Harmony Search Algorithm in TradingBlox Builder – single system optimization.

Ian Rayner, 2011 - 01 – 19, v1.0

For background refer to:

Harmony Search Algorithm - My Blog

How Does Harmony Search Work? - My Blog

Harmony Search on Wikipedia

Introduction

Instructions to set up Harmony Search Algorithm (HSA). Uses the ATR Channel Breakout system as an example.

Can have multiple systems within a suite, but HSA Blox only designed to work within one system.

System Set-Up

1. Copy the system that will be the subject of the optimization. I suggest changing its name from "System Name" to "HS System Name". "ATR Channel Breakout" becomes "HS ATR Channel Breakout".
2. Make a list of each of the blox in the system: TradeDirection Portfolio Manager, ATR Channel Breakout Entry Exit and Fixed Fractional Money Manager.
3. Make a copy of ANY blox whose input parameters will be varied in the search for an optimum. I suggest giving each blox with name "Blox Name" a new name: "HS Blox Name". Be careful to maintain any execution order dependencies. We are going to vary inputs to all the blox so they are all copied: HS TradeDirection Portfolio Manager, HS ATR Channel Breakout Entry Exit and HS Fixed Fractional Money Manager.
4. In the copy of the system, replace each "Blox Name" with the corresponding "HS Blox Name". So the HS ATR Channel Breakout system should now contain three blox: HS TradeDirection Portfolio Manager, HS ATR Channel Breakout Entry Exit and HS Fixed Fractional Money Manager
5. Make a copy of the Harmony Search Blox and give it a name appropriate to the system. I suggest naming it as an abbreviation of the system name, "HS Sys Name". Add it to the system. Let's call it "HS ATR Channel BO". Important: this blox must execute before the blox that do the trading. Therefore make sure its name sorts alphabetically ahead of the blox that make trading decisions.
6. Add the "Ruinous Exit" block to the system.
7. Create a new suite and put the "HS" version of your system in it. So we should now have HS Test Bed made up of the new HS Channel Breakout system.

Variables

We need to ensure that any indicators that depend upon input parameters that will be varied during the optimization AND figure into the trading decisions are handled correctly. It is handy to make a table of these variables containing various useful bits of information such as plot colors, plot areas, calculations, etc In the ATR Channel Breakout System these are:

- AverageClose blue, price chart

- AverageTrueRange, not plotted
- ChannelTop, green, price chart, averageClose + entryThreshold x averageTrueRange
- ChannelBottom, green, price chart, averageClose – entryThreshold x averageTrueRange
- ExitTop, red, price chart, averageClose + exitThreshold x averageTrueRange
- ExitBottom, red, price chart, averageClose – exitThreshold x averageTrueRange

For each such indicator do the following

1 rename them xIndicatorName and change plot to trace, switch off display value.

2 modify any indicator names used in calculated indicators to xIndicatorName.

3 create an IPV called indicatorName (auto-index, plot thin line)

4 In Script:Update Indicators add an appropriate calculation using the parameter input and assigning the result to indicatorName. Make sure you put this early in the script, before the value is used.

5 All the pesky interpreter messages should now have disappeared – check each script to ensure this is so. Fix any errors you find.

Hint: for average true range create a new indicator “trueRange” and set it to average true range with a constant of 1 bar – then average this in update indicators.

You have to do this because TBB gets the inputs to any indicator calculations from the user-entered inputs BEFORE HS gets a chance to change those inputs.

In our example, Script: Update Indicators becomes:

```
instrument.averageClose = EMA(instrument.close,
instrument.averageClose[1], closeAverageDays)
instrument.averageTrueRange = EMA(instrument.trueRange,
instrument.averageTrueRange[1], atrDays)
instrument.channelTop = instrument.averageClose + (entryThreshold
* instrument.averageTrueRange)
instrument.channelBottom = instrument.averageClose -
(entryThreshold * instrument.averageTrueRange)
instrument.exitTop = instrument.averageClose + (exitThreshold *
instrument.averageTrueRange)
instrument.exitBottom = instrument.averageClose - (exitThreshold
* instrument.averageTrueRange)
```

Blox: HS Portfolio Manager (HS Trade Direction PM)

Now we have to do a few minor edits to make things run smoothly. In the portfolio manager we need to ensure that if the system settings result in “ruin” (i.e. draw-down exceeds an amount set in “Ruinous Exit”) we stop trading and ignore the results of that set of parameters.

