of the regression specification

$$\log\left[\tilde{p}_{i,\tau}\right] - \log\left[\tilde{p}_{i,\tau-1}\right] = \lambda_i \sum_{t \in B_{i,\tau}} \left(b_{i,t}\sqrt{p_{i,t}V_{i,t}}\right) + \varepsilon_{i,\tau}$$

where $B_{i,\tau}$ is the set of trades included in bar τ for security i, with $i = 1, \ldots, I$, $\tilde{p}_{i,\tau}$ is the closing price of bar τ for security i, $b_{i,t} \in \{-1, 1\}$ indicates whether trade $t \in B_{i,\tau}$ was buy-initiated or sell-initiated; and $p_{i,t}V_{i,t}$ is the dollar volume involved in trade $t \in B_{i,\tau}$. We can then estimate λ_i for every security i, and use it as a feature that approximates the effective cost of trading (market impact).

Consistent with most of the literature, Hasbrouck recommends 5-minute time-bars for sampling ticks. However, for the reasons discussed in Chapter 2, better results can be achieved through stochastic sampling methods that are synchronized with market activity. Figure 19.3 plots the histogram of Hasbrouck's lambdas estimated on the E-mini S&P 500 futures series.

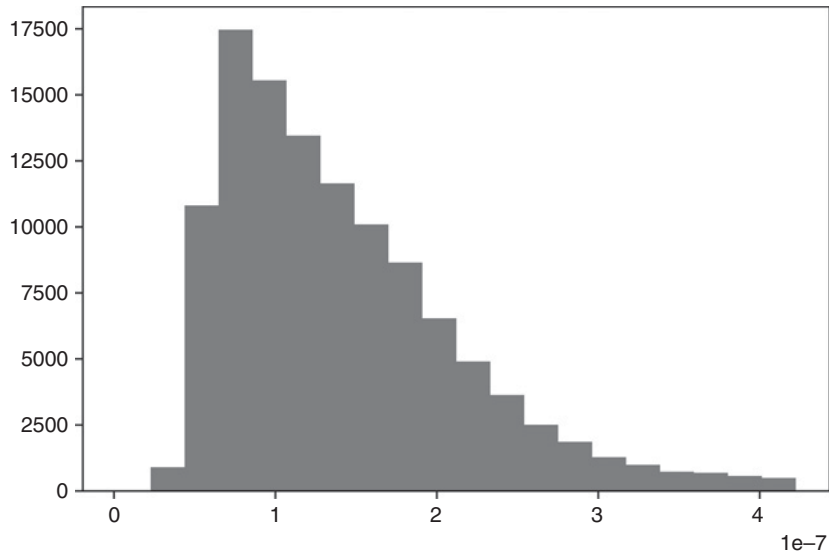


FIGURE 19.3 Hasbrouck's lambdas estimated on E-mini S&P 500 futures

19.5 THIRD GENERATION: SEQUENTIAL TRADE MODELS

As we have seen in the previous section, strategic trade models feature a single informed trader who can trade at multiple times. In this section we will discuss an alternative kind of model, where randomly selected traders arrive at the market sequentially and independently.

Since their appearance, sequential trade models have become very popular among market makers. One reason is, they incorporate the sources of uncertainty faced by liquidity providers, namely the probability that an informational event has taken place, the probability that such event is negative, the arrival rate of noise traders, and the arrival rate of informed traders. With those variables, market makers must update quotes dynamically, and manage their inventories.

19.5.1 Probability of Information-based Trading

Easley et al. [1996] use trade data to determine the probability of information-based trading (PIN) of individual securities. This microstructure model views trading as a game between market makers and position takers that is repeated over multiple trading periods.

Denote a security's price as S, with present value S_0. However, once a certain amount of new information has been incorporated into the price, S will be either S_B (bad news) or S_G (good news). There is a probability α that new information will arrive within the timeframe of the analysis, a probability δ that the news will be bad,

and a probability $(1-\delta)$ that the news will be good. These authors prove that the expected value of the security's price can then be computed at time t as

$$\mathrm{E}\left[S_t\right] = \left(1-\alpha_t\right) S_0 + \alpha_t \left[\delta_t S_B + \left(1-\delta_t\right) S_G\right]$$

Following a Poisson distribution, informed traders arrive at a rate μ, and uninformed traders arrive at a rate ε. Then, in order to avoid losses from informed traders, market makers reach breakeven at a bid level B_t,

$$\mathrm{E}\left[B_t\right] = \mathrm{E}\left[S_t\right] - \frac{\mu\alpha_t\delta_t}{\varepsilon + \mu\alpha_t\delta_t}\left(\mathrm{E}\left[S_t\right] - S_B\right)$$

and the breakeven ask level A_t at time t must be,

$$\mathrm{E}\left[A_t\right] = \mathrm{E}\left[S_t\right] + \frac{\mu\alpha_t\left(1-\delta_t\right)}{\varepsilon + \mu\alpha_t\left(1-\delta_t\right)}\left(S_G - \mathrm{E}\left[S_t\right]\right)$$

It follows that the breakeven bid-ask spread is determined as

$$\mathrm{E}\left[A_t - B_t\right] = \frac{\mu\alpha_t\left(1-\delta_t\right)}{\varepsilon + \mu\alpha_t\left(1-\delta_t\right)}\left(S_G - \mathrm{E}\left[S_t\right]\right) + \frac{\mu\alpha_t\delta_t}{\varepsilon + \mu\alpha_t\delta_t}\left(\mathrm{E}\left[S_t\right] - S_B\right)$$

For the standard case when $\delta_t = \frac{1}{2}$, we obtain

$$\delta_t = \frac{1}{2} \Rightarrow \mathrm{E}\left[A_t - B_t\right] = \frac{\alpha_t\mu}{\alpha_t\mu + 2\varepsilon}\left(S_G - S_B\right)$$

This equation tells us that the critical factor that determines the price range at which market makers provide liquidity is

$$PIN_t = \frac{\alpha_t\mu}{\alpha_t\mu + 2\varepsilon}$$

The subscript t indicates that the probabilities α and δ are estimated at that point in time. The authors apply a Bayesian updating process to incorporate information after each trade arrives to the market.

In order to determine the value PIN_t, we must estimate four non-observable parameters, namely $\{\alpha, \delta, \mu, \varepsilon\}$. A maximum-likelihood approach is to fit a mixture of three Poisson distributions,

$$\begin{aligned}\mathrm{P}[V^B, V^S] = {} & (1-\alpha)\mathrm{P}[V^B, \varepsilon]\mathrm{P}[V^S, \varepsilon] \\ & + \alpha(\delta\mathrm{P}[V^B, \varepsilon]\mathrm{P}[V^S, \mu+\varepsilon] + (1-\delta)\mathrm{P}[V^B, \mu+\varepsilon]\mathrm{P}[V^S, \varepsilon])\end{aligned}$$

where V^B is the volume traded against the ask (buy-initiated trades), and V^S is the volume traded against the bid (sell-initiated trades).

19.5.2 Volume-Synchronized Probability of Informed Trading

Easley et al. [2008] proved that

$$\begin{aligned}\mathrm{E}\left[V^B - V^S\right] &= (1-\alpha)(\varepsilon-\varepsilon) + \alpha(1-\delta)(\varepsilon-(\mu+\varepsilon)) + \alpha\delta(\mu+\varepsilon-\varepsilon) \\ &= \alpha\mu(1-2\delta)\end{aligned}$$

and in particular, for a sufficiently large μ,

$$\mathrm{E}[|V^B - V^S|] \approx \alpha\mu$$

Easley et al. [2011] proposed a high-frequency estimate of PIN, which they named volume-synchronized probability of informed trading (VPIN). This procedure adopts a *volume clock*, which synchronizes the data sampling with market activity, as captured by volume (see Chapter 2). We can then estimate

$$\frac{1}{n}\sum_{\tau=1}^{n}\left|V_\tau^B - V_\tau^S\right| \approx \alpha\mu$$

where V_τ^B is the sum of volumes from buy-initiated trades within volume bar τ, V_τ^S is the sum of volumes from sell-initiated trades within volume bar τ, and n is the number of bars used to produce this estimate. Because all volume bars are of the same size, V, we know that by construction

$$\frac{1}{n}\sum_{\tau=1}^{n}\left(V_\tau^B + V_\tau^S\right) = V = \alpha\mu + 2\varepsilon$$

Hence, PIN can be estimated in high-frequency as

$$VPIN_\tau = \frac{\sum_{\tau=1}^{n}\left|V_\tau^B - V_\tau^S\right|}{\sum_{\tau=1}^{n}\left(V_\tau^B + V_\tau^S\right)} = \frac{\sum_{\tau=1}^{n}\left|V_\tau^B - V_\tau^S\right|}{nV}$$

For additional details and case studies of VPIN, see Easley et al. [2013]. Using linear regressions, Andersen and Bondarenko [2013] concluded that VPIN is not a good predictor of volatility. However, a number of studies have found that VPIN indeed has predictive power: Abad and Yague [2012], Bethel et al. [2012], Cheung et al. [2015], Kim et al. [2014], Song et al. [2014], Van Ness et al. [2017], and Wei et al. [2013], to cite a few. In any case, linear regression is a technique that was already known to 18th-century mathematicians (Stigler [1981]), and economists should not be surprised when it fails to recognize complex non-linear patterns in 21st-century financial markets.

19.6 ADDITIONAL FEATURES FROM MICROSTRUCTURAL DATASETS

The features we have studied in Sections 19.3 to 19.5 were suggested by market microstructure theory. In addition, we should consider alternative features that, although not suggested by the theory, we suspect carry important information about the way market participants operate, and their future intentions. In doing so, we will harness the power of ML algorithms, which can learn how to use these features without being specifically directed by theory.

19.6.1 Distibution of Order Sizes

Easley et al. [2016] study the frequency of trades per trade size, and find that trades with round sizes are abnormally frequent. For example, the frequency rates quickly decay as a function of trade size, with the exception of round trade sizes $\{5, 10, 20, 25, 50, 100, 200, \ldots\}$. These authors attribute this phenomenon to so-called "mouse" or "GUI" traders, that is, human traders who send orders by clicking buttons on a GUI (Graphical User Interface). In the case of the E-mini S&P 500, for example, size 10 is 2.9 times more frequent than size 9; size 50 is 10.9 times more likely than size 49; size 100 is 16.8 times more frequent than size 99; size 200 is 27.2 times more likely than size 199; size 250 is 32.5 times more frequent than size 249; size 500 is 57.1 times more frequent than size 499. Such patterns are not typical of "silicon traders," who usually are programmed to randomize trades to disguise their footprint in markets.

A useful feature may be to determine the normal frequency of round-sized trades, and monitor deviations from that expected value. The ML algorithm could, for example, determine if a larger-than-usual proportion of round-sized trades is associated with trends, as human traders tend to bet with a fundamental view, belief, or conviction. Conversely, a lower-than-usual proportion of round-sized trades may increase the likelihood that prices will move sideways, as silicon traders do not typically hold long-term views.

19.6.2 Cancellation Rates, Limit Orders, Market Orders

Eisler et al. [2012] study the impact of market orders, limit orders, and quote cancellations. These authors find that small stocks respond differently than large stocks to these events. They conclude that measuring these magnitudes is relevant to model the dynamics of the bid-ask spread.

Easley et al. [2012] also argue that large quote cancellation rates may be indicative of low liquidity, as participants are publishing quotes that do not intend to get filled. They discuss four categories of predatory algorithms:

- **Quote stuffers:** They engage in "latency arbitrage." Their strategy involves overwhelming an exchange with messages, with the sole intention of slowing down competing algorithms, which are forced to parse messages that only the originators know can be ignored.

- **Quote danglers:** This strategy sends quotes that force a squeezed trader to chase a price against her interests. O'Hara [2011] presents evidence of their disruptive activities.
- **Liquidity squeezers:** When a distressed large investor is forced to unwind her position, predatory algorithms trade in the same direction, draining as much liquidity as possible. As a result, prices overshoot and they make a profit (Carlin et al. [2007]).
- **Pack hunters:** Predators hunting independently become aware of one another's activities, and form a pack in order to maximize the chances of triggering a cascading effect (Donefer [2010], Fabozzi et al. [2011], Jarrow and Protter [2011]). NANEX [2011] shows what appears to be pack hunters forcing a stop loss. Although their individual actions are too small to raise the regulator's suspicion, their collective action may be market-manipulative. When that is the case, it is very hard to prove their collusion, since they coordinate in a decentralized, spontaneous manner.

These predatory algorithms utilize quote cancellations and various order types in an attempt to adversely select market makers. They leave different signatures in the trading record, and measuring the rates of quote cancellation, limit orders, and market orders can be the basis for useful features, informative of their intentions.

19.6.3 Time-Weighted Average Price Execution Algorithms

Easley et al. [2012] demonstrate how to recognize the presence of execution algorithms that target a particular time-weighted average price (TWAP). A TWAP algorithm is an algorithm that slices a large order into small ones, which are submitted at regular time intervals, in an attempt to achieve a pre-defined time-weighted average price. These authors take a sample of E-mini S&P 500 futures trades between November 7, 2010, and November 7, 2011. They divide the day into 24 hours, and for every hour, they add the volume traded at each second, irrespective of the minute. Then they plot these aggregate volumes as a surface where the x-axis is assigned to volume per second, the y-axis is assigned to hour of the day, and the z-axis is assigned to the aggregate volume. This analysis allows us to see the distribution of volume within each minute as the day passes, and search for low-frequency traders executing their massive orders on a chronological time-space. The largest concentrations of volume within a minute tend to occur during the first few seconds, for almost every hour of the day. This is particularly true at 00:00–01:00 GMT (around the open of Asian markets), 05:00–09:00 GMT (around the open of U.K. and European equities), 13:00–15:00 GMT (around the open of U.S. equities), and 20:00–21:00 GMT (around the close of U.S. equities).

A useful ML feature may be to evaluate the order imbalance at the beginning of every minute, and determine whether there is a persistent component. This can then be used to front-run large institutional investors, while the larger portion of their TWAP order is still pending.

19.6.4 Options Markets

Muravyev et al. [2013] use microstructural information from U.S. stocks and options to study events where the two markets disagree. They characterize such disagreement by deriving the underlying bid-ask range implied by the put-call parity quotes and comparing it to the actual bid-ask range of the stock. They conclude that disagreements tend to be resolved in favor of stock quotes, meaning that option *quotes* do not contain economically significant information. At the same time, they do find that option *trades* contain information not included in the stock price. These findings will not come as a surprise to portfolio managers used to trade relatively illiquid products, including stock options. Quotes can remain irrational for prolonged periods of time, even as sparse prices are informative.

Cremers and Weinbaum [2010] find that stocks with relatively expensive calls (stocks with both a high volatility spread and a high change in the volatility spread) outperform stocks with relatively expensive puts (stocks with both a low volatility spread and a low change in the volatility spread) by 50 basis points per week. This degree of predictability is larger when option liquidity is high and stock liquidity is low.

In line with these observations, useful features can be extracted from computing the put-call implied stock price, derived from option trades. Futures prices only represent mean or expected future values. But option prices allow us to derive the entire distribution of outcomes being priced. An ML algorithm can search for patterns across the Greek letters quoted at various strikes and expiration dates.

19.6.5 Serial Correlation of Signed Order Flow

Toth et al. [2011] study the signed order flow of London Stock Exchange stocks, and find that order signs are positively autocorrelated for many days. They attribute this observation to two candidate explanations: Herding and order splitting. They conclude that on timescales of less than a few hours, the persistence of order flow is overwhelmingly due to splitting rather than herding.

Given that market microstructure theory attributes the persistency of order flow imbalance to the presence of informed traders, it makes sense to measure the strength of such persistency through the serial correlation of the signed volumes. Such a feature would be complementary to the features we studied in Section 19.5.

19.7 WHAT IS MICROSTRUCTURAL INFORMATION?

Let me conclude this chapter by addressing what I consider to be a major flaw in the market microstructure literature. Most articles and books on this subject study asymmetric information, and how strategic agents utilize it to profit from market makers. But how is information exactly defined in the context of trading? Unfortunately, there is no widely accepted definition of information in a microstructural sense, and the literature uses this concept in a surprisingly loose, rather informal way (López de Prado [2017]). This section proposes a proper definition of information, founded on signal processing, that can be applied to microstructural studies.

Consider a features matrix $X = \{X_t\}_{t=1,\ldots,T}$ that contains information typically used by market makers to determine whether they should provide liquidity at a particular level, or cancel their passive quotes. For example, the columns could be all of the features discussed in this chapter, like VPIN, Kyle's lambda, cancellation rates, etc. Matrix X has one row for each decision point. For example, a market maker may reconsider the decision to either provide liquidity or pull out of the market every time 10,000 contracts are traded, or whenever there is a significant change in prices (recall sampling methods in Chapter 2), etc. First, we derive an array $y = \{y_t\}_{t=1,\ldots,T}$ that assigns a label 1 to an observation that resulted in a market-making profit, and labels as 0 an observation that resulted in a market-making loss (see Chapter 3 for labeling methods). Second, we fit a classifier on the training set (X, y). Third, as new out-of-sample observations arrive $\tau > T$, we use the fit classifier to predict the label $\hat{y}_\tau = \mathrm{E}_\tau[y_\tau|X]$. Fourth, we derive the cross-entropy loss of these predictions, L_τ, as described in Chapter 9, Section 9.4. Fifth, we fit a kernel density estimator (KDE) on the array of negative cross-entropy losses, $\{-L_t\}_{t=T+1,\ldots,\tau}$, to derive its cumulative distribution function, F. Sixth, we estimate the microstructural information at time t as $\phi_\tau = F[-L_\tau]$, where $\phi_\tau \in (0, 1)$.

This microstructural information can be understood as the complexity faced by market makers' decision models. Under normal market conditions, market makers produce *informed forecasts* with low cross-entropy loss, and are able to profit from providing liquidity to position takers. However, in the presence of (asymmetrically) informed traders, market makers produce *uninformed forecasts*, as measured by high cross-entropy loss, and they are adversely selected. In other words, microstructural information can only be defined and measured relative to the predictive power of market makers. The implication is that $\{\phi_\tau\}$ should become an important feature in your financial ML toolkit.

