A tutorial of 'August: The program for calculation of optimal trading rules using fat-tail distributions'

Pablo Risueño[1]

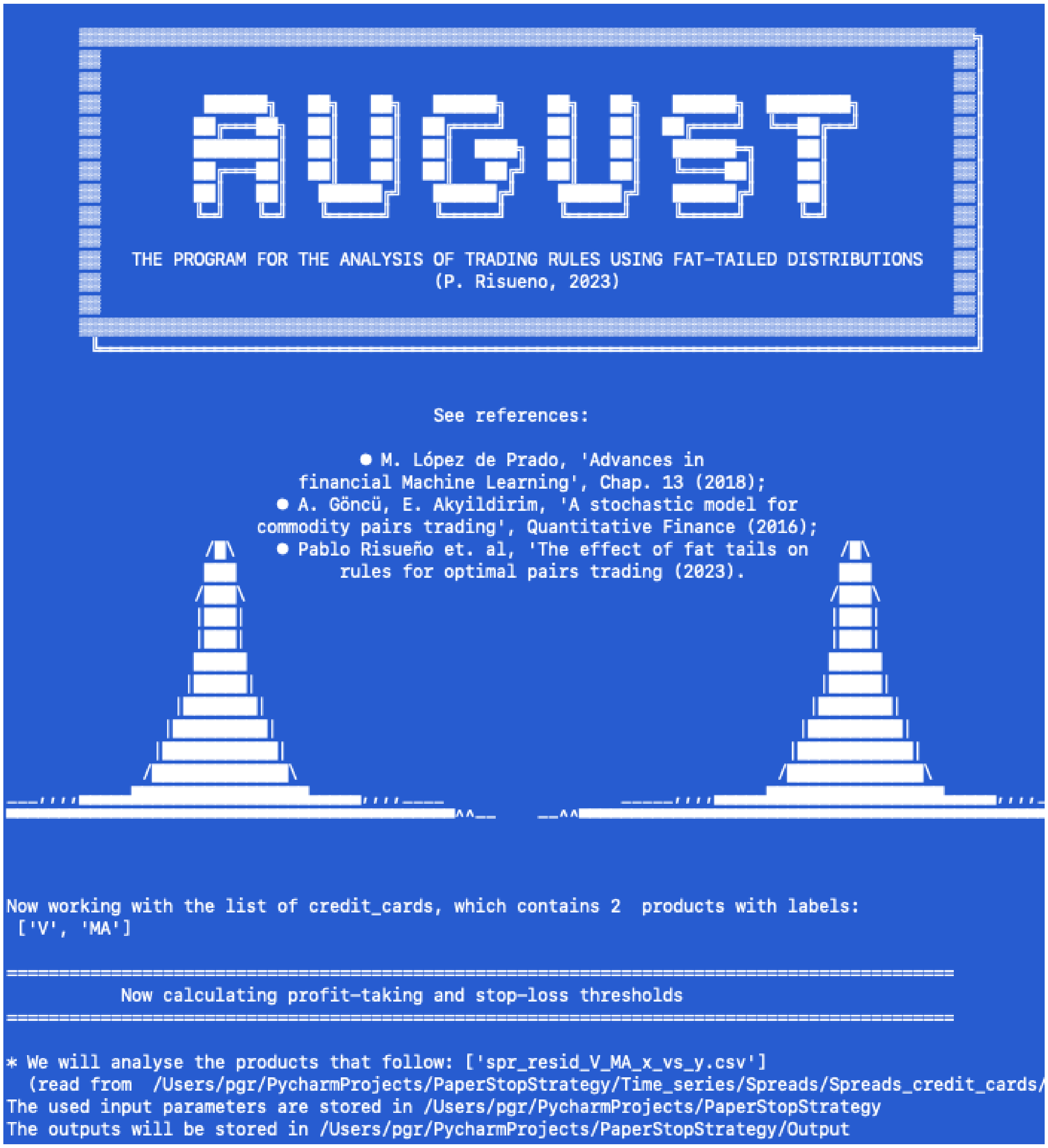
Independent scholar

(Dated: 10 October 2023)

This document explains how to perform calculations using the code `August`. This software makes it possible to download data from Yahoo Finance and to find optimal trading rules for them. Both pairs trading and trading of single financial products can be simulated. The code also permits to fit the observed data to normal or fat-tailed distributions, and to perform out-of-sample calculations. The corresponding calculations can be executed in a single PC or laptop.

Important notice

The August code is freely distributed as open-source software, and can be freely used and modified upon the citation of its author and its corresponding paper [*The effect of fat tails on rules for optimal pairs trading*, by Pablo Risueño et al. (2023)]. It is designed for non-profit research, it is by no means a recommendation for investing. It comes with absolutely no guarantee.



```
THE PROGRAM FOR THE ANALYSIS OF TRADING RULES USING FAT-TAILED DISTRIBUTIONS
                          (P. Risueno, 2023)


                           See references:

                    ● M. López de Prado, 'Advances in
              financial Machine Learning', Chap. 13 (2018);
            ● A. Göncü, E. Akyildirim, 'A stochastic model for
          commodity pairs trading', Quantitative Finance (2016);
            ● Pablo Risueño et. al, 'The effect of fat tails on
                rules for optimal pairs trading (2023).



Now working with the list of credit_cards, which contains 2  products with labels:
 ['V', 'MA']

===========================================================================================
           Now calculating profit-taking and stop-loss thresholds
===========================================================================================

* We will analyse the products that follow: ['spr_resid_V_MA_x_vs_y.csv']
  (read from  /Users/pgr/PycharmProjects/PaperStopStrategy/Time_series/Spreads/Spreads_credit_cards/
The used input parameters are stored in /Users/pgr/PycharmProjects/PaperStopStrategy
The outputs will be stored in /Users/pgr/PycharmProjects/PaperStopStrategy/Output
```

CONTENTS

I. INTRODUCTION

This is a commentary on how to use the program *August*. This software is aimed at calculating optimal trading rules of single products and long-short pairs taking into account fat-tail probability distributions. The code is downloadable at:

`https://github.com/pablogr/FIN_optimal_trading_rules_fat_tails`

There are four main tasks that can be performed by the code:

- Downloading data of financial time-series from Yahoo Finances (e.g. prices of stocks or cryptocurrencies).

- Calculating returns and spreads of long-short pairs, and performing a fitting to normal or fat-tailed probability distributions.

- Calculating optimal trading rules (enter value, profit-taking, stop-loss, maximum horizon) using synthetic data.

- Calculating out-of-sample profits using the observed data.

