

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

Jean-Marc Mercier [‡] *

09 11 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

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

This presentation is structured as follows :

Contents

1	Introduction	1
2	A quick tour to Artificial Intelligence - Five CoDpy python methods	3
2.1	Predictive methods	3
2.1.1	Training sets and test sets	3
2.1.2	Predictive methods	4
2.1.3	Benchmark of predictive methods	5
2.1.4	Kernel projection operator	6
2.2	Error estimates with kernels : Dcrepancies and norms	7
2.3	Clustering method with kernels : sharp discrepancy sequences	8
2.3.1	Application of clustering methods I : classification	9
2.3.2	Application of clustering methods II : computations enhancement	9
2.4	Generative methods with kernels - the sampler method	10
2.4.1	Application of generative methods : produce samples of any distribution . .	11
2.5	Conditional Expectation Algorithms	12
3	A toy risk framework	13
3.1	Application settings - Input data	13
3.1.1	Retrieve market data	13
3.1.2	Set a portfolio of instruments	14
3.1.3	Set a pricing engine	15
3.2	Synthetic market data generation	16
3.2.1	Fit a model to input data	16
3.2.2	Generate the distribution	17
3.2.3	Check generated distribution	17
3.2.4	Build and check generated paths	18
3.3	Predictive pricing methods	19
3.3.1	Training set - VaR scenarios	19
3.3.2	Predict prices	20
3.3.3	Predict greeks	21
4	Conclusions	22
4.1	What are the main points highlighted by this presentation	22
4.2	Going further	22

2 A quick tour to Artificial Intelligence - Five CoDpy python methods

2.1 Predictive methods

2.1.1 Training sets and test sets

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.

A simple example is given below : a collection of 100 one dimensional points X lying on the segment $[-1,1]$, together with their values given by a sinusoidal function $f(X)$. The test set are points on the segment $[-1.5,1.5]$, allowing to check extrapolations behaviors.

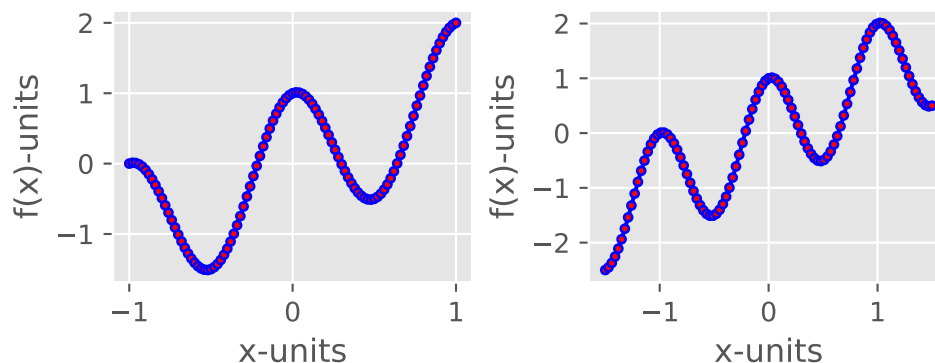


Figure 1: training set, test set and ground truth values for a one dimensional example.

Another example of more elaborate training set : the MNIST database, composed of 60,000 images defining a training set of handwritten digits. Each image is a vector having dimensions 784 (a 28×28 grayscale image flattened in row-order), with their labels, $0 \sim 9$. The test set is composed of 10,000 images.

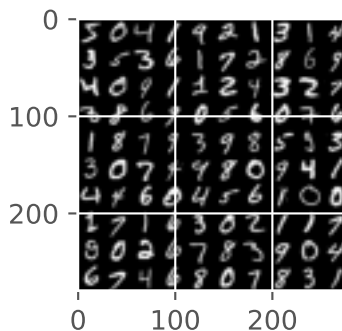


Figure 2: MNIST - distribution of 60 000 hand-written digits

2.1.2 Predictive methods

There exists a lot of predictive machines or algorithms. To unify them, a predictor, denoted by $\mathcal{P}_m$, can be described as an extrapolation or interpolation operator, that defines a prediction as follows:

$$f_Z = \mathcal{P}_m(X, Y = \square, Z = X, f(X)). \quad (2.1)$$

f_Z , the prediction, has to be compared to $f(Z)$, the ground truth values.

Codpy provides facilities to compare methods, as illustrated in these figures, retrieved from our user manual for the one-dimensional problem.

