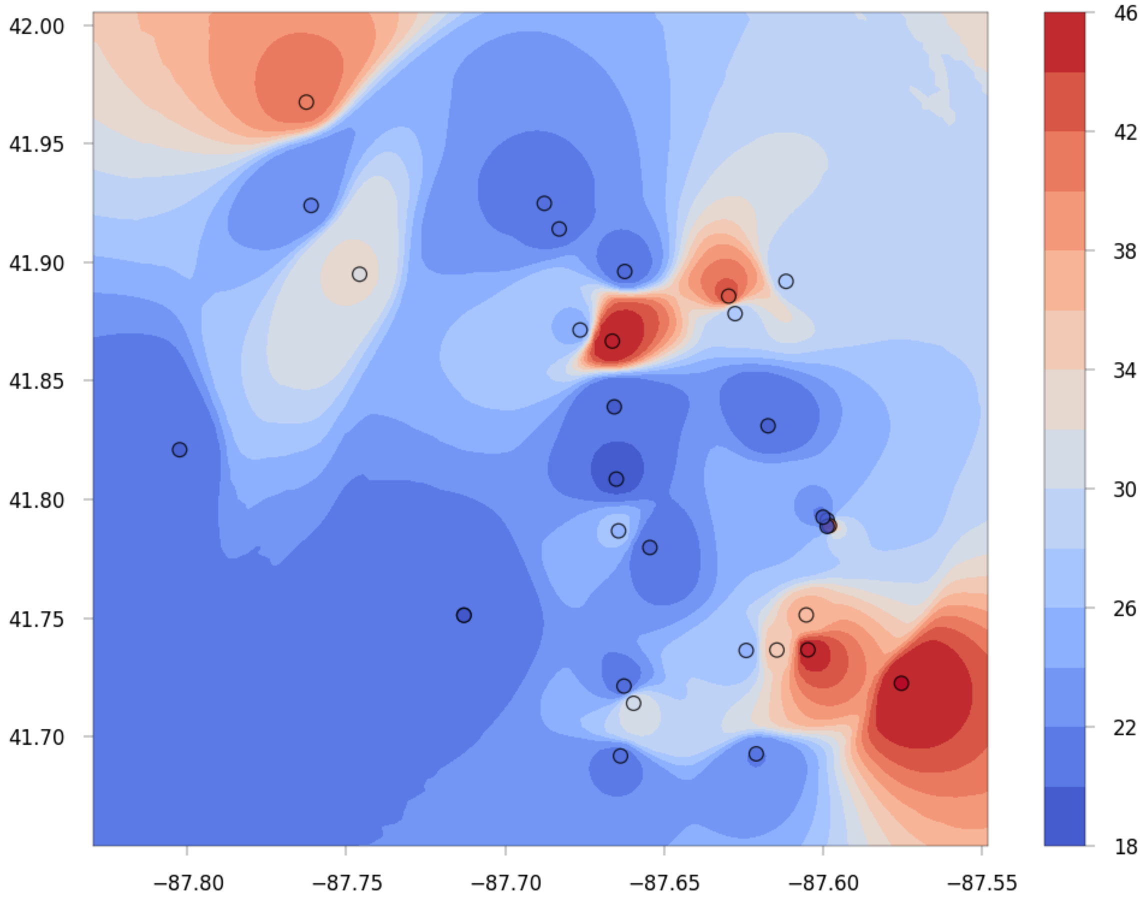
Statistical Optimization for AI and Machine Learning



Vincent Granville, Ph.D. | www.MLTechniques.com | Version 2.2, November 2023

Preface

This book covers optimization techniques pertaining to machine learning and generative AI, with an emphasis on producing better synthetic data with faster methods, some not even involving neural networks. NoGAN for tabular data is described in detail, along with full Python code, and case studies in healthcare, insurance, cybersecurity, education, and telecom. This low-cost technique is a game changer: it runs 1000x faster than generative adversarial networks (GAN) while consistently producing better results. Also, it leads to replicable results and auto-tuning.

Many evaluation metrics fail to detect defects in synthesized data, not because they are bad, but because they are poorly implemented: due to the complexity, the full multivariate version is absent from vendor solutions. In this book, I describe an implementation of the full version, tested on numerous examples. Known as the multivariate Kolmogorov-Smirnov distance (KS), it is based on the joint empirical distributions attached to the datasets, and works in any dimension on categorical and numerical features. Python libraries, both for NoGAN and KS, are now available and presented in this book.

A very different synthesizer also discussed, namely NoGAN2, is based on resampling, model-free hierarchical methods, auto-tuning, and explainable AI. It minimizes a particular loss function, also without gradient descent. While not based on neural networks, it nevertheless shares many similarities with GAN. Thus you can use it as a sandbox to quickly test various features and hyperparameters before adding the ones that work best, to GAN. Even though NoGAN and NoGAN2 don't use traditional optimization, gradient descent is the topic of the first chapter. Applied to data rather than math functions, there is no assumption of differentiability, no learning parameter, and essentially no math. The second chapter introduces a generic class of regression methods covering all existing ones and more, whether your data has a response or not, for supervised or unsupervised learning. I use gradient descent in this case.

