Smart Grid Search: Case Study with Hybrid Zeta-geometric Distributions and Synthetic Data Generation

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com
Version 1.0, March 2023

The objective is two-fold. First, I introduce a 2-parameter generalization of the discrete geometric and zeta distributions. Indeed, a combination of both. It allows you to simultaneously match the variance and mean in observed data, thanks to the two parameters p and α. To the contrary, each distribution taken separately only has one parameter, and can not achieve this goal. The zeta-geometric distribution offers more flexibility, especially when dealing with unusual tails in your data. I illustrate the concept when synthesizing real-life tabular data with parametric copulas, for one of the features in the dataset: the number of children per policyholder.

Then, I show how to significantly improve grid search, and make it a viable alternative to gradient methods to estimate the two parameters p and α. The cost function – that is, the error to minimize – is the combined distance between the mean and variance computed on the real data, and the mean and variance of the target zeta-geometric distribution. Thus the mean and variance are used as proxy estimators for p and α. This technique is known as minimum contrast estimation, or moment-based estimation in statistical circles. The "smart" grid search consists of narrowing down on smaller and smaller regions of the parameter space over successive iterations.

The zeta-geometric distribution is just one example of an hybrid distribution. I explain how to design such hybrid models in general, using a very simple technique. They are useful to combine multiple distributions into a single one, leading to model generalizations with an increased number of parameters. The goal is to design distributions that are a good fit when some in-between solutions are needed to better represent the reality.

1 Introduction to hybrid distributions

The approach in this article is very much typical of machine learning. There is no statistical theory involved. Yet in the end, I sample data with the desired distribution, in order to fit the observed data. The zeta-geometric distribution is defined as follows:

$$P(X = k) = C \cdot \frac{p^k}{(k+1)^\alpha}, \quad k = 0, 1, 2, \ldots \tag{1}$$

The constant C depends on p and α; it is chosen so that the the probabilities add up to 1. It makes sense to require $0 < p \leq 1$. It also makes sense to require $\alpha \geq 0$, and even $\alpha > 1$ if $p = 1$. However, in my example, a negative value for α combined with $p < 1$ works well: it tends to increase the probabilities attached to small values of k, that is, families with a small number of children. Two particular cases are:

- Geometric distribution [Wiki]: $\alpha = 0$. In that case, we have $C = 1 - p$. The expectation and variance are respectively
$$\mathrm{E}[X] = \frac{p}{1-p}, \quad \mathrm{Var}[X] = \frac{p}{(1-p)^2}.$$
- Zeta distribution [Wiki]: $p = 1, \alpha > 1$. In that case, $C = \zeta(\alpha)$ is the Riemann zeta function [Wiki]. The expectation and variance are respectively
$$\mathrm{E}[X] = \frac{\zeta(\alpha-1)}{\zeta(\alpha)} \text{ for } \alpha > 2, \quad \mathrm{Var}[X] = \frac{\zeta(\alpha)\zeta(\alpha-2) - \zeta^2(\alpha-1)}{\zeta^2(\alpha)} \text{ for } \alpha > 3.$$

Here I use the zeta-geometric distribution to model the number of children per family. A geometric distribution may provide the perfect mean, but it tends to produce a few outliers with too many kids per family (above 10) even on samples with fewer than 2000 observations. As a result, the variance is too large, compared to that on the real data. A truncated geometric distribution can fix this issue, but it comes with an arbitrary threshold. The zeta-geometric distribution addresses these two issues. For other generalizations of the geometric distribution with a 5 parameters family, see [1].

The general idea of hybrid distributions is as follows: given two parametric distributions with probability density functions (PDF) f_1 and f_2, respectively with parameters p and α (these parameters can be multidimensional), you want to create a new one with the following PDF:

$$f(x) = C \cdot f_1^{\beta_1}(x) f_2^{\beta_2}(x)$$

where $\beta_1, \beta_2 \geq 0$ are two additional parameters and C is a normalization constant depending on all the parameters, chosen so that the new density f integrates to one. In our case, the densities f_1, f_2 are discrete, thus I use k instead of x, and summations rather than integrals. More precisely, in my example, f_1, f_2 are respectively a geometric and zeta density. The original distributions correspond respectively to $\beta_1 = 1, \beta_2 = 0$ and $\beta_1 = 0, \beta_2 = 1$.

