

Quantitative Trading

*How to Build Your Own
Algorithmic Trading Business*

Second Edition

ERNEST P. CHAN



Copyright © 2021 by Ernest P. Chan.

Published by arrangement with John Wiley & Sons, Inc..

Used with permission by Echo Point Books & Media, LLC.

For personal use only with purchase of audiobook.

AUDIOBOOK SUPPLEMENT

Chapter 2	1
Table 2.1	2
How to Identify a Strategy That Suits You	3
Table 2.2	3
A Taste for Plausible Strategies and Their Pitfalls.	3
Figure 2.1.	3
Summary	4
References	5
Chapter 3	6
Common Backstelling Platforms	7
Example 3.1: Using MATLAB to Retrieve Yahoo! Finance Data .	7
Example 3.1: Using Python to Retrieve Yahoo! Finance Data . .	8
Example 3.1: Using R to Retrieve Yahoo! Finance Data	9
Finding and Using Historical Databases	10
Table 3.1	10
Example 3.2: Adjusting for Splits and Dividends	10
Example 3.3: An Example of How Survivorship Bias Can Artificially Inflate a Strategy's Performance	13
Performance Measurement	15
Example 3.4: Calculating Sharpe Ratio for Long-Only Versus Market-Neutral Strategies	15
Example 3.5: Calculating Maximum Drawdown and Maximum Drawdown Duration	19
Figure 3.1.	21
Common Backtesting Pitfalls to Avoid	24
Example 3.6: Pair Trading of GLD and GDX	24
Transaction Costs	32
Example 3.7: A Simple Mean-Reverting Model with and without Transaction Costs	32
Strategy Refinement.	37
Example 3.8: A Small Variation on an Existing Strategy	37

Summary	37
References	38
Chapter 4	40
Business Structure: Retail or Proprietary?	40
Box 4.1	40
Table 4.1	42
Summary	43
References	44
Chapter 5	45
What an Automated Trading System Can Do for You	45
Figure 5.1	45
Figure 5.2	46
Summary	47
Chapter 6	49
Optimal Capital Allocation and Leverage	50
Example 6.1: An Interesting Puzzle (or Why Risk Is Bad For You)	50
Example 6.2: Calculating the Optimal Leverage Based on the Kelly Formula	51
Example 6.3: Calculating the Optimal Allocation Using the Kelly Formula	52
Psychological Preparedness	58
Box 6.1	58
Summary	60
Appendix: A Simple Derivation of the Kelly Formula When Return Distribution is Gaussian.	61
References	62
Chapter 7	63
Regime Change and Conditional Parameter Optimization	64
Example 7.1: Conditional Parameter Optimization applied to an ETF trading strategy	64
Figure 7.1	70
Stationarity and Cointegration	71
Figure 7.2	71

Example 7.2: How to Form a Good Cointegrating (and Mean-Reverting) Pair of Stocks	72
Figure 7.3.	76
Figure 7.4.	77
Example 7.3: Testing to Cointegration versus Correlation Properties between KO and PEP.	77
Factor Models	82
Example 7.4: Principal Component Analysis as an Example of the Factor Model	82
What is Your Exit Strategy?	88
Example 7.5: Calculation of the Half-Life of a Mean-Reverting Time Series.	88
Seasonal Trading Strategies	90
Example 7.6: Backtesting the January Effect.	90
Example 7.7: Backtesting a Year-on-Year Seasonal Trending Strategy	94
A Seasonal Trade in Gasoline Futures.	98
A Seasonal Trade in Natural Gas Futures	99
Summary	100
References	102
<i>Chapter 8.</i>	103
References	103
Appendix	104
Bibliography	110
About the Author	114

CHAPTER 2

Fishing for Ideas

*Where Can We Find Good
Strategies?*

TABLE 2.1 Sources of Trading Ideas

Type	URL
Academic	
Business schools' finance professors' websites	www.hbs.edu/research/research.html
Social Science Research Network	www.ssrn.com
National Bureau of Economic Research	www.nber.org
Business schools' quantitative finance seminars	www.ieur.columbia.edu/seminars/financialengineering
Quantpedia (aggregator of all academic papers on quantitative trading strategies!)	quantpedia.com
Financial blogs and podcasts	
Flirting with Models	www.thinknewfound.com
Mutiny Fund	mutinyfund.com/podcast/
Chat with Traders	chatwithtraders.com
Eran Raviv	eranraviv.com
Sibyl/Godot Finance	godotfinance.com
Party at the Moontower	moontowermeta.com
My own!	epchan.blogspot.com
Trader forums	
Elite Trader	www.Elitetrader.com
Wealth-Lab	www.wealth-lab.com
Twitter	
Benn Eifert	@benneifert
Corey Hoffstein	@choffstein
Quantocracy (retweet of new articles)	@Quantocracy
Mike Harris	@mikeharrisNY
Euan Sinclair	@sinclaireuan
My own!	@chanep
Newspaper and magazines	
<i>Stocks, Futures and Options</i> magazine	www.sfomag.com

HOW TO IDENTIFY A STRATEGY THAT SUITS YOU

TABLE 2.2 How Capital Availability Affects Your Many Choices

Low Capital	High Capital
Proprietary trading firm's membership	Retail brokerage account
Futures, currencies, options	Everything, including stocks
Intraday	Both intra- and interday (overnight)
Directional	Directional or market neutral
Small stock universe for intraday trading	Large stock universe for intraday trading
Daily historical data with survivorship bias	High-frequency historical data, survivorship bias-free
Low-coverage or delayed news source	High-coverage, real-time news source
No historical news database	Survivorship bias-free historical news database
No historical fundamental data on stocks	Survivorship bias-free historical fundamental data on stocks

A TASTE FOR PLAUSIBLE STRATEGIES AND THEIR PITFALLS

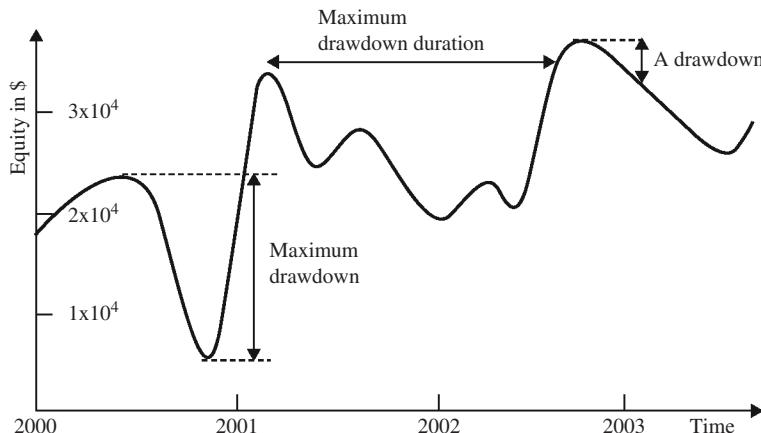


FIGURE 2.1 Drawdown, maximum drawdown, and maximum drawdown duration.

SUMMARY

Finding prospective quantitative trading strategies is not difficult. There are:

- Business school and other economic research websites.
- Financial websites and blogs focusing on the retail investors.
- Trader forums where you can exchange ideas with fellow traders.
- Twitter!

After you have done a sufficient amount of Net surfing or scrolling through your Twitter feed, you will find a number of promising trading strategies. Whittle them down to just a handful, based on your personal circumstances and requirements, and by applying the screening criteria (more accurately described as healthy skepticism) that I listed earlier:

- How much time do you have for babysitting your trading programs?
- How good a programmer are you?
- How much capital do you have?
- Is your goal to earn steady monthly income or to strive for a large, long-term capital gain?

Even before doing an in-depth backtest of the strategy, you can quickly filter out some unsuitable strategies if they fail one or more of these tests:

- Does it outperform a benchmark?
- Does it have a high enough Sharpe ratio?
- Does it have a small enough drawdown and short enough drawdown duration?
- Does the backtest suffer from survivorship bias?
- Does the strategy lose steam in recent years compared to its earlier years?
- Does the strategy have its own “niche” that protects it from intense competition from large institutional money managers?

After making all these quick judgments, you are now ready to proceed to the next chapter, which is to rigorously backtest the strategy yourself to make sure that it does what it is advertised to do.

REFERENCES

- Duhigg, Charles. 2006. “Street Scene; A Smarter Computer to Pick Stock.” *New York Times*, November 24.
- Sharpe, William. 1994. “The Sharpe Ratio.” *The Journal of Portfolio Management*, Fall. Available at: www.stanford.edu/~wfsharpe/art/sr/sr.htm.

CHAPTER 3

Backtesting

COMMON BACKTESTING PLATFORMS

Example 3.1: Using MATLAB to Retrieve Yahoo! Finance Data

MATLAB is not only useful for numerical computations but also for text parsing. Following is an example of using MATLAB to retrieve a stock's historical price information from Yahoo! Finance. First, copy the file `getMarketDataViaYahoo.m` from github.com/Lenskiy/market-data-functions and save it to your local folder. Then save the following code as `example3_1.m` to your local folder, too:

```
% Example 3.1: Download data from Yahoo
initDate = '1-Sep-2020';
symbol = 'AAPL';
aaplusd_yahoo_raw = getMarketDataViaYahoo(symbol,
    initDate);
aaplusd_yahoo= timeseries([aaplusd_yahoo_raw.Close,
    aaplusd_yahoo_raw.High, aaplusd_yahoo_raw.Low],
    datestr(aaplusd_yahoo_raw(:,1).Date));
aaplusd_yahoo.DataInfo.Units = 'USD';
aaplusd_yahoo.Name = symbol;
aaplusd_yahoo.TimeInfo.Format = "dd-mm-yyyy";

figure,
plot(aaplusd_yahoo);
legend({'Close', 'High', 'Low'}, 'Location',
    'northwest');
disp(aaplusd_yahoo_raw.Close)
```

This program file `example3_1.m` is available for download from epchan.com/book, with “sharperatio” as both username and password. You can easily modify this code to download as many tickers as you like. This program requires the `getMarketDataViaYahoo` function to work. That `.m` file can be downloaded from www.mathworks.com/matlabcentral/fileexchange/68361-yahoo-finance-and-quandl-data-downloader. (You should save it in the same folder as `example3_1.m`.)

Example 3.1: Using Python to Retrieve Yahoo! Finance Data

Following is an example of using Python to retrieve a stock's historical price information from Yahoo! Finance. First, run pip install pandas_datareader on your Anaconda prompt. Then save the following code as example3_1.py to your local folder:

```
% Example 3.1: Download data from Yahoo
from pandas_datareader import data as pdr
def test_yfinance():
    for symbol in ['AAPL', 'MSFT', 'VFINX', 'BTC-USD']:
        print(">>", symbol, end=' ... ')
        data = pdr.get_data_yahoo(symbol, st
art='2020-09-25', end='2020-10-02')
        print(data)
if __name__ == "__main__":
    test_yfinance()
```

This program file example3_1.py is available for download from epchan.com/book, with “sharperatio” as both username and password. You can easily modify this code to download as many tickers as you like.

Example 3.1: Using R to Retrieve Yahoo! Finance Data

Following is an example of using R to retrieve a stock's historical price information from Yahoo! Finance. This program file example3_1.R is available for download from epchan.com/book, with "sharperatio" as both username and password. Just save it to your local folder and run in RStudio. (Installation of required packages is included. Once they are installed, you can comment out the install.packages lines.)

```
# tidyverse contains the packages tidyr, ggplot2, dplyr,
# readr, purrr and tibble
install.packages("tidyverse")
install.packages("lubridate")
install.packages("readxl")
install.packages("highcharter")
install.packages("tidyquant")
install.packages("timetk")
install.packages("tibbletime")
install.packages("quantmod")
install.packages("PerformanceAnalytics")
install.packages("scales")
library(tidyverse)
library(lubridate)
library(readxl)
library(highcharter)
library(tidyquant)
library(timetk)
library(tibbletime)
library(quantmod)
library(PerformanceAnalytics)
library(scales)
symbols <- c("SPY", "EFA", "IJS", "EEM", "AGG")
prices <-
  getSymbols(symbols,
             src = 'yahoo',
             from = "2012-12-31",
             to = "2017-12-31",
             auto.assign = TRUE,
             warnings = FALSE)
print(SPY)
```

You can easily modify this code to download as many tickers as you like.

FINDING AND USING HISTORICAL DATABASES

TABLE 3.1 Historical Databases for Backtesting

Source	Pros	Cons
Daily Stock Data		
finance.yahoo.com	Free. Split/dividend adjusted.	Has survivorship bias. Can download only one symbol at a time.
Sharadar.com	Survivorship bias free.	
Algoseek.com	Rent, don't buy, tick data! Enriched with identifiers and tags.	Moderately priced.
CSIdata.com	Low cost. Source of Yahoo! and Google's historical data. Software enables download of multiple symbols.	Has survivorship bias, though delisted stocks' history can be purchased.
CRSP.com	Survivorship bias free.	Expensive. Updated only once a month.
Daily Futures Data		
Algoseek.com	(See above.)	
CSIdata.com	(See above.)	
Intraday Stock / Futures Data		
Algoseek.com	(See above.)	
Tickdata.com	Institutional quality.	Expensive.
Daily / Intraday Forex Data		
Interactive Brokers	Free if you have an account.	

Example 3.2: Adjusting for Splits and Dividends

Here we look at IGE, an ETF that has had both splits and dividends in its history. It had a 2:1 split on June 9, 2005 (the ex-date). Let's look at the unadjusted prices around that date (you can download the historical prices of IGE from Yahoo! Finance into an Excel spreadsheet):

Date	Open	High	Low	Close
6/10/2005	73.98	74.08	73.31	74
6/9/2005	72.45	73.74	72.23	73.74
6/8/2005	144.13	146.44	143.75	144.48
6/7/2005	145	146.07	144.11	144.11

We need to adjust the prices prior to 6/9/2005 due to this split. This is easy: $N = 2$ here, and all we need to do is to multiply those prices by $\frac{1}{2}$. The following table shows the adjusted prices:

Date	Open	High	Low	Close
6/10/2005	73.98	74.08	73.31	74
6/9/2005	72.45	73.74	72.23	73.74
6/8/2005	72.065	73.22	71.875	72.24
6/7/2005	72.5	73.035	72.055	72.055

Now, the astute reader will notice that the adjusted close prices here do not match the adjusted close prices displayed in the Yahoo! Finance table. The reason for this is that there have been dividends distributed after 6/9/2005, so the Yahoo! prices have been adjusted for all those as well. Since each adjustment is a multiplier, the aggregate adjustment is just the product of all the individual multipliers. Here are the dividends from 6/9/2005 to November 2007, together with the unadjusted closing prices of the previous trading days and the resulting individual multipliers:

Ex-Date	Dividend	Prev Close	Multiplier
9/26/2007	0.177	128.08	0.998618
6/29/2007	0.3	119.44	0.997488
12/21/2006	0.322	102.61	0.996862
9/27/2006	0.258	91.53	0.997181
6/23/2006	0.32	92.2	0.996529
3/27/2006	0.253	94.79	0.997331
12/23/2005	0.236	89.87	0.997374
9/26/2005	0.184	89	0.997933
6/21/2005	0.217	77.9	0.997214

(Check out the multipliers yourself on Excel using the formula I gave above to see if they agree with my values here.) So the aggregate multiplier for the dividends is simply $0.998618 \times 0.997488 \times \dots \times 0.997214 = 0.976773$. This multiplier should be applied to all the unadjusted prices on or after 6/9/2005. The aggregate multiplier for the dividends and the split is $0.976773 \times 0.5 = 0.488386$, which should be applied to all the unadjusted prices before 6/9/2005. So let's look at the resulting adjusted prices after applying these multipliers:

Date	Open	High	Low	Close
6/10/2005	72.26163	72.35931	71.6072	72.28117
6/9/2005	70.76717	72.02721	70.55228	72.02721
6/8/2005	70.39111	71.51929	70.20553	70.56205
6/7/2005	70.81601	71.33858	70.38135	70.38135

Now compare with the table from Yahoo! around November 1, 2007.

Date	Open	High	Low	Close	Volume	Adj Close
6/10/2005	73.98	74.08	73.31	74	179300	72.28
6/9/2005	72.45	73.74	72.23	73.74	853200	72.03
6/8/2005	144.13	146.44	143.75	144.48	109600	70.56
6/7/2005	145	146.07	144.11	144.11	58000	70.38

You can see that the adjusted closing prices from our calculations and from Yahoo! are the same (after rounding to two decimal places). But, of course, when you are reading this, IGE will likely have distributed more dividends and may have even split further, so your Yahoo! table won't look like the one above. It is a good exercise to check that you can make further adjustments based on those dividends and splits that result in the same adjusted prices as your current Yahoo! table.

Example 3.3: An Example of How Survivorship Bias Can Artificially Inflate a Strategy's Performance

Here is a toy “buy low-priced stocks” strategy. (*Warning:* This toy strategy is hazardous to your financial health!) Let’s say from a universe of the 1,000 largest stocks (based on market capitalization), we pick 10 that have the lowest closing prices at the beginning of the year and hold them (with equal initial capital) for one year. Let’s look at what we would have picked if we had a good, survivorship-bias-free database:

SYMBOL	Closing Price on 1/2/2001	Closing Price on 1/2/2002	Terminal Price
ETYS	0.2188	NaN	0.125
MDM	0.3125	0.49	0.49
INTW	0.4063	NaN	0.11
FDHG	0.5	NaN	0.33
OGNC	0.6875	NaN	0.2
MPLX	0.7188	NaN	0.8
GTS	0.75	NaN	0.35
BUYX	0.75	NaN	0.17
PSIX	0.75	NaN	0.2188
RTHM	0.8125	NaN	0.3000

All but MDM were delisted sometime between 1/2/2001 and 1/2/2002 (after all, the dot-com bubble was seriously bursting then!). The NaNs indicate those with nonexistent closing prices on 1/2/2002. The Terminal Price column indicates the last prices at which the stocks were traded on or before 1/2/2002. The total return on this portfolio in that year was -42 percent.

Now, let’s look at what we would have picked if our database had survivorship bias and actually missed all those stocks that were delisted that year. We would then have picked the following list instead:

SYMBOL	Closing Price on 1/2/2001	Closing Price on 1/2/2002
MDM	0.3125	0.49
ENGA	0.8438	0.44
NEOF	0.875	27.9

SYMBOL	Closing Price on 1/2/2001	Closing Price on 1/2/2002
ENP	0.875	0.05
MVL	0.9583	2.5
URBN	1.0156	3.0688
FNV	1.0625	0.81
APT	1.125	0.88
FLIR	1.2813	9.475
RAZF	1.3438	0.25

Notice that since we select only those stocks that “survived” until at least 1/2/2002, they all have closing prices on that day. The total return on this portfolio was 388 percent!

In this example, -42 percent was the actual return a trader would experience following this strategy, whereas 388 percent is a fictitious return that was due to survivorship bias in our database.

PERFORMANCE MEASUREMENT

Example 3.4: Calculating Sharpe Ratio for Long-Only Versus Market-Neutral Strategies

Let's calculate the Sharpe ratio of a trivial long-only strategy for IGE: buying and holding a share since the close of November 26, 2001, and selling it at close of November 14, 2007. Assume the average risk-free rate during this period is 4 percent per annum in this example. You can download the daily prices from Yahoo! Finance, specifying the date range desired, and store them as an Excel file (or the comma-separated file for use in the R code), which you can call IGE.xls. The next steps can be done in either Excel or MATLAB:

Using Excel

1. The file should have columns A–G already from the download.
2. Sort all the columns in ascending order of Date (use the Data-Sort function, choose the “Expand the selection” radio button, and choose the “Ascending” as well as the “My data has Header row” radio buttons).
3. In cell H3, type “=(G3-G2)/G2”. This is the daily return.

4. Double-clicking the little black dot at the lower right corner of the cell H3 will populate the entire column H with daily returns of IGE.
5. For clarity, you can type “Dailyret” in the header cell H1.
6. In cell I3, type “=H3-0.04/252,” which is the excess daily return, assuming a 4 percent per annum risk-free rate and 252 trading days in a year.
7. Double-clicking the little black dot at the lower right corner of cell I3 will populate the entire column I with excess daily returns.
8. For clarity, type “Excess Dailyret” in the header cell I1.
9. In cell I1506 (the last row in the next column), type “=SQRT(252)*AVERAGE(I3:I1506)/STDEV(I3:I1506)”.
10. The number displayed in cell I1506, which should be “0.789317538,” is the Sharpe ratio of this buy-and-hold strategy.

The finished spreadsheet is available at my website at epchan.com/book/example3_4.xls.

Using MATLAB

```
% make sure previously defined variables are erased.
clear;
% read a spreadsheet named "IGE.xls" into MATLAB.
[num, txt]=xlsread('IGE');
% the first column (starting from the second row)
% contains the trading days in format mm/dd/yyyy.
tday=txt(2:end, 1);
% convert the format into yyyyymmdd.
tday=datestr(datenum(tday, `mm/dd/yyyy'), `yyyyymmdd');
% convert the date strings first into cell arrays and
% then into numeric format.
tday=str2double(cellstr(tday));
% the last column contains the adjusted close prices.
cls=num(:, end);
% sort tday into ascending order.
[tday sortIndex]=sort(tday, `ascend');
% sort cls into ascending order of dates.
cls=cls(sortIndex);
% daily returns
dailyret=(cls(2:end)-cls(1:end-1))./cls(1:end-1);
% excess daily returns assuming risk-free rate of 4%
% per annum and 252 trading days in a year
excessRet=dailyret - 0.04/252;
% the output should be 0.7893
sharpeRatio=sqrt(252)*mean(excessRet)/std(excessRet)
```

This MATLAB code is also available for download at my website (epchan.com/book/example3_4.m).

