DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

Orla: A data flow programming system for very large time series

Gilles Zumbach* **and Adrian Trapletti***

Abstract

To analyze tick-by-tick financial time series, programs are needed which are able to handle several millions of data points. For this purpose we have developed a data flow programming framework called "Orla".[1] The basic processing unit in Orla is a "block", and blocks are connected to form a "network". During execution, the "data" flow through the network and are processed as they pass through each block. The main advantages of Orla are that there is no limit to the size of the data sets, and that the same program works both with historical data and in real time mode. In order to tame the diversity of financial time series, the Orla data structure is specified through a BNF description called SQDADL, and the Orla data are expressions in this language. For storage, the times series are written in a "tick warehouse" which is configured completely by the SQDADL description. Queries to the tick warehouse are SQDADL expressions and the repository returns the matching time series. In this way, we achieve a seamless integration between storage and processing, including real time mode. Currently, our tick warehouse contains 20'000 "elementary" time series. In this paper, we provide a brief overview of Orla and present a few examples of actual statistical analysis computed with Orla.

*Olsen & Associates, Research Institute for Applied Economics, Seefeldstrasse 233, 8008 Zürich, Switzerland. e-mail: firstname@olsen.ch

[1] Orla and the tick warehouse are the result of a company wide development over several years, involving many individuals. The main contributers are: David Beck, Devon Bowen, Dan Dragomirescu, Paul Giotta, Loic Jaouen, Martin Lichtin, Roland Loser, Kris Meissner, James Shaw, Leena Tirkkonen, Robert Ward, and Gilles Zumbach.

1 Introduction

The basic idea of Orla can be summarized as "data flowing through a network of blocks" and is nicely illustrated by looking at the example given in figure 1. Every Orla application can be represented by a network which in turn consists of interconnected blocks. Each block performs a computation on the data flowing through it, and the data are flowing between blocks along the connections.

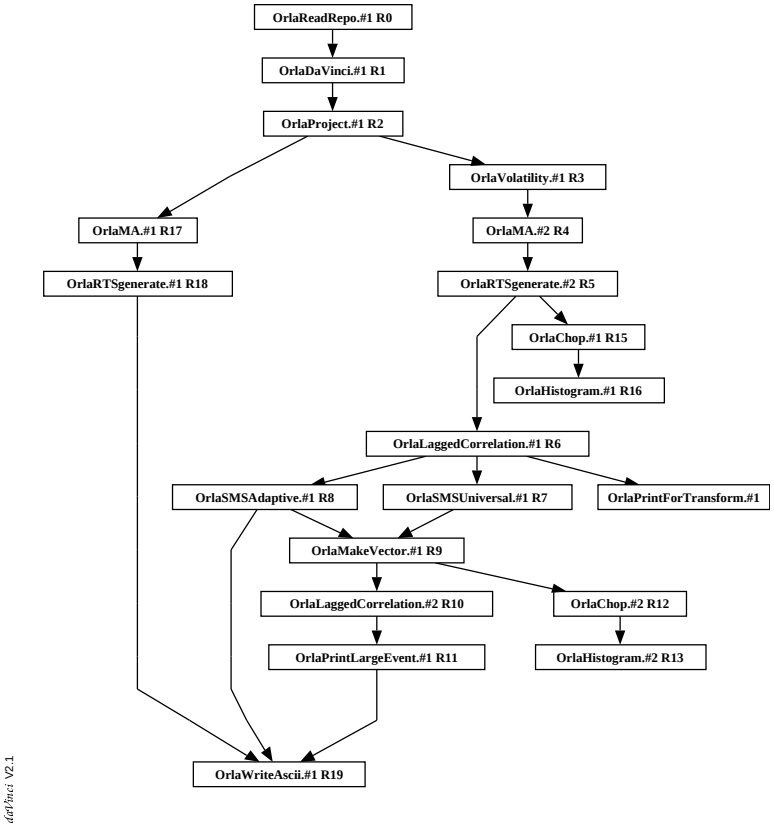


Figure 1: An example of an Orla network.