One chapter is devoted to NLP, featuring an efficient technique to process large amounts of text data: hidden decision trees, presenting some similarities with XGBoost. A similar technique is used in NoGAN. Then I discuss other GenAI methods and various optimization techniques, including feature clustering, data thinning, smart grid search and more. Multivariate interpolation is used for time series and geospatial data, while agent-based modeling applies to complex systems.

Methods are accompanied by enterprise-grade Python code, also available on GitHub. Chapters are mostly independent from each other, allowing you to read in random order. The style is very compact, and suitable to business professionals with little time. Jargon and arcane theories are absent, replaced by simple English to facilitate the reading by non-experts, and to help you discover topics usually made inaccessible to beginners. While state-of-the-art research is presented in all chapters, the prerequisites to read this book are minimal: an analytic professional background, or a first course in calculus and linear algebra.

About the author

Vincent Granville is a pioneering GenAI scientist and machine learning expert, co-founder of Data Science Central (acquired by a publicly traded company in 2020), Chief AI Scientist at MLTechniques.com, former VC-funded executive, author and patent owner – one related to LLM. Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, and CNET.

Vincent is also a former post-doc at Cambridge University, and the National Institute of Statistical Sciences (NISS). He published in *Journal of Number Theory*, *Journal of the Royal Statistical Society* (Series B), and *IEEE Transactions on Pattern Analysis and Machine Intelligence*. He is the author of multiple books, available here, including "Synthetic Data and Generative AI" (Elsevier, 2024). Vincent lives in Washington state, and enjoys doing research on stochastic processes, dynamical systems, experimental math and probabilistic number theory. He recently launched a GenAI certification program, offering state-of-the-art, enterprise grade projects to participants. The program, based on his books, is discussed here.

Contents

```
plt.plot(lags[1:nlags], acf_x_raw[1:nlags], linewidth = 0.3)
plt.plot(lags[1:nlags], acf_y_raw[1:nlags], linewidth = 0.3)
plt.legend(['1st feature','2nd feature'],fontsize = 7, loc='upper right')
plt.xlabel('Autocorrelation function before reshuffling', fontsize=7)
plt.subplot(2,1,2)
plt.plot(lags[1:nlags], acf_x[1:nlags], linewidth = 0.3)
plt.plot(lags[1:nlags], acf_y[1:nlags], linewidth = 0.3)
plt.legend(['1st feature','2nd feature'],fontsize = 7, loc='upper right')
plt.xlabel('Autocorrelation function after reshuffling', fontsize=7)
plt.show()

plt.scatter(data[:,0], data[:,1], s = 0.1)
plt.show()
```

10.4 Easy Trick to Debias GenAI Models: Quantile Convolution

All of the GenAI apps that I tested, including my own, have the same problem. They cannot easily generate data outside the observation range. As an example, let's focus on the insurance dataset discussed in section 5.2.1. I use it to generate synthetic data with GAN (generative adversarial networks) and the NoGAN models discussed in chapters 6 and 7. In the training set, one of the features is "charges", that is, the medical expenses incurred by the policy holder, in a given year. The range is from \$1121 to \$63,770. In the synthesized data, the amount always stays within these two bounds. Worst, most models are unable to produce a synthetic maximum above \$60,000. The issue is undetected due to poor evaluation metrics, and compounded by the small size of the training set. The same is true for all the other features. The problem shows up in all the tested datasets, no matter how many observations you generate.

The consequences are persistent algorithm bias, and the inability to generate enriched or unusual data. The solution currently adopted is to work with gigantic training sets, further increasing costs linked to training, cloud and GPU time usage. What I propose here goes in the opposite direction: cost reduction, smaller training sets, high quality output based on the best evaluation metrics (see section 6.2.4), and the ability to generate more diversified data, including meaningful outliers. All this with a fast, simple algorithm based on a clever idea.

10.4.1 Quantile convolution

I now discuss the concept in layman's terms, and then briefly explain the underlying theory. By comparison to diffusion models [Wiki] used in computer vision to adress the issue, the technique is a lot simpler. Here I focus on the one-dimensional case here, but it generalizes to higher dimensions.

The training set consists of n observations $x_1, \dots, x_n$. I then create a Gaussian mixture model (GMM) with n components, all having the same weight $1/n$ and same variance σ_n^2. The k-th component is a Gaussian centered at x_k, with variance σ_n^2. I then sample N deviates from the GMM to compute its quantile function (the inverse of the CDF), typically with N much larger than n. At this point, we have three distributions:

- The empirical distribution H_n attached to the Gaussian mixture.
- The empirical distribution F_n attached to the training set observations.
- A generic normal distribution G_n, called kernel, with zero mean and variance σ_n^2.

