Contents

1 Optimize the performance of Geniustrader

This module can be used to improve the performance of any program using the GeniusTrader modules (e.g. the scripts graphic.pl, backtest.pl, ...).

To use the module you have to make sure that you have write permission for the directory the variable $newpath points to.

Next you have to insert the following two lines **above** any other use- or require-statement that includes a GT-module:

```
use OptimizeGT;
use lib $OptimizeGT::newpath;
```

Now you can use the script the normal way. When you first call the script this module will need a little bit of time to do some optimization stuff but therefore the next run will be much faster.

If you normally don't change your version of GT but simply use it, then you can set the variable $PERIODIC_UPDATE to a higher value. If this variable is non-zero the script is looking only every $PERIODIC_UPDATE seconds for an update of the GT-modules. In this case it might be that changes on a module are not detected and a old version is use instead!

Warning

This module is using regular expressions to modify the GT modules source code on disk. This might lead to a problem if the one of the function names that should be replaced is osed in a different context.

If your program produces an error you don't understand please you should try to comment out the two lines mentioned above.

2 analyze_backtest.pl

SYNOPSIS

```
./analyze_backtest.pl
```

OPTIONS

–set=SETNAME

Restricts the analysis to a specific set. A set is simply an identifier that you put on data when you add it to the "backtests" directory (refer to your options file for the location of this directory) when using backtest_many.pl. Using the –set option you can differentiate between the different backtest results in your directory.

–template=<template file>

Output is generated using the indicated HTML::Mason component. For Example, –template="analyze_backtest.mpl" The template directory is defined as Template::directory in the options file. Each template can be predefined by including it into the options file For example, Template::analyze_backtest analyze_backtest.mpl

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

DESCRIPTION

This tool runs an analysis against existing backtest spool files. The location of the spool files is defined as BackTest::Directory in the options file.

3 Analyzer - Shell

The command help shows you a comprehensive summary of the available commands:

anashell> help

By typing set without a parameter it will show you the current settings:

anashell> set Settings: expert => 1 code => 13000 system => Systems::Generic {S:Generic:CrossOverUp {I:SMA 20} {I:SMA 60}} {S:Generic:CrossOverDown {I:SMA 20} {I:SMA 60}} broker => InteractiveBrokers first => auto tf => OneTrade full => 0 last => auto mm [0] => Basic cs => OppositeSignal

Most of the settings are very easy to understand:

```
expert => 1
 If set to 1 every command that can't be interpreted by the internal
 parser is interpreted by perl.

code => 13000
 The code :)

system => Systems::Generic {S:Generic:CrossOverUp {I:SMA 20} {I:SMA 60}}
{S:Generic:CrossOverDown {I:SMA 20} {I:SMA 60}}
 The System to test

broker => InteractiveBrokers
 The broker

first => auto
last => auto
 Can be set to a date; auto results in the same settings as in
 backtest.pl

tf => OneTrade
mm [0] => Basic
cs => OppositeSignal
 Tradefilters, Money-Management and Closing-Strategies

full => 0
 Test the full history?
```

For our backtest we want to test the full history so we use:

anashell> set full 1

And we add one more closing-strategy by using the following command:

anashell> set +cs Stop::KeepRunUp 10

(set cs[1] Xxxxx would have changed the second array element) After this we start the backtest...

anashell> btest Tested ... ok in 104 seconds

...and view the result:

anashell> report report_summary.ash

By using the HTML::Mason-framework you can generate every report you want...

Now we can save the system to a directory

anashell> save TEST /tmp Saved TEST in /tmp...

So that we can load it the next time we start a session:

anashell> load TEST /tmp Loaded Portfolio TEST...

If we don't know the name of our system we can list all systems in the directory:

anashell> list /tmp ==> SY:Generic {S:Generic:CrossOverUp {I:SMA 20 {I:Prices CLOSE}} {I:SMA 60 {I:Prices CLOSE}}} {S:Generic:CrossOverDown {I:SMA 20 {I:Prices CLOSE}} {I:SMA 60 {I:Prices CLOSE}}}|TF:OneTrade 50|CS:OppositeSignal|CS:Stop:KeepRunUp 10 -> 13000

Now let's do some analysis on the backtest:

First I would like to know the costs for each trade

anashell> calc_array {A:OpenDate} {A:Costs} Number OpenDate[] Costs[] [0] 1993-05-13 20.76 [1] 1993-07-01 22.65 [...] [34] 2002-01-17 20.3

But we can also calculate the average cost of one trade:

anashell> calc {A:Avg {A:Costs}} 20.1568571428571

- you see it is the same as using an indicator...