Note that in my zeta-geometric example, you don't need β_1, β_2 as you can particularize the generalized density by playing with p and α alone. Of course, you can combine more than two distributions into a single one. When $\beta_1 = \beta_2 = 1$, it has a Bayesian interpretation, with (say) f_2 playing the role of a prior, and $f = f_1 f_2$ being the posterior.

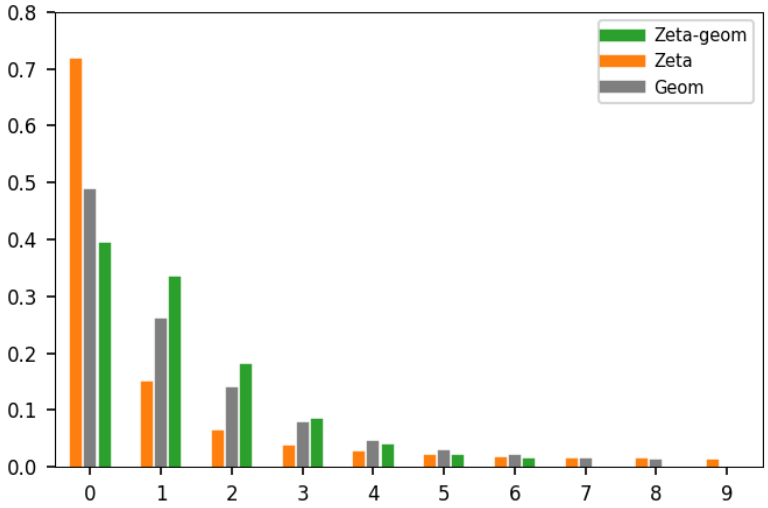

Figure 1: Three discrete PDFs with same expectation, different variances

Figure 1 shows three discrete probability density functions with the same expectation: zeta-geometric, zeta, and geometric. The parameters were chosen so that the expectation matches the average value observed in the dataset. Here, the feature of interest is the number of children per family. Because the geometric and zeta distributions only have one parameter, it was possible to match the average number of children, but not the standard deviation. The zeta-geometric distribution, with its two parameters p and α, is able to match both. The tail of the geometric distribution is too thick to fit nicely to the data. The zeta distribution is even worse.

So how can we combine both to get a distribution with a much thinner tail? The answer is in the fact that I use a negative value for α. A zeta distribution with a negative α can not exist. But when multiplied by a geometric density, the resulting distribution is well defined and has a much thinner tail. It fits much better to the data. In particular, the zeta-geometric PDF in Figure 1 (the green bars) is the only one with the correct variance, identical to that in the dataset.

Another generalization of the geometric distribution is as follows. Let X be the standard geometric distribution defined by (1). Then the power-geometric distribution of parameters p, γ is defined by

$$P(X_\gamma = k) = \frac{1-p}{\mu_\gamma} \cdot k^\gamma p^k, \quad k = 0, 1, 2, \ldots$$

Here $\mu_\gamma = \mathrm{E}[X^\gamma]$ and $\gamma \geq 0$. The standard geometric distribution corresponds to $\gamma = 0$. Also, the moments are related to those of the geometric distribution via the formula

$$\mathrm{E}[X_\gamma^\lambda] = \frac{\mathrm{E}[X^{\gamma+\lambda}]}{\mathrm{E}[X^\gamma]} = \frac{\mu_{\gamma+\lambda}}{\mu_\gamma}, \quad \gamma + \lambda \geq 0.$$

More generally, multiplying a density $P(X = k)$ by a power k^γ with $\gamma \geq 0$ results in an hybrid distribution that satisfies similar properties. This is also true for positive continuous densities, where the integer k is replaced by a real number $x \geq 0$. If γ is a positive integer, the restriction to $x \geq 0$ is not needed.

