

The Virtue of Simplicity

As we keep pointing out, automated/algorithmic trading does not automatically imply ultra high frequency trading. Plenty of Automated Trader traders we talk to seem to do just fine deploying automated models that hold positions for periods ranging from seconds to days. At the same time, rather a lot of them appear to use MATLAB and more than a few use Interactive Brokers. All of which rather inevitably led the Wrecking Crew and Automated Trader's Founder, Andy Webb, to take a look at IB-MATLAB...

First, an important message from the Automated Trader software review team. If you are looking for a high frequency trading interface to Interactive Brokers, please stop reading now; IB-MATLAB isn't intended for you. Why not flick to Peek Ahead at the end of the magazine and read this issue's morally improving message from the Editor instead?

IB-MATLAB *is* intended to provide a quick and simple path to interfacing MATLAB with Interactive Brokers' IB API (as opposed to the FIX CTCTI API that Interactive Brokers also offers). The intention is that IB-MATLAB will be primarily used to send orders and receive order fills - not for receiving real time streaming data from the IB API. For one thing, the IB API has a specified limit of 50 messages per second that would quickly be consumed by real time data from even a modest portfolio of highly active securities. For another, MATLAB is also not really intended for such a purpose and is likely to complain if its interfaces are swamped by a blizzard of market data.

While the IB API offers various means of connecting, including ActiveX, DDE/ActiveX for Excel and C++, IB-MATLAB uses the Interactive Brokers Java API, which has the advantage of being OS independent. So if you run MATLAB on Linux (as a lot of Automated Trader readers do) you can still use IB-MATLAB. Of course there's nothing to stop you from using the MATLAB Java API directly yourself to interface MATLAB and IB, but the point of IB-MATLAB is

to simplify that so you can spend more time testing trading models and less time fixing the plumbing.

Data

While IB-MATLAB isn't intended for high-volume streaming of real time data, it happily supports one-off requests for current market prices. The syntax is pretty simple and follows the same basic method as most other IB-MATLAB functions, which consist of pairing an input parameter with a user specified parameter value. The generic syntax is:

```
IB_trade('parameter 1', 'parameter value 1', 'parameter 2', 'parameter value 2' ... 'parameter n', 'parameter value n')
```

For example, to get a quick update on Cisco, just type the following into the MATLAB console (or invoke it in a MATLAB function or script):

```
data = IB_trade('action', 'QUERY', 'symbol', 'CSCO')
```

This returns a MATLAB struct called 'data' that includes items such as bid, ask, open, high, low, close, volume, tick size and time stamp. As a standard MATLAB struct, this means that any of these items can be accessed in MATLAB by using 'dot referencing'. For example, entering:

```
data.high
```

...in the MATLAB console returns the high of the session:

```
ans =  
15.6200
```

It is also possible to retrieve historical data for the current trading session. For example...

```
histdata = IB_trade('action', 'HISTORY_DATA', 'symbol', 'IBM', 'barSize', '1 min', 'useRTH', 1);
```

...is pretty self explanatory ('useRTH' means 'use regular trading hours') and returns the following struct:



