Vector Graphic Compression in Python

Jakub Soboń,[1] Maksymilian Ogiela,[1] and Paweł Ptaszek[1]
[1] *University of Warsaw*
(Dated: December 2024)

CONTENTS

I. ABSTRACT

Due to the rising amount of experimental data produced in research studies, the efficiency of data visualization becomes important to consider. One specific issue that arises frequently when dealing with large datasets is an overlapping of data points that, when plotted with vector graphics, leads to a vast increase of plot filesize without actual changes to the plot's appearance. This report details a Python package for vector PDF compression that aims to alleviate that problem.

II. INTRODUCTION

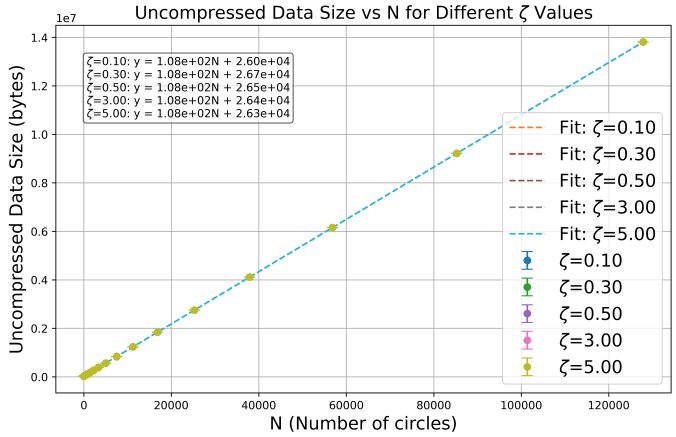
Given that Python, with libraries like `Matplotlib` [1], is a predominant tool for creating graphs, researchers often find themselves constrained by its limitations. To address these challenges, this project proposes readily available solutions aimed at enhancing graph plotting functionality and supporting researchers in overcoming these visualization bottlenecks. Nowadays, it is common to plot data on graphs using different markers, yet they are all coded using vector graphics. This means that every single curve or line is parametrized using equations. In such a parametrization system a circle is described with 8 cubic Bezier curves one for each $(\pi/4)^\circ$ of a circle. When there are many circles (of the same diameter), graphic packages such as `matplotlib` will save some space by creating one set of Bezier curves and then assigning this set to many points in space. In other words, one so-called master circle is saved, and later only the mid-points of other circles are being added to the dataset. This, however, might create an overlap of points, which is of no use for any reader of this graph. Such technique will generate only additional data which must be saved. Those empty data should be taken care of without affecting the graphical outcome for example by eliminating doubled data points (note that this is only a visual way of representing dataset and graphically no information is lost).

III. OUR APPROACH

To challenge this problem, it has been decided that the appropriate approach would be to convert such circles written as Bezier curves (the default setting in the `matplotlib` library when saving as an SVG file) into multipolygons. We were able to do this using the open source Shapely [2] library. Then an operation of unary union is

performed on these multipolygons to create one shape containing all the data. This way we eliminate unnecessary points that are covered by others.

By default, `matplotlib` generates a single circle by encoding an excessively large number of points (on the order of N or more) and then drawing a Bezier curve to approximate the circle with a cubic polynomial. For a large number of circles, $N > 10^3$ ("large" number of circles will be defined precisely later), the plot consists of N circles, so the memory usage of the plot grows linearly with N, as shown in Figure 1a.



(a) Plot of data size depending on the number of circles for different ratios of radius to sigma. The linear progression is clearly visible, especially with the fitted linear function.

FIG. 1: Graph of data size for different N.

IV. PROBLEM DEFINITION

Our compression program is based on the probabilistic fact that randomly placed circles on a bounded surface have a nonzero probability of overlapping. Moreover, this probability increases with the number of circles. Because of these overlaps, the effective perimeter is much smaller than the sum of the perimeters of all circles. In particular, some circles end up being completely covered, so their perimeters do not contribute at all to the final shape on the plot.

In physics, it is common for experimental data to follow a normal (Gaussian) distribution. According to the Central Limit Theorem, many independent random variables with the same expected value tend to a normal distribution.

Plots of such experimental data are an ideal example of the inefficiency of generating large numbers of circles. Most of the circles drawn near the peak of the distribution do not contribute to the total visible perimeter on the plot.