The setting is identical to kernel density estimation [Wiki]. The derivative of H_n plays the role of the smooth density function estimate. I denote the corresponding probability density functions (PDF) as h_n, f_n and g_n. We have:

$$H_n(z) = \frac{1}{n}\sum_{k=1}^{n} G_n(z-x)I(x=x_k), \tag{10.3}$$

where I is the indicator function, equal to 1 if $x = x_k$, and 0 otherwise. When n is large and the discrete PDF f_n is well approximated by a continuous density f, we have $H_n(z) \sim \int G_n(z-x)f(x)dx$. Thus, taking the derivative with respect to z, we obtain:

$$h_n(z) \sim \int g_n(z-x)f(x)dx = (g_n * f)(z). \tag{10.4}$$

The $*$ symbol denotes the convolution product, in this case the convolution of probability distributions [Wiki]. If $\sigma_n \to 0$ as $n \to \infty$, then $h_n \to f$. Also, if $\sigma_n = 0$, then $H_n = F_n$ corresponds to the empirical distribution

(ECDF) computed on the training set. The ECDF F_n is known to converge to the true continuous underlying CDF F. This is another way to look at the asymptotic behavior: $\sigma_n \to 0 \Rightarrow H_n \sim F_n \to F$.

So, by choosing $\sigma_n > 0$ yet small enough, especially if n is large, we achieve the following comprise: h_n is some intermediate PDF between the discrete, chaotic f_n and the smooth, continuous theoretical but unknown limit, f. In practice, for σ_n, you can choose the standard deviation computed on the training set, multiplied by a small positive factor denoted as v_n.

With this framework, it is fast and easy to sample (say) $N = 10^6$ deviates from the H_n distribution and sort them to compute (say) 10^3 quantiles, from 0.0005 to 0.9995 by increments of 10^{-3}, and store them in an array. More granular quantiles are obtained by interpolating the pre-computed values. If σ_n is not too small, it will generate values outside the observation range in the training set. Unlike the standard method described here and here to sample from a mixture, it does not involve inverting the CDF H_n, resulting in a much faster implementation. The NoGAN techniques to generate synthetic data (see chapters 6 and 7) heavily rely on quantiles in high dimensions, and the convoluted quantiles discussed here can easily be integrated into these algorithms.

In the end, my technique is a fully automated, data-driven version of quantile extrapolation. For more on this topic, see “Nonparametric Extrapolation of Extreme Quantiles: a Comparison Study”, published in 2022 [3]. Besides smoothing empirical quantiles and outside-the-range synthetizations, another application is the generation of meaningful outliers: those not resulting from some error, but naturally occurring, albeit rarely, in the real world. This could be useful in contexts such as fraud detection.

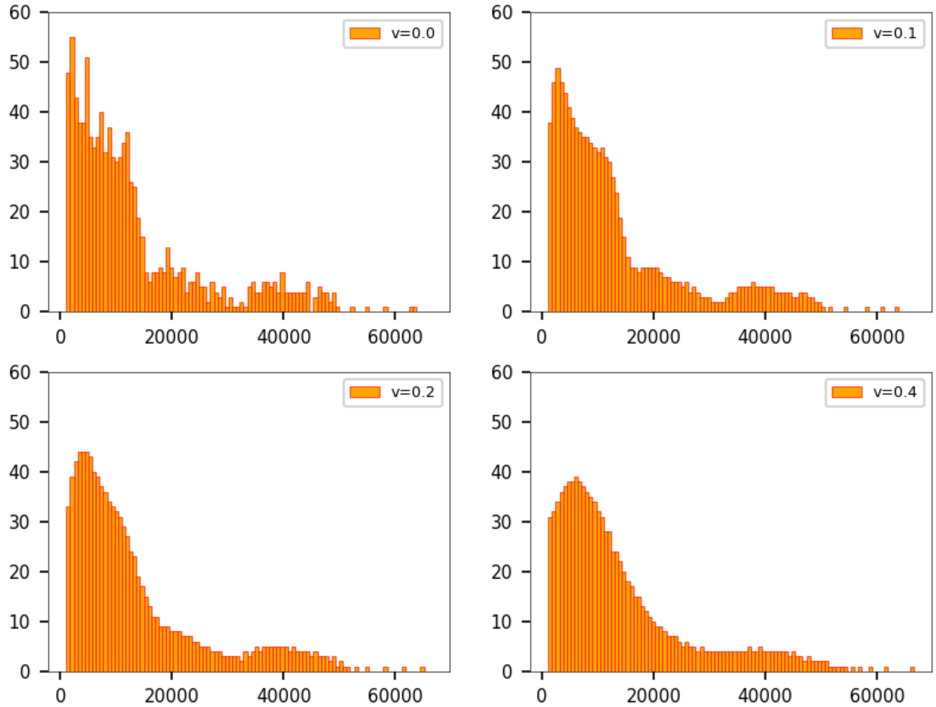


Figure 10.7: Histograms for extrapolated “charges” ($v = 0$ is the training set)

10.4.2 Truncated Gaussian mixtures and bias detection

One may use kernels other than Gaussian. Discrete kernels are the best solution when dealing with a categorical or discrete feature, such as “number of children” in the insurance dataset. In this section, I discuss a different type of kernel: truncated Gaussian. You use it if your data is constrained to stay within a specific domain D. For instance, in the insurance dataset, “charges” must be positive, and possibly above \$1000 due to business rules. They may be capped at \$100,000. For business reasons, “age” must be between 18 and 64 inclusive.