Consider the events of the flash crash of May 6, 2010. Market makers wrongly predicted that their passive quotes sitting on the bid could be filled and sold back at a higher level. The crash was not caused by a single inaccurate prediction, but by the accumulation of thousands of prediction errors (Easley et al. [2011]). If market makers had monitored the rising cross-entropy loss of their predictions, they would have recognized the presence of informed traders and the dangerously rising probability of adverse selection. That would have allowed them to widen the bid-ask spread to levels that would have stopped the order flow imbalance, as sellers would no longer have been willing to sell at those discounts. Instead, market makers kept providing liquidity to sellers at exceedingly generous levels, until eventually they were forced to stop-out, triggering a liquidity crisis that shocked markets, regulators, and academics for months and years.

EXERCISES

19.1 From a time series of E-mini S&P 500 futures tick data,

- **(a)** Apply the tick rule to derive the series of trade signs.
- **(b)** Compare to the aggressor's side, as provided by the CME (FIX tag 5797). What is the accuracy of the tick rule?

(c) Select the cases where FIX tag 5797 disagrees with the tick rule.

(i) Can you see anything distinct that would explain the disagreement?

(ii) Are these disagreements associated with large price jumps? Or high cancelation rates? Or thin quoted sizes?

(iii) Are these disagreements more likely to occur during periods of high or low market activity?

19.2 Compute the Roll model on the time series of E-mini S&P 500 futures tick data.

(a) What are the estimated values of σ_u^2 and c?

(b) Knowing that this contract is one of the most liquid products in the world, and that it trades at the tightest possible bid-ask spread, are these values in line with your expectations?

19.3 Compute the high-low volatility estimator (Section19.3.3.) on E-mini S&P 500 futures:

(a) Using weekly values, how does this differ from the standard deviation of close-to-close returns?

(b) Using daily values, how does this differ from the standard deviation of close-to-close returns?

(c) Using dollar bars, for an average of 50 bars per day, how does this differ from the standard deviation of close-to-close returns?

19.4 Apply the Corwin-Schultz estimator to a daily series of E-mini S&P 500 futures.

(a) What is the expected bid-ask spread?

(b) What is the implied volatility?

(c) Are these estimates consistent with the earlier results, from exercises 2 and 3?

19.5 Compute Kyle's lambda from:

(a) tick data.

(b) a time series of dollar bars on E-mini S&P 500 futures, where

(i) b_t is the volume-weighted average of the trade signs.

(ii) V_t is the sum of the volumes in that bar.

(iii) Δp_t is the change in price between two consecutive bars.

19.6 Repeat exercise 5, this time applying Hasbrouck's lambda. Are results consistent?

19.7 Repeat exercise 5, this time applying Amihud's lambda. Are results consistent?

19.8 Form a time series of volume bars on E-mini S&P 500 futures,

(a) Compute the series of VPIN on May 6, 2010 (flash crash).

(b) Plot the series of VPIN and prices. What do you see?

19.9 Compute the distribution of order sizes for E-mini S&P 500 futures

(a) Over the entire period.

(b) For May 6, 2010.

(c) Conduct a Kolmogorov-Smirnov test on both distributions. Are they significantly different, at a 95% confidence level?

19.10 Compute a time series of daily quote cancellations rates, and the portion of market orders, on the E-mini S&P 500 futures dataset.

(a) What is the correlation between these two series? Is it statistically significant?

(b) What is the correlation between the two series and daily volatility? Is this what you expected?

19.11 On the E-mini S&P 500 futures tick data:

(a) Compute the distribution of volume executed within the first 5 seconds of every minute.

(b) Compute the distribution of volume executed every minute.

(c) Compute the Kolmogorov-Smirnov test on both distributions. Are they significantly different, at a 95% confidence level?

19.12 On the E-mini S&P 500 futures tick data:

(a) Compute the first-order serial correlation of signed volumes.

(b) Is it statistically significant, at a 95% confidence level?

REFERENCES

Abad, D. and J. Yague (2012): "From PIN to VPIN." *The Spanish Review of Financial Economics*, Vol. 10, No. 2, pp.74-83.

Aitken, M. and A. Frino (1996): "The accuracy of the tick test: Evidence from the Australian Stock Exchange." *Journal of Banking and Finance*, Vol. 20, pp. 1715–1729.

Amihud, Y. and H. Mendelson (1987): "Trading mechanisms and stock returns: An empirical investigation." *Journal of Finance*, Vol. 42, pp. 533–553.

Amihud, Y. (2002): "Illiquidity and stock returns: Cross-section and time-series effects." *Journal of Financial Markets*, Vol. 5, pp. 31–56.

Andersen, T. and O. Bondarenko (2013): "VPIN and the Flash Crash." *Journal of Financial Markets*, Vol. 17, pp.1-46.

Beckers, S. (1983): "Variances of security price returns based on high, low, and closing prices." *Journal of Business*, Vol. 56, pp. 97–112.

Bethel, E. W., Leinweber. D., Rubel, O., and K. Wu (2012): "Federal market information technology in the post–flash crash era: Roles for supercomputing." *Journal of Trading*, Vol. 7, No. 2, pp. 9–25.

Carlin, B., M. Sousa Lobo, and S. Viswanathan (2005): "Episodic liquidity crises. Cooperative and predatory trading." *Journal of Finance*, Vol. 42, No. 5 (October), pp. 2235–2274.

Cheung, W., R. Chou, A. Lei (2015): "Exchange-traded barrier option and VPIN." *Journal of Futures Markets*, Vol. 35, No. 6, pp. 561-581.

Corwin, S. and P. Schultz (2012): "A simple way to estimate bid-ask spreads from daily high and low prices." *Journal of Finance*, Vol. 67, No. 2, pp. 719–760.

Cremers, M. and D. Weinbaum (2010): "Deviations from put-call parity and stock return predictability." *Journal of Financial and Quantitative Analysis*, Vol. 45, No. 2 (April), pp. 335–367.

Donefer, B. (2010): "Algos gone wild. Risk in the world of automated trading strategies." *Journal of Trading*, Vol. 5, pp. 31–34.

Easley, D., N. Kiefer, M. O'Hara, and J. Paperman (1996): "Liquidity, information, and infrequently traded stocks." *Journal of Finance*, Vol. 51, No. 4, pp. 1405–1436.

Easley, D., R. Engle, M. O'Hara, and L. Wu (2008): "Time-varying arrival rates of informed and uninformed traders." *Journal of Financial Econometrics*, Vol. 6, No. 2, pp. 171–207.

Easley, D., M. López de Prado, and M. O'Hara (2011): "The microstructure of the flash crash." *Journal of Portfolio Management*, Vol. 37, No. 2 (Winter), pp. 118–128.

Easley, D., M. López de Prado, and M. O'Hara (2012a): "Flow toxicity and liquidity in a high frequency world." *Review of Financial Studies*, Vol. 25, No. 5, pp. 1457–1493.

Easley, D., M. López de Prado, and M. O'Hara (2012b): "The volume clock: Insights into the high frequency paradigm." *Journal of Portfolio Management*, Vol. 39, No. 1, pp. 19–29.

Easley, D., M. López de Prado, and M. O'Hara (2013): *High-Frequency Trading: New Realities for Traders, Markets and Regulators*, 1st ed. Risk Books.

Easley, D., M. López de Prado, and M. O'Hara (2016): "Discerning information from trade data." *Journal of Financial Economics*, Vol. 120, No. 2, pp. 269–286.

Eisler, Z., J. Bouchaud, and J. Kockelkoren (2012): "The impact of order book events: Market orders, limit orders and cancellations." *Quantitative Finance*, Vol. 12, No. 9, pp. 1395–1419.

Fabozzi, F., S. Focardi, and C. Jonas (2011): "High-frequency trading. Methodologies and market impact." *Review of Futures Markets*, Vol. 19, pp. 7–38.

Hasbrouck, J. (2007): *Empirical Market Microstructure*, 1st ed. Oxford University Press.

Hasbrouck, J. (2009): "Trading costs and returns for US equities: Estimating effective costs from daily data." *Journal of Finance*, Vol. 64, No. 3, pp. 1445–1477.

Jarrow, R. and P. Protter (2011): "A dysfunctional role of high frequency trading in electronic markets." *International Journal of Theoretical and Applied Finance*, Vol. 15, No. 3.

Kim, C., T. Perry, and M. Dhatt (2014): "Informed trading and price discovery around the clock." *Journal of Alternative Investments*, Vol 17, No. 2, pp. 68-81.

Kyle, A. (1985): "Continuous auctions and insider trading." *Econometrica*, Vol. 53, pp. 1315–1336.

Lee, C. and M. Ready (1991): "Inferring trade direction from intraday data." *Journal of Finance*, Vol. 46, pp. 733–746.

López de Prado, M. (2017): "Mathematics and economics: A reality check." *Journal of Portfolio Management*, Vol. 43, No. 1, pp. 5–8.

Muravyev, D., N. Pearson, and J. Broussard (2013): "Is there price discovery in equity options?" *Journal of Financial Economics*, Vol. 107, No. 2, pp. 259–283.

NANEX (2011): "Strange days: June 8, 2011—NatGas Algo." NANEX blog. Available at www.nanex.net/StrangeDays/06082011.html.

O'Hara, M. (1995): *Market Microstructure*, 1st ed. Blackwell, Oxford.

O'Hara, M. (2011): "What is a quote?" *Journal of Trading*, Vol. 5, No. 2 (Spring), pp. 10–15.

Parkinson, M. (1980): "The extreme value method for estimating the variance of the rate of return." *Journal of Business*, Vol. 53, pp. 61–65.

Patzelt, F. and J. Bouchaud (2017): "Universal scaling and nonlinearity of aggregate price impact in financial markets." Working paper. Available at https://arxiv.org/abs/1706.04163.

Roll, R. (1984): "A simple implicit measure of the effective bid-ask spread in an efficient market." *Journal of Finance*, Vol. 39, pp. 1127–1139.

Stigler, Stephen M. (1981): "Gauss and the invention of least squares." *Annals of Statistics*, Vol. 9, No. 3, pp. 465–474.

Song, J, K. Wu and H. Simon (2014): "Parameter analysis of the VPIN (volume synchronized probability of informed trading) metric." In Zopounidis, C., ed., *Quantitative Financial Risk Management: Theory and Practice*, 1st ed. Wiley.

Toth, B., I. Palit, F. Lillo, and J. Farmer (2011): "Why is order flow so persistent?" Working paper. Available at https://arxiv.org/abs/1108.1632.

Van Ness, B., R. Van Ness, and S. Yildiz (2017): "The role of HFTs in order flow toxicity and stock price variance, and predicting changes in HFTs' liquidity provisions." *Journal of Economics and Finance*, Vol. 41, No. 4, pp. 739–762.

Wei, W., D. Gerace, and A. Frino (2013): "Informed trading, flow toxicity and the impact on intra-day trading factors." *Australasian Accounting Business and Finance Journal*, Vol. 7, No. 2, pp. 3–24.

PART 5

High-Performance Computing Recipes

CHAPTER 20

Multiprocessing and Vectorization

20.1 MOTIVATION

Multiprocessing is essential to ML. ML algorithms are computationally intensive, and they will require an efficient use of all your CPUs, servers, and clusters. For this reason, most of the functions presented throughout this book were designed for asynchronous multiprocessing. For example, we have made frequent use of a mysterious function called `mpPandasObj`, without ever defining it. In this chapter we will explain what this function does. Furthermore, we will study in detail how to develop multiprocessing engines. The structure of the programs presented in this chapter is agnostic to the hardware architecture used to execute them, whether we employ the cores of a single server or cores distributed across multiple interconnected servers (e.g., in a high-performance computing cluster or a cloud).

20.2 VECTORIZATION EXAMPLE

Vectorization, also known as array programming, is the simplest example of parallelization, whereby an operation is applied at once to the entire set of values. As a minimal example, suppose that you need to do a brute search through a 3-dimensional space, with 2 nodes per dimension. The un-vectorized implementation of that Cartesian product will look something like Snippet 20.1. How would this code look if you had to search through 100 dimensions, or if the number of dimensions was defined by the user during runtime?

SNIPPET 20.1 UN-VECTORIZED CARTESIAN PRODUCT

```
# Cartesian product of dictionary of lists
dict0={'a':['1','2'],'b':['+','*'],'c':['!','@']}
for a in dict0['a']:
    for b in dict0['b']:
        for c in dict0['c']:
            print {'a':a,'b':b,'c':c}
```

A vectorized solution would replace all explicit iterators (e.g., `For. . .loops`) with matrix algebra operations or compiled iterators or generators. Snippet 20.2 implements the vectorized version of Snippet 20.1. The vectorized version is preferable for four reasons: (1) slow nested `For. . .loops` are replaced with fast iterators; (2) the code infers the dimensionality of the mesh from the dimensionality of `dict0`; (3) we could run 100 dimensions without having to modify the code, or need 100 `For. . .loops`; and (4) under the hood, Python can run operations in C or C++.

SNIPPET 20.2 VECTORIZED CARTESIAN PRODUCT

```
# Cartesian product of dictionary of lists
from itertoolsimport izip,product
dict0={'a':['1','2'],'b':['+','*'],'c':['!','@']}
jobs=(dict(izip(dict0,i)) for i in product(*dict0.values()))
for i in jobs:print i
```

20.3 SINGLE-THREAD VS. MULTITHREADING VS. MULTIPROCESSING

A modern computer has multiple CPU sockets. Each CPU has many cores (processors), and each core has several threads. Multithreading is the technique by which several applications are run in parallel on two or more threads under the same core. One advantage of multithreading is that, because the applications share the same core, they share the same memory space. That introduces the risk that several applications may write on the same memory space at the same time. To prevent that from happening, the Global Interpreter Lock (GIL) assigns write access to one thread per core at a time. Under the GIL, Python's multithreading is limited to one thread per processor. For this reason, Python achieves parallelism through multiprocessing rather than through actual multithreading. Processors do not share the same memory space, hence multiprocessing does not risk writing to the same memory space; however, that also makes it harder to share objects between processes.

Python functions implemented for running on a single-thread will use only a fraction of a modern computer's, server's, or cluster's power. Let us see an example of how a simple task can be run inefficiently when implemented for single-thread execution. Snippet 20.3 finds the earliest time 10,000 Gaussian processes of length 1,000 touch a symmetric double barrier of width 50 times the standard deviation.

SNIPPET 20.3 SINGLE-THREAD IMPLEMENTATION OF A ONE-TOUCH DOUBLE BARRIER

```
import numpy as np
#-------------------------------------------
def main0():
    # Path dependency: Sequential implementation
    r=np.random.normal(0,.01,size=(1000,10000))
    t=barrierTouch(r)
    return
#-------------------------------------------
def barrierTouch(r,width=.5):
    # find the index of the earliest barrier touch
    t,p={},np.log((1+r).cumprod(axis=0))
    for j in xrange(r.shape[1]): # go through columns
        for i in xrange(r.shape[0]): # go through rows
            if p[i,j]>=width or p[i,j]<=-width:
                t[j]=i
                continue
    return t
#-------------------------------------------
if __name__=='__main__':
    import timeit
    print min(timeit.Timer('main0()',setup='from __main__ import main0').repeat(5,10))
```

Compare this implementation with Snippet 20.4. Now the code splits the previous problem into 24 tasks, one per processor. The tasks are then run asynchronously in parallel, using 24 processors. If you run the same code on a cluster with 5000 CPUs, the elapsed time will be about 1/5000 of the single-thread implementation.

SNIPPET 20.4 MULTIPROCESSING IMPLEMENTATION OF A ONE-TOUCH DOUBLE BARRIER

```
import numpy as np
import multiprocessing as mp
#-------------------------------------------
def main1():
    # Path dependency: Multi-threaded implementation
    r,numThreads=np.random.normal(0,.01,size=(1000,10000)),24
    parts=np.linspace(0,r.shape[0],min(numThreads,r.shape[0])+1)
    parts,jobs=np.ceil(parts).astype(int),[]
    for i in xrange(1,len(parts)):
        jobs.append(r[:,parts[i-1]:parts[i]]) # parallel jobs
```

```
    pool,out=mp.Pool(processes=numThreads),[]
    outputs=pool.imap_unordered(barrierTouch,jobs)
    for out_ in outputs:out.append(out_) # asynchronous response
    pool.close();pool.join()
    return
#———————————————————————————————————
if __name__=='__main__':
    import timeit
    print min(timeit.Timer('main1()',setup='from __main__ import main1').repeat(5,10))
```

Moreover, you could implement the same code to multiprocess a vectorized function, as we did with function `applyPtSlOnT1` in Chapter 3, where parallel processes execute subroutines that include vectorized pandas objects. In this way, you will achieve two levels of parallelization at once. But why stop there? You could achieve three levels of parallelization at once by running multiprocessed instances of vectorized code in an HPC cluster, where each node in the cluster provides the third level of parallelization. In the next sections, we will explain how multiprocessing works.

20.4 ATOMS AND MOLECULES

When preparing jobs for parallelization, it is useful to distinguish between atoms and molecules. Atoms are indivisible tasks. Rather than carrying out all these tasks sequentially in a single thread, we want to group them into molecules, which can be processed in parallel using multiple processors. Each molecule is a subset of atoms that will be processed sequentially, by a callback function, using a single thread. Parallelization takes place at the molecular level.

20.4.1 Linear Partitions

The simplest way to form molecules is to partition a list of atoms in subsets of equal size, where the number of subsets is the minimum between the number of processors and the number of atoms. For N subsets we need to find the $N + 1$ indices that enclose the partitions. This logic is demonstrated in Snippet 20.5.

