Nonlinear Factor Model User Guide

April 28, 2014

Contents

1 DataBase Tables

Three tables with scheme name `anl` are created.

`modelmstr`

Record every model using nonlinear factor combination techneque and associated parameters.

`modelid`	int	model identifier.
`aggid`	char	universe identifier.
`window`	int	number of periods an investment cycle has, used as smoothing window.
`tcost`	float	transaction cost.
`PFMethod`	char	portfolio construction method, either 'EqualWeight' or 'SectorNeutral'
`prctileFilter`	char	percentile filter condition for all factors, form of `[lp1 rp1; lp2 rp2 ...]`, just like a normal 2-column matrix in MATLAB with each row defines a percentile range, and stocks with factor value located in the the range(s) will be selected.
`specificFilter`	char	allows to set different filtering percentiles for different factors, form of `factorid1 prctileFilter1, factorid2 prctileFilter2...` where `prctileFilter`*n* has the same format as that of `prctileFilter`.
`freq`	char	sampling frequency, used when loading data.
`univname`	char	a name give to the universe indicated by column `aggid`.
`normlevel`	int	normalization level when apply normalization with GICS to factor and stock data.

`annealingFilter`

Record factors used by model

`modelid`	int	mode identifier registered in table `modelstr`.
`caseid`	int	corresponding to every model there are many cases involved different factors. `caseid` used to differentiate these cases.
`factorid`	char	factor id.

`weights`

Record portfolios generated by nonlinear combination.

`modelid`	int	mode identifier registered in table `modelstr`.
`caseid`	int	corresponding to every model there are many cases involved different factors. `caseid` used to differentiate these cases.
`date`	datetime	date of weight
`secid`	char	security identifiers
`weight`	float	weight of the security in the portfolio

2 Production Usage

To run nonlinear combination in production, invoke command like this:

```
anl <modelid> <rundate>
```

where `modelid` is the the model identifier recored in database, `rundate` is the date until then the data will be loaded.

In production, the program load model parameters from table `modelmstr` and load all necessary data (stock- and factor-related) based on these parameters and information in table `annealingFilter` to build an `Annealing` object, then construc protfolio upon them. Resulting portfolio will be stored in table `weights`.

3 Research Usage

In research statge, we need to build models using annealing code by trying different parameters. The rersult is presented by a set of combinations of factors which is picked up based on their performance measures (information ratio). Each set of factor combination is called a case (and corresponds to a caseid in production). After having an approriate case in hand, you can write the related information to database (model parameters go to `modelmstr`, factor conbinations go to `annealingFilter`) for production use.

The script file `test.m` shows how to use annealing code to build a model. There are many parameters involved controlling different aspects of annealing process. We will now go through the script.

1. Create an `Annealing` object.

```
1  if exist('o.mat', 'file')
2      load('o.mat');
3  else
4      facids = mat2cell(num2str((1:220)','F%5.5d'), ones(220,1));
5      o = Annealing('00053', facids, '2003-12-31', '2011-12-31', 'M', 'SP500');
6      save('o.mat', 'o');
7  end
```

   Note code checking whether a cache file `o.mat` exists. Usually loading data from database is slow, and we are almost always in need to try different parameter sets to a dataset. The file `o.mat` serves as a cache so that we don't need to load the same dataset from database more than once.

   If no cache file exists, `Annealing` object will be created from database using specified universe (`aggid`,1st parameter) and factors (`facids`, 2nd parameter) over specified period (`startDate` and `endDate`, 3rd and 4th parameters respectively) in specified frequency (`freq`, 5th parameter). The last parameter is an universe name associated with the first parameter (`aggid`) to provide a name to the given universe which will be used and shown in report file generated by calling `report()` function. Finally, in line 6 the object just created is saved to a cache file.

   Note the cache mechanism is kind of rough; *if anything related to the parameters in construction is going to change, please first remove the old cache file (if there is one).*

2. Set parameters. Most of parameters involved get default values and only need to be set if you want to try different values than the default ones. There are a few, however, are mandatory; i.e., they must be set explicitly by users. The next code snippet shows all the mandatory parameters. You can refer the comments for their meaning.

```
8  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9  % Following parameters must be set by user, otherwise program won't work.
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11
12 %% In-sample periods
13 o.samplePeriods(1).start  = '2003-12-31';
14 o.samplePeriods(1).end    = '2006-08-31';
15 o.samplePeriods(1).weight = 1/3;
16 o.samplePeriods(2).start  = '2006-09-01';
17 o.samplePeriods(2).end    = '2010-12-31';
18 o.samplePeriods(2).weight = 2/3;
19 % ... you can continue this with something like
20 %        o.samplePeriods(3).start  = sth;
21 %        o.samplePeriods(3).end    = sth;
22 %        o.samplePeriods(3).weight = sth;
23 % if you like.
```

```
24
25      %% Hard constraint parameters, index can only be 1.
26      o.hardMetrics(1).IR = 0.2;      % hard constraint: IR of factors picked must be > this value
27      o.hardMetrics(1).IC = 0.005;    % hard constraint: weighted (by o.samplePeriod.weight) average of IC
28                                      %    must be > this value
29      o.hardMetrics(1).IC_means = [0 0];% hard constraint: average of IC in each insample period must be >
30                                        %   this value. # of elements should == number of insample periods
```

