University of Amsterdam

Assignment 1

Approximating the area of the Mandelbrot set using the Monte Carlo approach

November 22, 2021

Students:
Daniel Versluis
11681101

Sander Broos
11895616

Course:
Stochastic Simulation

Abstract

The area of the Mandelbrot set can be approximated using the Monte Carlo approach. The accuracy of this value of the area does not only depend on the amount of samples points, but also on the distribution of these points and the chosen search area. We investigated how quickly the radius of the confidence interval for the area converges with the amount of samples taken (s) using four different approaches of picking the points for the Monte Carlo simulation. The most straightforward method of generating points is choosing their coordinates completely randomly and independently. This approach is the least accurate of the ones tested, and converges the slowest. The Latin hypercube method distributes the points more evenly over the search area, resulting in a quicker convergence. The orthogonal sampling method divides the search area into smaller regions with an equal amount of sample points. This method is more accurate than the previously mentioned two, and becomes more accurate when the search space is divided into more smaller blocks. Finally, we investigated a method to reduce the size of the search space by generating a low-resolution low-iteration grid of the Mandelbrot set.

Contents

1 Introduction **3**

2 Methods **3**
2.1 Mandelbrot set . . . 3
2.2 Monte Carlo method . . . 3
2.2.1 Pure random sampling . . . 4
2.2.2 Latin hypercube sampling . . . 4
2.2.3 Orthogonal sampling . . . 4
2.2.4 Selective search area . . . 4
2.3 Choosing i . . . 5
2.4 Investigating convergence as $s \to \infty$ . . . 5
2.5 Tools . . . 6

3 Results **6**
3.1 Influence of i on convergence . . . 6
3.2 Influence of s on $A_{i,s}$ for different sampling methods . . . 7
3.3 Accuracy of s for different sampling methods . . . 7

4 Discussion and conclusion **7**

1 Introduction

The Mandelbrot set, and specifically its visual representation, is one of the most well-known mathematical concepts. While the formula it is based on is seemingly simple, plotting it on a grid reveals a striking image with endlessly complex and detailed repeating fractals. There are many aspects that can be investigated about the Mandelbrot set - this report will look at ways to determine the set's area, specifically using the Monte Carlo method. Here, sample "points" are distributed over a region, and the area of the Mandelbrot set can be approximated by determining the fraction of points that are determined to lie in the set. However, the way these sample points are distributed can make a notable difference to the accuracy of the approximation. This report looks at the difference in accuracy between different sampling methods: completely random sampling, Latin hypercube sampling, orthogonal sampling, and a newly proposed method to reduce the size of the search area.

2 Methods

2.1 Mandelbrot set

In order to check whether or not a complex number c is a part of the Mandelbrot set, the iterative formula $z_n = z_{n-1}^2 + c$ is investigated. Starting with $z_0 = 0$, the value of z is changed by calculating the formula repeatedly. If the value of z does not eventually diverge to infinity after iterating, the value c lies in the Mandelbrot set.

An easy way to check if the formula diverges is by checking if the absolute value of z at any point exceeds 2. If it does, the value of z will go to infinity.(Bogdan 2015) The only problem left in this method, is that z could diverge at any iteration n, or not at all when a value lies in the set. It is not possible to calculate an infinite number of iterations, so a stopping point must be chosen - the maximum number of iterations i before we assume a point lies within the Mandelbrot set. The way this stopping point i was chosen for this report is discussed in section 2.3.

To represent the Mandelbrot set visually, this calculation is performed for many points c, where the result for c is plotted on an (x, y) grid using the real and imaginary parts of $c = x+iy$. It can be interesting to not only show whether a point lies in the set, but also show how many iterations it took the points outside of the set to diverge (when the absolute value crosses 2). Such a visualisation is shown in Figure 2a, where a pixel is brighter if it took many iterations for the absolute value to cross 2. Thus, the points that do not diverge within i iterations and are assumed to be in the set are white in the figure.