The main concepts used in Orla are therefore the datum, the block, the connection and the network:

- **Datum** (pl. Data): the unit of information flowing in the network.
  Since we work with financial time series, the diversity of financial assets needs to be represented in the datum. A datum can be as simple as a price for a spot foreign exchange rate or as complicated as a forecast for an interest rate yield curve. Because the data are not simple pairs of time stamps and values, but are instead highly structured, a type is associated with each kind of data. Datum structures and types are further explained in section 4.

- **Block:** the basic processing unit.
  A block receives data through a number of input ports, processes the data according to some algorithm, and sends the data further down through its output ports. The philosophy behind blocks is to implement at the block level fairly elementary functionality and to obtain more complex algorithms through a chain or network of blocks.

- **Connection:** the link between blocks.
  A connection establishes a path for the data flow between output ports and input ports of different blocks. Several connections may start from the same output port, but only one connection must arrive at an input port (this is required for a meaningful time ordering). The flow through a connection is always a "valid" time series, i.e., a time ordered series of data with an associated type.

- **Network:** the set of blocks, connections and data.
  When a network is evaluated, the data flows through the network. Results are written either continuously during the computation (e.g., when recording a time trace of some computed time series quantities), or at the end of the computation (e.g., when evaluating a global average).

The block structure of Orla imposes a "local" processing model. This has mainly two implications: First, there are no other dependencies between blocks than the connections in the network. A block communicates only through its ports on which it receives and sends valid time series. Second, each datum carries all the needed information for its processing by the blocks. For example, as some algorithms can be implemented much more efficiently for regularly spaced time series, the typing system of the data includes the "kind" of the time series (irregular or regular with a given time interval between two subsequent data points), and the blocks can use this information to optimize their computations. The block structure of Orla is similar to a "Lego" world, and allows to achieve an optimal code reuse.

To summarize, Orla is a **data-flow** programming system. Its purpose is to:

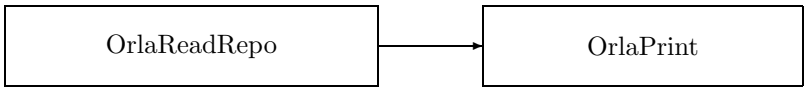
- analyze **unevenly spaced** time series,
- process data sets of **unlimited** size, and
- work both in **historical** mode and in **real time**.

The first point is mainly a question of algorithms used for the actual statistical computations. Most empirical studies work with discretely and equally spaced time series. However, simple and efficient algorithms exist for the analysis of irregularly spaced time series and are presented for example in [1]. To support this type of analysis, the time stamp in Orla is not just an index but a meaningful time that is also used for timing and synchronization issues. The second and third points in the above list are byproducts of the data flow paradigm. Because the time series are not loaded into main memory, Orla is not limited with respect to the size of the data sets. We are working routinely with time series containing 10^5 to 10^7 data

points, possibly computing vector valued time series with 50 variables per vector. On the other hand, this implies that "incremental" algorithms have to be used as for example the computation of the mean, but not of the median (which would essentially require to order the full time series). Furthermore, because Orla is data driven, it also works in real time for the delivery of value added information. A key advantage here is that the same code can be used both for research using historical data and for the real time delivery of products. This avoids the reimplementation and testing of software already used and tested in historical mode. Hence, for us Orla is **the** platform for economic research (in particular, for time series analysis) and for the delivery of real time information. Moreover, Orla is also very flexible with respect to the type of time series which can be analyzed. In principle, any time series can be analyzed as long as the particular data structure is appropriately specified (e.g., Orla could be used to analyze weather data, industry processing data, electricity network data, etc.).

2 Working with Orla

Concretely, Orla is written in C++ and appears as a set of classes and libraries. The blocks are grouped into libraries according to their functionality, and the user needs to create the desired instances with the corresponding parameters. Presently, the user needs only to write a main program which can be compiled and linked as usual. [2] An elementary example that just read and write data is as follows:



The corresponding C++ code is

```
// create a network
OrlaNetwork network( "A simple network" );

// create a block which gets data from the repository
OrlaReadRepo in( "FX(USD,ATS),Quote(,,),Source(,,),Filter(,,)" );

// create a block which prints the data
OrlaPrint out( cout );

// bind together the network and the blocks
network >> in >> out;

// and run the network
network.run("01.01.2001",  "31.01.2001");
```

[2] In principle, it would be easy to put a GUI interface on top of Orla where for example blocks are created from pull down menus. However, the current implementation does not support such a GUI interface.