Now let's do some graphics (make sure you have R (www.r-project.org) installed):

anashell> @gain = calc_array("{A:NetGain}") anashell> r_hist(\@gain)

You can also have a look at the distribution of the gains over time:

anashell> r_bar(\@gain)

Or we create a second array and see if there is a correlation between the Duration of a trade and its gain...

anashell> @duration = calc_array("{A:Duration}") r_corr(\@duration, \@gain)

Now let's leave the program and save the history for the next session:

anashell> bye Exiting Olf's Analyzer... Save settings? [Y/n]: Y

Requirements

```
This module needs Term::ReadLine to process the interactive commands.
```

4 ./backtest.pl [options] <code>

5 ./backtest.pl [options] <system_alias> <code>

6 ./backtest.pl [options] "<full_system_name>" <code>

Description

Backtest will run a backtest of a system on the indicated code.

You can either describe the system using options, give a full system name, or you can give a system alias. An alias is defined in the configuration file with entries of the form Aliases::Global::<alias_name> <full_system_name>.

The full system name consists of a set of properties, such as trade filters, close strategy, etc., together with their parameters, separated by vertical bars ("|"). Multiple properties of the same type can be defined, e.g., there could be a set of close strategies. For example, System:ADX 30 | TradeFilters:Trend 2 5 | MoneyManagement:Normal defines a system based on the "ADX" system, using a trend following trade filter "Trend", and the "Normal" money management.

The following abbreviations are supported: Systems = SY CloseStrategy = CS TradeFilters = TF MoneyManagement = MM OrderFactory = OF Signals = S Indicators = I Generic = G

Another example of a full system name is SY:TFS|CS:SY:TFS|CS:Stop:Fixed 4|MM:VAR.

Options

Backtest provide a set of options, so that you can use a combination of MoneyManagement, TradeFilters, OrderFactory an CloseStrategy modules.

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval over which to perform the backtest. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates over which to perform the backtest. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the analysis.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these

indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)
2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–template="backtest.mpl"

Output is generated using the indicated HTML::Mason component. For Example, –template="backtest.mpl" The template directory is defined as Template::directory in the options file. Each template can be predefined by including it into the options file For example, Template::backtest backtest.mpl

–html

Output is generated in html

–graph="filename.png"

Generate a graph of your portfolio value over the time of the backtest and display it in the generated html.

–display-trades

Display the trades with little symbols on the graph. This works well if trades last long enough otherwise your graph will be overwhelmed with unsignificant symbols.

–store="portfolio.xml"

Store the resulting portfolio in the indicated file.

–broker="NoCosts"

Calculate commissions and annual account charge, if applicable, using GT::Brokers::<broker_name> as broker.

–system="<system_name>"

use the GT::Systems::<system_name> as the source of buy/sell orders.

–money-management="<money_management_name>"

use the GT::MoneyManagement::<money_management_name> as money management system.

–trade-filter="<filter_name>"

use the GT::TradeFilters::<filter_name> as a trade filter.

–order-factory="<order_factory_name>"

use GT::OrderFactory::<order_factory_name> as an order factory.

–close-strategy="<close_strategy_name>"

use GT::CloseStrategy::<close_strategy_name> as a close strategy.

–set=SETNAME

Stores the backtest results in the "backtests" directory (refer to your options file for the location of this directory) using the set name SETNAME. Use the –set option of analyze_backtest.pl to differentiate between the different backtest results in your directory.

–output-directory=DIRNAME

Override the "backtests" directory in the options file.

