Fast and adaptive cointegration based model for forecasting financial time series

Seminario Informática CCTVal

Paola Arce

December 11, 2017

UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Departamento de Informática
Universidad Técnica Federico Santa María

Outline

- Motivation: High Frequency Trading
- Cointegration and the VECM model
- Challenges and Hypothesis
- Contributions
- Limitations and Applicability
- Conclusions and Future Research

Motivation

HFT: High Frequency Trading

High Frequency Trading has motivated computer-driven strategies capable of processing large amounts of data in short periods of time.

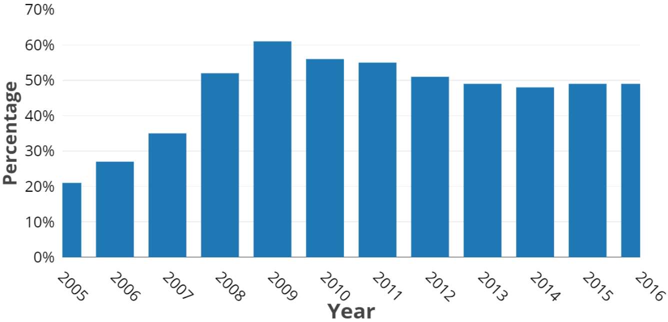


Figure 1: Percentage of HFT trades of total US Equity Trading. Source: TABB group

HFT Revenue

However, the revenues have fallen dramatically mainly because of more regulation, increasing the cost of HFT infrastructure and more competitive algorithms.

Figure 2: Revenue in the US. Source: TABB group

High Frequency data

High frequency time series are commonly:

- Non-stationary
- Irregularly spaced over time.
- Potentially cointegrated

This implies that classical statistical models as such can't be used. Therefore, financial time series forecasting is still a challenge.

Stationarity

Strong definition

A strictly stationary time series $y_t \in \mathbb{R}$, $\forall t \in \mathbb{Z}$, is one for which the probabilistic behaviour of every collection of values $\{y_{t_1}, y_{t_2}, \ldots, y_{t_L}\}$ is identical to that of the time shifted set, more precisely:

$$P\{y_{t_1} \leq c_1, .., y_{t_L} \leq c_L\} = P\{y_{t_1+h} \leq c_1, .., y_{t_L+h} \leq c_L\}$$

$\forall L \in \mathbb{N}, \forall h \in \mathbb{Z}$, where $c_1, \ldots, c_L$ are constants.

Stationarity

Weak definition

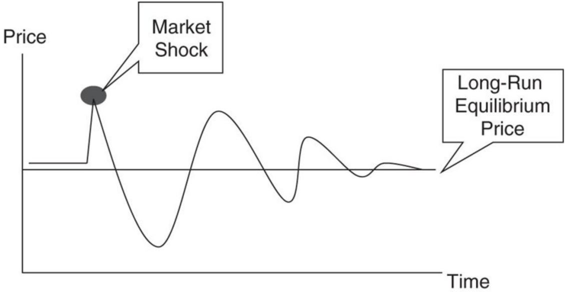
A weakly stationary time series is a process where the mean, variance and auto covariance do not change over time:

$$
\begin{aligned}
E(y_t) &= \mu \quad \forall t \in \mathbb{N} \\
E(y_t^2) &= \sigma^2 \quad \forall t \in \mathbb{N} \\
\lambda(s,t) &= \lambda(s+h, t+h) \quad \forall s,t \in \mathbb{N}, \forall h \in \mathbb{Z}
\end{aligned}
$$

with $\lambda(s,t) = E[(y_s - \mu)(y_t - \mu)]$

Stationarity: consequences

One of the consequences of stationary processes is how they recover from a shock. A shock represents an unexpected change in a variable in a particular period of time. For stationary time series, shocks to the system will gradually die away.



There are different types of non-stationary time series models often found in economics. Unit root process is one of them commonly used to model prices:

Unit root process

Unit root or I(1) processes are also called stochastic trend if they have the following form:

$$y_t = \mu + y_{t-1} + \epsilon_t$$

where ϵ_t is a stationary process. When $\mu = 0$ the process is called **pure random walk** and when $\mu \neq 0$ the process is called **random walk with drift**.