SNIPPET 20.5 THE `linParts` FUNCTION

```
import numpy as np
#———————————————————————————————————
def linParts(numAtoms,numThreads):
    # partition of atoms with a single loop
    parts=np.linspace(0,numAtoms,min(numThreads,numAtoms)+1)
    parts=np.ceil(parts).astype(int)
    return parts
```

It is common to encounter operations that involve two nested loops. For example, computing a SADF series (Chapter 17), evaluating multiple barrier touches (Chapter 3), or computing a covariance matrix on misaligned series. In these situations, a linear partition of the atomic tasks would be inefficient, because some processors would have to solve a much larger number of operations than others, and the calculation time will depend on the heaviest molecule. A partial solution is to partition the atomic tasks in a number of jobs that is a multiple of the number of processors, then front-load the jobs queue with the heavy molecules. In this way, the light molecules will be assigned to processors that have completed the heavy molecules first, keeping all CPUs busy until the job queue is depleted. In the next section, we will discuss a more complete solution. Figure 20.1 plots a linear partition of 20 atomic tasks of equal complexity into 6 molecules.

20.4.2 Two-Nested Loops Partitions

Consider two nested loops, where the outer loop iterates $i = 1, \ldots, N$ and the inner loop iterates $j = 1, \ldots, i$. We can order these atomic tasks $\{(i,j)|1 \leq j \leq i, i = 1, \ldots, N\}$ as a *lower* triangular matrix (including the main diagonal). This entails $\frac{1}{2}N(N-1) + N = \frac{1}{2}N(N+1)$ operations, where $\frac{1}{2}N(N-1)$ are off-diagonal and N are diagonal. We would like to parallelize these tasks by partitioning the atomic tasks into M subsets of rows, $\{S_m\}_{m=1,\ldots,M}$, each composed of approximately $\frac{1}{2M}N(N+1)$ tasks. The following algorithm determines the rows that constitute each subset (a molecule).

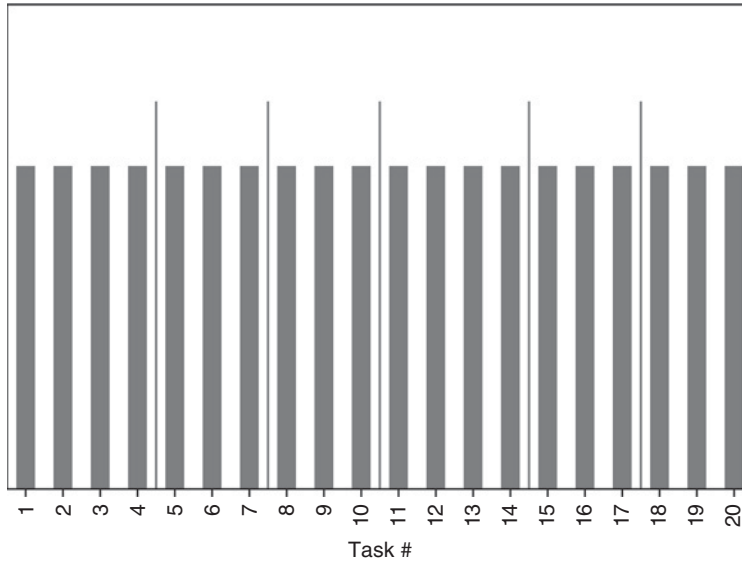


FIGURE 20.1 A linear partition of 20 atomic tasks into 6 molecules

The first subset, S_1, is composed of the first r_1 rows, that is, $S_1 = \{1, \ldots, r_1\}$, for a total number of items $\frac{1}{2}r_1(r_1+1)$. Then, r_1 must satisfy the condition $\frac{1}{2}r_1(r_1+1) = \frac{1}{2M}N(N+1)$. Solving for r_1, we obtain the positive root

$$r_1 = \frac{-1+\sqrt{1+4N(N+1)M^{-1}}}{2}$$

The second subset contains rows $S_2 = \{r_1+1, \ldots, r_2\}$, for a total number of items $\frac{1}{2}(r_2+r_1+1)(r_2-r_1)$. Then, r_2 must satisfy the condition $\frac{1}{2}(r_2+r_1+1)(r_2-r_1) = \frac{1}{2M}N(N+1)$. Solving for r_2, we obtain the positive root

$$r_2 = \frac{-1+\sqrt{1+4\left(r_1^2+r_1+N(N+1)M^{-1}\right)}}{2}$$

We can repeat the same argument for a future subset $S_m = \{r_{m-1}+1, \ldots, r_m\}$, with a total number of items $\frac{1}{2}(r_m+r_{m-1}+1)(r_m-r_{m-1})$. Then, r_m must satisfy the condition $\frac{1}{2}(r_m+r_{m-1}+1)(r_m-r_{m-1}) = \frac{1}{2M}N(N+1)$. Solving for r_m, we obtain the positive root

$$r_m = \frac{-1+\sqrt{1+4\left(r_{m-1}^2+r_{m-1}+N(N+1)M^{-1}\right)}}{2}$$

And it is easy to see that r_m reduces to r_1 where $r_{m-1} = r_0 = 0$. Because row numbers are positive integers, the above results are rounded to the nearest natural number. This may mean that some partitions' sizes may deviate slightly from the $\frac{1}{2M}N(N+1)$ target. Snippet 20.6 implements this logic.

SNIPPET 20.6 THE `nestedParts` FUNCTION

```
def nestedParts(numAtoms,numThreads,upperTriang=False):
    # partition of atoms with an inner loop
    parts,numThreads_=[0],min(numThreads,numAtoms)
    for num in xrange(numThreads_):
        part=1+4*(parts[-1]**2+parts[-1]+numAtoms*(numAtoms+1.)/numThreads_)
        part=(-1+part**.5)/2.
        parts.append(part)
    parts=np.round(parts).astype(int)
    if upperTriang: # the first rows are the heaviest
        parts=np.cumsum(np.diff(parts)[::-1])
        parts=np.append(np.array([0]),parts)
    return parts
```

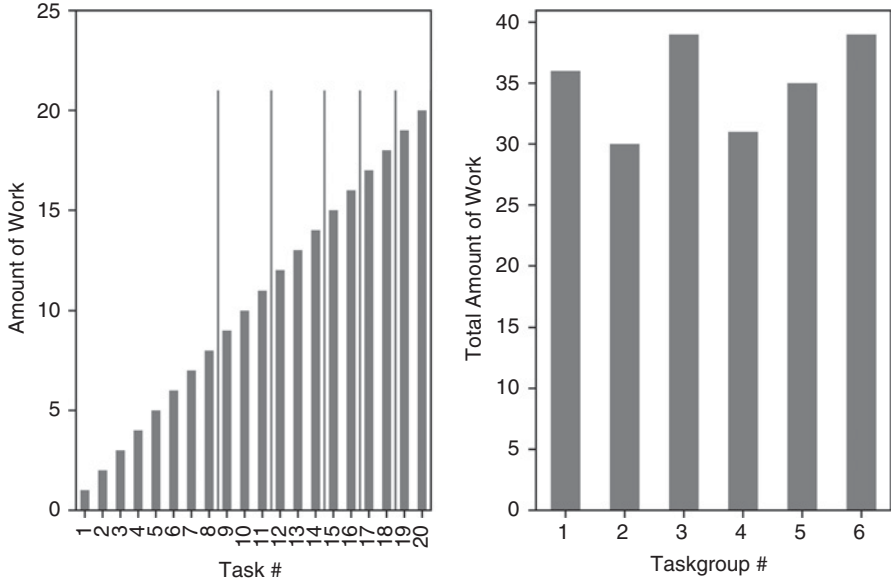


FIGURE 20.2 A two-nested loops partition of atoms into molecules

If the outer loop iterates $i = 1, \ldots, N$ and the inner loop iterates $j = i, \ldots, N$, we can order these atomic tasks $\{(i,j) \mid 1 \leq i \leq j, j = 1, \ldots, N\}$ as an *upper* triangular matrix (including the main diagonal). In this case, the argument `upperTriang=True` must be passed to function `nestedParts`. For the curious reader, this is a special case of the bin packing problem. Figure 20.2 plots a two-nested loops partition of atoms of increasing complexity into molecules. Each of the resulting 6 molecules involves a similar amount of work, even though some atomic tasks are up to 20 times harder than others.

20.5 MULTIPROCESSING ENGINES

It would be a mistake to write a parallelization wrapper for each multiprocessed function. Instead, we should develop a library that can parallelize unknown functions, regardless of their arguments and output structure. That is the goal of a multiprocessing engine. In this section, we will study one such engine, and once you understand the logic, you will be ready to develop your own, including all sorts of customized properties.

20.5.1 Preparing the Jobs

In previous chapters we have made frequent use of the `mpPandasObj`. That function receives six arguments, of which four are optional:

- `func`: A callback function, which will be executed in parallel
- `pdObj`: A tuple containing:
  - The name of the argument used to pass molecules to the callback function
  - A list of indivisible tasks (atoms), which will be grouped into molecules
- `numThreads`: The number of threads that will be used in parallel (one processor per thread)
- `mpBatches`: Number of parallel batches (jobs per core)
- `linMols`: Whether partitions will be linear or double-nested
- `kargs`: Keyword arguments needed by `func`

Snippet 20.7 lists how `mpPandasObj` works. First, atoms are grouped into molecules, using `linParts` (equal number of atoms per molecule) or `nestedParts` (atoms distributed in a lower-triangular structure). When `mpBatches` is greater than 1, there will be more molecules than cores. Suppose that we divide a task into 10 molecules, where molecule 1 takes twice as long as the rest. If we run this process in 10 cores, 9 of the cores will be idle half of the runtime, waiting for the first core to process molecule 1. Alternatively, we could set `mpBatches=10` so as to divide that task in 100 molecules. In doing so, every core will receive equal workload, even though the first 10 molecules take as much time as the next 20 molecules. In this example, the run with `mpBatches=10` will take half of the time consumed by `mpBatches=1`.

Second, we form a list of jobs. A job is a dictionary containing all the information needed to process a molecule, that is, the callback function, its keyword arguments, and the subset of atoms that form the molecule. Third, we will process the jobs sequentially if `numThreads==1` (see Snippet 20.8), and in parallel otherwise (see Section 20.5.2). The reason that we want the option to run jobs sequentially is for debugging purposes. It is not easy to catch a bug when programs are run in multiple processors.[1] Once the code is debugged, we will want to use `numThreads>1`. Fourth, we stitch together the output from every molecule into a single list, series, or dataframe.

SNIPPET 20.7 THE `mpPandasObj`, USED AT VARIOUS POINTS IN THE BOOK

```
def mpPandasObj(func,pdObj,numThreads=24,mpBatches=1,linMols=True,**kargs):
    '''
    Parallelize jobs, return a DataFrame or Series
    + func: function to be parallelized. Returns a DataFrame
    + pdObj[0]: Name of argument used to pass the molecule
    + pdObj[1]: List of atoms that will be grouped into molecules
    + kargs: any other argument needed by func
```

[1] *Heisenbugs*, named after Heisenberg's uncertainty principle, describe bugs that change their behavior when scrutinized. Multiprocessing bugs are a prime example.

```
Example: df1=mpPandasObj(func,('molecule',df0.index),24,**kargs)
'''
import pandas as pd
if linMols:parts=linParts(len(argList[1]),numThreads*mpBatches)
else:parts=nestedParts(len(argList[1]),numThreads*mpBatches)
jobs=[]    for i in xrange(1,len(parts)):
    job={pdObj[0]:pdObj[1][parts[i-1]:parts[i]],'func':func}
    job.update(kargs)
    jobs.append(job)
if numThreads==1:out=processJobs_(jobs)
else:out=processJobs(jobs,numThreads=numThreads)
if isinstance(out[0],pd.DataFrame):df0=pd.DataFrame()
elif isinstance(out[0],pd.Series):df0=pd.Series()
else:return out
for i in out:df0=df0.append(i)
df0=df0.sort_index()
return df0
```

In Section 20.5.2 we will see the multiprocessing counterpart to function `processJobs_` of Snippet 20.8.

SNIPPET 20.8 SINGLE-THREAD EXECUTION, FOR DEBUGGING

```
def processJobs_(jobs):
    # Run jobs sequentially, for debugging
    out=[]
    for job in jobs:
        out_=expandCall(job)
        out.append(out_)
    return out
```

20.5.2 Asynchronous Calls

Python has a parallelization library called `multiprocessing`. This library is the basis for multiprocessing engines such as `joblib`,[2] which is the engine used by many `sklearn` algorithms.[3] Snippet 20.9 illustrates how to do an asynchronous call to Python's `multiprocessing` library. The `reportProgress` function keeps us informed about the percentage of jobs completed.

[2] https://pypi.python.org/pypi/joblib.

[3] http://scikit-learn.org/stable/developers/performance.html#multi-core-parallelism-using-joblib-parallel.

SNIPPET 20.9 EXAMPLE OF ASYNCHRONOUS CALL TO PYTHON'S MULTIPROCESSING LIBRARY

```
import multiprocessing as mp
#———————————————————————————————
def reportProgress(jobNum,numJobs,time0,task):
    # Report progress as asynch jobs are completed
    msg=[float(jobNum)/numJobs,(time.time()-time0)/60.]
    msg.append(msg[1]*(1/msg[0]-1))
    timeStamp=str(dt.datetime.fromtimestamp(time.time()))
    msg=timeStamp+' '+str(round(msg[0]*100,2))+'% '+task+' done after '+ \
        str(round(msg[1],2))+' minutes. Remaining '+str(round(msg[2],2))+' minutes.'
    if jobNum<numJobs:sys.stderr.write(msg+'\r')
    else:sys.stderr.write(msg+'\n')
    return
#———————————————————————————————
def processJobs(jobs,task=None,numThreads=24):
    # Run in parallel.
    # jobs must contain a 'func' callback, for expandCall
    if task is None:task=jobs[0]['func'].__name__
    pool=mp.Pool(processes=numThreads)
    outputs,out,time0=pool.imap_unordered(expandCall,jobs),[],time.time()
    # Process asynchronous output, report progress
    for i,out_ in enumerate(outputs,1):
        out.append(out_)
        reportProgress(i,len(jobs),time0,task)
    pool.close();pool.join() # this is needed to prevent memory leaks
    return out
```

20.5.3 Unwrapping the Callback

In Snippet 20.9, the instruction `pool.imap_unordered()` parallelized `expandCall`, by running each item in `jobs` (a molecule) in a single thread. Snippet 20.10 lists `expandCall`, which unwraps the items (atoms) in the job (molecule), and executes the callback function. This little function is the trick at the core of the multiprocessing engine: It transforms a dictionary into a task. Once you understand the role it plays, you will be able to develop your own engines.

SNIPPET 20.10 PASSING THE JOB (MOLECULE) TO THE CALLBACK FUNCTION

```
def expandCall(kargs):
    # Expand the arguments of a callback function, kargs['func']
    func=kargs['func']
    del kargs['func']
    out=func(**kargs)
    return out
```

20.5.4 Pickle/Unpickle Objects

Multiprocessing must pickle methods in order to assign them to different processors. The problem is, bound methods are not pickable.[4] The work around is to add functionality to your engine, that tells the library how to deal with this kind of objects. Snippet 20.11 contains the instructions that should be listed at the top of your multiprocessing engine library. If you are curious about the precise reason this piece of code is needed, you may want to read Ascher et al. [2005], Section 7.5.

SNIPPET 20.11 PLACE THIS CODE AT THE BEGINNING OF YOUR ENGINE

```
def _pickle_method(method):
    func_name=method.im_func.__name__
    obj=method.im_self
    cls=method.im_class
    return _unpickle_method,(func_name,obj,cls)
#————————————————————————————
def _unpickle_method(func_name,obj,cls):
    for cls in cls.mro():
        try:func=cls.__dict__[func_name]
        except KeyError:pass
        else:break
    return func.__get__(obj,cls)
#————————————————————————————
import copy_reg,types,multiprocessing as mp
copy_reg.pickle(types.MethodType,_pickle_method,_unpickle_method)
```

20.5.5 Output Reduction

Suppose that you divide a task into 24 molecules, with the goal that the engine assigns each molecule to one available core. Function `processJobs` in Snippet 20.9 will capture the 24 outputs and store them in a list. This approach is effective in problems that do not involve large outputs. If the outputs must be combined into a single output, first we will wait until the last molecule is completed, and then we will process the items in the list. The latency added by this post-processing should not be significant, as long as the outputs are small in size and number.

However, when the outputs consume a lot of RAM, and they need to be combined into a single output, storing all those outputs in a list may cause a memory error. It would be better to perform the output reduction operation on the fly, as the results are returned asynchronously by `func`, rather than waiting for the last molecule to be completed. We can address this concern by improving processJobs. In particular,

[4] http://stackoverflow.com/questions/1816958/cant-pickle-type-instancemethod-when-using-pythons-multiprocessing-pool-ma.

we are going to pass three additional arguments that determine how the molecular outputs must be *reduced* into a single output. Snippet 20.12 lists an enhanced version of `processJobs`, which contains three new arguments:

- `redux`: This is a callback to the function that carries out the reduction. For example, `redux=pd.DataFrame.add`, if output dataframes ought to be summed up.
- `reduxArgs`: This is a dictionary that contains the keyword arguments that must be passed to `redux` (if any). For example, if `redux=pd.DataFrame.join`, then a possibility is `reduxArgs={'how':'outer'}`.
- `reduxInPlace`: A boolean, indicating whether the `redux` operation should happen *in-place* or not. For example, `redux=dict.update` and `redux=list.append` require `reduxInPlace=True`, since appending a list and updating a dictionary are both in-place operations.

SNIPPET 20.12 ENHANCING `processJobs` TO PERFORM ON-THE-FLY OUTPUT REDUCTION

```
def processJobsRedux(jobs,task=None,numThreads=24,redux=None,reduxArgs={},
                     reduxInPlace=False):
    '''
    Run in parallel
    jobs must contain a 'func' callback, for expandCall
    redux prevents wasting memory by reducing output on the fly
    '''
    if task is None:task=jobs[0]['func'].__name__
    pool=mp.Pool(processes=numThreads)
    imap,out,time0=pool.imap_unordered(expandCall,jobs),None,time.time()
    # Process asynchronous output, report progress
    for i,out_ in enumerate(imap,1):
        if out is None:
            if redux is None:out,redux,reduxInPlace=[out_],list.append,True
            else:out=copy.deepcopy(out_)
        else:
            if reduxInPlace:redux(out,out_,**reduxArgs)
            else:out=redux(out,out_,**reduxArgs)
        reportProgress(i,len(jobs),time0,task)
    pool.close();pool.join() # this is needed to prevent memory leaks
    if isinstance(out,(pd.Series,pd.DataFrame)):out=out.sort_index()
    return out
```

Now that `processJobsRedux` knows what to do with the outputs, we can also enhance `mpPandasObj` from Snippet 20.7. In Snippet 20.13, the new function `mpJobList` passes the three output reduction arguments to `processJobsRedux`.

