Generative modeling Project

Course "Monte Carlo Methods: from MCMC to Data-based Generative Model" M2 Data Science

Gérard Théo
ENSAE Paris
theo.gerard@ensae.fr

Le Clerc Florent
ENSAE Paris
florent.leclerc@ensae.fr

December 2023

Contents

1 Introduction

The recent literature has delved into the generation of extreme events, primarily focusing on meteorology, as seen in Peard and Hall 2023. This exploration holds significant importance as it allows us to contemplate unlikely yet potential occurrences that could yield drastic consequences if not anticipated.

This project, however, shifts its focus to the generation of financial tabular data. Within the existing literature, many papers on generative models for tabular data gravitate towards GANs, exemplified by Xu et al. 2019 and Zhang et al. 2021. On the flip side, alternative generative methods like diffusion models, as discussed in Kotelnikov et al. 2023, and VAEs, as explored also in Xu et al. 2019, have also been considered. Notably, this project will exclusively delve into GAN-based methods.

Adding to the complexity, the dataset under scrutiny contains minimal extreme data. Consequently, the challenge lies in generating data that closely align with the original distribution. This entails incorporating some extreme values into the generated data, albeit in moderation. To address this hurdle, the project will delve into the study of three distinct methods. These methods will also be compared with a conventional GAN. The code containing the implementations can be found on `https://github.com/GerardTho/Project_GAN_Extreme_values`.

More precisely, the data we consider in this project contains $n = 746$ i.i.d. financial log-returns of $d = 4$ different assets in $\mathbb{R}^+$. The distribution of the dataset is shown in figure 1.

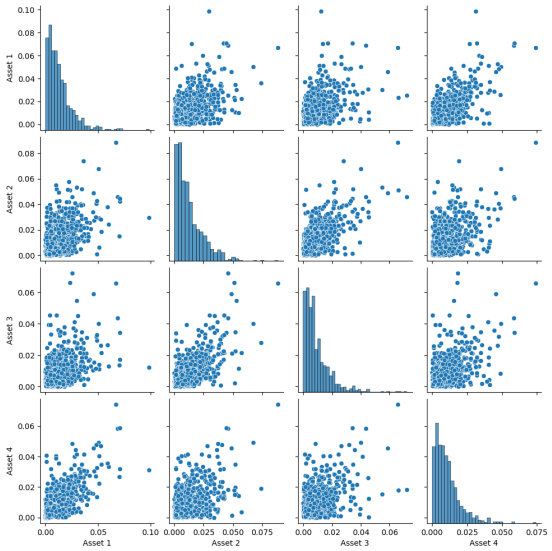


Figure 1: Distribution of the dataset

2 Methods

In this section, after talking about the standard GANs, methods drawn from three papers will be considered. These three methods will be summarized from the papers, and some figures and algorithms will be copied as they are from the papers. The reader will therefore note that the methods studied are merely applications of already existing methods to the framework of the problem.

2.1 Initial method: a standard GAN

As it was explained before, all the methods we will study in this project are based on GANs. This generative model seemed natural since most of the literature on generative models for tabular data considered also GANs. Even if some papers also studied VAE, in the scope of this project, it didn't seem relevant to study this model since the dimension of the output is already really small ($d = 4$).

In this section, some implementation considerations of the basic GAN will be studied. As it was presented in the lecture about GANs, even if these are really powerful models, GANs tend to have stability problems. As advised by A. Durmus, this project takes into account all the "GAN hacks" described in the advised GitHub repository: `https://github.com/soumith/ganhacks`. Among these

numerous hacks, some of them are, for instance: using dropout in both train and test, considering noisy labels, flipping labels sometimes, decaying noise over time...

Yet, although all these hacks increased dramatically the performance of the GANs, adding batch normalization was associated with lower performance. What's more, some hacks are not an option for some of the methods described. For instance, the second and the third methods considered proper input for the GANs that are not necessarily in $[-1, 1]$.

