WOLFRAM WHITE PAPER

Backtesting Trading Strategies Using Wolfram *Finance Platform*™

WOLFRAM

Introduction

Wolfram *Finance Platform*'s capabilities in data acquisition and management, computation, and report generation make it the platform of choice for implementing a backtesting engine to simulate trading strategies. Thanks to its interoperability with languages like C++, Java, and R, quantitative analysts can use their *Finance Platform* code in a heterogeneous environment. In this paper we explore one possibility for evaluating portfolio performance using Wolfram *Finance Platform* and describe the thought process behind it.

Data Acquisition

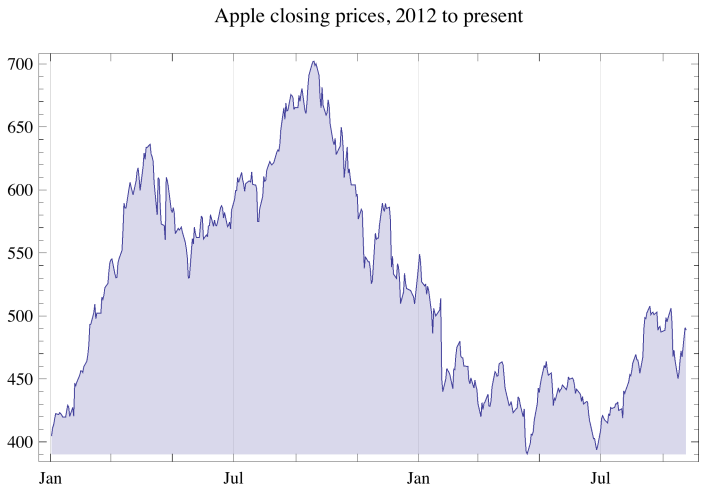
WolframAlpha integration

There are many ways to obtain financial data through Wolfram *Finance Platform*. In the example below, we directly access Apple's stock prices from 2012 to today using the built-in WolframAlpha functionality:

```
WolframAlpha["AAPL from 2012 to today",
{{"DateRangeSpecified:Close:FinancialData", 1}, "TimeSeriesData"}];
```

These can be readily plotted directly in Wolfram *Finance Platform*:

```
DateListPlot[%, Joined -> True, Filling -> Axis, PlotLabel -> "Apple closing
prices, 2012 to present"]
```



This integration is extremely powerful, making financial data, economic time series, weather, and geographical data available to Wolfram *Finance Platform* users.

FinancialData function

We can also use the built-in `FinancialData` function to obtain the same data for Apple's stock prices from 2012 to today. Note that `FinancialData` is one of those superfunctions that collate data from a wide range of sources and is described at reference.wolfram.com/mathematica/ref/FinancialData.html:

```
data = FinancialData["AAPL", "OHLCV", {{2012, 1, 1}, DateList[]}];
TradingChart[data]
```

See CDF version for dynamic data display with mouseover.

BloombergLink

We can use *Finance Platform*'s *BloombergLink* to access intraday data, present it in a table, and plot it:

```
Needs["BloombergLink`"]
MarketDataConnect[BloombergDesktopAPISource];
```

Request a snapshot from the data source:

```
MarketDataGet["Snapshot", "Instruments" -> {"IBM Equity", "AAPL Equity", "BNNY
Equity"}, "Fields" -> {"CUR_MKT_CAP", "LAST_PRICE"}]
```

```
MarketDataTable[{{IBM Equity, 2.24664 × 10^11, 194.77},
  {AAPL Equity, 5.47913 × 10^11, 585.964},
  {BNNY Equity, 6.23376 × 10^8, 36.54}},
 {id, CUR_MKT_CAP, LAST_PRICE}, Snapshot]
```

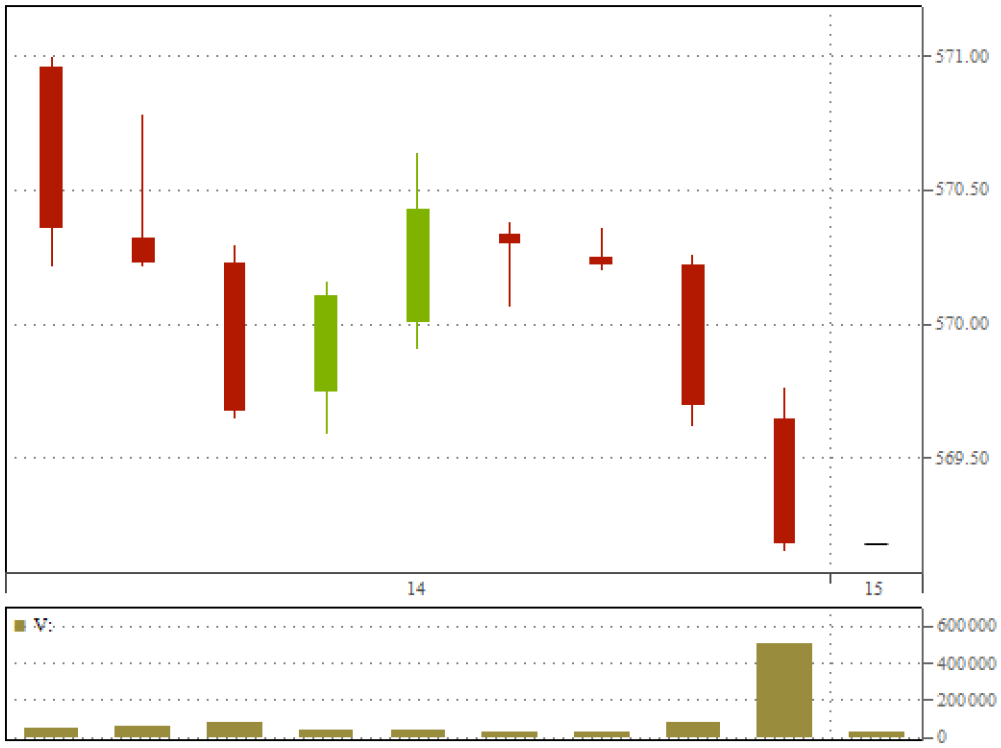
Create a table using `Grid`:

```
Grid[%]
```

Snapshot		
id	CUR_MKT_CAP	LAST_PRICE
IBM Equity	2.24664×10^{11}	194.77
AAPL Equity	5.47913×10^{11}	585.964
BNNY Equity	6.23376×10^{8}	36.54

The result from the command `MarketDataSubscribe` can be directly applied to `TradingChart` to generate a candlestick OHLC chart with a volume bar chart:

```
subscription = MarketDataSubscribe["MarketBarSubscription",
  "Instrument" -> "AAPL US Equity",
  "Fields" -> "LAST_PRICE"];
MarketDataSubscriptionStart[subscription]
TradingChart[MarketDataGet[subscription]]
```



