

A MultiCore Bollinger Band Analyzer

By Ruslan Ardashev

Duke University
Pratt School of Engineering
Class of 2015

SOURCE CODE AVAILABLE AT
<https://github.com/ruslan120101/BollingerAnalyzer>

STOCKS BLOG AVAILABLE AT
<http://stockswithruslan.blogspot.com>

Table of Contents

- 1. Introduction – Page 5**
- 2. Class Hierarchy – Page 7**
 - 1. Analyzer
 - 2. DataGetter
 - 3. StockEntry
 - 4. Calculator
- 3. How It Works – Page 8**
 - 1. Plotting
 - 2. Trading Simulation
- 4. Conclusions – Page 9**
- 5. Future Work – Page 9**
- 6. Appendix – Page 10**

1. Introduction

I like to dabble in stock trading. I am a firm believer in picking a concrete strategy that is right “most of the time,” and sticking to it. Even if it's right 55% of the time, and 45% of the time you lose money, you end in a net-positive in the long term. Following this mindset, I've created a tool that uses technical analysis to guarantee a net profit when trading stocks.

This technical analysis is based on Bollinger Bands. Bollinger Bands help a trader approximate levels of resistance and support – both, upper and lower bands are two standard deviations away a stock's Simple Moving Average (SMA). When a stock's price is at or near the upper Bollinger Band, chances are good that this stock will shortly decrease in price. Conversely, when a stock is decreasing and is near the lower Bollinger Band, chances are that this stock will shortly increase in price.



Figure 1. Shown above are upper and lower Bollinger Bands in purple. Observe that when the stock (in blue) approaches the upper band, it shortly decreases. The converse holds true as well.

From Figure 1, we can see that this limit is a great indicator. Most of the time, the stock reverses direction when too close to a Bollinger Band.

This leads to questions such as, “How often are Bollinger Bands a good approximation of when to buy, and when to sell? What's the exact percentage? **Can we use such a percentage in an algorithmic manner to make money?**”

A MultiCore Bollinger Band Analyzer

Flow

- Takes as many stock tickers as the user gives it.
- For each stock, the analyzer then gets historical stock data from January 1st, 2000.
- Using this data, the analyzer finds all points at which the stock price pivoted.
- If the price pivoted closer to the upper band, the percentage distance from the upper band is recorded.
- If the price pivoted closer to the lower band, the percentage distance from the lower band is recorded.
- These percentages (distance in price as a percentage of closing price) are then plotted individually for each stock.
 - **Plots Include**
 - A **scatter plot** to display individual stocks.
 - A **frequency plot** to display the frequencies of percentages that occur (rounded to the nearest hundredth of a percent).
- The analyzer then uses the average percentage distance from the upper and lower Bollinger Bands to make trades.
 - It starts with a position of \$100,000 in each stock, looks at the closing price in January 1st, 2000 in \$/share, and divides starting cash by initial stock price, giving us the number of initial shares.
 - **Sample Run**
 - The price just pivoted. Was it closer to the upper band? Yes.
 - Compute the distance in percent to the upper Bollinger Band. Was it less than or equal to the average percentage distance from the Bollinger Band? Yes.
 - SELL. (If not, do not sell).
- The analyzer finally plots, and reports all final values.

Key Features:

- Is multicore enabled for computational speed.
 - Plotting, simulation, and computing of results are all done independently by separate threads and cores.
- Pulls data from the web (Yahoo Finance).
 - Handles failures to do so (incorrect stock tickers entered, failed web requests, etc.).
- **Makes** (virtual) **money** successfully.

Assumptions

- All trades are at the closing price.
 - This might not be picture-perfect, but this can be fine-tuned in a more complex program with more time invested.
- Owned position in each stock starting January 1st (first day of trading was technically the 3rd).
- Fractional shares are allowed. All cash is used to buy and sell.

2. Class Hierarchy

2.1 Analyzer

The main instance of Analyzer has four main responsibilities:

- (1) Being the main dispatcher. Creating DataGetters, and sending them off to do work.
- (2) Averaging the results of DataGetters' calculations.
- (3) Plotting all results and presenting them to the user.
- (4) Concatenating all trading simulation results.