This eliminates the need to process an outputed list, as `mpPandasObj` did, hence saving memory and time.

SNIPPET 20.13 ENHANCING `mpPandasObj` TO PERFORM ON-THE-FLY OUTPUT REDUCTION

```
def mpJobList(func,argList,numThreads=24,mpBatches=1,linMols=True,redux=None,
              reduxArgs={},reduxInPlace=False,**kargs):
    if linMols:parts=linParts(len(argList[1]),numThreads*mpBatches)
    else:parts=nestedParts(len(argList[1]),numThreads*mpBatches)
    jobs=[]
    for i in xrange(1,len(parts)):
        job={argList[0]:argList[1][parts[i-1]:parts[i]],'func':func}
        job.update(kargs)
        jobs.append(job)
    out=processJobsRedux(jobs,redux=redux,reduxArgs=reduxArgs,
                         reduxInPlace=reduxInPlace,numThreads=numThreads)
    return out
```

20.6 MULTIPROCESSING EXAMPLE

What we have presented so far in this chapter can be used to speed-up, by several orders of magnitude, many lengthy and large-scale mathematical operations. In this section we will illustrate an additional motivation for multiprocessing: memory management.

Suppose that you have conducted a spectral decomposition of a covariance matrix of the form $Z'Z$, as we did in Chapter 8, Section 8.4.2, where Z has size TxN. This has resulted in an eigenvectors matrix W and an eigenvalues matrix Λ, such that $Z'ZW = W\Lambda$. Now you would like to derive the orthogonal principal components that explain a user-defined portion of the total variance, $0 \leq \tau \leq 1$. In order to do that, we compute $P = Z\tilde{W}$, where $\tilde{W}$ contains the first $M \leq N$ columns of W, such that $(\sum_{m=1}^{M} \Lambda_{m,m})(\sum_{n=1}^{N} \Lambda_{n,n})^{-1} \geq \tau$. The computation of $P = Z\tilde{W}$ can be parallelized by noting that

$$P = Z\tilde{W} = \sum_{b=1}^{B} Z_b \tilde{W}_b$$

where Z_b is a sparse TxN matrix with only TxN_b items (the rest are empty), $\tilde{W}_b$ is a NxM matrix with only $N_b xM$ items (the rest are empty), and $\sum_{b=1}^{B} N_b = N$. This sparsity is created by dividing the set of columns into a partition of B subsets of columns, and loading into Z_b only the bth subset of the columns. This notion of sparsity may sound a bit complicated at first, however Snippet 20.14 demonstrates how pandas

allows us to implement it in a seamless way. Function `getPCs` receives $\tilde{W}$ through the argument `eVec`. The argument `molecules` contains a subset of the file names in `fileNames`, where each file represents Z_b. The key concept to grasp is that we compute the dot product of a Z_b with the slice of the rows of $\tilde{W}_b$ defined by the columns in Z_b, and that molecular results are aggregated on the fly (`redux=pd.DataFrame.add`).

SNIPPET 20.14 PRINCIPAL COMPONENTS FOR A SUBSET OF THE COLUMNS

```
pcs=mpJobList(getPCs,('molecules',fileNames),numThreads=24,mpBatches=1,
    path=path,eVec=eVec,redux=pd.DataFrame.add)
#---------------------------------------
def getPCs(path,molecules,eVec):
    # get principal components by loading one file at a time
    pcs=None
    for i in molecules:
        df0=pd.read_csv(path+i,index_col=0,parse_dates=True)
        if pcs is None:pcs=np.dot(df0.values,eVec.loc[df0.columns].values)
        else:pcs+=np.dot(df0.values,eVec.loc[df0.columns].values)
    pcs=pd.DataFrame(pcs,index=df0.index,columns=eVec.columns)
    return pcs
```

This approach presents two advantages: First, because `getPCs` loads dataframes Z_b sequentially, for a sufficiently large B, the RAM is not exhausted. Second, `mpJobList` executes the molecules in parallel, hence speeding up the calculations.

In real life ML applications, we often encounter datasets where Z contains billions of datapoints. As this example demonstrates, parallelization is not only beneficial in terms of reducing run time. Many problems could not be solved without parallelization, as a matter of memory limitations, even if we were willing to wait longer.

EXERCISES

20.1 Run Snippets 20.1 and 20.2 with `timeit`. Repeat 10 batches of 100 executions. What is the minimum elapsed time for each snippet?

20.2 The instructions in Snippet 20.2 are very useful for unit testing, brute force searches, and scenario analysis. Can you remember where else in the book have you seen them? Where else could they have been used?

20.3 Adjust Snippet 20.4 to form molecules using a two-nested loops scheme, rather than a linear scheme.

20.4 Compare with `timeit`:

(a) Snippet 20.4, by repeating 10 batches of 100 executions. What is the minimum elapsed time for each snippet?

(b) Modify Snippet 20.4 (from exercise 3), by repeating 10 batches of 100 executions. What is the minimum elapsed time for each snippet?

20.5 Simplify Snippet 20.4 by using `mpPandasObj`.

20.6 Modify `mpPandasObj` to handle the possibility of forming molecules using a two-nested loops scheme with an upper triangular structure.

REFERENCE

Ascher, D., A. Ravenscroft, and A. Martelli (2005): *Python Cookbook*, 2nd ed. O'Reilly Media.

BIBLIOGRAPHY

Gorelick, M. and I. Ozsvald (2008): *High Performance Python*, 1st ed. O'Reilly Media.

López de Prado, M. (2017): "Supercomputing for finance: A gentle introduction." Lecture materials, Cornell University. Available at https://ssrn.com/abstract=2907803.

McKinney, W. (2012): *Python for Data Analysis*, 1st ed. O'Reilly Media.

Palach, J. (2008): *Parallel Programming with Python*, 1st ed. Packt Publishing.

Summerfield, M. (2013): *Python in Practice: Create Better Programs Using Concurrency, Libraries, and Patterns*, 1st ed. Addison-Wesley.

Zaccone, G. (2015): *Python Parallel Programming Cookbook*, 1st ed. Packt Publishing.

CHAPTER 21

Brute Force and Quantum Computers

21.1 MOTIVATION

Discrete mathematics appears naturally in multiple ML problems, including hierarchical clustering, grid searches, decisions based on thresholds, and integer optimization. Sometimes, these problems do not have a known analytical (closed-form) solution, or even a heuristic to approximate it, and our only hope is to search for it through brute force. In this chapter, we will study how a financial problem, intractable to modern supercomputers, can be reformulated as an integer optimization problem. Such a representation makes it amenable to quantum computers. From this example the reader can infer how to translate his particular financial ML intractable problem into a quantum brute force search.

21.2 COMBINATORIAL OPTIMIZATION

Combinatorial optimization problems can be described as problems where there is a finite number of feasible solutions, which result from combining the discrete values of a finite number of variables. As the number of feasible combinations grows, an exhaustive search becomes impractical. The traveling salesman problem is an example of a combinatorial optimization problem that is known to be NP hard (Woeginger [2003]), that is, the category of problems that are at least as hard as the hardest problems solvable is nondeterministic polynomial time.

What makes an exhaustive search impractical is that standard computers evaluate and store the feasible solutions sequentially. But what if we could evaluate and store all feasible solutions at once? That is the goal of quantum computers. Whereas the bits of a standard computer can only adopt one of two possible states ($\{0, 1\}$) at once, quantum computers rely on qubits, which are memory elements that may hold a *linear superposition* of both states. In theory, quantum computers can accomplish this thanks

to quantum mechanical phenomena. In some implementations, qubits can support currents flowing in two directions at once, hence providing the desired superposition. This linear superposition property is what makes quantum computers ideally suited for solving NP-hard combinatorial optimization problems. See Williams [2010] for a general treatise on the capabilities of quantum computers.

The best way to understand this approach is through a particular example. We will now see how a dynamic portfolio optimization problem subject to generic transaction cost functions can be represented as a combinatorial optimization problem, tractable to quantum computers. Unlike Garleanu and Pedersen [2012], we will not assume that the returns are drawn from an IID Gaussian distribution. This problem is particularly relevant to large asset managers, as the costs from excessive turnover and implementation shortfall may critically erode the profitability of their investment strategies.

21.3 THE OBJECTIVE FUNCTION

Consider a set on assets $X = \{x_i\}$, $i = 1, \ldots, N$, with returns following a multivariate Normal distribution at each time horizon $h = 1, \ldots, H$, with varying mean and variance. We will assume that the returns are multivariate Normal, time-independent, however not identically distributed through time. We define a trading trajectory as an *NxH* matrix ω that determines the proportion of capital allocated to each of the N assets over each of the H horizons. At a particular horizon $h = 1, \ldots, H$, we have a forecasted mean μ_h, a forecasted variance V_h and a forecasted transaction cost function $\tau_h[\omega]$. This means that, given a trading trajectory ω, we can compute a vector of expected investment returns r, as

$$r = \text{diag}[\mu'\omega] - \tau[\omega]$$

where $\tau[\omega]$ can adopt any functional form. Without loss of generality, consider the following:

- $\tau_1[\omega] = \sum_{n=1}^{N} c_{n,1} \sqrt{|\omega_{n,1} - \omega_n^*|}$
- $\tau_h[\omega] = \sum_{n=1}^{N} c_{n,h} \sqrt{|\omega_{n,h} - \omega_{n,h-1}|}$, for $h = 2, \ldots, H$
- ω_n^* is the initial allocation to instrument n, $n = 1, \ldots, N$

$\tau[\omega]$ is an *Hx1* vector of transaction costs. In words, the transaction costs associated with each asset are the sum of the square roots of the changes in capital allocations, re-scaled by an asset-specific factor $C_h = \{c_{n,h}\}_{n=1,\ldots,N}$ that changes with h. Thus, C_h is an *Nx1* vector that determines the relative transaction cost across assets.

The Sharpe Ratio (Chapter 14) associated with r can be computed as (μ_h being net of the risk-free rate)

$$SR[r] = \frac{\sum_{h=1}^{H} \mu_h^{'} \omega_h - \tau_h[\omega]}{\sqrt{\sum_{h=1}^{H} \omega_h^{'} V_h \omega_h}}$$

21.4 THE PROBLEM

We would like to compute the optimal trading trajectory that solves the problem

$$\max_{\omega} \; SR[r]$$
$$\text{s.t.} : \sum_{i=1}^{N} |\omega_{i,h}| = 1, \; \forall h = 1, \ldots, H$$

This problem attempts to compute a global dynamic optimum, in contrast to the static optimum derived by mean-variance optimizers (see Chapter 16). Note that non-continuous transaction costs are embedded in r. Compared to standard portfolio optimization applications, this is not a convex (quadratic) programming problem for at least three reasons: (1) Returns are not identically distributed, because μ_h and V_h change with h. (2) Transaction costs $\tau_h[\omega]$ are non-continuous and changing with h. (3) The objective function $SR[r]$ is not convex. Next, we will show how to calculate solutions without making use of any analytical property of the objective function (hence the generalized nature of this approach).

21.5 AN INTEGER OPTIMIZATION APPROACH

The generality of this problem makes it intractable to standard convex optimization techniques. Our solution strategy is to discretize it so that it becomes amenable to integer optimization. This in turn allows us to use quantum computing technology to find the optimal solution.

21.5.1 Pigeonhole Partitions

Suppose that we count the number of ways that K units of capital can be allocated among N assets, where we assume $K > N$. This is equivalent to finding the number of non-negative integer solutions to $x_1 + \ldots + x_N = K$, which has the nice combinatorial solution $\binom{K+N-1}{N-1}$. This bears a similarity to the classic integer partitioning problem in number theory for which Hardy and Ramanujan (and later, Rademacher) proved an asymptotic expression (see Johansson [2012]). While order does not matter in the partition problem, order is very relevant to the problem we have at hand.

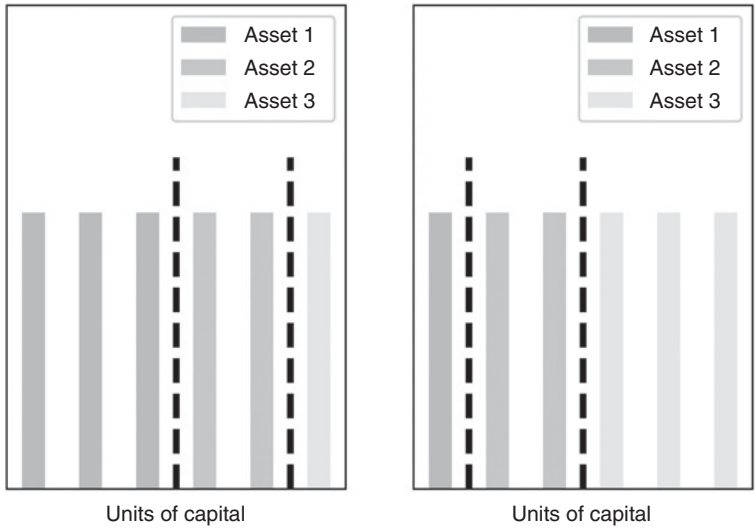


FIGURE 21.1 Partitions (1, 2, 3) and (3, 2, 1) must be treated as different

For example, if $K = 6$ and $N = 3$, partitions $(1, 2, 3)$ and $(3, 2, 1)$ must be treated as different (obviously $(2, 2, 2)$ does not need to be permutated). Figure 21.1 illustrates how order is important when allocating 6 units of capital to 3 different assets. This means that we must consider all distinct permutations of each partition. Even though there is a nice combinatorial solution to find the number of such allocations, it may still be computationally intensive to find as K and N grow large. However, we can use Stirling's approximation to easily arrive at an estimate.

Snippet 21.1 provides an efficient algorithm to generate the set of all partitions, $p^{K,N} = \left\{ \{p_i\}_{i=1,\ldots,N} | p_i \in \mathbb{W}, \sum_{i=1}^{N} p_i = K \right\}$, where $\mathbb{W}$ are the natural numbers including zero (whole numbers).

SNIPPET 21.1 PARTITIONS OF *k* OBJECTS INTO *n* SLOTS

```
from itertools import combinations_with_replacement
#------------------------------------------------
def pigeonHole(k,n):
    # Pigeonhole problem (organize k objects in n slots)
    for j in combinations_with_replacement(xrange(n),k):
        r=[0]*n
        for i in j:
            r[i]+=1
        yield r
```

21.5.2 Feasible Static Solutions

We would like to compute the set of all feasible solutions at any given horizon h, which we denote Ω. Consider a partition set of K units into N assets, $p^{K,N}$. For each partition $\{p_i\}_{i=1,\ldots,N} \in p^{K,N}$, we can define a vector of absolute weights such that $|\omega_i| = \frac{1}{K} p_i$, where $\sum_{i=1}^{N} |\omega_i| = 1$ (the full-investment constraint). This full-investment (without leverage) constraint implies that every weight can be either positive or negative, so for every vector of absolute weights $\{|\omega_i|\}_{i=1,\ldots,N}$ we can generate 2^N vectors of (signed) weights. This is accomplished by multiplying the items in $\{|\omega_i|\}_{i=1,\ldots,N}$ with the items of the Cartesian product of $\{-1, 1\}$ with N repetitions. Snippet 21.2 shows how to generate the set Ω of all vectors of weights associated with all partitions, $\Omega = \left\{ \left\{ \frac{s_j}{K} p_i \right\} \middle| \{s_j\}_{j=1,\ldots,N} \in \underbrace{\{-1, 1\} \text{x} \ldots \text{x} \{-1, 1\}}_{N}, \{p_i\}_{i=1,\ldots,N} \in p^{K,N} \right\}$.

SNIPPET 21.2 SET Ω OF ALL VECTORS ASSOCIATED WITH ALL PARTITIONS

```
import numpy as np
from itertools import product
#———————————————————————————————
def getAllWeights(k,n):
    #1) Generate partitions
    parts,w=pigeonHole(k,n),None
    #2) Go through partitions
    for part_ in parts:
        w_=np.array(part_)/float(k) # abs(weight) vector
        for prod_ in product([-1,1],repeat=n): # add sign
            w_signed_=(w_*prod_).reshape(-1,1)
            if w is None:w=w_signed_.copy()
            else:w=np.append(w,w_signed_,axis=1)
    return w
```

21.5.3 Evaluating Trajectories

Given the set of all vectors Ω, we define the set of all possible trajectories Φ as the Cartesian product of Ω with H repetitions. Then, for every trajectory we can evaluate its transaction costs and SR, and select the trajectory with optimal performance across Φ. Snippet 21.3 implements this functionality. The object `params` is a list of dictionaries that contain the values of C, μ, V.

