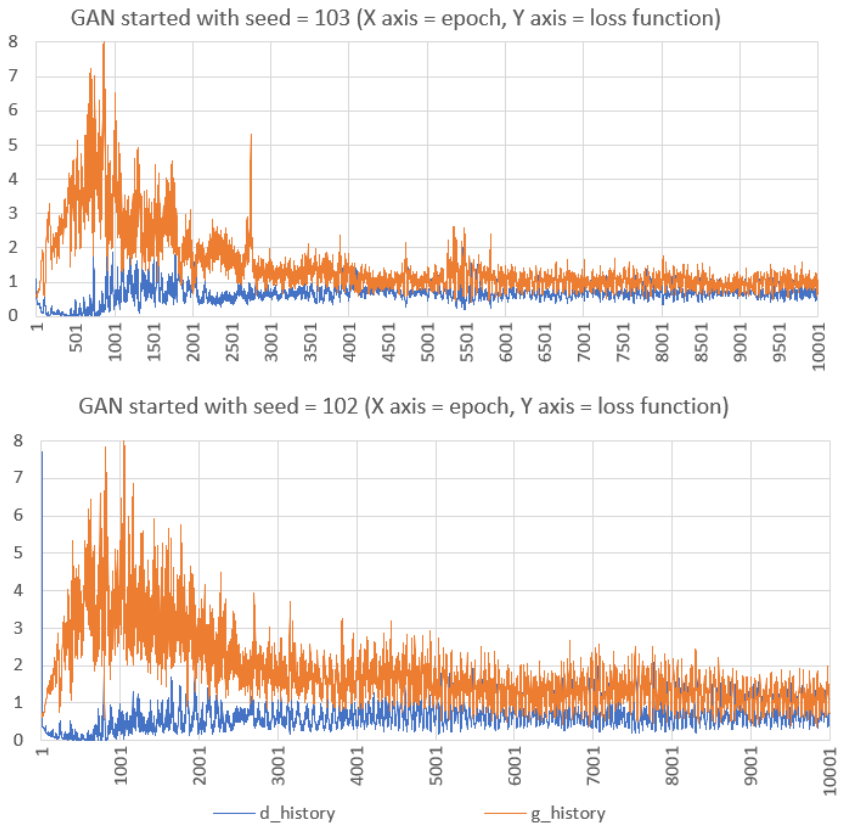
Practical AI & Machine Learning Projects and Datasets



Vincent Granville, Ph.D. | www.MLTechniques.com | Version 1.0, July 2023

Preface

This book is intended to participants in the AI and machine learning certification program organized by my AI/ML research lab MLtechniques.com. It is also an invaluable resource to instructors and professors teaching related material, and to their students. If you want to add serious, enterprise-grade projects to your curriculum, with deep technical dive on modern topics, you are welcome to re-use my projects in your classroom. I provide my own solution to each of them.

This book is also useful to prepare for hiring interviews. And for hiring managers, it is full of original and open questions, never posted before, encouraging candidates to think outside the box, with applications on real data. The amount of Python code accompanying the solutions is tremendous, using a vast array of libraries as well as home-made implementations showing the inner workings and improving existing black-box algorithms. By itself, this book constitutes a solid introduction to Python. The code is also on my GitHub repository.

The topics cover generative AI, synthetic data, machine learning optimization, scientific computing with Python, data visualizations and animations, time series and spatial processes, NLP and large language models, as well as graph applications and more. These projects based on real life data, with solution and enterprise-grade Python code, are a great addition to your portfolio (GitHub) when completed. Hiring managers and instructors can use them as as a complement to their battery of tests and projects, to differentiate themselves from competitors relying on overused, run-of-the-mill exercises.

To see how the program works and earn your certification(s), check out our FAQ posted here, or click on the "certification" tab on our website MLtechniques.com. Certifications can easily be displayed on your LinkedIn profile page in the credentials section, with just one click. Unlike many other programs, there is no exam or meaningless quizzes. Emphasis is on projects with real-life data, enterprise-grade code, efficient methods, and modern applications to build a strong portfolio and grow your career in little time. The guidance to succeed is provided by the founder of the company, one of the top and most well-known experts in machine learning, Dr. Vincent Granville. Jargon and unnecessary math are avoided, and simplicity is favored whenever possible. Nevertheless, the material is described as advanced by everyone who looked at it.