Analyzer is the main thread. Important values, such as the “Final Average Percent Distance from Bollinger Band at Pivots” value, are stored in the Class Analyzer, whereas things like “Data Getter Children” are stored in the main instance of Analyzer, called “mainAnalyzer.” This mainAnalyzer adheres to a delegate pattern, and is passed to DataGetters and Calculators alike, so that one can have access to the Analyzer at all times to contribute data.

2.2 DataGetter

DataGetters have two main responsibilities:

- (1) Pulling data from Yahoo Finance.
- (2) Simulating trades once an average percent has been computed.

DataGetters are multicore enabled. They “own” calculators which do the dirty work.

2.3 StockEntry

Containers for stock data. If a DataGetters pulls 1,005 entries from Yahoo, each with a date, closing price, etc., it will create 1,005 “StockEntry” instances that each hold a date, closing price, etc.

2.4 DataCalculator

Do the dirty work. Each DataGetter owns its own DataCalculator.

Each DataCalculator calculates pivots, the SMA, and the Bollinger Bands.

3. How It Works

- **Analyzer** fires up inside of main and looks at what stocks were entered.
 - For each stock that was entered, it creates a **DataGetter** (thread), and sends it off to fetch data from Yahoo Finance.
- Each **DataGetter** pulls the data, and fills up a series of **StockEntry** objects with the data.
- The **DataGetter** then creates a **DataCalculator** and gives the calculator all of the data it just pulled.
- The **DataCalculator** uses all of the **StockEntry** objects from within its parent, DataGetter, to make arrays full of processed data.
 - The SMA
 - From this SMA, Bollinger Bands
 - Where the pivots occurred
 - At those pivots, was the price closer to the upper or lower band
 - What was the percentage distance to said band
 - **Store all of that information in the Analyzer.**
- The **Analyzer** anxiously waits for all of the DataGetters to complete. Once they do, it
 - fires off plotting functions that plot all of said data
 - calculates average percentage distance from both bands at pivots
 - reports all calculations for the user to see
- Once the **Analyzer** has done so, it dispatches all of the **DataGetters** once more, who have been patiently waiting, for them to now run through their respective stock's data, and simulate trading using the previously calculated average percent distance from the upper and lower Bollinger Bands.
- Each **DataGetter** has a simple policy.
 - Start with a position. Keep track of whether you can sell or not.
 - If you just sold, “canBuy” turns true. The simulator will be looking to buy.
 - If you just bought, “canSell” turns true. The simulator will be looking to sell.
 - Cycle through all trading days based on closing price.
 - **Buy / Sell**
 - If you are looking to buy (“canBuy” is true), the simulator looks at proximity to the lower band.
 - If you are looking to sell (“canSell” is true), the simulator looks at proximity to the upper band.
 - If the percentage distance (previously calculated) falls close to either band, trades **are executed.**
- Each **DataGetter**, when done, reports its average earnings back to the main **Analyzer.**
- The main **Analyzer** averages all values, and reports average earnings.

4. Conclusions

I found that for the following stocks:

{"AAPL", "MSFT", "S", "ARR", "FB", "F", "BAC", "YHOO", "C", "TSLA", "KO"}

The average pivot occurred at

- **BUY** signal at **3.936%** or less from the **lower** Bollinger Band
- **SELL** signal at **3.841%** or less from the **upper** Bollinger Band