2.2 Monte Carlo method

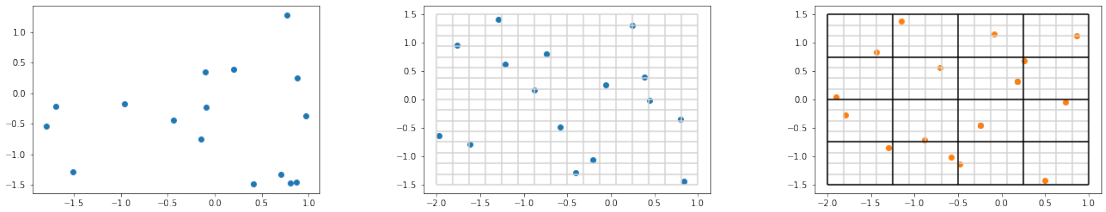
The area of the Mandelbrot set was approximated using a Monte Carlo approach. In this context, given the visual interpretation of the Mandelbrot set on an (x, y) grid where $c = x + iy$, suppose the random vector (X, Y) is uniformly distributed in a predetermined search area on the grid. This search area is usually a rectangle. The probability that the random point is in the Mandelbrot set is given by:

$$P((X, Y) \text{ in Mandelbrot set}) = \frac{\text{Area of Mandelbrot set}}{\text{Area of search area}} \tag{1}$$

We sample s random points in the search area, evaluating for every point whether z_n diverges to infinity in i iterations. By counting the fraction of points for which this happens, we are able to approximate the probability in equation 1. Since the area of the search area is known, this enables us to then make an estimate $A_{i,s}$ of the area of the Mandelbrot set itself:

$$A_{i,s} = \text{Fraction of points inside set} \cdot \text{Area of search area} \tag{2}$$

The accuracy of this value for the area not only depends on the number of samples s, but also on other factors. We investigated the effect of choosing different methods of random number sampling on the convergence rate of $A_{i,s}$ as s and i tended to infinity. The sampling methods are described below.



(a) Random sampling

(b) Latin hypercube sampling

(c) Orthogonal sampling with 4 divisions (16 blocks)

Figure 1: Different sampling methods with $s = 16$ sample points.

2.2.1 Pure random sampling

To generate uniform random numbers for the sampling of points, we used the *random.uniform* function from the Python package *numpy*. By default, this function uses a pseudorandom number generator known as the PCG 64 algorithm, which uses a Permuted Congruential Generator(O'Neill 2014). For each of the s sample points, one random X coordinate is chosen between the left and right bounds of the rectangular search area using this function. The same is done for the point's Y coordinate, between the bottom and top bounds of the rectangle. An example of points distributed this way is shown in Figure 1a.

This method can have some downsides. Since the coordinates of the points are completely independent from each other, it is possible that, by chance, some regions emerge with a higher density of sample points than the rest of the search area. If this cluster is inside the Mandelbrot area, it could skew the results and result in a larger approximated value of the area than the true value. Thus, the goal is to make sure the random sample points are as uniformly distributed over the search area as possible, and some other sampling methods are used to achieve this.

2.2.2 Latin hypercube sampling

In the **Latin hypercube sampling** method, the rectangular search area is divided up into s columns and s rows, where each column and each row can only contain one of the s samples. To choose the coordinates of a new sample point, a random column and a random row are chosen from the ones that do not already contain another point. Then, using the same *random.uniform* function as in section 2.2.1, a random point is picked between the column and row bounds. An example of 16 points distributed this way is shown in Figure 1b with the rows and columns clearly visualised. It can be seen how each row and column contains 1 sample point.