The related teaching and technical material (textbooks) can be purchased at MLtechniques.com/shop/. MLtechniques.com, the company offering the certifications, is a private, self-funded AI/ML research lab developing state-of-the-art open source technologies related to synthetic data, generative AI, cybersecurity, geospatial modeling, stochastic processes, chaos modeling, and AI-related statistical optimization.

About the author

Vincent Granville is a pioneering data scientist and machine learning expert, co-founder of Data Science Central (acquired by TechTarget), founder of MLTechniques.com, former VC-funded executive, author and patent owner.

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 also the author of multiple books, available here. He lives in Washington state, and enjoys doing research on stochastic processes, dynamical systems, experimental math and probabilistic number theory.

Chapter 4

Scientific Computing

Many projects throughout this book feature scientific programming in Python. This section offers a selection that best illustrates what scientific computing is about. In many cases, special libraries are needed, or you have to process numbers with billions of digits. Applications range from heavy simulations to cybersecurity. In several instances, very efficient algorithms are required.

4.1 The music of the Riemann Hypothesis: sound generation

This first project is an introduction to sound generation in Python with wave files, as well as the MPmath Python library to work with special functions including in the complex plane. This library is particularly useful to physicists. No special knowledge of complex functions is required: we work with the real and imaginary part separately, as illustrated in Figure 4.1: in this example, the frequency attached to a musical note is the real part of the complex Dirichlet eta function η [Wiki] (re-scaled to be positive), the duration of a note is the imaginary part of η after re-scaling, and the volume – also called amplitude – is the the modulus [Wiki].

The project consists of the following steps:

Step 1: Read my article "The Sound that Data Makes", available here. It contains Python code to turn random data into music. Also download the technical paper "Math-free, Parameter-free Gradient Descent in Python" available from here. It contains Python code to compute the Dirichlet eta function (among others) using the MPmath library.

Step 2: Using the material gathered in step 1, generate a sound file (wav extension) for the Dirichlet eta function $\eta(\sigma + it)$, with $\sigma = \frac{1}{2}$ and t between 400,000 and 400,020. Create 300 musical notes, one for each value of t equally spaced in the interval in question. Each note has 3 components: the frequency, duration, and volume, corresponding respectively to the real part, imaginary part, and modulus of $\eta(\sigma + it)$ after proper re-scaling.

Step 3: Same as step 2, but this time with a different function of your choice. Or better, with actual data rather than sampled values from mathematical functions. For instance, try with the ocean tide data, or the planet inter-distance data investigated in project 3.1.

Step 4: Optional. Create a sound track for the video featured in Figure 4.2. You can watch the video on YouTube, here. The code to produce this video (with detailed explanations) is in section 4.3.2 in my book [6], and also on GitHub, here. To blend the sound track and the video together, see solution on Stack Exchange, here. An alternative is to convert the wav file to mp4 formart (see how to do it here) then use the Moviepy library to combine them both.

Data visualizations offer colors and shapes, allowing you to summarize multiple dimensions in one picture. Data animations (videos) go one step further, adding a time dimension. See examples on my YouTube channel, and in my book [6]. Then, sound adds multiple dimensions: amplitude, volume and frequency over time. Producing pleasant sound, with each musical note representing a multivariate data point, is equivalent to data binning or bukectization. Stereo and the use of multiple musical instruments (synthesized) add more dimensions. Once you have a large database of data music, you can use it for generative AI: sound generation to mimic existing datasets. Of course, musical AI art is another application, all the way to creating synthetic movies.

Figure 4.1 shows the frequency, duration and volume attached to each of the 300 musical notes in the wav file, prior to re-scaling. The volume is maximum each time the Riemann zeta function hits a zero on the critical line. This is one of the connections to the Riemann Hypothesis. Note that in the solution, I use the

Dirichlet eta function $\eta(\sigma+it)$ with $\sigma=\frac{1}{2}$, corresponding to the critical line [Wiki]. According to the Riemann Hypothesis, this is the only positive value of σ where all the zeros of the Riemann zeta function $\zeta(\sigma+it)$ occur. Indeed, there are infinitely many t for which $\zeta(\frac{1}{2}+it)=0$. You can see the first 100 billion of them, here. The Dirichlet eta function has the same zeros. This is the connection to the Riemann Hypothesis. The notation $\sigma+it$ represents the complex argument of the functions involved, with σ the real part and t the imaginary part. More on this topic in chapter 17 in my book [6].