Using Python

```
This is the code that can be run in a Jupyter notebook. The code is available for download at epchan.com/book/example3_4.ipynb
Calculating Sharpe Ratio for Long-Only Vs Market Neutral Strategies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
First part of example
df=pd.read_excel('IGE.xls')
df.sort_values(by='Date', inplace=True)
dailyret=df.loc[:, 'Adj Close'].pct_change() # daily
    returns
excessRet=dailyret-0.04/252 # excess daily returns =
    strategy returns - financing cost, assuming risk-free
    rate of
sharpeRatio=np.sqrt(252)*np.mean(excessRet) /
    np.std(excessRet)
sharpeRatio
0.789580250130583
```

Using R

```
The code is available for download at epchan.com/book/example3_4.R
library('zoo')
data1 <- read.csv("IGE.csv") # Comma delimited
data_sort <- data1[order(as.Date(data1[,1],
    '%m/%d/%Y')),] # sort in ascending order of dates (1st
    column of data)
adjcls <- data_sort[,ncol('Adj.Close')]
adjcls[ is.nan(adjcls) ] <- NA
mycls <- na.fill(adjcls, type="locf", nan=NA, fill=NA)
dailyret <- diff(mycls)/mycls[1:(length(mycls)-1)]
excessRet <- dailyret - 0.04/252
sharpeRatio <- sqrt(252)*mean(excessRet, na.rm = TRUE) /
    sd(excessRet, na.rm = TRUE)
sharpeRatio
```

Now let's calculate the Sharpe ratio of a long-short market-neutral strategy. In fact, it is a very trivial twist of the buy-and-hold strategy above: at the time we bought IGE, let's suppose we just shorted an equal dollar amount of Standard & Poor's depositary receipts (SPY) as a hedge, and closed both positions at the same time in November 2007. You can also

download SPY from Yahoo! Finance and store it in a file SPY.xls. You can go through very similar steps as previously covered in both Excel and MATLAB, and I will leave it as an exercise for the reader to perform the exact steps:

Using Excel

1. Sort the columns in SPY.xls in ascending order of date.
2. Copy column G (Adj Close) in SPY.xls and paste it onto column J of IGE.xls.
3. Check that column J has the same number of rows as columns A–I.
If not, you have a different set of dates—make sure you download the matching date range from Yahoo!.
4. Perform the same steps as previously covered to calculate the daily returns in column K.
5. For clarity, type “dailyretSPY” as the header in column K.
6. In column L, compute the net returns for the hedged strategy as the difference between column H and K divided by 2. (Divide by 2 because we now have twice the capital.)
7. In cell L1506, compute the Sharpe ratio of this hedged strategy. You should get “0.783681.”

Using MATLAB

```
% Assume this is a continuation of the above MATLAB
% code.
% Insert your own code here to retrieve data from
% SPY.xls just as done previously.
% Name the array that contains the daily returns of
% SPY "dailyretSPY".
% net daily returns
% (divide by 2 because we now have twice as much capital.)
netRet=(dailyret - dailyretSPY)/2;
% the output should be 0.7837.
sharpeRatio=sqrt(252)*mean(netRet)/std(netRet)
```

Using PYTHON

```
Second part of example
df2=pd.read_excel('SPY.xls')
df=pd.merge(df, df2, on='Date', suffixes=('_IGE', '_SPY'))
df['Date']=pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
dailyret=df[['Adj Close_IGE', 'Adj Close_SPY']].pct_
change() # daily returns
```

```

dailyret.rename(columns={"Adj Close_IGE": "IGE", "Adj
Close_SPY": "SPY"}, inplace=True)
netRet=(dailyret['IGE']-dailyret['SPY'])/2
sharpeRatio=np.sqrt(252)*np.mean(netRet)/np.std(netRet)
sharpeRatio
0.7839419359681374

```

Using R

This code can be downloaded as example3_4.R:

```

# 2nd part of example 3.4
data2 <- read.delim("SPY.txt") # Tab-delimited
data_sort <- data2[order(as.Date(data2[,1],
      '%m/%d/%Y')),] # sort in ascending order of dates (1st
      column of data)
adjcls <- data_sort[,ncol('Adj.Close')]
adjcls[ is.nan(adjcls) ] <- NA
mycls <- na.fill(adjcls, type="locf", nan=NA, fill=NA)
dailyretSPY <- diff(mycls)/mycls[1:(length(mycls)-1)]
excessRet <- (dailyret - dailyretSPY)/2
informationRatio <- sqrt(252)*mean(excessRet, na.rm =
      TRUE)/sd(excessRet, na.rm = TRUE)
informationRatio

```

Example 3.5: Calculating Maximum Drawdown and Maximum Drawdown Duration

We shall continue the preceding long-short market-neutral example in order to illustrate the calculation of maximum drawdown and maximum drawdown duration. The first step in this calculation is to calculate the “high watermark” at the close of each day, which is the maximum cumulative return of the strategy up to that time. (Using the cumulative return curve to calculate high watermark and drawdown is equivalent to using the equity curve, since equity is nothing more than initial investment times 1 plus the cumulative return.) From the high watermark, we can calculate the drawdown, the maximum drawdown, and maximum drawdown duration:

Using Excel

1. In cell M3, type “=L3”.
2. In cell M4, type “=(1+M3)*(1+L4)-1”. This is the cumulative compounded return of the strategy up to that day. Populate the entire column M with the cumulative compounded returns of the strategy and erase the last cell of the column. Name this column Cumret.

3. In cell N3, type “=M3”.
4. In cell N4, type “=MAX(N3, M4)”. This is the high watermark up to that day. Populate the entire column N with the running high watermark of the strategy and erase the last cell of the column. Name this column High watermark.
5. In cell O3, type “=(1+M3)/(1+N3)-1”. This is the drawdown at that day’s close. Populate the entire column O with the drawdowns of the strategy.
6. In cell O1506, type “=MAX(O3:O1505)”. This is the maximum drawdown of the strategy. It should have a value of about 0.1053, that is, a maximum drawdown of 10.53 percent.
7. In cell P3, type “=IF(O3=0, 0, P2+1)”. This is the duration of the current drawdown. Populate the entire column R with the drawdown durations of the strategy and erase the last cell of the column.
8. In cell P1506, type “=MAX(P3:P1505)”. This is the maximum drawdown duration of the strategy. It should have a value of 497, that is, a maximum drawdown duration of 497 trading days.

Using MATLAB

```
% Assume this is a continuation of the above MATLAB
% code.
% cumulative compounded returns
cumret=cumprod(1+netRet)-1;plot(cumret);
[maxDrawdown maxDrawdownDuration]=...
calculateMaxDD(cumret);
% maximum drawdown. Output should be -0.0953
maxDrawdown
% maximum drawdown duration. Output should be 497.
maxDrawdownDuration
```

Notice the code fragment above calls a function “calculateMaxDrawdown,” which I display below.

```
function [maxDD maxDDD]=calculateMaxDD(cumret)
% [maxDD maxDDD]=calculateMaxDD(cumret)
% calculation of maximum drawdown and maximum drawdown
% duration based on cumulative compounded returns.
% initialize high watermarks to zero.
highwatermark=zeros(size(cumret));
% initialize drawdowns to zero.
drawdown=zeros(size(cumret));
% initialize drawdown duration to zero.
drawdownduration=zeros(size(cumret));
```

```

for t=2:length(cumret)
    highwatermark(t)=...
max(highwatermark(t-1), cumret(t));
% drawdown on each day
drawdown(t)=(1+ cumret(t))/(1+ highwatermark(t))-1;
if (drawdown(t)==0)
    drawdownduration(t)=0;
else
    drawdownduration(t)=drawdownduration(t-1)+1;
end
end
maxDD=max(drawdown); % maximum drawdown
% maximum drawdown duration
maxDDD=max(drawdownduration);

```

The file that contains this function is available as epchan.com/book/calculateMaxDD.m. You can see in Figure 3.1 where the maximum drawdown and maximum drawdown duration occurred in this plot of the cumulative returns.

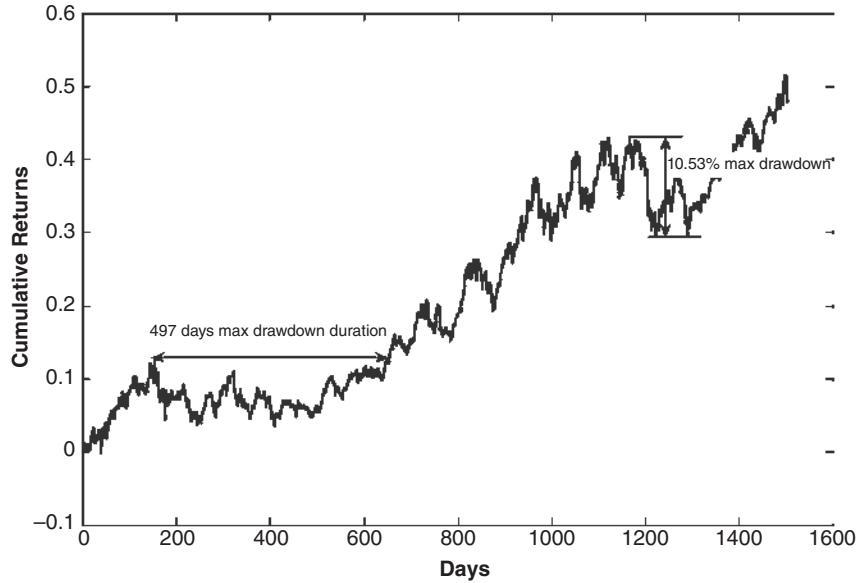


FIGURE 3.1 Maximum drawdown and maximum drawdown duration for Example 3.4.

Using Python

You will need to download the following calculateMaxDD.py code to your folder first:

```
import numpy as np
def calculateMaxDD(cumret):
# =====
# calculation of maximum drawdown and maximum drawdown
duration based on
# cumulative COMPOUNDED returns. cumret must be a com-
pounded cumulative return.
# i is the index of the day with maxDD.
# =====
    highwatermark=np.zeros(cumret.shape)
    drawdown=np.zeros(cumret.shape)
    drawdownduration=np.zeros(cumret.shape)
    for t in np.arange(1, cumret.shape[0]):
        highwatermark[t]=np.maximum(highwatermark[t-1],
        cumret[t])
        drawdown[t]=(1+cumret[t])/(1+highwatermark[t])-1
        if drawdown[t]==0:
            drawdownduration[t]=0
        else:
            drawdownduration[t]=drawdownduration[t-1]+1

    maxDD, i=np.min(drawdown), np.argmin(drawdown) #
    drawdown < 0 always
    maxDDD=np.max(drawdownduration)
    return maxDD, maxDDD, i
```

Then you can run the rest of Jupyter notebook:

```
cumret=np.cumprod(1+netRet)-1
plt.plot(cumret)
from calculateMaxDD import calculateMaxDD
maxDrawdown, maxDrawdownDuration, startDrawdownDay=calcu-
lateMaxDD(cumret.values)
maxDrawdown
-0.09529268047208683
maxDrawdownDuration
497.0
startDrawdownDay
1223
```

Using R

You first need to download the function calculateMaxDD.R to your folder.

```
# calculateMaxDD.R
calculateMaxDD <- function(cumret) {
  # Assume compounded cumulative return as input
  highwatermark <- rep(0, length(cumret))
  highwatermark
  drawdown <- rep(0, length(cumret))
  drawdownduration <- rep(0, length(cumret))

  for (t in 2:length(cumret)) {
    highwatermark[t] <- max(highwatermark[t-1],
    cumret[t])
    drawdown[t] <- (1+cumret[t])/(1+highwatermark[t])-1
    if (drawdown[t]==0) {
      drawdownduration[t]=0
    } else {
      drawdownduration[t]=drawdownduration[t-1]+1
    }
  }

  maxDD <- min(drawdown)
  maxDDD <- max(drawdownduration)
  return(c(maxDD, maxDDD))
}
```

Then you can run the following code, which is the third part of example3_4.R:

```
# 3rd part of example 3.4
source('calculateMaxDD.R')
cumret <- cumprod(1+excessRet[!is.nan(excessRet)])-1
plot(cumret)
output <- calculateMaxDD(cumret)
maxDD <- output[1]
maxDD
maxDDD <- output[2]
maxDDD
```

COMMON BACKTESTING PITFALLS TO AVOID

Example 3.6: Pair Trading of GLD and GDX

This example will illustrate how to separate the data into a training set and a test set. We will backtest a pair-trading strategy and optimize its parameters on the training set and look at the effect on the test set.

GLD versus GDX is a good candidate for pair trading because GLD reflects the spot price of gold, and GDX is a basket of gold-mining stocks. It makes intuitive sense that their prices should move in tandem. I have discussed this pair of ETFs extensively on my blog in connection with cointegration analysis (see, e.g., Chan, 2006b). Here, however, I will defer until Chapter 7 the cointegration analysis on the training set, which demonstrates that the spread formed by long GLD and short GDX is mean reverting. Instead, we will perform a regression analysis on the training set to determine the hedge ratio between GLD and GDX, and then define entry and exit thresholds for a pair-trading strategy. We will see how optimizing these thresholds on the training set changes the performance on the test set. (This program is available as on epchan.com/book/example3_6.m. The data files are available as GDX.xls and GLD.xls. This program uses a lag1 function, which will lag the time series by one time period. It is included at epchan.com/book as well. It also uses a function "ols" for linear regression, which is part of a free package downloaded from spatial-econometrics.com.)

Using MATLAB

```
clear; % make sure previously defined variables are
       erased.

[num, txt]=xlsread('GLD'); % read a spreadsheet named
                           "GLD.xls" into MATLAB.

tday1=txt(2:end, 1); % the first column (starting from
                     the second row) is the trading days in format mm/dd/
                     YYYY.

tday1=datestr(datenum(tday1, 'mm/dd/yyyy'), 'yyyymmdd');
% convert the format into yyyymmdd 20041118-20071130.
```

```

tday1=str2double(cellstr(tday1)); % convert the date
    strings first into cell arrays and then into numeric
    format.

adjcls1=num(:, end); % the last column contains the
    adjusted close prices.

[num, txt]=xlsread('GDX'); % read a spreadsheet named
    "GDX.xls" into MATLAB.

tday2=txt(2:end, 1); % the first column (starting from
    the second row) is the trading days in format mm/dd/
    YYYY.

tday2=datestr(datenum(tday2, 'mm/dd/yyyy'), 'yyyyymmdd');
    % convert the format into yyyyymmdd 20060523-20071130.

tday2=str2double(cellstr(tday2)); % convert the date
    strings first into cell arrays and then into numeric
    format.

adjcls2=num(:, end); % the last column contains the
    adjusted close prices.

[tday, idx1, idx2]=intersect(tday1, tday2); % find the
    intersection of the two data sets, and sort them in
    ascending order

cl1=adjcls1(idx1);

cl2=adjcls2(idx2);

trainset=1:252; % define indices for training set

testset=trainset(end)+1:length(tday); % define indices
    for test set

% determines the hedge ratio on the trainset
results=ols(cl1(trainset), cl2(trainset)); % use regres-
    sion function
hedgeRatio=results.beta; % 1.6368

spread=cl1-hedgeRatio*cl2; % spread = GLD -
    hedgeRatio*GDX

plot(spread(trainset));

```

```

figure;

plot(spread(testset));

figure;

spreadMean=mean(spread(trainset)); % mean of spread on
trainset

spreadStd=std(spread(trainset)); % standard deviation of
spread on trainset

zscore=(spread - spreadMean)./spreadStd; % z-score of
spread

longs=zscore<=-2; % buy spread when its value drops
below 2 standard deviations.

shorts=zscore>=2; % short spread when its value rises
above 2 standard deviations.

exitLongs=zscore>=-1; % exit any spread position when
its value is within 1 standard deviation of its mean.
exitShorts=zscore<=1; % exit any spread position when
its value is within 1 standard deviation of its mean.

positionsL=zeros(length(tday), 2); % initialize long
positions array
positionsS=zeros(length(tday), 2); % initialize long
positions array

positionsS(shorts, :)=repmat([-1 1],
[length(find(shorts)) 1]); % long entries

positionsL(longs, :)=repmat([1 -1],
[length(find(longs)) 1]); % short entries

positionsL(exitLongs, :)=zeros(length(find(exitLongs)),
2); % exit positions
positionsS(exitShorts, :)=zeros(length(find(exitSho
rts)), 2); % exit positions

positions=positionsL+positionsS;

```

```

positions=fillMissingData(positions); % ensure existing
    positions are carried forward unless there is an exit
    signal

cl=[cl1 cl2]; % combine the 2 price series

dailyret=(cl - lag1(cl))./lag1(cl);

pn1=sum(lag1(positions).*dailyret, 2);

sharpeTrainset=sqrt(252)*mean(pn1(trainset(2:end)))./
    std(pn1(trainset(2:end))) % the Sharpe ratio on the
    training set should be about 2.3

sharpeTestset=sqrt(252)*mean(pn1(testset))./
    std(pn1(testset)) % the Sharpe ratio on the test set
    should be about 1.5

% sharpeTrainset =
%
%      2.0822
%
%
% sharpeTestset =
%
%      1.4887

plot(cumsum(pn1(testset)));

save example3_6_positions positions; % save positions
file for checking look-ahead bias.

In file lag1.m:
function y=lag1(x)
% y=lag(x)
if (isnumeric(x))
% populate the first entry with NaN
y=[NaN(1,size(x,2));x(1:end-1, :)];elseif (ischar(x))
% populate the first entry with "
y=[repmat(", [1 size(x,2)]);x(1:end-1, :)];else
error(`Can only be numeric or char array');
end

```

So this pair-trading strategy has excellent Sharpe ratios on both the training set and the test set. Therefore, this strategy can be considered free of data-snooping bias. However, there may be room for improvement. Let's

see what happens if we change the entry thresholds to 1 standard deviation and exit threshold to 0.5 standard deviation. In this case, the Sharpe ratio on the training set increases to 2.9 and the Sharpe ratio on the test set increases to 3.0. So, clearly, this set of thresholds is better.

Often, however, optimizing the parameters on the training set may decrease the performance on the test set. In this situation, you should choose a set of parameters that result in good (but may not be the best) performance on both training and test sets.

I have not incorporated transaction costs (which I discuss in the next section) into this analysis. You can try to add that as an exercise. Since this strategy doesn't trade very frequently, transaction costs do not have a big impact on the resulting Sharpe ratio.

To see why this strategy works, just take a look at Figure 7.4 of the spread, which I will discuss in connection with stationarity and cointegration in Chapter 2. You can see that the spread behaves in a highly mean-reverting manner. Hence, buying low and selling high over and over again works well here.

One last check, though, that we should perform before calling this a success: We need to check for any look-ahead bias in the backtest program. Add the following code fragment to the previous MATLAB code after the line “`cl2=adjcls2(idx2);`”

```
% number of most recent trading days to cut off  
cutoff=60;% remove the last cutoff number of days.  
tday(end-cutoff+1:end, :)=[];  
cl1(end-cutoff+1:end, :)=[];  
cl2(end-cutoff+1:end, :)=[];
```

Add the following code fragment to the very end of the previous MATLAB program, replacing the line “`save example3_6_positions positions`.”

```
% step two of look-forward-bias check  
oldoutput=load('example3_6_positions');  
oldoutput.positions(end-cutoff+1:end, :)=[];  
if (any(positions~=oldoutput.positions))  
fprintf(1, 'Program has look-forward-bias!\n');  
end
```

Save this new program into file “`example3_6_1.m`” and run it. You will find that the sentence “Program has look-forward-bias” is not printed out—this indicates that our algorithm passed our test.