The last statement triggers the run time engine of Orla that will let the data with time stamps in January 2001 flow through the network.

More generally, the user writes a high-level C++ program to perform the desired task. This consists of statements creating an empty network, constructing and binding the blocks, and running the network. All the "magic" happens when running the network. During execution, the blocks are activated by the network (more precisely by the scheduler contained in the network) by two kinds of events:

- The data: these events depend on the inputed time series and the blocks upstream. They are essentially "random" as a block has no control over its inputed data.
- A timer: scheduled events according to the physical clock, e.g., an event every day at 16:00. Timers are used for example to create a regularly spaced time series, say the number of ticks within the last day. These events are essentially "deterministic" as a block itself must create the timer according to its behavior. Timers are also used in real time to implement a "grace" time interval (see below).

The scheduler guarantees that every block gets the events time ordered. In this way, the block writer does not have to care about scheduling issues. It should also be emphasized that the scheduling has to be time-wise efficient, particularly because the granularity of the tasks performed by the blocks is small. In practical tests with networks which perform mildly computationally intensive tasks, the scheduling overhead is below 5% of the overall used CPU time. Another issue which we do not further discuss is the strategy of the block scheduling in a network in order to avoid large queues of data on input ports leading to an excessive memory consumption. Essentially, this is controlled by "following" the data, i.e., by activating the blocks downward of the last produced data (deep first strategy).

3 How Orla is working

The complete evaluation of a network is a succession of tasks:

- Check the sanity of the network, mainly data type checking and checking the number of connections. This is done by each block by checking that it receives the appropriate data type on each port. Possibly, blocks can grab memory according to the number of ports or the length of the received data vector and set timers.
- Activate the producer blocks. This will start the flow of data.
- Process incoming data or timer events for all the blocks.
- Possibly, switch to real time mode.
- Possibly, process and forward end-of-data conditions.

- Let the network finish once everything is idle, namely return from the run method.

The timing paradigm is to start networks in historical-time processing mode, and to switch to real-time when caught up with the wall clock. This allows blocks to build up their internal states with historical data, and then to switch to real time. The changes of processing mode (historical → real-time, processing → end-of-data) are trickling downward the network, similarly to the data. The important point is that the scheduling is done differently in historical and real time mode. For historical time, the scheduling is data and timer driven and the wall clock is irrelevant. This corresponds to a simple ordering of the events (for every block), essentially because the incoming events are known. In real time, the data are processed as they are coming, while the timers are handled according to the wall clock. Moreover, we want to have identical results if the same data are processed in real time or historical mode; this is essential to have reproducibility of the real time computations. A subtle point arising in real time is to add a bit of "slack" in the scheduling. We are working in a distributed environment, where the data are received, filtered, and stored on other computers in the local network. The time stamps put on the data are set by the first computer that receives them from the external source (e.g., from the data provider Reuters). Then, there are delays in the processing upstream of an Orla network, while the data are filtered, stored and forwarded inside the local network. The end result is that a datum with a given time stamp is actually received a few milliseconds to a few seconds later in a running Orla network. A clear scheduling problem appears with timers, e.g., if a datum has a time stamp of 11:59:59, a timer is set at 12:00:00 and this datum is received by the network only at 12:00:02. For this reason, a "grace" time interval δt is added to the incoming events of blocks with multiple inputs (input ports and/or timers). Namely for every event (data or timer) arriving at t, a timer is set at $t + \delta t$ for the actual processing of this event. This allows to correctly time order the events occurring inside the grace time interval.

The main design criteria for Orla are:

- A high level of abstraction to hide underlying details. The users and even the block writers are unaware of the scheduling and its complexity. This makes Orla easy to learn and safe to use.

- Minimal overhead in scheduling the blocks and forwarding the data. The goal has been to have a modular structure, which is efficient.