–verbose

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

Examples

./backtest.pl TFS 13000

./backtest.pl –full TFS 13000

./backtest.pl –close-strategy="Systems::TFS" –close-strategy="Stop::Fixed 6" –money-management="VAR" –money-management="OrderSizeLimit" –system="TFS" –broker="SelfTrade Intégral" 13000

./backtest.pl –broker="SelfTrade IntĄégral" "SY:TFS|CS:SY:TFS|CS:Stop:Fixed 6|MM:VAR|MM:OrderSizeLimit" 13000

7 ./backtest_many.pl [options] <market file> <system file>

Description

Backtest_many will test all system listed in a system file on all the values listed in the market file.

The <system file> contains one line per defined system, where each system is defined by its full system name or by an alias. An alias is defined in the configuration file with entries of the form Aliases::Global::<alias_name> <full_system_name>.

The full system name consists of a set of properties, such as trade filters, close strategy, etc., together with their parameters, separated by vertical bars ("|"). Multiple properties of the same type can be defined, e.g., there could be a set of close strategies. For example, System:ADX 30 | TradeFilters:Trend 2 5 | MoneyManagement:Normal defines a system based on the "ADX" system, using a trend following trade filter "Trend", and the "Normal" money management.

The following abbreviations are supported: Systems = SY CloseStrategy = CS TradeFilters = TF MoneyManagement = MM OrderFactory = OF Signals = S Indicators = I Generic = G

Another example of a full system name is SY:TFS|CS:SY:TFS|CS:Stop:Fixed 4|MM:VAR.

Options

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval to consider for analysis. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for analysis. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the analysis.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)

2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–broker="NoCosts"

Calculate commissions and annual account charge, if applicable, using GT::Brokers::<broker_name> as broker.

–nbprocess=2

If you want to start two (or more) backtests in parallel (useful for machines with several CPUs for example).

–set=SETNAME

Stores the backtest results in the "backtests" directory (refer to your options file for the location of this directory) using the set name SETNAME. Use the –set option of analyze_backtest.pl to differentiate between the different backtest results in your directory.

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

Examples

./backtest_many.pl ../Listes/fr/CAC40 ../BackTest/HCB.txt –output-dir=../BackTest/ –set=HCB –full

Example of system description

SY:TFS 50 7|CS:SY:TFS 50|CS:Stop:Fixed 6|MM:VAR 10 2|MM:PositionSizeLimit 100

8 ./backtest_multi.pl [options] <market file> <system file>

Description

Backtest_many will test all system listed in a system file on all the values listed in the market file.

The <system file> contains one line per defined system, where each system is defined by its full system name or by an alias. An alias is defined in the configuration file with entries of the form Aliases::Global::<alias_name> <full_system_name>.

The full system name consists of a set of properties, such as trade filters, close strategy, etc., together with their parameters, separated by vertical bars ("|"). Multiple properties of the same type can be defined, e.g., there could be a set of close strategies. For example, System:ADX 30 | TradeFilters:Trend 2 5 | MoneyManagement:Normal defines a system based on the "ADX" system, using a trend following trade filter "Trend", and the "Normal" money management.

The following abbreviations are supported: Systems = SY CloseStrategy = CS TradeFilters = TF MoneyManagement = MM OrderFactory = OF Signals = S Indicators = I Generic = G

Another example of a full system name is SY:TFS|CS:SY:TFS|CS:Stop:Fixed 4|MM:VAR.

Options

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval to consider for analysis. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for analysis. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the analysis.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)

2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–broker="NoCosts"

Calculate commissions and annual account charge, if applicable, using GT::Brokers::<broker_name> as broker.

–set=SETNAME

Stores the backtest results in the "backtests" directory (refer to your options file for the location of this directory) using the set name SETNAME. Use the –set option of analyze_backtest.pl to differentiate between the different backtest results in your directory.

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

Examples

./backtest_many.pl ../Listes/fr/CAC40 ../BackTest/HCB.txt –output-dir=../BackTest/ –set=HCB –full

Example of system description

SY:TFS 50 7|CS:SY:TFS 50|CS:Stop:Fixed 6|MM:VAR 10 2|MM:PositionSizeLimit 100

9 ./display_indicator.pl [options] <indicatorname> <code> [args...]

Description

Computes the indicator <indicatorname> on market <code> over the selected interval.

Options

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval to consider for analysis. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for analysis. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the analysis. Default is 200.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)
2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–last-record

Display results for the last period only. Overrides any other options given to determine the interval.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–tight

Displays indicator values in concise tabular format.