In the case studies in section 10.4.3, I use rejection sampling [Wiki] to meet these needs. The principle is simple: if a generated deviate is outside the domain D, reject it and continue sampling until you get one that lies inside D. This may result in noticeable bias in the synthetic data if σ_n is not small enough. However, one might say that the bias is in the real data, not in the synthetic data, especially if n is small. The synthetization may be a better representation of the reality. This is true especially when the truncation is one-sided, with a hard minimum (values must be above zero) but no hard maximum.

To assess whether the bias is in the real or in the synthetic data, proceed as follows. Create a training set using Monte-carlo simulations and known distribution, with $2n$ values. Then:

- Use n values (half of the training set) to compute the quantile table Q_n based on H_n.
- Compute the mean both on the half training set S_n, and on the quantile table Q_n. They are denoted respectively as $\mu(S_n)$ and $\mu(Q_n)$.
- Compute the mean $\mu(S_{2n})$ on the full training set.

If $|\mu(Q_n) - \mu(S_{2n})| < |\mu(S_n) - \mu(S_{2n})|$, then the bias is likely more pronounced in the real data (the half training set), rather than in the synthetic data! Alternatively, you can transform the generated quantiles so that the mean and variance match those measured on the training set. This makes sense if you use an unusually large σ_n with one-sided truncation, resulting in generated values far beyond the maximum or minimum observed in the training set.

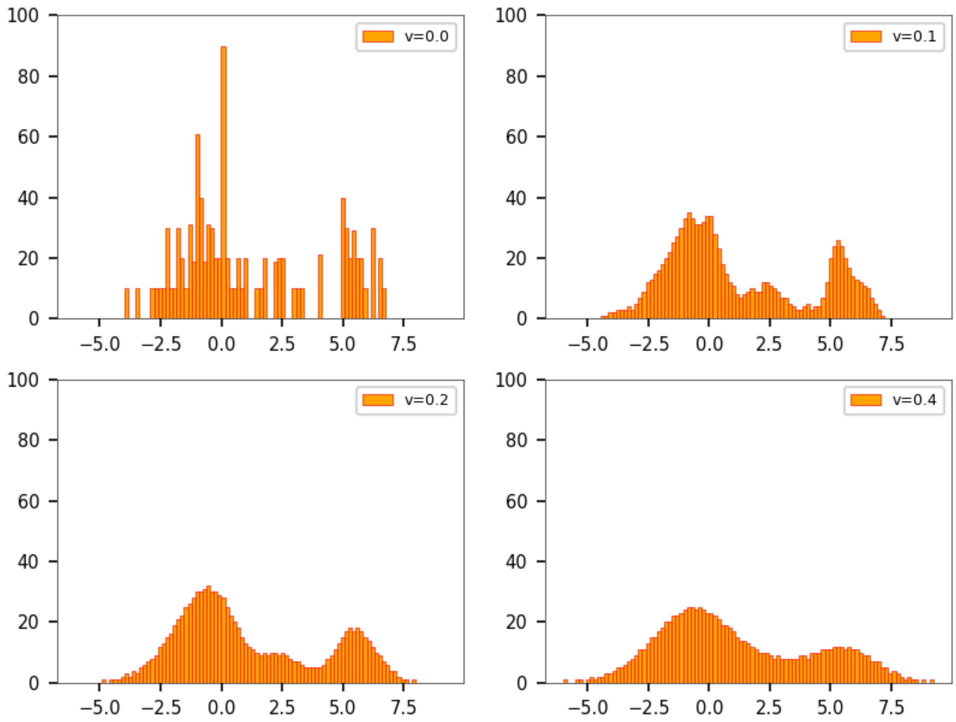


Figure 10.8: Histograms for extrapolated mixture ($v = 0$ is the training set)

Feature	v	$P_{.0005}$	$P_{.9995}$	Median	Stdev
charges	0.0	1122	63,770	9374	12,103
charges	0.1	1076	63,709	9393	12,079
charges	0.2	1076	64,538	9485	12,090
charges	0.4	1077	66,520	10,392	12,113
bmi	0.0	15.96	53.13	30.40	6.09
bmi	0.1	15.89	53.10	30.39	6.13
bmi	0.2	15.09	53.44	30.42	6.21
bmi	0.4	12.77	54.31	30.42	6.57
simulated	0.0	−3.87	6.66	0.05	2.87
simulated	0.1	−4.35	7.20	0.13	2.89
simulated	0.2	−4.87	7.85	0.21	2.93
simulated	0.4	−5.99	9.21	0.43	3.09

Table 10.3: Extreme values as a function of v (training set: $v = 0$)

10.4.3 Case studies

Figures 10.7, 10.8 and 10.9 show histograms associated to H_n, each with 50 bins, with 4 plots in each picture. For σ_n, I chose the standard deviation computed on the training set, multiplied by a small factor v, ranging from $v = 0.0$ (top left plot) to $v = 0.4$ (bottom right). So, the top left plot corresponds to $\sigma_n = 0$. It represents the frequency distribution in the training set. The Y-axis features bin counts, totaling $n = 1000$ across all 50 bins in each plot. The X-axis represents the observed values: the extended range, after extrapolation based on quantile convolution.