- A versatile datum structure. This topic deserves a full section, see section 4.

- Automatic "vectorization" on the time horizons. When analyzing financial time series, it is often very instructive to investigate different time horizons. An example is to analyze the price differences (returns) at increasing time intervals, say 1h, 2h, 4h, etc.. In this case, the block that computes the returns can be used with a vector of time horizons as argument in the constructor. The output of this block is then a vector containing the respective returns. This block can feed, say, a histogram block in order to compute the probability

distribution of the returns. The histogram block itself recognizes that the input is a vector and therefore computes a vector of histograms corresponding to each value in the inputed vector. In this way, computational and diagnostic blocks can be chained using scalar or vector data. This is a very powerful feature of Orla as one simple network can analyze at once a time series at various time horizons.

- Easily extensible by adding new blocks. This is important in order to get new blocks written by users that are not aware of all the foundations and subtleties of Orla.

- Only "valid values" are send. All algorithms computing moving values need to be initialized with some data. Consider the example of a 1 day moving average computation. Although a value could be issued already at the first tick, one day of data is needed to properly initialize the internal states and to provide a meaningful result. This first day during which the result is "invalid" is called the build-up period. For some algorithms, no results can be produced during the build-up of the internal states. Orla is designed to send only valid data, i.e., to send only data after the build-up time interval of the particular algorithm is elapsed. In this way, blocks are guaranteed to receive only valid data. Moreover, blocks can return their build-up time, and the network can calculate the cumulative build-up time of a network. In this way, the user can easily specify a time interval for valid values, and the network can process earlier data according to the network cumulative build-up time interval.

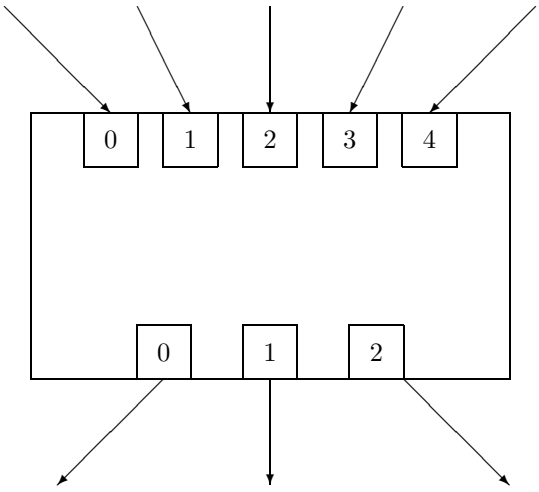


Figure 2: The general block set-up.

The block programmer's view of a general block is as in figure 2. The block has input and output ports with corresponding data types. The input types can be queried (and are provided by the block upstream of each port). On the output side, each block must provide the actual types of each output port. The blocks must provide essentially three methods corresponding to the block initialization and set-up, the processing of the data, and the end-of-data handling. As the network is taking care of all synchronization and scheduling issues, this simplifies the task of the block writer. For example, the data processing method possesses as argument a time and a vector of simultaneous data corresponding to each input ports, with a nil pointer indicating that one of the input ports has no datum at this time.

4 The data representation

Our goal is to work with financial data and with the whole variety of possible financial contracts. Financial contracts range from the simplest instruments such as spot foreign exchange rates which are characterized by a single number (the price) to complex derivatives including more "parameters" like the expiry date, the strike price, the detail of the derivative (e.g., European call option, American put option, etc.) and the characteristic of the underlying. This is a vast world which contains plenty of different instruments such as interest rates, futures, forwards, options, swaps, options on futures, and so on. Beside, depending of the computation, a time series with some fixed parameters might be desired, e.g., a 3 months interest rate on USD. Or say for another computation, all values of the same parameters are desired, e.g., in the construction of a yield curve on the USD all maturities are needed, but the currency is still fixed. Clearly, there is no general rule to fix what are the (constant) parameters and what is part of the time series (with changing values). The distinction between "parameters" and time series "values" is meaningless as this changes with the use of the time series. Therefore, the data structure needs to be flexible enough to accommodate all possible financial contracts with their particular characteristics. Potentially, all fields can be "values" or "parameters" depending on the computation. For storing the data, the same problem occurs.