Using Python

You can download the Python code in the Jupyter notebook `example3_6.ipynb`.

```

Pair Trading of GLD and GDX
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
df1=pd.read_excel('GLD.xls')
df2=pd.read_excel('GDX.xls')
df=pd.merge(df1, df2, on='Date', suffixes=('_GLD', '_GDX'))
df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
trainset=np.arange(0, 252)
testset=np.arange(trainset.shape[0], df.shape[0])

Determine hedge ratio on trainset
model=sm.OLS(df.loc[:, 'Adj Close_GLD'].iloc[trainset],
               df.loc[:, 'Adj Close_GDX'].iloc[trainset])
results=model.fit()
hedgeRatio=results.params
hedgeRatio
Adj Close_GDX    1.631009
dtype: float64

spread=GLD - hedgeRatio*GDX
spread=df.loc[:, 'Adj Close_GLD']-hedgeRatio[0]*df.
loc[:, 'Adj Close_GDX']
plt.plot(spread.iloc[trainset])
plt.plot(spread.iloc[testset])
spreadMean=np.mean(spread.iloc[trainset])
spreadMean
0.05219623850035999
spreadStd=np.std(spread.iloc[trainset])
spreadStd
1.944860873496509
df['zscore']=(spread-spreadMean)/spreadStd
df['positions_GLD_Long']=0
df['positions_GDX_Long']=0
df['positions_GLD_Short']=0
df['positions_GDX_Short']=0
df.loc[df.zscore>=2, ('positions_GLD_Short', 'positions_
GDX_Short')]=[-1, 1] # Short spread
df.loc[df.zscore<=-2, ('positions_GLD_Long', 'positions_
GDX_Long')]=[1, -1] # Buy spread
df.loc[df.zscore<=1, ('positions_GLD_Short', 'positions_
GDX_Short')]=0 # Exit short spread

```

```

df.loc[df.zscore>=-1, ('positions_GLD_Long', 'positions_GDX_Long')]=0 # Exit long spread
df.fillna(method='ffill', inplace=True) # ensure existing positions are carried forward unless there is an exit signal
positions_Long=df.loc[:, ('positions_GLD_Long', 'positions_GDX_Long')]
positions_Short=df.loc[:, ('positions_GLD_Short', 'positions_GDX_Short')]
positions=np.array(positions_Long)+np.array(positions_Short)
positions=pd.DataFrame(positions)
dailyret=df.loc[:, ('Adj Close_GLD', 'Adj Close_GDX')].pct_change()
pnl=(np.array(positions.shift())*np.array(dailyret)).sum(axis=1)
sharpeTrainset=np.sqrt(252)*np.mean(pnl[trainset[1:]]) / np.std(pnl[trainset[1:]])
sharpeTrainset
1.9182982282569077
sharpeTestset=np.sqrt(252)*np.mean(pnl[testset]) / np.std(pnl[testset])
sharpeTestset
1.494313761833427
plt.plot(np.cumsum(pnl[testset]))
positions.to_pickle('example3_6_positions')

```

Using R

You can download the R code as example3_6.R.

```

library('zoo')
source('calculateReturns.R')
source('calculateMaxDD.R')
source('backshift.R')
data1 <- read.delim("GLD.txt") # Tab-delimited
data_sort1 <- data1[order(as.Date(data1[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates (1st column of data)
tday1 <- as.integer(format(as.Date(data_sort1[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls1 <- data_sort1[,ncol(data_sort1)]
data2 <- read.delim("GDX.txt") # Tab-delimited
data_sort2 <- data2[order(as.Date(data2[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates (1st column of data)

```

```

tday2 <- as.integer(format(as.Date(data_sort2[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls2 <- data_sort2[,ncol(data_sort2)]
# find the intersection of the two data sets
tday <- intersect(tday1, tday2)
adjcls1 <- adjcls1[tday1 %in% tday]
adjcls2 <- adjcls2[tday2 %in% tday]
# define indices for training and test sets
trainset <- 1:252
testset <- length(trainset)+1:length(tday)
# determines the hedge ratio on the trainset
result <- lm(adjcls1 ~ 0 + adjcls2, subset=trainset )
hedgeRatio <- coef(result) # 1.631
spread <- adjcls1-hedgeRatio*adjcls2 # spread = GLD -
  hedgeRatio*GDX
plot(spread)
dev.new()
plot(spread[trainset])
dev.new()
plot(spread[testset])
# mean of spread on trainset
spreadMean <- mean(spread[trainset]) # 0.05219624
# standard deviation of spread on trainset
spreadStd <- sd(spread[trainset]) # 1.948731
zscore <- (spread-spreadMean)/spreadStd
longs <- zscore <= -2 # buy spread when its value drops
  below 2 standard deviations.
shorts <- zscore >= 2 # short spread when its value
  rises above 2 standard deviations.
# exit any spread position when its value is within 1
  standard deviation of its mean.
longExits <- zscore >= -1
shortExits <- zscore <= 1
posL <- matrix(NaN, length(tday), 2) # long positions
poss <- matrix(NaN, length(tday), 2) # short positions
# initialize to 0
posL[1,] <- 0
poss[1,] <- 0
posL[longs, 1] <- 1
posL[longs, 2] <- -1
poss[shorts, 1] <- -1
poss[shorts, 2] <- 1
posL[longExits, 1] <- 0
posL[longExits, 2] <- 0
poss[shortExits, 1] <- 0

```

```

poss[shortExits, 2] <- 0
# ensure existing positions are carried forward unless
# there is an exit signal
posL <- zoo::na.locf(posL)
poss <- zoo::na.locf(poss)
positions <- posL + poss
cl <- cbind(adjcls1, adjcls2) # last row is [385,]
    77.32   46.36
# daily returns of price series
dailyret <- calculateReturns(cl, 1) # last row is
[385,] -0.0122636689 -0.0140365802
pn1 <- rowSums(backshift(1, positions)*dailyret)
sharpeRatioTrainset <- sqrt(252)*mean(pn1[trainset],
  na.rm = TRUE)/sd(pn1[trainset], na.rm = TRUE)
sharpeRatioTrainset # 2.327844
sharpeRatioTestset <- sqrt(252)*mean(pn1[testset], na.rm
  = TRUE)/sd(pn1[testset], na.rm = TRUE)
sharpeRatioTestset # 1.508212
This codes makes use of the function backshift, which
you can download as backshift.R.
backshift <- function(mylag, x) {
  rbind(matrix(NaN, mylag, ncol(x)),
  as.matrix(x[1:(nrow(x)-mylag),]))
}

```

TRANSACTION COSTS

Example 3.7: A Simple Mean-Reverting Model with and without Transaction Costs

Here is a simple mean-reverting model that is attributable to Amir Khandani and Andrew Lo at MIT (available at web.mit.edu/alo/www/Papers/august07.pdf). This strategy is very simple: Buy the stocks with the worst previous one-day returns, and short the ones with the best previous one-day returns. Despite its utter simplicity, this strategy has had great performance since 1995, ignoring transaction costs (it has a Sharpe ratio of 4.47 in 2006).

Our objective here is to find out what would happen to its performance in 2006 if we assume a standard 5-basis-point-per-trade transaction cost. (A *trade* is defined as a buy or a short, not a round-trip transaction.) This example strategy not only allows us to illustrate the impact of transaction costs, it also illustrates the power of MATLAB, Python, or R in backtesting a model that trades multiple securities—in other words, a typical statistical arbitrage model. Backtesting a model with a large number of symbols over multiple years is often too cumbersome to perform in Excel. But even assuming that you have MATLAB, Python, or R at your disposal, there is still the question of how to retrieve historical data for hundreds of symbols, especially survivorship-bias-free data. Here, we will put aside the question of survivorship bias because of the expensive nature of such data and just bear in mind that whatever performance estimates we obtained are upper bounds on the actual performance of the strategy.

Whenever one wants to backtest a stock selection strategy, the first question is always: Which universe of stocks? The typical starting point is the S&P 500 stock universe, which is the most liquid set of stocks available. The current list of stocks in the S&P 500 is available for download at the Standard & Poor's website (www.standardandpoors.com). Since the constituents of this universe change constantly, the list that you download will be different from mine. For ease of comparison, you can find my list saved as epchan.com/book/SP500_20071121.xls. The easiest way to download historical data for all these stocks is to modify `example3_1.m`, `example3_1.py`, or `example3_1.R`, and save only the Date and Close fields for each stock symbol, into a file `SPX_20071123.txt` (downloadable from epchan.com/book).

Next, we can use this historical data set to backtest the mean-reverting strategy without transaction cost:

Using MATLAB

```
clear;
startDate=20060101;
endDate=20061231;
T=readtable('SPX_20071123.txt');
tday=T{:, 1};
cl=T{:, 2:end};
% daily returns
dailyret=(cl-lag1(cl))./lag1(cl);
% equal weighted market index return
marketDailyret=smartmean(dailyret, 2);
% weight of a stock is proportional to the negative
% distance to the market index.
weights=...
```

```

- (dailyret-repmat(marketDailyret,[1
    size(dailyret,2)]))./ repmat(smartsum(isfinite(cl),
    2), ...
[1 size(dailyret, 2)]);
% those stocks that do not have valid prices or
% daily returns are excluded.
weights(~isfinite(cl) | ~isfinite(lag1(cl)))=0;
dailypnl=smartsum(lag1(weights).*dailyret, 2);
% remove pnl outside of our dates of interest
dailypnl(tday < startDate | tday > endDate) = [];
% Sharpe ratio should be about 0.25
sharpe=...
sqrt(252)*smartmean(dailypnl, 1)/smartstd(dailypnl, 1)

```

This file was saved as epchan.com/book/example3_7.m on my website. Notice that the Sharpe ratio in 2006 is only 0.25, not 4.47 as stated by the original authors. The reason for this drastically lower performance is due to the use of the large market capitalization universe of S&P 500 in our backtest. If you read the original paper by the authors, you will find that most of the returns are generated by small and microcap stocks.

In this MATLAB program, I have used three new functions: "smartsum," "smartmean," and "smartstd." They are very similar to the usual "sum," "mean," and "std" functions, except they skip all the NaN entries in the data. These functions are very useful in backtesting because a price series for stocks often starts and stops. These files are all available at epchan.com/book.

```

function y = smartsum(x, dim)
%y = smartsum(x, dim)
%Sum along dimension dim, ignoring NaN.
hasData=isfinite(x);
x(~hasData)=0;
y=sum(x,dim);
y(all(~hasData, dim ))=NaN;
"martmean.m"
function y = smartmean(x, dim)
% y = smartmean(x, dim)
% Mean value along dimension dim, ignoring NaN.
hasData=isfinite(x);
x(~hasData)=0;
y=sum(x,dim)./sum(hasData, dim);
y(all(~hasData, dim ))=NaN; % set y to NaN if all entries
are NaNs.
"smartstd.m"
function y = smartstd(x, dim)
%y = smartstd(x, dim)
% std along dimension dim, ignoring NaN and Inf

```

```

hasData=isfinite(x) ;
x(~hasData)=0;
y=std(x) ;
y(all(~hasData, dim ))=NaN;

```

Now, continuing with our backtest, let's see what happens if we deduct a 5-basis-point transaction cost for every trade.

```

% daily pnl with transaction costs deducted
onewaycost=0.0005; % assume 5 basis points
% remove weights outside of our dates of interest
weights(tday < startDate | tday > endDate, :) = [];
% transaction costs are only incurred when
% the weights change
dailypnlminustcost=...
dailypnl - smartsum(abs(weights-lag1(weights)), 2).*onewaycost;
% Sharpe ratio should be about -3.19
sharpeminustcost=...
sqrt(252)*smartmean(dailypnlminustcost, 1)/...
smartstd(dailypnlminustcost, 1)

```

The strategy is now very unprofitable!

Using Python

This file was saved as epchan.com/book/example3_7.ipynb

```

Simple Mean-Reverting Model with and without Transaction
    Costs
import numpy as np
import pandas as pd
startDate=20060101
endDate=20061231
df=pd.read_table('SPX_20071123.txt')
df['Date']=df['Date'].astype('int')
df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
dailyret=df.pct_change()
marketDailyret=dailyret.mean(axis=1)
weights=-(np.array(dailyret)-np.array(marketDailyret) .
    reshape((dailyret.shape[0], 1)))
wtsum=np.nansum(abs(weights), axis=1)
weights[wtsum==0,]=0
wtsum[wtsum==0]=1
weights=weights/wtsum.reshape((dailyret.shape[0],1))
dailypnl=np.nansum(np.array(pd.DataFrame(weights) .
    shift())*np.array(dailyret), axis=1)

```

```

dailypnl=dailypnl[np.logical_and(df.index >= startDate,
    df.index <= endDate)]
sharpeRatio=np.sqrt(252)*np.mean(dailypnl) /
    np.std(dailypnl)
sharpeRatio
0.957785681010386
With transaction costs
onewaytcost=0.0005
weights=weights[np.logical_and(df.index >= startDate,
    df.index <= endDate)]
dailypnlminustcost=dailypnl - (np.nansum(abs(weights-
    np.array(pd.DataFrame(weights).shift()))),
    axis=1)*onewaytcost)
sharpeRatioMinusTcost=np.sqrt(252)*np.
    mean(dailypnlminustcost)/np.std(dailypnlminustcost)
sharpeRatioMinusTcost
-2.1617433718962276

```

Using R

This file was saved as epchan.com/book/example3_7.R.

```

library('zoo')
source('calculateReturns.R')
source('calculateMaxDD.R')
source('backshift.R')
startDate <- 20060101
endDate   <- 20061231
data1 <- read.delim("SPX_20071123.txt") # Tab-delimited
cl <- data.matrix(data1[, 2:ncol(data1)])
tday <- data.matrix(data1[, 1])
dailyret <- calculateReturns(cl, 1)
marketDailyret <- rowMeans(dailyret, na.rm = TRUE) #
First few entries should be c(NaN, -0.001131229,
-0.010251817, -0.000813797, ...)
weights <- -(dailyret - marketDailyret)
weights <- weights/rowSums(abs(weights), na.rm = TRUE)
pnl <- rowSums(backshift(1, weights)*dailyret, na.rm =
    TRUE)
testset <- which(tday >= startDate & tday <= endDate)
sharpeRatioTestset <- sqrt(252)*mean(pnl[testset], na.rm
    = TRUE)/sd(pnl[testset], na.rm = TRUE)
sharpeRatioTestset # 0.4170207
# With transaction costs
onewaytcost <- 5/10000 # 5 bps

```

```

pnl <- pnl - rowSums(abs(weights-backshift(1,
  weights))*onewaycost, na.rm = TRUE)
sharpeRatioTestset <- sqrt(252)*mean(pnl[testset], na.rm
  = TRUE)/sd(pnl[testset], na.rm = TRUE)
sharpeRatioTestset # -3.378513

```

STRATEGY REFINEMENT

Example 3.8: A Small Variation on an Existing Strategy

Let's refine the mean-reverting strategy described in Example 3.7. Recall that strategy has a mediocre Sharpe ratio of 0.25 and a very unprofitable Sharpe ratio of -3.19 after transaction costs in 2006. The only change we will make here is to update the positions at the market open instead of the close. In the MATLAB code, simply replace "cl" with "op" everywhere. You can download the R code as example3_8.R and Python code as example3_8.ipynb.

Lo and behold, the Sharpe ratio before-and-after costs are both very positive! I will leave it as an exercise for the reader to improve the Sharpe ratio further by testing the strategy on the S&P 400 mid-cap and S&P 600 small-cap universes.

SUMMARY

Backtesting is about conducting a realistic historical simulation of the performance of a strategy. The hope is that the future performance of the strategy will resemble its past performance, though as your investment manager will never tire of telling you, this is by no means guaranteed!

There are many nuts and bolts involved in creating a realistic historical backtest and in reducing the divergence of the future performance of the strategy from its backtest performance. Issues discussed here include:

- **Data:** Split/dividend adjustments, noise in daily high/low, and survivorship bias.
- **Performance measurement:** Annualized Sharpe ratio and maximum drawdown.

- **Look-ahead bias:** Using unobtainable future information for past trading decisions.
- **Data-snooping bias:** Using too many parameters to fit historical data, tweaking a strategy too many times in backtest, and avoiding it using a large enough sample, out-of-sample testing, and sensitivity analysis.
- **Transaction cost:** Impact of transaction costs on performance.
- **Strategy refinement:** Common ways to make small variations on the strategy to optimize performance.

After going through this chapter and working through some of the examples and exercises, you should have gained some hands-on experience in how to retrieve historical data and backtest a strategy with Excel, MATLAB, Python, or R.

When one starts testing a strategy, it may not be possible to avoid all these pitfalls due to constraints of time and other resources. In this case, it is okay to skip a few precautions to achieve a quick sense of whether the strategy has potential and is worthy of closer examination. Sometimes, even the most thorough and careful backtest cannot reveal problems that would be obvious after a few months of paper or real trading. One can always revisit each of these issues after the model has gone live.

Once you have backtested a strategy with reasonable performance, you are now ready to take the next step in setting up your trading business.

REFERENCES

- Bailey, David, and Marcos López de Prado. 2012. “The Sharpe Ratio Efficient Frontier.” *Journal of Risk* 15 (2 Winter 2012/13). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1821643.
- Bailey, David, J. Borwein, Marcos 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 (7): 802–812.

American Mathematical Society 61 (5 May): 458–471. <https://ssrn.com/abstract=2308659>.

Chan, Ernest. 2006. “Reader Suggests a Possible Trading Strategy with the GLD–GDX Spread.” *Quantitative Trading*, November 17. <http://epchan.blogspot.com/2006/11/reader-suggested-possible-trading.html>.

Chan, Ernest. 2017. “Paradox Resolved: Why Risk Decreased Expected Log Return but Not Expected Wealth.” *Quantitative Trading*, May 4. <http://epchan.blogspot.com/2017/05/paradox-resolved-why-risk-decreases.html>.

Sharpe, William. 1994. “The Sharpe Ratio.” *Journal of Portfolio Management*, Fall. <https://jpm.pm-research.com/content/21/1/49>.

Setting Up Your Business

BUSINESS STRUCTURE: RETAIL OR PROPRIETARY?

BOX 4.1 SHOULD YOU INCORPORATE BEFORE YOU TRADE?

If you open a brokerage account as an individual, your liability is *not* limited to what you deposit in that account. Retail brokerages are known to ask their customers to “make whole” the negative equity in their account when a highly levered position loses more than 100 percent of the margin deposit. This happened in the aftermath of the Swiss Franc unpegging in January 2015. Do not be surprised to get a stern legal letter requesting a large payment from your broker when that happens, and perhaps endless phone calls from a debt collector if you don’t pay up.

To avoid this personal liability, I would recommend you set up a corporation as your trading vehicle and open an account through this entity at a retail brokerage. Your loss will then be limited to your initial investment, though you may have to file corporate bankruptcy when things go wrong. In the United States, a limited liability company or S-Corp. may

work best, as tax gains or losses can be passed directly to you personally. Even if you are not a US resident, you can still easily incorporate a US LLC for trading in a US brokerage account. I used bizfilings.com for that, but there are many other similar services, such as Stripe Atlas. No legal help is required if you are the sole shareholder. Of course, you will still be subject to the various rules and regulations imposed by the SEC, such as Regulation T.

I made them sound like bad things when I spoke of rules and regulations imposed by proprietary trading firms. But, actually, some of these rules (such as the prohibition from trading penny stocks or the prohibition from carrying short positions overnight) are actually risk management measures for your own protection. Often, when the going is good, traders will bemoan these constraints limiting their flexibility and profitability. They may even decide to start their own retail trading accounts and start trading on their own. However, when they suffer the (almost inevitable) big drawdown, they will wish that someone were there to restrain their risk appetites and come to regret this unfettered freedom. (The teenager in us has never left, after all.)

The decision whether to go retail or to join a proprietary trading firm is generally based on your need of capital, the style of your strategy, and your skill level. For example, if you run a low-risk, market-neutral strategy that nevertheless requires a much higher leverage than allowed by Regulation T in order to generate good returns, a proprietary account may be better for you. However, if you engage in high-frequency futures trading that does not require too much capital, a retail account may save you a lot of costs and hassles. Similarly, a very experienced trader with strong risk management practices and emotional stability probably doesn't need the guidance given by a proprietary firm, but less experienced traders may benefit from the imposed restraints.

There is another consideration that applies to those of you who have discovered some unique, highly profitable strategies. In this situation, you may prefer to open a retail trading account, because if you trade through a proprietary account, your proprietary trading firm is going to find out about your highly profitable strategy and may "piggyback" on your strategy with a lot of its own capital. In this case, your strategy will suffer more market impact trading cost as time goes on.

Table 4.1 summarizes the pros and cons for each choice.

TABLE 4.1 Retail versus Proprietary Trading

Issue	Retail Trading	Proprietary Trading
Legal requirement to open account	None.	Need to pass FINRA Series 7 examination and satisfy other FINRA-imposed restrictions.
Initial capital requirement	Substantial.	Small.
Available leverage or buying power	Determined by SEC Regulation T. Generally 2x leverage for overnight positions, and 4x for intraday positions.	Based on firm's discretion. Can be as high as 20x or more for intraday or hedged positions.
Liability to losses	Unlimited, unless account is opened through an S corporation or LLC.	Limited to initial investment.
Commissions and fees	Low commissions (perhaps less than 0.5 cent a share) and minimal monthly fees for data.	Higher commissions and significant monthly fees.
Bankruptcy risk of brokerage	No risk. Account insured by Securities Investor Protection Corporation (SIPC).	Has risk. Account is not insured.
Training, mentoring, guidance	None.	May provide such services, sometimes at a fee.
Disclosure of trade secrets	Little or no risk, especially if retail brokerage does not have proprietary trading unit.	Has risk. Managers can easily "piggyback" on profitable strategies.
Restrictions on trading style	No restrictions, as long as it is allowed by SEC.	May have restrictions, such as prohibitions on holding overnight short positions.
Risk management	Mostly self-imposed.	More comprehensive and imposed by managers.

SUMMARY

This chapter focused on those decisions and steps that you need to take to bridge the research phase and the execution phase of your trading business. I have covered the pros and cons of retail trading versus proprietary trading and the issues to consider in choosing a brokerage or proprietary trading firm.

In a nutshell, retail brokerages give you complete freedom and better capital protection but smaller leverage, while proprietary trading firms give you less freedom and less capital protection but much higher leverage. Finding a suitable retail brokerage is relatively easy. It took me less than a month to research and settle on one, and I have not found a reason to switch yet. Finding a suitable proprietary trading firm is much more involved, since there are contracts to sign and an exam (Series 7) to pass. It took me several months to get my account set up at one.

Of course, you can choose to have both retail and proprietary accounts, each tailored to the specific needs of your strategies. This way, you can also easily compare their speed of execution and depth of liquidity.

Regardless of whether you have chosen to trade in a retail brokerage or join a proprietary trading firm, you need to make sure their trading account and systems have these features:

- Relatively low commissions
- Trade a good variety of financial instruments
- Access to deep pool of liquidity
- Most importantly, API for real-time data retrieval and order transmission

I also described the progressive buildup of the physical infrastructure you need to build in order to run a trading business. Some of the components of a trader's operating environment mentioned are:

- Personal computer
- High-speed internet connection
- Noninterruptible power supply
- Real-time data and news feed and subscription to financial TV news channels
- VPS

Building out the physical trading infrastructure is actually quite easy, since in the beginning you probably have all the components ready in your home office already. I have found that it is easy to trade a million-dollar portfolio with nothing more than a few thousand dollars' initial investment in your physical infrastructure and a few hundred dollars a month in operating cost. But if you want to increase your trading capacity or improve your returns, additional incremental investments will be needed.