Database connectivity

Using *DatabaseLink*, we can connect to an SQL database to obtain data. The following function shows how to query a database to retrieve prices for a list of tickers:

```
Options[LoadDataFromDatabase] = {ConnectionName -> "", Tickers->{}};

LoadDataFromDatabase[universe_List, fields_List, outputNames_List, dataPer_List,
  opts : OptionsPattern[]] :=
    Module[{tickers, strFlds, connName, strSQLQuery, conn, rawData, stDate,
        enDate, dataPeriod},

      tickers = OptionValue[Tickers];
      If[Length[tickers]==0,
        tickers = universe;
      ];

      dataPeriod = dataPer;
      If[! ((dataPeriod === All) || ((Head[dataPeriod] == List) &&
        (Length[dataPeriod] == 2))),
        Message[LoadDataFromDatabase::dtper, dataPeriod];
        dataPeriod = All;
      ];

      If[Length[fields] > 0,
        strFlds = "pr_date";
        (strFlds = StringJoin[strFlds, ",", #]) & /@ fields;

        If[dataPeriod === All,
          strSQLQuery = "ORDER BY internalID,pr_date;";
        ,
          stDate = DateString[dataPeriod[[1]], {"Year", "-", "Month", "-", "Day"}];
          enDate = DateString[dataPeriod[[2]], {"Year", "-", "Month", "-", "Day"}];
          strSQLQuery = "AND (pr_date>='" <> stDate <> "') AND (pr_date<='" <> enDate
            <> "') ORDER BY internalID,pr_date;";
        ];

        connName = OptionValue[ConnectionName];
        conn = OpenSQLConnection[connName];
        rawData = {#, outputNames, SQLExecute[ conn, "SELECT " <> strFlds <>
          " FROM Prices WHERE internalID='" <> # <> "' " <> strSQLQuery] /.
          {SQLDateTime[x_] -> x}} & /@ tickers;

        CloseSQLConnection[conn];

        Return[rawData]
      ,
        Return[{}]
      ];
    ];
```

Note the use of pattern matching that easily converts the `SQLDateTime` object into a Wolfram Language date (`/. {SQLDateTime[x_] -> x}`).

Data Preprocessing

After acquiring data, it is preprocessed to detect anomalies such as missing data or outliers. Using multiple sources is usually a good remedy for these issues, but is not always an option. Although problem-dependent, preprocessing can be simplified with Wolfram *Finance Platform*'s pattern-matching functionality.

Another preprocessing task is data preparation. Data needs to be transformed depending on the model used. Data normalization, for example, makes data comparable and avoids overweighting a variable because of its relatively large amplitude. Another example is dimensionality reduction, in which the initial dataset is embedded in a new, smaller space, providing better models and a more concise representation of the data.

The following sequence demonstrates preprocessing of raw data:

```
data = {{{2010, 1, 4}, 208.14`}, {{2010, 1, 5}, 208.5`}, {{2010, 1, 6}, 205.18`},
{{2010, 1, 7}, 204.8`}, {{2010, 1, 8}, 206.16`}, {{2010, 1, 11}, 204.34`},
{{2010, 1, 12}, 202.02`}, {{2010, 1, 13}, -204.87`}, {{2010, 1, 14}, 203.68`},
{{2010, 1, 15}, 20028}, Missing[]};
```

Use `Cases` to remove data that contains non-numeric values:

```
data = Cases[data, {_String | _List, _?NumericQ}]
```

Use `Select` to remove numeric data points that could be numerically incorrect:

```
Select[data, 0 < #[[2]] < 1000 &]
```

In this next sequence, we use WolframAlpha as another source to validate data:

```
data = {{{2010, 1, 4}, 214.01`}, {{2010, 1, 5}, 214.38`}, {{2010, 1, 6},
210.97`}, {{2010, 1, 7}, 210.58`}, {{2010, 1, 8}, 211.98`}, {{2010, 1, 11},
210.11`}, {{2010, 1, 12}, 207.72`}, {{2010, 1, 13}, 210.65`}, {{2010, 1, 14},
209.43`}, {{2010, 1, 15}, 205.93`}, {{2010, 1, 19}, 215.04`}, {{2010, 1, 20},
211.73`}, {{2010, 1, 21}, 208.07`}, {{2010, 1, 22}, 197.75`}, {{2010, 1, 25},
203.07`}, {{2010, 1, 26}, 205.94`}, {{2010, 1, 27}, 207.88`}, {{2010, 1, 28},
199.29`}, {{2010, 1, 29}, 192.06`}, {{2010, 2, 1}, 194.73`}, {{2010, 2, 2},
195.86`}, {{2010, 2, 3}, 199.23`}, {{2010, 2, 4}, 192.05`}, {{2010, 2, 5},
195.46`}, {{2010, 2, 8}, 194.12`}, {{2010, 2, 9}, 196.19`}, {{2010, 2, 10},
195.12`}, {{2010, 2, 11}, 198.67`}, {{2010, 2, 12}, 200.38`}, {{2010, 2, 16},
203.4`}, {{2010, 2, 17}, 202.55`}, {{2010, 2, 18}, 202.93`}, {{2010, 2, 19},
201.67`}, {{2010, 2, 22}, 200.42`}, {{2010, 2, 23}, 197.06`}, {{2010, 2, 24},
200.66`}, {{2010, 2, 25}, 202.`}, {{2010, 2, 26}, 204.62`}, {{2010, 3, 1},
208.99`}, {{2010, 3, 2}, 208.85`}, {{2010, 3, 3}, 209.33`}, {{2010, 3, 4},
210.71`}, {{2010, 3, 5}, 218.95`}, {{2010, 3, 8}, 219.08`}, {{2010, 3, 9},
223.02`}, {{2010, 3, 10}, 224.84`}, {{2010, 3, 11}, 225.5`}, {{2010, 3, 12},
226.6`}, {{2010, 3, 15}, 223.84`}, {{2010, 3, 16}, 224.45`}, {{2010, 3, 17},
224.12`}, {{2010, 3, 18}, 224.65`}, {{2010, 3, 19}, 222.25`}, {{2010, 3, 22},
224.75`}, {{2010, 3, 23}, 228.36`}, {{2010, 3, 24}, 229.37`}, {{2010, 3, 25},
226.65`}, {{2010, 3, 26}, 230.9`}, {{2010, 3, 29}, 232.39`}, {{2010, 3, 30},
235.85`}, {{2010, 3, 31}, 235.`}, {{2010, 4, 1}, 235.97`}, {{2010, 4, 5},
238.49`}, {{2010, 4, 6}, 239.54`}, {{2010, 4, 7}, 240.6`}, {{2010, 4, 8},
239.95`}, {{2010, 4, 9}, 241.79`}, {{2010, 4, 12}, 242.29`}, {{2010, 4, 13},
242.43`}, {{2010, 4, 14}, 245.69`}, {{2010, 4, 15}, 248.92`}, {{2010, 4, 16},
247.4`}, {{2010, 4, 19}, 247.07`}, {{2010, 4, 20}, 244.59`}, {{2010, 4, 21},
259.22`}, {{2010, 4, 22}, 266.47`}, {{2010, 4, 23}, 270.83`}, {{2010, 4, 26},
269.5`}, {{2010, 4, 27}, 26204}, {{2010, 4, 28}, 261.6`}, {{2010, 4, 29},
268.64`}, {{2010, 4, 30}, 261.09`}};
```