2.2 First method: distribution shifting

The first method is based on the first algorithm described in Bhatia, Jain, and Hooi 2021. This method considers a conventional GAN, but change iteratively the data on which the GAN is trained.

As described in the article, the goal of the algorithm is to repeatedly shift the distribution by filtering away the less extreme proportion of the data $(1 - c)$. In detail, the original dataset $\mathcal{X}$ of size n is first sorted out in decreasing order of extremeness, initializing the shifted dataset $\mathcal{X}_s$. Then, at each iteration i, a GAN (with generator G) is fitted on $\mathcal{X}_s$. Then, $\mathcal{X}_s$ is truncated by its top $\lfloor c^i \cdot n \rfloor$ extreme values. Next, $\lceil (n - \lfloor c^i \cdot n \rfloor) \cdot \frac{1}{c} \rceil$ are generated using the previously trained G, and the most extreme $n - \lfloor c^i \cdot n \rfloor$ are inserted into $\mathcal{X}_s$.

Hence, the algorithm is parameterized by: a shift parameter $c \in]0, 1[$, an iteration count $k \in \mathbb{N}$, and an extremeness measure E. For the extremeness measure, as the data is positive-valued, a natural choice is to consider the sum along the four dimensions. Considering the parameters c and k, several values were studied in this project. Large values of c (close to 1) means that the shift is slower. Indeed, for large values of c, an important sample of the original dataset is saved, and only few generated data are inserted. As discussed by the author, a large c is preferable to help the neural network to smoothly interpolate and adapt to the shifted distribution. Nevertheless, larger values of c imply larger values of k to get the same shifting, meaning an increasing training time. The algorithm described in the article is returned below.

Algorithm 1: Distribution Shifting

Data: dataset $\mathcal{X}$, extremeness measure E, shift parameter c, iteration count k

`// Sort` $\mathcal{X}$ `in decreasing order of extremeness`

Sort $\mathcal{X}$ in decreasing order of E;

`// Initialize` $X_s \leftarrow X$

Initialize $X_s \leftarrow X$;

for $i \leftarrow 1$ **to** k **do**

  `// Shift the data distribution by a factor of` c

  Train GAN G and D on $\mathcal{X}_s$;

  $X_s \leftarrow$ top $\lfloor c^i \cdot n \rfloor$ extreme samples of $\mathcal{X}$;

  Generate $\lceil (n - \lfloor c^i \cdot n \rfloor) \cdot \frac{1}{c} \rceil$ data points using G;

  Insert most extreme $n - \lfloor c^i \cdot n \rfloor$ samples into $\mathcal{X}_s$;

2.3 Second method: evtGAN

The second method is based on the algorithm described in Boulaguiem et al. 2022. As for the first method, a conventional GAN is considered.

The first trick of this method is to train the model on the maximum by block of the original dataset. Instead of training on the data $X_1, \dots X_n$, a block size k is defined and the model is trained on $Z_1, \dots Z_m$ with $Z_1 = \max(X_1, \dots X_k)$..., and $n = k \cdot m$ (some values of (X_i) might not be considered to ensure a constant number of samples for each block).

Then, the authors suppose that for each dimension $j \in [\![1, d]\!]$, the (Z_i) can be approximated by a generalized extreme value distribution [1]. This distribution is then fitted to the (Z_i), and the CDF of this estimated distribution is applied on the (Z_i) for the entry of the GAN.

To retrieve a sample (Z_i) from the original distribution, the generated data are eventually normalized back using the quantile of the estimated GEV distribution. Note that the authors of this paper were initially interested in sampling plausible extreme values rather than just plausible values from a distribution with some extreme values. Hence, the evtGAN algorithm is to sample data from (Z_i) rather than (X_i). Yet, one might think that this algorithm would force to generate more extreme values than the other

[1] This distribution (GEV) is describe in Bali 2003

methods, and considering small blocks would prevent from considering too many of them. The algorithm described in the paper is reported below.