Because of the prevalence and utility of the Gaussian distribution, we will use it to model and test our compression program.

To understand the effectiveness of our compression method, we need to model the distribution of circles and the resulting effective perimeter.

V. STOCHASTIC DISTRIBUTIONS

To model the random distribution of circles, we can, without loss of generality, assume that all circles have the same radius R. Their centers are drawn from a probability density $f(x, y)$. We will assume that the distributions in x and y are identical and that x and y are independent random variables. Hence, we define the following probability density function:

$$f(x,y) = \frac{1}{2\pi\sigma^2} \exp\Big(-\frac{x^2+y^2}{2\sigma^2}\Big). \tag{1}$$

An important parameter affecting the behavior of the effective perimeter is the ratio of the circle radius to the width of the distribution, $\zeta = R/\sigma$.

This ratio is significant because the overall shape is completely different for different ratios. When ζ is large, each circle covers a large part of the Gaussian distribution, so after only a few circles, they begin to overlap significantly. Conversely, for small ζ, even for large N, few circles overlap, or they overlap only partially. For very large N, in the center of the distribution, circles start to cover each other completely, while the boundary has a "porous" structure.

This qualitative analysis shows that the key parameters in this system are the dimensionless ratio ζ (the ratio of circle radius to the standard deviation) and the number of circles N.

VI. SIMULATION OF AREA AND PERIMETER

To check the accuracy of our predictions, we conducted simulations of random distributions.

In Figure 2 simulation results for various values of N and $\zeta = 0.1$ are visible.

In Figure 3 simulation results for various values of ζ with a fixed $N = 1000$ are visible.

We then analyzed the total area covered by the circles and the total external perimeter.

Because random distributions yield different circle placements each time—and thus different areas and perimeters—we performed 5 iterations for each setting. We then computed the mean and standard deviation across these 5 iterations, reporting the average values and their uncertainties. The results are shown below:

A clear logarithmic trend can be observed in these functions, which will become more evident as a natural consequence of the stochastic geometry described in the next section.

(a) $N = 100$ (b) $N = 1000$ (c) $N = 10^3$ (d) $N = 10^6$

FIG. 2: Randomly distributed circles for various values of N.

(a) $\zeta = 0.05$ (b) $\zeta = 0.1$ (c) $\zeta = 0.3$ (d) $\zeta = 0.5$

FIG. 3: Randomly distributed circles for various values of ζ with $N = 1000$.

VII. COVERED AREA

Ultimately, we want to compute the predicted perimeter as a function of the number of circles. However, the perimeter turns out to be more challenging to calculate than the covered area. Solving the area problem first provides important insights that will help in determining the perimeter.

A. Probabilistic Model

To estimate the expected covered area for N circles, we assume that the position of each circle's center is given by the Gaussian probability density $f(X, Y)$:

$$f(X,Y) = \frac{1}{2\pi\sigma^2} \exp\Big(-\frac{X^2+Y^2}{2\sigma^2}\Big). \tag{2}$$

The probability $p(x, y)$ that a single circle covers a point (x, y) is:

$$p(x,y) = \int_S f(X,Y)\, dX\, dY, \tag{3}$$

where S is the region defined by

$$\{(X,Y) \in \mathbb{R}^2 : (x-X)^2 + (y-Y)^2 \le R^2\}.$$

1. Coverage Probability

Assuming N circles are placed independently, the probability that none of the circles covers a fixed point (x, y) is

$$(1 - p(x,y))^N . \tag{4}$$

Thus, the probability that (x, y) is covered by at least one circle is

$$P_{\text{cov}}(x,y) = 1 - (1 - p(x,y))^N . \tag{5}$$

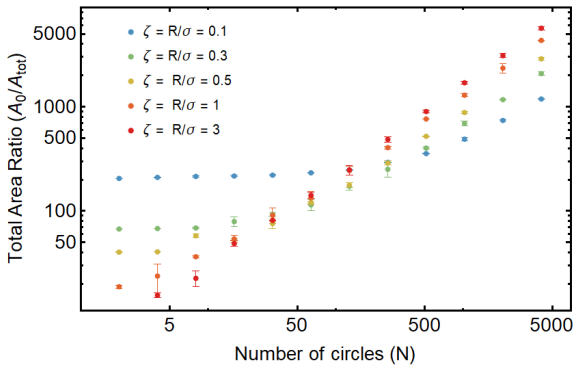
The expected covered area is given by

$$A(N) = \int_{\mathbb{R}^2} P_{\text{cov}}(x,y)\, dx\, dy = \int_{\mathbb{R}^2} \Big[1 - \big(1 - p(x,y)\big)^N\Big]\, dx\, dy.$$