Once you have considered and taken these steps, you are now positioned to build an automated trading environment to execute your strategy, which will be covered in the next chapter.

REFERENCES

Economist. 2007. "Too Much Information." July 12. www.economist.com/finance/displaystory.cfm?story_id=9482952.

Markoff, John. 2007. "Faster Chips Are Leaving Programmers in Their Dust." *New York Times*, December 17. www.nytimes.com/2007/12/17/technology/17chip.html?ex=1355634000&en=a81769355deb7953&ei=5124&partner=permalink&expref=permalink.

Oldfield, Richard. 2007. *Simple but Not Easy*. Doddington Publishing.

CHAPTER 5

Execution Systems

WHAT AN AUTOMATED TRADING SYSTEM CAN DO FOR YOU

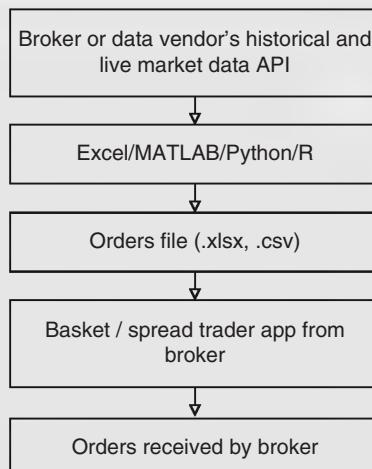


FIGURE 5.1 Semiautomated trading system.

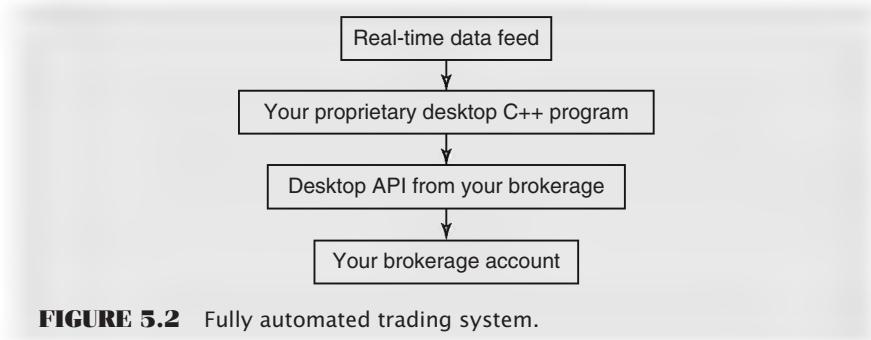


FIGURE 5.2 Fully automated trading system.

SUMMARY

An automated trading system is a piece of software that automatically generates and transmits orders to your brokerage account based on your trading strategy. There are three advantages to having this software:

- It ensures the faithful adherence to your backtested strategy.
- It eliminates manual operation so that you can simultaneously run multiple strategies.
- Most importantly, it allows speedy transmissions of orders, which is essential to high-frequency trading strategies.

Regarding the difference between a semiautomated trading system and a fully automated trading system:

- In a semiautomated trading system, the trader still needs to manually upload a text file containing order details to a basket trader or spread trader, and manually press a button to transmit the orders at the appropriate time. However, the order text file can be automatically generated by a program such as MATLAB, Python, or R.
- In a fully automated trading system, the program will be able to automatically upload data and transmit orders throughout the trading day or even over many days.

After the creation of an ATS, you can then focus on the various issues that are important in execution: minimizing transaction costs

and paper trading. Minimizing transaction costs is mainly a matter of not allowing your order size to be too big relative to its average trading volume and relative to its market capitalization. Paper trading allows you to:

- Discover software bugs in your trading strategy and execution programs.
- Discover look-ahead or even data-snooping bias.
- Discover operating difficulties and plan for operating schedules.
- Estimate transaction costs more realistically.
- Gain important intuition about P&L volatility, capital usage, portfolio size, and trade frequency.

Finally, what do you do in the situation where your live trading underperforms your backtest? You can start by addressing the usual problems: Eliminate bugs in the strategy or execution software; reduce transaction costs; and simplify the strategy by eliminating parameters. But, fundamentally, your strategy still may have suffered from data-snooping bias or regime shift.

If you believe (and you can only believe, as you can never *prove* this) that your poor live-trading performance is due to bad luck and not to data-snooping bias in your backtest nor to a regime shift, how should you proceed when the competing demands of perseverance and capital preservation seem to suggest opposite actions? This critical issue will be addressed in the next chapter, which discusses systematic ways to preserve capital in the face of losses and yet still be in a position to recover once the tide turns.

CHAPTER 6

Money and Risk Management

OPTIMAL CAPITAL ALLOCATION AND LEVERAGE

Example 6.1: An Interesting Puzzle (or Why Risk Is Bad for You)¹

Here is a little puzzle that may stymie many a professional trader. Suppose a certain stock exhibits a true (geometric) random walk, by which I mean there is a 50–50 chance that the stock is going up 1 percent or down 1 percent every minute. If you buy this stock, are you most likely—in the long run and ignoring financing costs—to make money, lose money, or be flat?

Most traders will blurt out the answer “Flat!” and that is wrong. The correct answer is that you will lose money, at the rate of 0.005 percent

¹This example was reproduced with corrections from my blog article “Maximizing Compounded Rate of Return,” which you can find at epchan.blogspot.com/2006/10/maximizing-compounded-rate-of-return.html.

(or 0.5 basis point) every minute! This is because for a geometric random walk, the average compounded rate of return is not the short-term (or one-period) return m (0 here), but is $g = m - s^2/2$. This follows from the general formula for compounded growth $g(f)$ given in the appendix to this chapter, with the leverage f set to 1 and risk-free rate r set to 0. This is also consistent with the fact that the geometric mean of a set of numbers is always smaller than the arithmetic mean (unless the numbers are identical, in which case the two means are the same). When we assume, as I did, that the arithmetic mean of the returns is zero, the geometric mean, which gives the average compounded rate of return, must be negative.

The take-away lesson here is that risk always decreases long-term growth rate—hence the importance of risk management! (See also Box 6.1 on “Loss Aversion Is Not a Behavioral Bias.”)

Example 6.2: Calculating the Optimal Leverage Based on the Kelly Formula

Let's see an example of the Kelly formula at work. Suppose our portfolio consists of just a long position in SPY, the exchange-traded fund (ETF) tracking the S&P 500 index. Let's suppose that the mean annual return of SPY is 11.23 percent, with an annualized standard deviation of 16.91 percent, and that the risk-free rate is 4 percent. Hence, the portfolio has an annual mean excess return of 7.231 percent and an annual standard deviation of 16.91 percent, giving it a Sharpe ratio of 0.4275. The optimal leverage according to the Kelly formula is $f = 0.07231/0.1691^2 = 2.528$. (Notice one interesting tidbit: The Kelly f is independent of time scale, so it actually does not matter whether you annualize your return and standard deviation, as opposed to the Sharpe ratio, which is time scale dependent.) Finally, the annualized compounded, levered growth rate is 13.14 percent, which includes the financing costs.

You can verify these numbers yourselves by downloading the SPY daily prices from Yahoo! Finance and computing the various quantities on a spreadsheet. I did that on December 29, 2007, and my spreadsheet is available at epchan.com/book/example6_2.xls. In column H, I have computed the daily returns of the (adjusted) closing prices of SPY, while in row 3760 starting at column H, I have computed the (annualized) mean return of SPY, the standard deviation of SPY, the mean excess return of the portfolio, the Sharpe ratio of the portfolio, the Kelly leverage, and, finally, the compounded growth rate.

The Kelly leverage of 2.528 that we computed is saying that, for this strategy, if you have \$100,000 in cash to invest, and if you really believe the expected values of your returns and standard deviations, you should borrow money to buy \$252,800 worth of SPY. Furthermore, expect an annual compounded return on your \$100,000 investment to be 13.14 percent.

For comparison, let's see what compounded growth rate we will get if we did not leverage our investment (see the formula in the appendix to this chapter): $g = r + m - s^2/2 = 0.1123 - (0.1691)^2/2 = 9.8$ percent, where m is the annualized mean return and s is the annualized standard deviation of returns. This, and not mean annual return of 11.23 percent, is the long-term growth rate of buying SPY with cash only.

Example 6.3: Calculating the Optimal Allocation Using the Kelly Formula

We pick three sector-specific ETFs and see how we should allocate capital among them to achieve the maximum growth rate for the portfolio. The three ETFs are: OIH (oil service), RKH (regional bank), and RTH (retail). The daily prices are downloaded from Yahoo! Finance and saved in epchan.com/book as OIH.xls, RKH.xls, and RTH.xls.

Using MATLAB

Here is the MATLAB program (epchan.com/book/example6_3.m) to retrieve these files and calculate M , C , and F^* .

```
% make sure previously defined variables are erased.  
clear;  
% read a spreadsheet named "OIH.xls" into MATLAB.  
[num1, txt1]=xlsread('OIH');  
% the first column (starting from the second row) is  
% the trading days in format mm/dd/yyyy.  
tday1=txt1(2:end, 1);  
tday1=datestr(datenum(tday1, 'mm/dd/yyyy'), 'yyyyymmdd');  
% convert the format into yyyyymmdd.  
  
% convert the date strings first into cell arrays  
% and then into numeric format.  
tday1=str2double(cellstr(tday1));  
% the last column contains the adjusted close prices.  
adjcls1=num1(:, end);  
% read a spreadsheet named "RKH.xls" into MATLAB.  
[num2, txt2]=xlsread('RKH');  
% the first column (starting from the second row) is  
% the trading days in format mm/dd/yyyy.  
tday2=txt2(2:end, 1);  
% convert the format into yyyyymmdd.  
tday2=..  
datestr(datenum(tday2, 'mm/dd/yyyy'), 'yyyyymmdd');  
  
% convert the date strings first into cell arrays and  
% then into numeric format.  
tday2=str2double(cellstr(tday2));  
adjcls2=num2(:, end);  
  
% read a spreadsheet named "RTH.xls" into MATLAB.  
[num3, txt3]=xlsread('RTH');  
% the first column (starting from the second row) is
```

```

% the trading days in format mm/dd/yyyy.
tday3=txt3(2:end, 1);
% convert the format into yyyyymmdd.
tday3=..
datestr(datenum(tday3, 'mm/dd/yyyy'), 'yyyyymmdd');

% convert the date strings first into cell arrays and
% then into numeric format.
tday3=str2double(cellstr(tday3));
adjcls3=num3(:, end);

% merge these data
tday=union(tday1, tday2);
tday=union(tday, tday3);
adjcls=NaN(length(tday), 3);

[foo idx1 idx]=intersect(tday1, tday);
adjcls(idx, 1)=adjcls1(idx1);
[foo idx2 idx]=intersect(tday2, tday);
adjcls(idx, 2)=adjcls2(idx2);
[foo idx3 idx]=intersect(tday3, tday);
adjcls(idx, 3)=adjcls3(idx3);

ret=(adjcls-lag1(adjcls))./lag1(adjcls); % returns

% days where any one return is missing
baddata=find(any(~isfinite(ret), 2));
% eliminate days where any one return is missing
ret(baddata,:)=[] ;
% excess returns: assume annualized risk free rate is 4%
excessRet=ret-repmat(0.04/252, size(ret));

% annualized mean excess returns
M=252*mean(excessRet, 1)'; % M =
%
% 0.1396
% 0.0294
% -0.0073

C=252*cov(excessRet) % annualized covariance matrix
% C =
%
% 0.1109 0.0200 0.0183
% 0.0200 0.0372 0.0269
% 0.0183 0.0269 0.0420

```

```

F=inv(C)*M % Kelly optimal leverages
%
% F =
%
% 1.2919
% 1.1723
% -1.4882

```

Notice that the mean excess return of RTH is negative. Given this, it is not surprising that the Kelly formula recommends we short RTH.

You might wonder what the Sharpe ratio and the maximum compounded growth rate generated using this optimal allocation are. It turns out that the maximum growth rate of a multistrategy Gaussian process is

$$g(F^*) = r + F^{*T} C F^*/2$$

and the Sharpe ratio is given by

$$S = \sqrt{F^{*T} C F^*}.$$

Here is the MATLAB code snippet that calculates these two quantities:

```

% Maximum annualized compounded growth rate
g=0.04+F'*C*F/2 % g =
%
% 0.1529

S=sqrt(F'*C*F) % Sharpe ratio of portfolio
%
% S =
%
% 0.4751

```

Notice that the compounded growth rate of the portfolio is 15.29 percent, higher than that of the maximum growth rate achievable by any of the individual stocks. (As an exercise, you can verify that the compounded growth rate of OIH, which has the highest one-period return among the three stocks, is 12.78 percent.)

Using Python

Here is the equivalent Python Jupyter Notebook code, downloadable as example6_3.ipynb.

```

Calculating the Optimal Allocation Using Kelly formula
import numpy as np
import pandas as pd
from numpy.linalg import inv
df1=pd.read_excel('OIH.xls')

```

```

df2=pd.read_excel('RKH.xls')
df=pd.merge(df1, df2, on='Date', suffixes=('_OIH', '_RKH'))
df.set_index('Date', inplace=True)
df3=pd.read_excel('RTH.xls')
df=pd.merge(df, df3, on='Date')
df.rename(columns={"Adj Close": "Adj Close_RTH"}, inplace=True)
df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
dailyret=df.loc[:, ('Adj Close_OIH', 'Adj Close_RKH',
'Adj Close_RTH')].pct_change()
dailyret.rename(columns={"Adj Close_OIH": "OIH",
"Adj Close_RKH": "RKH", "Adj Close_RTH": "RTH"}, inplace=True)
excessRet=dailyret-0.04/252
M=252*excessRet.mean()
M
OIH      0.139568
RKH      0.029400
RTH     -0.007346
dtype: float64
C=252*excessRet.cov()
C
OIH
RKH
RTH
OIH
0.110901
0.020014
0.018255
RKH
0.020014
0.037165
0.026893
RTH
0.018255
0.026893
0.041967
F=np.dot(inv(C), M)
F
array([ 1.2919082 ,  1.17226473, -1.48821285])
g=0.04+np.dot(F.T, np.dot(C, F))/2
g
0.1528535789840623
S=np.sqrt(np.dot(F.T, np.dot(C, F)))
S
0.47508647420035505

```

Using R

Here is the R code, downloadable as example6_3.R.

```
library('zoo')
source('calculateReturns.R')
source('calculateMaxDD.R')
source('backshift.R')

data1 <- read.delim("OIH.txt") # Tab-delimited
data_sort1 <- data1[order(as.Date(data1[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates
(1st column of data)
tday1 <- as.integer(format(as.Date(data_sort1[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls1 <- data_sort1[,ncol(data_sort1)]

data2 <- read.delim("RKH.txt") # Tab-delimited
data_sort2 <- data2[order(as.Date(data2[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates
(1st column of data)
tday2 <- as.integer(format(as.Date(data_sort2[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls2 <- data_sort2[,ncol(data_sort2)]

data3 <- read.delim("RTH.txt") # Tab-delimited
data_sort3 <- data3[order(as.Date(data3[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates
(1st column of data)
tday3 <- as.integer(format(as.Date(data_sort3[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls3 <- data_sort3[,ncol(data_sort3)]

# merge these data
tday <- union(tday1, tday2)
tday <- union(tday, tday2)
tday <- tday[order(tday)]

adjcls <- matrix(NaN, length(tday), 3)

adjcls[tday %in% tday1, 1] <- adjcls1
adjcls[tday %in% tday2, 2] <- adjcls2
adjcls[tday %in% tday3, 3] <- adjcls3

ret <- calculateReturns(adjcls, 1) # daily returns
excessRet <- ret - 0.04/252 # excess returns: assume
annualized risk free rate is 4%
```

```

# annualized mean excess returns
M <- 252*colMeans(excessRet, na.rm = TRUE)
# c(0.143479780597111, 0.0560305170502084,
# -0.0073464585163155)
# annualized covariance matrix
C<- 252*cov(excessRet, use = "pairwise.complete.obs")
# 0.11305722  0.01986547  0.01825486
# 0.01986547  0.04263365  0.02689284
# 0.01825486  0.02689284  0.04196684

# Kelly optimal leverage
F <- solve(C) %*% M
# 1.240554
# 1.992328
# -1.991381

# Maximum annualized compounded growth rate
g <- 0.04+t(F) %*% C %*% F/2 # 0.1921276

# Sharpe ratio of portfolio
S <- sqrt(t(F) %*% C %*% F) # 0.5515933

```

PSYCHOLOGICAL PREPAREDNESS

BOX 6.1 LOSS AVERSION IS NOT A BEHAVIORAL BIAS*

In his famous book *Thinking, Fast and Slow*, the Nobel laureate Daniel Kahneman (2011) described one common example of a behavioral finance bias:

“You are offered a gamble on the toss of a [fair] coin.
If the coin shows tails, you lose \$100.
If the coin shows heads, you win \$110.
Is this gamble attractive? Would you accept it?”

(I have modified the numbers to be more realistic in a financial market setting, but otherwise it is a direct quote.)

Experiments show that most people would not accept this gamble, even though the expected gain is \$5. This is the so-called “loss aversion” behavioral bias, and is considered irrational. Kahneman went on to write that “professional risk takers” (read “traders”) are more willing to act rationally and accept this gamble.

It turns out that the loss-averse “layman” is the one acting rationally here. It is true that if we have infinite capital, and can play infinitely many rounds of this game simultaneously, we should expect a \$5 gain per round. But trading isn’t like that. We are dealt one coin at a time, and if we suffer a string of losses, our capital will be depleted and we will be in debtor prison if we keep playing. The proper way to evaluate whether this game is attractive is to evaluate the expected compound rate of growth of our capital.

Let’s say we are starting with a capital of \$1,000. The expected return of playing this game once is initially 0.005. The standard deviation of the

*This commentary was reproduced from my blog article of the same title at [predictnow.ai
/blog/loss-aversion-is-not-a-behavioral-bias/](http://predictnow.ai/blog/loss-aversion-is-not-a-behavioral-bias/)

return is 0.105. To simplify matters, let's say we are allowed to adjust the payoff of each round so we have the same expected return and standard deviation of return each round. For example, if at some point we earned so much that we doubled our capital to \$2,000, we are allowed to win \$220 or lose \$200 per round. What is the expected growth rate of our capital? As Example 6.1 shows, in the continuous approximation it is -0.0005125 per round – we are losing, not gaining! The layman is right to refuse this gamble.

Loss aversion, in the context of a risky game played repeatedly, is rational, and not a behavioral bias. Our primitive, primate instinct grasped a truth that behavioral economists cannot. It only seems like a behavioral bias if we take an “ensemble view” (i.e., allowed infinite capital to play many rounds of this game simultaneously), instead of a “time series view” (i.e. allowed only finite capital to play many rounds of this game in sequence, provided we don't go broke at some point). The time series view is the one relevant to all traders. In other words, take time average, not ensemble average, when evaluating real-world risks.

The important difference between ensemble average and time average has been raised in a paper by physicist Ole Peters and Nobel laureate Murray Gell-Mann (Peters, et al., 2016). It deserves to be much more widely read in the behavioral economics community. But beyond academic interest, there is a practical importance in emphasizing that loss aversion is rational. As traders, we should not only focus on average returns: risks can depress compound returns severely.

SUMMARY

Risk management is a crucial discipline in trading. The trading world is littered with numerous examples of giant hedge funds and investment banks laid low by enormous losses due to a single trade or in a very short period of time. Most of these losses are due to overleveraging positions and not to an inherently erroneous model. Typically, traders will not overleverage a model that has not worked very well. It is a hitherto superbly performing model that is at the greatest risk of huge loss due to overconfidence and overleverage. This chapter therefore provides an important tool for risk management: the determination of the optimal leverage using the Kelly formula.

Besides the determination of the optimal leverage, the Kelly formula has a very useful side benefit: It also determines the optimal allocation of capital among different strategies, based on the covariance of their returns.

But no risk management formula or system will prevent disasters if you are not psychologically prepared for the ups and downs of trading and thus deviating from the prescriptions of rational decision making (i.e., your models). The ultimate risk management mind-set is very simple: Do not succumb to either despair or greed. To gain practice in this psychological discipline, one must proceed slowly with small position size, and thoroughly test various aspects of the trading business (model, software, operational procedure, money and risk management) before scaling up according to the Kelly formula.

I have found that in order to proceed slowly and cautiously, it is helpful to have other sources of income or other businesses to help sustain yourself either financially or emotionally (to avoid the boredom associated with slow progress). It is indeed possible that finding a diversion, whether income producing or not, may actually help improve the long-term growth of your wealth.

APPENDIX: A SIMPLE DERIVATION OF THE KELLY FORMULA WHEN RETURN DISTRIBUTION IS GAUSSIAN

If we assume that the return distribution of a strategy (or security) is Gaussian, then the Kelly formula can be derived very easily. We start with the formula for a compounded, levered growth rate applicable to a Gaussian process:

$$g(f) = r + fm - s^2 f^2 / 2$$

where f is the leverage; r is the risk-free rate; m is the average simple, uncompounded one-period excess return; and s is the standard deviation of those uncompounded returns. This formula for compounded growth rate can itself be derived quite simply, but not as simply as the Kelly formula, so I leave its derivation for the reader to look up in the Thorp article referenced earlier.

To find the optimal f , which maximizes g , simply take its first derivative with respect to f and set the derivative to zero:

$$dg/df = m - s^2 f = 0$$

Solving this equation for f gives us $f = m/s^2$, the Kelly formula for one strategy or security under the Gaussian assumption.