The Virtue of Simplicity

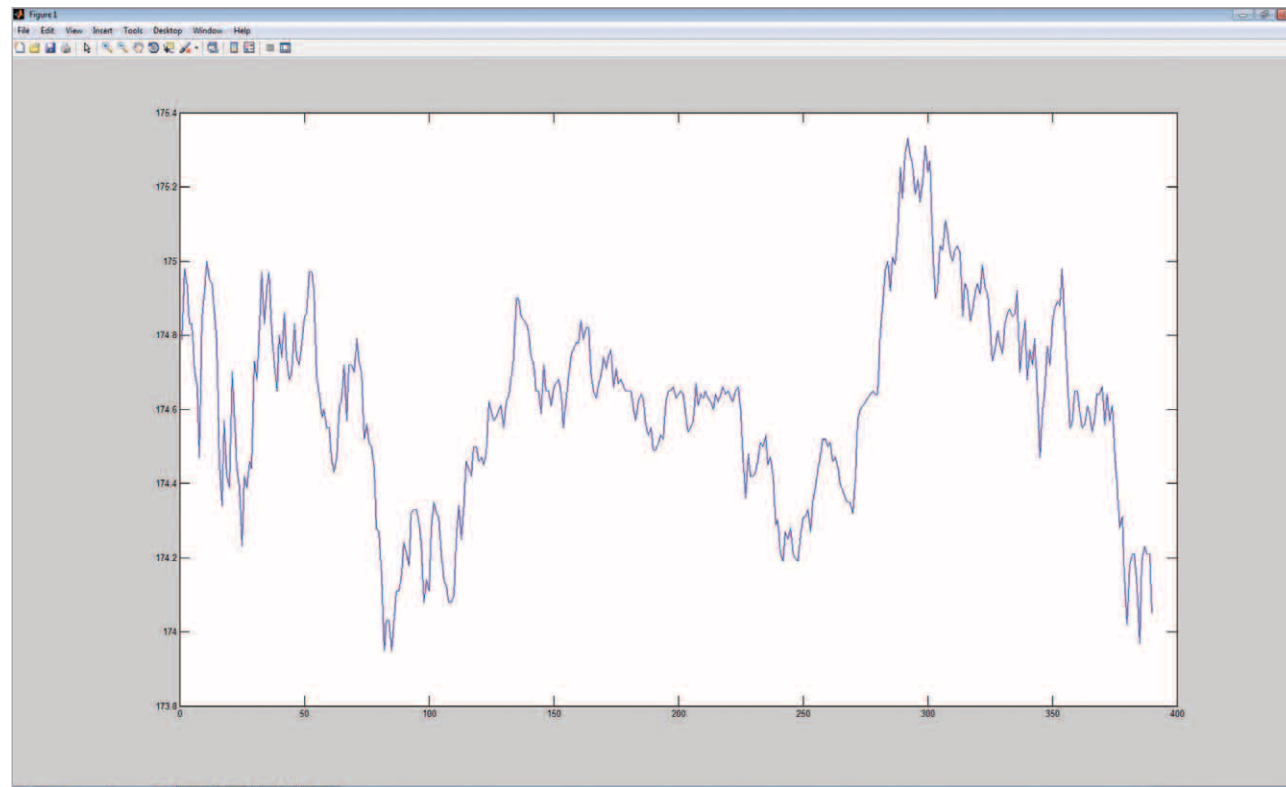


Figure 1

```

histdata =
    dateTime: {1x390 cell}
    open: [1x390 double]
    high: [1x390 double]
    low: [1x390 double]
    close: [1x390 double]
    volume: [1x390 double]
    count: [1x390 double]
    WAP: [1x390 double]
    hasGaps: [1x390 logical]

```

As with the previous example, because a standard MATLAB struct is returned, individual data categories can be dot referenced. For instance, typing...

```
plot(histdata.close(1,1:390))
```

...in the console generates a chart of the closing values of each one minute bar as shown in Figure 1.

The output to data requests can immediately be reused as inputs to an order. For example, the bid value from the struct that results from the following...

```
data = IB_trade('action','QUERY',
'symbol','CSCO')
```

...is used ('data.bid') in an attempt to buy Cisco at the bid in the following order:

```

orderId = IB_trade('action','BUY',
'symbol',stk.stk{rndsec,1},
'quantity',100, 'limitPrice',data.
bid);

```

The 'orderId' variable is immediately re-output to the MATLAB console together with any resulting fill information:

```

openOrder:      orderId =89326106
orderStatus:    orderId =89326106
orderStatus:    status =Filled
orderStatus:    filled =100
orderStatus:    remaining =0
orderStatus:    avgFillPrice =67.73
orderStatus:    permId =395159315
orderStatus:    parentId =0
orderStatus:    lastFillPrice =67.73
orderStatus:    clientId =884191
orderStatus:    whyHeld =

```

While on the subject of console output, it's worth pointing out that IB-MATLAB defaults to displaying all messages in the console, which can get a bit overwhelming. Fortunately you can suppress this so only error messages are displayed by setting the 'msgDisplayLevel' fieldname to 1.

Crew Views

Since they kindly obliged us with their assistance in the last review of an individual trading API we conducted (Bloomberg Tradebook's in Q3 2010) we thought we'd bully Wrecking Crew occasionalists Steve K and Martin S into participating again. The scar tissue must have healed, because much to our surprise they both agreed. Thanks gents.