Integration

Integration of order d

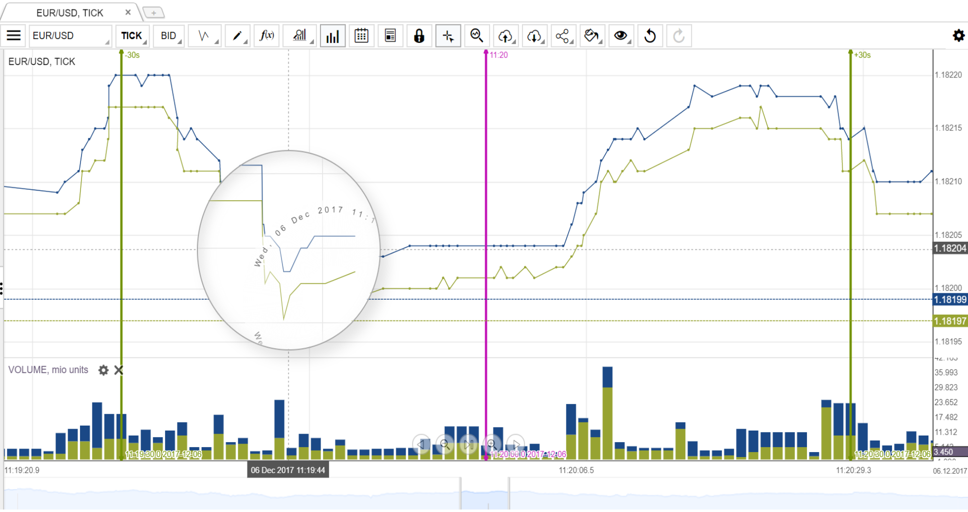
A time series $\mathbf{y}$ is said to be I(d), or integrated of order d, if after differentiating the variable d times, we get an stationary process, more precisely:

$$(1 - L)^d \mathbf{y} \sim \mathrm{I}(0)\,,$$

where I(0) is a stationary time series and L is the lag operator:

$$(1 - L)\mathbf{y} = \Delta \mathbf{y} = \mathbf{y}_t - \mathbf{y}_{t-1} \quad \forall t$$

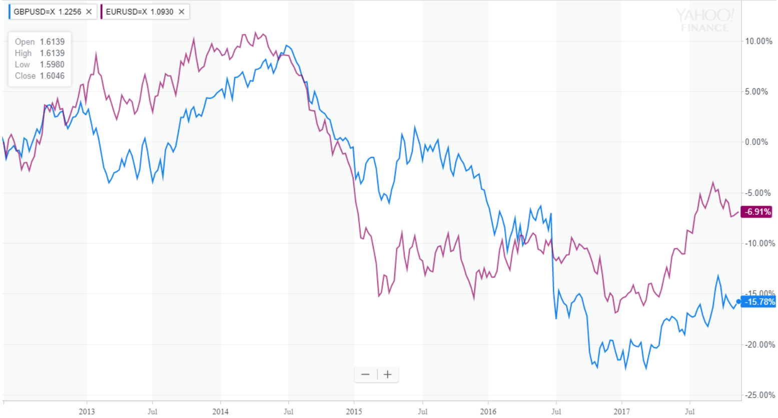
Irregularly spaced over time



Cointegration and the VECM model

Cointegration

Cointegration reflects the idea of that some set of non-stationary variables cannot wander too far from each other.



Cointegration definition

Cointegration

The relationship between non-stationary time series can be modelled if a **stationary linear combination** is shown to exist. When this happens it is said they have a **long-run equilibrium** relationship and the variables are cointegrated.

Cointegration: formal definition

Cointegration definition

Let $\mathbf{y} = \{\mathbf{y}^1, \dots, \mathbf{y}^l\}$ be a set of l time series of order I(1) which are said to be cointegrated if a vector $\boldsymbol{\beta} = [\beta(1), \dots, \beta(l)]^\intercal \in \mathbb{R}^l$ exists such that the time series,

$$\mathbf{Z}_t := \boldsymbol{\beta}^\intercal \mathbf{y} = \beta(1)\mathbf{y}^1 + \cdots + \beta(l)\mathbf{y}^l \sim \mathrm{I}(0)\,.$$

The Vector Error correction

The Vector Error correction model (VECM) is used to model cointegrated time series:

$$\Delta \mathbf{y}_t = \underbrace{\Omega \mathbf{y}_{t-1}}_{\text{Error correction term}} + \underbrace{\sum_{i=1}^{p-1} \phi_i^* \Delta \mathbf{y}_{t-i}}_{\text{Autorregresive term}} + \underbrace{c}_{\text{Intercept}} + \underbrace{\epsilon_t}_{\text{i.i.d}}$$

where p is the number of lags and $\boldsymbol{\Omega} = \boldsymbol{\alpha}\boldsymbol{\beta}^\intercal$. The columns of $\boldsymbol{\beta}$ contains the cointegration vectors and the rows of $\boldsymbol{\alpha}$ correspond with the adjusted vectors.