REFERENCES

- Chan, Ernest. 2006a. “A ‘Highly Improbable’ Event? A Historical Analysis of the Natural Gas Spread Trade That Bought Down Amaranth.” *Quantitative Trading* blog, October 2, <http://epchan.blogspot.com/2006/10/highly-improbable-event.html>.
- Kahneman, Daniel. 2011. *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- Khandani, Amir E., and Andrew Lo. 2007. “What Happened to the Quants in August 2007?” MIT. <https://web.mit.edu/Alo/www/Papers/august07.pdf>.
- Lowenstein, Roger. 2000. *When Genius Failed: The Rise and Fall of Long-Term Capital Management*. Random House.
- Peters, O., and M. Gell-Mann. 2016. “Evaluating Gambles Using Dyanmics.” *Chaos* 26, 023103. <https://doi.org/10.1063/1.4940236>.
- Poundstone, William. 2005. *Fortune’s Formula*. New York: Hill and Wang.
- Ritter, Jay. 2003. “Behavioral Finance.” *Pacific-Basin Finance Journal* 11(4, September): 429–437.
- Taleb, Nassim. 2007. *The Black Swan: The Impact of the Highly Improbable*. Random House.
- Thaler, Richard. 1994. *The Winner’s Curse*. Princeton, NJ: Princeton University Press.
- Thorp, Edward. 1997. “The Kelly Criterion in Blackjack, Sports Betting, and the Stock Market.” *Handbook of Asset and Liability Management*, Volume I, Zenios and Ziemba (eds.). Elsevier 2006. www.EdwardOThorp.com.

CHAPTER 7

Special Topics in Quantitative Trading

REGIME CHANGE AND CONDITIONAL PARAMETER OPTIMIZATION

Example 7.1: Conditional Parameter Optimization applied to an ETF trading strategy

(This example is reproduced from a blog post on predictnow.ai/blog.)

To illustrate the CPO technique, we describe below an example trading strategy on an ETF.

This strategy uses the lead-lag relationship between the GLD and GDX ETFs using 1-minute bars from January 1, 2006, until December 31, 2020, splitting it 80%/20% between train/test periods. The trading strategy has 3 trading parameters: the hedge ratio (GDX_weight), entry threshold (entry_threshold), and a moving lookback window (lookback). The spread is defined as

$$\text{Spread}(t) = \text{GLD_close}(t) - \text{GDX_close}(t) \times \text{GDX_weight}. \quad (1)$$

We may enter a trade for GLD at time t , and exit it at time $t + 1$ minute, hopefully realizing a profit. We want to optimize the three trading parameters on a $5 \times 10 \times 8$ grid. The grid is defined as follows:

$$\begin{aligned} \text{GDX_weight} &= \{2, 2.5, 3, 3.5, 4\} \\ \text{entry_threshold} &= \{0.2, 0.3, 0.4, 0.5, 0.7, 1, 1.25, 1.5, 2, 2, 5\} \\ \text{lookback} &= \{30, 60, 90, 120, 180, 240, 360, 720\} \end{aligned}$$

To be clear, even though we are using GLD and GDX prices and functions of these prices to make trading decisions, we only trade GLD, unlike the typical long-short pair trading setup.

Every minute we compute $\text{Spread}(t)$ in equation (1), and compute its “Bollinger Bands,” conventionally defined as

$$Z_score(t) = \frac{\text{Spread}(t) - \text{Spread_EMA}(t)}{\sqrt{\text{Spread_VAR}(t)}} \quad (2)$$

where Spread_EMA is the exponential moving average of the Spread, and Spread_VAR is its exponential moving variance (see the endnote for their conventional definitions).

Similar to a typical mean-reverting strategy using Bollinger Bands, we trade into a new GLD position based on these rules:

- a. Buy GLD if $Z_score < -\text{entry_threshold}$ (resulting in long position).
- b. Short GLD if $Z_score > \text{entry_threshold}$ (resulting in short position).

- c. Liquidate long position if $Z_score > \text{exit_threshold}$.
- d. Liquidate short position if $Z_score < -\text{exit_threshold}$.

exit_threshold can be anywhere between entry_threshold and $-\text{entry_threshold}$. After optimization in the train set, we set $\text{exit_threshold} = -0.6 * \text{entry_threshold}$ and keep that relationship fixed when we vary entry_threshold in our future (unconditional or conditional) parameter optimizations. We trade the strategy on 1-minute bars between 9:30 and 15:59 ET, and liquidate any position at 16:00. For each combination of our three trading parameters, we record the daily return of the resulting intraday strategy and form a time series of daily strategy returns, to be used as labels for our machine learning step in CPOs. Note that since the trading strategy may execute multiple round trips per day before forced liquidation at the market close, this daily strategy return is the sum of such round-trip returns.

Unconditional vs. Conditional Parameter Optimizations

In conventional, unconditional, parameter optimization, we select the three trading parameters (GDX_weight, entry threshold, and lookback) that maximize cumulative in-sample return over the three-dimensional parameter grid using exhaustive search. (Gradient-based optimization did not work due to multiple local maxima.) We use that fixed set of three optimal trading parameters to specify the strategy out-of-sample on the test set.

With conditional, parameter optimization, the set of trading parameters used each day depends on a predictive machine-learning model trained on the train set. This model will predict the future one-day return of our trading strategy, given the trading parameters and other market conditions. Since the trading parameters can be varied at will (i.e., they are control variables), we can predict a different future return for many sets of trading parameters each day, and select the optimal set that predicts the highest future return. That optimal parameter set will be used for the trading strategy for the next day. This step is taken after the current day's market close and before the market open of the next day.

In addition to the three trading parameters, the predictors (or "features") for input to our machine learning model are eight technical indicators obtained from the Technical Analysis Python library: Bollinger Bands Z-score, Money Flow, Force Index, Donchian Channel, Average True Range, Awesome Oscillator, and Average Directional Index. We choose these indicators to represent the market conditions. Each indicator actually produces 2×7 features, since we apply them to each of the ETFs GLD and GDX price

series, and each was computed using seven different lookback windows: 50, 100, 200, 400, 800, 1600, and 3200 minutes. (Note: This is not the same as the trading parameter “lookback” described earlier.) Hence, there are a total of $3 + 8 \times 2 \times 7 = 115$ features used in predicting the future one-day return of the strategy. But because there are $5 \times 10 \times 8 = 400$ combinations of the three trading parameters, each trading day comes with 400 rows of training data that looks something like the table below (labels are not displayed):

GDX_weight	entry_threshold	look back	Z-score-GLD(50)	Z-score-GDX(50)	Money-Flow-GLD(50)	Money-Flow-GDX(50)	...
2	0.2	30	0.123	0.456	1.23	4.56	...
2	0.2	60	0.123	0.456	1.23	4.56	...
2	0.2	90	0.123	0.456	1.23	4.56	...
...							
4	5	240	0.123	0.456	1.23	4.56	...
4	5	360	0.123	0.456	1.23	4.56	...
4	5	720	0.123	0.456	1.23	4.56	

After the machine learning model is trained, we can use it for live predictions and trading. Each trading day after the market closes, we prepare an input vector, which is structured like one row of the table above, populated with one particular set of the trading parameters and the current values of the technical indicators, and use the machine learning model to predict the trading strategy’s return on the next day. We do that 400 times, varying the trading parameters, but obviously not the technical indicators’ values, and find out which trading parameter set predicts the highest return. We adopt that optimal set for the trading strategy next day. In mathematical terms,

$$(GDX_weight_optimal, entry_threshold_optimal, lookback_optimal) = \\ argmax(GDX_weight, entry_threshold, lookback) \{... \\ predict(GDX_weight, entry_threshold, lookback, technical indicators) \} \quad (3)$$

where *predict* is the predictive function available from our machine learning website predictnow.ai’s API, which uses random forest with boosting as the training algorithm. The sample Python Jupyter notebook code fragment for training a model and using it for predictions is displayed here. (The code won’t work unless you sign up for a trial with predictnow.ai.)

```

# TO BEGIN ANY WORK WITH PREDICTNOW.AI CLIENT, WE START
# BY IMPORTING AND CREATING A CLASS INSTANCE
from predictnow.pdapi import PredictNowClient
import pandas as pd
api_key = "%KeyProvidedToEachOfOurSubscriber"
api_host = "http://12.34.567.890:1000" # our SaaS server
username = "helloWorld"
email = "helloWorld@yourmail.com"
client = PredictNowClient(api_host,api_key)
# You will need to edit this input dataset file path and
# labelname!
file_path = 'my_amazing_features.xlsx'
labelname = 'Next_day_strategy_return'
import os
# FANTASTIC JOB! NOW YOUR PREDICTNOW.AI CLIENT HAS BEEN
# SETUP.
# For classification problems
#params = {'timeseries': 'yes', 'type':
#    'classification', 'feature_selection': 'shap', 'anal-
#ysis': 'none', 'boost': 'gbdt', 'testsize': '0.2',
#    'weights': 'no', 'eda': 'yes', 'prob_calib': 'no',
#    'mode': 'train'}
# For regression problems, suitable for CPO
params = {'timeseries': 'yes', 'type': 'regression',
    'feature_selection': 'none', 'analysis': 'none',
    'boost': 'gbdt', 'testsize': '0.2', 'weights': 'no',
    'eda': 'yes', 'prob_calib': 'no', 'mode': 'train'}

# LET'S CREATE THE MODEL BY SENDING THE PARAMETERS TO
# PREDICTNOW.AI
response = client.create_model(
    username=username, # only letters, numbers, or underscores
    model_name="test1",
    params=params,
)

# LET'S LOAD UP THE FILE TO PANDAS IN THE LOCAL ENVIRONMENT
from pandas import read_csv # If you have the Excel
    file, replace read_csv with read_excel
from pandas import read_excel
df = read_excel(file_path, engine="openpyxl") # Same here
df.name = "testdataframe" # Optional, but recommended
response = client.train(
    model_name="test1",
    input_df=df,
)

```

```

        label=labelname,
        username=username,
        email=email,
        return_output=False,
    )
    print("FANTASTIC! YOUR FIRST-EVER MODEL TRAINING AT PRE-
          DICTNOW.AI HAS BEEN COMPLETED!")
print(response)
# User can now examine the train/test sets results from
the model by calling the getresult function (and
providing the name of the model that resides on
Predictnow.ai server
status = client.getstatus(username=username, train_
    id=response["train_id"])
if status["state"] == "COMPLETED":
    response = client.getresult(
        model_name="test1",
        username=username,
    )
    import pandas as pd
    predicted_targets_cv = pd.read_json(response.
        predicted_targets_cv)
    print("predicted_targets_cv")
    print(predicted_targets_cv)
    predicted_targets_test = pd.read_json(response.
        predicted_targets_test)
    print("predicted_targets_test")
    print(predicted_targets_test)
    performance_metrics = pd.read_json(response.
        performance_metrics)
    print("performance_metrics")
    print(performance_metrics)
# # Now we can make LIVE predictions for many
combinations of the parameters by populating many
rows in the example_input_live.csv file with these
parameter combinations
if status["state"] == "COMPLETED":
    df = read_csv("example_input_live.csv") # Input data
    for live prediction
        df.name = "myfirstpredictname" # optional, but
            recommended
    # Making live predictions
    response = client.predict(
        model_name="test1",
        input_df=df,
        username=username,

```

```

        eda="yes",
        prob_calib=params["prob_calib"],
    )
# FOR LIVE PREDICTION: (remember labels and
# probabilities each can have many rows
# corresponding to many combinations of parameters
y_pred = pd.read_json(response.labels)
print("THE LABELS")
print(labels)

```

An example output labels file from this step looks like this:

Date	pred_target
2020-12-24 2.5_30_0.2 20218132334	0.011875
2020-12-24 2.5_60_0.2 20218132344	0.012139
2020-12-24 2.5_90_0.2 20218132354	0.012139
2020-12-24 2.5_120_0.2 20218132364	0.012975
2020-12-24 2.5_180_0.2 20218132374	0.012975
2020-12-24 2.5_240_0.2 20218132384	0.012975
2020-12-24 2.5_360_0.2 20218132394	0.012975
2020-12-24 2.5_720_0.2 20218132404	0.012975

where 2.5_30_0.2 is one parameter combination, and 2.5_60_0.2 is another.

```

input_features= df['Date'].values
for i in range(len(input_features)):
    #/ split y_pred['Date'] into actual date and
    # parameters string
    date_params = input_features[i].split(' ')
    params = date_params[1]
    if i==0:
        #initializing max_index and its parameters value
        #E.g. (2.5, 60, 0.2)
        params_cond_optimized = params
        y_pred_max = y_pred[i].values
    else:
        if y_pred[i].values >= y_pred_max:
            #updating max_index and its parameters value
            params_cond_optimized = params
            y_pred_max = y_pred[i].values
# params_cond_optimized is "conditionally optimal"
# parameters for the
# next day

```

It is important to understand that unlike a naïve application of machine learning to predict GLD's one-day return using technical indicators, we are using machine learning to predict the return of a trading strategy applied to GLD given a set of trading parameters, and using those predictions to optimize these parameters on a daily basis. The naïve approach is less likely to succeed because everybody is trying to predict GLD's (i.e., gold's) returns and inviting arbitrage activities, but nobody (until they read this book!) is predicting the returns of this particular GLD trading strategy. Furthermore, many traders do not like using machine learning as a black box to predict returns. In CPO, the trader's own strategy is making the actual predictions. Machine learning is merely used to optimize the parameters of this trading strategy. This provides for a much greater degree of transparency and interpretability.

Performance Comparisons

We compare out-of-sample test set performance of Unconditional vs. Conditional Parameter Optimization on the last three years of data ending on December 31, 2020, and find the cumulative three-year return to be 73% and 83%, respectively. All other metrics are improved using CPO. The comparable equity curves can be found in Figure 7.1.

	Unconditional Optimization	Conditional Optimization
Annual Return	17.29%	19.77%
Sharpe Ratio	1.947	2.325
Calmar Ratio	0.984	1.454

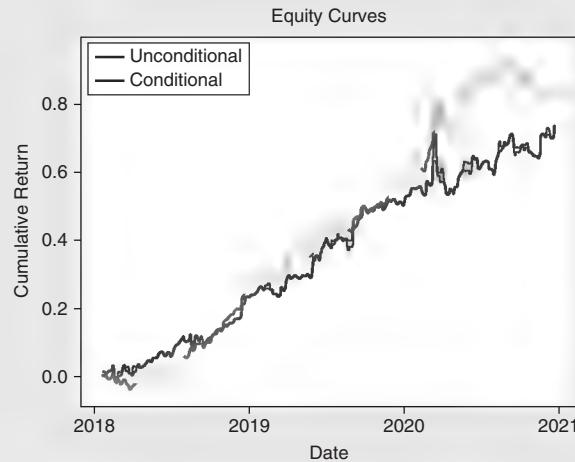


FIGURE 7.1 Cumulative returns of strategies based on Conditional vs. Unconditional Parameter Optimization.

Endnote: Definitions of *Spread_EMA* and *Spread_VAR*

$$\text{Spread_EMA}(0) = \text{Spread}(0),$$

$$\text{Spread_EMA}(t+1) = \frac{2}{\text{lookback_period}} \text{Spread}(t+1) + \\ \left(1 - \frac{2}{\text{lookback_period}}\right) \text{Spread_EMA}(t),$$

$$\text{Spread_VAR}(1) = (\text{Spread}(1) - \text{Spread}(0))^2,$$

$$\text{Spread_VAR}(t+1) = \frac{2}{\text{lookback_period}} (\text{Spread}(t+1) - \text{Spread_EMA}(t+1))^2 + \\ \left(1 - \frac{2}{\text{lookback_period}}\right) * \text{Spread_VAR}(t)$$

STATIONARITY AND COINTEGRATION

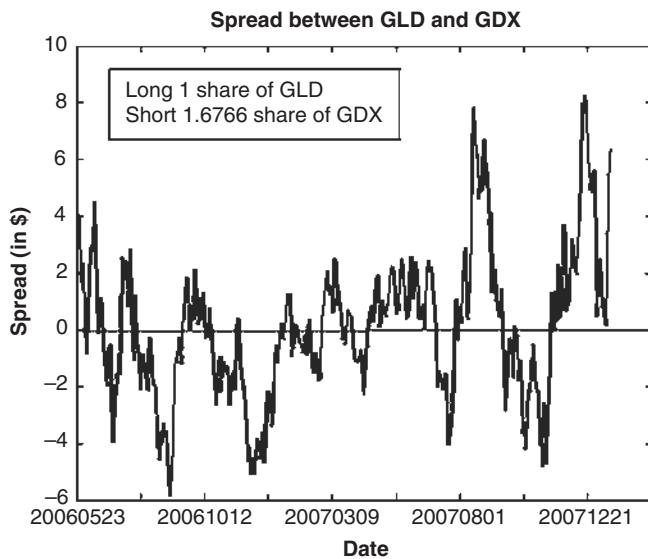


FIGURE 7.2 A stationary time series formed by the spread between GLD and GDX.

Example 7.2: How to Form a Good Cointegrating (and Mean-Reverting) Pair of Stocks

As I explained in the main text, if you are long one security and short another one in the same industry group and in the right proportion, sometimes the combination (or "spread") becomes a stationary series. A stationary series is an excellent candidate for a mean-reverting strategy. This example teaches you how to use a free MATLAB package, downloadable at www.spatial-econometrics.com, to determine if two price series are cointegrated and, if so, how to find the optimal *hedge ratio* (i.e., the number of shares of the second security versus one share of the first security).

The main method used to test for cointegration is called the *cointegrating augmented Dickey-Fuller test*, hence the function name *cadf*. A detailed description of this method can be found in the manual, also available on the same website mentioned earlier.

Using MATLAB

The following program is available online as epchan.com/book/example7_2.m:

```
% make sure previously defined variables are erased.  
clear;  
% read a spreadsheet named "GLD.xls" into MATLAB.  
[num, txt]=xlsread('GLD');  
% the first column (starting from the second row) is  
% the trading days in format mm/dd/yyyy.  
tday1=txt(2:end, 1);  
% convert the format into yyyyymmdd.  
tday1=..  
datestr(datenum(tday1, 'mm/dd/yyyy'), 'yyyyymmdd');  
% convert the date strings first into cell arrays and  
% then into numeric format.  
tday1=str2double(cellstr(tday1));  
% the last column contains the adjusted close prices.  
adjcls1=num(:, end);  
% read a spreadsheet named "GDX.xls" into MATLAB.  
[num2, txt2]=xlsread('GDX');  
% the first column (starting from the second row) is  
% the trading days in format mm/dd/yyyy.  
tday2=txt2(2:end, 1);  
% convert the format into yyyyymmdd.  
tday2=..  
datestr(datenum(tday2, 'mm/dd/yyyy'), 'yyyyymmdd');  
% convert the date strings first into cell arrays and  
% then into numeric format.  
tday2=str2double(cellstr(tday2));  
adjcls2=num(:, end);  
% find all the days when either GLD or GDX has data.  
tday=union(tday1, tday2);  
[foo idx idx1]=intersect(tday, tday1);  
% combining the two price series  
adjcls=NaN(length(tday), 2);  
adjcls(idx, 1)=adjcls1(idx1);
```

```

[foo idx idx2]=intersect(tday, tday2);
adjcls(idx, 2)=adjcls2(idx2);
% days where any one price is missing
baddata=find(any(~isfinite(adjcls), 2));
tday(baddata)=[];
adjcls(baddata,:)=[];
trainset=1:252; % define indices for training set
vnames=strvcat('GLD', 'GDX');
adjcls=adjcls(trainset, :);
tday=tday(trainset, :);

% run cointegration check using
% augmented Dickey-Fuller test
res=cadf(adjcls(:, 1), adjcls(:, 2), 0, 1);
prt(res, vnames);
% Output from cadf function:
% Augmented DF test for co-integration variables:
GLD,GDX
% CADF t-statistic      # of lags    AR(1) estimate
%      -3.18156477           1        -0.070038
%
%      1% Crit Value      5% Crit Value    10% Crit Value
%      -3.924             -3.380          -3.082
% The t-statistic of -3.18 which is in between the 5%
% Crit Value of -3.38
% and the 10% Crit Value of -3.08 means that there is a
% better than 90%
% probability that these 2 time series are cointegrated.
results=ols(adjcls(:, 1), adjcls(:, 2));
hedgeRatio=results.beta
z=results.resid;
% A hedgeRatio of 1.6766 was found.
% I.e. GLD=1.6766*GDX + z, where z can be
% interpreted as the
% spread GLD-1.6766*GDX and should be stationary.
% This should produce a chart similar to Figure 7.2.
plot(z);

```

Using Python

The following program is available as epchan.com/book/example7_2.ipynb:

How to Form a Good Cointegrating (and Mean-Reverting)

Pair of Stocks

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

```

```

from statsmodels.tsa.stattools import coint
from statsmodels.api import OLS
df1=pd.read_excel('GLD.xls')
df2=pd.read_excel('GDX.xls')
df=pd.merge(df1, df2, on='Date', suffixes=('_GLD', '_GDX'))
df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
trainset=np.arange(0, 252)
df=df.iloc[trainset,:]
Run cointegration (Engle-Granger) test
coint_t, pvalue, crit_value=coint(df['Adj Close_GLD'],
    df['Adj Close_GDX'])
(coint_t, pvalue, crit_value) # abs(t-stat) > critical
    value at 95%. pvalue says probability of null hypo-
    thesis (of no cointegration) is only 1.8%
(-2.3591268376687244,
 0.3444494880427884,
 array([-3.94060523, -3.36058133, -3.06139039]))
Determine hedge ratio
model=OLS(df['Adj Close_GLD'], df['Adj Close_GDX'])
results=model.fit()
hedgeRatio=results.params
hedgeRatio
Adj Close_GDX      1.631009
      dtype: float64
spread = GLD - hedgeRatio*GDX
spread=df['Adj Close_GLD']-hedgeRatio[0]*df['Adj Close_GDX']
plt.plot(spread)

```

You may notice that the Python code's Engle-Granger test generates a t -statistic of -2.4 , whose absolute value is less than the 90% critical value, indicating that the two series are *not* cointegrating. This contradicts the results of the MATLAB `cadf` test. Which should we trust? Let me just say that Python's libraries are free and come with no guarantees on accuracy nor correctness, whereas MATLAB employs a staff of numerous PhD computer scientists and statisticians.

Using R

You can download the R code as `example7_2.R`.