STEVE K (prop trader, European bank): Very simple to install - just drop the IBMatlab.jar file in your static Java classpath and drop two other files into any folder on the MATLAB path and off you go. IB-MATLAB suits the type of multi leg strategy that we normally use on a small stat arb portfolio I'm responsible for; even though that results in a fair number of orders and amends flying around, the holding period for our trades is usually measured in hours not milliseconds.

Though it obviously slows things up a bit, the deliberate pauses built into the underlying IB-MATLAB Java code are probably wise. Our models aren't usually chasing the same inefficiencies as everyone else so we aren't that time sensitive. On the other hand, because each trade can be quite complex in terms of leg count, something that fell

over easily would cause all sorts of housekeeping and order management issues.

MARTIN S (programmer/quant, US proprietary trading firm): We already use the Interactive Brokers' FIX CTCTI API having migrated to that from the IB API about a year ago. While both these APIs work fine, I wouldn't describe either as particularly easy to get to grips with. The documentation is good as it goes, but (particularly with the IB API) you tend to get some rather cryptic error messages that don't help much with the debugging process. For that reason, I can see IB-MATLAB having considerable appeal for a lot of professional traders as it hugely streamlines productivity by wrapping a lot of functionality into some very sparse syntax. As a result, it would take very little time to get a model deployed in the market using the product. The slight downside is that not every last piece of IB API functionality is accessible, but I'd say that IB-MATLAB covers the bases needed by the vast majority of traders.

Thanks also to the other two members of this issue's Wrecking Crew. Sorry we couldn't include your comments, but your frustration at being unable to break anything (other than my desk) means that your views didn't quite make it past the Editor's new profanity checker. There's always a next time...

Pumping trades

In order to test order throughput, we assembled the following rather pointless MATLAB function:

```

function IB_testtrade1
stk = load('dow_stks.mat');
for i = 1:100
    rndsec = randi(15,1);
    data = IB_trade('action',
'QUERY', 'symbol',stk.
stk{rndsec,1});
    orderId = IB_trade
('action','BUY', 'symbol',stk.
stk{rndsec,1},
'quantity',100,
'limitPrice',data.bid);
    pause(1);
end
end

```

This opens and loads the contents of a '.mat' file into a two column struct called 'stk' (see Figure 2). It then loops 100 times, while on each iteration it generates a new random integer which it uses as a row reference to select a stock symbol from the first column of the 'stk' struct. It then retrieves a current data snapshot for the symbol and uses that as the basis for an attempt to buy 100 shares on the bid. It then pauses for one second before starting the next iteration.

AA	JPM
AXP	KFT
BA	KO
BAC	MCD
CAT	MMM
CSCO	MRK
CVX	MSFT
DD	PFE
DIS	PG
GE	T
HD	TRV
HPQ	UTX
IBM	VZ
INTC	WMT
JNJ	XOM

Figure 2

The Virtue of Simplicity

Contract	Last	Change	Change (%)	Bid Size	Bid	Ask	Ask Size	Position	Avg Price	P&L
AA	15.97	+0.26	1.65%	420	15.97	15.98	707	300	15.98	
CSIQ	15.68	-0.08	-0.51%	570	15.68	15.69	2,394	0		
IBM	175.65	175.67		4	175.65	175.67	4	600	175.702	
GE	18.69	18.69		1,075	18.69	18.69		100	18.76	
BA	72.64	72.65		6	72.64	72.65	8	400	72.65	
AAPL	52.63	52.63		23	52.63	52.63		400	52.63	
DD	54.92	54.92		50	54.92	54.92		100	54.90	

Figure 3

Rather predictably, this quickly filled up the pending orders tab of the Interactive Brokers' Trader Workstation (see Fig 3). However, it didn't cause any stability problems. So just for the sheer irresponsible hell of it, we removed the pause, and were disappointed when it didn't appear to make any difference. We then of course realised there was another inherent pause in the code, in that the order execution line would not run until it had a populated 'data' struct to work with.

So in a concerted attempt to break something the interests of academic research, we did away with the data gathering step *and* switched to market orders *and* removed the pause:

```
function IB_testtrade2
stk = load('dow_stks.mat');
for i = 1:100
    rndsec = randi(15,1);
    orderId = IB_
trade('action','BUY','symbol',stk.
stk{rndsec,1},
'quantity',100, 'type','MKT');
end
end
```

Nope - rock solid.

