Generative Models and Time Series Predictions for Financial Applications with Kernels Methods

Jean-Marc Mercier [‡] [*]

31 10 2022

Abstract

This presentation deals with generative and predictive models based on kernel methods. After introducing our kernel library, We illustrate these models with a toy-example of risk framework based on time series forecasting, that could be used for real-time P&L explanation of large derivative portfolios, or other risk measures as Expected Positive Exposure or Credit Valuation Risk.

1 Introduction

In this jupyter notebook presentation, we would like to introduce our internal AI library, highlighting it with two interesting applications of machine learning to finance, namely:

- Synthetic data generation. For finance applications, there exists numerous applications, among them are time series forecast of risk sources that we illustrate in this presentation. The most-known technologies for producing synthetic datas are neural networks based, as for instance GAN, WGAN, CLGAN, that are at heart of synthetic images or videos production.
- Predictives methods. Here too, applications for finance are numerous, and we illustrate here an application to real-time P&L computations. The most-known technologies for predictive methods are for instance neural networks, decision trees, etc...

For this presentation, we used our open source library, codpy [1]. Codpy is a kernel based technology (RHKS - Reproducing Kernel Hilbert Space theory), that we have developped and are using for the internal needs at MPG-Partners. Codpy is an alternative library to other AI libraries (pytorch, theano, tensorflow, etc. . .) that has some nice properties for finance applications:

- It is an explainable method. All produced results come with computable error estimates allowing to qualify them. Error estimates allow moreover to link naturally to Optimal Transport Theory based tools, used thoroughly by this library.
- It is a performing and accurate library, particularly while dealing with sparse input data (small training set).
- It can compute efficiently a wide zoo of differential operators. Indeed, this library was thought primarily to solve mathematical physics based problems, and is used to approximate some dynamical systems described by a PDE (Partial Differential Equations) model.

[*]MPG-Partners, 136 Boulevard Haussmann, 75008 Paris, France. jean-marc.mercier@mpg-partners.com

[1]The codpy user manual is accessible clicking here. For installation, follow the guidelines clicking here.

This presentation is structured as follows :

Contents

2 A quick tour to kernels methods

2.1 Predictive methods with kernels

Artificial Intelligence (AI) is mainly based over predictions, that are nothing else but extrapolation methods.

AI Vocabulary:. $X, f(X)$ is the **training set**. Z is the **test set**, f_z is the **prediction set**, $f(Z)$ is the reference (ground truth) value. Y is the **parameter set**, that are internal parameters to a prediction algorithm.

2.1.1 Projection (extrapolation) and differential operators

Kernel methods define a quite simple prediction operator, that we present now briefly. Let X (resp. Y, Z) be a set of N_x (resp. N_y, N_Z) **distinct** points in dimension D, f any continuous, vector valued, function. Notations

$$X := \{x_d^n\}_{n,d=1}^{N_x,D} \in \mathbb{R}^{N_x,D}, \quad f(X) \in \mathbb{R}^{N_x,D_f} \tag{2.1}$$

Let k be a kernel, that is a symmetric and positive definite (see [2] for a definition) scalar function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. Consider $K(X,Y)$ the Gram matrix, i.e. $K(X,Y) := (k(x^n, y^m))_{n,m=1}^{N_x,N_y}$. We define a **prediction** as

$$f_Z := \mathcal{P}_k(X,Y,Z)f(X), \quad \mathcal{P}_k(X,Y,Z) := K(Y,Z)K(X,Y)^{-1}. \tag{2.2}$$

The inverse is computed using a least-squares approach, as follows, $K(X,Y)^{-1} = (K(Y,X)K(X,Y) + \epsilon)^{-1}K(Y,X)$, where ϵ is a (optional) regularization term, as is the Tychonov regularization method, or other ones. A classical choice for the parameter set are $Y = X$ (extrapolation), or $Y \subset X$ (interpolation - Nystrom method). Starting from the formula (2.2), we can define all kind of differential operators, as for instance the gradient

$$\nabla f_Z = (\nabla_Z K)(Y,Z)K(X,Y)^{-1}f(X). \tag{2.3}$$

CoDpy function: $f_z =$ op.projection$(X,Y,Z,f(X),k,...), \nabla f_Z =$ op.nabla$(X,Y,Z,f(X),k,...)$.

2.1.2 Illustration with MNIST : hand-written recognition

2.2 Error estimates

The projection operator (2.2) benefits from the following error estimate (see [5]), that are confidence levels

$$\|f(Z) - f_Z\|_{\ell^2} \leq D_k(X,Y,Z)\|f\|_{\mathcal{H}_k}, \tag{2.4}$$

where $D_k(X,Y,Z) := D_k(X,Y) + D_k(Y,Z)$, and where the discrepancy between two discrete probability measures X and Y, induced by a kernel k is

$$D_k\big(X,Y\big)^2 := \frac{\sum_{n,m=1}^{N_x} k(x^n,x^m)}{N_x^2} + \frac{\sum_{n,m}^{N_y} k(y^n,y^m)}{N_y^2} - \frac{2\sum_{n,m=1}^{N_x,N_y} k(x^n,y^m)}{N_xN_y}, \tag{2.5}$$

CoDpy function: $D_k\big(X,Y\big)^2 =$ op.Dnm$(X,Y,k), \|f\|_{\mathcal{H}_k} =$ op.norm$(X,f(X),k)$

2.2.1 Illustration with MNIST : discrepancy errors and scores

2.3 Clustering method

CoDpy defines a clustering method as follows:

$$\bar{Y} = \arg \inf_{Y \in \mathbb{R}^{D,N_Y}} D_k(X,Y), \tag{2.6}$$

called **sharp discrepancy sequences**. This approach is an alternative to more classical clustering algorithms as K-means ones. CoDpy function: op.sharp_discrepancy(X,Y,k,…).