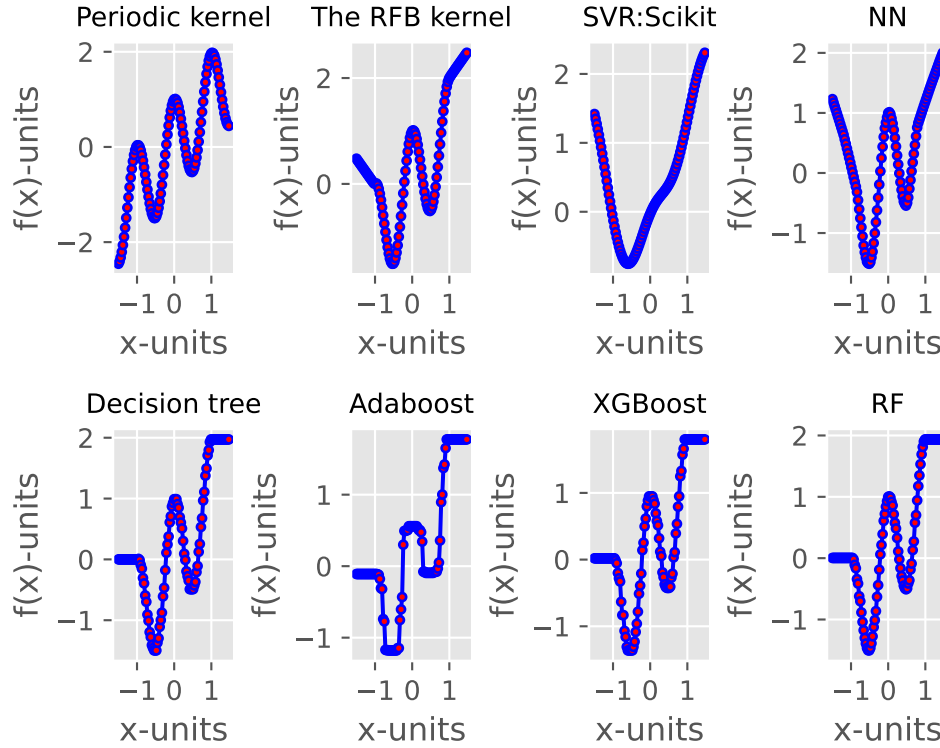


Figure 3: Example of predictions : Periodic kernel:CodPy, the RBF kernel: SciPy, SVR: Scikit, Neural Network: TensorFlow, Decision tree: Scikit, Adaboost: Scikit, XGBoost, Random Forest: Scikit

2.1.3 Benchmark of predictive methods

Once a prediction f_Z is made, one want to compare to the ground truth values $f(Z)$, using some metrics, here a confusion matrix for the MNIST problem for codpy prediction method

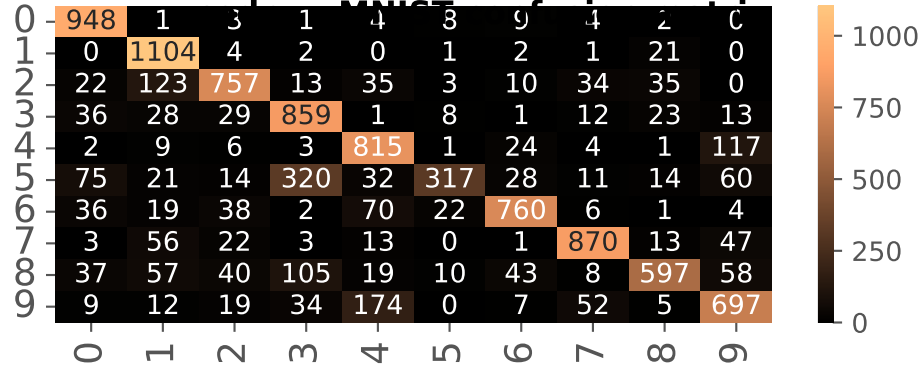


Figure 4: Confusion matrix for codpy: Classifier

For classification problems as MNIST, scores is a normalized metric (the higher the better) :

$$0 \leq \frac{\#\{f_Z == f(Z)\}}{N_Z} \leq 1$$

Below are scores for the MNIST for several projection methods for different settings, corresponding to varying training set sizes.

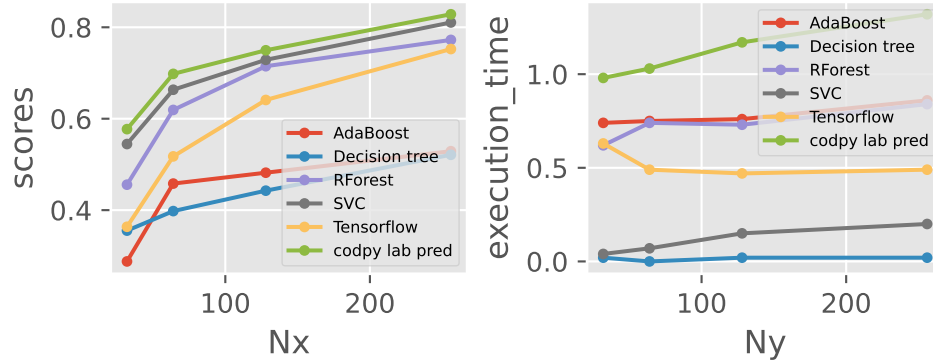


Figure 5: Scores and execution time

2.1.4 Kernel projection operator

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

$$X := \{x_d^n\}_{n,d=1}^{N_x,D} \in \mathbb{R}^{N_x,D}, \quad f(X) \in \mathbb{R}^{N_x,D_f} \quad (2.2)$$

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

$$f_Z := \mathcal{P}_k(X, Y, Z)f(X), \quad \mathcal{P}_k(X, Y, Z) := K(Y, Z)K(X, Y)^{-1}. \quad (2.3)$$

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 (2.3), we can define all kind of differential operators, as for instance the gradient

$$\nabla f_Z = (\nabla_Z K)(Y, Z)K(X, Y)^{-1}f(X). \quad (2.4)$$

CoDpy function:

$$f_z = \text{op.projection}(X, Y, Z, f(X), k, \dots), \nabla f_Z = \text{op.nabla}(X, Y, Z, f(X), k, \dots)$$

.