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

Examples

./display_indicator.pl I:SMA IBM 100
./display_indicator.pl –full I:RSI 13000
args (if any) are passed to the new call that will create the indicator.
./display_indicator.pl –nb 10 I:EMA IBM 120

10 ./display_signal.pl [options] <signalname> <code> [args...]

Description

Computes the value of signal <signalname> for market <code> over the selected interval.

Options

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval to consider for analysis. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for analysis. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the analysis. Default is 200.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)
2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–last-record

Display results for the last period only. Overrides any other options given to determine the interval.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–change (or -c)

show on output only those dates that signal changed

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

Arguments

<signalname>

The name of the signal you want to display. This can be any module under GT/Signals. For instance, S::Generic::CrossOverUp.

<code>

The symbol for which you wish to display the signal. Use whatever symbols are available in your database.

[args...]

Args are passed to the new call that will create the signal. args is a string that specifies the signal in gt terms, spaces and other chars that the shell interprets will need to be quoted in some way.

Examples

./display_signal.pl S:Prices:GapUp IBM

Test for the GapUp signal in symbol IBM. By default, use daily data, and display the last available 200 periods.

./display_signal.pl –full S:Generic:CrossOverUp EURUSD {I:EMA 50} {I:EMA 200}

Test for the EMA50 crossing over up EMA200. Do the test over the full available history data.

11 ./display_system.pl [options] <systemname> <code> [args...]

Description

Displays the signals generated for the system <systemname> for market <code> over the selected interval. A system is comprised of a long and a short signal which are each displayed when they trigger (printing 'long' and 'short', respectively).

Options

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval to consider for analysis. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for analysis. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the analysis.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)
2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–last-record

Display results for the last period only. Overrides any other options given to determine the interval.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, `-option=DB::Text::format=0` sets the format used to parse markets via the DB::Text module to 0.

Arguments

<systemname>

The name of the system you want to test. This can be any module under GT/Systems.

<code>

The symbol for which you wish to display the signal. Use whatever symbols are available in your database.

[args...]

Args are passed to the new call that will create the system. args is a string that specifies the signal in gt terms, spaces and other chars that the shell interprets will need to be quoted in some way.

Examples

```
./display_system.pl SY:G IBM {S:G:CrossOverUp {I:Prices CLOSE} {I:EMA}} {S:G:CrossOverD
```

A long signal is generated when Prices cross up over the EMA, while a short signal is generated when Prices cross down over the EMA.

12 ./graphic.pl [options | additional graphical elements] <code>

Synopsis

./graphic.pl [–timeframe=timeframe] [–nb-item=100] \ [–start=2005-06-01] [–end=2006-01-01] \ [–type=candle|candlevol|candlevolplace|barchart|line|none] \ [–volume] \ [–volume-height=150] [–title="Daily Chart"] \ [–width=200] [–height=230] [–logarithmic] \ [additional graphical elements] \ [–file=conf] [–driver={GD|ImageMagick}] \ [–out=imagefile] \ <code>

Use "graphic.pl –man" to list all available options

Description

graphic.pl can generate charts in png format, including indicators and system signals, either as overlays of the original price data, or in different regions of the chart.

Various options are available to control color, size and other graphic properties.

Additional Graphical Elements

By default, only the price will be plotted, however, you can add other indicators, either as overlays or in different zones of the chart.

To plot overlays, simply add them to the graphic. For instance:

–add="Curve(Indicators::EMA 50, blue)" –add="Curve(Indicators::EMA 200, red)"

To plot indicators in a different zone, first create a new-zone, then add the indicators:

```
--add="New-Zone(100)"
--add="Histogram(I:MACD/3, brown)"

--add="New-Zone(100)"
--add="Curve(I:MACD/1, red)"
--add="Curve(I:MACD/2, green)"

--add="New-Zone(100)"
--add="Set-Scale(0,100)" --add="set-title(RSI,tiny)" \
--add="Curve(Indicators::RSI/3)"
```

Full details about the available methods you can use with the –add option follow.

Note that all objects that might affect the scale (such as, e.g., an indicator) must be added to a zone before the scale is changed.