2 Case study: parametric copulas to synthesize data

I used the methodology discussed in this article in the context of data synthetization, to improve the quality of synthetic data produced by the copula technique. It relies on empirical quantiles [Wiki] measured on the real data, to replicate the correlation structure and the distribution attached to each feature, in the original dataset. Thus the technique is parameter-free. But one of the drawbacks is its inability to sample outside the range observed in the real data. One way to address this issue is to replace the empirical quantiles by those from a parametric family of distributions, and choose the parameters to fit to the real data.

I was particularly interested in one of the features in the data: the number of children per policyholder. The insurance dataset consisted of 7 features: age, gender, smoking status, number of children, location, and charges for 1400 policyholders. The goal was to predict the cost (charges) to the insurance company, based on the other features. The company was interested in synthetic data to enrich the training set, and for compliance with regulations pertaining to privacy and security.

To summarize, the maximum number of children was 5 in the dataset. This feature was well modeled by the one-parameter geometric distribution except that it produced a maximum of 10 to 11 children per family, depending on the generated sample. It proved that a parametric distribution could definitely sample outside the observation range (if enough observations are generated), but I wanted a better model. One that would replicate not only the average number of children per family (close to one, with many having no child), but also the standard deviation. The zeta-geometric distribution accomplished this goal, thanks to its two parameters. It lowered the maximum number of children to 7, closer to the observed value in the real data.

In the remaining of this section, I quickly describe the copula method [Wiki] and show where the zeta-geometric distribution fits in. Then I show how to sample from this distribution to get the best possible fit to the real data (for the number of children), and thus, good quality synthetic data. The estimation of the two parameters p, α attached to the zeta-geometric is based on an enhanced version of the grid search algorithm and discussed in section 3. There is no maximum likelihood function or statistical theory involved. Instead, I use the parameter vector (p, α) of the zeta-geometric as a proxy for the mean and variance. The technique is known as maximum contrast estimation in operations research, or moment estimation method [Wiki] in statistics.

2.1 Brief overview of the copula method

The goal is to produce synthetic data that mimics the correlation structure and individual distributions attached to each feature in a real dataset. It is a multivariate version of inverse transform sampling [Wiki], and works well with a blend of ordinal and continuous features. The method, applied to the insurance data mentioned in this article, is discussed in detail in my book on synthetic data [3]. It is broken down into 4 steps:

- Step 1: Compute the $m \times m$ correlation matrix W associated to your real data, where m is the number of features. This symmetric matrix contains the cross-correlations for each pair of features. Each element in the diagonal is equal to 1.
- Step 2: Generate n deviates from a multivariate Gaussian distribution with zero mean and covariance matrix W. Each deviate is a row vector Z_i $(i = 1, \ldots, n)$, with the components matching the features in the real data set.
- Step 3: For each generated Z_{ij} (the j-th feature in your i-th vector, with $1 \leq i \leq n$ and $1 \leq j \leq m$) compute $U_{ij} = \Phi(Z_{ij})$, where Φ is the CDF (cumulative distribution function) of a univariate standard normal distribution. Thus $0 \leq U_{ij} \leq 1$. These are uniform deviates on $[0, 1]$, with the correct feature cross-correlation structure.
- Step 4: Compute $S_{ij} = Q_j(U_{ij})$ where Q_j is the univariate empirical quantile distribution (the inverse of the empirical distribution) attached to the j-th feature, and computed on the real data.

To get a better understanding of what this algorithm accomplishes, look at what happens when $m = 1$. What I described here is the Gaussian copula method, one of the most popular. The last step is inverse transform sampling applied to each feature separately, to reconstruct the individual distributions.