Algorithm 2: evtGAN

Data: Observations $Z_i = (Z_{i1}, \dots, Z_{id})$, $i = 1, \dots, n$

`// Fit GEV distribution to each margin`

for $j = 1$ **to** d **do**

  Fit GEV distribution $\hat{G}_j$ to data $Z_{1j}, \dots, Z_{nj}$ with estimated parameters $\hat{\mu_j}, \hat{\sigma_j}, \hat{\zeta_j}$;

`// Normalize all margins to standard uniform distribution`

for $i = 1$ **to** n **do**

  Normalize margins empirically to standard uniform distribution: $U_i = (\hat{F}_1(Z_{i1}), \dots, \hat{F}_d(Z_{id}))$;

`// Train GAN on normalized data`

Train GAN G on normalized data $U_1, \dots, U_n$;

`// Generate new data points with uniform margins`

Generate n^* new data points $U_1^*, \dots, U_{n^*}^*$ from G with uniform margins;

`// Normalize back to the scale of the original observations`

for $i = 1$ **to** n^* **do**

  $Z_i^* = (\hat{G_1}^{-1}(U_{i1}^*), \dots, \hat{G_d}^{-1}(U_{id}^*))$;

Result: Set of new generated observations $Z_i^* = (Z_{i1}^*, \dots, Z_{id}^*)$, $i = 1, \dots, n^*$

2.4 Third method: EV-GAN

The third and last method studied is based on the paper Allouche, Girard, and Gobet 2022. Contrary to the first two methods, this one deals with the problem of extreme values inside the generator of the GAN. In this section, the main idea of the method will be described, but the extreme value theory which is developed in the paper to justify the formulas won't be considered.

This first consideration of this method is that the universal approximation theorem is valid only for functions on a compact set. Meaning that approximating a distribution on $\mathbb{R}$ implies first to truncate the value. To do this, this paper considers a transformation that fits well for extreme values distribution. This transformation is defined as, for an output of the neural network y and an uniform noise input $u \in [0, 1]$ for the generator of the GAN:

$$H_u : \left| \begin{array}{ccc} \mathbb{R}^+ & \longrightarrow & \mathbb{R} \\ y & \longmapsto & -\frac{\log(y)}{\log(1-u^2)-\log(2)} \end{array} \right.$$

Hence the output of the final neural network should be :

$$f^{TIF} : u \mapsto H_u(q_X(u))$$

Here $q_X(u)$ is the final generated data - the quantile of the original distribution applied on a random uniform, according to the standard inversion method. "TIF" stands for "tail-index function" since $\lim_{u \to 1} f^{TIF}(u) = \gamma$ where γ is the tail-index of the distribution - $q_X(u)$ diverges as $u \mapsto 1$ at rate $(1-u)^{-\gamma}$.

To be more precise, in order to get even better result, the paper consider a corrected tail-index function (CTIF) defined as:

$$f^{CTIF} : \left| \begin{array}{ccc} [0, 1[& \longrightarrow & \mathbb{R} \\ u & \longmapsto & f^{\mathrm{TIF}}(u) - \sum_{k=1}^{6} \kappa_k e_k(u) \end{array} \right.$$

Here, $e_1, \dots e_6$ are universal functions[2] and $\kappa_1, \dots \kappa_6$ are trainable parameters.

The architecture of the whole generator is thus composed of two parts. A first part is a simple neural network which takes as input some uniform random noise, and is trained to produce an output G_θ close to the distribution of f^{CTIF}. Then, this result is added with a regularization part, whose trainable

[2]See the paper for the definition of these functions.