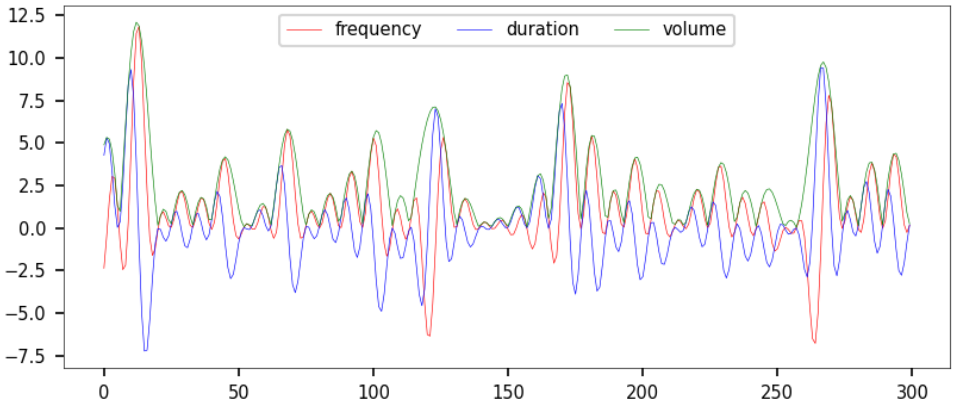


Figure 4.1: 300 musical notes, showing volume, duration and frequency

4.1.1 Project solution

The code in this section provides the answer to step 2. The variables `z.real` and `z.imag` correspond respectively to the real and imaginary part of z. The volume in the output wav file (the music) is maximum each time the Riemann zeta or Dirichlet eta function hits a zero on the critical line. The Python code is also on my GitHub repository here.

Figure 4.2 shows the final frame of the video discussed in step 4. It features the convergence path of the Dirichlet eta function in the complex plane, for a specific value of the complex argument $\sigma+it$, when adding more and more terms in the standard sine and cosine series to approximate the function. Here t is very large, and σ is in the critical band $\frac{1}{2}\leq\sigma<1$, where the most interesting action takes place.

Figure 4.2: Last frame from the video featuring the convergence of the Dirichlet eta function

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.io import wavfile
import mpmath
```

```
#-- Create the list of musical notes

scale=[]
for k in range(35, 65):
   note=440*2**((k-49)/12)
   if k%12 != 0 and k%12 != 2 and k%12 != 5 and k%12 != 7 and k%12 != 10:
      scale.append(note) # add musical note (skip half tones)
n_notes = len(scale) # number of musical notes


#-- Generate the data

n = 300
sigma = 0.5
min_t = 400000
max_t = 400020

def create_data(f, nobs, min_t, max_t, sigma):
   z_real = []
   z_imag = []
   z_modulus = []
   incr_t = (max_t - min_t) / nobs
   for t in np.arange(min_t, max_t, incr_t):
      if f == 'Zeta':
         z = mpmath.zeta(complex(sigma, t))
      elif f == 'Eta':
         z = mpmath.altzeta(complex(sigma, t))
      z_real.append(float(z.real))
      z_imag.append(float(z.imag))
      modulus = np.sqrt(z.real*z.real + z.imag*z.imag)
      z_modulus.append(float(modulus))
   return(z_real, z_imag, z_modulus)

(z_real, z_imag, z_modulus) = create_data('Eta', n, min_t, max_t, sigma)

size = len(z_real) # should be identical to nobs
x = np.arange(size)

# frequency of each note
y = z_real
min = np.min(y)
max = np.max(y)
yf = 0.999*n_notes*(y-min)/(max-min)

# duration of each note
z = z_imag
min = np.min(z)
max = np.max(z)
zf = 0.1 + 0.4*(z-min)/(max-min)

# volume of each note
v = z_modulus
min = np.min(v)
max = np.max(v)
vf = 500 + 2000*(1 - (v-min)/(max-min))

#-- plot data

mpl.rcParams['axes.linewidth'] = 0.3
fig, ax = plt.subplots()
ax.tick_params(axis='x', labelsize=7)
ax.tick_params(axis='y', labelsize=7)
plt.rcParams['axes.linewidth'] = 0.1
plt.plot(x, y, color='red', linewidth = 0.3)
plt.plot(x, z, color='blue', linewidth = 0.3)
```

```
plt.plot(x, v, color='green', linewidth = 0.3)
plt.legend(['frequency','duration','volume'], fontsize="7",
   loc ="upper center", ncol=3)
plt.show()