Each of the tasks listed above requires the ones previous to it. Each of the four blocks above has its own block the input file which contains the parameters specified by the user (`input.py`). In addition, there exist input parameters which affect to all or most of the tasks listed above; these parameters belong to the *general* block.

The name of the code is a tribute to the mathematician Augustin Louis Cauchy, a pioneer in the analysis of fat-tailed distributions.

This document is structured as follows: In sec. II we present a few remarks on the installation of the software. In sec. III we provide a simple example of input file which determines the user-specified parameters. In sec. IV we briefly explain the general input parameters. In sec. V we explain how the download and fit of data is performed. In sec. VI we explain how optimal trading rules are calculated. In sec. VII we present how out-of-sample profitability calculations are carried out. Finally, in sec. VIII we explain the meaning and possible values of each of the input parameters of `input.py`.

II. INSTALLATION

The code can be downloaded from GitHub and executed by running a standard distribution of Python 3.7 or higher. The list of modules to install in the corresponding Python interpreter is:

- `pandas`
- `numpy`
- `scipy`
- `scikit-learn`
- `statsmodels`
- `matplotlib`
- `seaborn` (to plot heatmaps; not essential)
- `yfinance` (to download time series from Yahoo Finance; not essential if the user provides .csv files with the time-series to use)

Other used modules which are usually installed by default are `math`, `random`, `os`, `sys`, and `glob`.

III. EXAMPLE INPUT FILE

Below we present a possible file which specifies the input parameters of the calculations. Its name must be `input.py`. This example file does not include explanations; the parameters can be changed as indicated in further sections.

```
from os import getcwd
from numpy import array, linspace, arange, log
input_directory = getcwd()

# ==============================================================
# GENERAL INPUT PARAMETERS (for all blocks)
# ================================================================

calculation_mode = "out_of_sample"
evolution_type = "Ornstein-Uhlenbeck_equation"
list_distribution_types = ['nct']
list_product_labels = [ "oil_companies", [ "XOM", "CVX", "COP", "PSX",
"VLO", "MPC", "MRO", "EOG", "BP", "SHEL", "TTE", "E", "EQNR", "CNQ",
"OXY" ] ]
verbose = 1
make_plots = False
only_plots = False


# ================================================================
# BLOCK FOR DOWNLOADING DATA AND FITTING THEM TO PROB. DISTRIB.
# ================================================================

first_downloaded_data_date = '2010-11-30'
last_downloaded_data_date = '2022-11-30'
downloaded_data_freq = '1d'

consider_skewness = True
consider_p = True

n_random_trials = 6
max_n_iter = 20

correlation_threshold = 0.5


# ================================================================
# BLOCK FOR EVALUATION OF OPTIMAL STRATEGY
# =============================================================

output_type = "heat-map"
path_rv_params = 'default'

list_max_horizon = [252]
list_enter_value = arange(-0.040000000, 0.00000000001, 0.001)
list_profit_taking = arange(0.000000001, 0.040000002, 0.001)
list_stop_loss = array([-999])

strategy_duration = 252
min_num_trials = 20000

quantity_to_analyse = "Sharpe_ratio"
method_for_calculation_of_profits = "enter_random_many_deals"

discount_rate = None
transaction_costs_rate = 0.02



# ===================================================
# BLOCK FOR OUT-OF-SAMPLE CALCULATIONS
# ===================================================

oos_spread_type = "x_vs_y"
oos_product_label_x = "BMW.DE"
oos_product_label_y = "MBG.DE"
list_product_labels_name = "car_manufacturers"
first_oos_date = '2018-11-30'
last_oos_date = '2022-11-29'
in_sample_t_interval = '5y'
```

```
in_sample_recalibration_freq = '1y'
```

IV. GENERAL PARAMETERS

In the previous section we find up to 7 input parameters in the first block of the input file. Note that in our example the directory of the input file is set to be the one where the python main program lies (through `input_directory = getcwd()`), but this is not really necessary; you can set it to your desired path. The 7 input parameters of the general block correspond to specifications which affect most of the calculation modes / tasks. The first one is `calculation_mode`, which determines which kind of task will be performed by the program. You can set it to `download` to simply download data, `fit` (or `download_and_fit`) to fit the data to probability distributions, `find_trading_rules` to calculate optimal trading rules, or `out_of_sample` to calculate out-of-sample profits.

The second parameter is `evolution_type`, which determines if we will simulate pairs trading (if set to `''Ornstein-Uhlenbeck_equation''`) or single-product trading (if set to `''single-product''`).

The third parameter (`list_distribution_types`) specifies which distributions will be analysed in the equations. It must be a list containing any combination of 'norm' (normal, Gaussian), 'nct' (non-centered t-student), 'genhyperbolic' (generalized hyperbolic) and 'levy_stable' (stable distribution).

The fourth parameter is `list_product_labels`. This specifies the name of the list of products and the labels of the financial products. It is essential for the code to properly download the data. For example, we can ask the code to download the time-series of prices of the stocks of The Coca-Cola Company and Pepsi setting it to `[''cola_companies'', [''KO'', ''PEP'']]`. The name of the set of products ("cola_companies" in this example) is arbitrary; however, the product labels ('KO","PEP" in this example) must be searchable (i.e. downloadable) in Yahoo Finance. The best way to find the appropriate labels is to simply visit `https://finance.yahoo.com` with your internet navigator and enter the name of the sought financial product.

The last three parameters `verbose`, `make_plots` and `only_plots` do not actually determine which calculations will be done. They specify, respectively, how much of the intermediate information is printed to screen, whether or not plots are done in the execution of the program, and whether or not actual calculations are performed.

V. DOWNLOAD AND FIT

If you set `calculation_mode` to `download` in `input.py` you will download data from Yahoo finance. As already stated, you need to specify the labels of the products whose time-series will be downloaded. In order to download the data you also need to specify the limit dates and the downloading frequency with the variables `first_downloaded_data_date`, `last_downloaded_data_date` and `downloaded_data_freq`. The time series will be downloaded to a directory called `Time_series`. Their returns will be also stored in that directory.

If you set `calculation_mode` to `fit` (after having run a `download` execution) or directly to `download_and_fit` and you set the `calculation_mode` to `''Ornstein-Uhlenbeck_equation''` then the *spreads* will be stored to a folder called:
`Time_series/Spreads/Spreads_name_of_your_list_product_labels/Data`.