2.2 Error estimates with kernels : Dcrepancies and norms

The projection operator (2.3) benefits from the following error estimate (see [5]), that are confidence levels

$$\|f(Z) - f_Z\|_{\ell^2} \leq D_k(X, Y, Z) \|f\|_{\mathcal{H}_k}, \quad (2.5)$$

where $D_k(X, Y, Z) := D_k(X, Y) + D_k(Y, Z)$, and where the discrepancy (called also MMD - Maximum Mean Discrepancy) between two discrete probability measures X and Y , induced by a kernel k is

$$D_k(X, Y)^2 := \frac{\sum_{n,m=1}^{N_x} k(x^n, x^m)}{N_x^2} + \frac{\sum_{n,m=1}^{N_y} k(y^n, y^m)}{N_y^2} - \frac{2 \sum_{n,m=1}^{N_x, N_y} k(x^n, y^m)}{N_x N_y}, \quad (2.6)$$

CoDpy function:

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

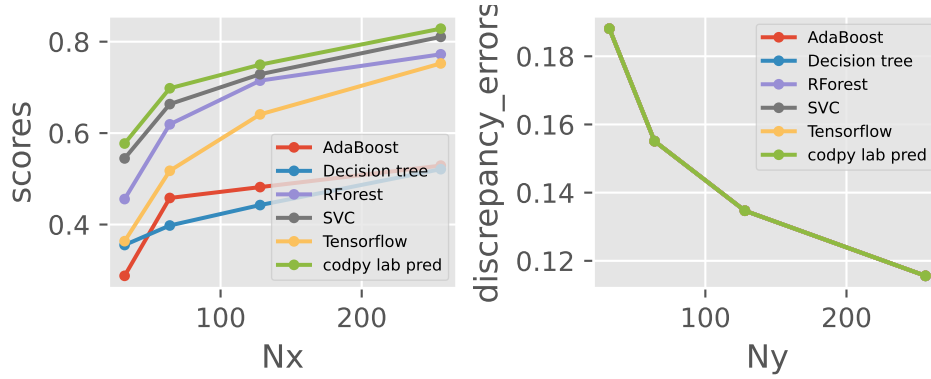


Figure 6: Scores and execution time

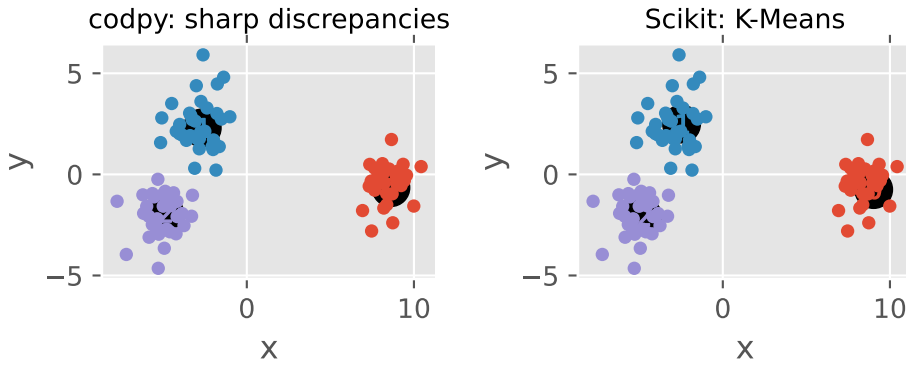
2.3 Clustering method with kernels : sharp discrepancy sequences

CoDpy defines a clustering method as follows:

$$\bar{Y} = \arg \inf_{Y \in \mathbb{R}^{D, N_Y}} D_k(X, Y), \quad (2.7)$$

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 Application of clustering methods I : classification

A natural applications of clustering methods are classification (unsupervised learning). Here is an example of classification of 60 stock market into 10 classes, where the training set X are daily stock price movements $X \in \mathbb{R}^{N_x \times D}$ (i.e. the dollar difference between the closing and opening prices for each trading day) from 2010 to 2015.

Table 1: Stock's clustering

	Scikit:k-means	CoDpy: sharp disc.
0	Amazon, Canon, Cisco, Ford, Google/Alphabet, Honda, HP, Mitsubishi, SAP, Sony, Sanofi-Aventis, Toyota, Xerox	AIG, American express, Bank of America, Goldman Sachs, JPMorgan Chase, Wells Fargo
1	American express, Boeing, British American Tobacco, DuPont de Nemours, Dell, General Electrics, GlaxoSmithKline, Home Depot, IBM, Intel, Lookheed Martin, MasterCard, McDonalds, 3M, Microsoft, Northrop Grumman, Novartis, Symantec, Total, Taiwan Semiconductor Manufacturing, Texas instruments, Unilever, Yahoo	Colgate-Palmolive, Johnson & Johnson, Kimberly-Clark, Coca Cola, Pepsi, Procter Gamble, Wal-Mart
2	AIG, Bank of America, Goldman Sachs, JPMorgan Chase, Wells Fargo	Apple, Caterpillar, ConocoPhillips, Chevron, DuPont de Nemours, General Electrics, IBM, 3M, Pfizer, Schlumberger, Valero Energy, Exxon
3	Colgate-Palmolive, Johnson & Johnson, Kimberly-Clark, Coca Cola, Pepsi, Pfizer, Procter Gamble, Philip Morris, Walgreen, Wal-Mart	Amazon, Boeing, Canon, Cisco, Dell, Ford, Google/Alphabet, Honda, HP, Intel, Lookheed Martin, Mitsubishi, Navistar, Northrop Grumman, Sony, Toyota, Taiwan Semiconductor Manufacturing, Texas instruments, Xerox, Yahoo
4	Apple, Caterpillar, ConocoPhillips, Chevron, Navistar, Royal Dutch Shell, Schlumberger, Valero Energy, Exxon	British American Tobacco, GlaxoSmithKline, Home Depot, MasterCard, McDonalds, Microsoft, Novartis, Philip Morris, Royal Dutch Shell, SAP, Sanofi-Aventis, Symantec, Total, Unilever, Walgreen

2.3.2 Application of clustering methods II : computations enhancement

Another quite interesting application to clustering methods are computations enhancement : recall the discrepancy error (2.5):

$$\|f(Z) - f_Z\|_{\ell^2} \leq (D_k(X, Y) + D_k(Y, Z)) \|f\|_{\mathcal{H}_k} \quad (2.8)$$

$\mapsto$ taking Y as a clustering of X into N_Y classes allows to boost accuracy of kernel predictions and differential operators with few computational resources as follows:

$$f_Z := \mathcal{P}_k(X, \bar{Y}, Z)f(X), \quad \bar{Y} = \arg \inf_{Y \in \mathbb{R}^{D, N_Y}} D_k(X, Y). \quad (2.9)$$

2.4 Generative methods with kernels - 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( $X, N, k, \dots$ )`

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 7 (resp. Figure ??) for the IID case (resp. SDS case).