New-Zone(height, [left, right, top, bottom])

This creates a new zone for displaying more indicators. It's created with the given height and the given border sizes.

Switch-Zone(zoneid)

This changes the current display zone. 0 is the main zone, 1 is the volume zone if it exists. 2 is the first indicator zone and so on. Usually you just need to it switch to the "Volume" zone because you start on the main zone and you automatically switch to any newly created zone.

Set-Scale(min,max,[logarithmic]) or Set-scale(auto,[logarithmic])

This defines the scale for the currently selected zone (by default the last zone created or the main zone if no zone has been created). If auto scale is used, the scale must be set after all objects that can affect the min or max values have been added.

Set-Special-Scale(min,max,[log]) or Set-Special-Scale(auto,[log])

The last created object will be displayed with its own scale (and not the default one of the zone). The scale may be given or it may be calculated to fit the full zone. If auto scale is used, the scale must be set after all objects that can affect the min or max values have been added.

Set-Axis(tick1,tick2,tick3...)

Define the ticks for the main axis of the current zone.

Set-Title{-left|-right|-top|-bottom}(title,font_size)

This adds a title to the currently selected zone. The title will be displayed in the given size (size can be tiny, small, medium, large and giant). If the title contains a %c, this is replaced by the <code>, if it contains %n, this is replaced by the long name of the <code>. See also ~/.gt/sharenames, which contains lines of the form <code>\t<long name> mapping a market to its long name.

Histogram(<datasource>, [color])

Curve(<datasource>, [color])

Marks(<datasource>, [color])

Mountain(<datasource>, [color])

MountainBand(<datasource1>, <datasource2>, [color])

This adds a new graphical object in the current zone. The datasource explains what data has to be displayed.

Text(text, x, y, [halign, valign, font_size, color, font_face])

This adds a block of text at the given coordinate (expressed in percent of the width/height of the zone).

halign can be one of "left", "center" or "right". valign can be one of "top", "bottom" or "center". font_size can be one of "tiny", "small", "medium", "large" or "giant". font_face can be one of "arial", "times" or "fixed".

BuySellArrows(Systems::...)

This adds buy and sell arrows in the main chart, based on systems signals.

VotingLine(Systems::..., [y])

Show buy and sell arrows in a Voting Line à la OmniTrader, based on a System Manager. You can indicate the y at which the line should be displayed.

Datasources

Sometimes you need to pass datasources to the graphical objects. The following are currently available.

Indicators::<indicatorname>

An indicator.

PortfolioEvaluation(<portfolio>)

This datasources returns the evaluation of any portfolio.

Other Objects

Some datasources may be parameterized by objects. The following are currently available.

BackTestPortfolio(<systemname>, [directory])

This returns a portfolio that has been saved for a backtest of the system "systemname". The given directory must be a spool of backtests.

Options

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval used to plot the graphics. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for the plot. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use to plot the graphics. Default is 120.

–full

Consider all available periods.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)
2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–type=candle|candlevol|candlevolplace|barchart|line|none

The type of graphic to plot. none causes the price not to be displayed, however, overlays can still be sketched in the graphic.

–volume

Should the volume be plotted (it is by default)? Use –no-volume if you want to omit it.

–volume-height=150

–title="Daily Chart"

–width=200

–height=230

–logarithmic

–driver=GD|ImageMagick

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

Configuration-File (–file=conf)

With this option, additional parameters are read from the configurationfile conf. Each line in this file corresponds to a command line parameter. Lines starting with # are ignored.

–out=imagefile

Send image output to specified file. If not specified, output goes to stdout.

Examples

./graphic.pl –add="Switch-Zone(0)" \ –add="Curve(Indicators::SMA 38, [0,0,255])" \ –add="Curve(Indicators::SMA 100, [0,255,0])" \ –add="Curve(Indicators::SMA 200, [255,0,0])" \ –title="Daily history of %c" \ 13000 > test.png

./graphic.pl –add="Curve(Indicators::EMA 5,[255,0,0])" \ –add="Curve(Indicators::EMA 20,[0,0,255])" \ –add="BuySellArrows(SY::Generic {S::Generic::CrossOverUp {I:EMA 5} {I:EMA 20}} {S::Generic::CrossOverDown {I:EMA 5} {I:EMA 20}})" \ 13000 > test.png