2.3.1 Illustration with the MNIST : clustering and scores enhancement

In this section, we would like to illustrate some benchmarks capabilities of our framework, as well as illustrating a first method to synthetic data generation, that is clustering.

A comparison between methods. Clustering provides an alternative, and interesting way to synthetic data generation. Codpy provides facilities to plug painlessly other algorithms libraries. We take advantages to benchmark scikit's k-means algorithm implementation, based over minimization of the "inertia" functional, that is the sum of distances of all points to the centroid in a cluster. We compare k-means to codpy clustering method (sharp dicrepancy sequences) in what follows.

```
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
## C:\informatique\Python39\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans i
##   warnings.warn(
```

Note that the cluster centroids themselves are 784-dimensional points, and can themselves be interpreted as the "typical" digit within the cluster. Figure 1 plots some examples of computed clusters, interpreted as images. As can be seen, they are perfectly recognizable.

We illustrate a benchmark plot, displaying the computed performance indicator of scikit's k-means and codpy's MMD minimization-based algorithm in terms of MMD, inertia, accuracy scores (when applicable) and execution time. The higher the scores and the lower are the inertia and MMD the better.

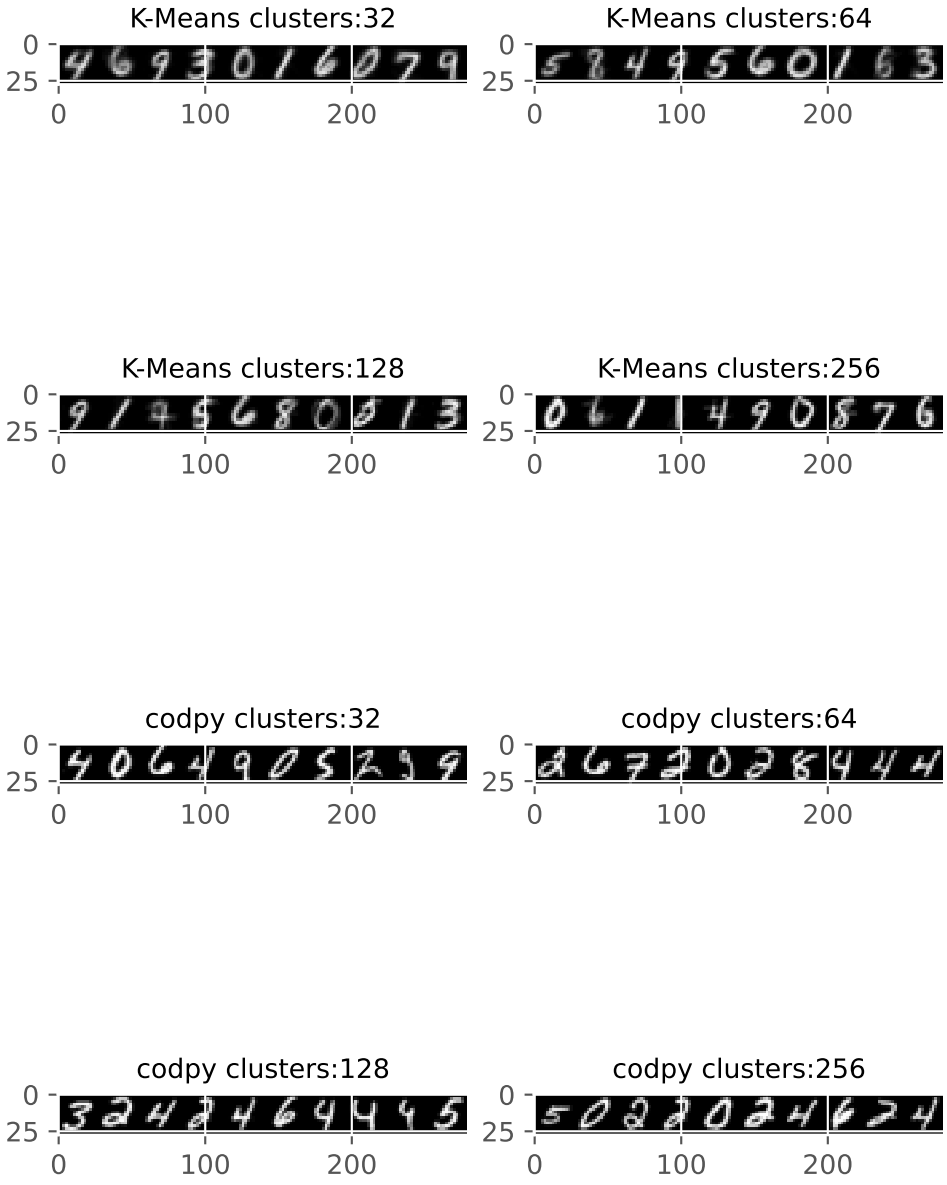
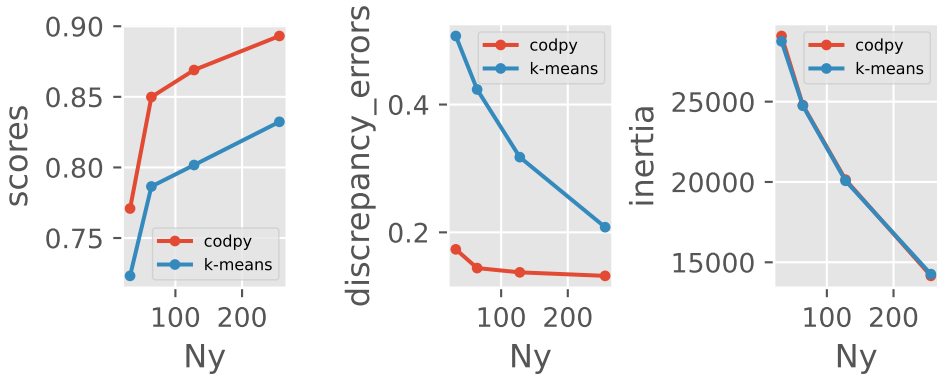


Figure 1: Scikit and codpy clusters interpreted as images



The scores are quite high, compared to supervised methods for similar size of training set.