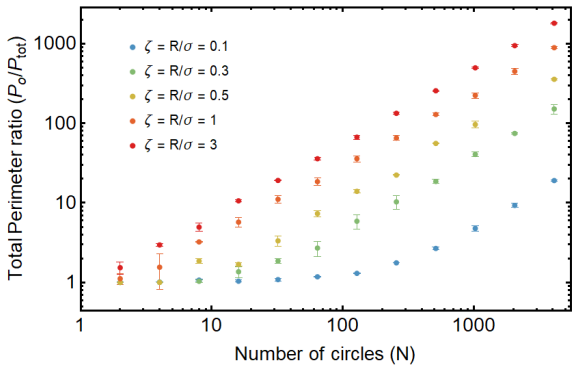
Because this integral does not have a simple closed-form solution, we performed numerical integration for various values of N and ζ. (See Figure 5a for a representative plot.)

B. Asymptotic Behavior

To predict the average covered area for a large number of circles, we estimate the "maximum" distance among



(a) Total area (simulation)



(b) Total perimeter (simulation)

FIG. 4: Simulation results for area and perimeter.

the circle centers. A single random point drawn from a 2D Gaussian $N(0, \sigma^2 I)$ follows the Rayleigh distribution with cumulative distribution function

$$F_R(r) = 1 - \exp\Big(-\frac{r^2}{2\sigma^2}\Big). \tag{6}$$

The probability that all N points lie within a distance r of the origin is

$$\big(F_R(r)\big)^N = \Big(1 - \exp\Big(-\frac{r^2}{2\sigma^2}\Big)\Big)^N. \tag{7}$$

Assuming

$$\big(F_R(r_{\max})\big)^N \approx 1 - \frac{1}{N}, \tag{8}$$

we find

$$r_{\max} \approx \sigma\sqrt{2\ln N}.$$

Assuming the effective area is approximated by a circle of radius $r_{\max} + R$:

$$A(N) \approx \pi(r_{\max} + R)^2 \approx \pi\Big(\sigma\sqrt{2\ln N} + R\Big)^2. \tag{9}$$

For large N, the area scales approximately as

$$A(N) \approx 2\pi\sigma^2 \ln N + c, \tag{10}$$

where

$$c \approx \pi\Big(2\sigma^2 \ln\Big(\frac{R^2}{\sigma^2}\Big) + R^2\Big). \tag{11}$$

C. Model Validation

The simulation results agree with these asymptotic predictions:

VIII. PERIMETER

We cannot easily compute the expected perimeter by standard stochastic geometry methods. For this reason, we use an approximation that provides qualitative predictions.

We choose a uniform grid distribution, placing all circles in a rectangular "crystal" arrangement with spacing d.

IX. ESTIMATION VIA A UNIFORM DISTRIBUTION

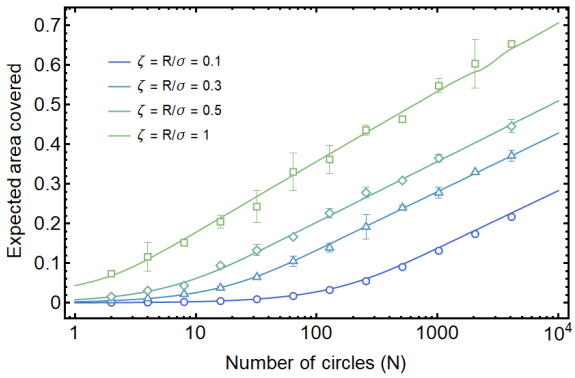
Our procedure for estimating the perimeter is as follows:

1. For given N and ζ, numerically compute the expected area $A(N, \sigma, \zeta)$ of the randomly placed circles.
2. Determine the spacing d in the uniform arrangement so that the total area $A_{\text{tot}}(N, d, \zeta)$ matches $A(N, \sigma, \zeta)$.
3. Compute the perimeter of the uniform arrangement with spacing d and use it to approximate the perimeter of the random distribution.