Figures 10.7 and 10.9 correspond to two of the features in the insurance dataset: "charges" (in dollar amount), and "bmi" (body mass index). I used a truncated Gaussian for the kernel. Figure 10.8 pictures an artificial dataset: the data was created using a mixture with three components, with $n = 100$ observations. Quantile convolution ($v > 0$) with a Gaussian kernel clearly generates values outside the observation range. The Python code is in section 10.4.5.

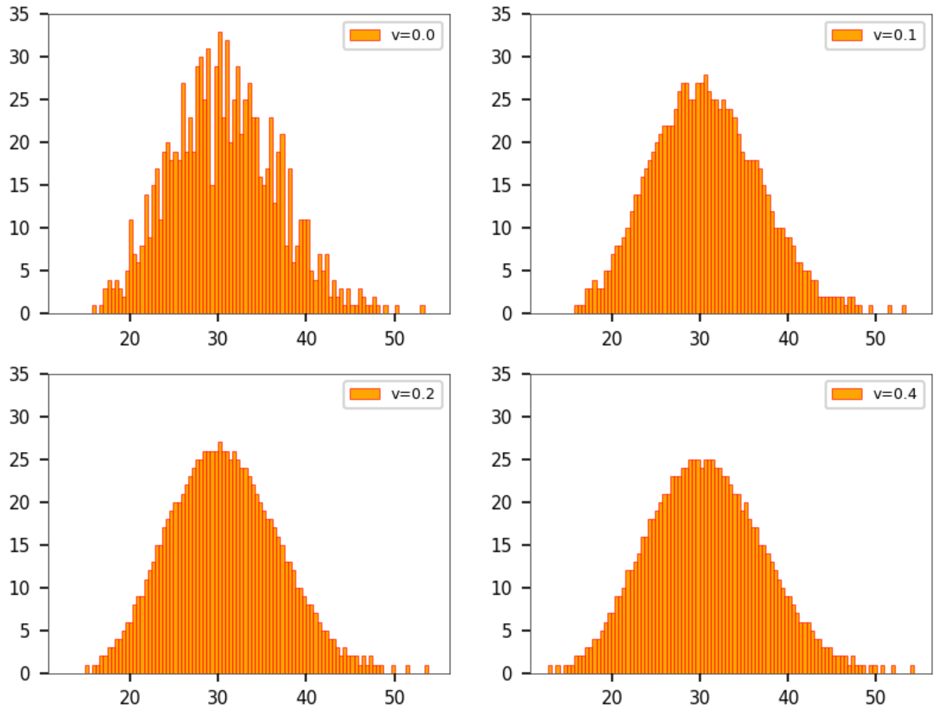


Figure 10.9: Histograms for extrapolated "bmi" ($v = 0$ is the training set)

In my examples, $v = 0.1$ seems to be the best value, preserving the patterns in the distribution attached to the training set, while generating extreme values that are not too far from the minimum and maximum in the real data. Table 10.3 summarizes the findings. In particular, the "simulated" feature was created as a mixture with 3 components, also called clusters. With $v = 0.1$ or $v = 0.2$, the 3 components are still visible: see Figure 10.8. The technique could also be used to detect the optimum number of clusters, and generalizes to higher dimensions. Note that with $v = 0.4$, generated values extend far beyond the observation range, allowing you to create meaningful outliers. In Table 10.3, $P_{.0005}$ and $P_{.9995}$ are extreme quantiles (convoluted if $v > 0$).

Finally, it would be interesting to see what happens when you iterate the method: starting with $H_{0,n} = F_n$ to produce $H_{1,n} = H_n$, then using $H_{1,n}$ to produce $H_{2,n}$ and so on. In short, synthesizing the synthetic data and so on. Most data synthesizers are unable to sample outside the observation range, resulting in successive iterations generating data within a shrinking range. Conversely, with a large σ_n, my method will generate data in an expanding range, over several iterations. The best solution is to choose σ_n that keeps the range stable over many iterations.

10.4.4 Conclusion

The quantile convolution technique helps you generate data outside the observation range, thus creating truly enriched datasets, contrarily to all the tools that I tried in the context of synthetic data, whether based on deep neural networks or not, whether open-source or vendor platforms. Generalizing quantiles to higher dimensions may not seem trivial, but it has been done with NoGAN and sister methods discussed in chapters 6 and 7. The new method, akin to quantile extrapolation, blends easily with NoGAN to enhance its performance.

Current techniques to evaluate the quality of synthetic fail to capture complex feature dependencies, resulting in false negatives: generated data scored as excellent, when it is actually very poor. Deep neural networks can be very slow and volatile, requiring ad-hoc tuning for each new dataset. In this book, the focus was on new algorithms – not necessarily neural networks – that are fast and easy to train, lead to explainable AI and auto-tuning, and require less rather than more data to address the traditional challenges. For instance, in section 8.2, I illustrate how you can get better results, in addition to saving time, by randomly deleting 50% of the data in the training set. All of this using sound evaluation metrics and cross-validation.

