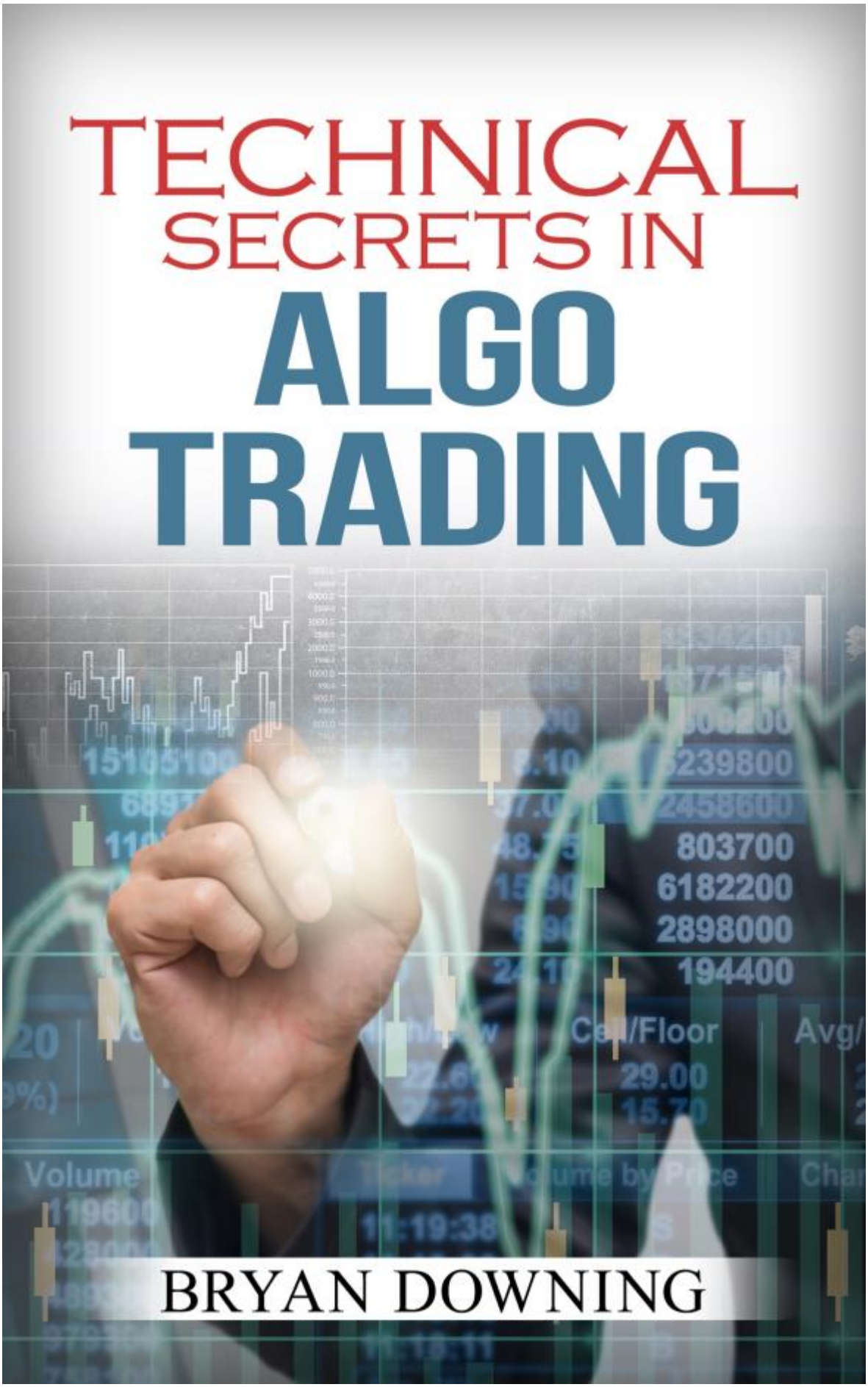
TECHNICAL
SECRETS IN
ALGO
TRADING
803700
6182200
2898000
194400
29.00
Volume
BRYAN DOWNING

THE STATS

Okay. Let's start talking about the first book. Let's talk about the facts on Algo trading, including high-frequency trading. The first thing to understand is about 80% of all volume is driven by electronic trading. Also, a lot of the professionals, a lot of the high frequency trading shops are part of this.