Distribution	p	α	μ	σ	max
Zeta-geometric	0.27	−1.65	1.09	1.20	10
Zeta	1.00	2.33	1.10	4.27	99
Geometric	0.52	0.00	1.10	1.53	17

Table 1: μ, σ, and max. children; sample size: $n = 50{,}000$

In the dataset, the number of children is the fourth feature ($j = 4$). In the last step, instead of using the empirical quantile distribution Q_4, I replace it with the quantile distribution of a zeta-geometric distribution of parameters p, α, with p, α estimated on the real data (the fourth feature), as described in section 3. More specifically, I generate n deviates of the target zeta-geometric distribution. Then I compute its empirical quantiles based on that sample, and replace Q_4 by the quantiles in question. The quantiles are also called percentiles.

The same method can be applied to any feature, not just the number of children. Or even to combinations of features rather than individual features, using multivariate rather than univariate hybrid distributions. Finally, Table 1 shows the dramatic improvement when using a zeta-geometric (ZG) instead of a zeta or geometric distribution. In each case, the generated sample has $n = 50{,}000$ observations. The 3 distributions produce the same mean $\mu \approx 1.09$ thanks to the choice of parameters, but only ZG can produce the target standard deviation $\sigma \approx 1.20$. The maximum number of children is 7 on a test sample with $n = 1400$, the same number of observations as in the real data. The case $n = 1400$ is featured in Figure 1. Confidence intervals are obtained by generating a large number of samples of same size. The theoretical standard deviation of the zeta distribution with $\alpha = 2.33$, is infinite.

2.2 Sampling from a zeta-geometric distribution

I use the standard method to sample deviates from an arbitrary discrete distribution. The process consists of the following steps:

- Compute the constant C in (1), given specific values of p and α.
- Recursively compute $P(X \leq k) = P(X = k) + P(X \leq k - 1)$ using (1).
- Generate a uniform deviate U on $[0, 1]$.
- Find the largest k such that $U \leq P(X \leq k)$, starting at $k = 0$.

The value of k obtained in the last step is a deviate from the zeta-geometric distribution of parameters p, α. Repeat the whole procedure a thousand times, and you get a thousand deviates. This is implemented in the `sample_from_CDF` function in the Python code in section 4.

3 Smart grid search: viable alternative to gradient descent

This algorithm is simple yet quite efficient. It combines features of grid search [Wiki] and random search [Wiki], and does not require the target function to have a gradient. In fact it works with pure data, in the absence of any mathematical function. The algorithm is a multivariate version of the bisection method [Wiki]. See also [2].

I use it to estimate the parameters p, α of a zeta-geometric distribution, to provide a good fit to real data, by matching the average and standard deviation – in this case for the number of children per family – in the insurance dataset. Once the parameters are estimated, I generate a sample from the zeta-geometric distribution in question, and use its quantiles in the copula method described in section 2.1, as a substitute to the empirical quantiles. The end-goal is to get better synthetic data, by sampling outside of the observed range, yet fitting nicely to to the real data. See the `grid_search` function in the Python code, and how I use this function iteratively.

The cost function to minimize is $\varphi(p, \alpha) = (\mu_{p,\alpha} - \mu_0)^2 + (\sigma_{p,\alpha} - \sigma_0)^2$ where

- μ_0 and σ_0 are the mean and standard deviation measured on the observed data,
- $\mu_{p,\alpha}$ and $\sigma_{p,\alpha}$ are the mean and standard deviation of a ZG(p, α) distribution.

Here ZG stands for zeta-geometric. The vector (p, α) achieving the minimum (if unique) is denoted as $(p^\star, \alpha^\star)$. Here we have $\mu_0 = 1.09$ and $\sigma_0 = 1.20$, resulting in $p^\star \approx 0.27$ and $\alpha^\star \approx -1.65$. Figure 2 shows a contour plot of φ, with p on the X-axis and α on the Y-axis. The minimum is indeed unique and global, at least in the pictured region.

The long narrow basin makes a gradient descent approach [Wiki] difficult to solve this problem. It also means that many values of (p, α) are nearly optimum: we could choose one that provides not only a good match for the first two moments (they all do in the basin), but also a good match for the third moment or some other statistic such as the maximum number of children.