```

# Need the zoo package for its na.locf function
install.packages('zoo')
# Need the CADFtest package for its CADFtest function
install.packages('CADFtest')
library('zoo')
library('CADFtest')

```

```

data1 <- read.delim("GLD.txt") # Tab-delimited
data_sort1 <- data1[order(as.Date(data1[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates (1st
  column of data)
tday1 <- as.integer(format(as.Date(data_sort1[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls1 <- data_sort1[,ncol(data_sort1)]
data2 <- read.delim("GDX.txt") # Tab-delimited
data_sort2 <- data2[order(as.Date(data2[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates (1st
  column of data)
tday2 <- as.integer(format(as.Date(data_sort2[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls2 <- data_sort2[,ncol(data_sort2)]
# find the intersection of the two data sets
tday <- intersect(tday1, tday2)
adjcls1 <- adjcls1[tday1 %in% tday]
adjcls2 <- adjcls2[tday2 %in% tday]
# CADFtest cannot have NaN values in input
adjcls1 <- zoo::na.locf(adjcls1)
adjcls2 <- zoo::na.locf(adjcls2)
mydata <- list(GLD=adjcls1, GDX=adjcls2);
trainset <- 1:252
res <- CADFtest(model=GLD~GDX, data=mydata, type =
  "drift", max.lag.X=1, subset=trainset)
summary(res) # As the following input shows, p-value is
  about 0.005; hence we can reject null hypothesis of no
  cointegration at 99.5% level.
# Covariate Augmented DF test
# CADF test
# t-test statistic: -3.240868894
# estimated rho^2: 0.260414676
# p-value: 0.004975155
# Max lag of the diff. dependent variable: 1.000000000
# Max lag of the stationary covariate(s): 1.000000000
# Max lead of the stationary covariate(s): 0.000000000
#
# Call:
#   dynlm(formula = formula(model), start = obs.1, end =
  obs.T)
#
# Residuals:
#   Min     1Q Median     3Q    Max
# -2.70728 -0.26235  0.00595  0.29684  1.47164
#

```

```

# Coefficients:
#   Estimate Std. Error t value Pr(>|t|)
# (Intercept) -0.07570    0.29814  -0.254  0.79970
# L(y, 1)     -0.03817    0.01178  -3.241  0.00498 ** 
# L(d(y), 1)   0.08542    0.03077   2.776  0.00578 ** 
# L(X, 0)      0.75428    0.02802  26.919 < 2e-16 *** 
# L(X, 1)     -0.68942    0.03161 -21.812 < 2e-16 *** 
# 
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 
# 0.1 ' ' 1
#
# Residual standard error: 0.474 on 378 degrees of freedom
# Multiple R-squared:  0.6577, Adjusted R-squared:  0.6541 
# F-statistic: 241.8 on 3 and 378 DF, p-value: < 2.2e-16
# determines the hedge ratio
lmresult <- lm(GLD ~ 0 + GDX, mydata, subset=trainset )
hedgeRatio <- coef(lmresult) # 1.631009
z <- residuals(lmresult) # The residuals should be
# stationary (mean-reverting)
plot(z) # This should produce a chart similar to Figure 7.2.

```

The R code's Engle-Granger test generates a t -statistic of -3.2 , which rejects the null hypothesis that the pair is not cointegrating. This corroborates the MATLAB cadf test, while repudiating the Python's result. Moral of the story: Do not trust Python's statistics and econometrics packages.

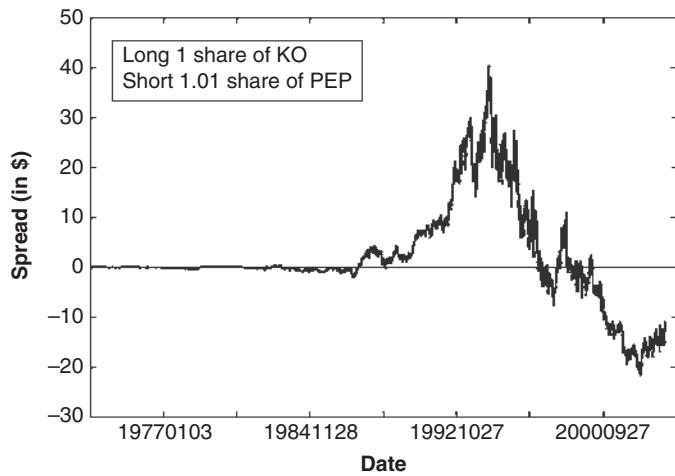


FIGURE 7.3 A nonstationary time series formed by the spread between KO and PEP.

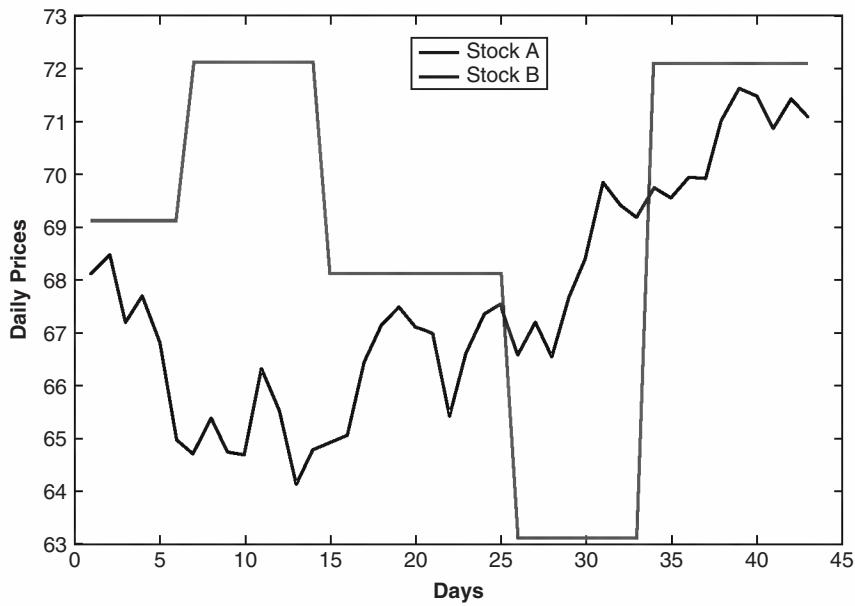


FIGURE 7.4 Cointegration is not correlation. Stocks A and B are cointegrated but not correlated.

Example 7.3: Testing the Cointegration versus Correlation Properties between KO and PEP

The cointegration test for KO and PEP is the same as that for GDX and GLD in Example 7.2, so it won't be repeated here. (It is available from epchan.com/book/example7_3.m.) The cointegration result shows that the t -statistic for the augmented Dicky-Fuller test is -2.14 , larger than the 10 percent critical value of -3.038 , meaning that there is a less than 90 percent probability that these two time series are cointegrated.

The following code fragment, however, tests for correlation between the two time series:

Using MATLAB

You can download the MATLAB code as example7_3.m.

```
% A test for correlation.  
dailyReturns=(adjcls-lag1(adjcls))./lag1(adjcls);  
[R,P]=corrcoef(dailyReturns(2:end,:));  
% R =  
%  
%      1.0000    0.4849  
%      0.4849    1.0000  
%  
%  
% P =  
%  
%      1          0  
%      0          1  
% The P value of 0 indicates that the two time series  
% are significantly correlated.
```

Using Python

You can download the Python Jupyter notebook as example7_3.ipynb.

```
How to Form a Good Cointegrating (and Mean-Reverting)  
Pair of Stocks  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from statsmodels.tsa.stattools import coint  
from statsmodels.api import OLS  
from scipy.stats import pearsonr  
df1=pd.read_excel('KO.xls')  
df2=pd.read_excel('PEP.xls')  
df=pd.merge(df1, df2, on='Date', suffixes=('_KO', '_PEP'))  
df.set_index('Date', inplace=True)  
df.sort_index(inplace=True)  
Run cointegration (Engle-Granger) test  
coint_t, pvalue, crit_value=coint(df['Adj Close_KO'],  
                                 df['Adj Close_PEP'])  
(coint_t, pvalue, crit_value) # abs(t-stat) < critical  
# value at 90%. pvalue says probability of null  
# hypothesis (of no cointegration) is 73%  
(-1.5815517041517178,  
 0.7286134576473527,  
 array([-3.89783854, -3.33691006, -3.04499143]))
```

```

Determine hedge ratio
model=OLS(df['Adj Close_KO'], df['Adj Close_PEP'])
results=model.fit()
hedgeRatio=results.params
hedgeRatio
Adj Close_PEP      1.011409
dtype: float64
spread = KO - hedgeRatio*PEP
spread=df['Adj Close_KO']-hedgeRatio[0]*df['Adj Close_PEP']
plt.plot(spread) # Figure 7.2
[<matplotlib.lines.Line2D at 0x2728e431b00>]
png
png
Correlation test
dailyret=df.loc[:, ('Adj Close_KO', 'Adj Close_PEP')].
pct_change()
dailyret.corr()
Adj Close_KO
Adj Close_PEP
Adj Close_KO
1.000000
0.484924
Adj Close_PEP
0.484924
1.000000
dailyret_clean=dailyret.dropna()
pearsonr(dailyret_clean.iloc[:,0], dailyret_clean.
    iloc[:,1]) # first output is correlation coefficient,
    second output is pvalue.
(0.4849239439370571, 0.0)

```

Using R

You can download the R code as example7_3.R.

```

# Need the zoo package for its na.locf function
# install.packages('zoo')
# Need the CADFtest package for its CADFtest function
# install.packages('CADFtest')
library('zoo')
library('CADFtest')
source('calculateReturns.R')

data1 <- read.delim("KO.txt") # Tab-delimited
data_sort1 <- data1[order(as.Date(data1[,1],
    '%m/%d/%Y')),] # sort in ascending order of dates
(1st column of data)

```

```

tday1 <- as.integer(format(as.Date(data_sort1[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls1 <- data_sort1[,ncol(data_sort1)]
data2 <- read.delim("PEP.txt") # Tab-delimited
data_sort2 <- data2[order(as.Date(data2[,1],
  '%m/%d/%Y')),] # sort in ascending order of dates
# (1st column of data)
tday2 <- as.integer(format(as.Date(data_sort2[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls2 <- data_sort2[,ncol(data_sort2)]
# find the intersection of the two data sets
tday <- intersect(tday1, tday2)
adjcls1 <- adjcls1[tday1 %in% tday]
adjcls2 <- adjcls2[tday2 %in% tday]
# CADFtest cannot have NaN values in input
adjcls1 <- zoo::na.locf(adjcls1)
adjcls2 <- zoo::na.locf(adjcls2)
mydata <- list(KO=adjcls1, PEP=adjcls2);
res <- CADFtest(model=KO~PEP, data=mydata, type =
  "drift", max.lag.X=1)
summary(res) # As the following input shows, p-value is
# about 0.16, hence we cannot reject null hypothesis.
# Covariate Augmented DF test
# CADF test
# t-test statistic: -2.2255225
# estimated rho^2: 0.8249085
# p-value: 0.1612782
# Max lag of the diff. dependent variable: 1.0000000
# Max lag of the stationary covariate(s): 1.0000000
# Max lead of the stationary covariate(s): 0.0000000
#
# Call:
#   dynlm(formula = formula(model), start = obs.1, end =
#     obs.T)
#
# Residuals:
#   Min    1Q Median    3Q   Max
# -4.7552 -0.0694 -0.0059  0.0576  4.5976
#
# Coefficients:
#   Estimate Std. Error t value Pr(>|t|)
# (Intercept)  0.0060546  0.0071439   0.848   0.397
# L(y, 1)      -0.0011457  0.0005148  -2.226   0.161
# L(d(y), 1)   0.0518074  0.0102210   5.069  4.1e-07 ***

```

```

#   L(X, 0)      0.5359345  0.0127566  42.012  < 2e-16 ***
#   L(X, 1)     -0.5348523  0.0127698 -41.884  < 2e-16 ***
#
#   ---
#   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 
#                   0.1 ' ' 1
#
# Residual standard error: 0.436 on 7828 degrees of freedom
# Multiple R-squared:  0.1852, Adjusted R-squared:  0.1848 
# F-statistic:  593 on 3 and 7828 DF,  p-value: < 2.2e-16

# determines the hedge ratio
lmresult <- lm(KO ~ 0 + PEP, mydata )
hedgeRatio <- coef(lmresult) # 1.011409
z <- residuals(lmresult) # The residuals should be
# stationary (mean-reverting)
plot(z) # This should produce a chart similar to Figure 7.3.
# A test for correlation
dailyReturns <- calculateReturns(cbind(adjcls1, adjcls2), 1)
result <- cor.test(dailyReturns[,1], dailyReturns[,2])
result # correlation coefficient is 0.4849239, with
# p-value < 2.2e-16, definitely correlated!
# Pearson's product-moment correlation
# data: dailyReturns[, 1] and dailyReturns[, 2]
# t = 49.0707, df = 7832, p-value < 2.2e-16
# alternative hypothesis: true correlation is not equal to 0
# 95 percent confidence interval:
#  0.4678028 0.5016813
# sample estimates:
#       cor
# 0.4849239

```

FACTOR MODELS

Example 7.4: Principal Component Analysis as an Example of the Factor Model

The examples of factor exposures I described above are typically economic (e.g., outperformance of value stocks), fundamental (e.g., book-to-price ratio), or technical (e.g., previous period's return). However, there is one kind of factor model that relies on nothing more than historical returns to construct. These are the so-called statistical factors, obtained using methods such as the principal component analysis (PCA).

If we use PCA to construct the statistical factor exposures and factor returns, we must assume that the factor exposures are constant (time independent) over the estimation period. (This rules out factors that represent mean reversion or momentum, since these factor exposures depend on the prior period returns.) In a sense, this is more similar to the time-series factors such as SMB than the cross sectional factors such as P/E, because

the factor exposures of time-series factors are also assumed to be constant over a long lookback period. However, unlike time-series factors, the statistical factors are unobservable, and unlike cross-sectional factor exposures, the statistical factor exposures are also unobservable. More importantly, we assume that the factor returns are *uncorrelated*; that is to say, their covariance matrix $\langle bb^T \rangle$ is diagonal. If we use the eigenvectors of the covariance matrix $\langle RR^T \rangle$ as the columns of the matrix \mathbf{X} in the APT equation $R = \mathbf{X} b + u$, we will find via elementary linear algebra that $\langle bb^T \rangle$ is indeed diagonal; and furthermore, the eigenvalues of $\langle RR^T \rangle$ are none other than the variances of the factor returns b . But of course, there is no point to use factor analysis if the number of factors is the same as the number of stocks—typically, we can just pick the eigenvectors with the top few eigenvalues to form the matrix \mathbf{X} . The number of eigenvectors to pick is a parameter that you can adjust to optimize your trading model.

In the following programs, I illustrate a possible trading strategy applying PCA to S&P 600 small-cap stocks. It is a strategy based on the assumption that factor returns have momentum: They remain constant from the current time period to the next. Hence, we can buy the stocks with the highest expected returns based on these factors, and short the ones with the lowest expected returns. The average annualized return of this strategy is only 2% (MATLAB) to 4% (Python and R), and only when we assume no transaction costs. (The difference in returns among the programs are essentially round off errors.)

Using MATLAB

You can download the MATLAB code as `example7_4.m`.

```
clear;
lookback=252; % use lookback days as estimation
    (training) period for determining factor exposures.
numFactors=5;
topN=50; % for trading strategy, long stocks with topN
    expected 1-day returns.
onewaycost=0/10000;

load('IJR_20080114.mat');
% test on SP600 smallcap stocks. (This MATLAB binary input
file contains tday, stocks, op, hi, lo, cl arrays.

mycls=fillMissingData(cl);
positionsTable=zeros(size(cl));

dailyret=(mycls-backshift(1, mycls))./backshift(1, mycls);
    % note the rows of dailyret are the observations at
different time periods
```

```

end_index = length(tday);

for t=lookback+2:end_index

R=dailyret(t-lookback:t-1,:); % here the columns of
% R are the different observations.

hasData=find(all(isfinite(R), 2)); % avoid any
stocks with missing returns

R=R(hasData, :);
[PCALoadings, PCAScores, PCAVar] = pca(R);
X = PCAScores(:,1:numFactors);
y = dailyret(t, hasData)';
Xreg = [ones(size(X, 1), 1) X];
[b,sigma]=mvregress(Xreg,y);
pred = Xreg*b;
Rexp=sum(pred,2); % Rexp is the expected return for
next period assuming factor returns remain constant.
[foo idxSort]=sort(Rexp, 'ascend');

positionsTable(t, hasData(idxSort(1:topN)))=-1; %
short topN stocks with lowest expected returns
positionsTable(t, hasData(idxSort(end-
topN+1:end)))=1; % buy topN stocks with highest
expected returns
end
ret=smartsum(backshift(1, positionsTable).*dailyret-
onewaycost*abs(positionsTable-backshift(1, positionsTa-
ble)), 2)/smartsum(abs(backshift(1, positionsTable)),
2); % compute daily returns of trading strategy
fprintf(1, 'AvgAnnRet=%f Sharpe=%f\n',
smartmean(ret,1)*252, sqrt(252)*smartmean(ret,1) /
smartstd(ret,1));
% AvgAnnRet=0.020205 Sharpe=0.211120

```

This program made use of a function mvregress for multivariate linear regression with possible missing or NaN values in the input matrix. Using this function, the computation time is under a minute. Otherwise, it may take hours.

Using Python

You can download the Python code as example7_4.py.

```

# Principal Component Analysis as an Example of Factor
Model
import math
import numpy as np

```

```

import pandas as pd
from numpy.linalg import eig
from numpy.linalg import eigh
#from statsmodels.api import OLS
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.linear_model import Ridge
import time
lookback=252 # training period for factor exposure
numFactors=5
topN=50 # for trading strategy, long stocks with topN
    exepcted 1-day returns
df=pd.read_table('IJR_20080114.txt')
df['Date']=df['Date'].astype('int')
df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
df.fillna(method='ffill', inplace=True)
dailyret=df.pct_change() # note the rows of dailyret are
    the observations at different time periods
positionsTable=np.zeros(df.shape)
end_index = df.shape[0]
#end_index = lookback + 10
for t in np.arange(lookback+1,end_index):
    R=dailyret.iloc[t-lookback+1:t,:].T # here the
        columns of R are the different observations.
    hasData=np.where(R.notna().all(axis=1))[0]
    R.dropna(inplace=True) # avoid any stocks with
        missing returns
    pca = PCA()
    X = pca.fit_transform(R.T)[:, :numFactors]
    X = sm.add_constant(X)
    y1 = R.T
    clf = MultiOutputRegressor(LinearRegression(fit_
        intercept=False),n_jobs=4).fit(X, y1)
    Rexp = np.sum(clf.predict(X),axis=0)
    R=dailyret.iloc[t-lookback+1:t+1,:].T # here the
        columns of R are the different observations.
    idxSort=Rexp.argsort()
    positionsTable[t, hasData[idxSort[np.arange(0,
        topN)]]]=-1
    # positionsTable[t, hasData[idxSort[np.arange(-
        topN,0)]]]=1

```

```

        positionsTable[t, hasData[idxSort[np.arange(-topN,
-1)]]] = 1
capital=np.nansum(np.array(abs(pd.
DataFrame(positionsTable)).shift(), axis=1)
positionsTable[capital==0,]=0
capital[capital==0]=1
ret=np.nansum(np.array(pd.DataFrame(positionsTable) .
shift())*np.array(dailyret), axis=1)/capital
avgret=np.nanmean(ret)*252
avgstdev = np.nanstd(ret)*math.sqrt(252)
Sharpe = avgret/avgstdev
print(avgret)
print(avgstdev)
print(Sharpe)
#0.04052422056844459
#0.07002908500498846
#0.5786769963588398

```

Using R

You can download the R code as example7_4.R.