The spreads are given by:

$$s(t) := \log[p^{A}(t)] - \gamma \log[p^{B}(t)] \,, \tag{1}$$

where γ is the slope of the regression of the logarithmic returns of products A and B. You can find the definition of the calculated pair spreads as well as further explanations in the 2023 article 'The effect of fat tails on rules for optimal pairs trading of stocks and cryptocurrencies based on the Ornstein-Uhlenbeck equation' [1]. The files which contain the spreads consist e.g. in the columns that follow:

- Date
- *labelX*_log_price_Close_corrected
- *labelX*_price_Close_corrected_log_ret

[1] `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4518354`

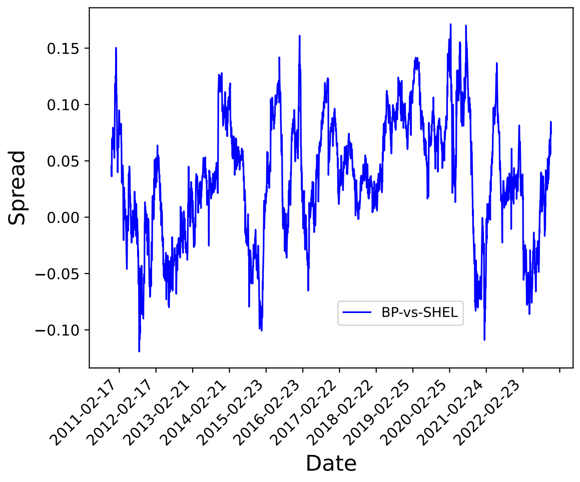


FIG. 1: Time-series of the spread of BP p.l.c. and Shell p.l.c.

- *labelY*_log_price_Close_corrected
- *labelY*_price_Close_corrected_log_ret
- Spread_y_vs_x
- Spread_x_vs_y

where *labelX* indicates the label of the first product of the pair (e.g. BP) and *labelY* indicates the label of the second product (e.g. SHEL). The suffix "corrected" indicates that the prices were recalculated by reinvesting dividends. The "log_ret" suffix indicates logarithmic return, i.e. $\log[p(t)] - \log[p(t-1)]$., where $p(t)$ indicates price at time t. Spread_y_vs_x corresponds to considering the label y corresponding to A and x corresponding to B in eq. (1); spread_x_vs_y corresponds to the converse (i.e. considering the label x corresponding to A and y corresponding to B). In most of the analysed cases just none or one of both spreads is stationary (it is rare that both are).

Plots of the spreads can be found at:
`Time_series/Spreads/Spreads_name_of_your_list_product_labels/Plots`.
An example is displayed in Fig. 1 These plots are useful to give an intuitive notion of the stationarity of the time-series of the spreads.

After having calculated the spreads, our code performs a regression of $s(t)$ vs $s(t-1)$ and finds the parameters (E_0, φ) of the discretized Ornstein-Uhlenbeck equation:

$$s(t) = (1-\varphi)E_0 + \varphi\, s(t-1) + \sigma\, \varepsilon(t-1)\,. \tag{2}$$

The residuals ($\sigma\,\varepsilon$) will be fitted to the probability distributions which were specified in the input parameter `list_distribution_types`. But before that is done, the code checks: i) that the correlation between the returns of the two products which form the pair are high (i.e. above the minimum threshold specified by the input parameter `correlation_threshold`); ii) That the spread to analyse is stationary (i.e. that stationarity cannot be rejected by the Augmented Dickey-Fuller test at above 95% probability). If both tests are passed, then the actual fitting to probability distributions is carried out.

For the normal distribution there is no actual numerical fitting; the parameters of the distribution are simply the mean and the standard deviation of the residuals. For the other three considered distributions (which are fat-tailed) the appropriate distribution parameters are sought using the maximum likelihood and gradient descent methods (the latter together with the Barzilai-Borwein step). The loss function which is minimized is:

$$\text{loss}(\mathbf{p}_d) := -\frac{1}{N_r}\sum_{i=1}^{N_r} \ln[\,\text{pdf}(\mathbf{p}_d)\,] \tag{3}$$

where $\mathbf{p}_d$ is the set of tested distribution parameters, N_r stands for the number of residuals and pdf is the probability density function. Since it has local minima, we test different values of the starting guess for the iterations. The number of times that the minimization procedure is run (for each starting guess) is determined by the input parameter `n_random_trials`. In addition, the maximum number of iterations of each of the runs is given by `max_n_iter`. The higher these parameters are, the more accurate the fitted distribution is expected to be; however, the higher they are, the higher the computing time. Therefore a tradeoff between accuracy and numerical complexity must be settled.

The parameters `consider_skewness` and `consider_p` affect to the three fat-tail distribution and to the generalized hyperbolic distribution, respectively. They determine if the corresponding parameters of the distribution must be considered in the analysis, or simply set to zero (set both to True for maximum accuracy of the fitting).

The output of the fitting is stored to a .csv file in the subfolder of `Data` called `Fitting_parameters`. This subfolder also stores the parameters of the Ornstein-Uhlenbeck equation. If you run it for all four distributions, the fitting output file will contain the columns that follow:

- spread_name: Identifier of the fitted spread.

- OU_phi, OU_E0, OU_Rsquared: Parameters of the Ornstein-Uhlenbeck equation: E_0 (i.e. the mean towards the mean-reverting process tends), φ (velocity of the trend) and the R^2 of the linear ordinary least-squares regression of $s(t)$ vs $s(t-1)$.

- normal_loc, normal_scale, normal_loss: Location and scale parameter of the normal distribution, as well as its loss function (eq. (3)).

- nct_loc, nct_scale, nct_skparam, nct_dfparam, nct_loss: Location, scale, skewness and number-of-degrees-of-freedom (tail) parameters of the best non-centered t-student distribution found, as well as its loss function (the lowest found).

- ghyp_loc, ghyp_scale, ghyp_b_param, ghyp_a_param, ghyp_p_param, ghyp_loss: The five parameters of the generalized hyperbolic distribution, as well as its corresponding (lowest found) loss function.

- stable_loc, stable_scale, stable_beta_param, stable_alpha_param, stable_loss: The four parameters of the levy stable function, as well as its loss function.

The plots located in the directory `Data/Plots` display graphs like those of Fig. 2.