Ruinous exit takes care of closing out positions when ruin occurs and sets the “ruined” flag (a test-scoped BPV).

First create an external BPV called “ruined” in the Portfolio Manager and give it a description such as “Flag indicating ruin”.

In Script: Filter Portfolio use the condition ruined = 1 (“TRUE”) to block trading:

```
IF NOT(ruined) THEN
```

```
' Set to trade TYPE: indicated by tradeDirection parameter
  if tradeDirection = TRADE_ALL then instrument.AllowAllTrades
  if tradeDirection = TRADE_LONG then instrument.AllowLongTrades
  if tradeDirection = TRADE_SHORT then
instrument.AllowShortTrades
ENDIF
```

Blox: HS System Name (HS ATR Channel Breakout Entry Exit)

If there are mutually exclusive settings (such as those for entryThreshold and exitThreshold), make sure there is logic to abort running any tests with meaningless combinations. In this case, we must put the following statement in Script: Before Test:

```
PRINT ">>>>>>>>>>"
PRINT system.name, block.name, "Script: Before Test",
test.currentParameterTest

' Test settings are reasonable
IF (entryThreshold <= exitThreshold) THEN
  PRINT "entryThreshold < exitThreshold:  Aborting"
  test.AbortTest()
ENDIF
```

Blox: HS Sys Name (HS ATR Channel BO)

We have to set up the variables we are going to use to pass values into the parameters of the blox doing the trading. These values override those the user enters in the Simulation Parameters Editor.

Make a table of all the parameters you wish to include in your search optimization process, decide on the nature of each and identify a maximum and minimum value for each. While you are doing this, make sure the scope of each parameter is “system”. By “nature” I mean

- Integer (i.e. takes integer values such as atrDays)
- Switch (i.e. takes integer values but the values themselves convey no sense of “size”, such as tradeDirection)
- Continuous (i.e. infinitely divisible values such as entryThreshold)

The maximum and minimum values define the search space. For “Switch” you need to understand how selectors and TRUE / FALSE works in TBB. These are essentially integer values. Selectors start at zero, so a three-option selector has a minimum of 0 and a maximum of 2. TRUE / FALSE are 1/0 respectively.

For each of the parameters create an external BPV. It must have the exact same name as the system-scoped parameters in the blox making the trading decisions. Make sure to set the “type”.

In the present example we have:

- 1 HS ATR Channel Breakout: atrDays, Integer, 1 -100
- 2 HS ATR Channel Breakout: closeAverageDays, Integer, 1 - 400
- 3 HS ATR Channel Breakout: entryThreshold, Continuous, 0 - 5
- 4 HS ATR Channel Breakout: exitThreshold, Continuous, 0 – 5, less than entryThreshold
- 5 HS Fixed Fractional MM: riskPerTrade, Continuous, 0.1% - 2%
- 6 HS Trade Direction PM: tradeDirection, Switch, 0 – 2 (i.e. 3 position)

Blox: HS Sys Name; Script: Before Simulation (HS ATR Channel BO)

There are two sections in this script for user-stuff. Sorry – one input is needed to size an array that the user then fills. Once again using a table to keep records is helpful.

Enter the first block of user-bits, mostly they are self-explanatory, I have used entries appropriate to our example:

- fileLocation = "C:\Documents and Settings\……."
- HMfileName = "HM.txt" ' name of file to save final harmony memory. Use ".xxx"
- HMheaderString = "atrDays,closeAvg,entry,exit,risk,direction,MAR"
- HMhistFileName = "HMHist.txt" ' name of file to save history of harmony memory. Use ".xxx"
- HMhistAddDataHdr = "Trades, R-Sq" ' List additional data fields sep by commas
- randSeed = 1 ' = 0 => for live use
- parameterCount = 6 ' # of paramters you want to vary in the search
- harmonyMemorySize = 32 ' 2 ^ ("C" + "I" parameters)
- harmonyMemChgRate = 0.89 ' suggest exp(ln(0.5) / parameterCount)
- pitchAdjustRate = 0.25 ' suggest 0.25
- fretWidth = 0.25 ' suggest 0.25

Whatever you use for filenames, the code will append a date/time stamp when creating the files. This is a blessing or a curse – you don't lose prior results, but the files breed like rabbits. The only difficult one is the "HMheaderString" – put what you want to see at the tops of the columns of results – include the name of the objective function (your bliss function, the thing you will use to judge optimization, I am using MAR in this example). There are hints in the code for values to use.