Now I explain how smart grid search works to solve the aforementioned optimization problem. The concept is very intuitive. Here ϵ_1, δ_1 are strictly positive real numbers, set respectively to 0.4 and 3.0 in this example.

- Start with initial values p_0, α_0 and a range $R_0 = [p_0 - \epsilon_0, p_0 + \epsilon_0] \times [\alpha_0 - \delta_0, \alpha_0 + \delta_0]$.

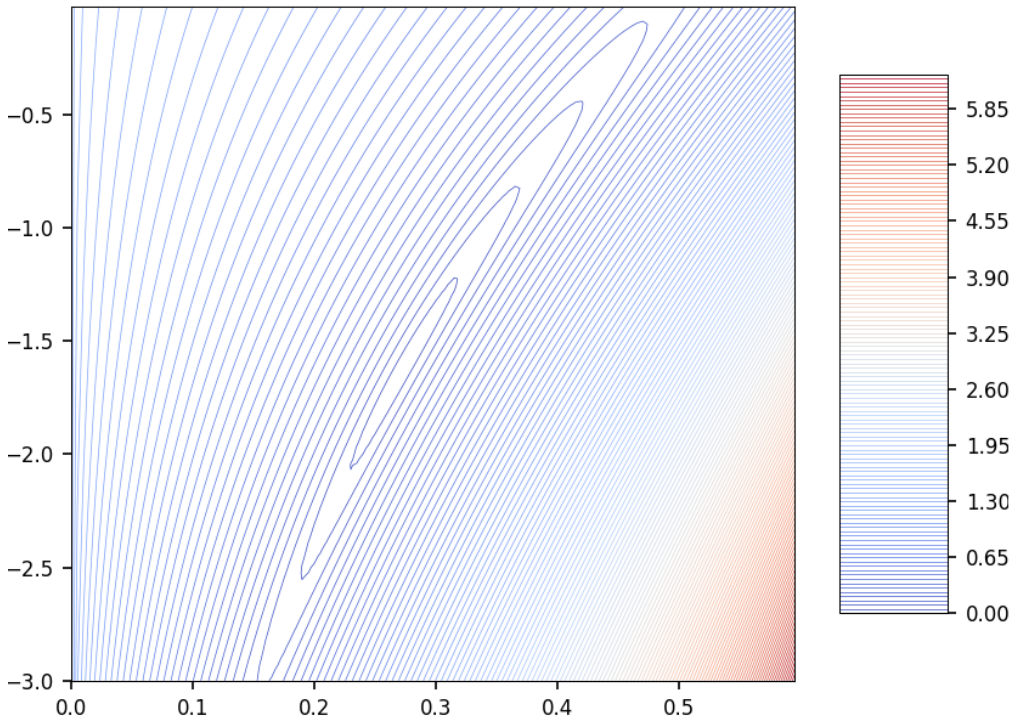

Figure 2: Error $\varphi(p, \alpha)$ to minimize, with p on the X-axis, α on the Y-axis

- Test 100 values of (p, α) evenly spread in R_0. Let (p_1, α_1) be the vector minimizing $\varphi(p, \alpha)$, among the 100 tests. With 10 different values for p, and 10 different values for α, the total is $10 \times 10 = 100$ tests.
- Let $\epsilon_1 = \rho\epsilon_0$ and $\delta_1 = \rho\delta_0$, with $0 < \rho < 1$.
- Go back to the first step, with p_0, α_0, R_0 replaced by p_1, α_1, R_1 with $R_1 = [p_1 - \epsilon_1, p_1 + \epsilon_1] \times [\alpha_1 - \delta_1, \alpha_1 + \delta_1]$.