So in a final attempt at chaos, we added a short sale to the code (symbol selected from column two of Figure 2) and also jacked up the order quantity for buys

and short sells, so we would have multiple partial fill messages coming back through the interface:

```
function IB_testtrade3
stk = load('dow_stks.mat');
for i = 1:100
    rndsec = randi(15,1);
    % Long leg
    orderId = IB_
trade('action','BUY','symbol',stk.
stk{rndsec,1}, 'quantity',10000,
'type','MKT');
    % Short leg
    orderId = IB_
trade('action','SELL',
'symbol',stk.stk{rndsec,2},
'quantity',10000, 'type','MKT');
end
end
```

Stop Press: Update on Streaming Quotes

The version of IB-MATLAB reviewed here does not support streaming quotes, so data requests have to be made individually. However, just as we went to press, we received an update from Yair Altman that a new version of IB-MATLAB was currently under development that does support streaming quotes.

Obviously the number of instruments for which you will be able to retrieve real time quotes will depend upon which instruments you choose. Even a small portfolio of stocks such as Google and IBM will quickly mop up the IB API's published capacity of 50 messages per second. If on the other hand you specialise in esoteric African stocks that print one trade a month, you should have plenty of elbow room.

Other factors to bear in mind will be network bandwidth, computer speed and of course what level of data you have subscribed for with your IB account.

But nothing - not even a flicker. Just an orderly procession of orders going out and an orderly procession of fills coming back; all very ~~frustrating~~ interesting.

Readers of Automated Trader will by now be aware that the Wrecking Crew's approach to software testing is probably best termed 'robust'. If they can't make something crash by doing something completely irrational and dangerous, they tend to get a little moody (i.e. start kicking the office furniture to matchsticks). IB-MATLAB's wholly unreasonable refusal to make either MATLAB or IB's Trader Workstation fall over therefore irked them not a little.

However, as an alternative to mindless violence, the more cerebral element of the Wrecking Crew (total headcount: 0.01) suggested speaking to IB-MATLAB's developer, Yair Altman to see if he could shed any light on this irritating stability. Mr Altman has clearly met people like the Wrecking Crew before - he also evidently lives in the real world. It emerged that when you send a message (such as an order or data request) to Interactive Brokers through their API, you don't necessarily get a single message containing the entire response in return. Therefore the IB-MATLAB code contains a number of deliberate delays to allow all the various message fragments time to arrive, after which they are consolidated and passed back to MATLAB.

Needless to say, certain members of the Wrecking Crew felt that this wasn't really playing the game, but even they grudgingly admitted that trading software that doesn't turn up its toes at the drop of a hat is actually not a bad idea...

Orders - coverage and tracking

Even the more jaded members of the Wrecking Crew admitted (through gritted teeth) that IB-MATLAB's syntax is intuitive. It says something that the explanation of this syntax in the help file consists of less than 100 lines. One nice touch that definitely assists this brevity is the way that it leverages existing Interactive Brokers API syntax. For example, its implementation of callback functions is ▶

IB-MATLAB User Case Study: Delta Hedging

*Andrea Gentilini, Senior Portfolio Manager,
Union Bancaire Privée, Geneva*

My colleague Cristiano Migliorini and I originally started using IB-MATLAB for a private proprietary trading project that involved the need to delta hedge an option portfolio automatically. We didn't want to have to write all the necessary code to interface Interactive Brokers and MATLAB ourselves, so were looking for a simpler and faster route to achieving this. Finding IB-MATLAB was therefore something of a blessing for us, as we found it very intuitive to install and use - to the extent that we hardly needed to refer to the help files.

Before switching to live trading, we used IB-MATLAB to conduct simulated trading via an Interactive Brokers' paper trading account. After a couple of days, we were confident that it would work as intended and so switched to live trading through Interactive Brokers. It wasn't long before we felt comfortable turning things on first thing in the morning before going out and leaving IB-MATLAB all day trading a live account. The IB-MATLAB connection proved extremely robust and capable of handling connection outages that sometimes occurred during the day.