Ω matrix properties

The matrix $\boldsymbol{\Omega}$ has the following properties:

- If $\boldsymbol{\Omega} = 0$ there is no cointegration
- If $rank(\boldsymbol{\Omega}) = l$ i.e full rank, then the time series are not $\mathbf{I}(1)$ but stationary
- If $rank(\boldsymbol{\Omega}) = r, \quad 0 < r < l$ then, there is cointegration.

VECM limitations

VECM limitations

The Vector Error correction model is used to model cointegrated time series but only with **batch data and rarely used with high frequency data** mainly due to **computational limitations**:

- Calculation of the cointegration vectors β is obtained by the Johansen procedure which is of order $O(n^3)$.
- VECM parameters are obtained using the ordinary least squares (**OLS**) method.

Online VECM

Recently, online learning algorithms have been proposed to solve problems with large data sets mainly because of:

- Their simplicity
- They process one instance at a time
- The hypothesis is updated every time new data arrives

Online learning algorithms

Algorithm 1 Structure of an Online Learning System

1: Receives input $\mathbf{x}_t$
2: Makes prediction $\hat{\mathbf{y}}_t$
3: Receives response $\mathbf{y}_t$
4: Incurs loss $l_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$

Performance is later measured after T trials as:

$$L_T = \sum_{t=1}^{T} l_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$$

Challenges and Hypothesis

Challenge

Proposal

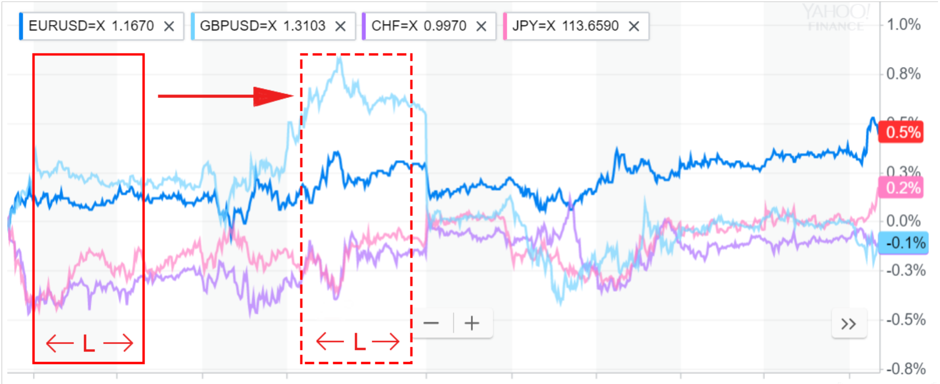
To adapt VECM to be used with high frequency data, where the best parameters are updated when new data arrives.

Hypothesis

An algorithm based on cointegration and high performance computing will allow faster forecasting algorithms for financial time series to be obtained while maintaining good accuracy levels.

Contributions

Adaptive VECM



VECM(p) matrix form

$$\Delta \mathbf{y}_t = \underbrace{\Omega \mathbf{y}_{t-1}}_{\text{Error correction term}} + \underbrace{\sum_{i=1}^{p-1} \phi_i^* \Delta \mathbf{y}_{t-i}}_{\text{Autorregresive term}} + \underbrace{c}_{\text{Intercept}} + \underbrace{\epsilon_t}_{\text{i.i.d}}$$