2.4 Generative methods with kernels - A quick introduction to the sampler method

A generative method is a method that take as input discrete samples of a given distribution, and reproduce it, hypothesizing that this distribution is continuous. CoDpy function: alg.sampler(X,Y,k,...)

For illustration goals, we apply this algorithm for two bi-modal distributions based on a Gaussian and a Student's distribution namely $\mathcal{N}(0,1)$ and $t(\nu = 5)$. We use here two distinct sets (training set X and test set Z) to highlight some convergence properties of the sampler algorithm:

- IID : X, Z are iid samples of $\mathbb{X}$.
- SDS : X, Z are sharp discrepancy sequences (SDS) of $\mathbb{X}$.

For both sets, the size of the training set is $N_x = 1000$, whereas the size of the test set is $N_z = 500$. We plot the results computed by the sampler algorithm in Figure 2 (resp. Figure 3) for the IID case (resp. SDS case).

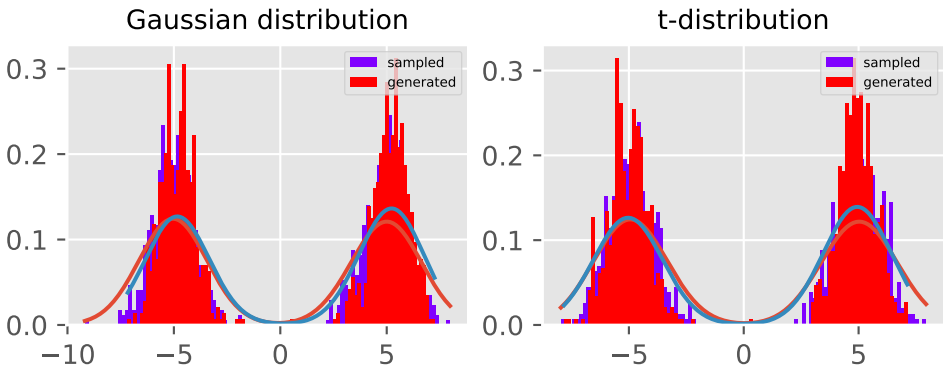


Figure 2: Density of generated IID distributions

Once generated, we compute various statistic indicators to check the similarities between both distributions (historical and generated)

Table 1: Statistics of IID-generated distributions

Mean	Variance	Skewness	Kurtosis	KS test
-0.047(0.39)	0.0047(-0.077)	26(26)	-1.9(-1.9)	0.0024(0.05)

As can be seen, these algorithms are sensitive to input data. In particular, using clustering methods improves algorithm performances.

Table 2: Statistics of SDS-generated distributions

Mean	Variance	Skewness	Kurtosis	KS test
0.013(0.17)	0.0013(-0.068)	26(26)	-1.9(-1.9)	0.15(0.05)

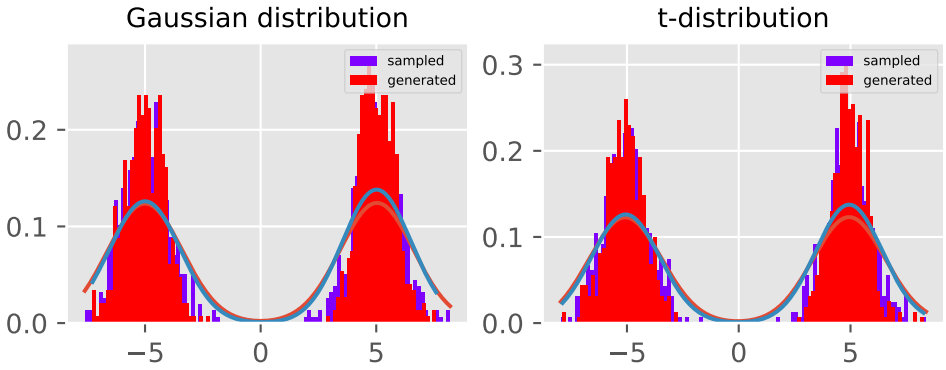


Figure 3: Density of generated SDS distributions

2.4.1 Retrieve any distribution, in any dimension...

The previous distribution is a 3-dimensional one, with three stocks market data. We outlight here that synthetic data generation is a general approach, and one can consider any distributions, as for instance the MNIST database, consisting of 60000 hand-written digits, having 28x28=784 pixels resolution. We consider it as a discrete 784-dimensional distribution having 60000 samples, the figure **??** showing the first hundred ones

The incentive to use this data set is to illustrate how our algorithms scale with dimensionalities, as well as linking towards classical learning machine problems.