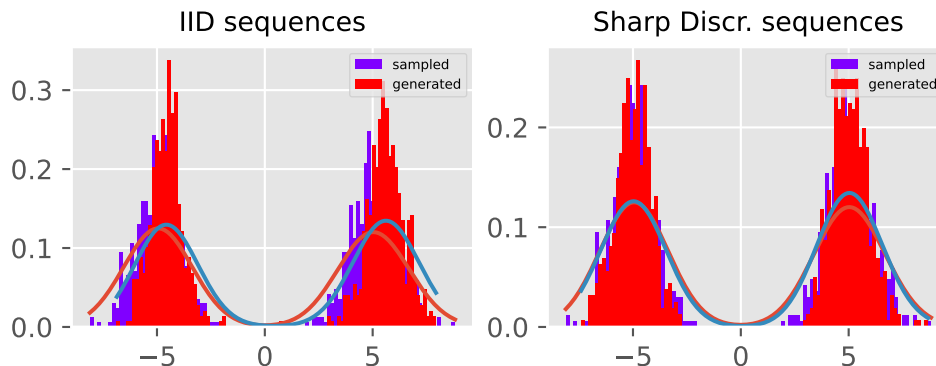


Figure 7: Density of generated IID distributions

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

Table 2: Statistics of IID-generated distributions

	Mean	Variance	Skewness	Kurtosis	KS test
IID sequences	0.047(0.74)	0.012(-0.068)	26(26)	-1.8(-1.9)	7.2e-12(0.05)
Sharp Discr. sequences	0.047(0.2)	0.012(-0.06)	26(26)	-1.8(-1.9)	0.13(0.05)

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

2.4.1 Application of generative methods : produce samples of any distribution

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 $28 \times 28 = 784$ pixels resolution. We consider it as a discrete 784-dimensional distribution having 60000 samples, this figure showing the first hundred ones

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

2

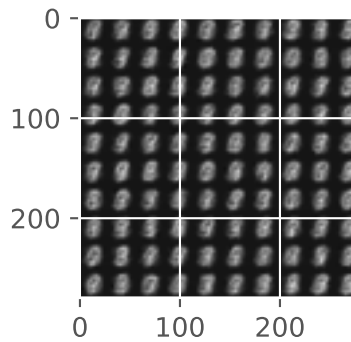


Figure 8: Generated distribution of hand-written digits

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

2.5 Conditional Expectation Algorithms

Consider any martingale process $t \mapsto X(t)$, and any positive definite kernel k , we define the operator Π - using python notations -

$$f_{Z|X} = \Pi(X, Z, f(Z))$$

where

- $X \in \mathbb{R}^{N_x \times D}$ is any set of points generated by a i.i.d sample of $X(t^1)$ where t^1 is any time.
- $Z \in \mathbb{R}^{N_z \times D}$ is any set of points, generated by a i.i.d sample of $X(t^2)$ at any time $t^2 > t^1$.
- $f(Z) \in \mathbb{R}^{N_z \times D_f}$ is any, optional, function.

The output is a matrix $f_{Z|X}$, representing the conditional expectation

$$f_{Z|X} \sim \mathbb{E}^{X(t^2)}(f(\cdot)|X(t^1)) \in \mathbb{R}^{N_x \times D_f} =:^{not.} f(Z|X).(\#eq : CE) \quad (2.10)$$

- if $f(Z)$ is let empty, the output $f_{Z|X} \in \mathbb{R}^{N_z \times N_x}$ is a matrix, representing a convergent approximation of the stochastic matrix $\mathbb{E}^{X(t^1)}(Z|X)$.
- if $f(Z) \in \mathbb{R}^{N_z \times D_f}$ is not empty, $f_{Z|X} \in \mathbb{R}^{N_z \times D_f}$ is a stochastic matrix, representing the conditional expectation $f(Z|X) := \mathbb{E}^{X(t^1)}(f(Z)|X)$.

$$\text{alg.Pi}(X, Z, f(Z))$$

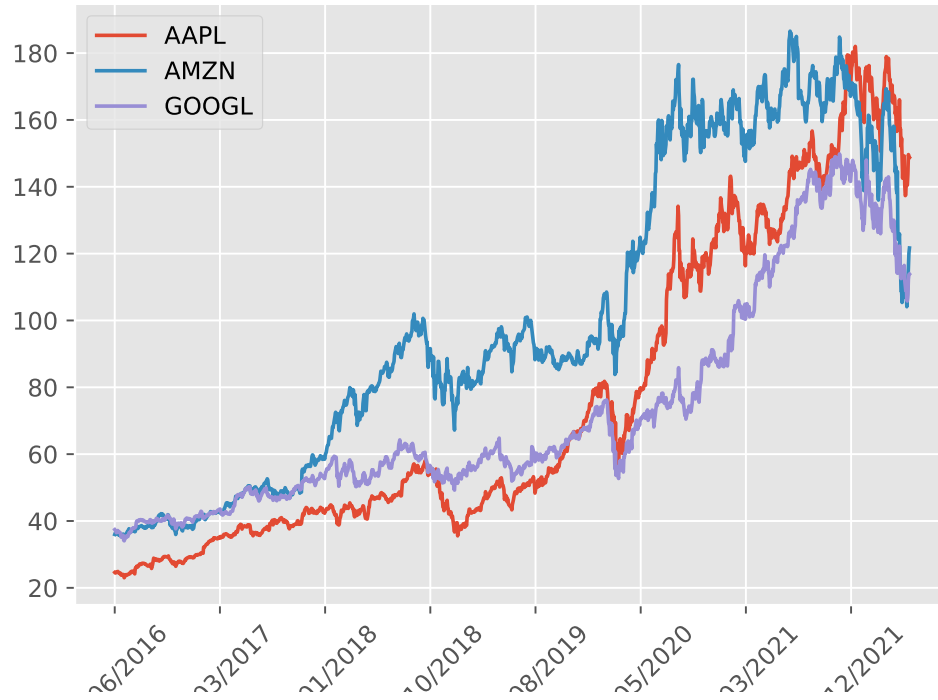
Note : this algorithm is an efficient alternative to **Sinkhorn** algorithm.