Following parameters are optional which all get default values:

```
31      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32      % All the following parameters got default values.
33      % NO NEED to have them in your code if you insist on the default values.
34      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35
36      %% Filtering parameters
37      o.nMinFacPerType = 1;           % minimum number of factors picked up for each type; DEFAULT 1
38      o.nMaxFacPerType = 1;           % maximum number of factors picked up for each type; DEFAULT 1
39      o.alwaysSelected = {'F00072'}; % list of factors must be selected like 'F00072' for S&P 500; DEFAULT {}
40      o.neverSelected  = {};          % list of factors never be selected; DEFAULT {}
41      o.prctileFilter  = [1/3 1];     % Stocks with factor value greater than corresponding percentile value
42                                      %    be selected; DEFAULT [1/3 1]
43      o.tcost          = 40/10000;    % transaction cost; DEFAULT 40/10000.
44      o.window         = 6;           % number of periods an investment cycle has; DEFAULT 6
45      o.coverage       = 0.5;         % coverage of factors must greater than this value; DEFAULT 0.5 (50%)
46      o.PFMethod  = 'EqualWeight';    % portfolio construction method, either 'EqualWeight' or
47                                      %   'SectorNeutral'; DEFAULT 'EqualWeight'
48      o.specificFilter = {'F00072' [0 2/3]};  % allows to set different filtering percentiles for
49                                            %    different factors; DEFAULT {}
50
51      %% Annealing parameters: increasing nIteration costs more time but gets more reliable results
52      o.temperatures = 0.5 * 0.75 .^ (0:19); % temperatures to be annealed; this is the DEFAULT value
53      o.nIteration   = 10000;         % number of loops for each temperature; DEFAULT 6000
```

Note that parameters `prctileFilter`, `specificFilter`, `tcost`, `windows`, `PFMethod`, along with the constructor parameter `aggid`, `freq` and `univname` are used in portfolio construction and therefore stored in database table `modelmstr`.

Right after setting all parameters, you must call `applyOptions` to bring them into effect:

```
54      o = o.applyOptions;    % must do this
```

3. Kick off annealing by runing

```
55      [energies, states] = o.run(inf);
```

where `run` can take two parameters:

- the first is a number specifying total number of states to be returned in order of performance measure(i.e., insample IR),
- the second is optional which allow user to provide an initiate state (*which must satisfy the constraints specified by parameters* `nMinFacPerType`, `nMaxFacPerType`, `alwaysSelected`, *and* `neverSelected`).

If the second parameter is omitted (this is the case in the code snippet above), `run` will internally generate a qualified initial state.

`run` returns two things:

- the first returned (`energies`) is the peformance meansures of states selected, it's a vector of size equal to the first parameter (number of states to be picked up) passed into it. [1] In this case it actually the negated IR.
- the second are the states matrix, with each column corresponds to a state and therefor to a portfolio, and the row dimension – number of states – equal to length of the first returned, the column dimension is the total number of factors (can be referenced by `length(o.facids)`) in dataset. Value greater than 0 means the corresponding factors are selected.

4. Generate report and get final nonlinearly combined portfolio weight (already smoothed over `window` periods).

```
56    [pfw, r] = o.report('test', states(:, 1:50));
```

`report` takes two parameters: the first gives the filename of the report (no file suffix needed), the second passes the states returned from the last step. You don't need to pass all states returned from step 3. In this example code, we passed 50 states (`states(:,1:50)`, which correspond to 50 portfolios). The final portfolio returned in `pfw` is the average of these 50 portfolios. `r` returned in the second output argument is the final portfolio's return series. Both `pfw` and `r` are `myfints`.

5. Finally, you can select to write the model to database if you decide this model should go to production.

```
modelid = o.saveModel(states(:,1:50));
```

`modeid` returned can be used to retrieve the model back from database later by

```
[o, caseids, states] = Annealing.LoadModel(modelid, rundate);
```

where `o` is an annealing object, `caseids` are ids for each state (columns of `states`,or factor conbinations).

At this point, constructing a (nonlinear-combined) portfolio is a piece of cake:

```
o.savePF(modelid, caseids, states);
```

which not only creates the portfolio, but also writes the portfolio weight to `weights` table. The last two code snippets actually is all the production code in `main.m`.

[1] Unless total number of all combinations possible are less than specified.