```

rm(list=ls()) # clear workspace
backshift <- function(mylag, x) {
  rbind(matrix(NaN, mylag, ncol(x)),
  as.matrix(x[1:(nrow(x)-mylag),]))
}

#install.packages('pls')
library("pls")
library('zoo')
source('calculateReturns.R')
#source('backshift.R')
lookback <- 252 # use lookback days as estimation
  (training) period for determining factor exposures.
numFactors <- 5
topN <- 50 # for trading strategy, long stocks with topN
  expected 1-day returns.
data1 <- read.csv("IJR_20080114.csv") # Tab-delimited
cl <- data.matrix(data1[, 2:ncol(data1)])
cl[ is.nan(cl) ] <- NA
tday <- data.matrix(data1[, 1])
mycls <- na.fill(cl, type="locf", nan=NA, fill=NA)
end_loop <- nrow(mycls)
positionsTable <- matrix(0, end_loop, ncol(mycls))
dailyret <- calculateReturns(mycls, 1)
dailyret[is.nan(dailyret)] <- 0

```

```

dailyret <- dailyret[1:end_loop]
for (it in (lookback+2):end_loop) {
  R <- dailyret[(it-lookback+2):it,]
  hasData <- which(complete.cases(t(R)))
  R <- R[, hasData]
  PCA <- prcomp(t(R))
  X <- t(PCA$x[1:numFactors,])
  Rexp <- rep(0,ncol(R))
  for (s in 1:ncol(R)){
    reg_result <- lm(R[,s] ~ X)
    pred <- predict(reg_result)
    pred[is.nan(pred)] <- 0
    Rexp[s] <- sum(pred)
  }
  result <- sort(Rexp, index.return=TRUE)

  positionsTable[it, hasData[result$ix[1:topN]] ] = -1
  positionsTable[it, hasData[result$ix[(length(result$ix)-topN-1):length(result$ix)]] ] = 1
}

capital <- rowSums(abs(backshift(1, positionsTable)),
na.rm = TRUE, dims = 1)
ret <- rowSums(backshift(1, positionsTable)*dailyret,
na.rm = TRUE, dims = 1)/capital
avgret <- 252*mean(ret, na.rm = TRUE)
avgstd <- sqrt(252)*sd(ret, na.rm = TRUE)
Sharpe = avgret/avgstd
print(avgret)
print(avgstd)
print(Sharpe)
#0.04052422056844459
#0.07002908500498846
#0.5786769963588398

```

WHAT IS YOUR EXIT STRATEGY?

Example 7.5: Calculation of the Half-Life of a Mean-Reverting Time Series

We can use the mean-reverting spread between GLD and GDX in Example 7.2 to illustrate the calculation of the half-life of its mean reversion.

Using MATLAB

The MATLAB code is available as `example7_5.m`. (The first part of the program is the same as `example7_2.m`.)

```
% === Insert example7_2.m in the beginning here ===
prevz=backshift(1, z); % z at a previous time-step
dz=z-prevz;
dz(1)=[] ;
prevz(1)=[] ;
% assumes dz=theta*(z-mean(z))dt+w,
% where w is error term
results=ols(dz, prevz-mean(prevz));theta=results.beta;
halflife=-log(2)/theta
% halflife =
%
% 7.8390
```

The program finds that the half-life for mean reversion of the GLD-GDX is about 10 days, which is approximately how long you should expect to hold this spread before it becomes profitable.

Using Python

The Python code is available as `example7_5.ipynb`. (The first part of the program is the same as `example7_2.ipynb`.)

```
Calculation of the Half-Life of a Mean-Reverting Time
Series
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import coint
from statsmodels.api import OLS
df1=pd.read_excel('GLD.xls')
df2=pd.read_excel('GDX.xls')
df=pd.merge(df1, df2, on='Date', suffixes=('_GLD', '_GDX'))
```

```

df.set_index('Date', inplace=True)
df.sort_index(inplace=True)
Run cointegration (Engle-Granger) test
coint_t, pvalue, crit_value=coint(df['Adj Close_GLD'],
    df['Adj Close_GDX'])
(coint_t, pvalue, crit_value) # abs(t-stat) > critical
    value at 95%. pvalue says probability of null hypo-
    thesis (of no cointegration) is only 1.8%
(-3.6981160763300593,
    0.018427835409537425,
    array([-3.92518794, -3.35208799, -3.05551324]))
Determine hedge ratio
model=OLS(df['Adj Close_GLD'], df['Adj Close_GDX'])
results=model.fit()
hedgeRatio=results.params
hedgeRatio
Adj Close_GDX      1.639523
    dtype: float64
z = GLD - hedgeRatio*GDX
z=df['Adj Close_GLD']-hedgeRatio[0]*df['Adj Close_GDX']
plt.plot(z)
prevz=z.shift()
dz=z-prevz
dz=dz[1:,]
prevz=prevz[1:,]
model2=OLS(dz, prevz-np.mean(prevz))
results2=model2.fit()
theta=results2.params
theta
x1      -0.088423
    dtype: float64
halflife=-np.log(2)/theta
halflife
x1      7.839031
    dtype: float64

```

Using R

The R code is available as example7_5.R. (The first part of the program is the same as example7_2.R.)

```

source(backshift.R')
data1 <- read.delim("GLD.txt") # Tab-delimited
data_sort1 <- data1[order(as.Date(data1[,1],
    '%m/%d/%Y')),] # sort in ascending order of dates
    (1st column of data)
tday1 <- as.integer(format(as.Date(data_sort1[,1],
    '%m/%d/%Y'), '%Y%m%d'))

```

```

adjcls1 <- data_sort1[,ncol(data_sort1)]
data2 <- read.delim("GDX.txt") # Tab-delimited
data_sort2 <- data2[order(as.Date(data2[,1],
  '%m/%d/%Y'))] # sort in ascending order of dates (1st
  column of data)
tday2 <- as.integer(format(as.Date(data_sort2[,1],
  '%m/%d/%Y'), '%Y%m%d'))
adjcls2 <- data_sort2[,ncol(data_sort2)]
# find the intersection of the two data sets
tday <- intersect(tday1, tday2)
adjcls1 <- adjcls1[tday1 %in% tday]
adjcls2 <- adjcls2[tday2 %in% tday]
# determines the hedge ratio on the trainset
result <- lm(adjcls1 ~ 0 + adjcls2 )
hedgeRatio <- coef(result) # 1.64
spread <- adjcls1-hedgeRatio*adjcls2 # spread = GLD -
  hedgeRatio*GDX
prevSpread <- backshift(1, as.matrix(spread))
prevSpread <- prevSpread - mean(prevSpread, na.rm = TRUE)
deltaSpread <- c(NaN, diff(spread)) # Change in spread
  from t-1 to t
result2 <- lm(deltaSpread ~ 0 + prevSpread )
theta <- coef(result2)
halflife <- -log(2)/theta # 7.839031

```

SEASONAL TRADING STRATEGIES

Example 7.6: Backtesting the January Effect

Here are the codes to compute the returns of a strategy applied to S&P 600 small-cap stocks based on the January effect.

Using MATLAB

The MATLAB codes can be found at epchan.com/book/example7_6.m, and the input data is also available there.

```

clear;
load('IJR_20080131');
onewaycost=0.0005; % 5bp one way transaction cost
years=year(datetime(tday, 'ConvertFrom', 'yyyymmdd'));
months=month(datetime(tday, 'ConvertFrom', 'yyyymmdd'));
nextdayyear=fwdshift(1, years);

```

```

nextdaymonth=fwdshift(1, months);
lastdayofDec=find(months==12 & nextdaymonth==1);
lastdayofJan=find(months==1 & nextdaymonth==2);
% lastdayofDec starts in 2004,
% so remove 2004 from lastdayofJan
lastdayofJan(1)=[];% Ensure each lastdayofJan date
after each
% lastdayofDec date
assert(all(tday(lastdayofJan) > tday(lastdayofDec)));
eoy=find(years~=nextdayyear); % End Of Year indices
eoy(end)=[]; % last index is not End of Year
% Ensure eoy dates match lastdayofDec dates
assert(all(tday(eoy)==tday(lastdayofDec)));
annret=..
(c1(eoy(2:end),:)-c1(eoy(1:end-1),:))/...
c1(eoy(1:end-1),:); % annual returns
janret=..
(c1(lastdayofJan(2:end),:)-
c1(lastdayofDec(2:end),:))/c1(lastdayofDec(2:end),:);
% January returns
for y=1:size(annret, 1)
    % pick those stocks with valid annual returns
    hasData=..
    find(isfinite(annret(y,:)));
    % sort stocks based on prior year's returns
    [foo sortidx]=sort(annret(y, hasData), 'ascend');
    % buy stocks with lowest decile of returns,
    % and vice versa for highest decile
    topN=round(length(hasData)/10);
    % portfolio returns
    portRet=..
    (smartmean(janret(y, hasData(sortidx(1:topN))))-...
    smartmean(janret(y, hasData(..%
    sortidx(end-topN+1:end))))) / 2 - 2 * onewaytcost;
    fprintf(1,'Last holding date %i: Portfolio
    return=%7.4f\n', tday(lastdayofJan(y+1)), portRet);
end
% These should be the output
% Last holding date 20060131: Portfolio return=-0.0244
% Last holding date 20070131: Portfolio return=-0.0068
% Last holding date 20080131: Portfolio return= 0.0881

```

This program uses a number of utility programs. The first one is the assert function, which is very useful for ensuring the program is working as expected.

```

function assert(pred, str)
% ASSERT Raise an error if the predicate is not true.
% assert(pred, string)
if nargin<2, str = ''; end
if ~pred
    s = sprintf('assertion violated: %s', str);
    error(s);
end

```

The second one is the fwdshift function, which works in the opposite way to the lag1 function: It shifts the time series one step forward.

```

function y=fwdshift(day,x)
assert(day>=0);
y=[x(day+1:end,:,:); ...
NaN*ones(day,size(x,2), size(x, 3))];

```

Using Python

The Python codes can be found at epchan.com/book/example7_6.py, and the input data is also available there.

```

# Backtesting the January Effect
import numpy as np
import pandas as pd
onewaytcost=0.0005
df=pd.read_table('IJR_20080131.txt')
df['Date']=df['Date'].round().astype('int')
df['Date']=pd.to_datetime(df['Date'], format='%Y%m%d')
df.set_index('Date', inplace=True)
eoyPrice=df.resample('Y').last()[0:-1] # End of December prices. Need to remove last date because it isn't really end of year
annret=eoyPrice.pct_change().iloc[1:,:] # first row has NaN
eojPrice=df.resample('BA-JAN').last()[1:-1] # End of January prices. Need to remove first date to match the years in lastdayofDec. Need to remove last date because it isn't really end of January.
janret=(eojPrice.values-eoyPrice.values)/eoyPrice.values
janret=janret[1:,:] # match number of rows in annret
for y in range(len(annret)):
    hasData=np.where(np.isfinite(annret.iloc[y, :]))[0]
    sortidx=np.argsort(annret.iloc[y, hasData])
    topN=np.round(len(hasData)/10)
    portRet=(np.nanmean(janret[y, hasData[sortidx].iloc[np.arange(0, topN)]])-np.nanmean(janret[y, hasData[sortidx].iloc[np.arange(-topN+1, -1)]]))/2-2*onewaytcost # portfolio returns

```

```

        print("Last holding date %s: Portfolio return=%f" %
(eojPrice.index[y+1], portRet))
#Last holding date 2006-01-31 00:00:00: Portfolio
return=-0.023853
#Last holding date 2007-01-31 00:00:00: Portfolio
return=-0.003641
#Last holding date 2008-01-31 00:00:00: Portfolio
return=0.088486

```

Using R

The R codes can be found at epchan.com/book/example7_6.R, and the input data is also available there.

```

# Need the lubridate package for its dates handling
# install.packages('lubridate')
library('lubridate')
source('calculateReturns.R')
source('fwdshift.R')
onewaytcost <- 5/10000 # 5bps one way transaction cost
data1 <- read.delim("IJR_20080131.txt") # Tab-delimited
cl <- data.matrix(data1[, 2:ncol(data1)])
tday <- ymd(data.matrix(data1[, 1])) # dates in lub-
    ridate format
years <- year(tday)
months <- month(tday)
years <- as.matrix(years, length(years), 1)
months <- as.matrix(months, length(months), 1)
nextdayyear <- fwdshift(1, years)
nextdaymonth <- fwdshift(1, months)

eom <- which(months!=nextdaymonth) # End of month indices.
eoy <- which(years!=nextdayyear) # End Of Year indices.
Note that in R, 2008!=NaN returns FALSE whereas in
Matlab 2008~=NaN returns TRUE
annret <- calculateReturns(cl[eoy,], 1) # annual returns
annret <- annret[-1,]
monret <- calculateReturns(cl[eom,], 1) # monthly returns
janret <- monret[months[eom]==1,] # January returns
janret <- janret[-(1:2),] # First January does not have
    preceding year
exitDay <- tday[months==1 & nextdaymonth==2] # Last day
    of Janurary
exitDay <- exitDay[-(c(1))] # Exclude first January
for (y in 1:nrow(annret)) {
    hasData <- which(is.finite(annret[y,])) # pick those
        stocks with valid annual returns

```

```

sortidx <- order(annret[y, hasData]) # sort stocks
based on prior year's returns
topN <- round(length(hasData)/10) # buy stocks with
lowest decile of returns, and vice versa for highest
decile
portRet <- (sum(janret[y,
hasData[sortidx[1:topN]]], na.rm=TRUE) -
sum(janret[y, hasData[sortidx[(length(sortidx)-
topN+1):length(sortidx)]]], na.rm=TRUE))/2/
topN-2*onewaytcost # portfolio returns
msg <- sprintf('Last holding date %s: Portfolio
return=%7.4f\n', as.character(exitDay[y+1]), portRet)
cat(msg)
}
# Last holding date 2006-01-31: Portfolio return=-0.0244
# Last holding date 2007-01-31: Portfolio return=-0.0068
# Last holding date 2008-01-31: Portfolio return= 0.0881

```

Example 7.7: Backtesting a Year-on-Year Seasonal Trending Strategy

Here are the codes for the year-on-year seasonal trending strategy I quoted earlier. Note that the data contains survivorship bias, as it is based on the S&P 500 index on November 23, 2007.

Using MATLAB

The source code can be downloaded from epchan.com/book/example7_7.m.
The data is also available at that site.

```
%  
% written by:  
clear;  
  
load('SPX_20071123', 'tday', 'stocks', 'cl');  
  
monthEnds=find(isLastTradingDayOfMonth(tday)); % find  
the indices of the days that are at month ends.  
tday=tday(monthEnds);  
cl=cl(monthEnds, :);  
  
monthlyRet=(cl-lag1(cl))./lag1(cl);  
  
positions=zeros(size(monthlyRet));  
  
for m=14:size(monthlyRet, 1)  
[monthlyRetSorted sortIndex]=sort(monthlyRet(m-12, :));  
badData=find(~isfinite(monthlyRet(m-12, :)) |  
~isfinite(cl(monthEnds(m-1), :)));  
sortIndex=setdiff(sortIndex, badData, 'stable');  
topN=floor(length(sortIndex)/10); % take top decile  
of stocks as longs, bottom decile as shorts  
positions(m-1, sortIndex(1:topN))=-1;  
positions(m-1, sortIndex(end-topN+1:end))=1;  
end  
  
ret=smartsum(lag1(positions).*monthlyRet, 2)./  
smartsum(abs(lag1(positions)), 2);  
ret(1:13)=[];  
  
avgannret=12*smartmean(ret);  
sharpe=sqrt(12)*smartmean(ret)/smartstd(ret);  
  
fprintf(1, 'Avg ann return=%7.4f Sharpe ratio=%7.4f\n',  
avgannret, sharpe);  
% Output should be  
% Avg ann return=-0.0129 Sharpe ratio=-0.1243
```

This program contains a few utility functions. The first one is `LastTradingDayOfMonth`, which returns a logical array of 1s and 0s, indicating whether a month in a trading-date array is the last trading day of a month.

```

function isLastTradingDayOfMonth=..
isLastTradingDayOfMonth(tday)
% isLastTradingDayOfMonth=
% isLastTradingDayOfMonth(tday) returns a logical
% array. True if tday(t) is last trading day of month.
tdayStr=datestr(datenum(num2str(tday)), 'yyyymmdd'));
todayMonth=month(tdayStr);
tmrMonth=fwdshift(1, todayMonth); % tomorrow's month
isLastTradingDayOfMonth=false(size(tday));
isLastTradingDayOfMonth(todayMonth~=tmrMonth & ..
isfinite(todayMonth) & isfinite(tmrMonth))=true;

```

Another is the backshift function, which is like the lag1 function except that one can shift any arbitrary number of periods instead of just 1.

```

function y=backshift(day,x)
% y=backshift(day,x)
assert(day>=0);
y=[NaN(day,size(x,2), size(x, 3));x(1:end-day,:,:)];

```

You can try the most recent five years instead of the entire data period, and you will find that the average returns are even worse.

Using Python

```

# Backtesting a Year-on-Year Seasonal Trending Strategy
import numpy as np
import pandas as pd
df=pd.read_table('SPX_20071123.txt')
df['Date']=df['Date'].round().astype('int')
df['Date']=pd.to_datetime(df['Date'], format='%Y%m%d')
df.set_index('Date', inplace=True)
eomPrice=df.resample('M').last()[:-1] # End of month
prices. Need to remove last date because it isn't
really end of January.
monthlyRet=eomPrice.pct_change(1, fill_method=None)
positions=np.zeros(monthlyRet.shape)
for m in range(13, monthlyRet.shape[0]):
    hasData=np.where(np.isfinite(monthlyRet.iloc[m-12,
:]))[0]
    sortidx=np.argsort(monthlyRet.iloc[m-12, hasData])
    badData=np.where(np.logical_not(np.
isfinite(monthlyRet.iloc[m-1, hasData[sortidx]])))
[0] # these are indices
    sortidx.drop(sortidx.index[badData], inplace=True)
    topN=np.floor(len(sortidx)/10).astype('int')

```

```

    positions[m-1, hasData[sortidx.values[np.arange(0,
topN)]]]=-1
    positions[m-1, hasData[sortidx.values[np.arange(-
topN,0)]]]=1
capital=np.nansum(np.array(pd.DataFrame(abs(positions)) .
shift()), axis=1)
capital[capital==0]=1
ret=np.nanmean(np.array(pd.DataFrame(positions) .
shift())*np.array(monthlyRet), axis=1)/capital
ret=np.delete(ret, np.arange(13))
avgret=np.nanmean(ret)*12
sharpe=np.sqrt(12)*np.nanmean(ret)/np.nanstd(ret)
print('Avg ann return=%f Sharpe ratio=%f' % (avgret,
sharpe))
#Avg ann return=-0.012679 Sharpe ratio=-0.122247

```

Using R

The source code can be downloaded as example7_7.R.

```

# Need the lubridate package for its dates handling
install.packages('lubridate')
library('lubridate')
source('calculateReturns.R')
source('backshift.R')
source('fwdshift.R')
data1 <- read.delim("SPX_20071123.txt") # Tab-delimited
cl <- data.matrix(data1[, 2:ncol(data1)])
tday <- ymd(data.matrix(data1[, 1])) # dates in lub-
ridate format
years <- year(tday)
months <- month(tday)
years <- as.matrix(years, length(years), 1)
months <- as.matrix(months, length(months), 1)
nextdaymonth <- fwdshift(1, months)
eom <- which(months!=nextdaymonth) # End of month
indices.
monret <- calculateReturns(cl[eom, ], 1) # monthly
returns
positions <- matrix(0, nrow(monret), ncol(monret))
for (m in 14:nrow(monret)) {
  prevYearSortIdx <- order(monret[m-12, ], na.last = NA)
  prevYearSortIdx <- setdiff(prevYearSortIdx, which(!is.
finite(cl[eom[m-1], ]))) # Note setdiff in R does
not re-sort data. It is equivalent to setdiff(x, y,
'stable') in Matlab

```