Number two. No one ever makes a living from automated trading through commercial automated trading systems. There are those that are selling them but they don't want you to know that.

Number three. All people who make a living from automated trading developed their own systems.

Number four. No one has ever succeeded in automated trading without effort. In this book, I hope to show you and introduce you to some very useful shortcuts.

Number five. What I've learned, over the years, from the multimillionaire, automated traders, or managers that have firms that are multimillion, the constant pattern I'm seeing is, simple is better. From what I've seen, that's a huge part of their success. I hope

it'll be something that you can also take away from it.

MY BACKGROUND

Here's my background. My name's Bryan Downing. I'm a well-known YouTube partner in the automated trading niche. My LinkedIn profile has reached over the top one percent of most viewed profiles in the entire site of LinkedIn. I also have over 8,000 members in one of Facebook's fastest growing groups for learning about trading software knowledge. I'm also the creator of the popular blog, Quantlabs.net.

I've also been researching high speed trading platforms and trading strategies for the last five years. All of this stuff, I want to show you, how to do that right now, as you read this.

1. YOUR CHOICE OF LANGUAGE

First of all, let's talk about the different languages out there, from a high level. These are some of the basic components and decisions that you need to make when implementing your own algorithmic trading system. The first thing you need to understand is about which programming language are you going to want to use? There are two subsets of programming languages.

The first set are compiled languages. This is where you're going to start putting in your execution to make the program more stable, more solid, more robust. There's only three languages that I can recommend.

Java, which came out in the early 90's, had a philosophy of, write once. Deploy anywhere on all the major operating systems. It's the nice thing about java.

The other language that you could use is C#, which is part of the Microsoft .net family. It's fairly easy to

use. Obviously it runs on Windows. They're trying to get it on to Linux. Personally, I'm no longer a fan of Microsoft and Windows10 product due to the security, the latency that this newer operating system brings to the table, and so on.

The last language of the compiled set is one of two, C++ or C. C++ can be a pretty tough language to learn but it's also the fastest and it's usually one step above Assembler, which is a very, very fast but very primitive and a very low level language. The difference between C++ and C is that C++ is more object oriented with few other functionality in that

language, which I'm not going to get into. Whereas C is a much older language, a much simpler language. It has a similar performance but does not offer the object oriented set of principles that object oriented brings to the table.

The second subset of programming languages that are out there could fall into interpretive or scripting languages. That brings us to the two popular languages used in data science/big data, R and Python.

R I've used for about six to nine months. That can be a fairly frustrating language. Obviously, it can be

slow. All these interpretive languages can be slow. R is used primarily for statistics. It has dozens upon dozens of packages that you'll be involved with, to transfer one data type to another. It can really bog down your programming style. It's something where at times, I just didn't like. There are some really good advantages with R, specifically around visualization, which we'll cover in an upcoming section.

The other language we could talk about is Python. Python is a very very popular language. It can be used to develop web applications. It could also be used as a shell-script on a lower level operating system, like

Linux. Python can also be used for research, as a scripting language as well. The next thing I like about Python is that there's usually no more than four, maximum five, packages that you'll use. The three packages that you'll typically use are interchangeable quite nicely. It will minimize your frustration. It will also minimize your coding, as well.

THEN THERE IS MATLAB…

The other language that can fall into both compiled and scripting/interpretive, is Matlab. Matlab is more like an ecosystem, where it can read in natively some

of the languages that I've already mentioned, except R. You can also take a Matlab script that you develop to code-generate that same M scripting language. You can code generate into something like C or C++, which is a really nice feature.