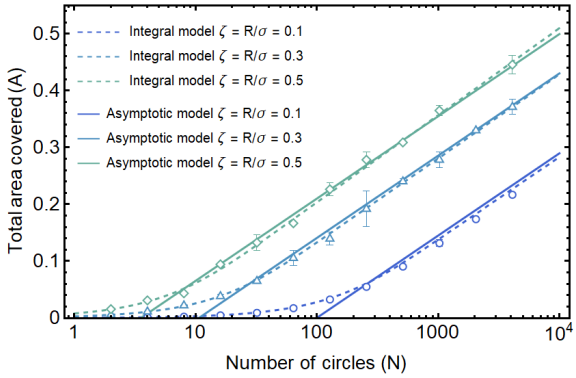
Uniform circle arrangement examples are shown here 6c

A. Effective Length Scale

An important parameter influencing the shape is the ratio of the circle radius to the spacing d. We distinguish

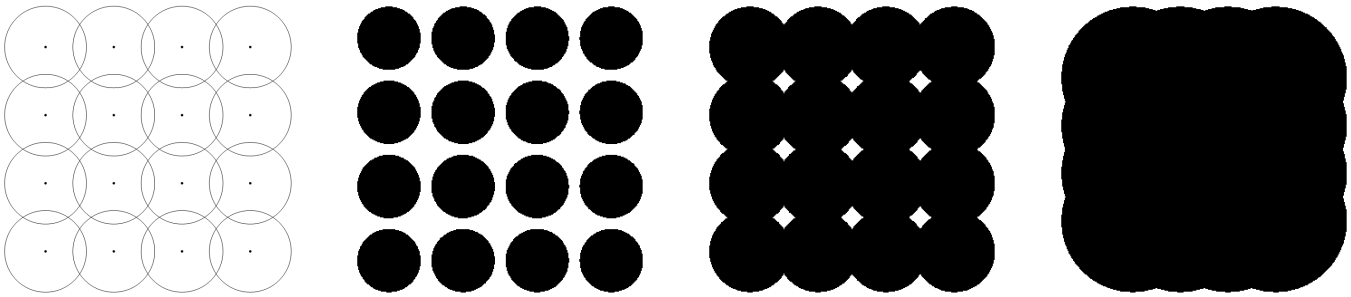


(a) Integral model vs. simulation ($\sigma = 0.1$)



(b) Integral, asymptotic, and simulation ($\sigma = 0.1$)

FIG. 5: Comparison of different models and simulation data.



(a) $d/R = 1.7$ (b) $d/R = 2.3$ (c) $d/R = 1.7$ (d) $d/R = 1$

FIG. 6: Uniform circle arrangement examples.

three regimes:

1. $d > 2R$ (see Figure 6c)
2. $\sqrt{2}\,R < d < 2R$ (see Figure 6c)
3. $d < \sqrt{2}\,R$ (see Figure 6c)

A useful angle is defined as

$$\theta = \cos^{-1}\Big(\frac{d}{2R}\Big),$$

the angle between the line connecting two nearest circle centers and the line from the center to the intersection of the circles.

B. Area

Assuming the circles are arranged in a square crystal of size $k \times k$, the total area is computed as follows:

1. For $d > 2R$:

$$A^1_{\text{tot}} = k^2\pi R^2.$$

2. For $\sqrt{2}\,R < d < 2R$:

$$A^2_{\text{tot}} = k\Big[(k-1)\Big(d\sqrt{4R^2-d^2} - 4R^2\sec^{-1}\Big(\frac{2R}{d}\Big)\Big) + \pi k R^2\Big].$$

3. For $d < \sqrt{2}\,R$:

$$A_{\text{in}} = ((k-1)d)^2, \quad A_{\text{out}} = R^2\Big[2(k-1)\Big(-2\theta+\sin(2\theta)+\pi\Big)+\pi\Big],$$

and

$$A^3_{\text{tot}} = A_{\text{in}} + A_{\text{out}}.$$

Thus, the total area is

$$A(d) = \begin{cases} A^1_{\text{tot}} & d > 2R, \\ A^2_{\text{tot}} & \sqrt{2}\,R < d < 2R, \\ A^3_{\text{tot}} & d < \sqrt{2}\,R. \end{cases} \tag{12}$$