3 A toy risk framework

3.1 Application settings - Input data

3.1.1 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	AAPL , GOOGL, AMZN

3.1.2 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 9.

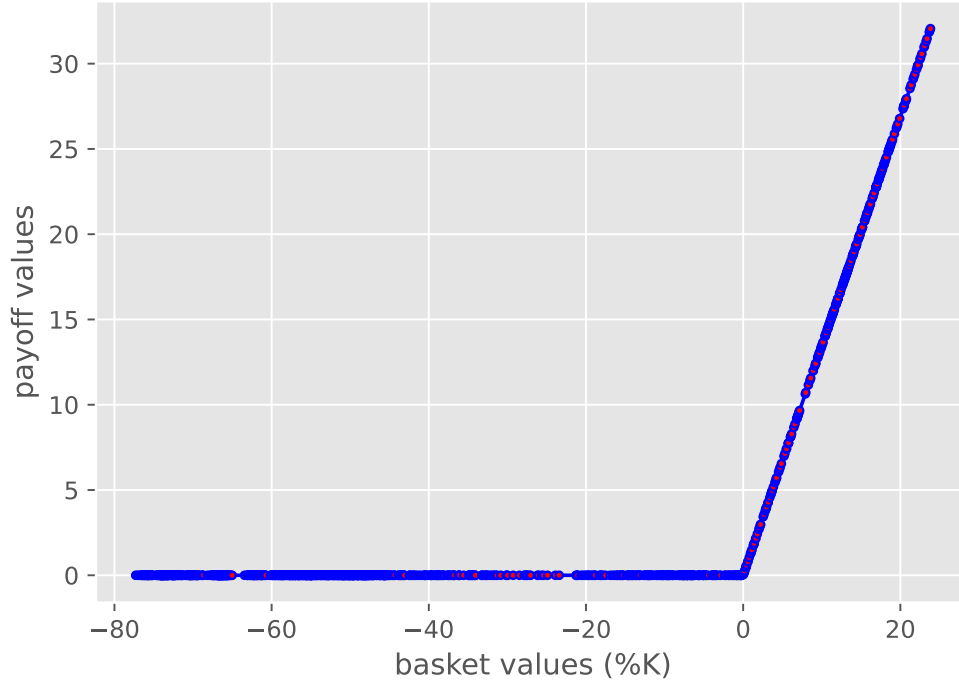


Figure 9: A payoff of an basket option

3.1.3 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 10 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

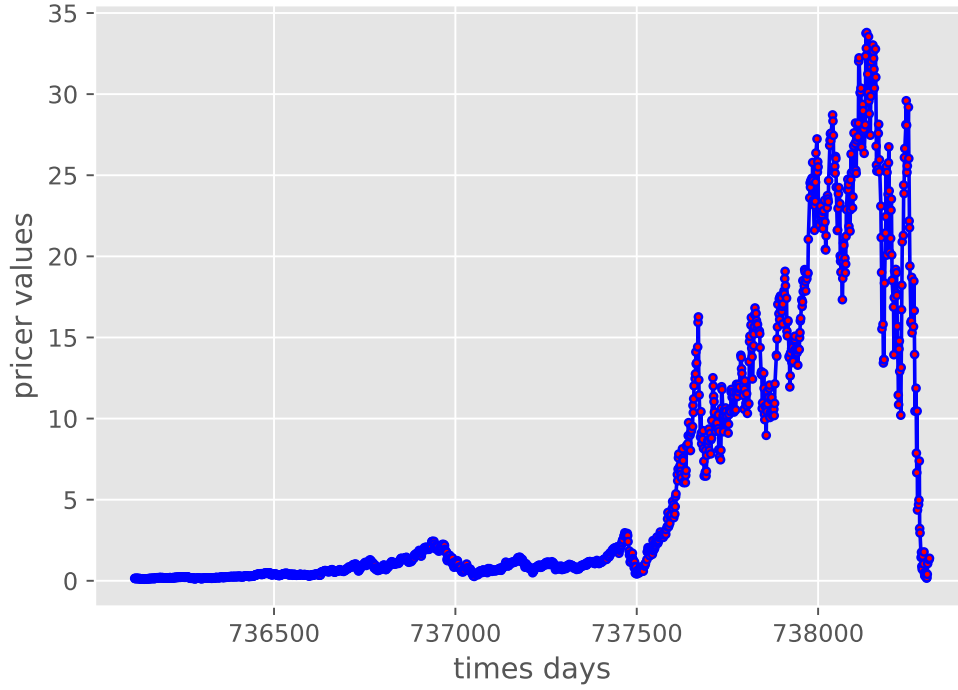


Figure 10: Pricing as a function of time

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

3.2 Synthetic market data generation

3.2.1 Fit a model to input data

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.