In order to represent any kind of financial securities in the Orla datum, the data structure is given through a BNF description called SQDADL (SeQuential DAta Description Language). The description of the whole world of financial securities is given in approximately 6 pages of text. This simplification is made possible because of the power and recursivity of the BNF description; the recursivity is used in the description of the derivatives (a derivative depends on one or possibly several underlying contracts). A general parser of SQDADL expressions is constructed from the BNF specification. This ensures a completely generic approach without hard coding of the financial peculiarities.

SQDADL expressions can be concrete or abstract. In a concrete expression, all the fields have a value, e.g., "FX(USD, CHF)" specifies a spot foreign exchange (FX) rate between the United States dollar (USD) and the Swiss franc (CHF). In an abstract expression, some of the fields are empty with the meaning of "anything", e.g., in "FX(,)" or "FX(USD,)". Furthermore, SQDADL expressions have a "is

a" relationship, e.g., "FX(USD, CHF)" "is a" "FX(USD,)" "is a" "FX(,)". This powerful mechanism is used for type checking in Orla.

For data storage and retrieval the same technique is applied. We have developed a **tick warehouse** as a special purpose financial time series repository. The tick warehouse is configured completely with the SQDADL description, while the internal organization of the tick warehouse is hidden to the users. The abstract view of the tick warehouse is that of a large store which understands SQDADL requests. For storage, the data are "thrown" in the repository. The queries to retrieve data are abstract SQDADL expressions, and the repository returns the matching time series of concrete SQDADL ticks. In this way, we have a seamless integration between the tick warehouse and Orla. The internal organization of the tick warehouse is optimized for time series in order to give a high throughput for the data request. The tick store works also both in historical mode and in real time. A general request contains a start and end time for the time series; no end time means that the request will continue with real time data. Currently, we are storing in the repository 20'000 time series (or 120'000 with a more stringent definition of "a" time series) with an increment of 10^6 ticks per day.

5 A few example computations done with Orla

In this section, we give four examples of statistical computations carried out with Orla. The data used are USD/CHF foreign exchange data from 1.1.1990 to 1.1.2000 and 1.1.2001, respectively.

Figure 3 shows an intra-week analysis of the volatility, namely a 30 minutes volatility is computed followed by an average conditional to the time in the week. Moreover, only the data during winter time are used, i.e., when Europe and the US have no daylight saving time (DST). The figure shows very clearly the weekly cycle of human activity with almost no activity during the week-end. Moreover, the volatility during each day is the signature of the different open markets: first the Japanese (and Asiatic) market with the very sharply defined opening at 0h and the lunch break in Tokyo at around 4h. Then, the opening of the European market occurs at around 7h followed by a weak lunch break. The sharp rise at around 13h corresponds to the opening of the American market, followed by the successive closing of the European and American market. This cycle is repeated over the five working days.

Figure 4 displays the correlation between the historical volatility $\sigma[\Delta t](t)$ and the realized volatility $\sigma[\Delta t'](t + \Delta t')$. The historical volatility is computed with data in the past of t over a time interval of Δt. The realized volatility corresponds to the future volatility and is computed using data in the future of t over a time interval of $\Delta t'$. Hence, the correlation between these two quantities is a measure of the information contained in the historical volatility about the realized volatility and can be used to forecast realized volatility. This computation has been done using the vectorization capabilities on a range of time horizons. The figure shows that fairly large correlations are present, and that realized volatility at a given time horizon $\Delta t'$ has the largest correlation with historical volatility with a slightly larger