If you want to print additional statistics to the history file, add their names (comma-separated) to the "HMhistAddDataHdr" variable. The header for the history file is constructed from the a default header plus HMheaderString plus HMhistAddDataHdr .

When de-bugging, set randSeed to a number so testing is repeatable. When you run the search in earnest you must set the seed to zero to get actual random numbers.

You don't have to go over the top in the harmony memory size – even here I am using 32 rather than 64. It will work fine.

Harmony Memory Array

Go to the second block of user-bits and enter values in the arrays for the type and max and min for each parameter – be sure to change the index values for each line. It should look like this:

```
' row 1 specifies parameter type: "C"ontinuous, "I"nteger,
"S"witch.
' "C" => infinitely divisible e.g. stop as multiple of ATR
' "I" => integer e.g. days in moving average
' "S" => selector / TRUE(1)/FALSE(0)
test.SetStringArrayElement(harmonyMemorySize + 1, 1, "I")
test.SetStringArrayElement(harmonyMemorySize + 1, 2, "I")
test.SetStringArrayElement(harmonyMemorySize + 1, 3, "C")
```

```
test.SetStringArrayElement(harmonyMemorySize + 1, 4, "C")
test.SetStringArrayElement(harmonyMemorySize + 1, 5, "C")
test.SetStringArrayElement(harmonyMemorySize + 1, 6, "S")

' row 2 specifies parameter minimum value
test.SetStringArrayElement(harmonyMemorySize + 2, 1, "1")
test.SetStringArrayElement(harmonyMemorySize + 2, 2, "1")
test.SetStringArrayElement(harmonyMemorySize + 2, 3, "0")
test.SetStringArrayElement(harmonyMemorySize + 2, 4, "0")
test.SetStringArrayElement(harmonyMemorySize + 2, 5, "0.001")
test.SetStringArrayElement(harmonyMemorySize + 2, 6, "0")

' row 3 specifies parameter maximum value
test.SetStringArrayElement(harmonyMemorySize + 3, 1, "100")
test.SetStringArrayElement(harmonyMemorySize + 3, 2, "600")
test.SetStringArrayElement(harmonyMemorySize + 3, 3, "5")
test.SetStringArrayElement(harmonyMemorySize + 3, 4, "5")
test.SetStringArrayElement(harmonyMemorySize + 3, 5, ".04")
test.SetStringArrayElement(harmonyMemorySize + 3, 6, "2")
```

Blox: HS Sys Name; Script: Before Test (HS ATR Channel BO)

Script: Before Test creates each successive “generation” of parameters for each test. The set of parameters so generated is stored in the “candidate” array. The last thing we need to to is hand off these values to the parameters that are used in the trade-generating blox.

First we need to create each of the parameters in the HS block. In the HS block they are going to be Block Permanent Variables. They must have the EXACT same names as their counter-parts in the blox that that make the trading decisions. They are set up as external variables with an appropriate data type (integer for True / false and selectors).

We assign the values at the end of the script:

```
atrDays = candidate[1]
closeAverageDays = candidate[2]
entryThreshold = candidate[3]
exitThreshold = candidate[4]
riskPerTrade = candidate[5]
tradeDirection = candidate[6]
```

Because these parameters were given “system” scope when they were set up in their respective blox, these statements will assign the values in the candidate array, overriding the user entered values.

Blox: HS Sys Name; Script: After Test (HS ATR Channel BO)

Once again there are two sections for user inputs in this script. One group has to be before the IF - THEN construct as it is part of the IF, the other has to be inside it as it is part of the THEN...

Near the top of this script you add in the assignment or calculation of the objective function (the statistic you want to maximize). This might be as simple as objectiveFunction = test.rar, or it might involve the calculation of some custom statistic YOU prefer. If you do any division, be sure to check for a zero denominator (use the IFTHENELSE structure). In this case we use the MAR ratio.

In addition, you may filter the tests by any other measures here. In this case we use the total number of trades in the test. If you have no need to filter then set acceptResults = 1, otherwise EVERY test will be filtered out.

```
objectiveFunction = test.marRatio
acceptResults = test.totalTrades > 1000
```

Remember to set the “Goodness” statistic to the same value as your objective function.