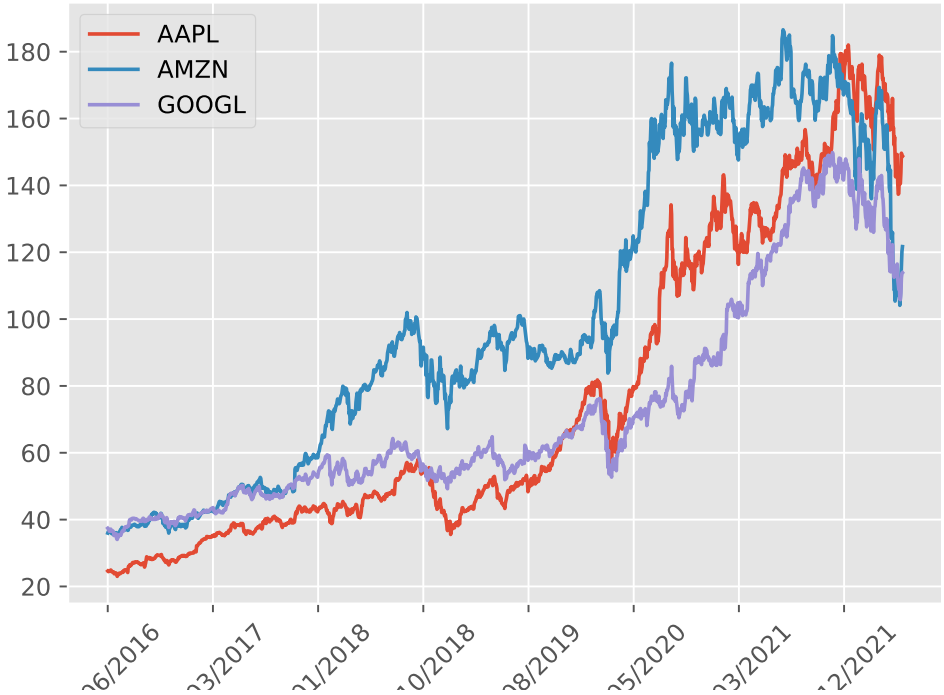
2.4.2 Illustration : resample the MNIST

Note that this resampling function is a general one, and one can generate from any distribution. For instance, the same algorithm applied to the hand-written digits distribution **??** produced the following resampling examples [2]

3 Application settings - Input data

3.1 Retrieve market data

Let us download real market data, retrieved from January 1, 2016 to December 31, 2021, for three assets: Google, Apple and Amazon. These data are plot Figure **??**.



To produce this figure, we use the following global settings:

Table 3: Global settings

begin date	end date	pricing date	symbols
01/06/2016	01/06/2022	01/06/2022	c("AAPL", "GOOGL", "AMZN")

[2] We learnt from a distribution consisting of the first 500 over the 60000 hand-written digits of the MNIST database images for performance purposes.

3.2 Set a portfolio of instruments

We define a payoff function as $P(t,x) \mapsto P(t,x) \in \mathbb{R}^{D_P}$, with D_P corresponding to the number of instrument. We consider here a single instrument $D_P = 1$, the instrument being a basket option written on our underlyings. We represent this payoff in a two-dimensional figure with axis basket values in Figure 4.

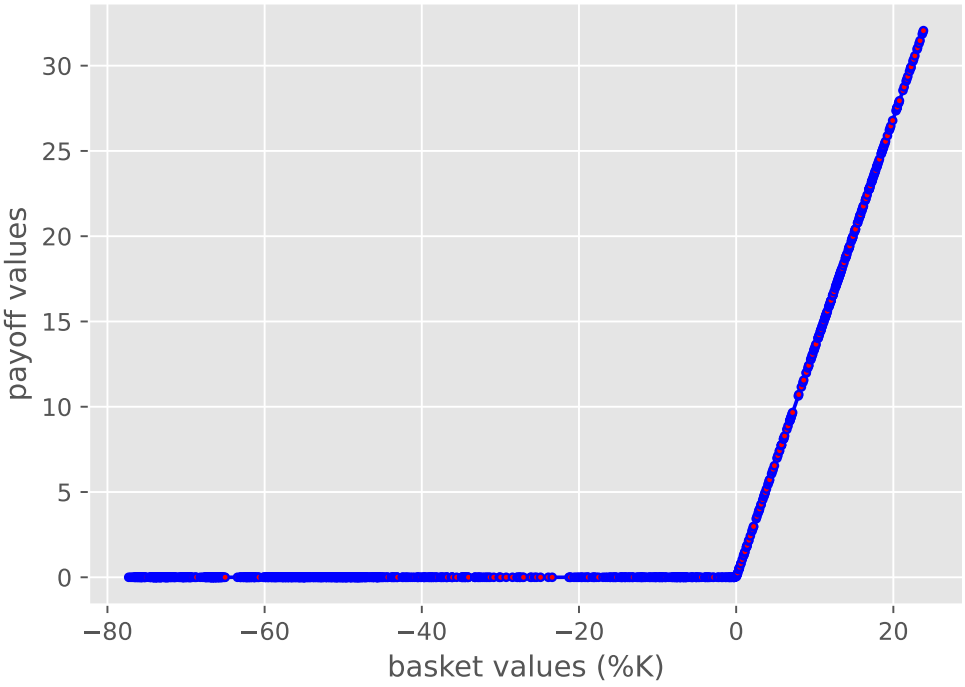


Figure 4: A payoff of an basket option

3.3 Set a pricing engine

We define a pricing function as a payoff, that is a vector-valued function $(t, x) \mapsto P(t, x) \in \mathbb{R}^{D_P}$. We represent this pricing function in a two-dimensional figure 5 with axis basket values.