Get data from the WolframAlpha API:

```
wdata = WolframAlpha["AAPL from 2010/1/4 to 2010/4/30",
{{"DateRangeSpecified:Close:FinancialData", 1}, "TimeSeriesData"}];
```

Check the dimensionality:

```
Dimensions[#] & /@ {data, wdata}
```

Since the dimensions of the dataset are different, we use `TemporalData` to synchronize them for comparison:

```
td = TemporalData[{data, wdata}];
slices = td["SliceData", #]& /@ td["PathTimes", 1]
```

Then we can check if our data is consistent with WolframAlpha source data:

```
compare = (Round[#[[1]], 0.01] == Round[QuantityMagnitude[#[[2]]], 0.01])& /@
slices;
FreeQ[compare, False]
```

Locate where data is not consistent and remove it:

```
newdata = Delete[data, Position[compare, False]]
```

Data Synchronization and Caching

Synchronizing the time series is an essential task when simulating strategies on a portfolio of assets. To do this, we use Wolfram *Finance Platform* functions that manage calendars to create a unique timeline, taking into account holidays and business days. We then use this timeline to synchronize the time series used in the simulation:

```
cacheDates = DayRange[cacheFromDate, tillDate, "BusinessDay",
  CalendarType → CALENDARTYPE,
  HolidayCalendar → HOLIDAYCALENDAR,
  IncludeEndPoints → {True,True}];
```

The `TemporalData` function is another useful tool for managing time stamps. We simply put time-stamp value pairs in `TemporalData` for one or multiple datasets, and its many properties can be queried for use later on.

To avoid having to recalculate every time a simulation is run, all results are stored in binary files of synchronized data series. When the dataset is updated with new data, only the last part of the cache is updated, thus reducing the computational burden.

Trading Logic and Portfolio Construction

After data is cleaned and correctly formatted, the next step is to apply trading logic to construct a portfolio, develop trading strategies, and generate buy and sell signals.

After developing several strategies, they are mixed into a single portfolio. This is accomplished by writing money management rules balancing risk and return.

Creating trading strategies

The high-level language underlying Wolfram *Finance Platform* provides all the necessary mathematical functions to write trading strategies. Using the different programming paradigms offered by this language, we can write very compact code to generate buy and sell signals. The sample strategy below generates a matrix of random weights whose columns are assets in the portfolio and whose rows are dates.

Using this sample trading strategy, we can generate random positions (-100% or +100%) on the different assets in our investment universe:

```
Options[RandomStrategy] = {Inputs -> {}, MaxBarsBack -> maxBarsBack};

RandomStrategy[Universe_List, btDates_List, opts : OptionsPattern[]] :=
Module[{inputs, mBarsBack, positions},
  inputs = OptionValue[Inputs];
  mBarsBack = OptionValue[MaxBarsBack];
  positions = {};

  If[NumberQ[mBarsBack] && CheckInputs[inputs, Universe, btDates, mBarsBack],
    positions = EquiWeightLS[#] & /@ Array[RandomInteger[{-1, 1}] &,
      {Length[btDates], Length[Universe]}];
    ];
  Return[positions]
  ];
```

Mixing trading strategies

A money manager function generates a final matrix of weights by mixing the matrices generated by all the strategies. In the sample below we use an equiweighted scheme to mix the strategies into a final portfolio. Each strategy's weight is one divided by the number of strategies.

Building on the trading strategies, the money manager generates a portfolio resulting from an equiweighted mix of the previously mentioned strategies:

```
Options[EquiWeightLS] = {Inputs->{}, MaxBarsBack → 21};

EquiWeightLS[weights_List, opts: OptionsPattern[]] := Module[{expS, expL, newWS,
newWL, newW},
  expS=Plus@@Select[weights, # < 0&];
  expL=Plus@@Select[weights, # > 0&];

If[expS ≠ 0,
  newWS=1/expS //N;
  ,
  newWS=0.;
  ];
If[expL ≠ 0,
  newWL=1/expL //N;
  ,
  newWL=0.;
  ];
newW = If[# > 0, newWL, If[# < 0, newWS, 0.]]& /@ weights;

Return[newW]
  ];
```

Financial indicators as signals for strategies

Wolfram *Finance Platform* comes with a wide range of financial indicators that can be used as the basis of any number of trading strategies. There are 98 such indicators all together:

```
FinancialIndicator[All]
```

```
{AbsolutePriceOscillator, AccelerationBands,
 AccumulationDistributionLine, AccumulativeSwingIndex,
 Aroon, AroonOscillator, AverageDirectionalMovementIndex,
 AverageDirectionalMovementIndexRating, AverageTrueRange,
 BollingerBands, ChaikinMoneyFlow, ChaikinOscillator,
 ChaikinVolatility, ChandeMomentumOscillator, Close,
 CloseLocationValue, CloseLocationValueVolume, CommodityChannelIndex,
 CommoditySelectionIndex, DemandIndex, DetrendedPriceOscillator,
 DirectionalMovement, DoubleExponentialMovingAverage,
 DynamicMomentumIndex, EaseOfMovement, ExponentialMovingAverage,
 FastStochastic, ForceIndex, ForecastOscillator, FullStochastic,
 High, HighestHigh, Inertia, IntradayMomentumIndex, KeltnerChannels,
 KlingerOscillator, LinearRegressionIndicator, LinearRegressionSlope,
 LinearRegressionTrendlines, Low, LowestLow, MarketFacilitation,
 MassIndex, MedianPrice, MESAPhase, MESASineWave, Momentum,
 MoneyFlowIndex, MovingAverageConvergenceDivergence,
 MovingAverageEnvelopes, NegativeVolumeIndex, OnBalanceVolume,
 Open, ParabolicStopAndReversal, PercentagePriceOscillator,
 PercentageVolumeOscillator, Performance, PolarizedFractalEfficiency,
 PositiveVolumeIndex, PriceChannels, PriceVolumeTrend,
 ProjectionBands, ProjectionOscillator, QStick, RaffRegressionChannel,
 RandomWalkIndex, RangeIndicator, RateOfChange, RelativeMomentumIndex,
 RelativeStrengthIndex, RelativeVolatilityIndex, RSquared,
 SimpleMovingAverage, SlowStochastic, StandardDeviation,
 StandardDeviationChannels, StandardError, StandardErrorBands,
 StochasticMomentumIndex, SwingIndex, TimeSegmentedVolume,
 TimeSeriesForecast, TriangularMovingAverage,
 TripleExponentialMovingAverage, TRIX, TrueRange, TypicalPrice,
 UltimateOscillator, VariableMovingAverage, VerticalHorizontalFilter,
 VolatilitySystem, Volume, VolumeRateOfChange,
 WeightedClose, WeightedMovingAverage, WildersMovingAverage,
 WilliamsAccumulationDistribution, WilliamsPercentR}
```

Default parameter values for an indicator can be called:

```
FinancialIndicator["BollingerBands"]
```

```
FinancialIndicator[BollingerBands, 20, 2]
```

See the notes section for an interactive example illustrating the default parameter values used by any indicator and how they display for a wide range of securities.