With a starting sum of **\$100,000**,

- The final sum was **\$463,602.68**
- A gain of **363.6%**

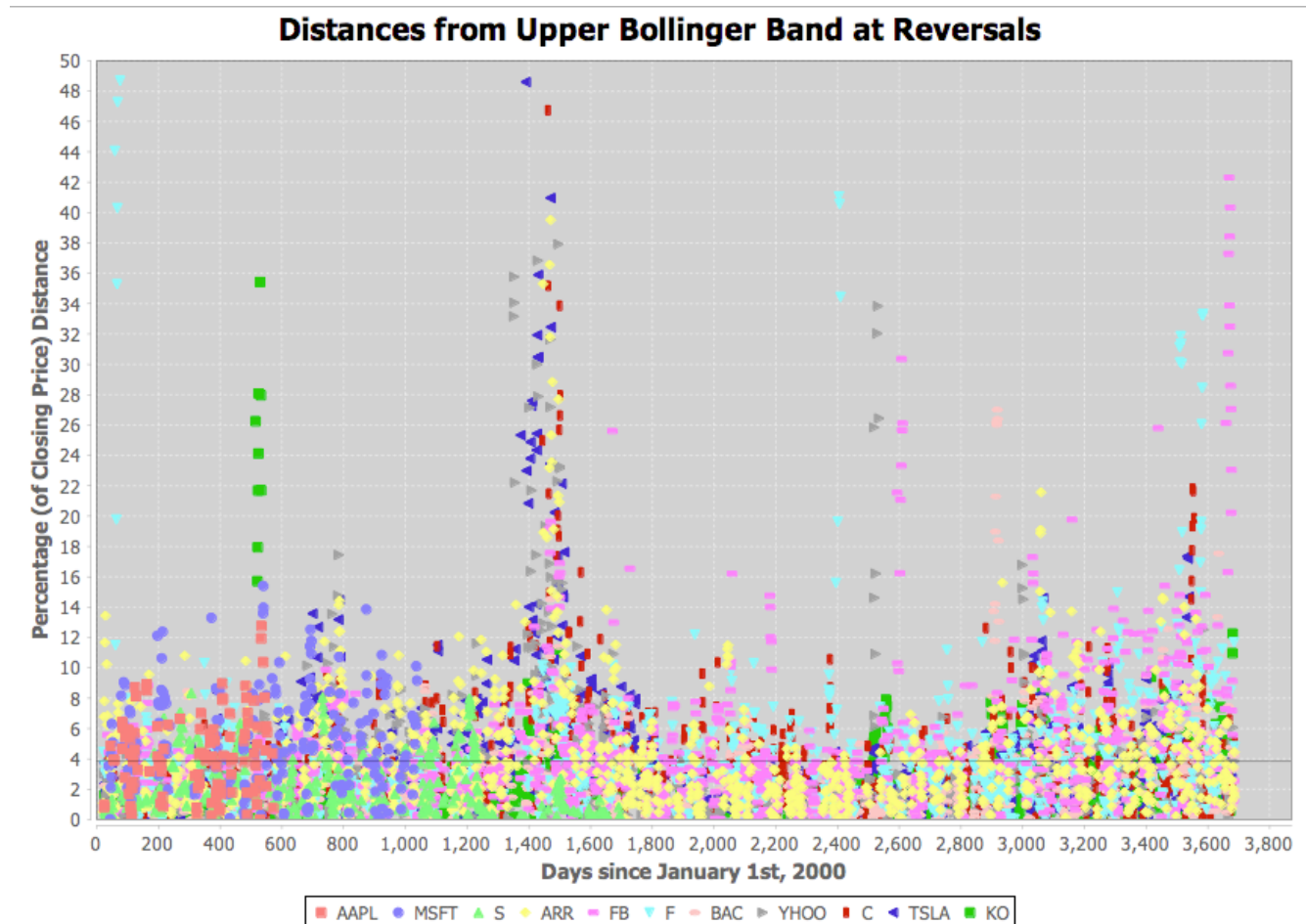
5. Future Work

Below are some questions that arose in the process of writing this simulator and looking at its results:

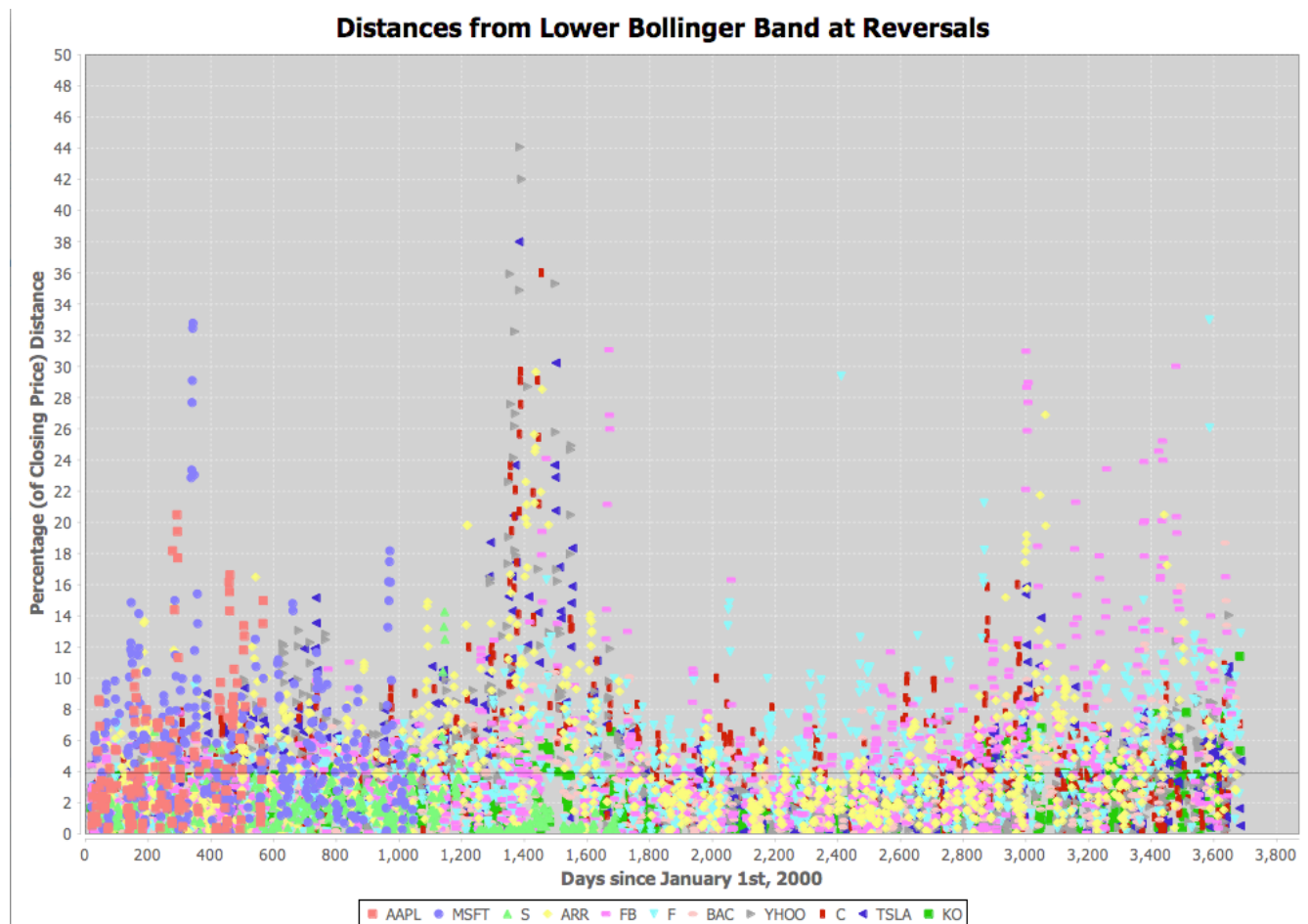
- The simulator prints individual stock gains. **What's the relationship between the stock's Beta, or risk factor, and the amount this algorithm earned?** It seems that there might be a correlation between a smaller beta and larger gains from this algorithm.
- Would results be different if we simulated intra-day trading as opposed to closing price trading?
- We allow one trade per day at the closing prices. **How would this differ if we took into account the T+3 day settlement policy on all trades?**
- (And of course) **Can this be made to run even faster?**