If we have N data points, with $N > p$, the VECM matrix form is as follows:

$$\underbrace{\begin{bmatrix} \Delta \mathbf{y}_{p+1}^{\mathsf{T}} \\ \Delta \mathbf{y}_{p+2}^{\mathsf{T}} \\ \vdots \\ \Delta \mathbf{y}_{N}^{\mathsf{T}} \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\left[\begin{array}{ccccc:c} \mathbf{y}_p^{\mathsf{T}} \boldsymbol{\beta} & \Delta \mathbf{y}_p^{\mathsf{T}} & \Delta \mathbf{y}_{p-1}^{\mathsf{T}} & \cdots & \Delta \mathbf{y}_2^{\mathsf{T}} & 1 \\ \mathbf{y}_{p+1}^{\mathsf{T}} \boldsymbol{\beta} & \Delta \mathbf{y}_{p+1}^{\mathsf{T}} & \Delta \mathbf{y}_p^{\mathsf{T}} & \cdots & \Delta \mathbf{y}_3^{\mathsf{T}} & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{y}_{N-1}^{\mathsf{T}} \boldsymbol{\beta} & \Delta \mathbf{y}_{N-1}^{\mathsf{T}} & \Delta \mathbf{y}_{N-2}^{\mathsf{T}} & \cdots & \Delta \mathbf{y}_{N-p-1}^{\mathsf{T}} & 1 \end{array}\right]}_{\mathbf{X}} \underbrace{\begin{bmatrix} \boldsymbol{\alpha}^{\mathsf{T}} \\ \boldsymbol{\Phi}_1^{*\top} \\ \boldsymbol{\Phi}_2^{*\top} \\ \vdots \\ \boldsymbol{\Phi}_{p-1}^{*\top} \\ \mathbf{c}^{\mathsf{T}} \end{bmatrix}}_{\mathbf{W}} + \underbrace{\begin{bmatrix} \boldsymbol{\epsilon}_{p+1}^{\mathsf{T}} \\ \boldsymbol{\epsilon}_{p+2}^{\mathsf{T}} \\ \vdots \\ \boldsymbol{\epsilon}_{N}^{\mathsf{T}} \end{bmatrix}}_{\mathbf{E}}$$

Online version of VECM

VECM can be solved using a sliding windows of data, the problem can be expressed as follows:

$$\mathbf{X}(t) = \begin{bmatrix} \mathbf{x}_{t-L}^\intercal \\ \vdots \\ \mathbf{x}_t^\intercal \end{bmatrix}, \mathbf{Y}(t) = \begin{bmatrix} \mathbf{y}_{t-L}^\intercal \\ \vdots \\ \mathbf{y}_t^\intercal \end{bmatrix}.$$

The optimal solution using a OLS is then:

$$\mathbf{W}(t)_* = (\mathbf{X}(t)^\intercal \mathbf{X}(t))^{-1} \mathbf{X}(t)^\intercal \mathbf{Y}(t) \quad (1)$$

$$= \left(\sum_{i=0}^{L} \mathbf{x}_{t-i} \mathbf{x}_{t-i}^\intercal \right)^{-1} \sum_{i=0}^{L} \mathbf{x}_{t-i} \mathbf{y}_{t-i}^\intercal \quad (2)$$

Algorithm 2 Online Ordinary Least Squares

Input:

$\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$: N input vectors

$\{\mathbf{y}_1, \dots, \mathbf{y}_N\}$: N targets

L: sliding window size ($L < N$)

Output:

$\{f(\mathbf{x}_{L+1}), \dots, f(\mathbf{x}_N)\}$: model predictions

Initialize $\mathbf{A} = \sum_{t=1}^{L} \mathbf{x}_t \mathbf{x}_t^\top$ and $\mathbf{b} = \sum_{t=1}^{L} \mathbf{x}_t \mathbf{y}_t^\top$

for $t = L + 1$ to N **do**

  read new $\mathbf{x}_t$

  $\mathbf{A} = \mathbf{A} + \mathbf{x}_t \mathbf{x}_t^\top - \mathbf{x}_{t-L-1} \mathbf{x}_{t-L-1}^\top$

  output prediction $f(\mathbf{x}_t) = \mathbf{b}^\top \mathbf{A}^{-1} \mathbf{x}_t$

  Read new $\mathbf{y}_t$

  $\mathbf{b} = \mathbf{b} + \mathbf{x}_t \mathbf{y}_t^\top$

end for

How to choose L?