The main goal of all this framework is cost savings while delivering better results: using less training, GPU and cloud time. It goes against the modern trend of using bigger and bigger datasets. The popularity of oversized datasets stems from the fact that it seems to be the easy solution. Yet my algorithms are simpler. Then, large companies offering cloud and GPU services have strong incentives to favor big data: the bigger, the more revenue for them, the higher the costs for the client. Since I offer free solutions, thus bearing the cost of computations, I have strong incentives to optimize for speed while maintaining high quality output. In the end, my goals are thus aligned with those of the client, not with those of cloud companies or vendor charging a premium for cloud usage, based on the volume of data.

10.4.5 Python code

The function `get_test_data` creates the simulated training set used in Figure 10.8, referenced as "simulated" in Table 10.3. The insurance data set is accessed from GitHib via the URL in the code, in the `get_real_data` function. Truncation is determined by the parameters `minz` and `maxz`, with no truncation if `minz>maxz`. I do not list v as a variable, but it is implicitly used in instructions such as `sigma3=0.2*np.std(data)`, where $v = 0.2$. Convoluted quantiles are stored in arrays, e.g. `e1quant1`, while standard quantiles are in `pquant`. The code is also on GitHub, here.

```
# equantile.py: extrapolated quantiles

import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd

seed = 76
np.random.seed(seed)

def get_test_data(n=100):
   data = []
   for k in range(n):
      u = np.random.uniform(0, 1)
      if u < 0.2:
         x = np.random.normal(-1, 1)
      elif u < 0.7:
         x = np.random.normal(0, 2)
      else:
         x = np.random.normal(5.5, 0.8)
      data.append(x)
   data = np.array(data)
   return(data)

def get_real_data():
   url = "https://raw.githubusercontent.com/VincentGranville/Main/main/insurance.csv"
   data = pd.read_csv(url)
   # features = ['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']
   data = data['bmi'] # choose 'bmi' or 'charges'
   data = np.array(data)
   return(data)

#--

def truncated_norm(mu, sigma, minz, maxz):
   z = np.random.normal(mu, sigma)
   if minz < maxz:
      while z < minz or z > maxz:
         z = np.random.normal(mu, sigma)
   return(z)

#- sample from mixture

def mixture_deviate(N, data, f, sigma, minz, maxz, verbose=False):
```

```
    sample = []
    point_idx = np.random.randint(0, len(data), N)
    mu = data[point_idx]
    for k in range(N):
        z = truncated_norm(mu[k], sigma, minz, maxz)
        sample.append(z)
        if verbose and k%10 == 0:
            print("sampling %6d / %6d" %(k, N))
    sample = np.array(sample)
    sample = np.sort(sample)
    return(sample)

#--- Main part

# data = get_test_data(100)
data = get_real_data()
N = 1000000
truncate = False

# minz > maxz is the same as (minz = -infinity, maxz = +infinity)
if truncate == True:
    minz = 0.50 * np.min(data) # use 0.95 for 'charges', 0.50 for 'bmi'
    maxz = 1.50 * np.max(data) # use 1.50 for 'charges', 1.50 for 'bmi'
else:
    minz = 1.00
    maxz = 0.00

sigma1 = 0.0 * np.std(data)
sample1 = mixture_deviate(N, data, truncated_norm, sigma1, minz, maxz)

sigma2 = 0.1 * np.std(data)
sample2 = mixture_deviate(N, data, truncated_norm, sigma2, minz, maxz)

sigma3 = 0.2 * np.std(data)
sample3 = mixture_deviate(N, data, truncated_norm, sigma3, minz, maxz)

sigma4 = 0.4 * np.std(data)
sample4 = mixture_deviate(N, data, truncated_norm, sigma4, minz, maxz)

arrq = []
equant1 = []
equant2 = []
equant3 = []
equant4 = []
pquant = []

pbins = 1000
step = N / pbins # N must be a multiple of pbins
for k in range(pbins):
    p = (k + 0.5) / pbins
    arrq.append(p)
    eq_index = int(step * (k + 0.5))
    equant1.append(sample1[eq_index])
    equant2.append(sample2[eq_index])
    equant3.append(sample3[eq_index])
    equant4.append(sample4[eq_index])
    pquant.append(np.quantile(data, p))

mpl.rcParams['axes.linewidth'] = 0.3
plt.rcParams['xtick.labelsize'] = 7
plt.rcParams['ytick.labelsize'] = 7

#--- Plot results

bins=np.linspace(np.min(equant4), np.max(equant4), num=100)
```

```
plt.subplot(2,2,1)
plt.hist(equant1,color='orange',edgecolor='red',bins=bins,linewidth=0.3,label='v=0.0')
plt.legend(loc='upper right', prop={'size': 6}, )
plt.ylim(0,35)
plt.subplot(2,2,2)
plt.hist(equant2,color='orange',edgecolor='red',bins=bins,linewidth=0.3,label='v=0.1')
plt.legend(loc='upper right', prop={'size': 6}, )
plt.ylim(0,35)
plt.subplot(2,2,3)
plt.hist(equant3,color='orange',edgecolor='red',bins=bins,linewidth=0.3,label='v=0.2')
plt.legend(loc='upper right', prop={'size': 6}, )
plt.ylim(0,35)
plt.subplot(2,2,4)
plt.hist(equant4,color='orange',edgecolor='red',bins=bins,linewidth=0.3,label='v=0.4')
plt.legend(loc='upper right', prop={'size': 6}, )
plt.ylim(0,35)
plt.show()

#--- Output some summary stats

print()
print("Observation range, min: %8.2f" %(np.min(data)))
print("Observation range, max: %8.2f" %(np.max(data)))
pmin = np.quantile(data, 0.5/pbins)
pmax = np.quantile(data, 1 - 0.5/pbins)
print("Python quantile %6.4f: %8.2f" % (0.5/pbins, pmin))
print("Python quantile %6.4f: %8.2f" % (1-0.5/pbins, pmax))
print("Python quantile %6.4f: %8.2f" % (0.5, np.quantile(data,0.5)))
print("Dataset stdev  : %8.2f" %(np.std(data)))

print()
print("sigma1: %6.2f" %(sigma1))
print("Equant quantile %6.4f: %8.2f" %(0.5/pbins, equant1[0]))
print("Equant quantile %6.4f: %8.2f" %(1-0.5/pbins, equant1[999]))
print("Equant quantile %6.4f: %8.2f" %(0.5, np.median(equant1)))
print("Equant-based stdev : %8.2f" %(np.std(equant1)))

print()
print("sigma2: %6.2f" %(sigma2))
print("Equant quantile %6.4f: %8.2f" %(0.5/pbins, equant2[0]))
print("Equant quantile %6.4f: %8.2f" %(1-0.5/pbins, equant2[999]))
print("Equant quantile %6.4f: %8.2f" %(0.5, np.median(equant2)))
print("Equant-based stdev : %8.2f" %(np.std(equant2)))

print()
print("sigma3: %6.2f" %(sigma3))
print("Equant quantile %6.4f: %8.2f" %(0.5/pbins, equant3[0]))
print("Equant quantile %6.4f: %8.2f" %(1-0.5/pbins, equant3[999]))
print("Equant quantile %6.4f: %8.2f" %(0.5, np.median(equant3)))
print("Equant-based stdev : %8.2f" %(np.std(equant3)))

print()
print("sigma4: %6.2f" %(sigma4))
print("Equant quantile %6.4f: %8.2f" %(0.5/pbins, equant4[0]))
print("Equant quantile %6.4f: %8.2f" %(1-0.5/pbins, equant4[999]))
print("Equant quantile %6.4f: %8.2f" %(0.5, np.median(equant4)))
print("Equant-based stdev : %8.2f" %(np.std(equant4)))
```