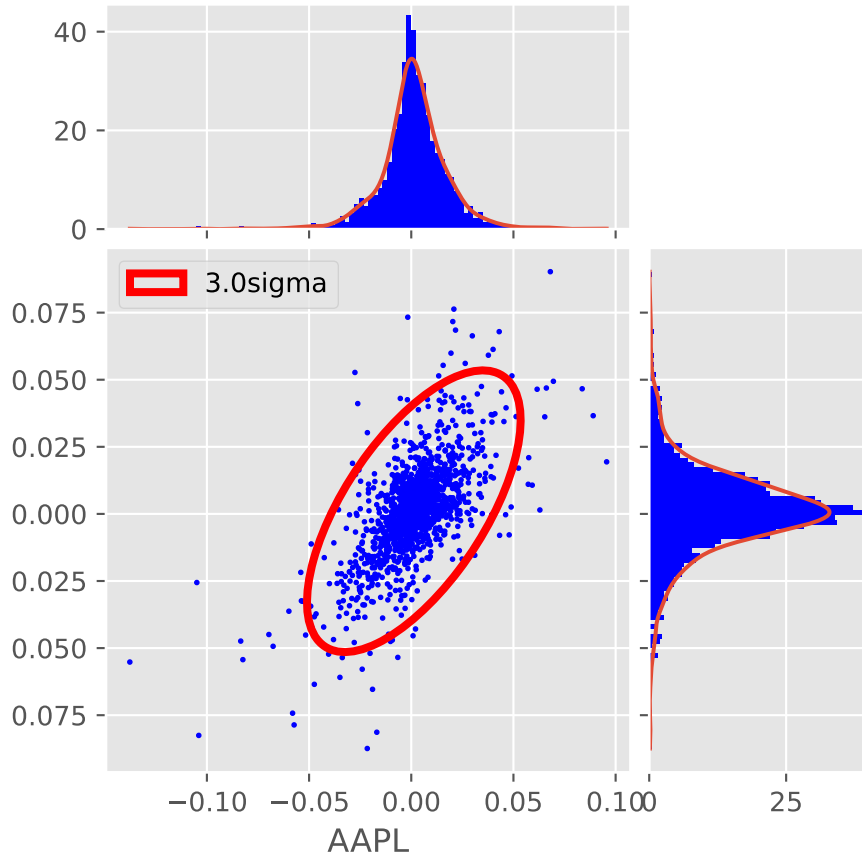


Figure 11: Log return distribution of historical market data

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

$$X_t = X_s \exp((t-s)\mu + \sqrt{t-s}\mathbb{X}), \quad (3.1)$$

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

$$\mathbb{X} = \frac{\ln(X_t) - \ln(X_s) - (t-s)\mu}{\sqrt{t-s}}, \quad (3.2)$$

in order to separate the random variable $\mathbb{X}$ to fit to historical data. Figure 11 plots the ditribution retrieved from our historical data after the transformation (3.1).

3.2.2 Generate the distribution

We then provide a function, producing a continuous sampling function from any discrete input distribution. Figure 12 is a result of a resample of the historical distribution

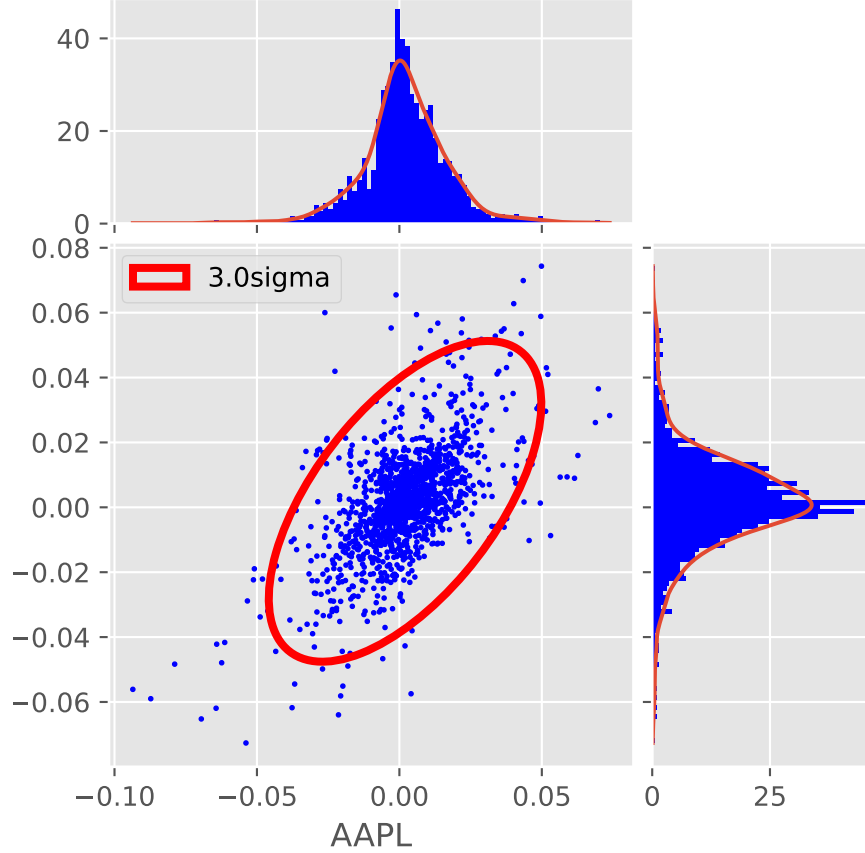


Figure 12: Log return distribution of generated market data

3.2.3 Check generated distribution

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)

3.2.4 Build and check generated paths

Ten examples of re-generated paths in figure 13. These paths can be used for Monte-Carlo sampling, and we can also build PDE (Partial Differential Equations) pricers, whatever the dimensions are.