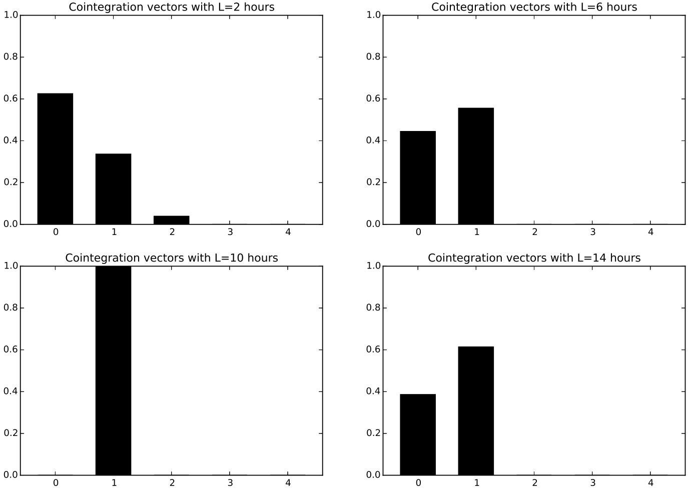


Figure 3: Distribution of the number of cointegration vectors using $p = 1$ lags.

Percentage of cointegration

Since $r = 0$ means no cointegration and $r = l = 4$ reveals that no process is I(1) but stationary. The interesting cases of cointegration are those where r lies strictly between 0 and l, i.e. $0 < r < l$.

Percentage of cointegration (PC)

$$PC = \frac{\#\{it \mid it \text{ has } r \text{ c.v. with } 0 < r < l\}}{\#it} \times 100$$

Data

- This data was collected from **Dukascopy**, a free database which gives access to the **Swiss Foreign Exchange marketplace**.
- Tests were carried out using four foreign exchange rates all related to USD: **EURUSD, GBPUSD, USDCHF** and **USDJPY**.
- The tests were done using **10-seconds frequency** from ask prices from the 11th to the 15th of August 2014.

PC vs MSE

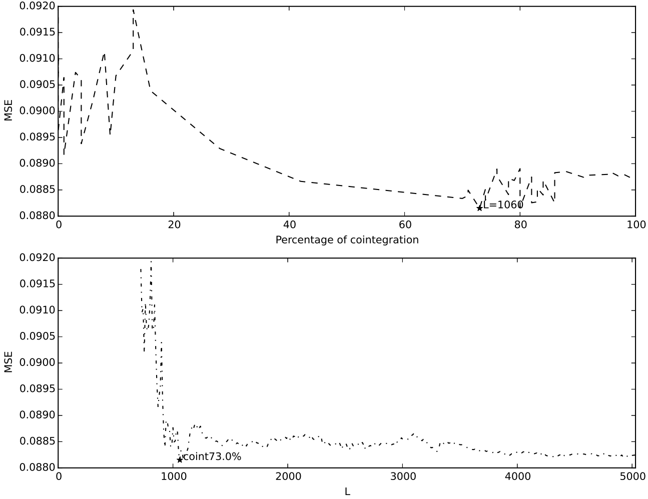


Figure 4: MSE versus the percentage of cointegration considering 1000 iterations.

Adaptive VECM algorithm (AVECM)

AVECM has the following features:

1. It uses VECM only considering a sliding windows of size L of data.
2. We proposed to choose L and the number of lags p in order to maximise the percentage of cointegration PC in the near past. This process was done every time that new data was processed.
3. And to use a distributed environment to make this parameters search, since this is the most expensive routine.

Unit root tests

Augmented Dickey Fuller (ADF) test with lags $p = 1, 2, 3, 4, 5$.

Variable	ADF(1)	ADF(2)	ADF(3)	ADF(4)	ADF(5)
EURUSD	-0.052	-0.054	-0.054	-0.054	-0.054
GBPUSD	-0.744	-0.784	-0.805	-0.837	-0.846
USDCHF	-0.476	-0.493	-0.493	-0.495	-0.502
USDJPY	0.357	0.360	0.360	0.367	0.367
Δ EURUSD	-128.4*	-128.4*	-96.85*	-89.12*	-89.12*
Δ GBPUSD	-131.4*	-112.7*	-102.5*	-92.86*	-88.29*
Δ USDCHF	-127.8*	-127.8*	-96.94*	-88.82*	-80.79*
Δ USDJPY	-135.1*	-135.1*	-101.2*	-101.2*	-101.2*

Table 1: Unit roots tests for EURUSD, GBPUSD, USDCHF and USDJPY at 10-second frequency.