SNIPPET 21.3 EVALUATING ALL TRAJECTORIES

```
import numpy as np
from itertools import product
#———————————————————————————————————
def evalTCosts(w,params):
    # Compute t-costs of a particular trajectory
    tcost=np.zeros(w.shape[1])
    w_=np.zeros(shape=w.shape[0])
    for i in range(tcost.shape[0]):
        c_=params[i]['c']
        tcost[i]=(c_*abs(w[:,i]-w_)**.5).sum()
        w_=w[:,i].copy()
    return tcost
#———————————————————————————————————
def evalSR(params,w,tcost):
    # Evaluate SR over multiple horizons
    mean,cov=0,0
    for h in range(w.shape[1]):
        params_=params[h]
        mean+=np.dot(w[:,h].T,params_['mean'])[0]-tcost[h]
        cov+=np.dot(w[:,h].T,np.dot(params_['cov'],w[:,h]))
    sr=mean/cov**.5
    return sr
#———————————————————————————————————
def dynOptPort(params,k=None):
    # Dynamic optimal portfolio
    #1) Generate partitions
    if k is None:k=params[0]['mean'].shape[0]
    n=params[0]['mean'].shape[0]
    w_all,sr=getAllWeights(k,n),None
    #2) Generate trajectories as cartesian products
    for prod_ in product(w_all.T,repeat=len(params)):
        w_=np.array(prod_).T # concatenate product into a trajectory
        tcost_=evalTCosts(w_,params)
        sr_=evalSR(params,w_,tcost_) # evaluate trajectory
        if sr is None or sr<sr_: # store trajectory if better
            sr,w=sr_,w_.copy()
    return w
```

Note that this procedure selects an globally optimal trajectory without relying on convex optimization. A solution will be found even if the covariance matrices are ill-conditioned, transaction cost functions are non-continuous, etc. The price we pay for this generality is that calculating the solution is extremely computationally intensive. Indeed, evaluating all trajectories is similar to the traveling-salesman problem.

Digital computers are inadequate for this sort of NP-complete or NP-hard problems; however, quantum computers have the advantage of evaluating multiple solutions at once, thanks to the property of linear superposition.

The approach presented in this chapter set the foundation for Rosenberg et al. [2016], which solved the optimal trading trajectory problem using a quantum annealer. The same logic can be applied to a wide range on financial problems involving path dependency, such as a trading trajectory. Intractable ML algorithm can be discretized and translated into a brute force search, intended for a quantum computer.

21.6 A NUMERICAL EXAMPLE

Below we illustrate how the global optimum can be found in practice, using a digital computer. A quantum computer would evaluate all trajectories at once, whereas the digital computer does this sequentially.

21.6.1 Random Matrices

Snippet 21.4 returns a random matrix of Gaussian values with known rank, which is useful in many applications (see exercises). You may want to consider this code the next time you want to execute multivariate Monte Carlo experiments, or scenario analyses.

SNIPPET 21.4 PRODUCE A RANDOM MATRIX OF A GIVEN RANK

```
import numpy as np
#---------------------------------------
def rndMatWithRank(nSamples,nCols,rank,sigma=0,homNoise=True):
    # Produce random matrix X with given rank
    rng=np.random.RandomState()
    U,_,_=np.linalg.svd(rng.randn(nCols,nCols))
    x=np.dot(rng.randn(nSamples,rank),U[:,:rank].T)
    if homNoise:
        x+=sigma*rng.randn(nSamples,nCols) # Adding homoscedastic noise
    else:
        sigmas=sigma*(rng.rand(nCols)+.5) # Adding heteroscedastic noise
        x+=rng.randn(nSamples,nCols)*sigmas
    return x
```

Snippet 21.5 generates H vectors of means, covariance matrices, and transaction cost factors, C, μ, V. These variables are stored in a `params` list.

SNIPPET 21.5 GENERATE THE PROBLEM'S PARAMETERS

```
import numpy as np
#-------------------------------------------
def genMean(size):
    # Generate a random vector of means
    rMean=np.random.normal(size=(size,1))
    return rMean
#-------------------------------------------
    #1) Parameters
    size,horizon=3,2
    params=[]
    for h in range(horizon):
        x=rndMatWithRank(1000,3,3,0.)
        mean_,cov_=genMean(size),np.cov(x,rowvar=False)
        c_=np.random.uniform(size=cov_.shape[0])*np.diag(cov_)**.5
        params.append({'mean':mean_,'cov':cov_,'c':c_})
```

21.6.2 Static Solution

Snippet 21.6 computes the performance of the trajectory that results from local (static) optima.

SNIPPET 21.6 COMPUTE AND EVALUATE THE STATIC SOLUTION

```
import numpy as np
#-------------------------------------------
def statOptPortf(cov,a):
    # Static optimal porftolio
    # Solution to the "unconstrained" portfolio optimization problem
    cov_inv=np.linalg.inv(cov)
    w=np.dot(cov_inv,a)
    w/=np.dot(np.dot(a.T,cov_inv),a) # np.dot(w.T,a)==1
    w/=abs(w).sum() # re-scale for full investment
    return w
#-------------------------------------------
#2) Static optimal portfolios
w_stat=None
for params_ in params:
    w_=statOptPortf(cov=params_['cov'],a=params_['mean'])
    if w_stat is None:w_stat=w_.copy()
    else:w_stat=np.append(w_stat,w_,axis=1)
tcost_stat=evalTCosts(w_stat,params)
sr_stat=evalSR(params,w_stat,tcost_stat)
print 'static SR:',sr_stat
```

21.6.3 Dynamic Solution

Snippet 21.7 computes the performance associated with the globally dynamic optimal trajectory, applying the functions explained throughout the chapter.

SNIPPET 21.7 COMPUTE AND EVALUATE THE DYNAMIC SOLUTION

```
import numpy as np
#-----------------------------------------------
#3) Dynamic optimal portfolios
w_dyn=dynOptPort(params)
tcost_dyn=evalTCosts(w_dyn,params)
sr_dyn=evalSR(params,w_dyn,tcost_dyn)
print 'dynamic SR:',sr_dyn
```

EXERCISES

21.1 Using the pigeonhole argument, prove that $\sum_{n=1}^{N} \binom{N}{n} = 2^N - 1$.

21.2 Use Snippet 21.4 to produce random matrices of size (1000, 10), `sigma=1` and

- **(a)** `rank=1`. Plot the eigenvalues of the covariance matrix.
- **(b)** `rank=5`. Plot the eigenvalues of the covariance matrix.
- **(c)** `rank=10`. Plot the eigenvalues of the covariance matrix.
- **(d)** What pattern do you observe? How would you connect it to Markowitz's curse (Chapter 16)?

21.3 Run the numerical example in Section 21.6:

- **(a)** Use `size=3`, and compute the running time with `timeit`. Repeat 10 batches of 100 executions. How long did it take?
- **(b)** Use `size=4`, and `timeit`. Repeat 10 batches of 100 executions. How long did it take?

21.4 Review all snippets in this chapter.

- **(a)** How many could be vectorized?
- **(b)** How many could be parallelized, using the techniques from Chapter 20?
- **(c)** If you optimize the code, by how much do you think you could speed it up?
- **(d)** Using the optimized code, what is the problem dimensionality that could be solved within a year?

21.5 Under what circumstances would the globally dynamic optimal trajectory match the sequence of local optima?

- **(a)** Is that a realistic set of assumptions?
- **(b)** If not,

(i) could that explain why naïve solutions beat Markowitz's (Chapter 16)?

(ii) why do you think so many firms spend so much effort in computing sequences of local optima?

REFERENCES

Garleanu, N. and L. Pedersen (2012): "Dynamic trading with predictable returns and transaction costs." *Journal of Finance*, Vol. 68, No. 6, pp. 2309–2340.

Johansson, F. (2012): "Efficient implementation of the Hardy-Ramanujan-Rademacher formula," *LMS Journal of Computation and Mathematics*, Vol. 15, pp. 341–359.

Rosenberg, G., P. Haghnegahdar, P. Goddard, P. Carr, K. Wu, and M. López de Prado (2016): "Solving the optimal trading trajectory problem using a quantum annealer." *IEEE Journal of Selected Topics in Signal Processing*, Vol. 10, No. 6 (September), pp. 1053–1060.

Williams, C. (2010): *Explorations in Quantum Computing*, 2nd ed. Springer.

Woeginger, G. (2003): "Exact algorithms for NP-hard problems: A survey." In Junger, M., G. Reinelt, and G. Rinaldi: *Combinatorial Optimization—Eureka, You Shrink!* Lecture notes in computer science, Vol. 2570, Springer, pp. 185–207.

CHAPTER 22

High-Performance Computational Intelligence and Forecasting Technologies

Kesheng Wu and Horst D. Simon

22.1 MOTIVATION

This chapter provides an introduction to the Computational Intelligence and Forecasting Technologies (CIFT) project at Lawrence Berkeley National Laboratory (LBNL). The main objective of CIFT is to promote the use of high-performance computing (HPC) tools and techniques for analysis of streaming data. After noticing the data volume being given as the explanation for the five-month delay for SEC and CFTC to issue their report on the 2010 Flash Crash, LBNL started the CIFT project to apply HPC technologies to manage and analyze financial data. Making timely decisions with streaming data is a requirement for many business applications, such as avoiding impending failure in the electric power grid or a liquidity crisis in financial markets. In all these cases, the HPC tools are well suited in handling the complex data dependencies and providing a timely solution. Over the years, CIFT has worked on a number of different forms of streaming data, including those from vehicle traffic, electric power grid, electricity usage, and so on. The following sections explain the key features of HPC systems, introduce a few special tools used on these systems, and provide examples of streaming data analyses using these HPC tools.

22.2 REGULATORY RESPONSE TO THE FLASH CRASH OF 2010

On May 6, 2010, at about 2:45 p.m. (U.S. Eastern Daylight Time), the U.S. stock market experienced a nearly 10% drop in the Dow Jones Industrial Average, only to recover most of the loss a few minutes later. It took about five months for regulatory

agencies to come up with an investigation report. In front of a congressional panel investigating the crash, the data volume (~20 terabytes) was given as the primary reason for the long delay. Since HPC systems, such as those at National Energy Research Scientific Computing (NERSC) center,[1] routinely work with hundreds of terabytes in minutes, we should have no problem processing the data from financial markets. This led to the establishment of the CIFT project with the mission to apply the HPC techniques and tools for financial data analysis.

A key aspect of financial big data is that it consists of mostly time series. Over the years, the CIFT team, along with numerous collaborators, has developed techniques to analyze many different forms of data streams and time series. This chapter provides a brief introduction to the HPC system including both hardware (Section 22.4) and software (Section 22.5), and recounts a few successful use cases (Section 22.6). We conclude with a summary of our vision and work so far and also provide contact information for interested readers.

22.3 BACKGROUND

Advances in computing technology have made it considerably easier to look for complex patterns. This pattern-finding capability is behind a number of recent scientific breakthroughs, such as the discovery of the Higgs particle (Aad et al. [2016]) and gravitational waves (Abbot et al. [2016]). This same capability is also at the core of many internet companies, for example, to match users with advertisers (Zeff and Aronson [1999], Yen et al. [2009]). However, the hardware and software used in science and in commerce are quite different. The HPC tools have some critical advantages that should be useful in a variety of business applications.

Tools for scientists are typically built around high-performance computing (HPC) platforms, while the tools for commercial applications are built around cloud computing platforms. For the purpose of sifting through large volumes of data to find useful patterns, the two approaches have been shown to work well. However, the marquee application for HPC systems is large-scale simulation, such as weather models used for forecasting regional storms in the next few days (Asanovic et al. [2006]). In contrast, the commercial cloud was initially motivated by the need to process a large number of independent data objects concurrently (data parallel tasks).

For our work, we are primarily interested in analyses of streaming data. In particular, high-speed complex data streams, such as those from sensor networks monitoring our nation's electric power grid and highway systems. This streaming workload is not ideal for either HPC systems or cloud systems as we discuss below, but we believe that the HPC ecosystem has more to offer to address the streaming data analysis than the cloud ecosystem does.

Cloud systems were originally designed for parallel data tasks, where a large number of independent data objects can be processed concurrently. The system is thus

[1] NERSC is a National User Facility funded by U.S. Department of Energy, located at LBNL. More information about NERSC can be found at http://nersc.gov/.

designed for high throughput, not for producing real-time responses. However, many business applications require real-time or near-real-time responses. For example, an instability event in an electric power grid could develop and grow into a disaster in minutes; finding the tell-tale signature quickly enough would avert the disaster. Similarly, signs of emerging illiquidity events have been identified in the financial research literature; quickly finding these signs during the active market trading hours could offer options to prevent shocks to the market and avoid flash crashes. The ability to prioritize quick turnaround time is essential in these cases.

A data stream is by definition available progressively; therefore, there may not be a large number of data objects to be processed in parallel. Typically, only a fixed amount of the most recent data records are available for analysis. In this case, an effective way to harness the computing power of many central processing units (CPUs) cores is to divide the analytical work on a single data object (or a single time-step) to many CPU cores. The HPC ecosystem has more advanced tools for this kind of work than the cloud ecosystem does.

These are the main points that motivated our work. For a more thorough comparison of HPC systems and cloud systems, we refer interested readers to Asanovic et al. [2006]. In particular, Fox et al. [2015] have created an extensive taxonomy for describing the similarities and differences for any application scenario.

In short, we believe the HPC community has a lot to offer to advance the state-of-the-art for streaming analytics. The CIFT project was established with a mission to transfer LBNL's HPC expertise to streaming business applications. We are pursuing this mission via collaboration, demonstration, and tool development.

To evaluate the potential uses of HPC technology, we have spent time working with various applications. This process not only exposes our HPC experts to a variety of fields, but also makes it possible for us to gather financial support to establish a demonstration facility.

With the generous gifts from a number of early supporters of this effort, we established a substantial computing cluster dedicated to this work. This dedicated computer (named dirac1) allows users to utilize an HPC system and evaluate their applications for themselves.

We are also engaged in a tool development effort to make HPC systems more usable for streaming data analysis. In the following sections, we will describe the hardware and software of the dedicated CIFT machine, as well as some of the demonstration and tool development efforts. Highlights include improving the data handling speed by 21-fold, and increasing the speed of computing an early warning indicator by 720-fold.

22.4 HPC HARDWARE

Legend has it that the first generation of big data systems was built with the spare computer components gleaned from a university campus. This is likely an urban legend, but it underscores an important point about the difference between HPC systems and cloud systems. Theoretically, a HPC system is built with custom

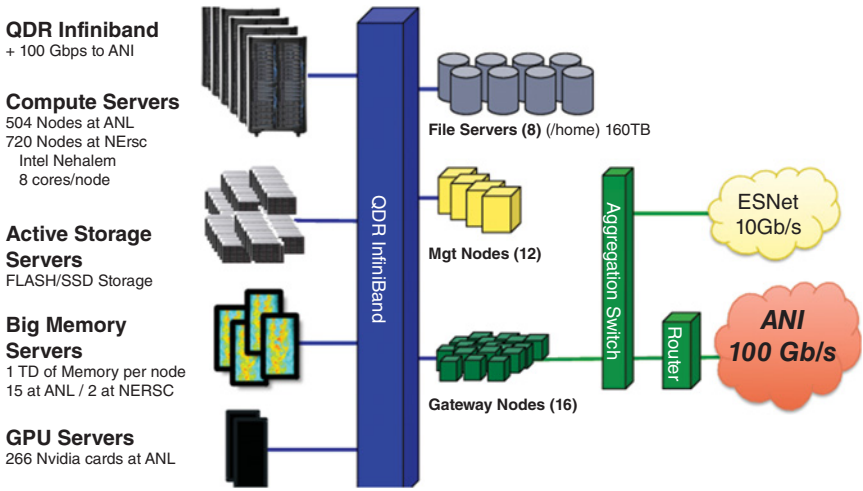


FIGURE 22.1 Schematic of the Magellan cluster (circa 2010), an example of HPC computer cluster

high-cost components, while cloud systems are built with standard low-cost commodity components. In practice, since the worldwide investment in HPC systems is much smaller than that of personal computers, there is no way for manufacturers to produce custom components just for the HPC market. The truth is that HPC systems are largely assembled from commodity components just like cloud systems. However, due to their different target applications, there are some differences in their choices of the components.

Let us describe the computing elements, storage system, and networking system in turn. Figure 22.1 is a high-level schematic diagram representing the key components of the Magellan cluster around year 2010 (Jackson et al. [2010]; Yelick et al. [2011]). The computer elements include both CPUs and graphics processing units (GPUs). These CPUs and GPUs are commercial products in almost all the cases. For example, the nodes on dirac1 use a 24-core 2.2Ghz Intel processor, which is common to cloud computing systems. Currently, dirac1 does not contain GPUs.

The networking system consists of two parts: the InfiniBand network connecting the components within the cluster, and the switched network connection to the outside world. In this particular example, the outside connections are labeled "ESNet" and "ANI." The InfiniBand network switches are also common in cloud computing systems.

The storage system in Figure 1 includes both rotating disks and flash storage. This combination is also common. What is different is that a HPC system typically has its storage system concentrated outside of the computer nodes, while a typical cloud computing system has its storage system distributed among the compute nodes. These two approaches have their own advantages and disadvantages. For example, the concentrated storage is typically exported as a global file system to all computer nodes, which makes it easier to deal with data stored in files. However, this requires a highly capable network connecting the CPUs and the disks. In contrast,

the distributed approach could use lower-capacity network because there is some storage that is close to each CPU. Typically, a distributed file system, such as the Google file system (Ghemawat, Gobioff, and Leung [2003]), is layered on top of a cloud computing system to make the storage accessible to all CPUs.

In short, the current generation of HPC systems and cloud systems use pretty much the same commercial hardware components. Their differences are primarily in the arrangement of the storage systems and networking systems. Clearly, the difference in the storage system designs could affect the application performance. However, the virtualization layer of the cloud systems is likely the bigger cause of application performance difference. In the next section, we will discuss another factor that could have an even larger impact, namely software tools and libraries.

Virtualization is generally used in the cloud computing environment to make the same hardware available to multiple users and to insulate one software environment from another. This is one of the more prominent features distinguishing the cloud computing environment from the HPC environment. In most cases, all three basic components of a computer system—CPU, storage, and networking—are all virtualized. This virtualization has many benefits. For example, an existing application can run on a CPU chip without recompiling; many users can share the same hardware; hardware faults could be corrected through the virtualization software; and applications on a failed compute node could be more easily migrated to another node. However, this virtualization layer also imposes some runtime overhead and could reduce application performance. For time-sensitive applications, this reduction in performance could become a critical issue.