Using interoperability to build more complex strategies

Wolfram *Finance Platform* interfaces with other programming languages like C++, Java, and R. For example, we could write trading rules that reuse some existing R code or existing toolboxes, as demonstrated in the document titled "Identifying Business Cycles Using *RLink* in Wolfram *Finance Platform*." The same could be achieved with Java programs. For example, we could access the machine-learning algorithms provided by the Weka toolbox.

Bringing It All Together

The above examples provide high-level structures for constructing trading strategies, but do not address how to incorporate such strategies into *Finance Platform* code for the purposes of back-testing the results. See the notes section for implementation of an example of a `BackTesting` function demonstrated below.

Simulating trading results using a trading simulator

Applying trading signals to historical asset prices produces an equity curve representing the evolution of a trading account balance.

We can write a generalized trading simulator function that runs the simulation and calculates trading results. A possible function signature could be:

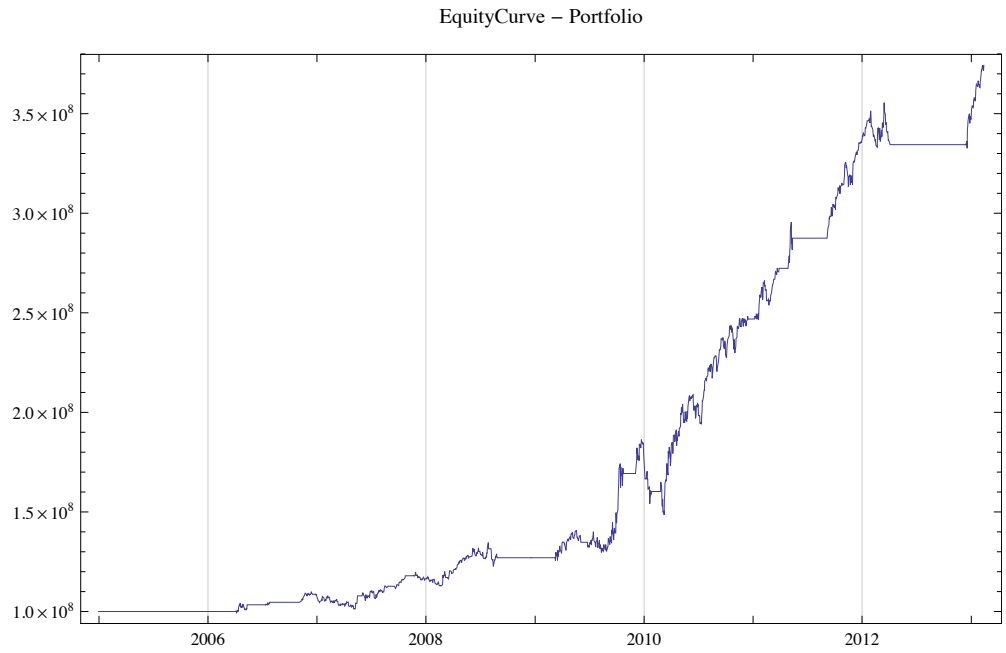
```
results = TradingSimulator[Universe, BTPeriod, Strategies, "close",
  FullStrategiesCalculations → True,
  InitialAccountSize → PortfolioSize,
  AccountCurrency → PortfolioCurrency,
  ExecutionPrices → "open",
  ExecutionLag → 1,
  QuantitiesAtExecutionPrice → True,
  AllowNetting → False,
  Fees → 0.00,
  Slippage → 0.,
  StrategiesPerformance → strategiesPerformance,
  FxRates → {},
  MaxBarsBack → maxBarsBack,
  DaysToRefresh → daysToRefresh,
  DataSources → DataSeries,
  DataTransformations → DataTransforms,
  MoneyManagementFunction → MoneyManager,
  MoneyManagementTimeWindow → 21,
  DataCachePath → dataCachePathName,
  InitialStrategyWeights → Table[1/Length[Strategies]//N, {Length[Strategies]}]
  ];
```

The following commands plot the equity curve of strategy:

```
{d,s,c,equityCurve} = ExtractSimulationResults[results,
  StrategiesToExtract → {"Portfolio"},
  ColumnsToExtract → {"Equity Curve"}];

equityCurve = Transpose[{d, equityCurve[[1, 1]]}];

DateListPlot[equityCurve, Joined → True, PlotLabel → StringJoin["EquityCurve - ",
s[[1]]]]
```



Example of a backtesting function

In the following examples, we take the example `BackTesting` function (implemented in the notes section) and introduce some data and a financial indicator or signal, forming the basis of a simple trading strategy. First, we import daily data for Apple Inc. for the first half of 2010.

Data

```
In[1]:= data = FinancialData["AAPL", "OHLCV", {{2010, 1, 1}, {2010, 6, 1}}];
```

We consider only the closing values:

```
In[2]:= close = data[[All, 2, 4]];
```

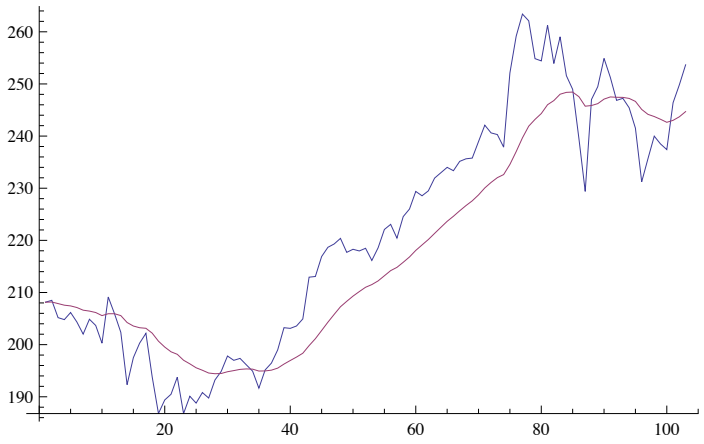
Signal

We can use any of Wolfram *Finance Platform*'s 100 financial indicators as a signal to create the strategy. These are listed at reference.wolfram.com/mathematica/guide/FinancialIndicators.html. We select the Exponential Moving Average (EMA) indicator with threshold 0.1:

```
In[3]:= sig = ExponentialMovingAverage[close, 0.1];
```

We plot the closing prices and the EMA indicator and observe that opportunities for trading occur where the two curves intersect:

```
ListLinePlot[{close, sig}]
```



Modeling and strategy

Here we consider a very simple strategy to buy when the closing price surpasses EMA, indicating that the current price has become greater than the long-run average:

```
tmp = Sign[close - sig]
```

```
{0, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1}
```

```
buypos = Position[Partition[tmp, 2, 1], {-1, 1}]
```

```
{{10}, {28}, {35}, {87}, {100}}
```

Then our strategy is to sell when the closing price drops below the EMA, indicating that the current price has become less than the long-run average:

```
sellpos = Position[Partition[tmp, 2, 1], {1, -1}]
```

```
{{2}, {11}, {33}, {85}, {91}}
```