parameter are the κ. Eventually, this output is transformed by H_u^{-1} to give the final sample G_ψ^{TIF}. The formulation of G_ψ^{TIF} is then:

$$G_\psi^{\text{TIF}} : z \mapsto H_z^{-1}\left(G_\theta(z) + \sum_{k=1}^{6} \kappa_k e_k(z)\right) \tag{1}$$

$$\text{with } G_\theta : z \mapsto \sum_{j=1}^{J} a_j \sigma\left(w_j z + b_j\right)$$

$$\text{and} \quad H_z^{-1} : x \mapsto \left(\frac{1-z^2}{2}\right)^{-x}$$

Finally, as the input noise has dimension $d' > 1$ and the output has dimension $d = 4 > 1$, the formulation of 1 has to be slightly modified. The strategy of the paper is then to consider, form $m \in [\![1, d]\!]$ (the dimension of the output):

$$G_\psi^{\text{TIF},(m)} : z \mapsto H_{z^{(m)}}^{-1}\left(G_\theta^{(m)}(z) + \sum_{k=1}^{6} \kappa_k^{(m)} e_k(z^{(m)})\right) \tag{2}$$

In particular, equation 2 implies that $d' > d$, and that, even if every dimension of the z appears in the generation, only the first d one are considered for the inversion of H_z.

The architecture of the network described in the paper is reported in the figure 2.

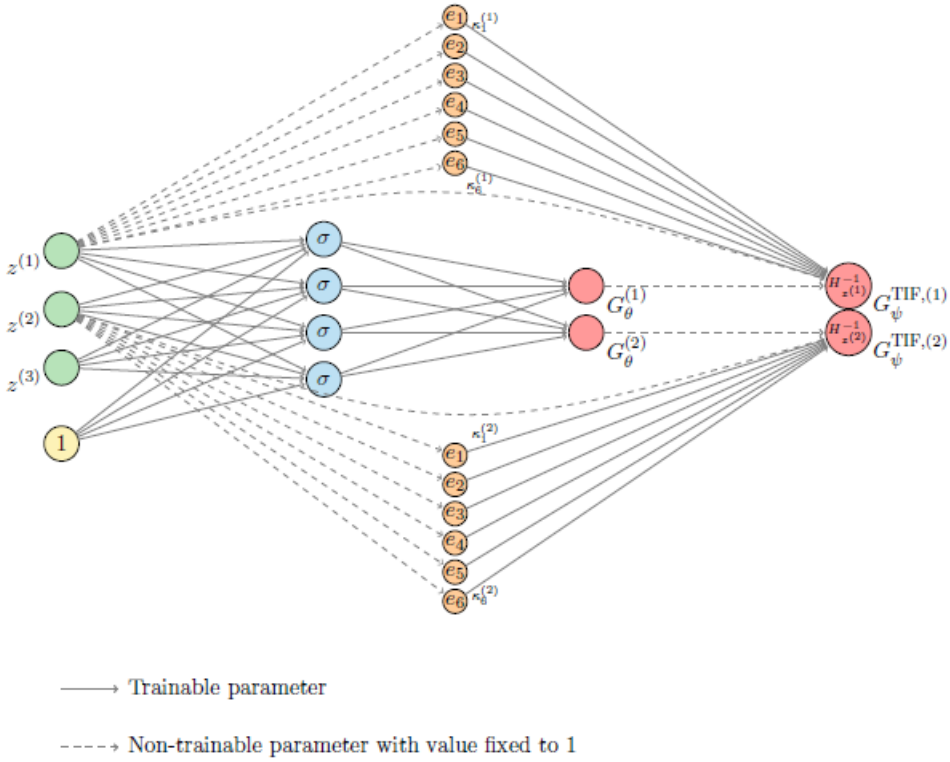


Figure 2: Generator of the EV-GAN with one hidden layer, $d' = 3$ and $d = 2$

3 Results

Now that all the methods have been described, this section will provide the score metrics for all four methods. The score metrics are the Anderson-Darling metrics and the absolute Kendall error, as described in the instruction.