This procedure is repeated a few times, generating a sequence $(p_n, \alpha_n) \in R_n$, with $\epsilon_n, \delta_n \to 0$. Thus R_n gets smaller and smaller at each iteration. In my test, an excellent approximation to the optimum was obtained in 3 iterations ($n = 3$), with $\rho = \frac{1}{2}$. See Table 2. In the table, μ_n, σ_n are the expectation and standard deviation of a GC(p_n, α_n) distribution. Again, the target values observed in the real dataset are $\mu = 1.09$ and $\sigma = 1.20$. The error $\varphi(p_n, \alpha_n)$ measures the distance between the target values, and those obtained at iteration n.

n	p_n	α_n	μ_n	σ_n	$\varphi(p_n, \alpha_n)$
0	0.5000	0.0000	–	–	–
1	0.3000	−1.5000	1.1612	1.2698	0.0926
2	0.3000	−1.3500	1.0640	1.2219	0.0353
3	0.2700	−1.6500	1.0962	1.2020	0.0032

Table 2: Convergence of smart grid search; $\varphi(p_n, \alpha_n)$ is the error

4 Python code

The code is also available on GitHub, here. Look for `ZetaGeom.py`.

```
import numpy as np

#--- compute mean and stdev of ZetaGeom[p, a]

def ZetaGeom(p, a):
   C = 0
   for k in range(200):
      C += p**k/(k+1)**a
```

```
   mu = 0
   m2 = 0
   for k in range(200):
      mu += k*p**k/(k+1)**a
      m2 += k*k*p**k/(k+1)**a
   mu /= C
   m2 /= C
   var = m2 - mu*mu
   stdev = var**(1/2)
   return(mu, stdev)

#--- smart grid search to find optimal p and a

def grid_search(grid_range):
   p_min = grid_range[0][0]
   p_max = grid_range[0][1]
   a_min = grid_range[1][0]
   a_max = grid_range[1][1]
   p_step = (p_max - p_min)/10
   a_step = (a_max - a_min)/10
   min_delta = 999999999.9
   for p in np.arange(p_min, p_max, p_step):
      for a in np.arange(a_min, a_max, a_step):
         (mu, std) = ZetaGeom(p, a)
         delta = np.sqrt((mu - target_mu)**2 + (std - target_std)**2)
         if delta < min_delta:
            p_best = p
            a_best = a
            mu_best = mu
            std_best = std
            min_delta = delta
   return(p_best, a_best, mu_best, std_best, min_delta)

#--- estimating p and a based on observed mean and standard deviation

target_mu = 1.095 # mean
target_std = 1.205 # standard deviation

p = 0.5
a = 0.0
step_p = 0.4
step_a = 3.0

for level in range(3):
   step_p /= 2
   step_a /= 2
   p_min = max(0, p - step_p)
   p_max = p + step_p
   a_min = a - step_a
   a_max = a + step_a
   grid_range = [(p_min, p_max),(a_min, a_max)]
   (p, a, mu, std, min_delta) = grid_search(grid_range)
   print("delta: %6.4f mu: %6.4f std: %6.4f p: %6.4f a: %6.4f"
       % (min_delta, mu, std, p, a))

# now (p_fit, a_fit) is such that (mean, std) = (target_mu, target_std)
p_fit = p
a_fit = a

#--- sampling from ZetaGeom[p, a]

def CDF(p, a):
   C = 0
   for k in range(100):
      C += p**k/(k+1)**a
   arr_CDF = []
```

```
    CDF = 0
    for k in range(100):
        CDF += (p**k/(k+1)**a)/C
        arr_CDF.append(CDF)
    return(arr_CDF)

def sample_from_CDF(p, a):
    u = np.random.uniform(0,1)
    k = 0
    arr_CDF = CDF(p, a)
    while u > arr_CDF[k]:
        k = k+1
    return(k)

#--- sample using estimated p, a to match target mean and stdev

nobs = 50000 # number of deviates to produce
seed = 500