We now line up the buy positions (1) and sell positions (-1) to correspond to each of the days in the trading period, and this becomes the strategy we test:

```
strategy = PadRight[
  Normal[SparseArray[Join[Thread[buypos → 1], Thread[sellpos → -1]]]],
  Length[tmp], 0]
```

```
{0, -1, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 1, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0}
```

Simulation

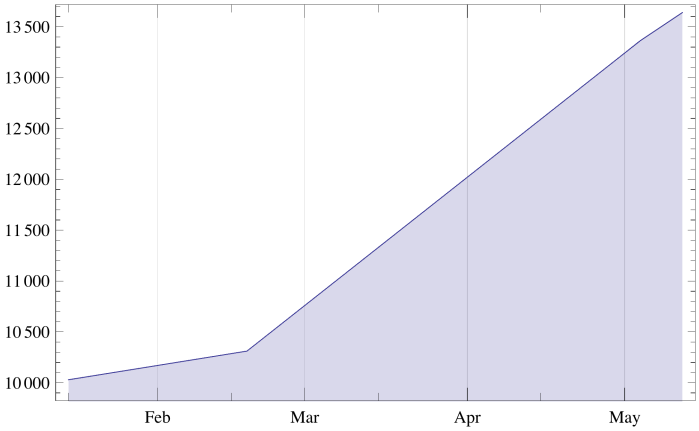
Finally, we can take the data and the strategy as defined above and indicate which of the two output options we want: either the raw data of buy and sell days and the prices at which the trade takes place, or the full report output giving the days, prices, equity plot, gross profit and loss, net profit, and current position:

```
BackTesting[data, strategy, Method → "RawData"]
```

```
{{{{2010, 1, 14}, 204.341}, {{2010, 1, 15}, 205.143}},
 {{{2010, 2, 10}, 190.509}, {{2010, 2, 18}, 196.096}},
 {{{2010, 2, 22}, 196.787}, {{2010, 5, 4}, 255.674}},
 {{{2010, 5, 6}, 246.862}, {{2010, 5, 12}, 252.128}}}
```

```
BackTesting[data, strategy, Method → "Report"]
```

Equity plot:



number of trades: 8

cost per trade: 4.5

total transaction costs: 36.

largest winning trade: 3062.13

largest losing trade: 0

gross profits: 3639.5

gross loss: 0

net profits: 3635.

current position:

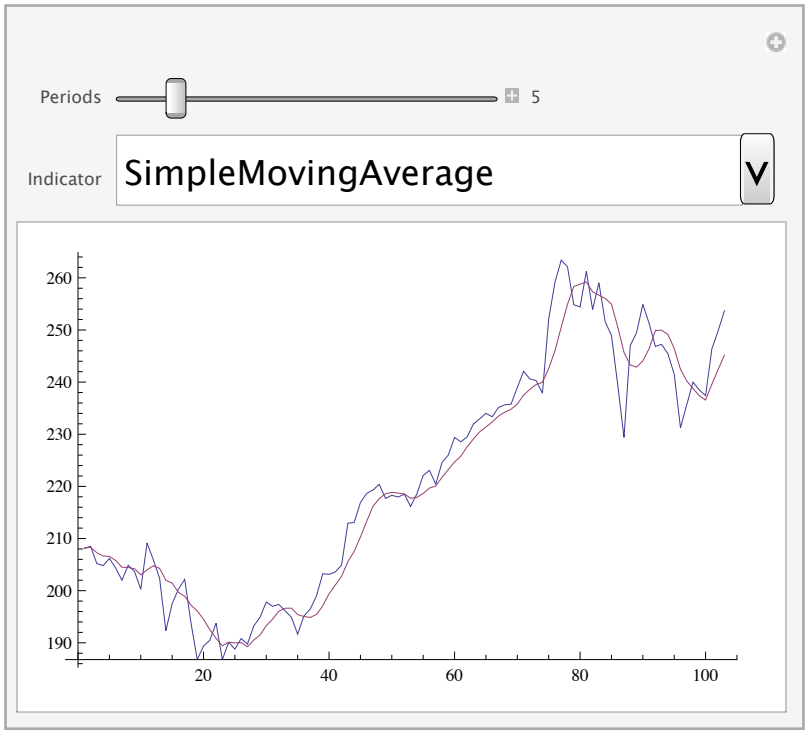
BuyDate	BuyPrice:
{2010, 5, 25}	232.781

BuyDate	BuyPrice	SellDate	SellPrice
{2010, 1, 14}	204.341	{2010, 1, 15}	205.143
{2010, 2, 10}	190.509	{2010, 2, 18}	196.096
{2010, 2, 22}	196.787	{2010, 5, 4}	255.674
{2010, 5, 6}	246.862	{2010, 5, 12}	252.128

Dynamic simulation

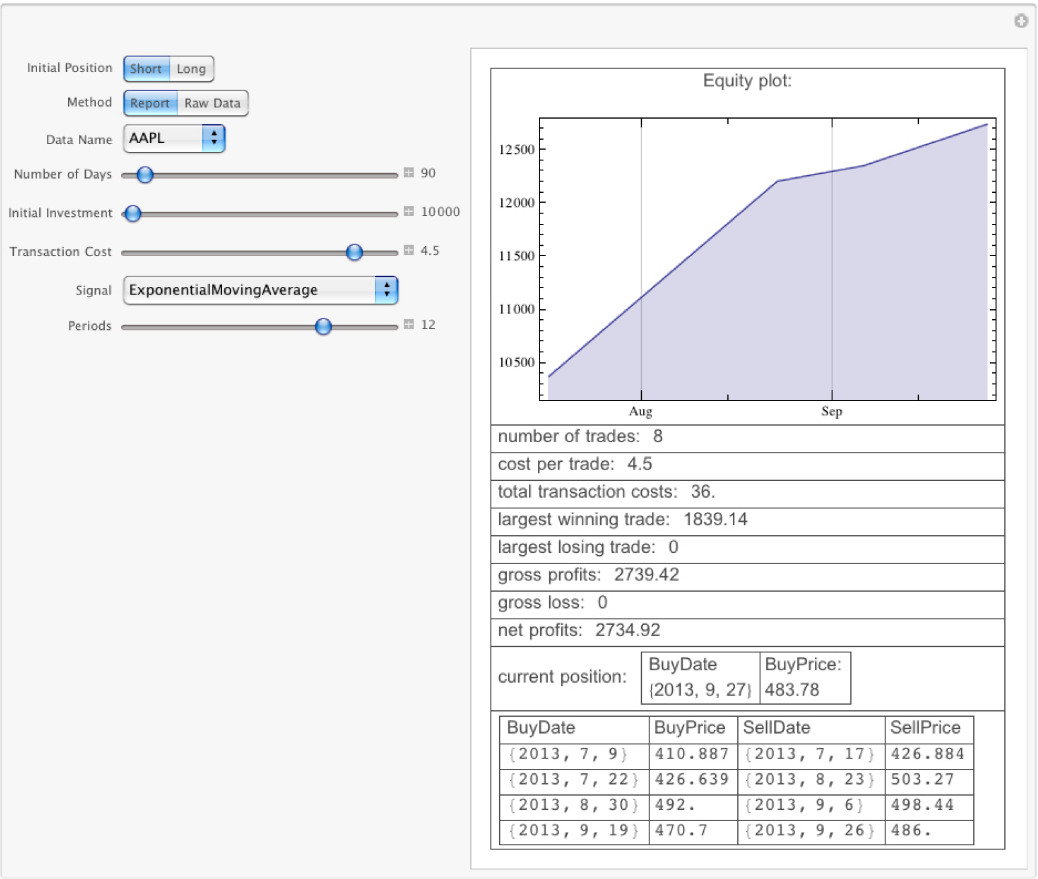
We can generalize this approach to treat many of the variables in the above example as parameters in a `Manipulate` simulation. Instead of choosing `ExponentialMovingAverage` as the signal to be used in the strategy, we can use the information obtained from the section "Financial Indicators as signals for strategies" for strategies and choose from a range of averaging financial indicators. Each averages over a number of periods in a different fashion. For instance, the indicator "`ExponentialMovingAverage`" has a smoothing constant of $\frac{2}{n+1}$ for n periods. In the following, we can choose from a wide range of these indicators and dynamically visualize how such indicators compare with the closing prices to create possible trading opportunities:

```
With[{indicators = {"SimpleMovingAverage",
    "ExponentialMovingAverage", "DoubleExponentialMovingAverage",
    "TripleExponentialMovingAverage", "WeightedMovingAverage",
    "TriangularMovingAverage", "VariableMovingAverage",
    "WildersMovingAverage", "TimeSeriesForecast"}},
 Manipulate[ListLinePlot[{close,
    Join[Take[close, Switch[indicator, "VariableMovingAverage",
       1, "WildersMovingAverage", periods - 1, _, 0]],
     FinancialIndicator[indicator, periods][data]]}],
  {{periods, 5, "Periods"}, 3, 20, 1, Appearance → "Labeled"},
  {{indicator, "SimpleMovingAverage", "Indicator"}, indicators,
   ControlType → PopupMenu}, SaveDefinitions → True]]
```