We were running an equity options portfolio that required delta hedging. We didn't actually need real-time streaming market data for our purposes, so we set up our hedging code in MATLAB so that it triggered a data request via IB-MATLAB every two minutes. This retrieved the net delta of our option portfolio; if it was above or below a certain threshold this would then trigger a hedging order to buy or sell stock index futures - either DAX or EURO STOXX 50. On the basis of this two minute data sampling we were probably doing on average five or six futures trades per day. Our control algorithms were programmed to handle bad ticks, gross errors, failed or partial executions, loss of connection, and provided a very stable fully unsupervised hedging machine.

We eventually stopped using IB-MATLAB, but only because we had to retire the options strategy due to the time pressure we were under on other projects. There certainly wasn't a problem with the software, which we used for every trading session over about six months and in that time it performed reliably and without problems.

The Virtue of Simplicity

based upon the string 'callbackXXX', where 'XXX' can be replaced with any one of more than 30 native IB API functions, including contract details, fundamental data, real time data, portfolio updates, market depth updates, order status and news bulletins.

A similar position applies to order types. While IB-MATLAB doesn't support every order type that Interactive Brokers offers, it still manages to cover:

- MKT - Market
- MKTCLS - Market-on-Close
- LMT - Limit
- LMTCLS - Limit-on-Close
- PEGMKT - Pegged-to-Market
- STP - Stop
- STPLMT -
- TRAIL -Trailing Stop
- REL - Relative
- VWAP - VWAP Best Effort
- TRAILLIMIT - Trailing Market If Touched
- SCALE - Scale
- GuaranteedVWAP - VWAP Guaranteed

Among many other things, the range of callback functions available in IB-MATLAB make keeping track of submitted orders pretty straightforward. For example, the following code tags a couple of callback functions onto our eventually (*see box "API Tricks of the Trade: Non-US instruments"*) successful order for ArcelorMittal.

```
[orderId, ibConnectionObject]
= IB_trade('action', 'BUY',
'symbol', 'MT', 'exchange', 'AEB',
'currency', 'EUR', 'sectype', 'STK',
'quantity', 10000, 'type', 'MKT',
'CallbackOpenOrder', @OpenOrdersFn,
'CallbackOrderStatus', @
OrderStatusFn);
```

The event data triggered in response to 'CallbackOpenOrder' and 'CallbackOrderStatus' can be captured and manipulated by writing two functions (OpenOrdersFn and OrderStatusFn) in MATLAB to handle it. This can be achieved by using the orderId and the ibConnectionObject returned by the IB-MATLAB after either typing the following at the MATLAB console or including it in a function:

```
ibConnectionObject.reqOpenOrders
```

This will invoke the two callback functions with the separate OrderStatus and OpenOrders events and the corresponding EventData can be searched by using the returned orderId value. One of the partial fill messages returned by this method for the ArcelorMittal order above is shown below.

```
orderId =89900419
orderStatus: orderId =89900419
orderStatus: status =Submitted
orderStatus: filled =7461
orderStatus: remaining =2539
orderStatus: avgFillPrice =22.67
orderStatus: permId =113248631
orderStatus: parentId =0
orderStatus: lastFillPrice =22.67
orderStatus: clientId =140475
orderStatus: whyHeld =
```

Details of open positions can be quickly accessed by using the following portfolio query syntax:

```
data = IB_trade('action',
'PORTFOLIO_DATA')
```

This returns a MATLAB struct with each element within the struct containing the details of one security in the portfolio, as shown in Figure 4 for a position in Caterpillar.

Field	Value	Min	Max
symbol	'CAT'		
exchange	'NYSE'		
secType	'STK'		
currency	'USD'		
position	300	300	300
marketValue	32613	32613	32613
marketPrice	108.7100	108.7100	108.7100
averageCost	108.7367	108.7367	108.7367

Figure 4

Conclusion

So, do we like it? Well, IB-MATLAB is robust, very easy to learn how to use and does exactly what it claims to do - namely provide a simple and efficient order interface between MATLAB and Interactive Brokers' IB API. It also costs peanuts; a license for a single computer for the first year costs \$250, which includes installation support, fixing bugs, and any fixes that may be required due to IB API changes. Annual renewal, which includes license, support and maintenance for subsequent years costs \$100 per annum.

So yes, we like it - a lot.