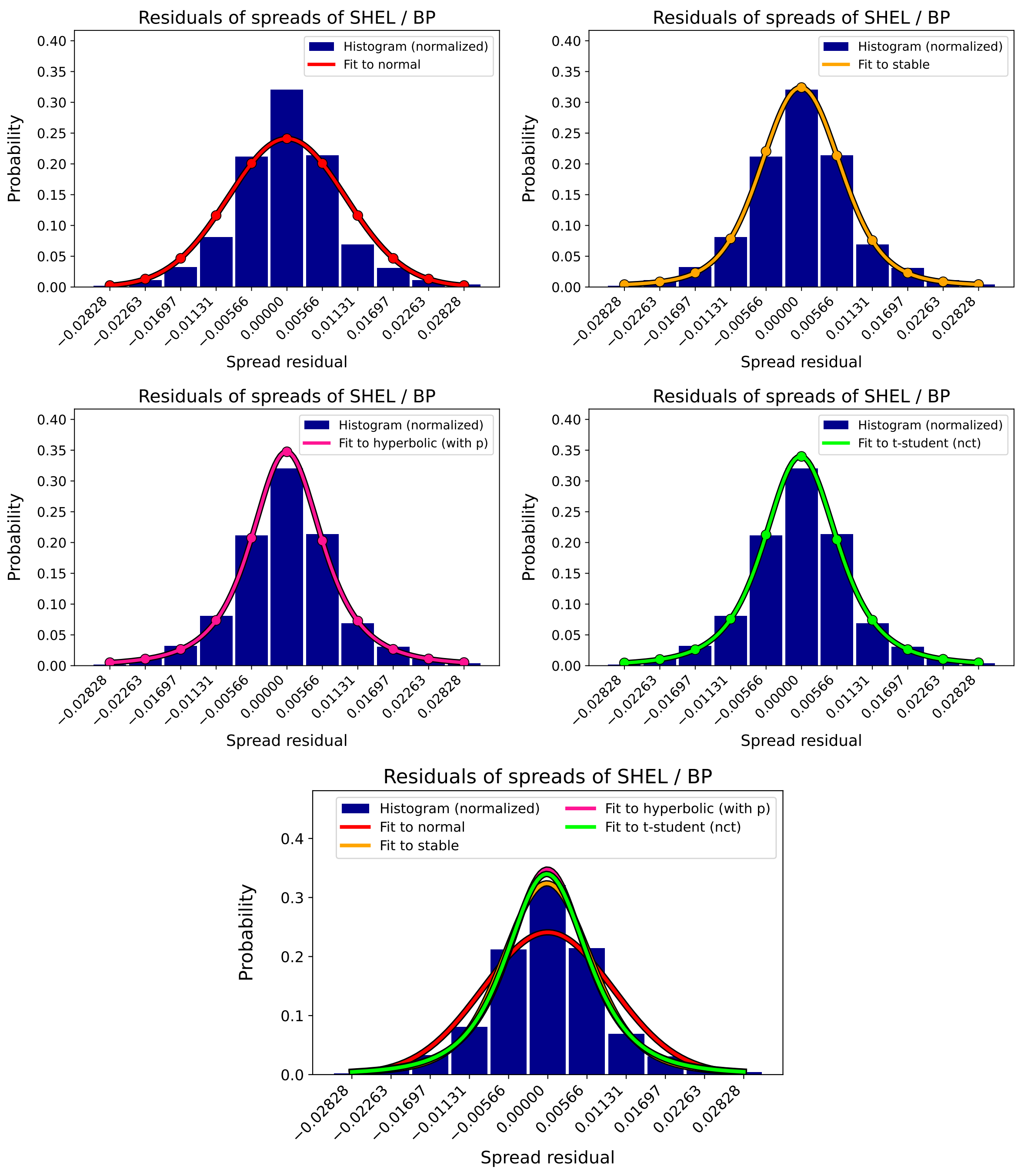


FIG. 2: Fitting of the residuals of the spread of BP p.l.c. and Shell p.l.c. in the Ornstein-Uhlenbeck equation to normal, stable, generalized hyperbolic and t-student distributions.

VI. FINDING OPTIMAL TRADING RULES

In this section we present the block which deals on calculating optimal trading rules using the Monte Carlo method. This block can be run without further calculations if one uses parameters for the simulated system which are arbitrary, i.e. not derived from observed pricing time-series. To do so, set `path_rv_params` to None (or leave it unset). In that case, you must set what follows:

- Parameters of the Ornstein-Uhlenbeck equation (E_0, φ) through `list_E0`, `list_tau` (note that tau is defined by $\tau := -1/\log_2[\varphi]$).

- Parameters of the random variable to simulate (returns for single-product trading, residuals for pairs trading). If the distribution chosen to simulate them is normal, then it suffices with specifying the location and scaling parameters (`list_mu` –which should be [0]–, `list_sigma`). For the fat-tailed distributions one needs to also specify the list of skewness and tail parameters (`list_3rdparam`, `list_4thparam`, respectively). If the distribution chosen is the generalized hyperbolic, then one also needs to specify the p parameter through `list_5thparam`.

On the other hand, the parameters of the random variable (and, eventually, also of the Ornstein-Uhlenbeck equation) can be chosen to be the ones corresponding to actually observed time-series. In that case, you must set `path_rv_params` to the path of the text file which stores those derived values, or simply set it to `path_rv_params`='default'. In the latter case, the code will search that file in a given standard location given by the names of the variables (this is the recommended option).

Regardless of whether the parameters of the random variable are arbitrary or derived from observed time-series, the user must specify the thresholds which he or she wants to analyse. There are four types of thresholds (*trading rules*) which have to be specified:

- Maximum horizon, through `list_max_horizon`.

- Enter values, through `list_enter_value`.

- Profit-taking, through `list_profit_taking`.

- Stop-loss, through `list_stop_loss`.

Further explanations on these parameters and their values can be found at the Glossary (sec. VIII).

The user can also specify if the analysis must be performed for grids of the input thresholds or if an optimal solution (a minimum of the loss function) must be calculated. This is set through the `output_type` parameter, setting it to `''heat-map''` or to `optimal_solution`. We recommend the former, because the loss functions tend to present plateaus and local minima (see article), and therefore heatmaps provide clearer insights. Examples of plotted heatmaps can be viewed in Figs. 3 and 4. Note that in the heatmaps the displayed enter value is presented with respect to E_0; for example, if $E_0 = 1.1$, then the enter value equal to 0.04 displayed in the axis actually corresponds to 1.14. In addition, the profit-taking and stop-loss values are presented with respect to the enter value.

The user can also set the `strategy_duration` parameter, which determines how many units of time (e.g. days) each of the simulated Monte Carlo paths has. Note that the 'unit of time' is by definition the spacing between consecutive data in the input (downloaded) time-series.