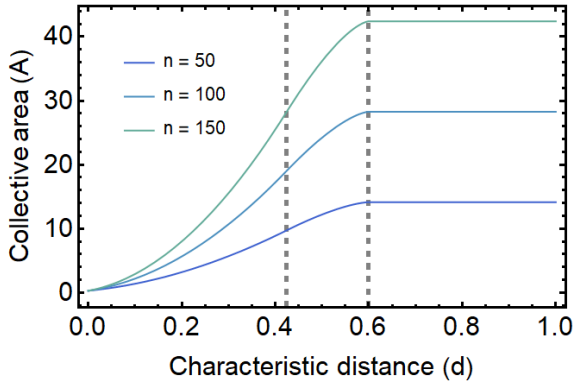
Our models for the uniform distribution are shown in Figure 7.

C. Perimeter

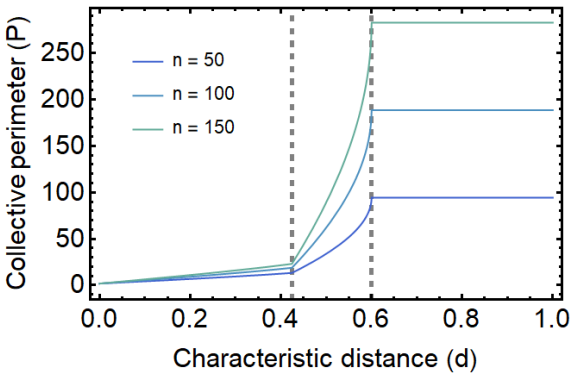
We similarly divide the perimeter into three regimes:

1. For $d > 2R$:

$$P^1_{\text{tot}} = 2k^2\pi R.$$



(a) Area model



(b) Perimeter model

FIG. 7: Model predictions for the uniform distribution.

2. For $\sqrt{2}\,R < d < 2R$:

$$P_{\text{in}}^2 = 2(\pi - 4\theta)(k-1)^2 R, \quad P_{\text{out}}^2 = 2R\Big(\pi(2k-1) - 4\theta(k-1)\Big),$$

so that

$$P_{\text{tot}}^2 = P_{\text{in}}^2 + P_{\text{out}}^2.$$

3. For $d < \sqrt{2}\,R$:

$$P_{\text{tot}}^3 = P_{\text{out}}.$$

Thus, the perimeter is

$$P(d) = \begin{cases} P_{\text{tot}}^1 & d > 2R, \\ P_{\text{tot}}^2 & \sqrt{2}\,R < d < 2R, \\ P_{\text{tot}}^3 & d < \sqrt{2}\,R. \end{cases} \tag{13}$$

Here are the model predictions 7.

X. DETERMINING THE EFFECTIVE PERIMETER

A. Determining the Effective Length Scale

Since we have a theoretical expression for the covered area of the Gaussian distribution, $A(N, \sigma, \zeta)$, and for the uniform arrangement, $A_{\text{tot}}(N, d, \zeta)$, we set them equal and solve for d as a function of N and ζ:

$$A(N, \sigma, \zeta) = A_{\text{tot}}(N, d, \zeta) \quad \Longrightarrow \quad d_{\text{eff}}(N, \sigma, \zeta).$$

For large N, typically $d_{\text{eff}} < \sqrt{2}\,R$.

XI. PERIMETER PREDICTIONS

We approximate the total perimeter of the random geometry by calculating the perimeter of the uniform arrangement with effective spacing:

$$P_{\text{eff}}(N, \sigma, \zeta) = P\Big(d_{\text{eff}}(N, \sigma, \zeta)\Big).$$

XII. PERIMETER RATIOS

To estimate the effectiveness of our compression, we examine the ratio between the naive sum of circle perimeters and the effective perimeter:

$$\xi = \frac{P_{\text{tot}}^1}{P_{\text{eff}}}.$$

The corresponding plot is shown in Figure 8.