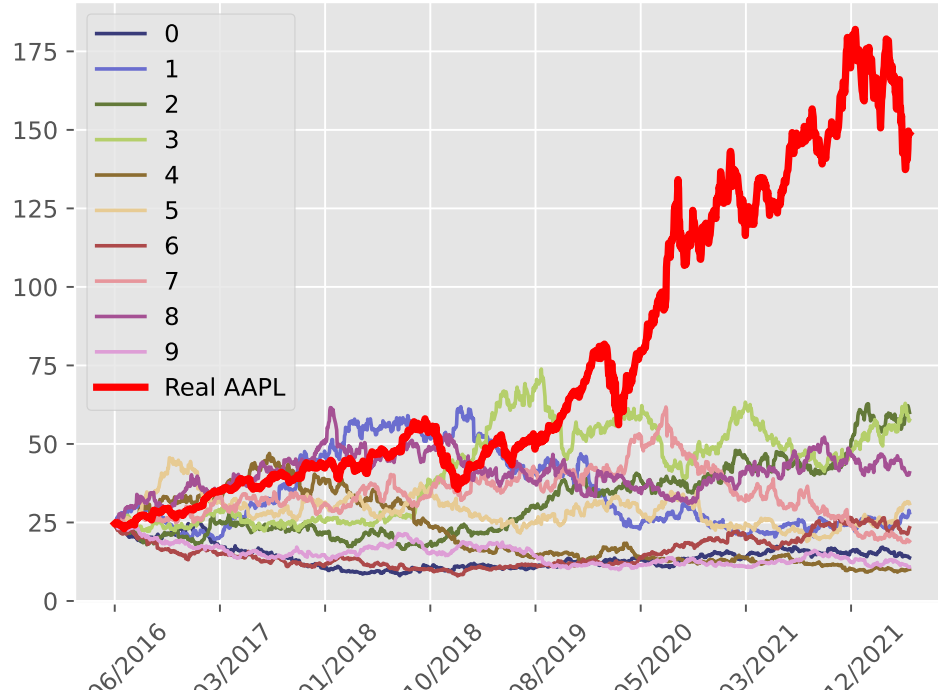


Figure 13: Ten examples of generated paths

3.3 Predictive pricing methods

3.3.1 Training set - VaR scenarios

According to (2.5), the interpolation error committed by the projection operator P_k (2.3), defined on a set X , is driven at any point z by the quantity $D_k(z, X)$. We plot at Figure 14 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).

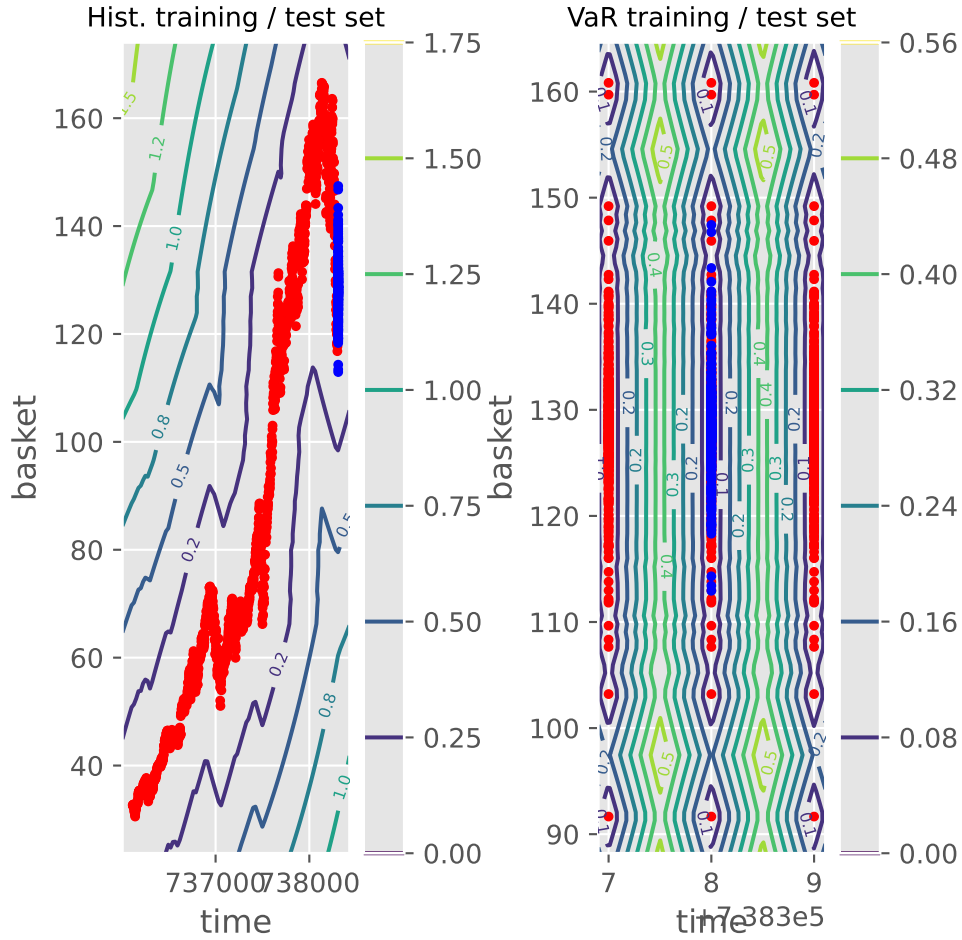


Figure 14: Training and test set

The blue dots in Figure 14 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 14-(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 $\partial_t P$.

3.3.2 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 15. We also plot the reference price (exact = reference price).