Tests show that the performance differences could be quite large. Next, we briefly describe a performance study reported by Jackson et al [2010]. Figure 22.2 shows the performance slowdown using different computer systems. The names below the horizontal axis are different software packages commonly used at NERSC. The left bar corresponds to the Commercial Cloud, the middle bar to Magellan, and the (sometimes missing) right bar to the EC2-Beta-Opt system. The non-optimized commercial

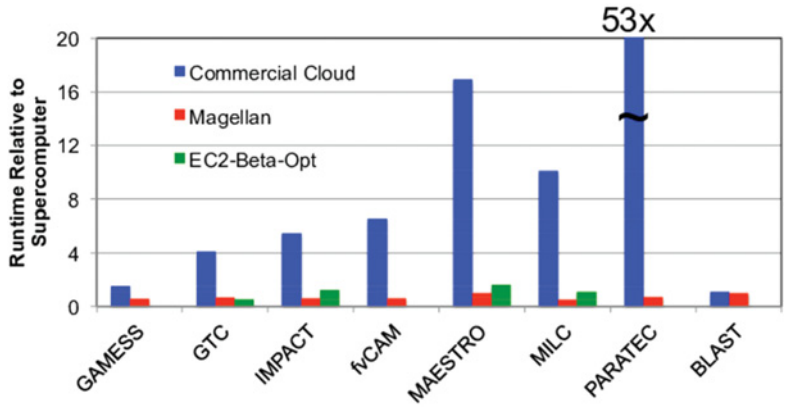


FIGURE 22.2 The cloud ran scientific applications considerably slower than on HPC systems (circa 2010)

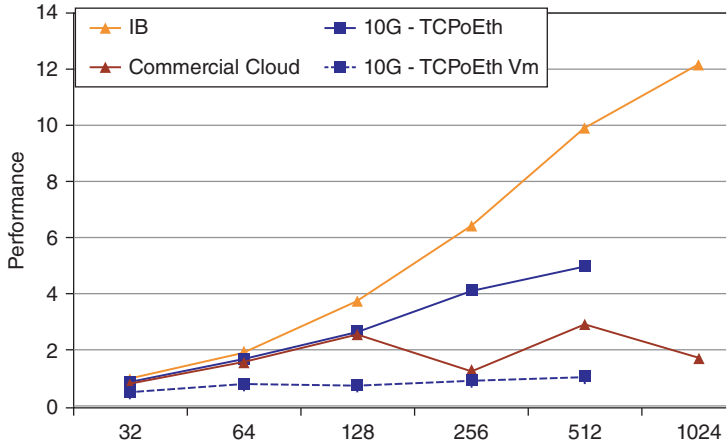


FIGURE 22.3 As the number of cores increases (horizontal axis), the virtualization overhead becomes much more significant (circa 2010)

cloud instances run these software packages 2 to 10 times slower than on a NERSC supercomputer. Even on the more expensive high-performance instances, there are noticeable slowdowns.

Figure 22.3 shows a study of the main factor causing the slowdown with the software package PARATEC. In Figure 2, we see that PARATEC took 53 times longer to complete on the commercial cloud than on an HPC system. We observe from Figure 3 that, as the number of cores (horizontal axis) increases, the differences among the measured performances (measured in TFLOP/s) become larger. In particular, the line labeled "10G- TCPoEth Vm" barely increases as the number of cores grows. This is the case where the network instance is using virtualized networking (TCP over Ethernet). It clearly shows that the networking virtualization overhead is significant, to the point of rendering the cloud useless.

The issue of virtualization overhead is widely recognized (Chen et al. [2015]). There has been considerable research aimed at addressing both the I/O virtualization overhead (Gordon et al. [2012]) as well as the networking virtualization overhead (Dong et al. [2012]). As these state-of-the-art techniques are gradually being moved into commercial products, we anticipate the overhead will decrease in the future, but some overhead will inevitably remain.

To wrap up this section, we briefly touch on the economics of HPC versus cloud. Typically, HPC systems are run by nonprofit research organizations and universities, while cloud systems are provided by commercial companies. Profit, customer retention, and many other factors affect the cost of a cloud system (Armburst et al. [2010]). In 2011, the Magellan project report stated that "Cost analysis shows that DOE centers are cost competitive, typically 3–7 × less expensive, when compared to commercial cloud providers" (Yelick et al. [2010]).

A group of high-energy physicists thought their use case was well-suited for cloud computing and conducted a detailed study of a comparison study (Holzman et al. [2017]). Their cost comparisons still show the commercial cloud offerings as

approximately 50% more expensive than dedicated HPC systems for comparable computing tasks; however, the authors worked with severe limitations on data ingress and egress to avoid potentially hefty charges on data movement. For complex workloads, such as the streaming data analyses discussed in this book, we anticipate that this HPC cost advantage will remain in the future. A 2016 National Academy of Sciences study came to the same conclusion that even a long-term lease from Amazon is likely 2 to 3 times more expensive than HPC systems to handle the expected science workload from NSF (Box 6.2 from National Academies of Sciences, [2016]).

22.5 HPC SOFTWARE

Ironically, the real power of a supercomputer is in its specialized software. There are a wide variety of software packages available for both HPC systems and cloud systems. In most cases, the same software package is available on both platforms. Therefore, we chose to focus on software packages that are unique to HPC systems and have the potential to improve computational intelligence and forecasting technologies.

One noticeable feature of the HPC software ecosystem is that much of the application software performs its own interprocessor communication through Message Passing Interface (MPI). In fact, the cornerstone of most scientific computing books is MPI (Kumar et al. [1994], Gropp, Lusk, and Skjellum [1999]). Accordingly, our discussion of HPC software tools will start with MPI. As this book relies on data processing algorithms, we will concentrate on data management tools (Shoshami and Rotem [2010]).

22.5.1 Message Passing Interface

Message Passing Interface is a communication protocol for parallel computing (Gropp, Lusk, and Skjellum [1999], Snir et al. [1988]). It defines a number of point-to-point data exchange operations as well as some collective communication operations. The MPI standard was established based on several early attempts to build portable communication libraries. The early implementation from Argonne National Lab, named MPICH, was high performance, scalable, and portable. This helped MPI to gain wide acceptance among scientific users.

The success of MPI is partly due to its separation of Language Independent Specifications (LIS) from its language bindings. This allows the same core function to be provided to many different programming languages, which also contributes to its acceptance. The first MPI standard specified ANSI C and Fortran-77 bindings together with the LIS. The draft specification was presented to the user community at the 1994 Supercomputing Conference.

Another key factor contributing to MPI's success is the open-source license used by MPICH. This license allows the vendors to take the source code to produce their own custom versions, which allows the HPC system vendors to quickly produce their own MPI libraries. To this day, all HPC systems support the familiar MPI on their computers. This wide adoption also ensures that MPI will continue to be the favorite communication protocol among the users of HPC systems.

22.5.2 Hierarchical Data Format 5

In describing the HPC hardware components, we noted that the storage systems in an HPC platform are typically different from those in a cloud platform. Correspondingly, the software libraries used by most users for accessing the storage systems are different as well. This difference can be traced to the difference in the conceptual models of data. Typically, HPC applications treat data as multi-dimensional arrays and, therefore, the most popular I/O libraries on HPC systems are designed to work with multi-dimensional arrays. Here, we describe the most widely used array format library, HDF5 (Folk et al. [2011]).

HDF5 is the fifth iteration of the Hierarchical Data Format, produced by the HDF Group.[2] The basic unit of data in HDF5 is an array plus its associated information such as attributes, dimensions, and data type. Together, they are known as a data set. Data sets can be grouped into large units called groups, and groups can be organized into high-level groups. This flexible hierarchical organization allows users to express complex relationships among the data sets.

Beyond the basic library for organizing user data into files, the HDF Group also provides a suite of tools and specialization of HDF5 for different applications. For example, HDF5 includes a performance profiling tool. NASA has a specialization of HDF5, named HDF5-EOS, for data from their Earth-Observing System (EOS); and the next-generation DNA sequence community has produced a specialization named BioHDF for their bioinformatics data.

HDF5 provides an efficient way for accessing the storage systems on HPC platform. In tests, we have demonstrated that using HDF5 to store stock markets data significantly speeds up the analysis operations. This is largely due to its efficient compression/decompression algorithms that minimize network traffic and I/O operations, which brings us to our next point.

22.5.3 *In Situ* Processing

Over the last few decades, CPU performance has roughly doubled every 18 months (Moore's law), while disk performance has been increasing less than 5% a year. This difference has caused it to take longer and longer to write out the content of the CPU memory. To address this issue, a number of research efforts have focused on *in situ* analysis capability (Ayachit et al. [2016]).

Among the current generation of processing systems, the Adaptable I/O System (ADIOS) is the most widely used (Liu et al. [2014]). It employs a number of data transport engines that allow users to tap into the I/O stream and perform analytical operations. This is useful because irrelevant data can be discarded in-flight, hence avoiding its slow and voluminous storage. This same *in situ* mechanism also allows it to complete write operations very quickly. In fact, it initially gained attention because of its write speed. Since then, the ADIOS developers have worked with a number of very large teams to improve their I/O pipelines and their analysis capability.

[2] The HDF Group web site is https://www.hdfgroup.org/.

Because ADIOS supports streaming data accesses, it is also highly relevant to CIFT work. In a number of demonstrations, ADIOS with ICEE transport engine was able to complete distributed streaming data analysis in real-time (Choi et al. [2013]). We will describe one of the use cases involving blobs in fusion plasma in the next section.

To summarize, *in situ* data processing capability is another very useful tool from the HPC ecosystem.

22.5.4 Convergence

We mentioned earlier that the HPC hardware market is a tiny part of the overall computer hardware market. The HPC software market is even smaller compared to the overall software market. So far, the HPC software ecosystem is largely maintained by a number of small vendors along with some open-source contributors. Therefore, HPC system users are under tremendous pressure to migrate to the better supported cloud software systems. This is a significant driver for convergence between software for HPC and software for cloud (Fox et al. [2015]).

Even though convergence appears to be inevitable, we advocate for a convergence option that keeps the advantage of the software tools mentioned above. One of the motivations of the CIFT project is to seek a way to transfer the above tools to the computing environments of the future.

22.6 USE CASES

Data processing is such an important part of modern scientific research that some researchers are calling it the fourth paradigm of science (Hey, Tansley, and Tolle [2009]). In economics, the same data-driven research activities have led to the wildly popular behavioral economics (Camerer and Loewenstein [2011]). Much of the recent advances in data-driven research are based on machine learning applications (Qiu et al. [2016], Rudin and Wagstaff [2014]). Their successes in a wide variety of fields, such as planetary science and bioinformatics, have generated considerable interest among researchers from diverse domains. In the rest of this section, we describe a few examples applying advanced data analysis techniques to various fields, where many of these use cases originated in the CIFT project.

22.6.1 Supernova Hunting

In astronomy, the determination of many important facts such as the expansion speed of the universe, is performed by measuring the light from exploding type Ia supernovae (Bloom et al. [2012]). The process of searching the night sky for exploding supernovae is called synoptic imaging survey. The Palomar Transient Factory (PTF) is an example of such a synoptic survey (Nicholas et al. [2009]). The PTF telescopes scan the night sky and produce a set of images every 45 minutes. The new image is compared against the previous observations of the same patch of sky to determine

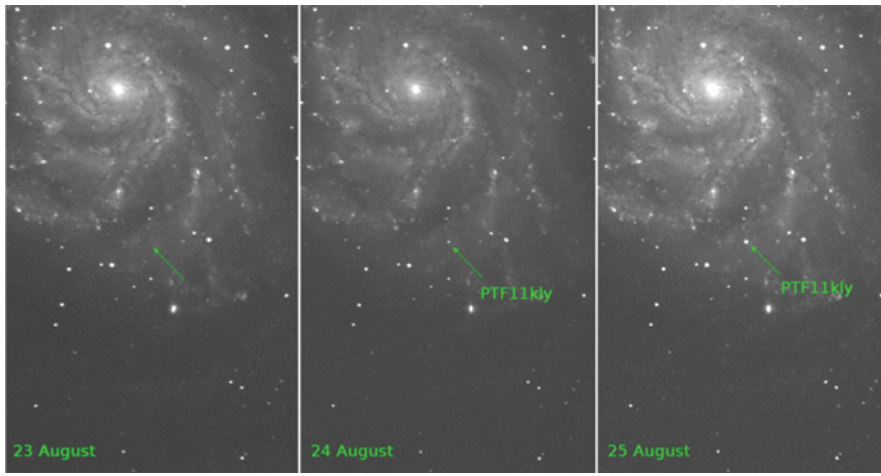


FIGURE 22.4 Supernova SN 2011fe was discovered 11 hours after first evidence of explosion, as a result of the extensive automation in classification of astronomical observations

what has changed and to classify the changes. Such identification and classification tasks used to be performed by astronomers manually. However, the current number of incoming images from the PTF telescopes is too large for manual inspection. An automated workflow for these image processing tasks has been developed and deployed at a number of different computer centers.

Figure 22.4 shows the supernova that was identified earliest in its explosion process. On August 23, 2011, a patch of the sky showed no sign of this star, but a faint light showed up on August 24. This quick turnover allowed astronomers around the world to perform detailed follow-up observations, which are important for determining the parameters related to the expansion of the universe.

The quick identification of this supernova is an important demonstration of the machine learning capability of the automated workflow. This workflow processes the incoming images to extract the objects that have changed since last observed. It then classifies the changed object to determine a preliminary type based on the previous training. Since follow-up resources for extracting novel science from fast-changing transients are precious, the classification not only needs to indicate the assumed type but also the likelihood and confidence of the classification. Using classification algorithms trained on PTF data, the mislabeling of transients and variable stars has a 3.8% overall error rate. Additional work is expected to achieve higher accuracy rates in upcoming surveys, such as for the Large Synoptic Survey Telescope.

22.6.2 Blobs in Fusion Plasma

Large-scale scientific exploration in domains such as physics and climatology are huge international collaborations involving thousands of scientists each. As these

collaborations produce more and more data at progressively faster rates, the existing workflow management systems are hard-pressed to keep pace. A necessary solution is to process, analyze, summarize, and reduce the data before it reaches the relatively slow disk storage system, a process known as in-transit processing (or in-flight analysis). Working with the ADIOS developers, we have implemented the ICEE transport engine to dramatically increase the data-handling capability of collaborative workflow systems (Choi et al. [2013]). This new feature significantly improved the data flow management for distributed workflows. Tests showed that the ICEE engine allowed a number of large international collaborations to make near real-time collaborative decisions. Here, we briefly describe the fusion collaboration involving KSTAR.

KSTAR is a nuclear fusion reactor with fully superconducting magnets. It is located in South Korea, but there are a number of associated research teams around the world. During a run of a fusion experiment, some researchers control the physics device at KSTAR, but others may want to participate by performing collaborative analysis of the preceding runs of the experiment to provide advice on how to configure the device for the next run. During the analysis of the experimental measurement data, scientists might run simulations or examine previous simulations to study parametric choices. Typically, there may be a lapse of 10 to 30 minutes between two successive runs, and all collaborative analyses need to complete during this time window in order to affect the next run.

We have demonstrated the functionality of the ICEE workflow system with two different types of data: one from the Electron Cyclotron Emission Imaging (ECEI) data measured at KSTAR, and the other involving synthetic diagnostic data from the XGC modelling. The distributed workflow engine needs to collect data from these two sources, extract a feature known as blobs, track the movement of these blobs, predict the movement of the blobs in the experimental measurements, and then provide advices on actions to be performed. Figure 22.5 shows how the ECEI data is processed. The workflow for the XGC simulation data is similar to what is shown in Figure 22.5, except that the XGC data is located at NERSC.

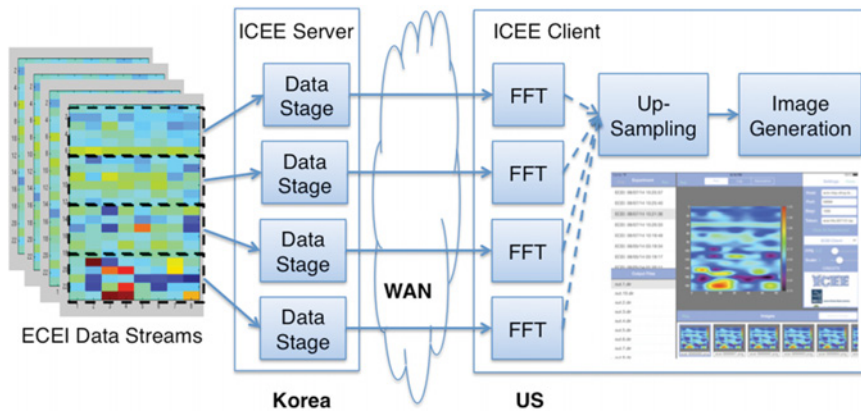


FIGURE 22.5 A distributed workflow for studying fusion plasma dynamics

To be able to complete the above analytical tasks in real-time, effective data management with ICEE transport engine of ADIOS is only part of the story. The second part is to detect blobs efficiently (Wu et al. [2016]). In this work, we need to reduce the amount of data transported across wide-area networks by selecting only the necessary chunks. We then identify all cells within the blobs and group these cells into connected regions in space, where each connected region forms a blob. The new algorithm we developed partitions the work into different CPU cores by taking full advantage of the MPI for communication between the nodes and the shared memory among the CPU cores on the same node. Additionally, we also updated the connected component label algorithm to correctly identify blobs at the edge, which were frequently missed by the earlier detection algorithms. Overall, our algorithm was able to identify blobs in a few milliseconds for each time step by taking full advantage of the parallelism available in the HPC system.

22.6.3 Intraday Peak Electricity Usage

Utility companies are deploying advanced metering infrastructure (AMI) to capture electricity consumption in unprecedented spatial and temporal detail. This vast and fast-growing stream of data provides an important testing ground for the predictive capability based on big data analytical platforms (Kim et al. [2015]). These cutting-edge data science techniques, together with behavioral theories, enable behavior analytics to gain novel insights into patterns of electricity consumption and their underlying drivers (Todd et al. [2014]).

As electricity cannot be easily stored, its generation must match consumption. When the demand exceeds the generation capacity, a blackout will occur, typically during the time when consumers need electricity the most. Because increasing generation capacity is expensive and requires years of time, regulators and utility companies have devised a number of pricing schemes intended to discourage unnecessary consumption during peak demand periods.