Bibliography

[1] Insaf Ashrapov. Tabular gans for uneven distribution. *Preprint*, pages 1–11, 2020. arXiv:2010.00638 [Link]. 78

[2] Caglar Aytekin. Neural networks are decision trees. *Preprint*, pages 1–8, 2022. arXiv:2210.05189 [Link]. 94

[3] Fabiola Banfi, Greta Cazzaniga, and Carlo De Michele. Nonparametric extrapolation of extreme quantiles: a comparison study. *Stochastic Environmental Research and Risk Assessment*, 36:1579–1596, 2022. [Link]. 114, 182

[4] Paul Beale. *Statistical Mechanics*. Academic Press, third edition, 2011. 165

[5] Marc G. Bellemare et al. The Cramer distance as a solution to biased Wasserstein gradients. *Preprint*, pages 1–20, 2017. arXiv:1705.10743 [Link]. 118

[6] Anthony J Bishara, Jiexiang Li, and Thomas Nash. Asymptotic confidence intervals for the Pearson correlation via skewness and kurtosis. *British Journal of Mathematical and Statistical Psychology*, pages 165–185, 2018. [Link]. 175

[7] Ali Borji. Pros and cons of GAN evaluation measures: New developments. *Preprint*, pages 1–35, 2021. arXiv:2103.09396 [Link]. 78

[8] Wei Chen and Mark Fuge. Synthesizing designs with interpart dependencies using hierarchical generative adversarial networks. *Journal of Mechanical Design*, 141:1–11, 2019. [Link]. 113

[9] Fida Dankar et al. A multi-dimensional evaluation of synthetic data generators. *IEEE Access*, pages 11147–11158, 2022. [Link]. 156

[10] Alvaro Figueira and Bruno Vaz. Survey on synthetic data generation, evaluation methods and GANs. *New Insights in Machine Learning and Deep Neural Networks*, 2022. MDPI [Link]. 78

[11] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLTechniques.com, 2022. [Link]. 140

[12] Vincent Granville. Generative AI: Synthetic data vendor comparison and benchmarking best practices. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 96, 99, 110, 118, 156

[13] Vincent Granville. *Gentle Introduction To Chaotic Dynamical Systems*. MLTechniques.com, 2023. [Link]. 177

[14] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [Link]. 8, 14, 35, 38, 39, 42, 53, 58, 63, 71, 73, 75, 83, 156, 167, 176

[15] Vincent Granville, Mirko Krivanek, and Jean-Paul Rasson. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:652–656, 1996. 120, 139

[16] Hui Guo et al. Eyes tell all: Irregular pupil shapes reveal gan-generated faces. *Preprint*, pages 1–7, 2021. arXiv:2109.00162 [Link]. 71

[17] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly, third edition, 2023. 44