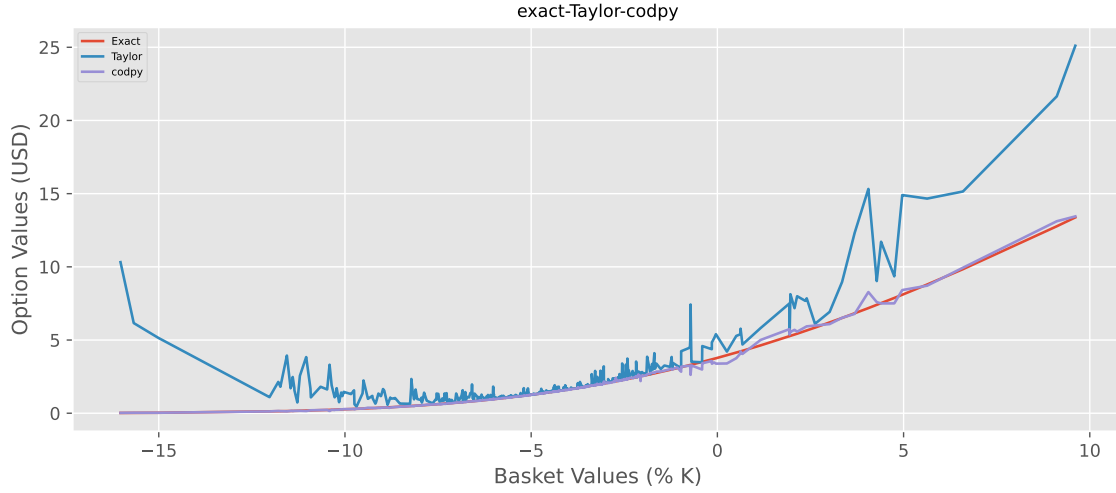


Figure 15: Prices output

3.3.3 Predict greeks

We can also compute greeks, using the operator $(\nabla P)_Z$ defined at (2.4). 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 16. We also plot the reference greeks (exact = reference greeks). This figure should thus produce $(\nabla P)_Z = ((\partial_t P)_Z, (\partial_{x_0} P)_Z, \dots, (\partial_{x_D} P)_Z)$, that are $D + 1$ plots.

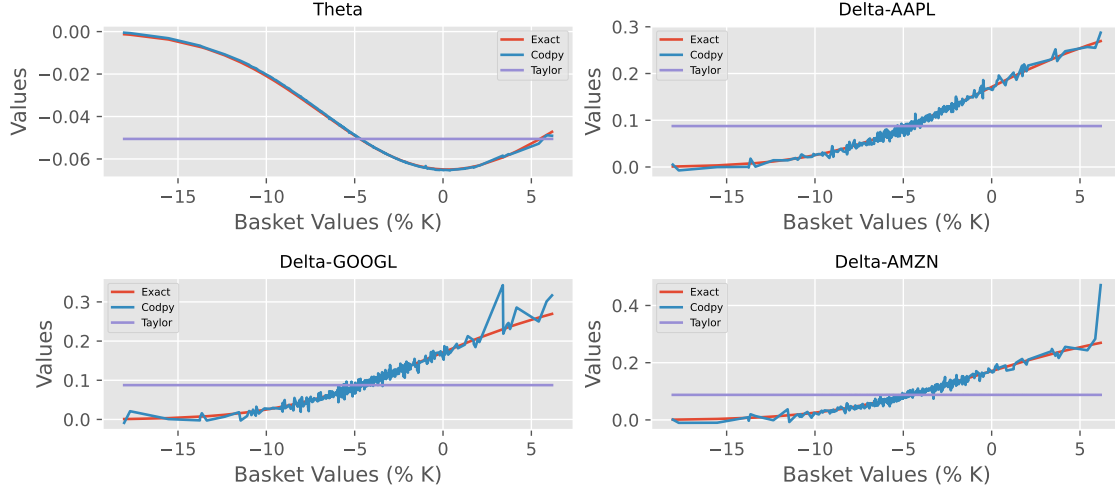


Figure 16: Greeks output

Note that deltas present spurious oscillations. However, using some engineering, we can smooth them out

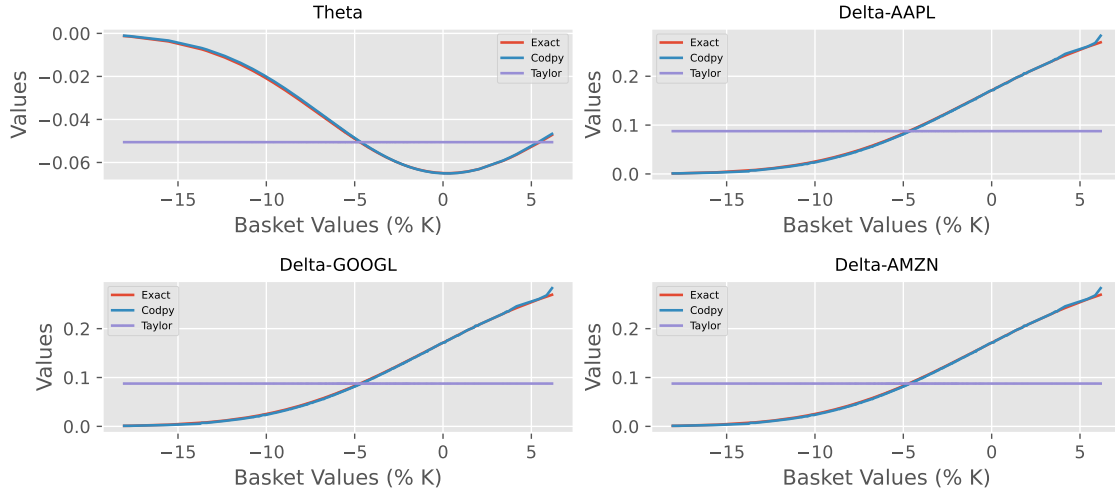


Figure 17: Greeks output after correction

4 Conclusions

4.1 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 revisited an existing diffusion model, based on a simple Brownian motions, and propose a general method to calibrate it. 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.

4.2 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).