#-- Turn the data into music

def get_sine_wave(frequency, duration, sample_rate=44100, amplitude=4096):
   t = np.linspace(0, duration, int(sample_rate*duration))
   wave = amplitude*np.sin(2*np.pi*frequency*t)
   return wave

wave=[]
for t in x: # loop over dataset observations, create one note per observation
   note = int(yf[t])
   duration = zf[t]
   frequency = scale[note]
   volume = vf[t] ## 2048
   new_wave = get_sine_wave(frequency, duration = zf[t], amplitude = vf[t])
   wave = np.concatenate((wave,new_wave))
wavfile.write('sound.wav', rate=44100, data=wave.astype(np.int16))
```

4.2 Cross-correlations in random number sequences

This short off-the-beaten-path project constitutes an excellent preparation for job interviews. In a nutshell, the task consists of implementing the grade-school multiplication algorithm [Wiki] for numbers with millions of digits in base 2, in this case random digits. The practical goal is to assess when and how sequences of bits or binary digits are correlated or not. It is an important issue in the quadratic irrational random number generator (PRNG) discussed in chapter 11 in my book [6].

Indeed, this PRNG relies on blending billions of digits from millions of irrational numbers, using a very fast algorithm. Three of these numbers could be $\sqrt{2341961}, \sqrt{1825361}$ and $\sqrt{2092487}$. It turns out that the binary digits of $\sqrt{2341961}$ and $\sqrt{1825361}$ are correlated, when computing the empirical correlation on a growing sequence of digits. But those of $\sqrt{2341961}$ and $\sqrt{2092487}$ are not, at the limit when the number of digits becomes infinite. In the former case, the correlation is about 1.98×10^{-5}. The exact value is $1/50429$. While very close to zero, it is not zero, and this is a critical cybersecurity issue when designing military-grade PRNGs. Irrational numbers must be selected in such a way that all cross-correlations are zero.

The theoretical explanation is simple, though hard to prove. If X is an irrational number, and p, q are two positive co-prime odd integers, then the correlation between the binary digits of pX and qX, is $1/(pq)$. Again, this is the limit value when the number n of digits in each sequence tends to infinity. In my example, $2341961 = 239^2 \times 41, 1825361 = 211^2 \times 41, 2092487 = 211^2 \times 47$, and $50429 = 239 \times 211$.

4.2.1 Project and solution

Write a program that computes the binary digits of pX using grade-school multiplication. Here p is a positive integer, and X is a random number in $[0, 1]$. Use this program to compute the correlation between the sequences of binary digits of pX and qX, where p, q are positive integers, and X a number in $[0, 1]$ with random binary digits. Focus on the digits after the decimal point (ignore the other ones).

For the solution, see my Python code below. It is also on GitHub, here. It creates the digits of X, then those of pX and qX, starting backward with the last digits. Finally it computes the correlation in question, assuming the digits of X are random. It works if X has a finite number of digits, denoted as `kmax` in the code. By increasing `kmax`, you can approximate any X with infinitely many digits, arbitrarily closely.

```
# Compute binary digits of X, p*X, q*X backwards (assuming X is random)
# Only digits after the decimal point (on the right) are computed
# Compute correlations between digits of p*X and q*X
# Include carry-over when performing grade school multiplication

import numpy as np

# main parameters
```

Bibliography

[1] Fida Dankar et al. A multi-dimensional evaluation of synthetic data generators. *IEEE Access*, pages 11147–11158, 2022. [Link]. 53

[2] Vincent Granville. Feature clustering: A simple solution to many machine learning problems. *Preprint*, pages 1–6, 2023. MLTechniques.com [Link]. 42

[3] Vincent Granville. Generative AI: Synthetic data vendor comparison and benchmarking best practices. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 35

[4] Vincent Granville. Massively speed-up your learning algorithm, with stochastic thinning. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 42

[5] Vincent Granville. Smart grid search for faster hyperparameter tuning. *Preprint*, pages 1–8, 2023. MLTechniques.com [Link]. 40, 42

[6] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [Link]. 14, 19, 20, 21, 29, 30, 32, 34, 35, 39, 40, 42, 43, 45, 53, 54

[7] Elisabeth Griesbauer. *Vine Copula Based Synthetic Data Generation for Classification*. 2022. Master Thesis, Technical University of Munich [Link]. 42

[8] 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]. 53

[9] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *Preprint*, pages 1–23, 2022. Submitted to IEEE PAMI [Link]. 40

Index