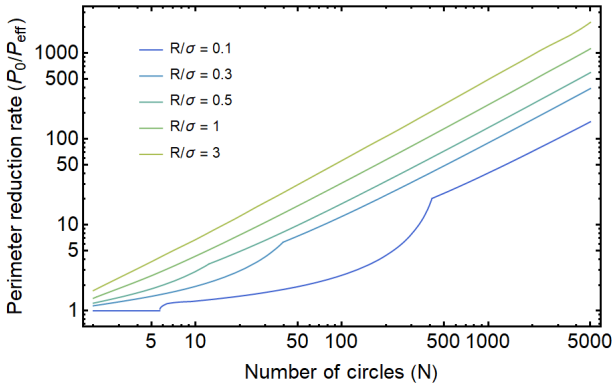


FIG. 8: Perimeter reduction value ξ.

XIII. GLOBAL SCALING

It is clear that our compression model becomes significantly more effective for large N, where it can be thousands of times more efficient than no compression at all. The efficiency, ξ, scales approximately as

$$\xi \sim N,$$

which means the efficiency of compression grows in proportion to the number of circles in the plot.

XIV. PREDICTIONS FOR FILE SIZE

To obtain a full prediction of the compression ratio, we relate the circle perimeters to file size. In our code, each circle is stored as a polygon rather than a true Bezier curve. Thus, the file size before compression is given by

$$W_{\text{before}} = M_{\text{bezier}}\, P^1_{\text{tot}},$$

and the file size after compression is

$$W_{\text{after}} = M_{\text{polygon}}\, P_{\text{eff}},$$

with typically $M_{\text{bezier}} < M_{\text{polygon}}$. The overall compression ratio is

$$\Xi = \frac{W_{\text{before}}}{W_{\text{after}}}.$$

INSERT PLOTS FOR Ξ

XV. THEORETICAL CONCLUSIONS

Conclusion coming from all this theoretical model should be that the compression behaves linearly as n growths'.

XVI. OUR PROGRAM

Our program takes as input the names of two files: the first is the graph we wish to compress, and the second is the same graph but without data points (only frame, legend, etc.). Using the module *xml.etree.ElementTree* [3], we retrieve all necessary information about the data points (position, radius, style) to create a compressed graph. Then, using the *shapely library*, we approximate the circles representing data points as multipolygons and perform a unary union to create one path describing the entire area covered on the graph. Finally, we insert this shape into the clear SVG plot (the second input), yielding an output SVG file of our compressed graph.

XVII. RESULTS OF COMPRESSION

A. Compression Criterion compression ratio

The compression algorithm is not always efficient as can be seen in Figure 9b. Two regimes can be observed in this graph, one in which compression size growths rapidly, till the area is almost fully covered with data points, and the other one in which the compressed size stops growing and only uncompressed size is getting larger thus obtaining linear growth of compression ratio which can be seen in Figure 9a. This specific compression criterion is dependent on the ratio between the radius of the circle and sigma of the Gaussian distribution, but is generally in range of 10 thousand and 100 thousand data points. The boundary of 10 thousand points can be now defined as a limiting criteria for compression as in general data points are within higher boundary of the ζ ratio

B. Time of Execution of the Program

One important factor for using the program is the execution time, in particular its dependence on the number of points in the plot. In order to examine this relation, a Gaussian distribution function with Gaussian noise (in our case $N(0,\, 0.1)$) was used. An example of such a function can be seen in Figure 10b. This procedure was repeated for different numbers of points, and the results are shown in Figure 10a. It is of no surprise that the algorithm has a complexity of O(n) as we are performing linear operations of addition and unary union without any multiplication in between. The strange behavior when the number of points is low should be attributed to some constant-time operations which do not depend on number of data points in particular yet are necessary to be executed.

C. Flaws

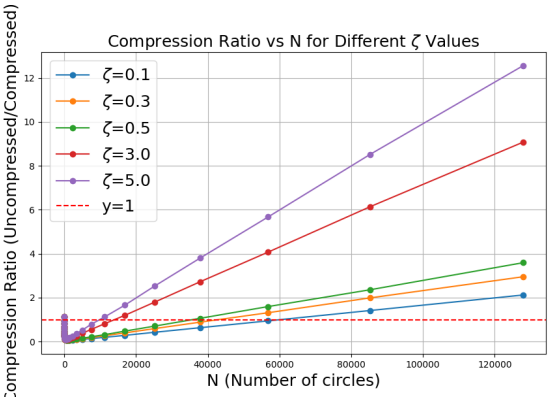
When zoomed in closely on a single point in the chart, slight bends in the lines become visible due to the approximation of circles as polygons. However, this effect is negligible, as illustrated in Figures 11a and 11b, since the number of line segments used to approximate a circle in a multipolygon is relatively large compared to the circle's radius. The amount of lines that are needed to approximate a circle is selected by shapely algorithm.