This section will study these metrics with different hyperparameter values for the distribution shifting method (ExGAN) and the evtGAN. For the other parameters, the GANs of all methods were trained

with 100 epochs each time we call the train of a GAN - with no significant improvement for later epochs -, except for the evtGAN where we considered only 30 epochs - it provides better results. The optimizer, as advised in the GitHub repository mentioned before, is ADAM, and the learning rate is 0.0001.

Methods (Parameters)	**AKE**	**MAD**	**Best AKE**	**Best MAD**
GAN	0.05 ± 0.02	5.28 ± 5.82	0.05 ± 0.02	5.28 ± 5.82
ExGAN, c=0.9, k=3	0.06 ± 0.02	1.01 ± 0.85	0.03 ± 0.01	0.91 ± 0.53
ExGAN, c=0.95, k=3	0.07 ± 0.02	$\mathbf{0.77 \pm 0.65}$		
ExGAN, c=0.99, k=3	0.06 ± 0.02	0.94 ± 1.43		
ExGAN, c=0.9, k=5	0.04 ± 0.06	1.16 ± 0.26		
ExGAN, c=0.95, k=5	$\mathbf{0.03 \pm 0.01}$	0.91 ± 0.53		
ExGAN, c=0.99, k=5	0.06 ± 0.02	1.12 ± 0.99		
ExGAN, c=0.9, k=10	0.05 ± 0.01	1.11 ± 1.10		
ExGAN, c=0.95, k=10	0.07 ± 0.03	0.84 ± 1.17		
ExGAN, c=0.99, k=10	0.06 ± 0.01	1.43 ± 0.74		
EvtGAN, block size=1	0.18 ± 0.01	7.08 ± 5.10	0.18 ± 0.01	7.08 ± 5.10
EvtGAN, block size=2	0.18 ± 0.01	24.12 ± 10.72		
EvtGAN, block size=5	0.19 ± 0.02	83.61 ± 11.36		
EvtGAN, block size=10	0.19 ± 0.01	139.99 ± 14.15		
EV-GAN	0.03 ± 0.01	216.35 ± 93.85	0.03 ± 0.01	216.35 ± 93.85

Table 1: Comparison of Methods with Absolute Kendal Error and Marginal Anderson Darling averaged over 5 runs

To begin with, the result of the ExGAN model seems coherent. As the parameter c decreases, the distribution shifting increases as there are more extreme value into the retraining of the GAN each time. Similarly, as the parameter k increases, the distribution shifting increases as the training lasts longer. Even if it is difficult to analyze the Marginal Anderson Darling due to its variability, it seems that it reaches its lowest point for $c = 0.95$. Conversely, for $k = 5$, it seems that the Absolute Kendal Error has better results. Therefore the best model of the ExGAN is found for $c = 0.95$ and $k = 5$.

A similar analysis can be done for the EvtGAN. While the Absolute Kendal Error does not change, the Marginal Anderson Darling is significantly increasing as the size of the blocks is increasing. Indeed, the extreme values are more taken into account if the block size is large, effectively creating a heavier tail in the distribution.

Eventually, it seems that the Evt-GAN and the EV-GAN create too heavy-tailed distribution compared to the original data (see Figure 5 and Figure 6). Using the parameter block size of the Evt-GAN is not sufficient to tailor the distribution to our dataset, as a block size of 1 is already creating a distribution with a big heavy tail. On the other hand, the GAN model doesn't manage to recreate an heavy tailed distribution and is only able to infer a light tail distribution (see Figure 3), as shown for instance by Huster et al. 2021. Graphically, it seems that the ExGAN is the model matching the most the original distribution (see Figure 4).

Furthermore, as shown in figure 7, the Absolute Kendal Error goes very low before going up again, while the Marginal Anderson Darling continues to go down. A similar odd behavior appears during the training of the EV-GAN, where the AKE goes very low but the Marginal Anderson Darling stops at a certain plateau.