6. Appendix

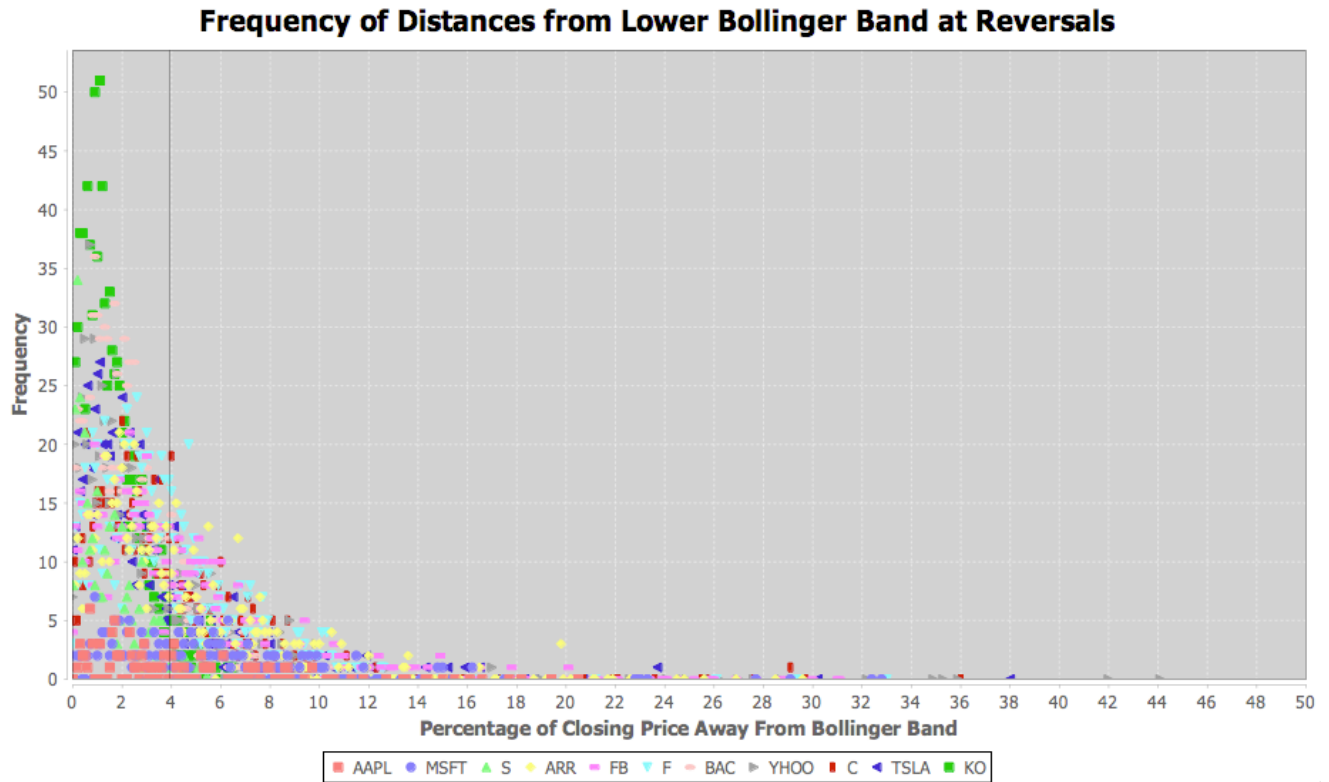
The following plots were created using the free library, **jFreeChart**. (Thank you!)