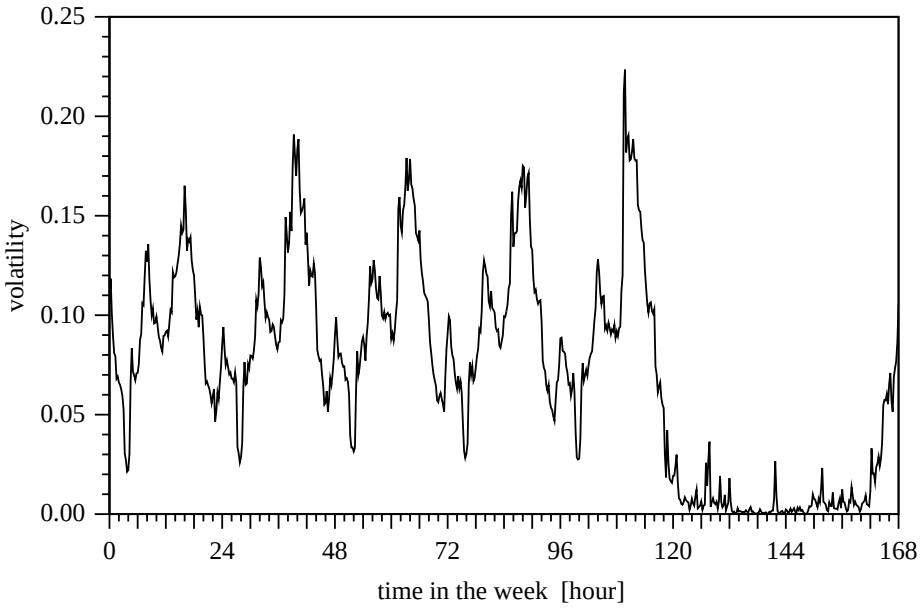


Figure 3: The volatility conditional to the time in the week for winter time (no DST) for USD/CHF. The axis is given in GMT time.

time horizon $\Delta t'$.

Figure 5 shows the correlation between a change in historical volatility and realized volatility. The change of volatility is similar to a derivative of volatility with respect to time. The correlation is essentially a measure for the response of the market to a change of volatility: if the volatility increases (decreases) and market participants mostly do (do not) trade, then the realized volatility will increase (decrease), resulting in a positive correlation. On the other hand, if market participants are not influenced by changes of volatility, a zero correlation would result. The figure shows the segmentation of the market in groups like intra-day traders reacting to changes at all time scales or like long term players (e.g., pension funds) influenced only by long term changes in the market. This correlation makes visible the cascade of information from long term to short time horizons.

Finally, an example of a real time application using Orla is our Olsen Information System (OIS). A demonstration version is available at `http://www.oanda.com/channels/investor/timing.shtml` and then clicking on one of the links for previewing the OIS. These computations are done at our company site using small Orla networks in real time and storing the computed values in the tick warehouse. Then, other C++ clients of the tick warehouse are broadcasting the values to the java applets of the web browsers.

References

[1] Gilles O. Zumbach and Ulrich A. Müller. Operators on inhomogeneous time series. *International Journal of Theoretical and Applied Finance*, 4(1):147–178, 2001.