The pricing function here is selected as a simple Black and Scholes formula, hence hypothesizing that the basket values are log normal [3]

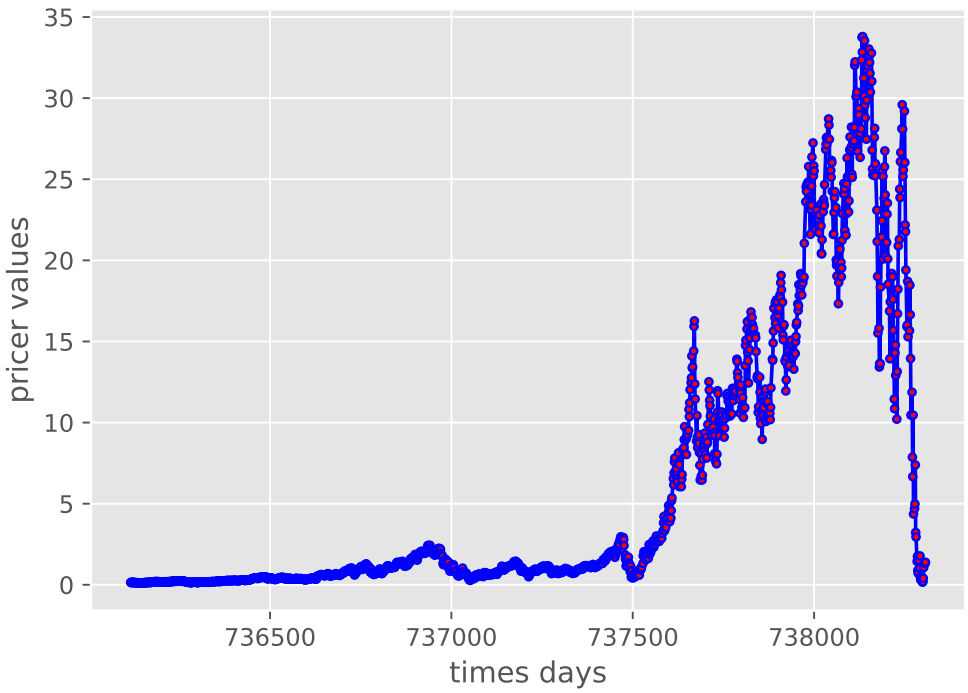


Figure 5: Pricing as a function of time

[3] this choice is made for performance purposes here, but any pricing function can be plugged in.

4 Synthetic market data generation

4.1 Fit a model to the historical distribution

A model can be described by a stochastic differential equation. For instance consider a log-normal process, described by $X_t = \mu X_t + X_t \sigma dB_t$, B_t being the standard Brownian motion, having solution $X_t = X_s \exp((t-s)(\mu - \sigma^2/2) + \sqrt{t-s}\sigma\mathcal{N}(0,1))$, $\mathcal{N}(0,1)$ being the normal law. Fitting this model to historical data plot at figure **??** would consist in fitting the parameters μ, σ to the historical data.

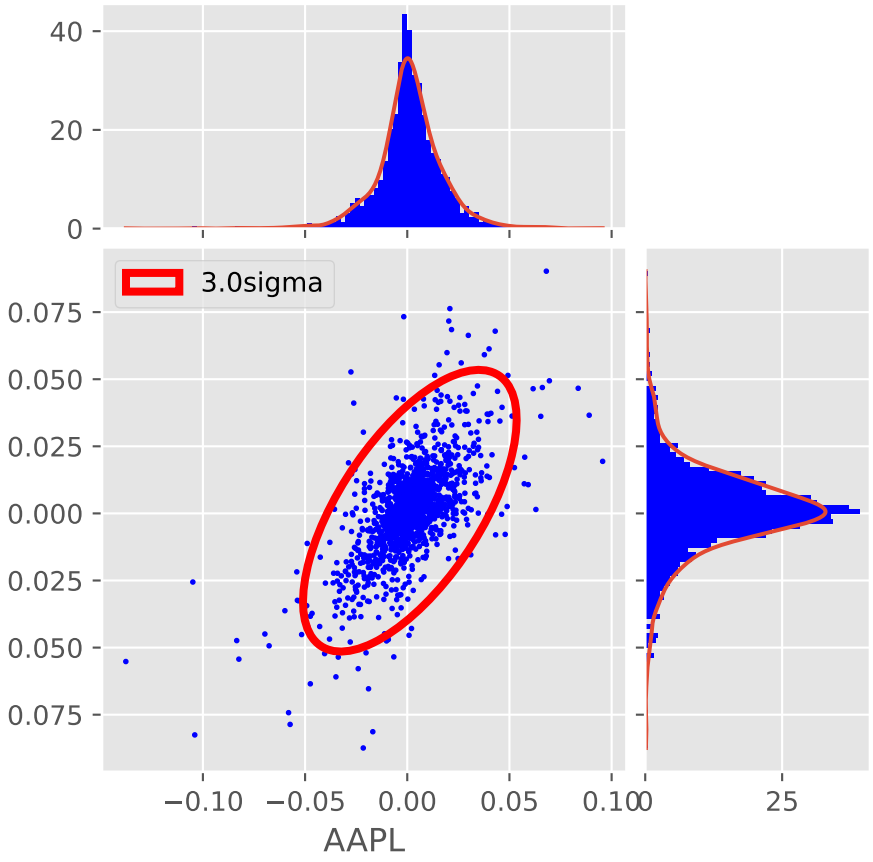


Figure 6: Log return distribution of historical market data

Synthetic data generation introduces somehow a new paradigm, where known processes (as the standard Brownian B_t), are replaced by unknown random variable to fit. So instead of (**??**), consider the following problem : define a process, having form

$$X_t = X_s \exp((t-s)\mu + \sqrt{t-s}\mathbb{X}), \tag{4.1}$$

where $\mathbb{X}$ is an unknown random variable to fit to historical data. To that aim, consider the log transformation

$$\mathbb{X} = \frac{\ln(X_t) - \ln(X_s) - (t-s)\mu}{\sqrt{t-s}}, \tag{4.2}$$

in order to separate the random variable $\mathbb{X}$ to fit to historical data. Figure 6 plots the ditribution retrieved from our historical data after the transformation (4.1).