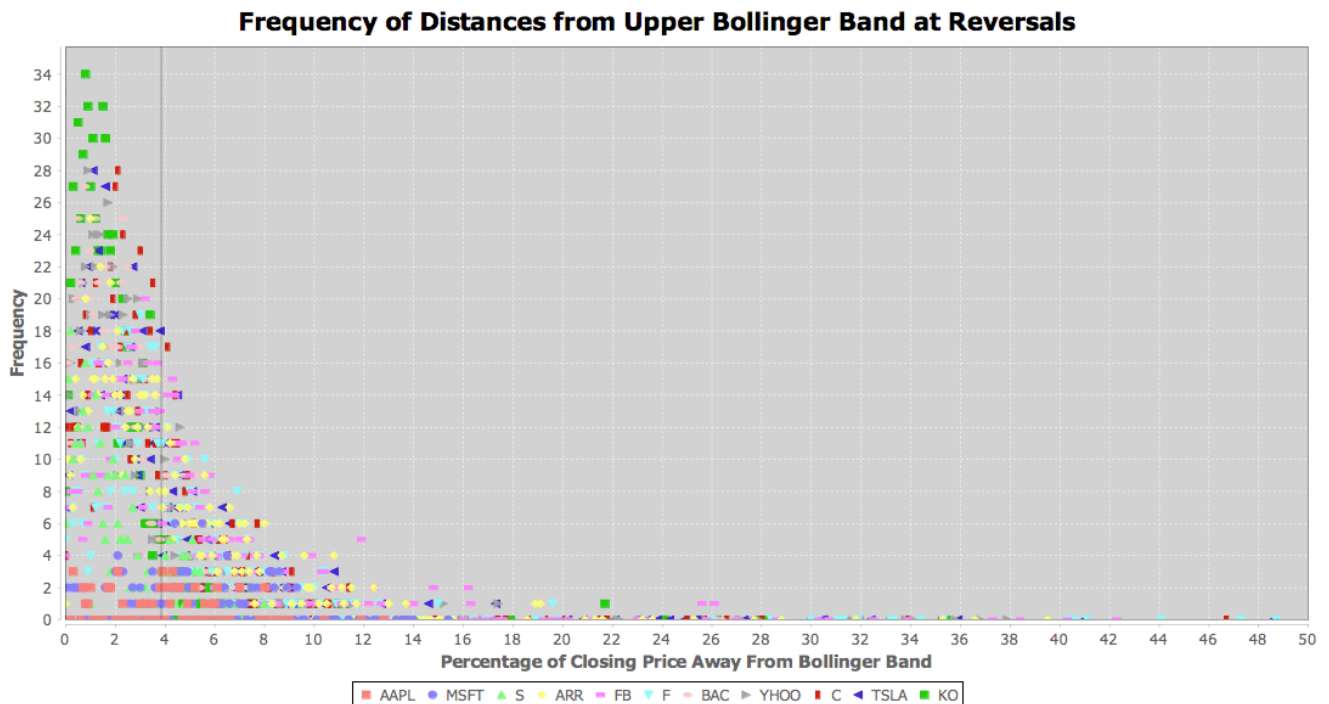
Plot 1. Distance from Upper Bollinger Band at Reversals (Pivots). Note the black horizontal line indicating the average percent distance at which pivots occurred.