np.random.seed(seed)
sample1 = np.empty(nobs)
for n in range(nobs):
    k = sample_from_CDF(p_fit, a_fit)
    sample1[n] = k

mean = np.mean(sample1)
std = np.std(sample1)
maxx = max(sample1)
print("\nSample stats: mean: %5.3f std: %5.3f max: %5.3f"
   % (mean, std, maxx))

#--- optional: plotting approximation error for p, a

from mpl_toolkits import mplot3d
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cm # color maps

xa = np.arange(0.0, 0.6, 0.005)
ya = np.arange(-3.0, 0.0, 0.025)
xa, ya = np.meshgrid(xa, ya)
za = np.empty(shape=(len(xa),len(ya)))

kk = 0
for p in np.arange(0.0, 0.6, 0.005):
    hh = 0
    for a in np.arange(-3.0, 0.0, 0.025):
        (mu, std) = ZetaGeom(p, a)
        delta = np.sqrt((mu - target_mu)**2 + (std - target_std)**2)
        za[hh, kk] = delta
        hh += 1
    kk += 1

mpl.rcParams['axes.linewidth'] = 0.5
fig = plt.figure()
axes = plt.axes()
axes.tick_params(axis='both', which='major', labelsize=8)
axes.tick_params(axis='both', which='minor', labelsize=8)
CS = axes.contour(xa, ya, za, levels=150, cmap=cm.coolwarm, linewidths=0.35)
cbar = fig.colorbar(CS, ax = axes, shrink = 0.8, aspect = 5)
cbar.ax.tick_params(labelsize=8)
plt.show()

#--- compare with zeta with same mean mu = 1.095

p = 1.0
a = 2.33 # a < 3 thus var is infinite
```

```
sample2 = np.empty(nobs)
for n in range(nobs):
   k = sample_from_CDF(p, a)
   sample2[n] = k

mean = np.mean(sample2)
std = np.std(sample2)
maxx = max(sample2)
print("Sample stats Zeta: mean: %5.3f std: %5.3f max: %5.3f"
  % (mean, std, maxx))

#--- compare with geom with same mean mu = 1.095

p = target_mu/(1 + target_mu)
a = 0.0
sample3 = np.empty(nobs)
for n in range(nobs):
   k = sample_from_CDF(p, a)
   sample3[n] = k

mean = np.mean(sample3)
std = np.std(sample3)
maxx = max(sample3)
print("Sample stats Geom: mean: %5.3f std: %5.3f max: %5.3f"
  % (mean, std, maxx))

#--- plot probability density functions

axes.tick_params(axis='both', which='major', labelsize=4)
axes.tick_params(axis='both', which='minor', labelsize=4)
mpl.rc('xtick', labelsize=8)
mpl.rc('ytick', labelsize=8)
plt.xlim(-0.5,9.5)
plt.ylim(0,0.8)

cdf1 = CDF(p_fit, a_fit)
cdf2 = CDF(1.0, 2.33)
cdf3 = CDF(target_mu/(1+target_mu), 0.0)

for k in range(10):
   if k == 0:
      pdf1 = cdf1[0]
      pdf2 = cdf2[0]
      pdf3 = cdf3[0]
   else:
      pdf1 = cdf1[k] - cdf1[k-1]
      pdf2 = cdf2[k] - cdf2[k-1]
      pdf3 = cdf3[k] - cdf3[k-1]
   plt.xticks(np.linspace(0,9,num=10))
   plt.plot([k+0.2,k+0.2],[0,pdf1],linewidth=5, c='tab:green', label='Zeta-geom')
   plt.plot([k-0.2,k-0.2],[0,pdf2],linewidth=5, c='tab:orange',label='Zeta')
   plt.plot([k,k],[0,pdf3],linewidth=5, c='tab:gray',label='Geom')

plt.legend(['Zeta-geom','Zeta','Geom'],fontsize = 7)
plt.show()
```

References

[1] Juliano Bortolini et al. The extended generalized gamma geometric distribution. *International Journal of Statistics and Probability*, 6:48–69, 2017. [Link]. 1

[2] Manuel Galvan. The multivariate bisection algorithm. *Preprint*, pages 1–19, 2017. arXiv:1702.05542 [Link]. 4

[3] Vincent Granville. *Synthetic Data and Generative AI*. MLTechniques.com, 2022. [Link]. 3