The number of Monte Carlo paths that are analysed for every value of the set of trading rules is set by `min_num_trials`. This quantity is central for the accuracy of the conclusions, but making it high also implies high numerical complexity. An appropriate number (at least for nct) is commonly 20,000 (see the Supporting Information of the paper). However, appropriate convergence tests on this variable should be carried out for every specific calculation.

The manner the profits are calculated is set by the `method_for_calculation_of_profits`, which we recommend to set to `enter_random_many_deals` (see Glossary for further details).

VII. OUT-OF-SAMPLE CALCULATIONS

The out-of-sample calculations calculate the profits measured in currency units (e.g. USD) of derived trading rules. To this end, the observed time-series of prices are used. Note that this way to proceed is prone to backtest overfitting, and hence it should be considered just an orientation. This block of calculations uses downloaded time series, fits them to one (just one per calculation) distribution, and uses such distribution to calculate optimal trading rules. Therefore, these calculations require having set the parameters of the three blocks presented so far. Just nine parameters correspond to this block. They are:

- `list_product_labels_name`; this is the name of the list of products, which must be

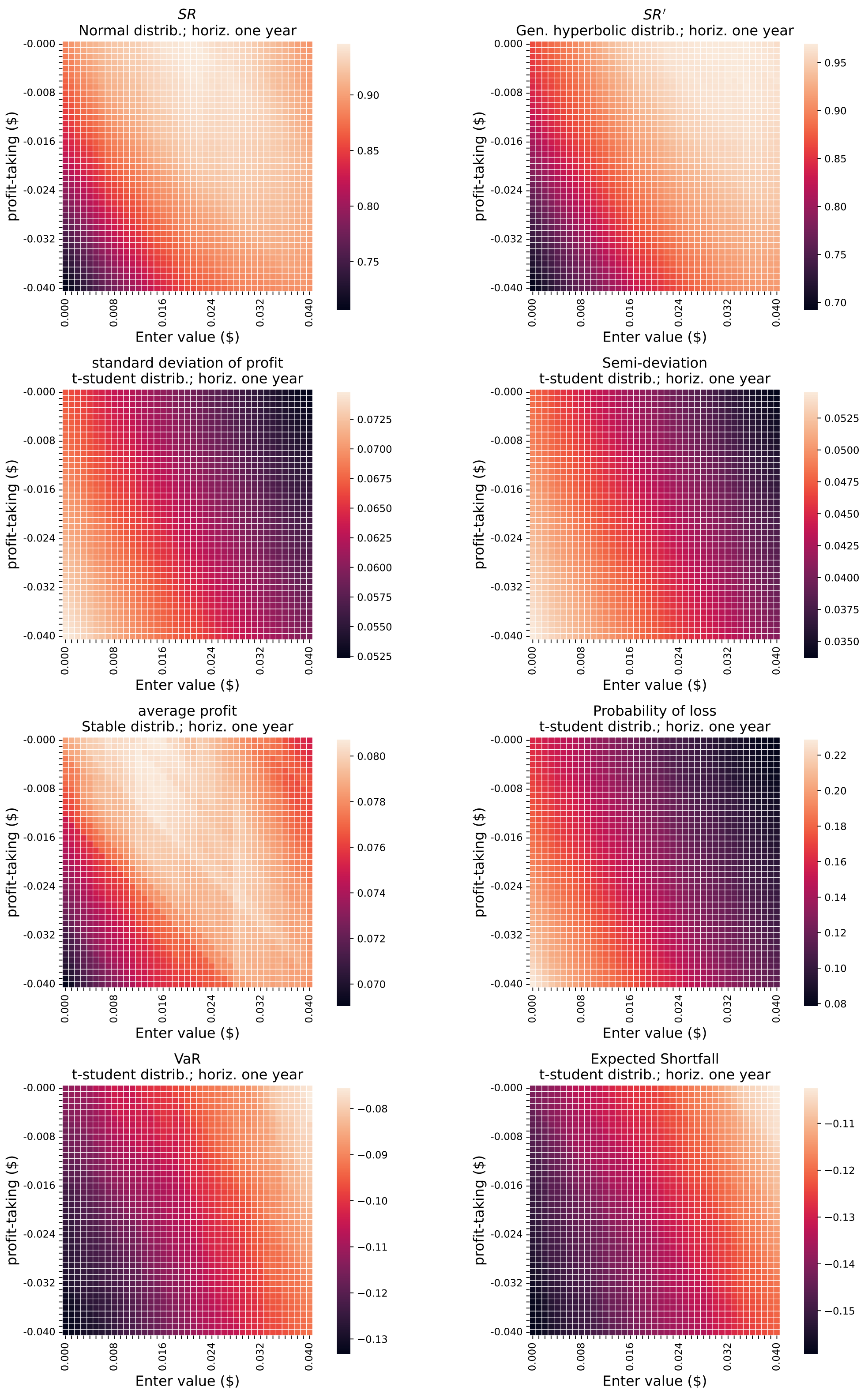


FIG. 3: Example of heatmaps of measures of performance and risk of the trading rules calculated by August for several distributions (GS/MS spread).