Usually, the lifecycle of a strategy, when it comes to professional, institutional shops like banks or hedge funds is quite cumbersome. What they'll do is they'll develop a strategy and prototype it using something like R or Python. Once they're comfortable with that trading algorithm, or strategy in one of those scripting languages, they then have to pass off that

strategy to another department who will hand code that into a compiled language, like the Java or C++. That's usually the methodology they use.

The nice thing with Matlab is it can cover all of the above, that I mentioned, where you develop a script in an interpretive language, like M language, which is part of Matlab and then code-generate it into C++. There's no hand coding process, which is very nice.

2. WHICH DATABASE?

Our second chapter is our database. Usually this is where things get confusing. I want to first talk about

the world that I like to choose. Open source databases are really good. The popular ones, right now, that are getting really popular are what they call, NOSQL databases. For example, there is no structure to the data. It's hard to explain but I won't do that now. Usually the two that I've looked at, that I like, are MongoDB and Redis.

MongoDB can have the ability to save the data on a drive and be fairly fast. Whereas Redis is more of an in-memory database. It can store everything in, what they call, key-store set of values that you store in memory, which makes this database very, very fast.

Both of these NOSQL databases are open-sourced, where you can actually download the source code. The cool thing about Redis, it's written in C. That means the database, itself, is embeddable into a flash memory environment. You can make this database as fast as it can get. Not only that, it enables you to connect into other popular programming languages, like R, Python, C#, Java and all the other languages I've already mentioned.

That's the advantages of NOSQL and my preferred database, which is Redis. Then we move into the more traditional relationship databases that involve

Oracle, SQL Server, which are the more popular ones. These databases are clunky because they have an engine to them. They also have the licensing that will limit on how many connections you can have, based upon the processors on your server. It's an old, old way of maintaining databases. More and more new projects have come online that are built in the world of software, are moving away from these type of databases, like Oracle or SQL Server.

The other problem with commercial databases, as I said, they're fairly clunky. They're going to be slower than the NOSQL.

Another alternative that we have are the open-source databases still relationship ones like MySQL or Postgres. These databases are fine. They're free. You can download the source code, so you can tinker with them as needed. They're still heavy though. Meaning they're going to have an engine to them and you will not be able to process them in a "light method". They also have a lot of operations in both these databases. You then have to have a dedicated server for one of these databases. Again, I've already talked about NOSQL.

3. BROKERS

The third topic we want to talk about is your broker. The brokers out there are really important. I will have another book built around secrets and whatnot for retail traders. To understand the other types of brokers and how those type of brokers can screw you, please refer to that book. This is in specific arenas, like forex international brokers. In that book, I give eleven reasons on how those brokers are not really wise to go with.

The “cleanest” broker out of the whole bunch is Interactive Brokers. They also have a piece of software called, Trader Work Station, or better known as TWS. This software can be good. Where you have your program connect into it and you can interface your market orders, your market data between your program and the TWS software. It includes charting and some other neat functionality. The nice thing about TWS is, it also includes all of your portfolio and all your trading account information in one piece of software. Very very useful when you're starting out.

The big reason why Interactive Brokers is also used is because it supports other programming languages, including C++, C#, or Java. The most popular one is Java. As I said earlier, that can be run on multiple operating systems, like Mac, or Linux, or Windows. Whereas, if you're choosing C#, you're going to be limited to Microsoft Windows.

The other way to interface with Interactive Brokers is through their software gateway. This is usually more for a program that is needed with faster processing, which is usually going to be on a Linux environment. Interactive Brokers will call it a Posix, which is a

Linux-like operating system. One of the popular ways to do it is using C++, which is very low level and can be fairly complicated to work with. I wouldn't recommend that when starting out.

4. MARKET DATA

The other thing you need to remember is your market data. There's multiple sources that you can go to. If you're on a budget or don't really have a budget, you can use Yahoo Finance, which is okay to use. Where you can download market data from Yahoo but in a very limited way. It will cap you on the number of requests

you can send out to the Yahoo Finance servers. That will, obviously, limit you in a lot of ways.