```

topN <- round(length(prevYearSortIdx)/10) # buy stocks
  with top decile of returns, and sell stocks of bot-
  tom decile
positions[m-1, prevYearSortIdx[1:topN]] <- -1
positions[m-1, prevYearSortIdx[(length(prevYearSort
  Idx)-topN+1):length(prevYearSortIdx)]] <- 1
}
ret <- rowSums(backshift(1, positions)*monret, na.rm
  = TRUE)/rowSums(abs(backshift(1, positions)),
  na.rm=TRUE)
ret <- ret[-(1:13)]
avgannret <- 12*mean(ret, na.rm = TRUE)
avgannret # -0.01139674
sharpe <- sqrt(12)*mean(ret, na.rm = TRUE)/sd(ret,
  na.rm=TRUE)
sharpe # -0.1095098

```

A SEASONAL TRADE IN GASOLINE FUTURES

Year	P&L in \$	Maximum Drawdown in \$
1995	1,037	0
1996	1,638	-2,226
1997	227	-664
1998	118	0
1999	197	-588
2000	735	-1,198
2001	1,562	-304
2002	315	-935
2003	1,449	-2,300
2004	361	-1,819
2005	6,985	-830
2006	890	-4,150
2007*	4,322	-5,279
2008*	9,740	-1,156
2009*	-890	-4,167
2010*	1,840	-3,251
2011*	3,381	-2,298
2012*	-7,997	-8,742
2013*	2,276	-2,573
2014*	1,541	-814
2015*	8,539	-1,753

* Out-of-sample results.

A SEASONAL TRADE IN NATURAL GAS FUTURES

Year	P&L in \$	Maximum Drawdown in \$
1995	1,970	0
1996	3,090	-630
1997	450	-430
1998	2,150	-1,420
1999	4,340	-370
2000	4,360	0
2001	2,730	-1,650
2002	9,860	0
2003	2,000	-5,550
2004	5,430	0
2005	2,380	-230
2006	2,250	-1,750
2007	800	-7,470
2008*	10,137	-1,604
2009*	-4,240	-8,013
2010*	-8,360	-10,657
2011*	1,310	-3,874
2012*	-7,180	-8,070
2013*	5,950	-1,219
2014*	40	-3,168
2015*	-2,770	-4,325
2016*	530	-1,166

* Out-of-sample results.

SUMMARY

This book has been largely about a particular type of quantitative trading called *statistical arbitrage* in the investment industry. Despite this fancy name, statistical arbitrage is actually far simpler than trading derivatives (e.g., options) or fixed-income instruments, both conceptually and mathematically. I have described a large part of the statistical arbitrageur's standard arsenal: mean reversion and momentum, regime switching, stationarity and cointegration, arbitrage pricing theory or factor model, seasonal trading models, and, finally, high-frequency trading.

Some of the important points to note can be summarized here:

- Mean-reverting regimes are more prevalent than trending regimes.
- There are some tricky data issues involved with backtesting mean-reversion strategies: Outlier quotes and survivorship bias are among them.
- Trending regimes are usually triggered by the diffusion of new information, the execution of a large institutional order, or "herding" behavior.
- Competition between traders tends to reduce the number of mean-reverting trading opportunities.
- Competition between traders tends to reduce the optimal holding period of a momentum trade.
- Trading parameters for each day or even each trade can be optimized using a machine-learning-based method we called CPO.
- A stationary price series is ideal for a mean-reversion trade.
- Two or more nonstationary price series can be combined to form a stationary one if they are "cointegrating."
- Cointegration and correlation are different things: Cointegration is about the long-term behavior of the *prices* of two or more stocks, while correlation is about the short-term behavior of their *returns*.
- Factor models, or arbitrage pricing theory, are commonly used for modeling how fundamental factors affect stock returns linearly.

- One of the most well-known factor models is the Fama-French Three-Factor model, which postulates that stock returns are proportional to their beta and book-to-price ratio, and negatively to their market capitalizations.
- Factor models typically have a relatively long holding period and long drawdowns due to regime switches.
- Exit signals should be created differently for mean-reversion versus momentum strategies.
- Estimation of the optimal holding period of a mean-reverting strategy can be quite robust, due to the Ornstein-Uhlenbeck formula.
- Estimation of the optimal holding period of a momentum strategy can be error prone due to the small number of signals.
- Stop loss can be suitable for momentum strategies but not reversal strategies.
- Seasonal trading strategies for stocks (i.e., calendar effect) have become unprofitable in recent years.
- Seasonal trading strategies for commodity futures continue to be profitable.
- High-frequency trading strategies rely on the “law of large numbers” for their high Sharpe ratios.
- High-frequency trading strategies typically generate the highest long-term compounded growth due to their high Sharpe ratios.
- High-frequency trading strategies are very difficult to backtest and very technology-reliant for their execution.
- Holding a highly leveraged portfolio of low-beta stocks should generate higher long-term compounded growth than holding an unleveraged portfolio of high-beta stocks.

Most statistical arbitrage trading strategies are some combination of these effects or models: Whether they are profitable or not is more of an issue of where and when to apply them than whether they are theoretically correct.

REFERENCES

- Alexander, Carol. 2001. *Market Models: A Guide to Financial Data Analysis*. West Sussex: John Wiley & Sons Ltd.
- Chan, Ernest. 2013. *Algorithmic Trading: Winning Strategies and Their Rationale*. Wiley.
- Economist*. 2007a. "This Year's Model." December 13. www.economist.com/finance/displaystory.cfm?story_id=10286619.
- Fama, Eugene, and Kenneth French. 1992. "The Cross-Section of Expected Stock Returns." *Journal of Finance* XLVII (2): 427–465.
- Fielden, Sandy. 2006. "Seasonal Surprises." *Energy Risk*, September. <https://www.risk.net/infrastructure/1523742/seasonal-surprises>.
- Grinold, Richard, and Ronald Kahn. 1999. *Active Portfolio Management*. New York: McGraw-Hill.
- Heston, Steven, and Ronnie Sadka. 2007. "Seasonality in the Cross-Section of Expected Stock Returns." AFA 2006 Boston Meetings Paper, July. lcb1.uoregon.edu/rcg/seminars/seasonal072604.pdf.
- Khandani, Amir, and Andrew Lo. 2007. "What Happened to the Quants in August 2007?" Preprint. web.mit.edu/alo/www/Papers/august07.pdf.
- Kochkodin, Brandon. 2021. "How WallStreetBets Pushed GameStop Shares to the Moon." www.bloomberg.com/news/articles/2021-01-25/how-wallstreetbetspushed-gamestop-shares-to-the-moon?sref=MqSE4VuP.
- Lewis, Michael. 2014. *Flash Boys*. W.W. Norton.
- Phillips, Daniel. 2020. "Investment Strategy Commentary: Value Stocks: Trapped or Spring-Loaded?" Northern Trust, September 24. <https://www.northerntrust.com/canada/insights-research/2020/investment-management/value-stocks>.
- Singal, Vijay. 2006. *Beyond the Random Walk*. Oxford University Press, USA.
- Schiller, Robert. 2008. "Economic View; How a Bubble Stayed under the Radar." *New York Times*, March 2. www.nytimes.com/2008/03/02/business/02view.html?ex=1362286800&en=da9e48989b6f937a&ei=5124&p=artner=permalink&exprod=permalink.
- Toepke, Jerry. 2004. "Fill 'Er Up! Benefit from Seasonal Price Patterns in Energy Futures." *Stocks, Futures and Options Magazine* (3) (March 3). www.sfonmag.com/issuedetail.asp?MonthNameID=March&yearID=2004.
- Uhlenbeck, George, and Leonard Ornstein. 1930. "On the Theory of Brownian Motion." *Physical Review* 36: 823–841.

CHAPTER 8

Conclusion

Can Independent Traders Succeed?

REFERENCES

- Clark, Nicola. 2008. "French Bank Says Its Controls Failed for 2 Years." *New York Times*, February 21. <http://www.nytimes.com/2008/02/21/business/worldbusiness/21bank.html?ex=1361336400&en=cf84f3776a877eac&ei=5124&partner=permalink&exprod=permalink>.

APPENDIX

A Quick Survey of MATLAB

MATLAB is a general-purpose software package developed by Mathworks, Inc., which is used by many institutional quantitative researchers and traders as their platform for backtesting, particularly those who work in statistical arbitrage. In Chapter 3, I introduced this platform and compared its pros and cons with some other alternatives. Most of the strategy examples in this book are written in MATLAB. Many of those strategies are portfolio-trading strategies involving hundreds of stocks that are very difficult to backtest in Excel. Here, I will provide a quick survey of MATLAB for those traders who are unfamiliar with the language, so they can see if it is worthwhile for them to invest in acquiring and learning to use this platform for their own backtesting.

MATLAB is not only a programming language; it is also an integrated development platform that includes a very user-friendly program editor and debugger. It is an interpreted language, meaning that it's similar to Visual Basic, but unlike a conventional programming language like C, it does not need to be compiled before it can be run. Yet it is much more flexible and powerful for backtesting than using Excel or Visual Basic because of the large number of built-in functions useful for mathematical computations, and because it is an array-processing language that is specially designed to make computations on arrays (i.e., vectors or matrices) simply

and quickly. In particular, many loops that are necessary in C or Visual Basic can be replaced by just one line of code in MATLAB. It also includes extensive text-processing facilities such that it is useful as a powerful tool for parsing and analyzing texts (such as web pages). Furthermore, it has a comprehensive graphics library that enables easy plotting of many types of graphs, even animations. (Many of the figures and charts in this book are created using MATLAB.) Finally, MATLAB codes can be compiled into C or C++ executables that can run on computers without the MATLAB platform installed. In fact, there is third-party software that can convert MATLAB code into C source codes, too.

The basic syntax of MATLAB is very similar to Visual Basic or C. For example, we can initialize the elements of an array x like this:

```
x(1)=0.1;
x(2)=0.3;
% 3 elements of an array initialized.
% This is by default a row-vector
x(3)=0.2;
```

Note that we don't need to first "declare" this array, nor do we need to tell MATLAB its expected size beforehand. If you leave out the ";" sign, MATLAB will print out the result of the content of the variable being assigned a value. Any comments can be written after the "%" sign. If you wish, you can initialize a large number of elements en masse to a common value:

```
% assigning the value 0.8 to all elements of a 3-vector y.
This is a row-vector.y=0.8*ones(1, 3)
```

Now if you want to do a vector addition of the two vectors, you can do it the old-fashioned way (just as you would in C), that is, using a loop:

```
for i=1:3
z(i)=x(i)+y(i) % z is [0.9 1.1 1]
end
```

But the power of MATLAB is that it can handle many array operations in parallel very succinctly, without using loops. (That's why

it is called a vector-processing language.) So instead of the previous loop, you can just write

```
z=x+y % z is the same [0.9 1.1 1]
```

Even more powerful, you can easily select part of the different arrays and operate on them. What do you think would be the results of the following?

```
w=x([1 3])+z([2 1])
```

$x([1 3])$ selected the first and third elements of x , so $x([1 3])$ is just [0.1 0.2]. $z([2 1])$ selected the second and first elements of y , *in that order*, so $z([2 1])$ is [1.1 0.9]. So w is [1.2 1.1].

You can delete parts of an array just as easily:

```
x([1 3])=[] % this leaves x as [0.3]
```

To concatenate two arrays is also trivial. To concatenate by rows, use the “;” to separate the arrays:

```
u=[z([1 1]); w]
% u is now
% [0.9000 0.9000;
% 1.2000 1.1000]
```

To concatenate by columns, omit the “;”:

```
v=[z([1 1]) w]
% v is now
% [0.9000 0.9000 1.2000 1.1000]
```

Selection of a subarray can be done not only with arrays containing indices; it can be done with arrays containing logical values as well. For example, here is a logical array:

```
vlogical=v<1.1
% vlogical is [1 1 0 0], where the 0s and 1's
% indicate whether that element is less than 1.1 or
% note.
vlt=v(vlogical) % vlt is [0.9 0.9]
```

In fact, we can select the same subarray with the oft-used shorthand.

```
vlt=v(v<1.1) % vlt is the same [0.9 0.9]
```

If, for some reason, you are interested in the actual indices of the elements of v that have value less than 1.1, you can use the “find” function:

```
idx=find(v<1.1); % idx is [1 2]
```

Naturally, you can use this index array to select the same subarray as before:

```
vlt=v(idx); % vlt is the again same [0.9 0.9]
```

So far, the array examples are all one-dimensional. But of course, MATLAB can deal with multidimensional arrays as well. Here is a two-dimensional example:

```
x=[1 2 3; 4 5 6; 7 8 9];
% x is
% 1 2 3
% 4 5 6
% 7 8 9
```

You can select the entire row or column of a multidimensional array by using the “`:`” symbol. For example:

```
xr1=x(1,:); % xr1 is the first row of x, i.e. xr1 is [1 2 3]
xc2=x(:, 2); % xc2 is the second column of x, i.e. xc2 is
% 2
% 5
% 8
```

Naturally, you can delete an entire row from an array using the same method.

```
x(1,:)=[] % x is now just [4 5 6; 7 8 9]
```

The transpose of a matrix is indicated by a simple “`'`”. So the transpose of x is just x' , which is

```
4 7
5 8
6 9
```

Elements of arrays do not have to be numbers. They can be strings, or even arrays themselves. This kind of array is called cell array in MATLAB. In the following example, C is just such a cell array:

```
C={[1 2 3]; ['a' 'b' 'c' 'd']}
```

One of the beauties of MATLAB is that practically all built-in functions can work on all elements of arrays concurrently. For example,

```
log(x) % this gives  
% 1.3863 1.6094 1.7918  
% 1.9459 2.0794 2.1972
```

There are a large number of such built-in functions. Some of the ones I have used are:

```
sum, cumsum, diag, max, min, mean, std, corrcoef,  
repmat, reshape, squeeze, sort, sortrow, rand, size,  
length, eigs, fix, round, floor, ceil, mod,  
factorial, setdiff, union, intersect, ismember,  
unique, any, all, eval, eye, ones, strmatch, regexp,  
regexpexpr, plot, hist, bar, scatter, try, catch,  
circshift, datestr, datenum, isempty, isfinite,  
isnan, islogical, randperm
```

If the built-in functions of the basic MATLAB platform do not meet all your needs, you can always purchase additional toolboxes from MATLAB. I will discuss some of them as follows.

There is also a newer data structure in MATLAB called “tables” that I found very useful. (It is very similar to Python’s Pandas Dataframes, in case any Python programmers are reading this.) Tables are arrays that come with column headings and possibly dates for rows, so you don’t need to remember that, for example, cell (1562, 244) refers to the closing price of Tesla on October 23, 2020. Instead, you can retrieve that price by writing:

```
TT{'23-Oct-2020', 'TSLA'}
```

Tables and timetables make for some powerful simplifications in manipulating time series data that has two dimensions (called *panel*

data by economists), such as the P&Ls of a portfolio of stocks. For example, you can use the synchronize function to quickly merge the P&L table and the capital allocation table, automatically aligning their dates. You can also use the retime function to convert daily P&Ls to monthly P&Ls with a single line of code.

Some of the toolboxes useful to quantitative traders are Optimization, Global Optimization (when you don't want to get stuck in a local minima/maxima and want to try simulated annealing or genetic algorithms), Statistics and Machine Learning (that's the most useful toolbox for me), Deep Learning, Signal Processing (technical indicators are but a small subset of signal processing functions), Financial (for backtesting!), Financial Instruments (good for options pricing, backing out implied volatility etc.), Econometrics (GARCH, ARIMA, VAR, and all the techniques you need for financial time series analysis), Datafeed (to retrieve data from IQFeed, Quandl, etc., easily), and Trading (to send orders to Interactive Brokers, TT, or even FIX). These toolboxes typically cost only about \$50 each for a home user. If they still do not meet all your needs, there are also a number of free user-contributed toolboxes available for download from the internet. I have introduced one of them in this book: the Econometrics toolbox developed by James LeSage (www.spatial-econometrics.com). There are a number of others that I have used before such as the Bayes Net toolbox from Kevin Murphy (github.com/bayesnet/bnt). The easy availability of these user-contributed toolboxes and the large community of MATLAB users from whom you can ask for help greatly enhance the usefulness of MATLAB as a computational platform.

You can, of course, write your own functions in MATLAB, too. I have given a number of example functions in this book, all of which can be downloaded from my website, www.epchan.com/book. In fact, it is very helpful for you to develop your own library of utilities functions that you often use for constructing trading strategies. As this homegrown library grows, your productivity in developing new strategies will increase as well.

Bibliography

- Alexander, Carol. 2001. *Market Models: A Guide to Financial Data Analysis*. West Sussex: John Wiley & Sons Ltd.
- Bailey, David, J. Borwein, Marcos 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) (May): 458–471. <https://ssrn.com/abstract=2308659>.
- Bailey, David, and Marcos López de Prado. 2012. “The Sharpe Ratio Efficient Frontier.” *Journal of Risk* 15 (2): Winter 2012/13. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1821643.
- Chan, Ernest. 2006a. “A ‘Highly Improbable’ Event? A Historical Analysis of the Natural Gas Spread Trade That Bought Down Amaranth.” *Quantitative Trading* blog, October 2, <http://epchan.blogspot.com/2006/10/highly-improbable-event.html>.
- Chan, Ernest. 2006b. “Reader Suggests a Possible Trading Strategy with the GLD–GDX Spread.” *Quantitative Trading*. November 17. <http://epchan.blogspot.com/2006/11/reader-suggested-possible-trading.html>.
- Chan, Ernest. 2013. *Algorithmic Trading: Winning Strategies and Their Rationale*. Wiley.
- Chan, Ernest. 2017. “Paradox Resolved: Why Risk Decreased Expected Log Return but Not Expected Wealth.” *Quantitative Trading*. May 4. <http://epchan.blogspot.com/2017/05/paradox-resolved-why-risk-decreases.html>.
- Chan, Ernest. 2020. “What Is the Probability of Your Profit?” PredictNow.ai. <https://www.predictnow.ai/blog/what-is-the-probability-of-profit-of-your-next-trade-introducing-predictnow-ai/>.

- Clark, Nicola. 2008. "French Bank Says Its Controls Failed for 2 Years." *New York Times*, February 21. <http://www.nytimes.com/2008/02/21/business/worldbusiness/21bank.html?ex=1361336400&en=cf84f3776a877eac&ei=5124&partner=permalink&exprod=permalink>.
- Duhigg, Charles. 2006. "Street Scene; A Smarter Computer to Pick Stock." *New York Times*, November 24.
- Economist*. 2007a. "This Year's Model." December 13. www.economist.com/finance/displaystory.cfm?story_id=10286619.
- Economist*. 2007b. "Too Much Information." July 12. www.economist.com/finance/displaystory.cfm?story_id=9482952.
- Economist*. 2019. "March of the Machines. The Stock Market Is Now Run by Computers, Algorithms, and Passive Managers." October 5. www.economist.com/briefing/2019/10/05/the-stockmarket-is-now-run-by-computers-algorithms-and-passive-managers.
- Fama, Eugene, and Kenneth French. 1992. "The Cross-Section of Expected Stock Returns." *Journal of Finance* **XLVII** (2): 427–465.
- Fielden, Sandy. 2006. "Seasonal Surprises." *Energy Risk*. September. <https://www.risk.net/infrastructure/1523742/seasonal-surprises>.
- Gershgorn. 2017. "The Data That Transformed AI Research—and Possibly the World." *Qz*. <https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/>.
- Grinold, Richard, and Ronald Kahn. 1999. *Active Portfolio Management*. New York: McGraw-Hill.
- Heston, Steven, and Ronnie Sadka. 2007. "Seasonality in the Cross-Section of Expected Stock Returns." AFA 2006 Boston Meetings Paper, July. [lcb1.uoregon.edu/rcg/seminars/seasonal072604.pdf](http://uoregon.edu/rcg/seminars/seasonal072604.pdf).
- Kahneman, Daniel. 2011. *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- Khandani, Amir E., and Andrew Lo. 2007. "What Happened to the Quants in August 2007?" MIT. <https://web.mit.edu/Alo/www/Papers/august07.pdf>.
- Kochkodin, Brandon. 2021. "How Wall Street Bets Pushed GameStop Shares to the Moon." Bloomberg. www.bloomberg.com/news/articles/2021-01-25/how-wallstreetbets-pushed-gamestop-shares-to-the-moon?sref=MqSE4VuP.
- Lewis, Michael. 2014. *Flash Boys*. W.W. Norton.

- Lo, Andrew. 2019. *Adaptive Markets: Financial Evolution at the Speed of Thought*. Princeton University Press.
- López de Prado, Marcos. 2018. *Advances in Financial Machine Learning*. Wiley.
- Lowenstein, Roger. 2000. *When Genius Failed: The Rise and Fall of Long-Term Capital Management*. Random House.
- Lux, Hal. 2000. “The Secret World of Jim Simons.” *Institutional Investor Magazine*, November 1.
- Markoff, John. 2007. “Faster Chips Are Leaving Programmers in Their Dust.” *New York Times*, December 17. www.nytimes.com/2007/12/17/technology/17chip.html?ex=1355634000&en=a81769355deb7953&ei=5124&partner=permalink&exprod=permalink.
- Oldfield, Richard. 2007. *Simple but Not Easy*. Doddington Publishing.
- Peters, O., and M. Gell-Mann 2016. “Evaluating Gambles Using Dyanmics.” *Chaos* 26: 023103. <https://doi.org/10.1063/1.4940236>.
- Phillips, Daniel. 2020. “Investment Strategy Commentary: Value Stocks: Trapped or Spring-Loaded?” Northern Trust, September 24. <https://www.northerentrust.com/canada/insights-research/2020/investment-management/value-stocks>.
- Poundstone, William. 2005. *Fortune's Formula*. New York: Hill and Wang.
- Ritter, Jay. 2003. “Behavioral Finance.” *Pacific-Basin Finance Journal* 11 (4) September: 429–437.
- Schiller, Robert. 2008. “Economic View; How a Bubble Stayed under the Radar.” *New York Times*, March 2. www.nytimes.com/2008/03/02/business/02view.html?ex=1362286800&en=da9e48989b6f937a&ei=5124&partner=permalink&exprod=permalink.
- Sharpe, William. 1994. “The Sharpe Ratio.” *Journal of Portfolio Management*, Fall. <https://jpm.pm-research.com/content/21/1/49>.
- Singal, Vijay. 2006. *Beyond the Random Walk*. Oxford University Press.
- Taleb, Nassim. 2007. *The Black Swan: The Impact of the Highly Improbable*. Random House.
- Thaler, Richard. 1994. *The Winner's Curse*. Princeton, NJ: Princeton University Press.

Thorp, Edward. 1997. "The Kelly Criterion in Blackjack, Sports Betting, and the Stock Market." *Handbook of Asset and Liability Management*, Volume I, Zenios and Ziembra (eds.). Elsevier 2006. www.EdwardOThorp.com.

Toepke, Jerry. 2004. "Fill 'Er Up! Benefit from Seasonal Price Patterns in Energy Futures." *Stocks, Futures and Options Magazine*, March 3 (3). www.sfomag.com/issuedetail.asp?MonthNameID=March&yearID=2004.

Uhlenbeck, George, and Leonard Ornstein. 1930. "On the Theory of Brownian Motion." *Physical Review* 36: 823–841.

About the Author

Ernest P. Chan is the founder of PredictNow.ai, a financial machine-learning SaaS available as a no-code service or via an API. He is also the founder of QTS Capital Management, LLC. (www.qtscm.com), which manages a hedge fund and separate client brokerage accounts. He has been quoted by the *Wall Street Journal*, *New York Times*, and *CIO* magazine on quantitative investing, and has appeared on CNBC's *Closing Bell*. His firm has also been profiled in a *Bloomberg Businessweek* article.

Ernie is the author of *Quantitative Trading: How to Build Your Own Algorithmic Trading Business*, *Algorithmic Trading: Winning Strategies and Their Rationale*, and *Machine Trading: Deploying Computer Algorithms to Conquer the Markets*, all published by John Wiley & Sons. To learn more about his books and training courses, visit www.epchan.com. He maintains a popular blog at predictnow.ai/blog, where readers can also download his other publications.

Ernie is an expert in developing statistical models and advanced computer algorithms to discover patterns and trends from large quantities of data. He was a machine-learning researcher at IBM's T. J. Watson Research Center's Human Language Technologies group, at the Data Mining and Artificial Intelligence group at Morgan Stanley, and at the Horizon proprietary trading group at Credit Suisse, as well as at various other hedge funds. He is also an adjunct faculty at Northwestern University's Master's in Data Science program. Ernie received his undergraduate degree from the University of Toronto and a doctor of philosophy degree in theoretical physics from Cornell University.