13 manage_portfolio.pl

SYNOPSIS

```
./manage_portfolio.pl <portfolio> create [<initial-sum>]
./manage_portfolio.pl <portfolio> bought <quantity> <share> \
  <price> [ <date> <source> ]
./manage_portfolio.pl <portfolio> sold <quantity> <share> \
  <price> [ <date> <source> ]
./manage_portfolio.pl <portfolio> stop <share> <price>
./manage_portfolio.pl <portfolio> set initial-sum <sum of money>
./manage_portfolio.pl <portfolio> set broker <broker>
./manage_portfolio.pl <portfolio> report { performance | positions \
 | historic | analysis }
./manage_portfolio.pl <portfolio> file <filename>
./manage_portfolio.pl <portfolio> db
```

where
<portfolio> is filename of portfolio to use. it can be a non-existent file, in which case it will be created

<quantity> <price> are numeric values for <share> which is the appropriate stock symbol or cusip or other identifier

<date> is optional, the date the transaction happened on. if not supplied the default value for 'today' will be supplied. preferred GT format for dates is 'YYYY-MM-DD'.

<source> is optional, a text string, it can be used to note the source of the stock transaction. typically an internal GT used field. for individual transactions <source> can be set via the --source='string' option.

<broker> name of broker module to use. see ../GT/Brokers/. there is no error checking. if the supplied broker module fails to exist the portfolio will be flawed.

OPTIONS

–marged

Only useful for "bought" and "sold" commands. It explains that the corresponding positions are marged, no personal money has been used for them, the money has been rented.

–source <source>

Useful to tag certain orders as the result of a particular strategy. All orders passed by following the advice of someone could be tagged with his name and later you'll be able to make stats on the performance you made with his advices.

–template=<template file>

Output is generated using the indicated HTML::Mason component. For example, when using "report historic" use

```
--template="manage_portfolio_historic.mpl"
```

when using "report positions" use

```
--template="manage_portfolio_positions.mpl"
```

The template directory is defined as Template::directory in the options file. Each template can be predefined by including it into the options file For example,

```
Template::manage_portfolio_positions manage_portfolio_positions.mpl
Template::manage_portfolio_historic manage_portfolio_historic.mpl
```

–timeframe { day | week | ... }

Tell how to parse the format of the date.

–noconfirm

Do not prompt for confirmation, just apply the request

–detailed

Add extra information into the output. On by default. Turn off by using –nodetailed

–backup

Make backup of <portfolio> if changes made and applied. Turn off by using –nobackup. backup portfolio filename will be <portfolio>.<yyyymmddhhmmss> where yyyymmddhhmmss is the date-time of the portfolio files' last modification date and time.

–since <date>

–until <date>

Those two options are used to restrict the result of a "report" command to a certain timeframe.

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

COMMANDS

create

creates <portfolio> and optionally set <initial-sum> available funds. <initial-sum> is unset if omitted

bought <quantity> <share> <price> [<date>]

sold <quantity> <share> <price> [<date>]

create a <portfolio> purchase (bought) or sale (sold) transaction

it doesn't appear to support long and short transactions but ...

stop <share> <price>

create a stop order for <share> at <price>

report { performance | positions | historic | analysis }

generate and output the specified report

set { initial-sum | broker } <value>

create and set the value of variables initial-sum or broker to <value>

file <filename>

Specifies the name of a file which contains a list of

bought <quantity> <share> <price> [<date> <source>]

and

sold <quantity> <share> <price> [<date> <source>]

commands, one per line. This allows you to submit multiple bought/sold commands in a single instance.

db

reads beancounter database portfolio table and creates <portfolio> from it.

negative stock quantities are considered sells, <source> is derived from the 'type' column. currency is ignored, as GT seems to do.

since the beancounter portfolio really doesn't have necessary functionality to manage closed positions it is probably best to manage sells in the GT portfolio using command line or command file features provided here. in addition, by not introducing negative stock quantities in the beancounter portfolio table you will also avoid tickling bugs and messing up the report formats with the unexpected negative quantity value.