Performance accuracy

	MSE			*U*-statistic	
	AVECM	ARIMA	p-value	AVECM	ARIMA
EURUSD	1.0702 e-09	1.1481 e-09	9.250 e-12	0.6863	0.7108
GBPUSD	1.6630 e-09	1.7408 e-09	6.951 e-02	0.6866	0.7025
USDCHF	5.8503 e-10	6.3545 e-10	2.899 e-14	0.6803	0.7091
USDJPY	6.3483 e-06	6.5194 e-06	6.853 e-05	0.6964	0.7057

Parallel implementation

The function that get optimal parameters L and p was implemented using MPI in Python. Tests ran in a cluster with 2 servers Xeon E5-2667 (2.90GHz) of 24 cores each (48 cores in total) and 24GB RAM.

Parameter settings:

- The L parameter was always chosen between 100 and 4000 and p always took values between 1 and 5.
- Parameter *nparams* represents the number of pairs (L,p) used to maximise the percentage of cointegration.

Execution times

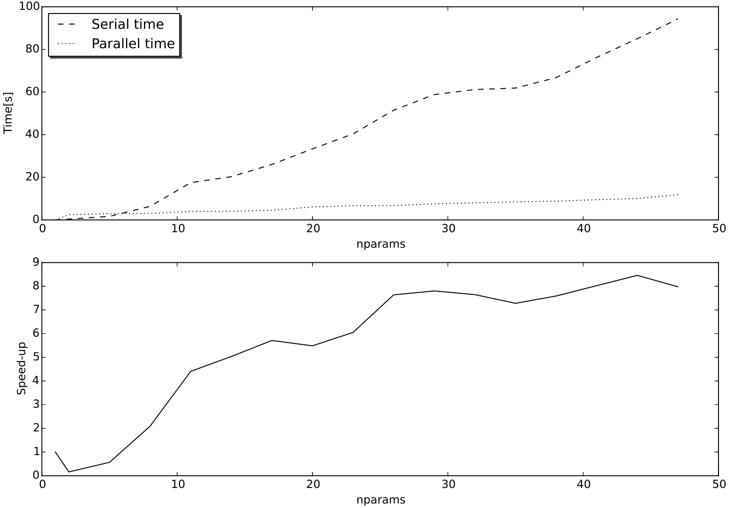


Figure 5: Computing time of sequential and parallel algorithm is shown in the upper figure. Speed-up is shown below.

Limitations and Applicability

Limitations and Applicability

- Cointegration information can be used as an integration tool to detect arbitrage opportunities.
- Use with stocks, forex rates and crypto-currency?
- Applicable to other non-stationary but cointegrated time series

Conclusions and Future Research

Conclusions I

- Cointegration relations change with time. We empirically showed that the Johansen method is sensitive to the number of lags but also to the amount of data considered.
- AVECM improves performance measures by finding parameters of L and p maximising the percentage of cointegration.
- AVECM performance is superior to ARIMA and the naive random walk model in terms of MSE and U-statistic.

Conclusions II

- The parallel implementation allowed the execution times to be reduced more than 9 times and therefore a response time was obtain before 10 seconds.
- We showed that VECM computational limitations can be tackled using a an online approach in a distributed environment.

Future research

- To add **matrix optimizations** to obtain VECM parameters.
- To add a regularization parameter to get better generalisation capabilities using **Ridge Regression** and the **Aggregating Algorithm for Regression**.
- To include **more explicative variables** such as bid-ask spread and change in volume.
- The online approach for **other econometrics models** it is also worthy of study.

Thank you for your attention

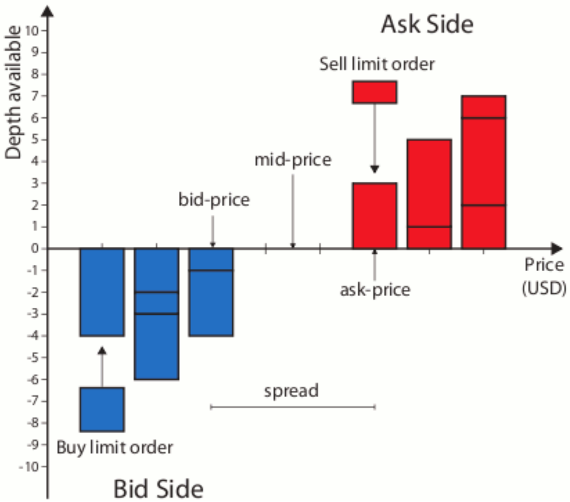
Questions?