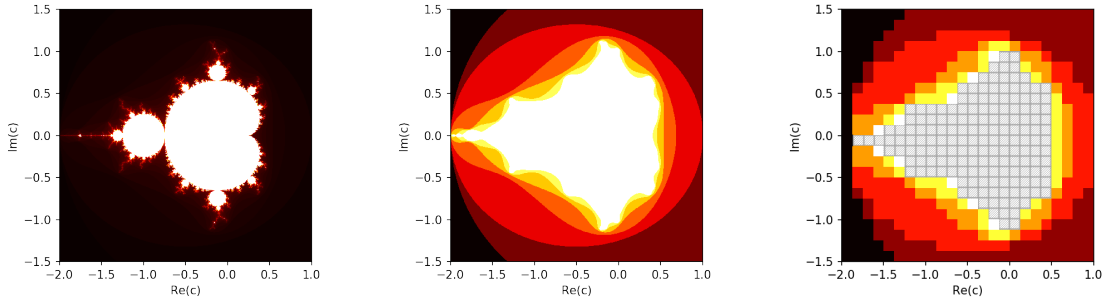
2.2.3 Orthogonal sampling

The **orthogonal sampling** method uses Latin hypercube sampling, but goes one step further to ensure a uniform distribution of sample points. Here, the rectangular search area is divided into b smaller rectangles (blocks), all of which must contain the exact same amount of sample points.

Since this method uses Latin hypercube sampling as well, first the rectangular search area is again subdivided into s rows and s columns, after which it is separately divided into the rectangular blocks. To choose the points in this method, the code handles each block one at a time, and randomly fills it with $\frac{s}{b}$ sample points in the columns and rows that do not already contain another point (even if it's in another block). An example of 16 points distributed this way with 4 divisions in the x and y directions (so $4^2 = 16$ blocks) is shown in Figure 1b with the rows, columns and blocks clearly visualised.

2.2.4 Selective search area

Besides the distribution of the sample points, another factor in the accuracy of the calculated area is the *size* of the search area. The closer the search area is to the actual Mandelbrot set, the more accurate the result will be, assuming we know the area of the search area. If the search



(a) Mandelbrot set with $i = 100$ on a 1000x1000 pixel grid.

(b) Mandelbrot set with $i = 7$ on a 1000x1000 pixel grid.

(c) Mandelbrot set with $i = 100$ on a 24x24 pixel grid, with the white pixels (pixels that did not converge) marked with gridlines.

Figure 2: The Mandelbrot set for different values of the maximum iterations i and the resolution of the grid.

area is much larger than that of the Mandelbrot set, only a small number of sample points would be in the set with an increasing chance that that number is zero - the relative error in the area would be high. If the search area was theoretically exactly equal to that of the Mandelbrot set, 100% of the sample points would be in the set resulting in a perfectly accurate resulting area from equation 2.

To reduce the size of the search area, an "outer bound" is needed for the Mandelbrot set. This can be obtained by rendering the Mandelbrot set with a low stopping point of the amount of iterations for checking if a point diverges. All points in the Mandelbrot set will never diverge to infinity, and the points near the Mandelbrot set will take a longer time to diverge so stopping early will include those points in the calculated set. This is shown in Figure 2b, where the outer dark red circle represents the points that immediately diverged (crossed the threshold value of 2) after one iteration. The brighter red oval within that represents points that diverged one iteration later, and so on for the shapes within those shapes.

Now we render the set in this way at a low resolution, and use the pixels that did not diverge as equally sized "blocks" that together form the search area. These are the marked pixels shown in Figure 2c, which will now be used as the search area. The low-resolution image could miss some of the thinner branches of the Mandelbrot set - for example, the leftmost center pixel in Figure 2c is black while it should be included too, otherwise parts of the set could possibly be excluded from the search resulting in a lower approximated area. Thus, the search area is expanded with a 1 pixel-wide "border" around every outer white pixel as a buffer. Sample points are then chosen by first randomly picking one of these blocks, then randomly placing a point within the block in the same way as it was done in the "pure random sampling" method described in section 2.2.1. The area of the search area (used in equation 2) can easily be calculated since all the blocks are square and adjacent in the grid.