We now put all of the above code together to create an interactive report on choosing different trading strategies for different stocks and indices over different time periods:

```
Module[{indicators = {"SimpleMovingAverage",
    "ExponentialMovingAverage", "DoubleExponentialMovingAverage",
    "TripleExponentialMovingAverage",
    "WeightedMovingAverage", "TriangularMovingAverage",
    "VariableMovingAverage", "WildersMovingAverage"},
  data, close, sig, tmp, buypos, sellpos, strategy},
 Manipulate[data = FinancialData[dataName, "OHLCV",
    DatePlus[-NumbOfDays]]; close = data[[All, 2, 4]];
  sig = Join[Take[close, Switch[indicator, "VariableMovingAverage",
      1, "WildersMovingAverage", periods - 1, _, 0]],
    FinancialIndicator[indicator, periods][data]];
tmp = Sign[close - sig]; buypos = Position[Partition[tmp, 2, 1], {-1, 1}];
sellpos = Position[Partition[tmp, 2, 1], {1, -1}];
strategy = PadRight[Normal[SparseArray[
      Join[Thread[buypos → 1], Thread[sellpos → -1]]]], Length[tmp], 0];
  Quiet[BackTesting[data, strategy, "Method" → method,
    "InitialPosition" → initPosn, "TransactionCost" → transCost,
    "InitialInvestment" → initInvestment]],
  {{initPosn, "Short", "Initial Position"},
   {"Short", "Long"}, ControlType → Setter},
  {{method, "Report", "Method"}, {"Report", "Raw Data"},
   ControlType → Setter}, {{dataName, "AAPL", "Data Name"},
   Join[indices, stocksUS, stocksForeign], ControlType → PopupMenu},
  {{NumbOfDays, 90, "Number of Days"}, 60,
   700, 1, Appearance → "Labeled"},
  {{initInvestment, 10 000, "Initial Investment"},
   10 000, 10^6, 1000, Appearance → "Labeled"},
  {{transCost, 4.5, "Transaction Cost"}, 1, 5,
   0.5, Appearance → "Labeled"},
  {{indicator, "ExponentialMovingAverage", "Signal"},
   indicators, ControlType → PopupMenu},
  {{periods, 12, "Periods"}, 3, 15, 1, Appearance → "Labeled"},
  ControlPlacement → Left, SaveDefinitions → True]]
```



Evaluating and analyzing simulation performance

The final step of the research process is evaluating the performance of our simulation. This is accomplished by studying different statistics and assessing the strategy's risk and return profile. The full power of *Finance Platform*, with its symbolic and report generation capabilities, simplifies this process.

Automatically generating performance reports

Using Wolfram *Finance Platform*'s built-in report generation functionality, you can automatically create reports containing interactive charts and dynamic computations. Data for the report can come from a variety of sources, such as the result of a computation, a database query, *Finance Platform*'s integrated computation data source, or a structured dataset. The appearance and style of the report is set by a customizable template document with easy-to-configure variables.

Automatically generated reports can include built-in interfaces for interactively manipulating data and can be saved as Computable Document Format™ (CDF) and shared with anyone using the free Wolfram *CDF Player*™. Other supported document formats include PDF, HTML, and TeX.

Watch the "Generating Finance Reports" overview video for more details at www.wolfram.com/broadcast/video.php?channel=249&video=1502. Report generation examples are available in the Wolfram *Finance Platform* Infokit under **Interactive Reports and Apps**.

Summary

The power of backtesting is dependent upon the reliability and uniformity of data, the flexibility of computational tools, and the ease of reporting and analysis. We have shown the agility of Wolfram *Finance Platform* in all three areas: first, in providing several methods for obtaining, processing, and synchronizing data; second, in powerful portfolio construction through interoperability and high-level language; and third, in generating simulations and reports that inform the decision-making process for developing trading strategies and enabling continued analysis.