To measure the effectiveness of a pricing policy on peak demand, one can analyze the electricity usage data generated by AMI. Our work focuses on extracting baseline models of household electricity usage for a behavior analytics study. The baseline models would ideally capture the pattern of household electricity usage including all features except the new pricing schemes. There are numerous challenges in establishing such a model. For example, there are many features that could affect the usage of electricity but for which no information is recorded, such as the temperature set point of an air-conditioner or the purchase of a new appliance. Other features, such as outdoor temperature, are known, but their impact is difficult to capture in simple functions.

Our work developed a number of new baseline models that could satisfy the above requirements. At present, the gold standard baseline is a well-designed randomized control group. We showed that our new data-driven baselines could accurately predict the average electricity usage of the control group. For this evaluation, we use a well-designed study from a region of the United States where the electricity usage is the highest in the afternoon and evening during the months of May through August.

Though this work concentrates on demonstrating that the new baseline models are effective for groups, we believe that these new models are also useful for studying individual households in the future.

We explored a number of standard black-box approaches. Among machine learning methods, we found gradient tree boosting (GTB) to be more effective than others. However, the most accurate GTB models require lagged variables as features (for example, the electricity usage a day before and a week before). In our work, we need to use the data from year T-1 to establish the baseline usage for year T and year T + 1. The lagged variable for a day before and a week before would be incorporating recent information not in year T-1. We attempted to modify the prediction procedure to use the recent predictions in place of the actual measured values a day before and a week before; however, our tests show that the prediction errors accumulate over time, leading to unrealistic predictions a month or so into the summer season. This type of accumulation of prediction errors is common to continuous prediction procedures for time series.

To address the above issue, we devised a number of white-box approaches, the most effective of which, known as LTAP, is reported here. LTAP is based on the fact that the aggregate variable electricity usage per day is accurately described by a piecewise linear function of average daily temperature. This fact allows us to make predictions about the total daily electricity usage. By further assuming that the usage profile of each household remains the same during the study, we are able to assign the hourly usage values from the daily aggregate usage. This approach is shown to be self-consistent; that is, the prediction procedure exactly reproduces the electricity usage in year T–1, and the predictions for the control group in both year T and T + 1 are very close to the actual measured values. Both treatment groups have reduced electricity usages during the peak-demand hours, and the active group reduced the usage more than the passive group. This observation is in line with other studies.

Though the new data-driven baseline model LTAP predicts the average usages of the control group accurately, there are some differences in predicted impact of the new time-of-use pricing intended to reduce the usage during the peak-demand hours (see Figure 22.6). For example, with the control group as the baseline, the active group reduces its usage by 0.277 kWh (out of about 2 kWh) averaged over the peak-demand hours in the first year with the new price and 0.198 kWh in the second year. Using LTAP as the baseline, the average reductions are only 0.164 kWh for both years. Part of the difference may be due to the self-selection bias in treatment groups, especially the active group, where the households have to explicitly opt-in to participate in the trial. It is likely that the households that elected to join the active group are well-suited to take advantage of the proposed new pricing structure. We believe that the LTAP baseline is a way to address the self-selection bias and plan to conduct additional studies to further verify this.

22.6.4 The Flash Crash of 2010

The extended time it took for the SEC and CFTC to investigate the Flash Crash of 2010 was the original motivation for CIFT's work. Federal investigators needed to

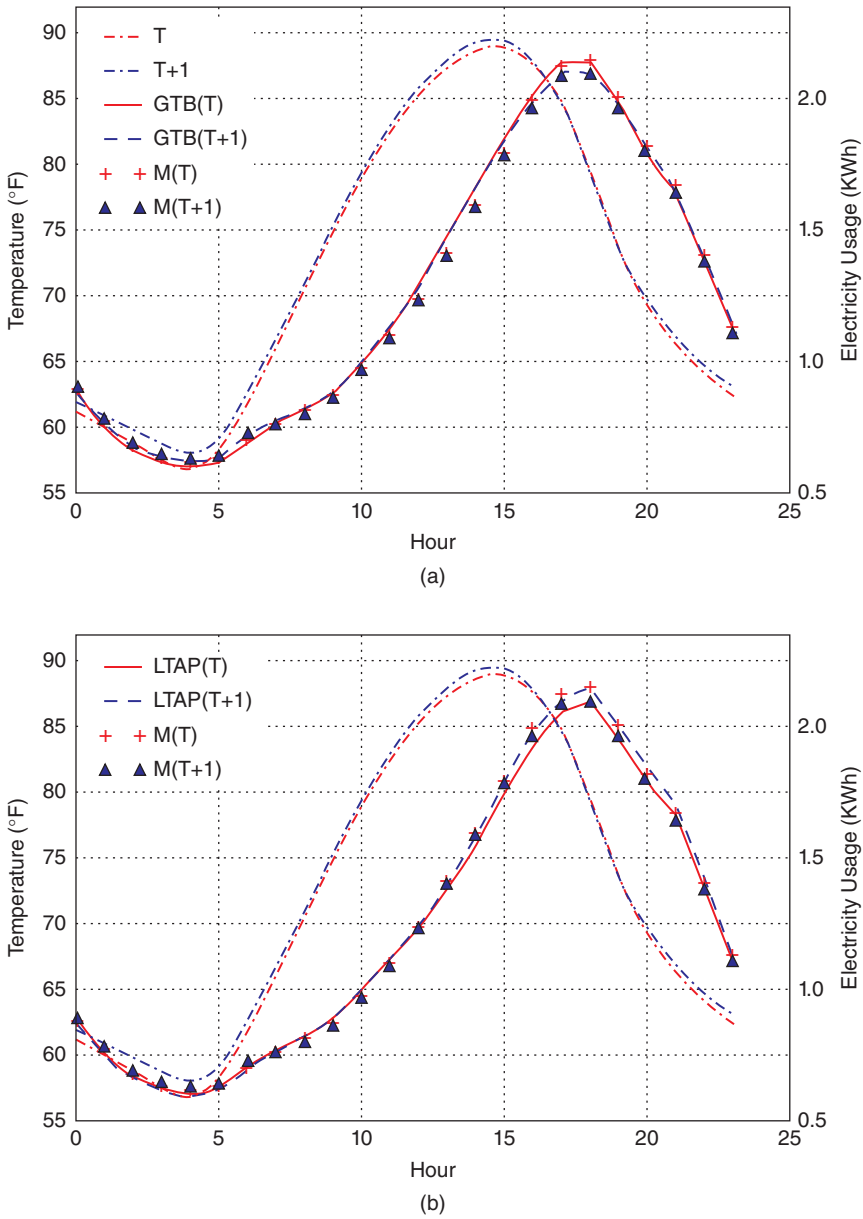
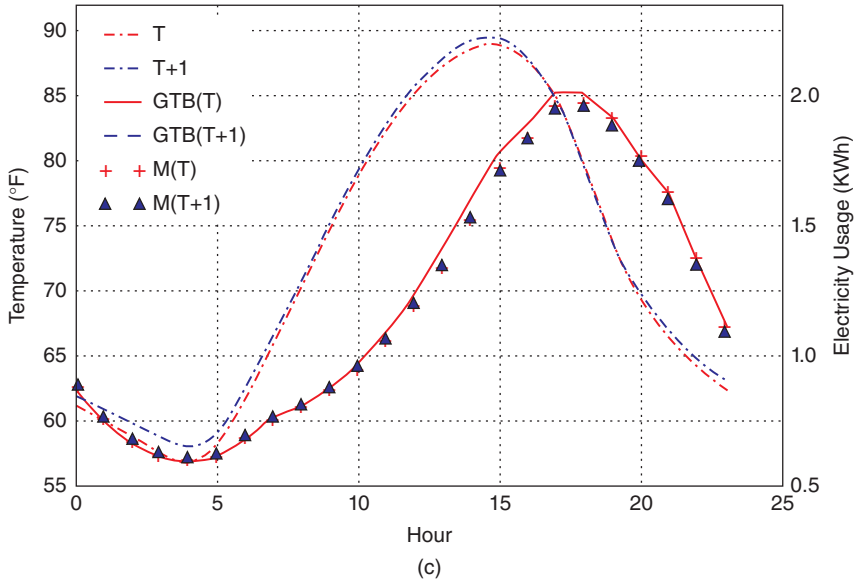


FIGURE 22.6 Gradient tree boosting (GBT) appears to follow recent usage too closely and therefore not able to predict the baseline usage as well as the newly develop method named LTAP. (a) GTB on Control group. (b) LTAP on Control group. (c) GTB on Passive group. (d) LTAP on Passive group. (e) GTB on Active group. (f) LTAP on Active group



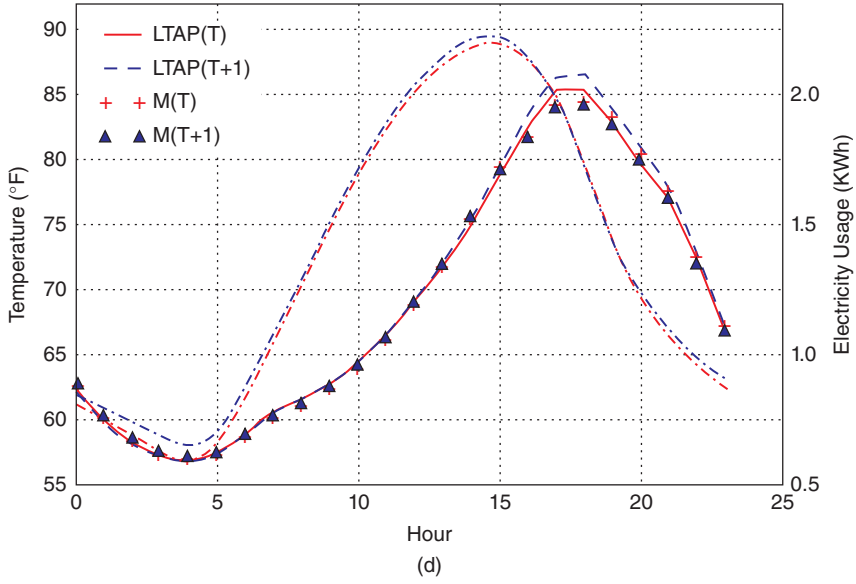
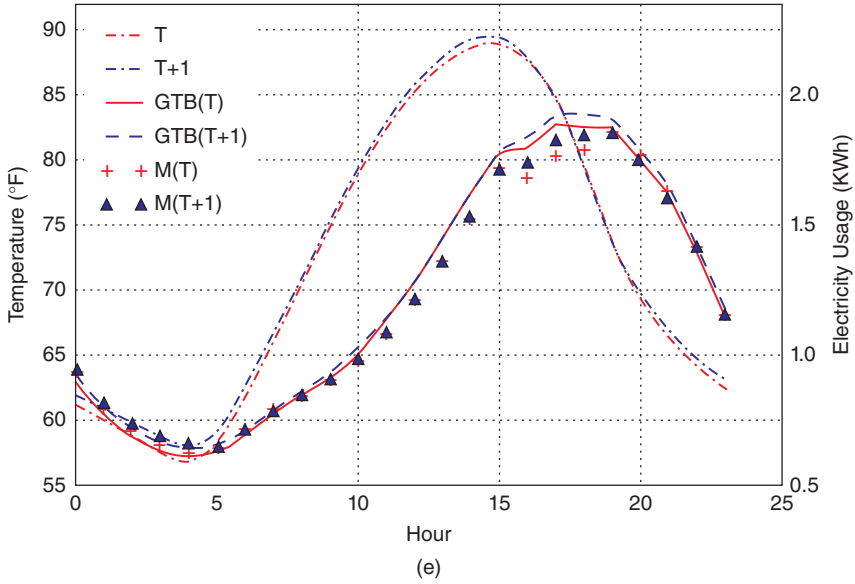


FIGURE 22.6 (*Continued*)

sift through tens of terabytes of data to look for the root cause of the crash. Since CFTC publicly blamed the volume of data to be the source of the long delay, we started our work by looking for HPC tools that could easily handle tens of terabytes. Since HDF5 is the most commonly used I/O library, we started our work by applying HDF5 to organize a large set of stock trading data (Bethel et al. [2011]).



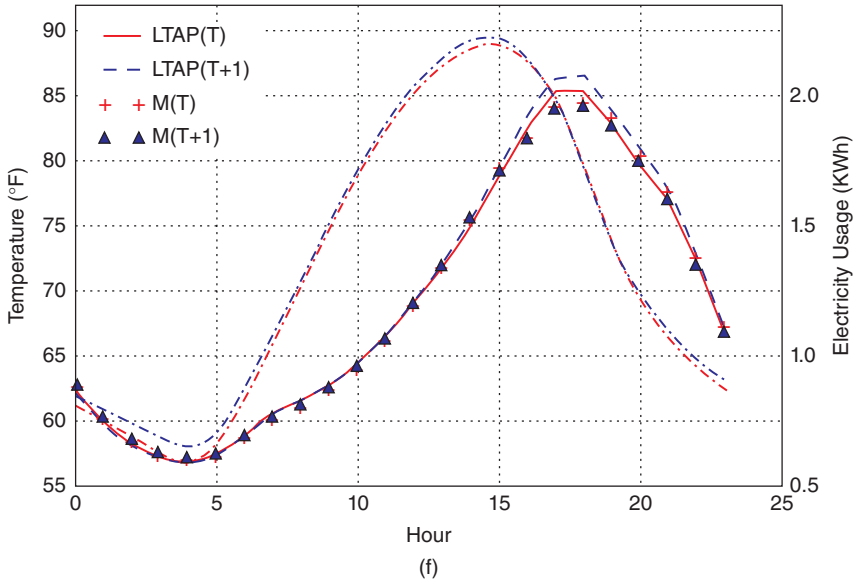


FIGURE 22.6 (*Continued*)

Let us quickly review what happened during the 2010 Flash Crash. On May 6, at about 2:45 p.m. (U.S. Eastern Daylight Time), the Dow Jones Industrial Average dropped almost 10%, and many stocks traded at one cent per share, the minimum price for any possible trade. Figure 22.7 shows an example of another extreme case, where shares of Apple (symbol AAPL) traded at $100,000 per share, the maximum

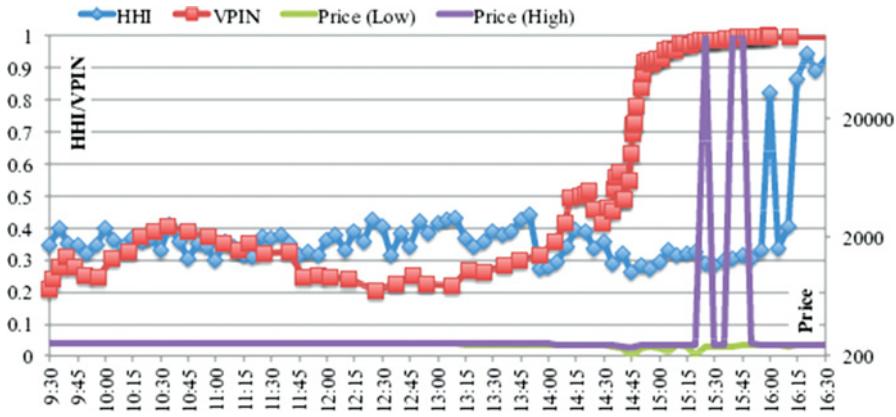


FIGURE 22.7 Apple Stock price on May 6, 2010, along with HHI and VPIN values computed every 5 minutes during the market hours

possible price allowed by the exchange. Clearly, these were unusual events, which undermined investors' faith and confidence in our financial markets. Investors demanded to know what caused these events.

To make our work relevant to the financial industry, we sought to experiment with the HDF5 software, and apply it to the concrete task of computing earlier warning indicators. Based on recommendations from a group of institutional investors, regulators, and academics, we implemented two sets of indicators that have been shown to have "early warning" properties preceding the Flash Crash. They are the Volume Synchronized Probability of Informed Trading (VPIN) (Easley, Lopez de Prado, and O'Hara [2011]) and a variant of the Herfindahl-Hirschman Index (HHI) (Hirschman [1980]) of market fragmentation. We implemented these two algorithms in the C++ language, while using MPI for inter-processor communication, to take full advantage of the HPC systems. The reasoning behind this choice is that if any of these earlier warning indicators is shown to be successful, the high-performance implementation would allow us to extract the warning signals as early as possible so there might be time to take corrective actions. Our effort was one of the first steps to demonstrate that it is possible to compute the earlier warning signals fast enough.

For our work, we implemented two versions of the programs: one uses data organized in HDF5 files, and another reads the data from the commonly used ASCII text files. Figure 22.8 shows the time required to process the trading records of all S&P 500 stocks over a 10-year timespan. Since the size of the 10-year trading data is still relatively small, we replicated the data 10 times as well. On a single CPU core (labeled "Serial" in Figure 22.8), it took about 3.5 hours with ASCII data, but only 603.98 seconds with HDF5 files. When 512 CPU cores are used, this time reduces to 2.58 seconds using HDF5 files, resulting in a speedup of 234 times.

On the larger (replicated) dataset, the advantage of HPC code for computing these indices is even more pronounced. With 10 times as much data, it took only about 2.3

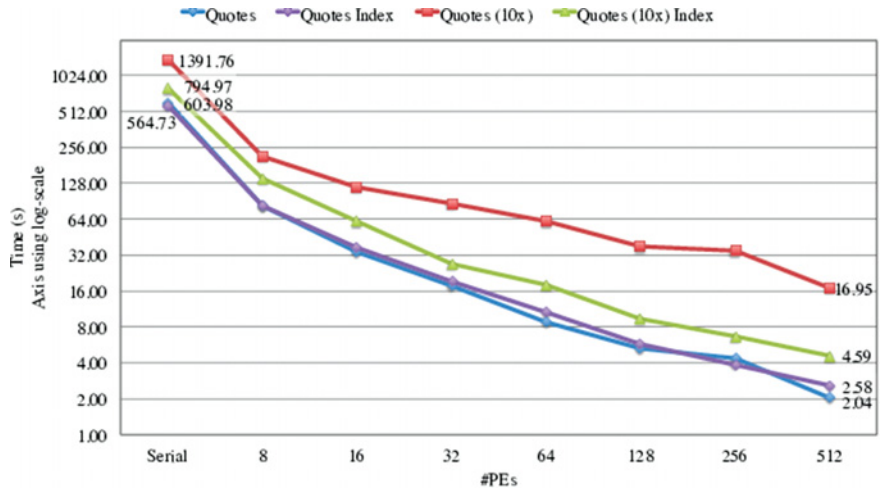


FIGURE 22.8 Time to process 10-year worth of SP500 quotes data stored in HDF5 files, which takes 21 times longer when the same data is in ASCII files (603.98 seconds versus approximately 3.5 hours)

times longer for the computer to complete the tasks, a below-linear latency increase. Using more CPU makes HPC even more scalable.