4.2 Generate the distribution

We then provide a function, producing a continuous sampling function from any discrete input distribution. Figure 7 is a result of a resample of the historical distribution

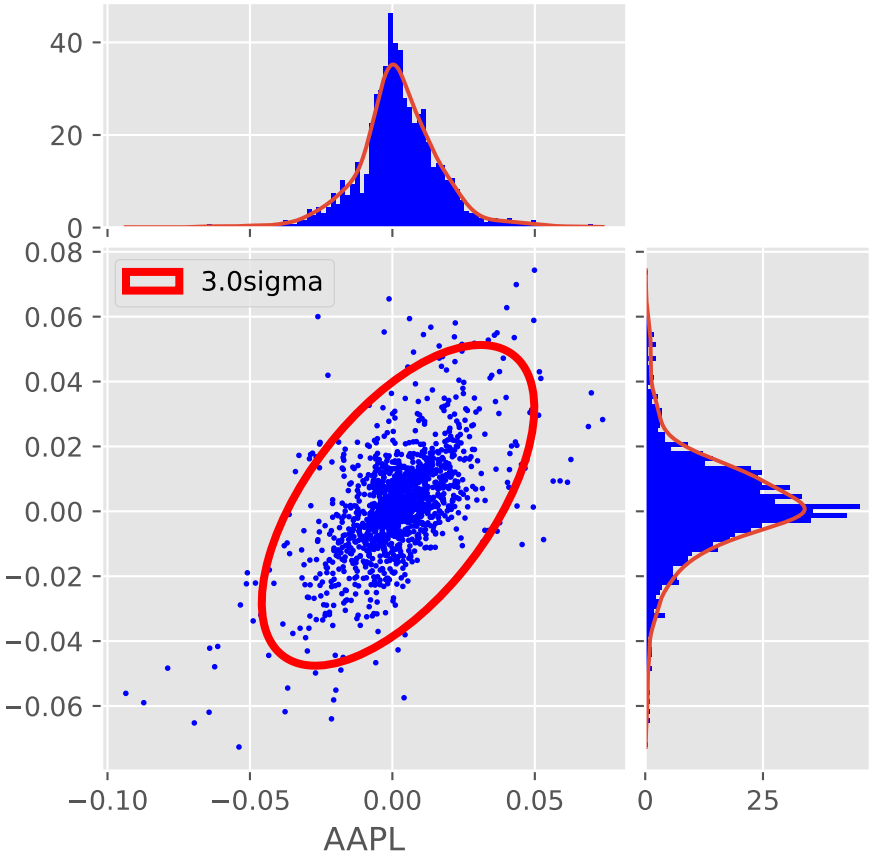


Figure 7: Log return distribution of generated market data

4.2.1 Check generated paths

In the table 4, we compute various statistical indicators, as the fourth moments and Kolmogorov-Smirnov tests, to challenge our generated data against the original one.

Table 4: Stats for historical (generated) data

	AAPL	AMZN	GOOGL
Mean	0.0013(0.002)	0.001(0.0018)	0.00073(0.0012)
Variance	-0.48(-0.26)	-0.14(0.026)	-0.48(-0.26)
Skewness	0.0003(0.00025)	0.00031(0.00027)	0.00025(0.00019)
Kurtosis	7.4(4.2)	3.7(3)	6.5(3.5)
KS test	0.41(0.05)	0.36(0.05)	0.49(0.05)

4.3 Build and check generated paths

Ten examples of re-generated paths in figure 8. These paths can be used for Monte-Carlo sampling, and we can also build PDE (Partial Differential Equations) pricers, whatever the dimensions are.

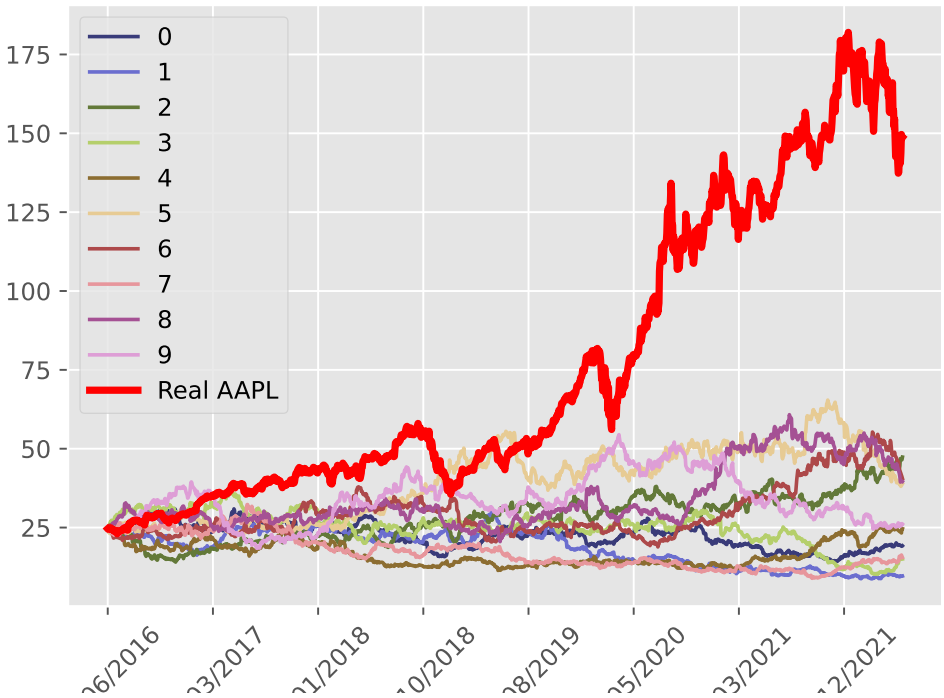


Figure 8: Ten examples of generated paths

5 Predictive pricing methods

5.1 Training set - VaR scenarios

According to (2.4), the interpolation error committed by the projection operator P_k (2.2), defined on a set X, is driven at any point z by the quantity $D_k(z, X)$. We plot at Figure 9 the isocontours of this error function for two distinct sets (red dots).