2.3 Choosing i

The effect of increasing the amount of iterations i on the area estimate was assessed by taking a fixed s and a maximum for i, then computing $A_{j,s}$ for $j = 1, 2, ..., i - 1$. By then studying the variable $A_{j,s} - A_{i,s}$ as a function of j, we could determine the effect of increasing i on the accuracy of the area estimation and choose a suitable value for i for the simulations where we investigated the effect of s

2.4 Investigating convergence as $s \to \infty$

Having chosen a value for i, we investigated the effect of s on the convergence rate of the area estimation. This was done by computing $A_{i,s}$ for increasing values of s and assessing the radius

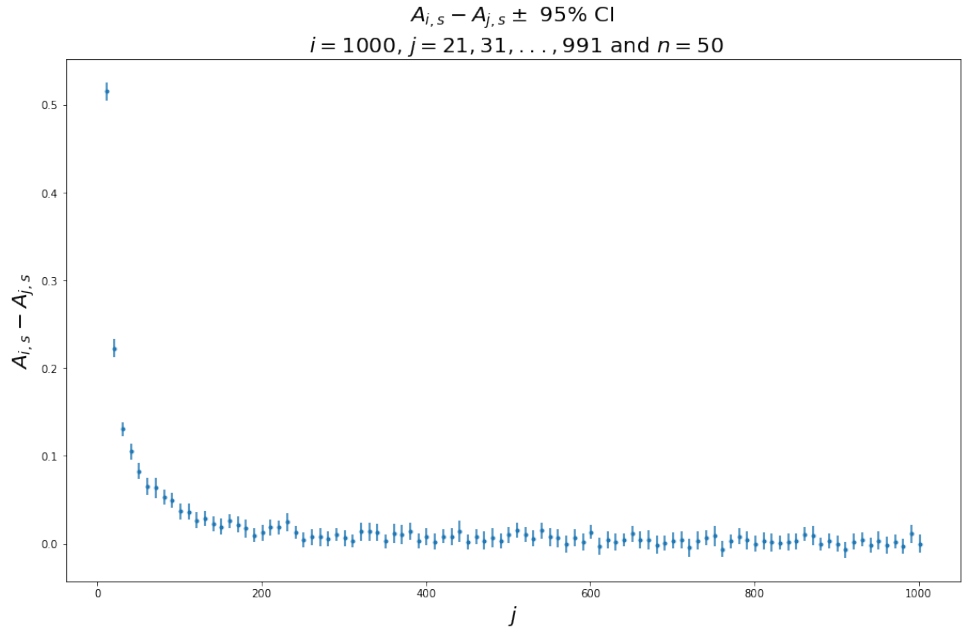


Figure 3: $A_{i,s} - A_{j,s}$ for $j < i$, $i = 1000$, $s = 10000$

a of the 95% confidence interval given by:

$$a = \frac{1.96S}{n} \tag{3}$$

where S is the sample standard deviation and n the number of area samples[1] taken.

2.5 Tools

All simulations were coded and run in Python using Jupyter Notebook. The *numpy* and *math* packages were used for mathematical functions and the *pandas* library for the handling of data. In order to create the plots, the *seaborn* and *matplotlib* libraries were used.

3 Results

3.1 Influence of i on convergence

To assess the effect of increasing i on $A_{i,s}$, the value $A_{i,s} - A_{j,s}$ for $j < i$ was computed using the pure random sampling method. The results for $i = 10^3$ and $s = 10^4$ can be seen in Figure 3. For increasing i, the value $A_{i,s} - A_{j,s}$ decreases in a seemingly exponential manner, eventually levelling out to a value close to 0. In order to more closely look into the behaviour for lower values of i, simulations were also run for $i = 100$ with smaller steps of j. The results of this can be seen in Figure 4.

From this data we were able to pick a suitable value for i to use in the simulations we performed next, where we investigated the influence of increasing s for different sampling methods. Since the simulations for large i quickly became computationally expensive and therefore time-consuming, we wished to pick a value of i that was as low as possible while still producing relatively low errors. We chose an $A_{i,s} - A_{j,s}$ value of 0.1 as the maximum error to tolerate, and then settled on the value $i = 50$ since $A_{i,s} - A_{50,s}$ for this value was lower than 0.1 and the function did not decrease very substantially for higher i.