Figure 22.8 also shows that with a large data set, we can further take advantage of the indexing techniques available in HDF5 to reduce the data access time (which in turn reduces the overall computation time). When 512 CPU cores are used, the total runtime is reduced from 16.95 seconds to 4.59 seconds, a speedup of 3.7 due to this HPC technique of indexing.

22.6.5 Volume-synchronized Probability of Informed Trading Calibration

Understanding the volatility of the financial market requires the processing of a vast amount of data. We apply techniques from data-intensive scientific applications for this task, and demonstrate their effectiveness by computing an early warning indicator called Volume Synchronized Probability of Informed Trading (VPIN) on a massive set of futures contracts. The test data contains 67 months of trades for the hundred most frequently traded futures contracts. On average, processing one contract over 67 months takes around 1.5 seconds. Before we had this HPC implementation, it took about 18 minutes to complete the same task. Our HPC implementation achieves a speedup of 720 times.

Note that the above speedup was obtained solely based on the algorithmic improvement, without the benefit of parallelization. The HPC code can run on parallel machines using MPI, and thus is able to further reduce the computation time.

The software techniques employed in our work include the faster I/O access through HDF5 described above, as well as a more streamlined data structure for storing the bars and buckets used for the computation of VPIN. More detailed information is available in Wu et al. [2013].

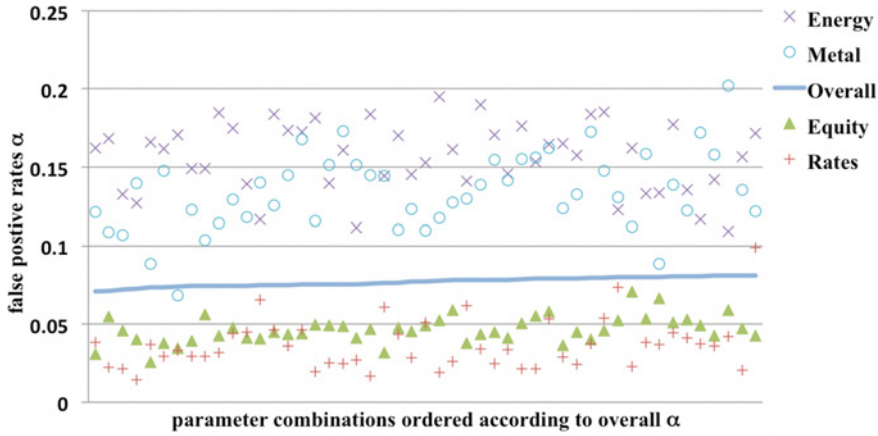


FIGURE 22.9 The average false positive rates (α) of different classes of futures contracts ordered according to their average.

With a faster program to compute VPIN, we were also able to explore the parametric choices more closely. For example, we were able to identify the parameter values that reduce VPIN's false positive rate over one hundred contracts from 20% to only 7%, see Figure 22.9. The parameter choices to achieve this performance are: (1) pricing the volume bar with the median prices of the trades (not the closing price typically used in analyses), (2) 200 buckets per day, (3) 30 bars per bucket, (4) support window for computing VPIN = 1 day, event duration = 0.1 day, (5) bulk volume classification with Student t-distribution with $\nu = 0.1$, and (6) threshold for CDF of VPIN = 0.99. Again, these parameters provide a low false positive rate on the totality of futures contracts, and are not the result of individual fitting.

On different classes of futures contracts, it is possible to choose different parameters to achieve even lower false positive rates. In some cases, the false positive rates can fall significantly below 1%. Based on Figure 22.9, interest rate and index futures contracts typically have lower false positive rates. The futures contracts on commodities, such as energy and metal, generally have higher false positive rates.

Additionally, a faster program for computing VPIN allows us to validate that the events identified by VPIN are "intrinsic," in the sense that varying parameters such as the threshold on VPIN CDF only slightly change the number of events detected. Had the events been random, changing this threshold from 0.9 to 0.99 would have reduced the number of events by a factor of 10. In short, a faster VPIN program also allows us to confirm the real-time effectiveness of VPIN.

22.6.6 Revealing High Frequency Events with Non-uniform Fast Fourier Transform

High Frequency Trading is pervasive across all electronic financial markets. As algorithms replace tasks previously performed by humans, cascading effects similar to the 2010 Flash Crash may become more likely. In our work (Song et al. [2014]), we

brought together a number of high performance signal-processing tools to improve our understanding of these trading activities. As an illustration, we summarize the Fourier analysis of the trading prices of natural gas futures.

Normally, Fourier analysis is applied on uniformly spaced data. Since market activity comes in bursts, we may want to sample financial time series according to an index of trading activity. For example, VPIN samples financial series as a function of volume traded. However, a Fourier analysis of financial series in chronological time may still be instructive. To this purpose, we use a non-uniform Fast Fourier Transform (FFT) procedure.

From the Fourier analysis of the natural gas futures market, we see strong evidences of High Frequency Trading in the market. The Fourier components corresponding to high frequencies are (1) becoming more prominent in the recent years and (2) are much stronger than could be expected from the structure of the market. Additionally, a significant amount of trading activity occurs in the first second of every minute, which is a tell-tale sign of trading triggered by algorithms that target a Time-Weighted Average Price (TWAP).

Fourier analysis on trading data shows that activities at the once-per-minute frequency are considerably higher than at neighboring frequencies (see Figure 22.10). Note that the vertical axis is in logarithmic scale. The strength of activities at once-per-minute frequency is more than ten times stronger than the neighboring frequencies. Additionally, the activity is very precisely defined at once-per-minute, which indicates that these trades are triggered by intentionally constructed automated events. We take this to be strong evidence that TWAP algorithms have a significant presence in this market.

We expected the frequency analysis to show strong daily cycles. In Figure 22.10, we expect amplitude for frequency 365 to be large. However, we see the highest

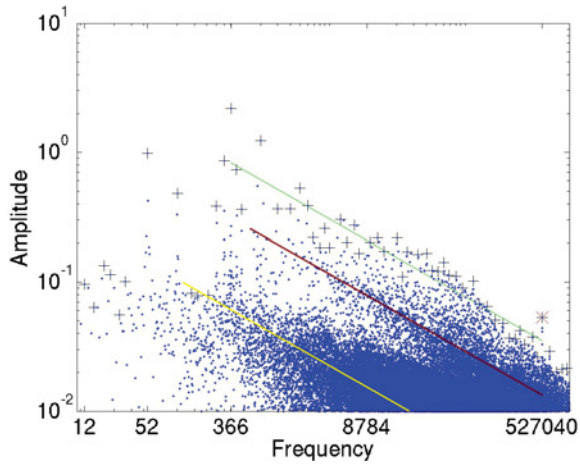


FIGURE 22.10 Fourier spectrum of trading prices of natural gas futures contracts in 2012. Non-uniform FFT identifies strong presence of activities happening once per day (frequency = 366), twice per day (frequency = 732), and once per minute (frequency = 527040 = 366*24*60).

amplitude was for the frequency of 366. This can be explained because 2012 was a leap year. This is a validation that the non-uniform FFT is capturing the expected signals. The second- and third-highest amplitudes have the frequencies of 732 and 52, which are twice-a-day and once-a-week. These are also unsurprising.

We additionally applied the non-uniform FFT on the trading volumes and found further evidence of algorithmic trading. Moreover, the signals pointed to a stronger presence of algorithmic trading in recent years. Clearly, the non-uniform FFT algorithm is useful for analyzing highly irregular time series.

22.7 SUMMARY AND CALL FOR PARTICIPATION

Currently, there are two primary ways to construct large-scale computing platforms: the HPC approach and the cloud approach. Most of the scientific computing efforts use the HPC approach, while most of the business computing needs are satisfied through the cloud approach. The conventional wisdom is that the HPC approach occupies a small niche of little consequence. This is not true. HPC systems are essential to the progress of scientific research. They played important roles in exciting new scientific discoveries including the Higgs particle and gravitational waves. They have spurred the development of new subjects of study, such as behavioral economics, and new ways of conducting commerce through the Internet. The usefulness of extremely large HPC systems has led to the 2015 National Strategic Computing Initiative.[3]

There are efforts to make HPC tools even more useful by accelerating their adoption in business applications. The HPC4Manufacturing[4] effort is pioneering this knowledge transfer to the U.S. manufacturing industry, and has attracted considerable attention. Now is the time to make a more concerted push for HPC to meet other critical business needs.

In recent years, we have developed CIFT as a broad class of business applications that could benefit from the HPC tools and techniques. In decisions such as how to respond to a voltage fluctuation in a power transformer and an early warning signal of impending market volatility event, HPC software tools could help determine the signals early enough for decision makers, provide sufficient confidence about the prediction, and anticipate the consequence before the catastrophic event arrives. These applications have complex computational requirements and often have a stringent demand on response time as well. HPC tools are better suited to meet these requirements than cloud-based tools.

In our work, we have demonstrated that the HPC I/O library HDF5 can be used to accelerate the data access speed by 21-fold, and HPC techniques can accelerate the computation of the Flash Crash early-warning indicator VPIN by 720-fold. We have developed additional algorithms that enable us to predict the daily peak electricity

[3] The National Strategic Computing Initiative plan is available online at https://www.whitehouse.gov/sites/whitehouse.gov/files/images/NSCI%20Strategic%20Plan.pdf. The Wikipedia page on this topic (https://en.wikipedia.org/wiki/National_Strategic_Computing_Initiative) also has some useful links to additional information.

[4] Information about HPC4Manufacturing is available online at https://hpc4mfg.llnl.gov/.

usage years into the future. We anticipate that applying HPC tools and techniques to other applications could achieve similarly significant results.

In addition to the performance advantages mentioned above, a number of published studies (Yelick et al. [2011], Holzman et al. [2017]) show HPC systems to have a significant price advantage as well. Depending on the workload's requirement on CPU, storage, and networking, using a cloud system might cost 50% more than using a HPC system, and, in some cases, as much as seven times more. For the complex analytical tasks described in this book, with their constant need to ingest data for analysis, we anticipate the cost advantage will continue to be large.

CIFT is expanding the effort to transfer HPC technology to private companies, so that they can also benefit from the price and performance advantages enjoyed by large-scale research facilities. Our earlier collaborators have provided the funds to start a dedicated HPC system for our work. This resource should make it considerably easier for interested parties to try out their applications on an HPC system. We are open to different forms of collaborations. For further information regarding CIFT, please visit CIFT's web page at http://crd.lbl.gov/cift/.

22.8 ACKNOWLEDGMENTS

The CIFT project is the brainchild of Dr. David Leinweber. Dr. Horst Simon brought it to LBNL in 2010. Drs. E. W. Bethel and D. Bailey led the project for four years.

The CIFT project has received generous gifts from a number of donors. This work is supported in part by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research also uses resources of the National Energy Research Scientific Computing Center supported under the same contract.

REFERENCES

Aad, G., et al. (2016): "Measurements of the Higgs boson production and decay rates and coupling strengths using *pp* collision data at $\sqrt{s} = 7$ and 8 TeV in the ATLAS experiment." *The European Physical Journal C*, Vol. 76, No. 1, p. 6.

Abbott, B.P. et al. (2016): "Observation of gravitational waves from a binary black hole merger." *Physical Review Letters*, Vol. 116, No. 6, p. 061102.

Armbrust, M., et al. (2010): "A view of cloud computing." *Communications of the ACM*, Vol. 53, No. 4, pp. 50–58.

Asanovic, K. et al. (2006): "The landscape of parallel computing research: A view from Berkeley." *Technical Report UCB/EECS-2006-183*, EECS Department, University of California, Berkeley.

Ayachit, U. et al. "Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press.

Bethel, E. W. et al. (2011): "Federal market information technology in the post Flash Crash era: Roles for supercomputing." Proceedings of WHPCF'2011. ACM. pp. 23–30.

Bloom, J. S. et al. (2012): "Automating discovery and classification of transients and variable stars in the synoptic survey era." *Publications of the Astronomical Society of the Pacific*, Vol. 124, No. 921, p. 1175.

Camerer, C.F. and G. Loewenstein (2011): "Behavioral economics: Past, present, future." In *Advances in Behavioral Economics*, pp. 1–52.

Chen, L. et al. (2015): "Profiling and understanding virtualization overhead in cloud." *Parallel Processing (ICPP)*, 2015 44th International Conference. IEEE.

Choi, J.Y. et al. (2013): ICEE: "Wide-area in transit data processing framework for near real-time scientific applications." 4th SC Workshop on Petascale (Big) Data Analytics: Challenges and Opportunities in Conjunction with SC13.

Dong, Y. et al. (2012): "High performance network virtualization with SR-IOV." *Journal of Parallel and Distributed Computing*, Vol. 72, No. 11, pp. 1471–1480.

Easley, D., M. Lopez de Prado, and M. O'Hara (2011): "The microstructure of the 'Flash Crash': Flow toxicity, liquidity crashes and the probability of informed trading." *Journal of Portfolio Management*, Vol. 37, No. 2, pp. 118–128.

Folk, M. et al. (2011): "An overview of the HDF5 technology suite and its applications." Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. ACM.

Fox, G. et al. (2015): "Big Data, simulations and HPC convergence, iBig Data benchmarking": 6th International Workshop, WBDB 2015, Toronto, ON, Canada, June 16–17, 2015; and 7th International Workshop, WBDB 2015, New Delhi, India, December 14–15, 2015, Revised Selected Papers, T. Rabl, et al., eds. 2016, Springer International Publishing: Cham. pp. 3–17. DOI: 10.1007/978-3-319-49748-8_1.

Ghemawat, S., H. Gobioff, and S.-T. Leung (2003): "The Google file system," *SOSP '03: Proceedings of the nineteenth ACM symposium on operating systems principles*. ACM. pp. 29–43.

Gordon, A. et al. (2012): "ELI: Bare-metal performance for I/O virtualization." *SIGARCH Comput. Archit. News*, Vol. 40, No. 1, pp. 411–422.

Gropp, W., E. Lusk, and A. Skjellum (1999): *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press.

Hey, T., S. Tansley, and K.M. Tolle (2009): *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Vol. 1. Microsoft research Redmond, WA.

Hirschman, A. O. (1980): *National Power and the Structure of Foreign Trade*. Vol. 105. University of California Press.

Holzman, B. et al. (2017): "HEPCloud, a new paradigm for HEP facilities: CMS Amazon Web Services investigation. *Computing and Software for Big Science*, Vol. 1, No. 1, p. 1.

Jackson, K. R., et al. (2010): "Performance analysis of high performance computing applications on the Amazon Web Services Cloud. *Cloud Computing Technology and Science (CloudCom)*. 2010 Second International Conference. IEEE.

Kim, T. et al. (2015): "Extracting baseline electricity usage using gradient tree boosting." IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). IEEE.

Kumar, V. et al. (1994): *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company.

Liu, Q. et al., (2014): "Hello ADIOS: The challenges and lessons of developing leadership class I/O frameworks." *Concurrency and Computation: Practice and Experience*, Volume 26, No. 7, pp. 1453–1473.

National Academies of Sciences, Engineering and Medicine (2016): *Future Directions for NSF Advanced Computing Infrastructure to Support U.S. Science and Engineering in 2017–2020*. National Academies Press.

Nicholas, M. L. et al. (2009): "The Palomar transient factory: System overview, performance, and first results." *Publications of the Astronomical Society of the Pacific*, Vol. 121, No. 886, p. 1395.

Qiu, J. et al. (2016): "A survey of machine learning for big data processing." *EURASIP Journal on Advances in Signal Processing*, Vol. 2016, No. 1, p. 67. DOI: 10.1186/s13634-016-0355-x

Rudin, C. and K. L. Wagstaff (2014) "Machine learning for science and society." *Machine Learning*, Vol. 95, No. 1, pp. 1–9.

Shoshani, A. and D. Rotem (2010): "Scientific data management: Challenges, technology, and deployment." *Chapman & Hall/CRC Computational Science Series*. CRC Press.

Snir, M. et al. (1998): *MPI: The Complete Reference. Volume 1, The MPI-1 Core*. MIT Press.

Song, J. H. et al. (2014): "Exploring irregular time series through non-uniform fast Fourier transform." Proceedings of the 7th Workshop on High Performance Computational Finance, IEEE Press.

Todd, A. et al. (2014): "Insights from Smart Meters: The potential for peak hour savings from behavior-based programs." Lawrence Berkeley National Laboratory. Available at https://www4.eere.energy.gov/seeaction/system/files/documents/smart_meters.pdf.

Wu, K. et al. (2013): "A big data approach to analyzing market volatility." *Algorithmic Finance*. Vol. 2, No. 3, pp. 241–267.

Wu, L. et al. (2016): "Towards real-time detection and tracking of spatio-temporal features: Blob-filaments in fusion plasma. *IEEE Transactions on Big Data*, Vol. 2, No. 3, pp. 262–275.

Yan, J. et al. (2009): "How much can behavioral targeting help online advertising?" Proceedings of the 18th international conference on world wide web. ACM. pp. 261–270.

Yelick, K., et al. (2011): "The Magellan report on cloud computing for science." U.S. Department of Energy, Office of Science.

Zeff, R.L. and B. Aronson (1999): *Advertising on the Internet*. John Wiley & Sons.

Index

Page numbers followed by *f* or *t* refer to figure or table, respectively.