- (right) X is generated as VaR scenarios for three dates $t^0 - 1, t^0, t^0 + 1$, with 10 days horizon.
- (left) X is the historical data set.

The test set is generated as VaR scenarios with 5 days horizon (blue dots).

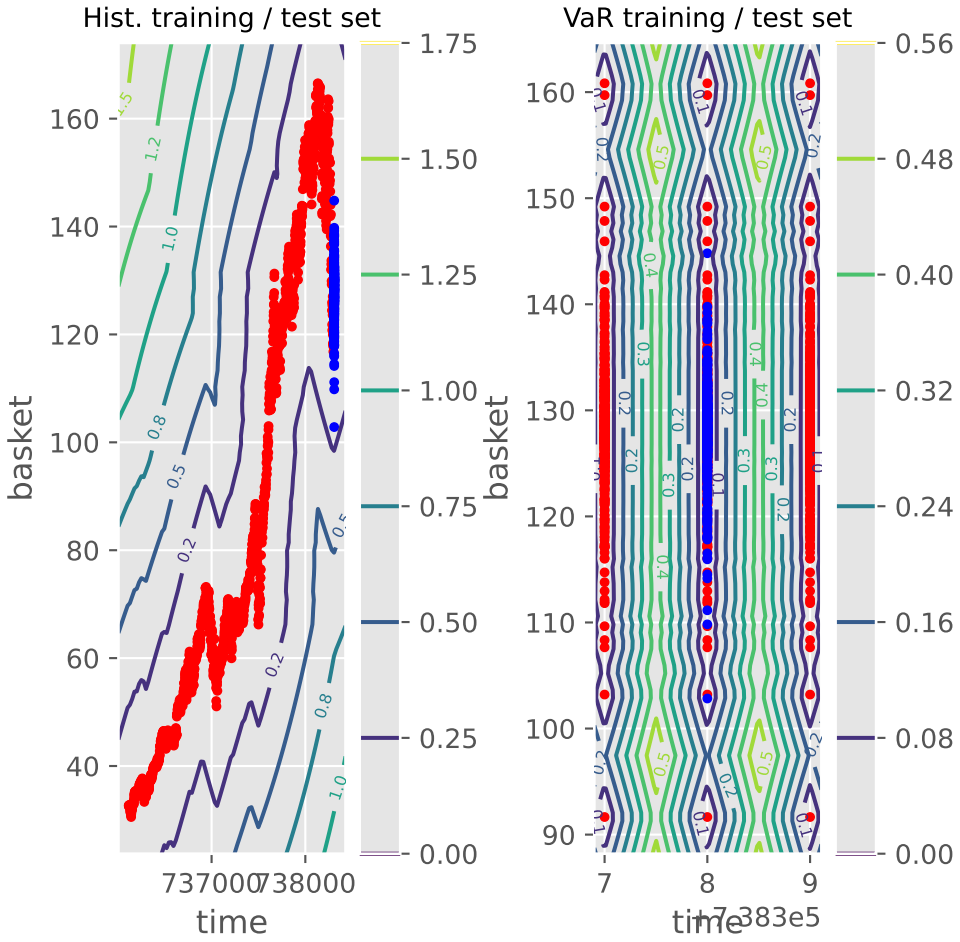


Figure 9: Training and test set

The blue dots in Figure 9 is the test set Z, and corresponds to simulated, intraday, market values. This figure motivates the VaR-type scenario dataset on the left-hand side to minimize the interpolation error. Note that using the historical data set, might be of interest, if only historical data are available.

Notice finally that there are three sets of red points at Figure 9-(a), as we considered VaR scenarios at three different times $t^0 - 1, t^0, t^0 + 1$, because we are interested in approximating time derivatives for risk management, as the theta $\partial_t P$.

5.2 Predict prices

We plot the results of two methods to extrapolate the pricer function on the test set Z (codpy = kernel prediction, taylor = Taylor second order approximation) in Figure 10.We also plot the reference price (exact = reference price).

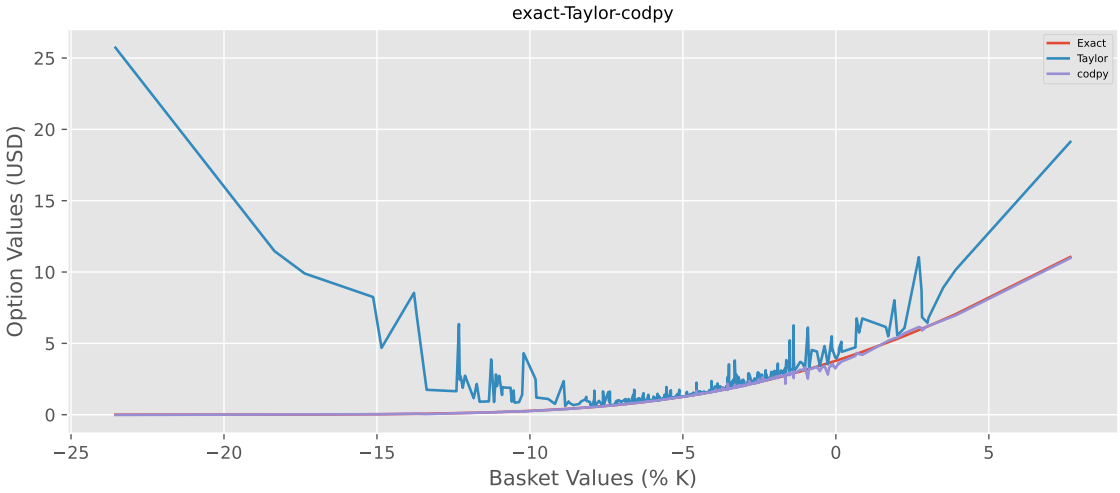


Figure 10: Prices output

5.3 Predict greeks

We can also compute greeks, using the operator $(\nabla P)_Z$ defined at (2.3). Here too, we plot the results of two methods to extrapolate the gradient of the pricer function on the test set Z (codpy = kernel prediction, taylor = Taylor second order approximation) in Figure 11. We also plot the reference greeks (exact = reference greeks). This figure should thus produce $(\nabla P)_Z = \big((\partial_t P)_Z, (\partial_{x_0} P)_Z, \dots, (\partial_{x_D} P)_Z\big)$, that are $D+1$ plots.