XVIII. LIMITATIONS AND FURTHER DEVELOPMENT

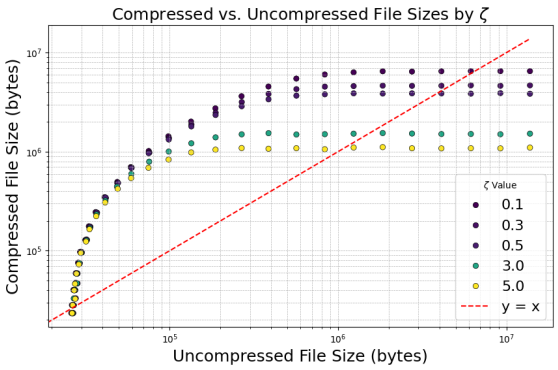
This algorithm requires further improvements as it is executable only for graphs generated in matplotlib and consisting of circles. Another part of the code that requires further investigation is memory constrain. this should be checked in later versions and some optimization might be required as currently from our use we can easily achieve almost 16Gb of RAM usage.

[1] T. M. D. Team, Matplotlib: Visualization with python (2025), accessed: [January 2025 Date].

[2] T. S. D. Team, Shapely 2.0.7 documentation (2025), accessed: [January 2025 Date].

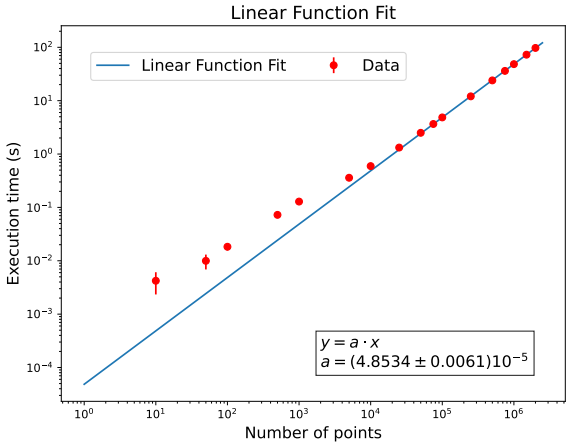


(a) Compression ratio for different amount of circles and different ratio of radius and sigma. Above the horizontal line compression occurs and linear progression of the compression is clearly visible as number of circles growths.

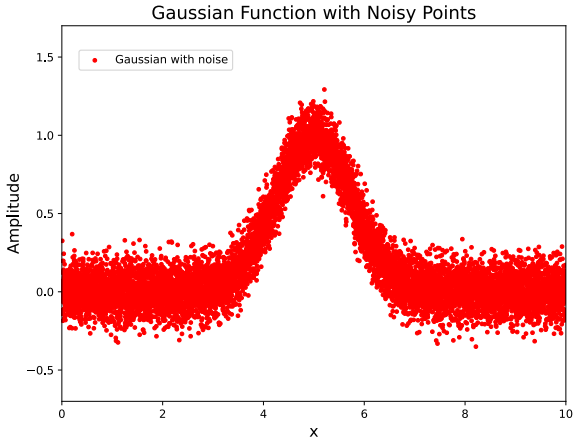


(b) Compressed and uncompressed file sizes. Inefficiency of the model is visible for small size of files

FIG. 9: Compression for different number of circles and correlating sizes



(a) Time of execution vs. number of points. The fitted function is $y = ax$, where $a = (4.8534 \pm 0.0061) \times 10^{-5}$ s.



(b) Example of a compressed plot consisting of 10,000 points.

FIG. 10: Comparison of time complexity and a compressed noisy Gaussian plot.

[3] Python Software Foundation, The ElementTree xml api (2024), accessed: 2025-02-15.

(a) Graph before compression (25600% zoom).

(b) Graph after compression (25600% zoom).

FIG. 11: Comparison of the graph before and after compression.