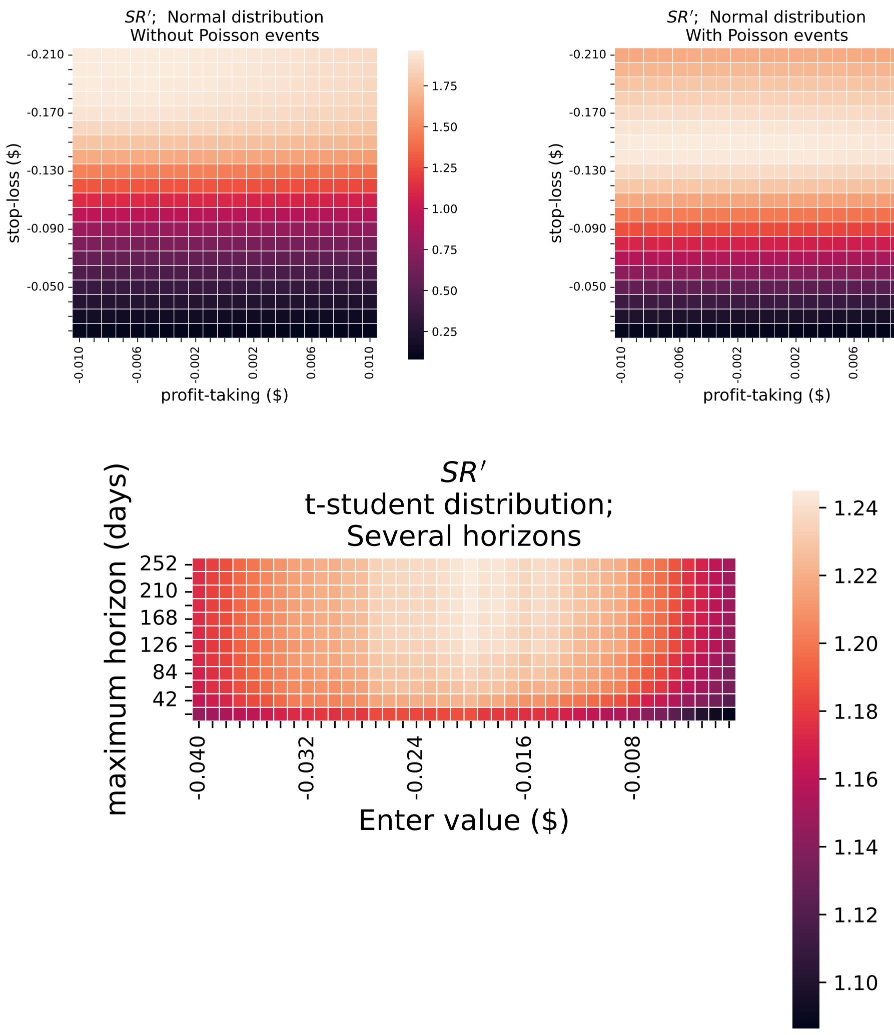


FIG. 4: Example of heatmaps (rescaled Sharpe ratio from semideviation) of the trading rules calculated by August. Top: BP/TTE spread with and without Poisson events (for a given enter value, $E_0 - 0.01$). Bottom: SHEL/BP for several time horizons (where local maxima are noticed).

the one specified in `list_product_labels` when the download was performed.

- `oos_product_label_x` and `oos_product_label_y` are the labels of the two products (pair) whose prices will take part in the out-of-sample calculation.

- `oos_spread_type` indicates whether the spread is calculated regressing the variable x vs y or the converse.

- `first_oos_date` and `last_oos_date` are the first and last dates to be considered in the out-of-sample calculation; this is, the calculated profits will be those that would be realized in the period comprised between both dates.

- `in_sample_t_interval` is the length of the in-sample period, this is the period whose data is used to do the fitting of the random variable, which will then be used to determine the optimal trading rules. The in-sample period ends immediately before the `first_oos_date`.

- `in_sample_recalibration_freq` is the frequency for recalculating the parameters of the random variable and their corresponding trading rules. Optimally this should be done as frequently as possible (e.g. every day). However, since the calculations are numerically heavy, longer recalibration periods may be necessary (e.g. 3 months), depending on the available computing capabilities.

- `quantity_to_analyse` this determines the criterion to set the optimal trading rules. If you set it to `Sharpe_ratio`, then the derived trading rules will be those which maximize the Sharpe ratio; if you set it to `semideviation` then the trading rules will be those which minimize the semideviation, etc.

Note that the parameters set in the block for finding optimal trading rules does also apply for the out-of-sample calculation. Specifically, note that in the out-of-sample calculation one needs to specify both positive and negative enter values. This can be done for example as follows: `list_enter_value = array( list( arange(-0.0400000001, 0., 00.001) ) + list( arange(0.00000001, 0.04000002, 0.001) ) )` .

VIII. GLOSSARY OF ALL INPUT PARAMETERS

Below we present the comprehensive list of all the input parameters that the user can set. Not all of them are necessary; many of them are either not used in some calculations or assigned default values by August. The list does not appear in alphabetical order, but in *chronological order*, this is showing together all the parameters for a given calculations block, and showing these blocks in the order that they must be run.

`calculation_mode`

Block: General. **Possible values:** ”download”, ”fit”, ”download_and_fit”, ”find_trading_rules” or ”out_of_sample”. **Meaning:** Kind of calculations to be performed; ”download” downloads the data from Yahoo Finance; ”fit” fits the data to probability distributions (and eventually to the Ornstein-Uhlenbeck equation), ”download_and_fit” makes both; ”find_trading_rules” calculates the optimal trading rules; ”out_of_sample” calculates the profitability in specified periods.

`evolution_type`

Block: General. **Possible values:** ”Ornstein-Uhlenbeck_equation” (for pairs trading) or ”single_product” (for single-product trading). **Meaning:** This indicates the type of trading that will be performed, either of long-short pairs or long on a single financial product.

`list_distribution_types`

Block: General. **Possible values:** [’norm’,’nct’,’genhyperbolic’,’levy_stable’] or subsets of this list. **Meaning:** Types of the probability distributions to fit the random variables.

`list_product_labels`

Block: General. **Possible values:** It must be a list whose first element is the list name (an arbitrary string) and whose second element is another list which contains at least two identifiers searchable in Yahoo Finance. For example: [”oil_companies”, [”BP”,”SHEL”]]. **Meaning:** Identifiers of the time series to download.

`verbose`

Block: General. **Possible values:** 0, 1 or 2. **Meaning:** Flag which determines how much information of the intermediate calculations is displayed to screen (irrelevant for the final results).

`make_plots`

Block: General. **Possible values:** True or False. **Meaning:** This determines whether plots must be done in the execution. If set to False, the plots can be swiftly done later by setting only_plots to True.

`only_plots`

Block: General. **Possible values:** True or False. **Meaning:** True if the execution of the code must only do plots, instead of calculating numerical output.

`consider_skewness`

Block: Downloading and fitting. **Possible values:** True or False. **Meaning:** If True, the random variables are fitted to distributions where a nonzero skewness parameter is possible.

`consider_p`

Block: Downloading and fitting. **Possible values:** True or False. **Meaning:** If True, when fitting to generalized hyperbolic ("genhyperbolic") distributions, then the p parameter is allowed to be nonzero.

`efficient_fit_to_levy_stable`

Block: Downloading and fitting. **Possible values:** True or False. **Meaning:** If True or unset, then the fitting to stable distributions is automatically performed with n_random_trials=1 and max_n_iter=12.

`n_random_trials`

Block: Downloading and fitting. **Possible values:** A natural number (e.g. 10). **Meaning:** Number of times that each set of starting parameters in the fitting of the random variable is tried. Each trial corresponds to a different (random) first iteration of the gradient descent algorithm. It is recommended to set it to 1 or 2 for fittings to the levy_stable distribution, because they are numerically heavy (though this is unnecessary if efficient_fit_to_levy_stable is set to True or unset).