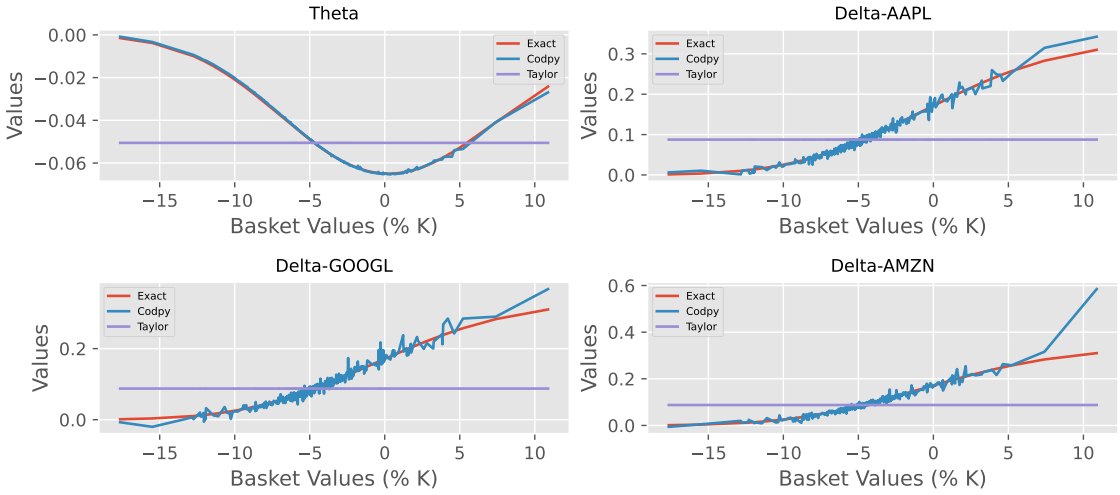


Figure 11: Greeks output

6 Conclusions

6.1 What are the main points highlighted by this presentation

- Synthetic data generation is a general, very handy tool, allowing to model and resample not only any given random variables, but also conditionally to other variables. For instance, we use our algorithms to generate synthetic risk measures conditioned to customers data.
- In this presentation, we start exploring a quite interesting application to risk modelling : we can revisit existing diffusion models, based usually on simple processes as Brownian motions, and propose a general method to calibration. Doing so, we hope to model risk sources more accurately.
- Predictive methods allow to approximate computationally expensive risk valuation functions by learning them from quite few discrete examples. This allows to build fast, real-time, pricing solutions, even for huge portfolios.
- However, one must be very careful as predictive methods, in the context of synthetic data generation, can be quite challenging, particularly while computing derivatives (greeks). We provide solutions, as clustering methods, to enhance the accuracy of such indicators.

6.2 Going further

- As we can resample from historical data, calibrating these data to quite general model, we can generate our own Monte-Carlo pricers built on top of these models.
- In the same vein, we can also build high dimensional PDE pricers using these models, a technology similar to Cox trees, but working whatever the number of risk sources are.
- PDE pricers avoid the "Monte-Carlo of Monte-Carlo" trap for risk-management systems. They allow to estimate risk measures as EEPE, or CVA, in very efficient manner.
- This presentation can be seen as a toy-prototype of a risk management system based on these ideas.

References

[1] A. Gretton, K.M. Borgwardt, M. Rasch, B. Schölkopf, and A.J. Smola, A kernel method for the two sample problems, Proc. 19th Int. Conf. on Neural Information Processing Systems, 2006, pp. 513–520.

[2] A. Berlinet and C. Thomas-Agnan, *Reproducing kernel Hilbert spaces in probability and statistics,* Springer Science, Business Media, LLC, 2004.

[3] Bernhard Schölkopf, Ralf Herbrich, and Alexander J. Smola. A generalized representer theorem. In Computational learning theory, pages 416?426. Springer, 2001.

[4] LeFloch, Philippe G. and Mercier, Jean-Marc and Miryusupov, Shohruh CodPy: A Python Library for Machine Learning, Mathematical Finance, and Statistics, (April 6, 2022). Available at SSRN: https://ssrn.com/abstract=4077158 or http://dx.doi.org/10.2139/ssrn.4077158. CoDpy is available at https://pypi.org/project/codpy/

[5] P.G. LeFloch and J.-M. Mercier, Mesh-free error integration in arbitrary dimensions: a numerical study of discrepancy functions, *Comput. Methods Appl. Mech. Engrg.* 369 (2020), 113245.

[6] P.G. LeFloch and J.-M. Mercier, The transport-based mesh-free method: a short review, *Wilmott* vol. 2020, iss. 109, p. 52–57, 2020.

[7] Eckerli, Florian and Osterrieder, Joerg, Generative Adversarial Networks in finance: an overview, *Comput. Methods Appl. Mech. Engrg.* http://dx.doi.org/10.48550/ARXIV.2106.06364 (2021).