Control of any extra statistics you want on the screen or in the history file is implemented in the second section of user inputs in this script. Use as many test.AddStatistic calls as you want. Then construct the addData string. Always use AsString() for the first statistic otherwise the “+” will actually add the numeric values together then convert to a string rather than concatenating the numerical values as a string. In this case I added the RAR and the R-Squared of the equity curve:

```
test.AddStatistic("RAR%", test.rar, 2, "Percent")
test.AddStatistic("R-Sq", test.rSquared, 2, "Percent")
addData = AsString(test.totalTrades) + “,” + test.rSquared
```

Running the Search

At this point we should be good to go. It is a good idea to set some meaningful values in the test parameter input screen so that you can look at the trade charts and see that the tests are actually using DIFFERENT values from the inputs. Open the log window.

First just run the simulation with Auxiliary: Cycle dummy variable = 0. This will make sure the system is functioning as it normally would. A key thing to observe is that the original indicators (displayed as traces) should be approximately the same as the calculated indicators (solid). They may not be exactly the same, and this will result in inconsistencies in the test runs between the HS and the regular operation – small inconsistencies in trades can add up dramatically. This is a lesson in the issue of “robustness”.

Next run the simulation with Auxiliary: Cycle dummy variable = 1 to 5 step 1. You should see lots of information in the log window. If you want to know what HS is doing, study the log. You must also locate the HM and HMhist files, open them and verify that they contain the right values. With the set up described above they should contain many common values, with more info in the history file.

Don’t be surprised if the history and HM files are empty – it is entirely possible to filter out all 5 of the tests. If this is the case, double up to 10. If you can’t seem to get a viable parameter set, check your work and relax some of the filter constraints.

Following is the meaning of the entries in the history file:

1. TotCyc: same value as the dummy stepped parameter. It is the number of the test.
2. CompCyc: the number of completed cycles. I.e. a count of the number of cycles NOT aborted.
3. HMRow: Which row in the harmony memory this test was added to. If the test completed but was not added, the value will be zero.
4. Replace: A count of the number of replacements made in the HM. If this number is “n” then this was the nth parameter set to be added to HM replacing an existing member.
5. Everything else should be self-explanatory.

The next test is to increase the number of runs above the HM size to check that new, fitter parameter sets are replacing old, weaker ones. Since in our example, 50% of the time we will abort the test due to entryThreshold and exitThreshold values being wrong, and HM size is set to 32, a good next step size is 100.

Watch the log scroll by. You should see 32 messages of the form: “adding current candidate (…) to HM at row N” messages. Eventually you will see messages of the form: “Replacing worst HM row: N”. This lets you know the harmony memory is being updated with “fitter” solution sets.

When the stepped variable chart is displayed, you should see a general improvement in the peaks of the goodness statistic as cycle count increases. You can see this better with runs in the thousands, when you can see the “envelope” containing the peaks gradually rising left to right.

Results

Remember you will unlikely get the same results as me or anyone else as differences in the data, starting dates, global parameters, etc will cause differences in the results.

After 5,000 runs, nearly every member of the harmony memory should be quite similar, with a couple of outliers. Using my data (the free portfolio from CSI converted to ratio adjusted contracts, ruin at 60% TEDD and aborting on ruin) I get optimum values:

- Trade Direction: Trade All
- Risk Per Trade ~ 1.6%
- ATR (Days) ~ 20
- Close Average (Days) ~ 490
- Entry Threshold ~ 1.37
- Exit Threshold ~ 0.85
- MAR ~ 0.9

I notice, however, there seem to be a couple of clusters in the results that had yet to finish the struggle for survival. I was still seeing a 7% replacement rate over the last 100 tests, so I imagine the results would continue to improve with longer runs.

In the early part of the optimization, only 35 out of the first 1000 tests ran to completion, the remainder were filtered out before running (entry threshold < exit threshold), during the test (due to ruin) or after the test completed (insufficient trades). Over the last 1000 cycles 448 of 1000 completed. These statistics can be deduced from the Hmhistory.txt file.

The solution is the BEST member of the harmony memory, NOT the average member.

Good luck and PM me with any questions or add them to the thread so everyone can benefit.