Paola Arce, Jonathan Antognini, Werner Kristjanpoller, and Luis Salinas, *An online vector error correction model for exchange rates forecasting*, Proceedings of the International Conference on Pattern Recognition Applications and Methods, 2015, pp. 193–200.

Paola Arce, Jonathan Antognini, Werner Kristjanpoller, and Luis Salinas, *Fast and adaptive cointegration based model for forecasting high frequency financial time series*, Computational Economics (2017), 1–14.

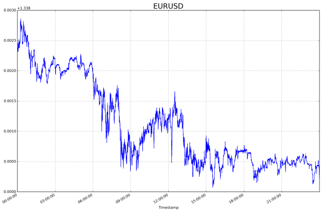
Paola Arce and Luis Salinas, *Online ridge regression method using sliding windows*, XXXI International Conference of the Chilean Computer Science Society (2012).

The order book

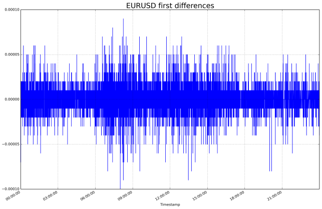


Integration example

High frequency financial time series are commonly an I(1) process



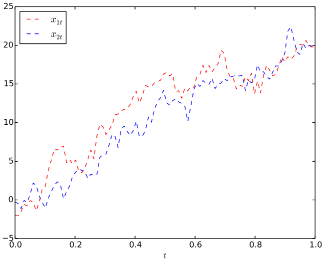
Since their differences are stationary (I(0) process).



Spurious regression

Standard regression techniques are applied to non-stationary data.

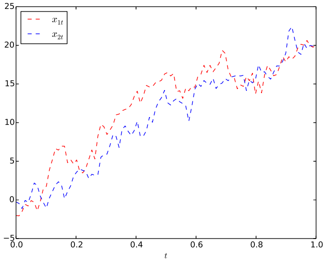
The use of non-stationary data can lead to spurious regressions.

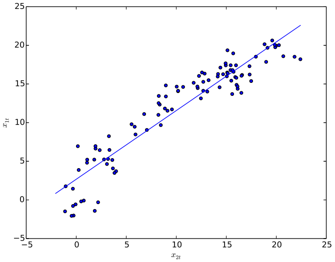


Spurious regression

Standard regression techniques are applied to non-stationary data.

The use of non-stationary data can lead to spurious regressions.





Random walk

The random walk model is defined as:

$$\mathbf{y}_t = \mathbf{y}_{t-1} + \epsilon_t \tag{3}$$

The naive forecast of the time series difference $\hat{\mathbf{y}}_{t+1}$ for the random walk model is defined as:

$$\hat{\mathbf{y}}_{t+1} = \mathbf{y}_t + \hat{\epsilon}_{t+1} \tag{4}$$

where $\hat{\epsilon}_{t+1} = \epsilon_t$.

ARIMA

A process can be modelled as an ARIMA(p, d, q) if $\mathbf{x}_t = \Delta^d \mathbf{y}_t$, is an ARMA($p, q$). An ARMA($p, q$) model is the following:

$$\mathbf{x}_t = \sum_{i=1}^{p} \phi_i \mathbf{x}_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{5}$$

with coefficients $\phi_p \neq 0$, $\theta_q \neq 0$ and $\sigma_\epsilon^2 > 0$.

AVECM Algorithm

Input:

$\mathbf{y}$: matrix with N input vectors and I time series

j: Starting point of testing

it: Ending point of testing

ps: list of p values

Ls: list of L values ($L < N$)

m: Iterations to determine parameters ($m < N - L$)

Output:

$\{\hat{\mathbf{y}}[1], \ldots, \hat{\mathbf{y}}[it]\}$: prediction vectors

for $i = j$ to it **do**

  $\mathbf{Y} \leftarrow \mathbf{y}[:, i-1]$

  $L, p \leftarrow \texttt{get\_best\_params}(Ls, ps, m, \mathbf{Y})$

  $model = \text{VECM}(\mathbf{Y}, L, p)$

  $\hat{\mathbf{y}}[i-j] = model.predict()$

end for