DESCRIPTION

This tool lets you create a portfolio object on your disk and update it regularly, this can be used to create virtual portfolio to test how a strategy works in real time or to track your real portfolio and use GeniusTrader to make some analysis on it.

The first parameter is a filename of a portfolio. The name is supposed to be relative to the portfolio directory which is $HOME/.gt/portfolio in Unix but can be overriden with the configuration item GT::Portfolio::Directory. If it doesn't exist, it will try in the local directory.

BUGS (or maybe just rough edges (in my opinion))

ha! i fixed this next bit

really should be a usage mode, invoked with args -h* | -? | -: and since the program requires a command, any instantiation without a valid one.

Needs to do a better job of checking input values for bought/sold operations or needs to provide a way of completely removing a bad entry

might also be nice to provide a couple of output modes; say one to generate a file that can used as input for the ./manage_portfolio.pl <portfolio> file <filename> capability and one to generate a textualized version of the portfolio, if that makes sense.

14 scan.pl

Scan the market looking for signals

SYNOPSIS

./scan.pl [options] <market file> <date> <system file> [<system file> ...]

DESCRIPTION

scan.pl will scan all stocks listed in <market file> looking for the signals indicated in each <system file> performing the analysis on the specified <date>. A system file must contain one or more description of GT::Signals or description of GT::Systems. You may list multiple system files on the command line. In the absence of a file standard input will be read instead.

NOTE – if you omit a system file name scan.pl will happily wait forever attempting to read from stdin.

The list of securities (code and name) that meet the specified signals is output at the end and grouped by signal.

Output can be either text (default) or html.

<market file> format:

stock or index symbols one per line

<date>

the date to perform the analysis on. the date string can be in any format that Date::Manip (if installed) can parse or the defacto gt standard date format (YYYY-MM-DD HH:MM:SS) where time is optional

<system file> format:

One or more GT::Signals or GT::Systems descriptions each on a separate line. The descriptions have the form of a signal or system name, followed by its arguments.

Example: S:Generic:And {S:Generic:CrossOverUp {I:SMA 5} {I:SMA 20}} {S:Generic:Increase {I:ADX}}

Description files can be formatted using the symbol '\' as the line continuation symbol. This symbol must appear as the last character on the line before the trailing line terminator (in unix that's a '\n' character). No whitespace must appear between the \ and the newline.

Example: S:Generic:And \ {S:Generic:CrossOverUp {I:SMA 5} {I:SMA 20}} \ {S:Generic:Increase {I:ADX}}

Blank lines and lines that start with # are comments and ignored. Note if you comment out the first line of multi-line description, the entire is effectively commented out.

Example: # the following signal description is commented out #S:Generic:And {S:Generic:Above {I:Prices} {I:EMA 30}} \ {S:Generic:Above {I:Prices} {I:EMA 150}}

OPTIONS

–full, –start=<date>, –end=<date>, –nb-item=<nr>

Determines the time interval over which the scan is run. In detail:

–start=2001-1-10, –end=2002-11-17

The start and end dates considered for the scan. The date needs to be in the format configured in ~/.gt/options and must match the timeframe selected.

–nb-items=100

The number of periods to use in the scan.

–full

Runs the scan with the full history.

The periods considered are relative to the selected time frame (i.e., if timeframe is "day", these indicate a date; if timeframe is "week", these indicate a week; etc.). In GT format, use "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss" for days (the latter giving intraday data), "YYYY-WW" for weeks, "YYYY/MM" for months, and "YYYY" for years.

The interval of periods examined is determined as follows:

1. if present, use –start and –end (otherwise default to last price)
2. use –nb-item (from first or last, whichever has been determined), if present
3. if –full is present, use first or last price, whichever has not yet been determined
4. otherwise, consider a two year interval.

The first period determined following this procedure is chosen. If additional options are given, these are ignored (e.g., if –start, –end, –full are given, –full is ignored).

–timeframe=1min|5min|10min|15min|30min|hour|3hour|day|week|month|year

The timeframe can be any of the available modules in GT/DateTime.

–max-loaded-items

Determines the number of periods (back from the last period) that are loaded for a given market from the data base. Care should be taken to ensure that these are consistent with the performed analysis. If not enough data is loaded to satisfy dependencies, for example, correct results cannot be obtained. This option is effective only for certain data base modules and ignored otherwise.