[1] Not to be confused with sample points

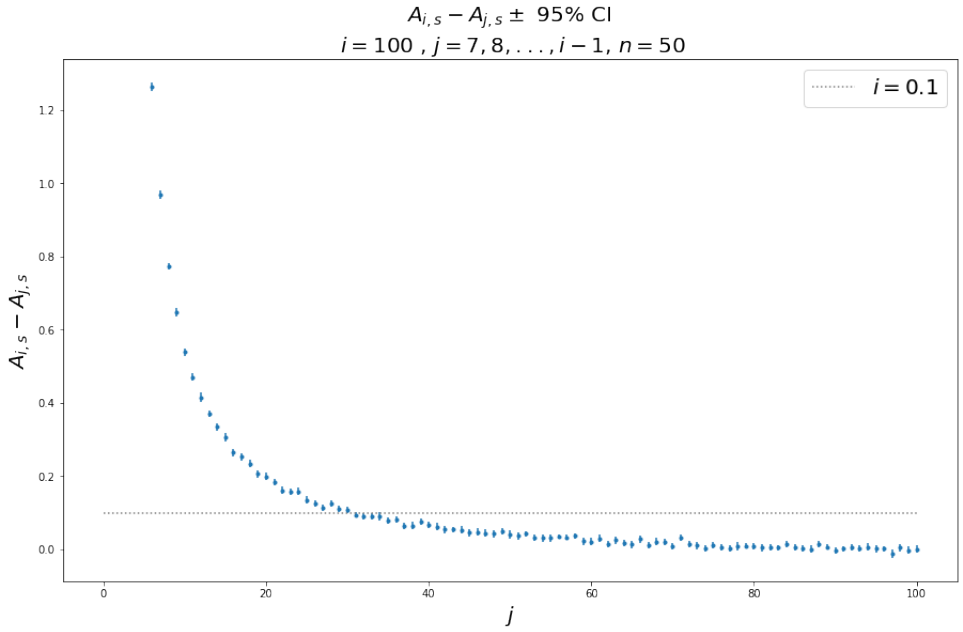


Figure 4: $A_{i,s} - A_{j,s}$ for $j < i$, $i = 100$

3.2 Influence of s on $A_{i,s}$ for different sampling methods

Having picked an i value of 50 to work with, we investigated the convergence of $A_{i,s}$ as s tended to infinity. In figure 5a, the results can be seen for pure random sampling. Even with the chosen sample size n of 100, $A_{i,s}$ experiences large fluctuations, especially for small values of s. As s becomes larger, $A_{i,s}$ settles and from the decreasing size of the confidence interval we can infer that the accuracy of the estimations increases.

Figure 5b, c, d and e0 show the results of the same simulation for Latin hypercube sampling, orthogonal sampling with 2 divisions and orthogonal sampling with 5 divisions, and the selective search area method, respectively. The amount of divisions refers to the divisions in both the x and y coordinates, so 2 divisions corresponds to 4 blocks while 5 divisions corresponds to 25 blocks. While it is difficult to draw conclusions regarding the accuracy of the estimations from this figure, note that the convergence of $A_{i,s}$ happens more quickly on the latter three sampling methods, and there are less fluctuations. The area computed by all methods converges to the same value around 1.59.

3.3 Accuracy of s for different sampling methods

To be able to draw conclusions on the accuracy of the estimations, the radius a of the 95% confidence interval was computed for all s values of the different sampling methods. A scatter plot of the results can be seen in figure 5. From these data, we can see that Latin hypercube sampling and orthogonal sampling with 2 divisions offer an increase in accuracy when compared to pure random sampling. There seems to be no difference between orthogonal sampling with 2 divisions and Latin hypercube sampling, the amount of samples is high enough that the points are already decently distributed over the 4 quadrants. The results of orthogonal sampling with 5 divisions has a much higher accuracy, as here there are $5^2 = 25$ blocks that have to have an equal amount of samples, resulting in a much more even distribution.

