MATLAB as an Automated Execution System

By Ernest P. Chan, Ph.D.

Many traders are familiar with MATLAB as a powerful software platform for backtesting trading strategies. This is especially true for those who would like to analyze and trade a large number (maybe thousands) of stocks simultaneously. MATLAB is a language that is built around matrix manipulation and processing, so many calculations involving multiple stocks are just as easy as calculations involving a single stock.

In my book *Quantitative Trading* (Wiley 2008), I have described a number of examples of how backtesting is usually done in MATLAB. However, it was also true that MATLAB suffered from a major deficiency relative to more familiar trading platforms such as TradeStation – after a strategy has been backtested, it wasn't easy to immediately turn it into an execution system and submit orders to your brokerage account. Brokerages that support Application Program Interfaces (API) to various other languages such as Visual Basic, Java, C# or C++ often does not support MATLAB. Therefore, building an automated execution engine involves re-programming the strategy in one of those languages that your brokerage supports, thereby losing the simplicity and ease-of-use of MATLAB.

Recently, however, I have discovered a piece of software that can seamlessly connect a MATLAB program to a brokerage account, allowing MATLAB to receive market data, execution and position reports from the brokerage, and to send orders for execution to the brokerage. In other words, turning MATLAB into an automated execution engine. This program is called Matlab2IBAPI (www.exchangeapi.com), created by Exchange Systems Inc. As its name imply, this program is built specifically for connection to Interactive Brokers (IB) – but given the widespread adoption of IB as an effective platform for algorithmic traders, this limitation is minor compared to the speed of building an execution engine. (Exchange Systems Inc. also created another API called MATLAB2TT for direct connection to futures exchanges using Trading Technologies International's FIX adaptor, but this is beyond the scope here.)

The use of an automated trading system (ATS) is advantageous but optional if your strategy generates orders infrequently. For example, if you send orders only near the market open or close, then you can simply specify all your orders on an Excel spreadsheet, load this spreadsheet up to a basket trader such as the one available from IB, press a button and have the basket trader send out all the orders on the spreadsheet in one fell swoop. This process does involve quite a few manual steps. Using an ATS in this case will allow you to generate and submit orders by one push of a button, and so will certainly make life much easier, though it is not essential. Of course, if you are running a few strategies that all require such manual steps simultaneously, an ATS may become essential even for such strategies with infrequent orders.

If your strategy generates orders at all times of the day, or if it generates orders at high frequency, then an ATS becomes critical. An example is a simple mean-reverting Bollinger-band strategy on ES (the S&P E-mini futures). The strategy simply buys when the price of ES hits the lower Bollinger band (determined by ten 1-minute lookback periods, and 2 standard deviations from the moving average), and sells when the price increases by 1 standard deviation from the moving average. Conversely, it shorts when ES hits the upper band, and then buy-cover when the price decreases by 1 standard deviation. I have backtested this strategy before, and as I mentioned in Chapter 2 of my book, its Sharpe ratio is a very attractive 3 without transaction cost, but is reduced to -3 if we subtract 1 basis point per trade as transaction cost. Even though the strategy as it stands will not survive commissions, not to mention bid-ask spread and slippage, it serves as a good illustration of how to use MATLAB in conjunction with Matlab2ibAPI to build an automated trading system. Furthermore, if you are ingenious enough, you can always take a basic strategy and add enough bells and whistles to make it profitable even in the face of transaction costs.

One of the beauties of an ATS is that you can start the program at your leisure and well in advance – MATLAB can wake itself up at the right time. Furthermore, MATLAB can automatically exit all existing positions and disconnect from your brokerage account near the market close. In other words, this is truly a turnkey trading program.

First, we define a few parameters for the program.

```
localSymbol='ESH9'; % Symbol for the front contract
symbol='ES'; % IB symbol for this future
secType='FUT'; % security type
exchange='GLOBEX';

accountName='U123456'; % This is your Interactive Brokers account name
numContract=1; % Your desired order size

lookback=10; % Number of periods used to compute Bollinger band
entryZ=2; % The distance between the upper and the lower band is
          % 2*entryZ
exitZ=1; % Exit when price revert to 1 standard deviation from the mean

starthh=9; % Start running strategy at 9:30 ET.
startmin=30;

endhh=16; % Stop running strategy at 16:14:59 ET.
endmm=14;
endss=59;

connect2TWS % Start a new connection to IB
pause(20) % need to pause program to wait for connection to be
          % established


while (1) % Wait until 2 minutes before 9:30 to begin program
    mytime=clock;
    myyyyy=mytime(1);
    mymm=mytime(2);
    mydd=mytime(3);
    myhh=mytime(4);
    mymin=mytime(5);

    if (myhh > starthh || (myhh == starthh & mymin >= startmin-2))
        break;
    end
end
```

OK, finally it is time to starting getting information from your brokerage account. Note that IB requires a subscription to each symbol (ES in our case) before you can receive market data updates.

```
requestAccountUpdates( 1,accountName); % Start receiving account
                                       % positions updates
reqMktData(1, symbol, secType, exchange, '', 0, '', '', localSymbol);
% Start receiving market data for this contract

isUpdated=false; % Indicate whether we have updated the last price in
                 % the current 1-minute bar
```