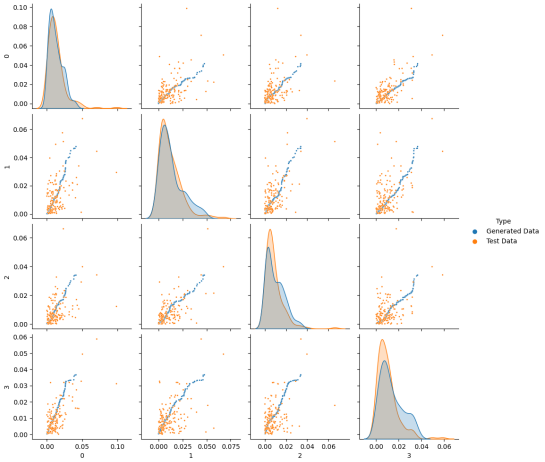


Figure 3: Distributions for data generated by GAN

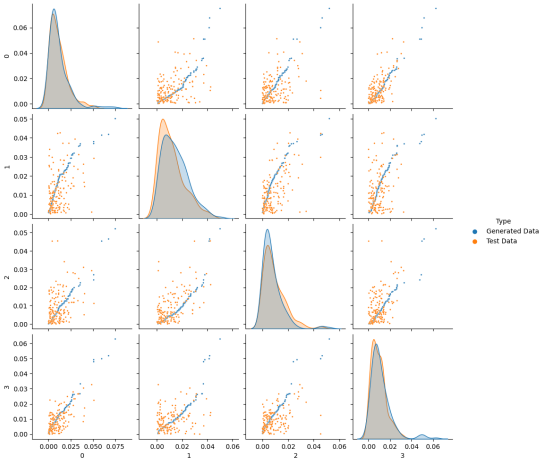
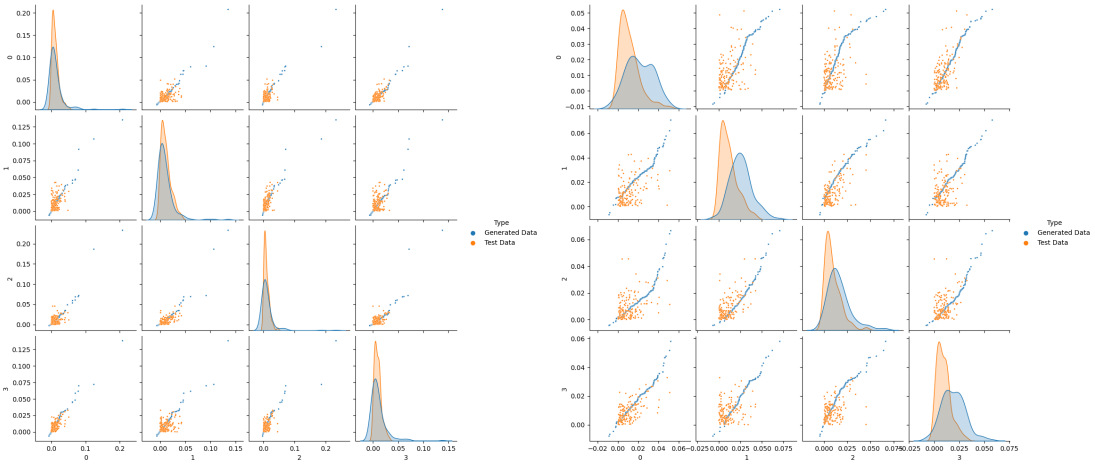