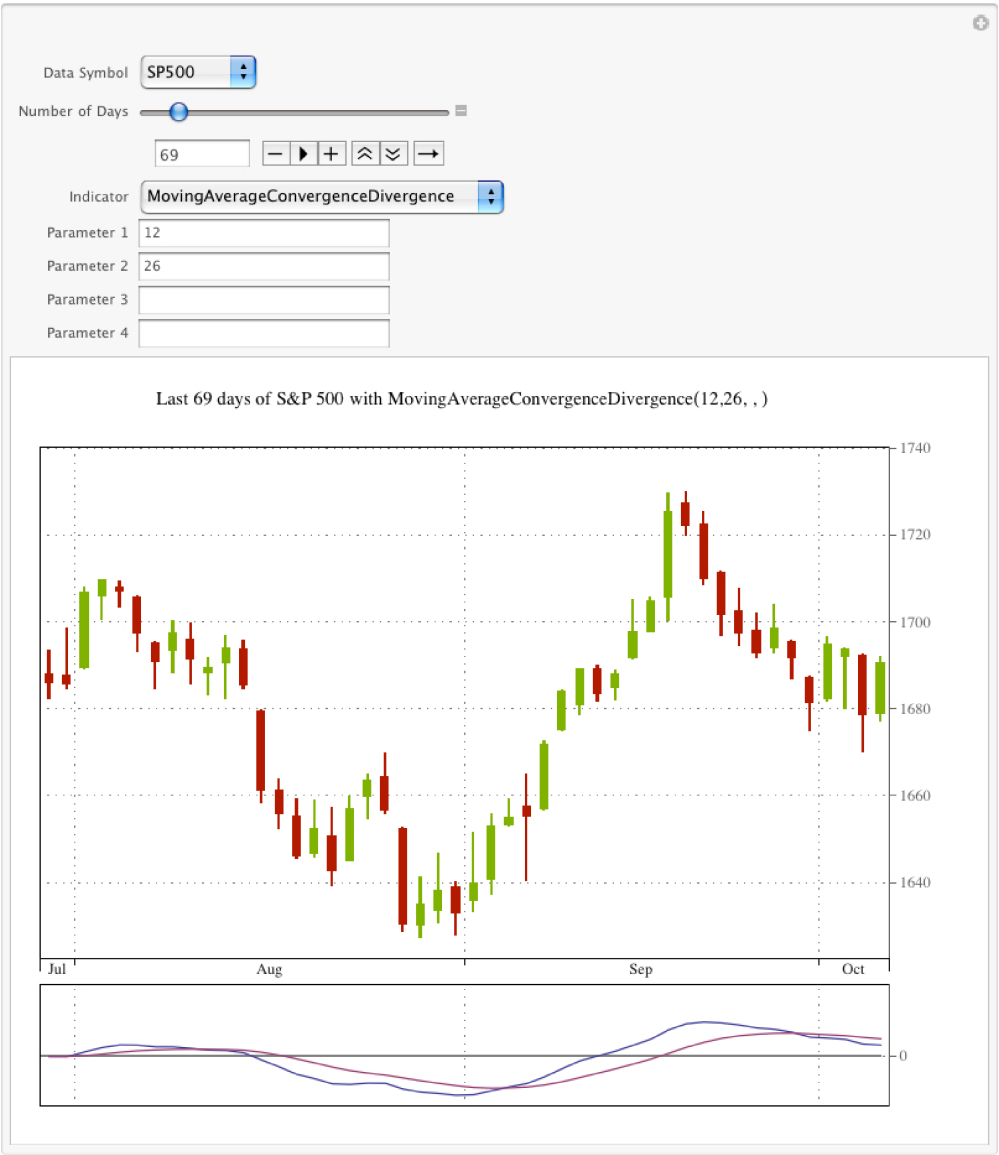
When every step is streamlined, seamless, and efficient, users such as portfolio managers can focus more of their time on doing real analysis. Wolfram *Finance Platform* is a completely integrated solution that achieves this goal.

Notes

Interactive example using FinancialIndicator

As mentioned in our discussion of financial indicators, we can create an interactive example to get the default parameter values used by any indicator and display results for a wide range of securities:

```
Module[{indic, par1, par2, par3, par4, name},
 Manipulate[indic = FinancialIndicator[indicator];
  par1 = If[Length[indic] > 1, indic[[2]], Null];
  par2 = If[Length[indic] > 2, indic[[3]], Null];
  par3 = If[Length[indic] > 3, indic[[4]], Null];
  par4 = If[Length[indic] > 4, indic[[5]], Null];
  name = If[MatchQ[FinancialData[DataSymbol, "Name"], Missing[
       "NotAvailable"]], DataSymbol, FinancialData[DataSymbol, "Name"]];
  Quiet[TradingChart[{DataSymbol, DatePlus[-NumbOfDays]},
    {FinancialIndicator[indicator, param1, param2, param3, param4]},
    PlotLabel → Row[{"Last ", NumbOfDays, " days of ", name,
        " with ", indicator, "(", param1, ",", param2, ",",
       param3, ",", param4, ")"}], ImageSize → {600, 480}]],
  {{DataSymbol, "SP500", "Data Symbol"}, Join[indices,
    stocksUS, stocksForeign], ControlType → PopupMenu},
  {{NumbOfDays, 60, "Number of Days"}, 2, 720, 1, Appearance → "Open"},
  {{indicator, "MovingAverageConvergenceDivergence", "Indicator"},
   FinancialIndicator[All], ControlType → PopupMenu},
  {{param1, par1, "Parameter 1"}, InputField},
  {{param2, par2, "Parameter 2"}, InputField},
  {{param3, par3, "Parameter 3"}, InputField},
  {{param4, par4, "Parameter 4"}, InputField}, SaveDefinitions → True,
  SynchronousUpdating → False, ContinuousAction → False]]
```



Implementing an example BackTesting function

Here we present some straightforward code implementing the means for any strategy to be input as an argument into a function, allowing us to test the strategy with any given dataset. The code consists of one general function, `BackTesting`, and its options and calls another function, `Simulation`, with the same arguments that apply the strategy to the given dataset. It then calls yet another function, `BackTestingReport`, which generates the final report consisting of the buy and sell data points and the corresponding equity plot. Examples of its application are discussed in the section Bringing It All Together:

In[4]:=
```
Options[BackTesting] =
  {"TransactionCost" → 4.5, "InitialInvestment" → 10 000};

BackTesting[d_, strategy_, o : OptionsPattern[]] :=
 Module[{open, high, low, close, volume, dates, flagTD, bkfun},
  dates = d[[All, 1]];
  {open, high, low, close, volume} = Transpose[d[[All, 2]]];
  Simulation[d, strategy, o]]
```

In[6]:=
```
Options[Simulation] =
  {"Method" → "Report", "InitialPosition" → "Short"};

Simulation[rdata_, sdata_, o : OptionsPattern[]] :=
 Module[{buy, sell, pos, profits, net, method, buydate, selldate,
   list, commission, principal, security, power, equity, dates,
   open, high, low, close, volume, max, min, win, los, cur, length},
  {method, pos, commission, principal} = Quiet@
    OptionValue[{Simulation, BackTesting}, Flatten[{o}], {"Method",
      "InitialPosition", "TransactionCost", "InitialInvestment"}];
  {dates, open, high, low, close, volume} =
   {rdata[[All, 1]], rdata[[All, 2, 1]], rdata[[All, 2, 2]],
    rdata[[All, 2, 3]], rdata[[All, 2, 4]], rdata[[All, 2, 5]]};
  (*initial*)
  buy = 0;
  sell = 0;
  profits = 0;
  list = {};
  net = {};
  security = 0;
  power = principal;
  equity = power;
  max = 0;
  min = 0;
  win = 0;
  los = 0;
  (*testing*)
  length = Min[Length /@ {rdata, sdata}];
  MapIndexed[Which[pos === "Short",
     If[# === 1, buy = open[[First[#2]]];
      buydate = dates[[First[#2]]];
      security = If[power > 0, Floor[(power - commission) / buy], 0];
      power = Mod[(power - commission), buy];
      equity = security * buy + power;
      pos = "Long"],

     pos === "Long",

     If[# === -1,
      sell = open[[First[#2]]];
      selldate = dates[[First[#2]]];
      equity = security * sell + power - commission;
      max = Max[max, security * (sell - buy)];
      min = Min[min, security * (sell - buy)];
      If[sell - buy > 0, win = win + security * (sell - buy) - 2 * commission,
       los = los + security * (sell - buy) - 2 * commission];
      power = equity;
      security = 0;
      pos = "Short";
      profits = sell - buy;
      net = Append[net, equity];
      list = Append[list, {{buydate, buy}, {selldate, sell}}];],
     True, Null] &, Rest[sdata]];
  cur = Switch[pos, "Long",
    {"long", buydate, buy}, "Short", {"short", selldate, sell}];
  (*result type*)
  Switch[method,
   Automatic | "RawData", list,
```

```
  "Report", BackTestingReport[list, commission,
   principal, equity, max, min, win, los, net, cur]
 ]
]
```