`max_n_iter`

Block: Downloading and fitting. **Possible values:** A natural number (e.g. 20). **Meaning:** Maximum number of iterations in each search for optimal parameters (using the gradient descent algorithm, with a given starting value for the distribution parameters). For fittings of the levy_stable distribution it is recommended to set it to 12 for efficiency reasons (though this is unnecessary if efficient_fit_to_levy_stable is set to True or unset).

`downloaded_data_freq`

Block: Downloading and fitting. **Possible values:** '1d' for one day; '5m' for five minutes. **Meaning:** Frequency of the data to download. Yahoo Finance makes it possible to download data with frequency higher to one day for recent data.

`first_downloaded_data_date`

Block: Downloading and fitting. **Possible values:** String in the format 'YYYY-MM-DD', e.g. '2010-11-30'. **Meaning:** First date of the data to be downloaded.

`first_downloaded_data_date`

Block: Downloading and fitting. **Possible values:** String in the format 'YYYY-MM-DD', e.g. '2022-11-30'. **Meaning:** Last date of the data to be downloaded.

`correlation_threshold`

Block: Downloading and fitting. **Possible values:** Real number between -1 and 1, e.g. 0.5. **Meaning:** Lower limit to the correlation to discard a pair in an analysis for spread.

`output_type`

Block: Seek for optimal trading rules. **Possible values:** "heat-map" (default) or "optimal_solution" (which is experimental and not recommended, because the outputs have local minima). **Meaning:** Kind of quantity which is calculated.

`method_optimal_solution`

Block: Seek for optimal trading rules. **Possible values:** "Barzilai-Borwein" or "bisection". **Meaning:** Just for runs with output_type="optimal_solution". It determines the method for calculating the optimal solution (either bisection or gradient descent with Barzilai-Borwein step).

`path_rv_params`

Block: Seek for optimal trading rules. **Possible values:** Either None, 'default' or a string specifying the path to the file which stores the parameters of the random variable to use in the seek of optimal trading rules. Set to 'default' to use the data calculated in previous steps (fitting of residuals from real data to chosen probability distributions). Set to None to specify the parameters of the random variables in input.py (through `list_enter_value`, `list_profit_taking`, `list_stop_loss`). **Meaning:** Path to the file which contains the parameters of the Ornstein-Uhlenbeck equation and its residuals.

`poisson_probability`

Block: Seek for optimal trading rules. **Possible values:** (Optional). Either None (for no Poisson events) or a real number between 0 and 1. **Meaning:** Probability of a Poisson event. Set to None for no Poisson events. The Poisson events determine the mean of the mean-reverting stochastic process if evolution_type is ”Ornstein-Uhlenbeck”; if evolution_type is ”single_product”, then a Poisson event sets the price to a predetermined value.

`new_value_after_poisson_event`

Block: Seek for optimal trading rules. **Possible values:** A real positive number. **Meaning:** If poisson_probability is not None or 0 and evolution_type is ”single_product”, then this is the value that the price takes if the Poisson event happens.

`new_value_after_poisson_event_increase`

Block: Seek for optimal trading rules. **Possible values:** A real positive number (e.g. 0.50). **Meaning:** If poisson_probability is not None or 0 and evolution_type is ”Ornstein-Uhlenbeck”, then this is the value that increases the mean (E_0) if a Poisson event happens (with 50% probability). For example, if $E_0 = 1.2$ and a Poisson event happens, if this variable is 0.6 then there is a 50% likelihood that E_0 will become 1.92.

`new_value_after_poisson_event_decrease`

Block: Seek for optimal trading rules. **Possible values:** A real negative number (e.g. -0.333). **Meaning:** If poisson_probability is not None or 0 and evolution_type is ”Ornstein-Uhlenbeck”, then this is the value that decreases the mean (E_0) if a Poisson event happens (with 50% probability). For example, if $E_0 = 1.2$ and a Poisson event happens, if this variable is -0.33333 then there is a 50% likelihood that E_0 will become 0.8.

`list_max_horizon`

Block: Seek for optimal trading rules. **Possible values:** List of integer numbers which are the maximum horizons to be tested (e.g. [252] or [7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91]). Set to e.g. [9999999] if you want not to be limited by a time horizon. **Meaning:** List of horizons to be tested in the trading rules; a position will be unwinded when the maximum horizon is reached, even if no profit-taking or stop-loss thresholds were reached.

`list_enter_value`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers, e.g. `arange(-0.04, 0.00000001, 0.001)`. **Meaning:** (Optional; just used if path_rv_params is None). It specifies the enter values whose performance is tested using Monte Carlo calculations.

`list_profit_taking`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers, e.g. `arange(0.0, 0.040000001, 0.001)`. **Meaning:** (Optional; just used if path_rv_params is None). It specifies the profit-taking thresholds whose performance is tested using Monte Carlo calculations.

`list_stop_loss`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers, e.g. `array([-9999])`. **Meaning:** (Optional; just used if path_rv_params is None). It specifies the stop-loss thresholds whose performance is tested using Monte Carlo calculations.

`strategy_duration`

Block: Seek for optimal trading rules. **Possible values:** Positive integer (e.g. 252). **Meaning:** Number of days that the strategy is expected to be used; in practice this means that its return is evaluated in one period with this duration (in each iteration of the Monte Carlo algorithm). Do not confuse with list_maximum_horizon: The maximum horizon is the maximum time that a portfolio is kept counting the time since its inception (i.e. since entering it); strategy_duration is the total period (e.g. number of days) that are simulated in each Monte Carlo path. Note that if strategy_duration is different from 252 (one year) then the outputed profits are annualized by multiplying them by strategy_duration/252.

`min_num_trials`

Block: Seek for optimal trading rules. **Possible values:** Positive integer (e.g. 20000). **Meaning:** Number of Monte Carlo path generated in the evaluation of each set of trading rules. This quantity is central for the numerical complexity of the calculations and for their accuracy. Set it carefully.

`discount_rate`

Block: Seek for optimal trading rules, out-of-sample. **Possible values:** A real positive number. **Meaning:** (Optional; if absent or set to None or to zero then discount rate will be zero, and hence all discount factors will be 1, i.e. no discount factor will be applied). Rate to be used for continuous compounding (exponential) discount factors; this is, between dates t_1 and t_2 the discount factor will be exp$[-(\text{discount_rate}) \cdot (t_2 - t_1)/365]$. This can give account of risk-free rates, or simply of a utility function which prioritizes earnings in the nearby future. Set e.g. to 0.02 for a discount rate of 2%.