[18] Radim Halir and Jan Flusser. Numerically stable direct least squares fitting of ellipses. *Preprint*, pages 1–8, 1998. [Link]. 26, 28

[19] Markus Herdin. Correlation matrix distance, a meaningful measure for evaluation of non-stationary MIMO channels. *Proc. IEEE 61st Vehicular Technology Conference*, pages 1–5, 2005. [Link]. 78

[20] Christian Hill. *Learning Scientific Programming with Python*. Cambridge University Press, 2016. [Link]. 28

[21] Pavel Krapivsky, Sidney Redner, and Eli Ben-Naim. *A Kinetic View of Statistical Physics*. Cambridge University Press, 2010. [Link]. 165

[22] Jogendra Nath Kundu et al. GAN-Tree: An incrementally learned hierarchical generative framework for multi-modal data distributions. *IEEE/CVF International Conference on Computer Vision*, pages 8190–8199, 2019. arXiv:1908.03919 [Link]. 113

[23] Nicolas Langrené and Xavier Warin. Fast multivariate empirical cumulative distribution function with connection to kernel density estimation. *Computational Statistics & Data Analysis*, 162:1–16, 2021. [Link]. 95, 96, 119

[24] Gary R. Lawlor. A l'Hospital's rule for multivariable functions. *Preprint*, pages 1–13, 2013. arXiv:1209.0363 [Link]. 56

[25] Tengyuan Liang. Estimating certain integral probability metric (IPM) is as hard as estimating under the IPM. *Preprint*, pages 1–15, 2019. arXiv:1911.00730 [Link]. 119

[26] Hui Liu et al. A new model using multiple feature clustering and neural networks for forecasting hourly $PM_{2.5}$ concentrations. *Engineering*, 6:944–956, 2020. [Link]. 76, 83, 132

[27] Mario Lucic et al. Are GANs created equal? a large-scale study. *Proc. NeurIPS Conference*, pages 1–10, 2018. [Link]. 78

[28] Christoph Molnar. *Interpretable Machine Learning*. ChristophMolnar.com, 2022. [Link]. 71

[29] Michael Naaman. On the tight constant in the multivariate Dvoretzky–Kiefer–Wolfowitz inequality. *Statistics & Probability Letters*, 173:1–8, 2021. [Link]. 95, 119

[30] Guillermo Navas-Palencia. Optimal binning: mathematical programming formulation. *Preprint*, pages 1–21, 2020. arXiv:2001.08025 [Link]. 46

[31] Sergey I. Nikolenko. *Synthetic Data for Deep Learning*. Springer, 2021. 78

[32] Peter Olver. *Complex Analysis and Conformal Mapping*. Preprint, 2022. University of Minnesota [Link][Mirror]. 13

[33] Alfred R.Osborne. Multidimensional Fourier series. *International Geophysics*, 97:115–145, 2010. [Link]. 63

[34] A Rény. On the theory of order statistics. *Acta Mathematica Academiae Scientiarum Hungaricae*, 4:191–231, 1953. 175

[35] Mahesh Shivanand and all. Fitting random regression models with Legendre polynomial and B-spline to model the lactation curve for Indian dairy goat of semi-arid tropic. *Journal of Animal Breeding and Genetics*, pages 414–422, 2022. [Link]. 63

[36] Joshua Snoke et al. General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society Series A*, 181:663–688, 2018. arXiv:1604.06651 [Link]. 44

[37] Bharath Sriperumbudur et al. On the empirical estimation of integral probability metrics. *Electronic Journal of Statistics*, pages 1550–1599, 2012. [Link]. 119

[38] Chang Su, Linglin Wei, and Xianzhong Xie. Churn prediction in telecommunications industry based on conditional Wasserstein GAN. *IEEE International Conference on High Performance Computing, Data, and Analytics*, pages 186–191, 2022. IEEE HiPC 2022 [Link]. 100, 113, 155, 156

[39] Chris Tofallis. Fitting equations to data with the perfect correlation relationship. *Preprint*, pages 1–11, 2015. Hertfordshire Business School Working Paper[Link]. 22

[40] D. Umbach and K.N. Jones. A few methods for fitting circles to data. *IEEE Transactions on Instrumentation and Measurement*, 52(6):1881–1885, 2003. [Link]. 23, 26

[41] D. A. Vaccari and H. K. Wang. Multivariate polynomial regression for identification of chaotic time series. *Mathematical and Computer Modelling of Dynamical Systems*, 13(4):1–19, 2007. [Link]. 26

[42] Fengyun Wang and all. Bivariate Fourier-series-based prediction of surface residual stress fields using stresses of partial points. *Mathematics and Mechanics of Solids*, 2018. [Link]. 63

[43] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *Preprint*, pages 1–12, 2018. arXiv:1811.11264 [Link]. 78

[44] Jinsung Yoon et al. GAIN: Missing data imputation using generative adversarial nets. *Preprint*, pages 1–10, 2018. arXiv:1806.02920 [Link]. 113

[45] Changgang Zheng et al. Reward-reinforced generative adversarial networks for multi-agent systems. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6:479–488, 2021. arXiv:2103.12192 [Link]. 83

Index