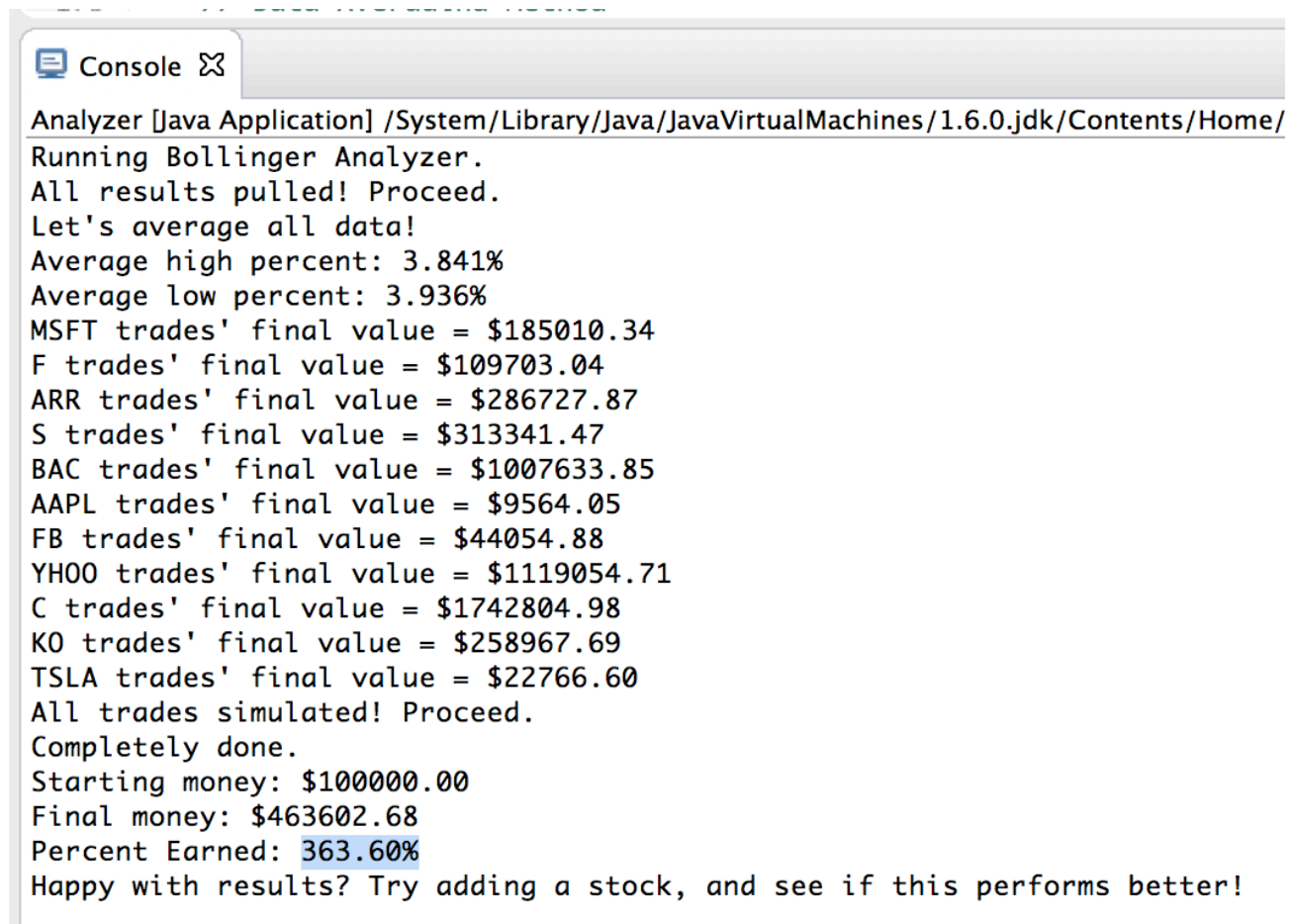
Plot 2. Distance from Lower Bollinger Band at Reversals (Pivots). Note the black horizontal line indicating the average percent distance at which pivots occurred.



Plot 3. Frequency of Distances from the Lower Bollinger Band at Reversals (pivots). NB the black vertical line (average %)



Plot 4. Frequency of Distances from the Upper Bollinger Band at Reversals (pivots). NB the black vertical line (average %)



```
Analyzer [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/
Running Bollinger Analyzer.
All results pulled! Proceed.
Let's average all data!
Average high percent: 3.841%
Average low percent: 3.936%
MSFT trades' final value = $185010.34
F trades' final value = $109703.04
ARR trades' final value = $286727.87
S trades' final value = $313341.47
BAC trades' final value = $1007633.85
AAPL trades' final value = $9564.05
FB trades' final value = $44054.88
YHOO trades' final value = $1119054.71
C trades' final value = $1742804.98
KO trades' final value = $258967.69
TSLA trades' final value = $22766.60
All trades simulated! Proceed.
Completely done.
Starting money: $100000.00
Final money: $463602.68
Percent Earned: 363.60%
Happy with results? Try adding a stock, and see if this performs better!
```

Figure 2. Sample output.

```
62 // Pulling Data
63+ private synchronized void pullData (String symbol) {}
116
117+ private void dispatchAnalysis() {}
136
137
138 // Simulating Trading
139+ private void simulateTrading() {}
177
178+ private void reportResults(){}
190
191+ private void assessBuy(Double currentStockPrice, Double currentLowerBollingerBandValue, int stockEntryId) {}
202
203+ private void assessSell(Double currentStockPrice, Double currentUpperBollingerBandValue, int stockEntryId) {}
214
215+ private void BUY(int stockEntryId) {}
223
224+ private void SELL(int stockEntryId) {}
232
233+ private void setInitialSimulationParameters() {}
241
242 }
```

Figure 3. Some methods of DataGetter