–verbose

Makes scan.pl and invoked methods talkative (default - false)

–nbprocess=2

If you want to start two (or more) scans in parallel (useful for machines with several CPUs for example).

–html

Output is generated in html (default - false)

–url="url"

If html output enabled then embed this url as href (default - http://finance.yahoo.com/l?s=<code>)

–options=<key>=<value>

A configuration option (typically given in the options file) in the form of a key=value pair. For example, –option=DB::Text::format=0 sets the format used to parse markets via the DB::Text module to 0.

EXAMPLES (culled from devel archive)

To scan for all stocks that are trading above both their 30 day and 150 day EMAs create a system file containing this GT::Signals description (as a single line)

S:Generic:And {S:Generic:Above {I:Prices} {I:EMA 30}} {S:Generic:Above {I:Prices} {I:EMA 150}}

To scan for all stocks that are trading below both their 30 day and 150 day EMAs create a system file containing this GT::Signals description (as a single line)

S:Generic:And {S:Generic:Below {I:Prices} {I:EMA 30}} {S:Generic:Below {I:Prices} {I:EMA 150}}

Dates

```
If the user has Date::Manip installed it allows the use of date strings
that can be parsed by Date::Manip in addition the to defacto standard
date-time format accepted by GT (YYYY-MM-DD HH:MM:SS) time part is optional

Date::Manip is not required, without it users cannot use short-cuts to
specify date strings. such short cuts include
--start '6 months ago'
--end 'today'

Date string checking includes verifying the date string format
is valid and the date is a valid date (and time if provided)

Errors will be displayed and the script will terminate.
```

```
The script also validates that the dates specified are consistent
with respect to their purpose (--start is earlier than --end etc)

Finally, appropriate timeframe conversion is performed so the user
need not convert command line date strings from the day time to
say week or month as it will be done automagically.

Usage examples:

with market_file (a file) containing the next 2 lines:
JAVA
AAPL

with system_file (a file) containing the next 6 lines:
# example system_file
#
# todays price close was above open

S:Generic:Above { I:Prices CLOSE } { I:Prices OPEN }
# end of system_file

with Date::Manip installed
%    scan.pl --timeframe day --start '6 months ago' --end 'today' market_file \
 'today' system_file
prints

Signal: S:Generic:Above {I:Prices CLOSE} {I:Prices OPEN}
 AAPL - APPLE INC

replace day with week and you will (should) get:

Signal: S:Generic:Above {I:Prices CLOSE} {I:Prices OPEN}
 AAPL - APPLE INC
 JAVA - SUN MICROSYS INC

without Date::Manip you will need to use:
%    scan.pl --timeframe day --start 2007-04-24 --end 2007-10-24 market_file \
 2007-10-24 system_file
or
%    scan.pl --timeframe week --start 2007-04-24 --end 2007-10-24 market_file \
 2007-10-24 system_file
and should get the same results respectively
```

"Bad system call" failure on cygwin

If you are using cygwin on Windows to run GT, and you encounter a "Bad system call" error when running scan.pl, you need to enable cygserver. cygserver is a utility that provides cygwin applications with persistent services. See

http://www.cygwin.com/cygwin-ug-net/using-cygserver.html for more detail. The first time you use cygserver, execute /usr/bin/cygserver-config to configure the service (there are many options but the above should suffice, see the manual for more). You can then invoke the service automatically through windows, or use net start cygserver to do so. You must also set the CYGWIN environment variable to 'server': CYGWIN=server export CYGWIN

select_combination.pl [–limit-ratio <min_ratio_perf/draw_down>] [–limit-performance <min_perf>] [–set <set>]

Display a list of the best "code <-> system" combination possible. It selects the system with highest ration "performance / maw_draw_down". You can decide to exclude some system if they have a ration less than a minimum ration by using –limit-ration, you can also exclude systems if they have a performance less than a minimum given by –limit-performance.

15 ./test_indicator.pl [–full] [–last-record] [–verbose] <indicatorname> <code> [args...]

Examples: ./test_indicator.pl SMA IBM [100] ./test_indicator.pl –full RSI 13000

Args are passed to the new call that will create the indicator.