If you're just going to be using Yahoo Finance for analytical purposes, this is a fine option. If you're trying to do live trading, Yahoo Finance will cut you off after a certain time. I would not recommend using Yahoo Finance if you're going to use your live trading in an intra day type of period, maybe even a higher frequency on an hourly, or a minute by minute basis.

The other service you could use is, pretty well my favorite, there's other ones as well that I'm not

going to get into. If you need tick data for forex or equity or future/options, you can download all that from IQFeed. It's fairly affordable. You can get tick data. They also, from my understanding, have some market instruments that you may be interested in are available for up to nine years of historical data, which will save you thousands of dollars. You don't need to download or purchase very expensive data sets from other providers. You can get that really cheaply through IQFeed. The cost range from thirty dollars a month for a pure forex set of data from FXCM. Or, if you need to scale up, you can spend in the neighborhood of $100+, depending on what kind of data

you want. Obviously, the futures data gets much more expensive but it's still fairly reasonable.

5. PYTHON PACKAGES

The other area that we need to look at is visualization. Here, all these topics that I'm covering are using a language like Python. Python, I find if you're starting out, you want to start with Python because it's an easy language to pick up. It's very popular. The community is huge on YouTube or online with some really great resources for Python communities. Even in the world of trading, or

specifically in the world of quantitative analysis, better known as Quant.

Knowing that, let's talk about some of the more popular visualization packages for Python. The big one, that's standard, is Matplotlib. Very simple technical analysis charts you can get out of it, some volume charts. You can get some statistical charts. Really easy.

There's two other ones that are not well known. PyQtChart, which is built off of Qt. It enables you to build live trading charts because it's using the

technology of Qt which makes if fairly fast and you can build up some really nice, sophisticated charts. Even some 3D plots that are fairly good looking, as compared to Matplotlib.

This third one, called Bokeh, is a really cool library, which enables you to set up a server and then have Python scripts call it. This means, you can then deploy, using Bokeh as a server, for hosting charting, even live charting, and do it through a web-enabled environment. You can have a Bokeh server set up where you can have browser sessions connect into this server. You could serve up multiple clients.

This is the mysterious one because right now, the GUI, or Graphical User Interface, Customization can be fairly wonky. There's Python, which my group, that I taught, had a real difficulty trying to implement. Basically, when it comes to your trading system, you have a choice to ask yourself. Do you want to develop a charting and a GUI-based front end that's going to be deployed on a desktop? Or is it going to be deployed on a mobile environment?

DESKTOP OR MOBILE (APPLE VS ANDROID)

I've looked at both. There's numerous options. You could look at desktop. Now I'm starting to think, PC sales are down. Windows is becoming less relevant. The direction of the market is obviously going into mobile. Then it comes back to the two popular mobile operating systems.

The first one's from Google, called Android. That operating system is fairly insecure. It's prone to hacking. There's hundreds of different devices out there with different ways of targeting the display. It becomes a real nightmare to work with.

The other popular mobile operating system is from Apple, called IOS. Recently, in the last few years or so, they've released a language called Swift, that is, technically, open-source, that enables you to develop your front end with a choice of an iPhone, an iPad and even a Macintosh computer.

If you chose, let's say, Swift, you have numerous options. You have a fairly rapid way to visually build your components with some Swift coding and deploy it onto an IOS device. Or, as I said, a Mac environment. I've also looked at some fairly affordable, high-end looking charting and graphical user libraries that all support Swift and they're very solid. This is probably the direction I'm going to go for my front end, GUI-based, graphical user interface, type of development, is using Swift. I didn't mention this, but the big, big selling point of IOS and overall Apple devices is the security that you get from the operating system.

It's built into the operating system with a few extra layers of security that you do not get with Android.

All in all, that's my basic summary of how to build out an overview of an algorithm trading system. I hope you enjoyed it and you took away from knowledge from it. If you've got any questions just hit me up on my social media channels and I'll do my best to address any concerns you may have.