API Tricks of the Trade: Non-US instruments

While the default format for orders shown in the body of the main review work fine for US securities, the Wrecking Crew developed some anger management issues when they tried to replicate their testing on European stocks. The original idea was to put together a pairs portfolio of NYSE Euronext stocks and then trigger buy and short sell orders at random time points for each pair.

We started by testing simple data retrieval for a single stock - AkzoNobel (listed as symbol AKZ in the IB symbol directory) which primarily trades in Amsterdam on the AEB. So in line with our successful retrieval of Google we entered:

```
data = IB_trade('action',
'QUERY', 'symbol', 'AKZ')
```

Nope. All we got back was:

```
error: message =No security definition
has been found for the request
data =
```

```
reqId: 89874462
reqTime: '15-Jul-2011
13:57:25'
dataTime: ''
dataTimestamp: -1
ticker: 'AKZ'
bid: -1
ask: -1
open: -1
close: -1
low: -1
high: -1
last: -1
volume: -1
tick: 0.0100
```

Hmmm. The confusion only became worse when we tried a simple market order for another AEB stock - ArcelorMittal (symbol MT). No error message in the console this time, but a pop up box appeared warning that the order would not be submitted until the start of US trading hours. Why? A bit of digging quickly revealed that ArcelorMittal also trades on NYSE, even though its primary listing is on the AEB. But the symbols for both incarnations of the stock were identical - MT. Attempts to place a manual order or view ArcelorMittal in the Trader Workstation showed the various possible alternatives that could be manually selected (see Figure 5) but how could we replicate this selection of the AEB version of the stock via the API? Our assumption that the default would be the stock's primary listed venue was clearly wrong.

So we tried explicitly setting the 'exchange' field in our API order with:

```
orderId = IB_trade('action', 'BUY',
'symbol', 'MT', 'exchange', 'AEB',
'quantity', 10000, 'type', 'MKT');
```

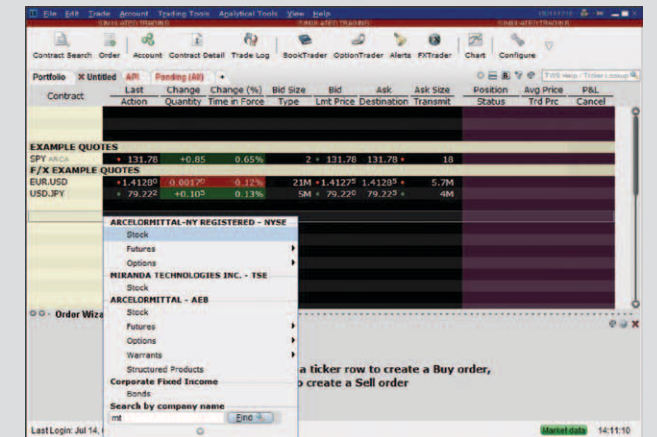


Figure 5

...but that didn't work either. After considerable further swearing, we found that IB defaults to the US version of dual listed stocks and if you want the non-US alternative, you really have to specify chapter and verse as follows:

```
[orderId, ibConnectionObject] = IB_
trade('action', 'BUY', 'symbol', 'MT',
'exchange', 'AEB', 'currency', 'EUR',
'sectype', 'STK', 'quantity', 10000,
'type', 'MKT');
```

So we then also applied the same principle to our original AkzoNobel data request with:

```
data = IB_trade('action', 'QUERY',
'symbol', 'AKZ', 'exchange', 'AEB',
'currency', 'EUR', 'sectype', 'STK')
```

...and bingo, we finally retrieved our data:

```
data =
reqId: 89879529
reqTime: '15-Jul-2011 14:21:44'
dataTime: '15-Jul-2011 14:21:45'
dataTimestamp: 7.3470e+005
ticker: 'AKZ'
bid: 42.0950
ask: 42.1150
open: 42.2150
close: 42.3750
low: 41.9600
high: 42.3000
last: 42.1000
volume: 240153
tick: 1.0000e-003
```

So the long and short of it is that if you want to deal with the non-US flavour of a stock via the IB API, then in addition to the symbol you have to specify the exchange, the currency and the fact that it's a stock. This isn't in any way a failing on the part of IB-MATLAB, it's just the way that Interactive Brokers chooses to do things - but those API trading non-US markets via IB will hopefully save themselves some teeth grinding by reading this box first...