Implement, Backtest and Analyze a Technical Trading Strategy with MATLAB

This example provides an introduction to MATLAB in implementing and backtesting a simple algorithmic trading strategy. The steps include importing historical data, building custom interactive charts of technical indicators, implementing a custom strategy, backtesting and optimizing parameters and analyzing the backtest results. This demo will walk you through these steps and culminate in the development of an automated MATLAB script that runs the entire analysis and generates a custom report.

This examples requires the following products

* Database Toolbox (for importing data from a Database, not necessary if using the MAT-file)
* Financial Toolbox (for the technical indicators and maximum drawdown calculations)
* Statistics Toolbox (for the percentile calculations)

If you do not have all the toolboxes, you can still view the result of the analysis in the following report

[$\html\marisaBacktest.html](html/marisaBacktest.html)

# Step 1: Import Historical Data

The first step of the demo involves importing data from a database. However, to reduce the size of the demo packet, the database is not included and a MATLAB MAT-file containing the data is provided instead. The file *getBundData.m* contains a function auto-generated by the Database Toolbox Visual Query Builder tool which shows the commands that can be used to connect to virtually any database known to your system. This function can be modified to be used with your own in-house databases.

To import data from the MAT-file, make the demo folder the MATLAB current folder and simply double-click it in the MATLAB Current Folder window or use the command

load bund1min

Extract the close prices (and subsample the data) by running the commands,

step = 60;

Close = data.Close(1:step:end);

# Step 2: Create a Custom Indicator Chart

The script *createIndicatorPlot.m* contains the commands you can use to create a custom plot of the Close prices, the exponential moving average and relative strength index. Plots such as these can be interactively laid out using the MATLAB plot picker and plot tools. The commands can then be auto-generated to create a template function such as *createIndicatorPlot*.

Try exploring the chart with zooming, panning, data linking and data brushing. These tools help you build a better intuition for your data and indicators and help find trading opportunities. For instance, notice how bullish-to-bearish trend reversals tend to correspond with the RSI crossing the 55-point threshold in the downward direction.

# Step 3: Implement a Trading Strategy

The function *marisa* contains the implementation of a trading strategy based on the moving average (MA) and relative strength index (RSI) – hence the name MARISa. This function creates long and short signals based on the MA and RSI independently. Only when both indicators agree does the strategy take a long or short position. The instantaneous profit and loss (PNL) and Sharpe’s ratio are calculated, both of which include a trading cost.

Calling the function for one choice of moving average and RSI window parameters runs the backtest and returns the positions and PNL. The following commands runs a backtest on the Close prices with a 200-period MA, a 50-period RSI and a trading cost of .01 units. The PNL and positions are then plotted with another custom visualization

N = 200;

M = 50;

[sh pnl pos] = marisa(Close, N, M, 0.01);

posPNLPlot(Close, pos, pnl);

fprintf('Sharpe''s Ratio: %0.2f\n\n', sh \* sqrt(60\*11/step));

# Step 4: Backtest and Optimize Parameters

At this point it is easier to step through the cells of the script *marisaBacktest* to run the strategy for multiple choices of the M and N parameter and visualize a Sharpe’s ratio surface of the performance of the strategy. The optimal choice of parameters (the one that yields the highest Sharpe’s ratio) is made and the strategy is rerun to further analyze the results.

## Parallel Computing

The double for-loop used for performing the parameter sweep over M and N can easily be parallelized by converting the outer loop into a parfor. This will use a MATLAB pool if available to run the parameter sweep in parallel. Parallel Computing Toolbox will be required to take advantage of this functionality.

# Step 5: Analyze the Backtest Results

The last two cells of the script *marisaBacktest* analyze the positions generated by the strategy for the optimal choice of parameters. The statistics calculated include the average position duration, the maximum drawdown, and a distribution of profits from each position. This enables the trader to gain further insight into the strategy and its performance.

# Step 6: Automate the Analysis

The entire analysis can be automated to the click of a button. By clicking the *publish* button or using the *publish* command, the script *marisaBacktest* is run and an automated report is generated with the results of the backtest, the parameter optimization and strategy position analysis. An example of such a report is available in the *html* folder in the demo packet.

Copyright 2013 MathWorks, Inc.