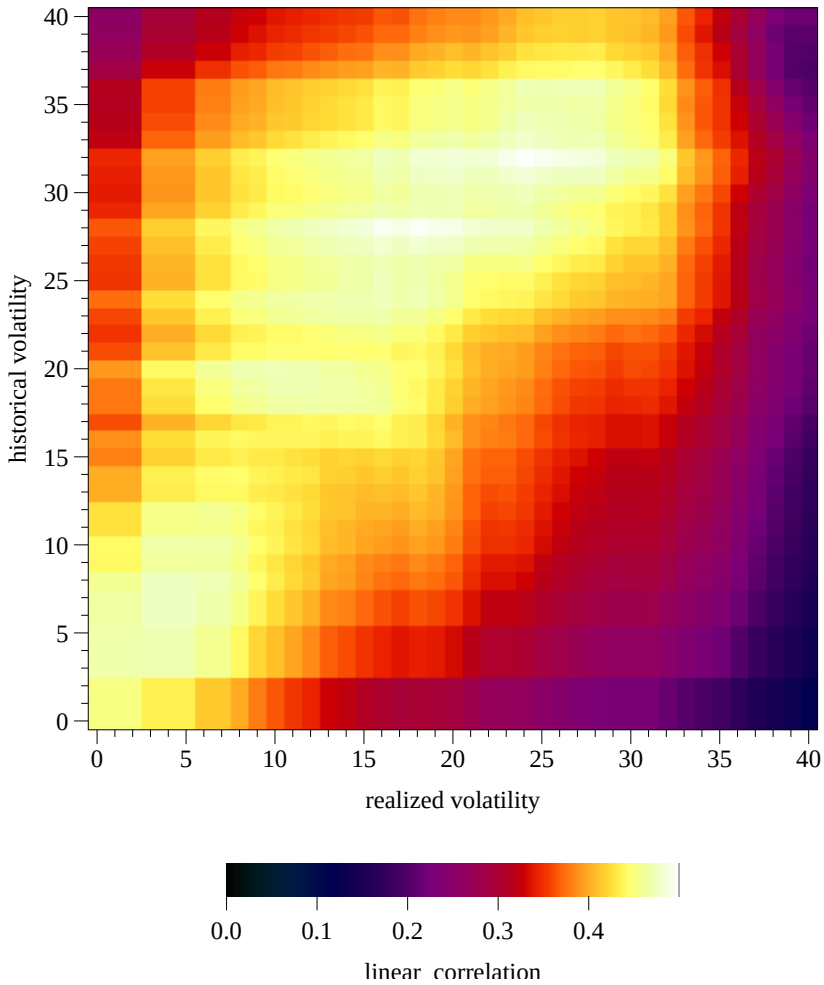


Figure 4: The correlation between historical and realized volatility. The axes corresponds to the time intervals Δt and $\Delta t'$ in a logarithmic scale for time intervals ranging from 1h20 to 57 days

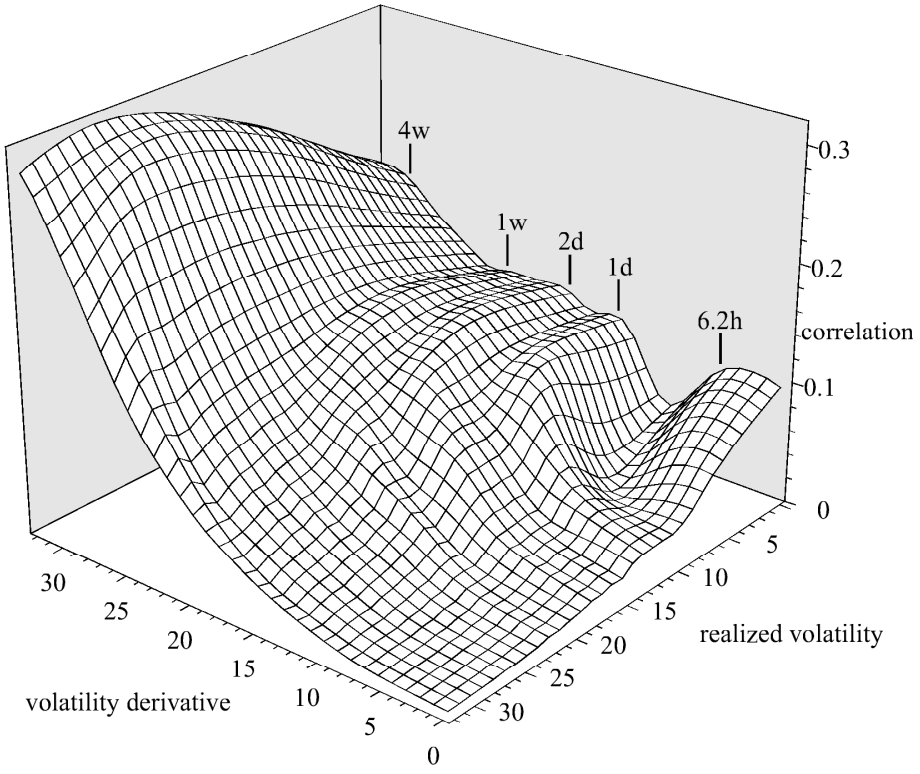


Figure 5: The correlation between the historical volatility derivative and the realized volatility. The axes correspond to the time intervals Δt and $\Delta t'$ in a logarithmic scale for time intervals ranging from 4h to 42 days