Now for the main loop for sending orders.

```
for i=1:Inf
    mytime=clock; % Unfortunately matlab2ibapi cannot yet retrieve
                  % server time from IB, hence we need to use local
                  % time

    % Date and time strings for use in requesting historical data
    myyyyy=mytime(1);
    mymm=mytime(2);
    if (mymm < 10)
        mymmstr=['0', num2str(mymm)];
    else
        mymmstr=num2str(mymm);
    end

    mydd=mytime(3);
    if (mydd < 10)
        myddstr=['0', num2str(mydd)];
    else
        myddstr=num2str(mydd);
    end

    myhh=mytime(4);
    mymin=mytime(5);
    myss=mytime(6);
    myss=str2double(regexprep(num2str(myss), '\..*', ''));

    if (i==1) % As a start, see what we already own in account
        port=getPortfolioUpdates(accountName, symbol);
        if (~isempty(port))
            assert(strcmp(port.symbol, symbol));
            pos=port.position;
        else
            pos=0;
        end
    end

    if (myhh >= endhh) % Near market close, exit all positions
        if (pos>0)
         % Send Market Sell orders to exit
         id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
            localSymbol,'SELL',pos,'MKT',0,0,'DAY','');
        elseif (pos<0)
         % Send Market Buy orders to exit
         id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
            localSymbol,'BUY',abs(pos),'MKT',0,0,'DAY','');
        end
        pos=0;
        break;
    end

    if (i==1) % First time around, get 1 day’s worth of 1-min bar
              % historical data from IB

        [es]=reqHistData(i, symbol, secType, exchange, '', 0, '', '',…
             localSymbol, [num2str(myyyyy) mymmstr myddstr ' ' …
             num2str(myhh) ':' num2str(mymin) ':' num2str(myss)], …
             '1 D','1 min','TRADES',1,1,2);
        tic;
    else % Thereafter, simply get last price every 60 seconds
        elapsed=toc;
        if (elapsed >= 60) % sample trade prices every minute
            lastPrice=getLastPrice(1);

            % Update historical price array with last price
            es.close(1:end-1)=es.close(2:end);
            es.close(end)=lastPrice;
            tic;
            isUpdated=true;
        end
    end

    if (i==1 || isUpdated)
        cls_hist=es.close; % Update historical price array with last
                           % price
        cls_hist=cls_hist(end-lookback+1:end);
        ma=mean(cls_hist); % moving average
        mstd=std(cls_hist); % moving standard deviation

        isUpdated=false;
    end

    askPrice=getAskPrice(1);
    bidPrice=getBidPrice(1);
    lastPrice=getLastPrice(1);
```

```
    % Calculate deviation of ask or bid price from moving average
    zscoreAsk=(askPrice-ma)./mstd;
    zscoreBid=(bidPrice-ma)./mstd;

   % Check if this is time to send orders
    if (myhh == starthh & mymin < startmin)
        continue;
    end

    if (pos==0 & zscoreAsk < -entryZ ) % entry only
        id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
           localSymbol, 'BUY', numContract, 'MKT',0,0,'DAY','');
        pos=numContract;
    elseif (pos < 0 & zscoreAsk < -entryZ) % exit and entry
        id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
           localSymbol, 'BUY', 2*numContract, 'MKT',0,0,'DAY','');
        pos=numContract;
    elseif (pos < 0 & zscoreAsk < -exitZ) % exit only
        id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
           localSymbol, 'BUY', numContract, 'MKT',0,0,'DAY','');
        pos=0;
    elseif (pos==0 & zscoreBid > entryZ) % entry only
        id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
           localSymbol, 'SELL', numContract, 'MKT',0,0,'DAY','');
        pos=-numContract;
    elseif (pos > 0 & zscoreBid > entryZ ) % exit and entry
        id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
           localSymbol, 'SELL', 2*numContract, 'MKT',0,0,'DAY','');
        pos=-numContract;
    elseif (pos > 0 & zscoreBid > exitZ ) % exit only
        id=placeOrder(symbol,secType,exchange, '', 0, '', '',…
           localSymbol, 'SELL', numContract, 'MKT',0,0,'DAY','');
        pos=0;
    end

end

disconnectFromTWS;
```

So this is the complete turnkey automated trading system for a Bollinger band strategy for trading ES. (This program can be downloaded from my website epchan.com/book/bollinger.m if you have purchased my aforementioned book, using the same password for all the other programs.) All you need to do is to start this program before the market opens everyday, sit back, and relax. As I said, it isn't likely to be profitable as it stands, but you can probably find enough improvements to make it so.

You can use this Matlab2api to trade a large number of stocks simultaneously throughout the day too, thus turning into a true statistical arbitrageur. However, if your strategy trades at high frequency, you will run into some limitations of the IB-Matlab-Matlab2IBapi combination. First, if your account does not generate more than $800 commissions a month for IB (or have more than $1M in equity), you can only get live market data feed on 100 symbols at a time. So if you trade more than 100 symbols, you have to rotate those symbols every few seconds. Second, it will take a second or more to get updates on your current account positions, or order status. In other words, the IB-Matlab-Matlab2IBapi combination won't work for strategies that have expected holding period of less than a second. Despite this limitation, I hope you will agree with me that Matlab2IBapi provides you with a powerful and convenient platform for implementing most intraday strategies.

About the Author

Ernie is the founder of the trading strategy consulting firm E.P. Chan & Associates (www.epchan.com) and co-founder of Chicago-based EXP Capital Management, LLC. He is also the author of *Quantitative Trading* (Wiley, 2008).