Figure 4: Distributions for data generated by ExGAN



(a) Block size = 1 (b) Block size = 2

Figure 5: Distributions for data generated by EvtGAN

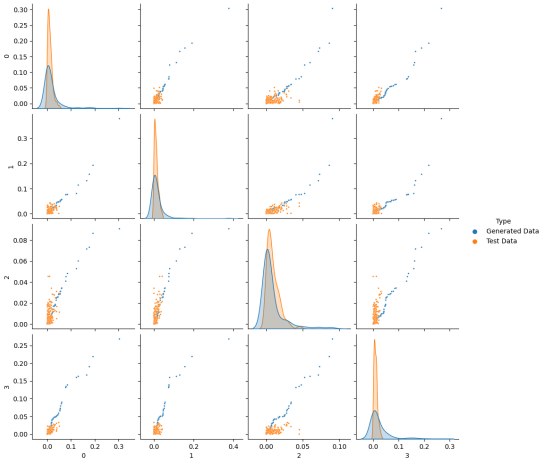
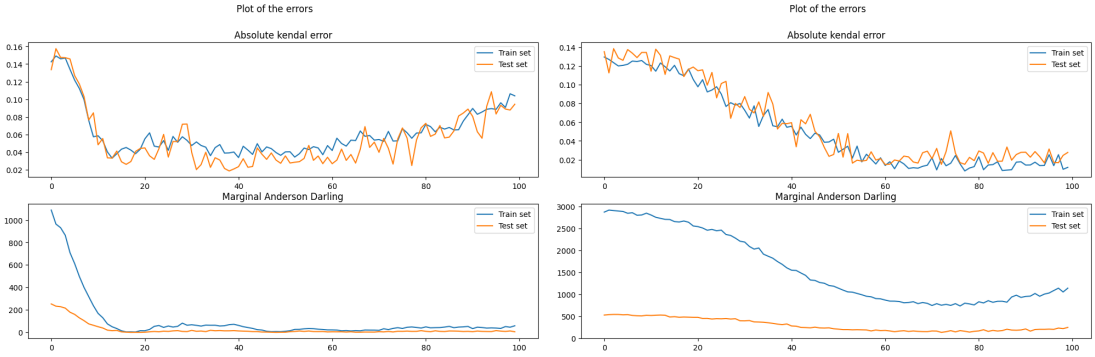


Figure 6: Distributions for data generated by EVGAN



(a) GAN (b) EVGAN

Figure 7: Score metrics over the epochs

4 Conclusion

Overall, it appears that while GANs typically generate a light-tailed distribution, they can still produce extreme values, even when such values are rare in the original dataset. The explored methods include:

- Shifting the data towards higher values by selecting the higher one (using sorting or block by block approach)
- Normalizing the data using a distribution taking into account extreme values (such as GEV)
- Modifying the architecture of the GAN itself

Among the four methods investigated in this project, the one yielding the best results is distribution shifting (ExGAN). This method minimizes both the Absolute Kendal Error and the Marginal Anderson Darling.

References

Bali, Turan G (2003). "The generalized extreme value distribution". In: *Economics letters* 79.3, pp. 423–427.

Xu, Lei et al. (2019). "Modeling tabular data using conditional gan". In: *Advances in neural information processing systems* 32.

Bhatia, Siddharth, Arjit Jain, and Bryan Hooi (2021). "Exgan: Adversarial generation of extreme samples". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8, pp. 6750–6758.

Huster, Todd et al. (2021). "Pareto gan: Extending the representational power of gans to heavy-tailed distributions". In: *International Conference on Machine Learning*. PMLR, pp. 4523–4532.

Zhang, Yishuo et al. (2021). "GANBLR: a tabular data generation model". In: *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, pp. 181–190.

Allouche, Michaël, Stéphane Girard, and Emmanuel Gobet (2022). "EV-GAN: Simulation of extreme events with ReLU neural networks". In: *The Journal of Machine Learning Research* 23.1, pp. 6723–6761.

Boulaguiem, Younes et al. (2022). "Modeling and simulating spatial extremes by combining extreme value theory with generative adversarial networks". In: *Environmental Data Science* 1, e5.

Kotelnikov, Akim et al. (2023). "Tabddpm: Modelling tabular data with diffusion models". In: *International Conference on Machine Learning*. PMLR, pp. 17564–17579.

Peard, Alison and Jim Hall (2023). "Combining deep generative models with extreme value theory for synthetic hazard simulation: a multivariate and spatially coherent approach". In: *arXiv preprint arXiv:2311.18521*.