`transaction_costs_rate`

Block: Seek for optimal trading rules, out-of-sample. **Possible values:** A real positive number. **Meaning:** (Optional; if absent or set to None or to zero then transaction costs rate will be zero and hence no transaction costs will be applied). Rate to be used to give account of the transaction costs. This corresponds to r_s in the main article[2], find further explanations there. Set e.g. to 0.02 for a discount rate of 2%.

`method_for_calculation_of_profits`

Block: Seek for optimal trading rules. **Possible values:** Either "enter_ensured", "enter_random_one_deal" or "enter_random_many_deals". **Meaning:** Method for the calculation of the profit; "enter_ensured" is the method presented in Chapter 13 of Marcos Lopez de Prado's "Advances in Financial machine learning". In it, the position is 100% sure built in t=0, and after reaching a threshold the process starts again. "enter_random_one_deal" means that the initial value of the spread is its average (for Ornstein-Uhlenbeck), then it randomly varies until the prescribed enter level is reached; after reaching a threshold the process starts again. "enter_random_many_deals" means that the initial value of the spread is its average (for Ornstein-Uhlenbeck), then it randomly varies until the prescribed enter level is reached; after this, the spread continues to evolve, and we can build new positions, whose return is added to make a cumulative return. The latter is the method used in the

[2] `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4518354`

2023 paper ”The effect of fat tails on rules for optimal pairs trading of stocks and cryptocurrencies based on the Ornstein-Uhlenbeck equation”.

The 7 lists below correspond to simulations with arbitrary (i.e. not derived from observed time-series) parameters for the random variables. This is, they (all or part of them) must be specified if `part_rv_parms` is unset or None. The first two parameters must be specified only if calculation_mode is ’Ornstein-Uhlenbeck_equation”; in that case, the last five parameters correspond to the random variable which is the spread residuals. Othewise, the last five parameters correspond to the results of the single traded product.’

`list_E0`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers, e.g. `[0]`. **Meaning:** List of values of E_0 (mean of the mean-reverting process behind the Ornstein-Uhlenbeck equation) to test.

`list_tau`

Block: Seek for optimal trading rules. **Possible values:** List of positive real numbers, e.g. `[25]`. **Meaning:** List of values of τ (velocity of the mean-reverting process behind the Ornstein-Uhlenbeck equation) to test. Note that $\tau := -1/\log_2[\varphi]$.

`list_mu`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers, e.g. `[0]`. **Meaning:** List of values of the location parameter to test.

`list_sigma`

Block: Seek for optimal trading rules. **Possible values:** List of positive real numbers, e.g. `[0.001, 0.002]`. **Meaning:** List of values of the scaling parameter to test.

`list_3rdparam`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers, e.g. `[0]`. **Meaning:** List of values of the skewness parameter to test.

`list_4thparam`

Block: Seek for optimal trading rules. **Possible values:** List of real numbers. Their appropriate values depend on the tested distribution. For example, if it is nct, then these numbers must be real positive numbers (though very low or very high numbers lead to problems with the scipy library); if the distribution is levy stable this corresponds to the a parameter, which must be between 0 and 2. **Meaning:** List of values of the tail parameter to test.

`list_5thparam`

Block: Seek for optimal trading rules. **Possible values:** (Just if the tested distribution is genhyperbolic). List of real numbers. **Meaning:** List of values of the p parameter to test.

`quantity_to_analyse`

Block: Seek for optimal trading rules / out-of-sample. **Possible values:** Either ”Sharpe_ratio”, ”average”, ”standard_deviation”, ”semideviation”, ”Sharpe_ratio_with_semideviation”, ”VaR”, ”ES”, or ”probab_loss”. **Meaning:** This determines which quantity is optimized if calculation_mode is ”out-of-sample” or if calculation_mode is ”find_trading_rules” and output_type=”optimal_solution”.

`oos_spread_type`

Block: Out-of-sample profit calculations. **Possible values:** Either ”x_vs_y” or ”y_vs_x”. **Meaning:** This parameter defines how the Spread is defined. If it is ”y_vs_x” then the Spread is log(Price_label_y) - γlog(Price_label_x), where γ is the slope of the Return_label_y-vs-Return_label_x. If it is ”x_vs_y”, then the prices and returns are swapped. Which label actually corresponds to x and y is specified by oos_product_label_x, oos_product_label_y.

`oos_product_label_x`

Block: Out-of-sample profit calculations. **Possible values:** String searchable in Yahoo Finance (e.g. ”BMW.DE”). **Meaning:** X label of the product whose price will take part in the out-of-sample calculation.

`oos_product_label_y`

Block: Out-of-sample profit calculations. **Possible values:** String searchable in Yahoo Finance (e.g. "MBG.DE"). **Meaning:** Y label of the product whose price will take part in the out-of-sample calculation.

`list_product_labels_name`

Block: Out-of-sample profit calculations. **Possible values:** String (e.g."car_manufacturers"). **Meaning:** Name of the list of products; it must be the one specified in list_product_labels when the download was performed.

`first_oos_date`

Block: Out-of-sample profit calculations. **Possible values:** String in the format 'YYYY-MM-DD', e.g. = '2018-11-30'. **Meaning:** First date for the out-of-sample calculation.

`last_oos_date`

Block: Out-of-sample profit calculations. **Possible values:** String in the format 'YYYY-MM-DD', e.g. = '2022-11-30'. **Meaning:** Last date for the out-of-sample calculation.

`in_sample_t_interval`

Block: Out-of-sample profit calculations. **Possible values:** String including either 'y', 'm' or 'd' (for years, months or days, respectively). E.g. = '5y' for five years. **Meaning:** Time interval prior to the out-of-sample period which forms the in-sample period (i.e. whose information is used to calibrate the random variables and to set the trading rules). Set e.g. to '100d' for 100 trading days or '2y' for two years.

`in_sample_recalibration_freq`

Block: Out-of-sample profit calculations. **Possible values:** String including either 'y', 'm' or 'd' (for years, months or days, respectively). E.g. = '3m' for one quarter. **Meaning:** Frequency for the recalibration of the parameters of the input variable (i.e. Ornstein-Uhlenbeck parameters and parameters of the random variable corresponding to its residuals) and recalculating the optimal trading rules.

DISCLOSURE STATEMENT

PR states that he had a job in a banking company during the time this research work was performed, and that it has been carried out exclusively in his non-working time and using his own resources, i.e. it is totally independent from his labor activities.

AUTHOR INFORMATION

Emain address: risueno@unizar.es, garcia.risueno@gmail.com;
https://www.linkedin.com/in/pablo-risue%C3%B1o-phd-487055160/
ORCID: 0000-0002-8142-9196.