In[8]:=
```
BackTestingReport[list_, commission_,
 principal_, equity_, max_, min_, win_, los_, net_, cur_] :=
Module[{buydate, selldate, buyprice, sellprice, table,
  not, cost, com, eq, lwt, llt, pos, neg, curr, plt},
 buydate = list[[All, 1, 1]];
 selldate = list[[All, 2, 1]];
 buyprice = list[[All, 1, 2]];
 sellprice = list[[All, 2, 2]];
 table = Grid[
   Transpose[{Join[{Style["BuyDate", FontFamily → "Arial"]}, buydate],
     Join[{Style["BuyPrice", FontFamily → "Arial"]}, buyprice],
     Join[{Style["SellDate", FontFamily → "Arial"]}, selldate],
     Join[{Style["SellPrice", FontFamily → "Arial"]}, sellprice]}],
   Alignment → {Left, ".", Left, "."}, Frame → All];
 not = Grid[{Join[{Style["number of trades:", FontFamily → "Arial"]},
     {Style[Length[list] * 2, FontFamily → "Arial"]}]}];
 cost = Grid[{Join[{Style["cost per trade:", FontFamily → "Arial"]},
     {Style[commission, FontFamily → "Arial"]}]}];
 com = Grid[{Join[{Style["total transaction costs:",
       FontFamily → "Arial"]},
     {Style[commission * Length[list] * 2, FontFamily → "Arial"]}]}];
 eq = Grid[{Join[{Style["net profits:", FontFamily → "Arial"]},
     {Style[equity - principal, FontFamily → "Arial"]}]}];
 lwt = Grid[{Join[{Style["largest winning trade:",
       FontFamily → "Arial"]}, {Style[max, FontFamily → "Arial"]}]}];
 llt = Grid[{Join[{Style["largest losing trade:",
       FontFamily → "Arial"]}, {Style[min, FontFamily → "Arial"]}]}];
 pos = Grid[{Join[{Style["gross profits:", FontFamily → "Arial"]},
     {Style[win, FontFamily → "Arial"]}]}];
 neg = Grid[{Join[{Style["gross loss:", FontFamily → "Arial"]},
     {Style[los, FontFamily → "Arial"]}]}];
 curr = Grid[{Join[{Style["current position:", FontFamily → "Arial"]},
     {If[cur[[1]] === "long",
       Grid[Transpose[{Join[{Style["BuyDate", FontFamily → "Arial"]},
            {Style[cur[[2]], FontFamily → "Arial"]}],
          Join[{Style["BuyPrice:", FontFamily → "Arial"]},
           {Style[cur[[3]], FontFamily → "Arial"]}]}],
        Alignment → {Left, ".", Left, "."}, Frame → {All, False}], Grid[
        Transpose[{Join[{Style["SellDate", FontFamily → "Arial"]},
            {Style[cur[[2]], FontFamily → "Arial"]}],
          Join[{Style["SellPrice:", FontFamily → "Arial"]},
           {Style[cur[[3]], FontFamily → "Arial"]}]}],
        Alignment → {Left, ".", Left, "."}, Frame → {All, False}]]}]}];
 If[Or @@ StringQ /@ selldate, selldate = Most[selldate]];
 plt = Grid[{{Style["Equity plot:", FontFamily → "Arial"]},
    {DateListPlot[Transpose[{selldate, net}], Joined → True,
      Filling → 0, ImageSize → Medium, PlotRange → Full]}}];
 Column[{plt, not, cost, com, lwt, llt, pos, neg, eq, curr, table},
  Frame → All]]
```

```
In[9]:= stocksUS = {"AAPL", "GOOG", "CTSH", "GRMN", "HANS", "AMZN", "DLB",
        "RIMM", "COH", "INFY", "DECK", "BIDU", "SYK", "NVDA", "EBIX",
        "DV", "FCX", "AFAM", "GILD", "FSLR", "MIDD", "ORCL", "QSII",
        "MSFT", "NFLX", "WDC", "LULU", "PG", "ARO", "AOB", "GE"};

In[10]:= stocksForeign = {"F:UTDI", "F:HMSB", "F:S92", "TSX:RIM",
        "F:SWV", "F:NDA", "F:SZG", "F:SDF", "F:MAN", "F:DBK", "F:DAI",
        "ASX:WOW", "F:QCE", "F:VIA", "F:APC", "F:BMW", "F:ALV",
        "F:SAP", "F:SIE", "F:WDI", "F:CBK", "F:IFX", "ASX:BHP", "F:DB1",
        "F:BAS", "F:LHA", "F:ADS", "F:EOAN", "F:MUV2", "TSX:POT"};

In[11]:= indices = {"^AEX", "^AORD", "^ATX", "^SSEC", "^HSI", "^N225",
        "^BFX", "^BSESN", "^FCHI", "^BXD", "^BXM", "^VIX", "^VXN",
        "FTSE100", "FTSE250", "FTSE350", "SP100", "SP400", "SP500",
        "SP600", "^SSMI", "^OSEAX", "^GDAXI", "^OMXSPI", "^KLSE",
        "^NZ50", "^KS11", "^JKSE", "^MXX", "^MERV", "^BVSP"};
```

Pricing and Licensing Information

Wolfram *Finance Platform* includes all of the technologies and support services that you need to be productive.

Purchase levels scale from small teams to entire enterprise-wide deployment. Contact our finance industry experts to discuss a quote.

Visit us online for more information:
www.wolfram.com/finance-platform/contact-us

Recommended Next Steps

Watch videos about Wolfram *Finance Platform*
www.wolfram.com/broadcast/video.php?channel=249

Request a free trial or schedule a technical demo
www.wolfram.com/finance-platform/contact-us

Learn more about Wolfram *Finance Platform*

US and Canada
1-800-WOLFRAM (965-3726)
info@wolfram.com

Europe
+44-(0)1993-883400
info@wolfram.co.uk

Outside US and Canada (except Europe and Asia)
+1-217-398-0700
info@wolfram.com

Asia
+81-(0)3-3518-2880
info@wolfram.co.jp