4 Discussion and conclusion

The goal of this report was to compare the results of different methods of random number sampling when applied to Monte Carlo integration of the area of the Mandelbrot set. After having

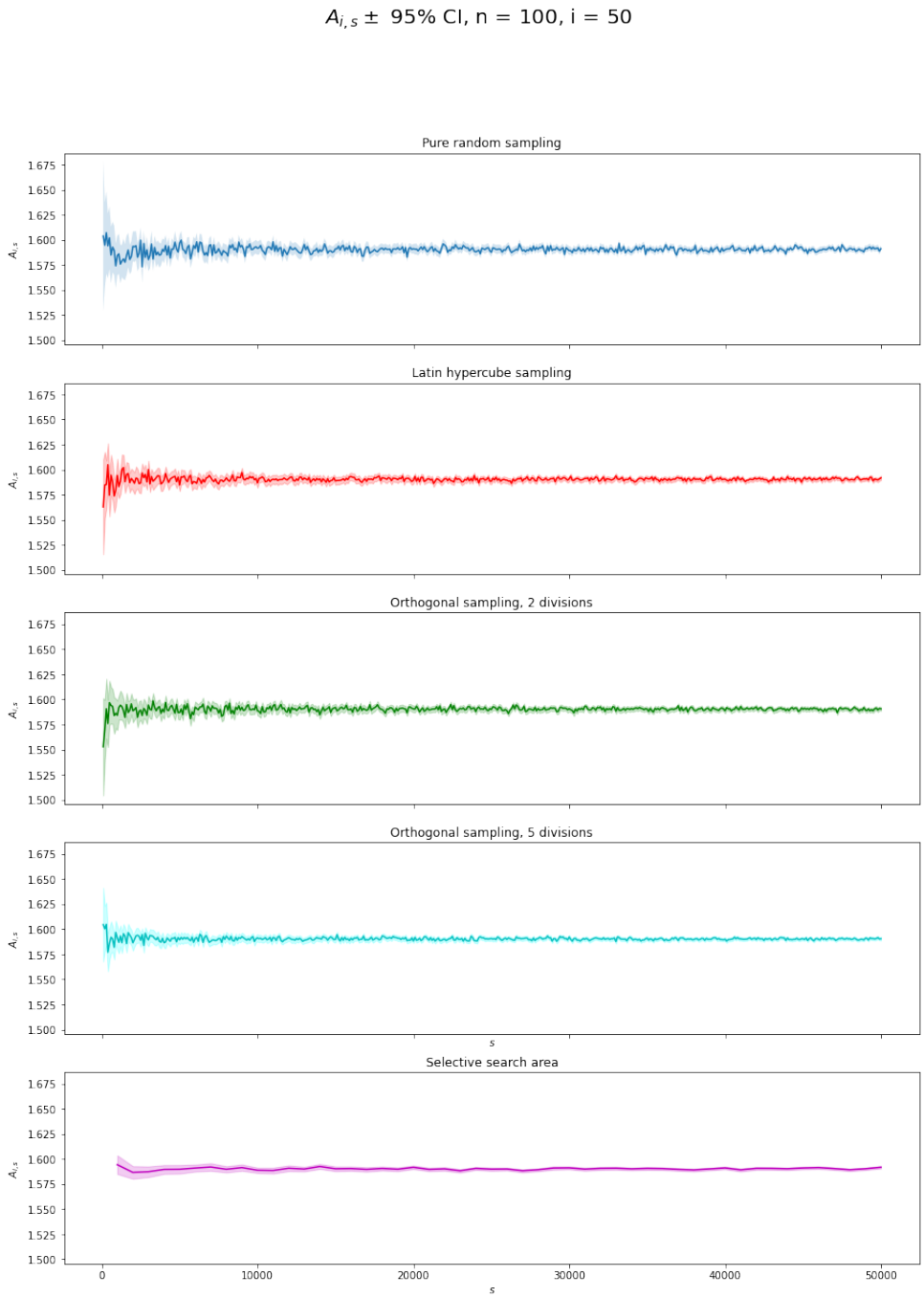


Figure 5: $A_{i,s}$ for different sampling methods, $i = 50$, $n = 100$

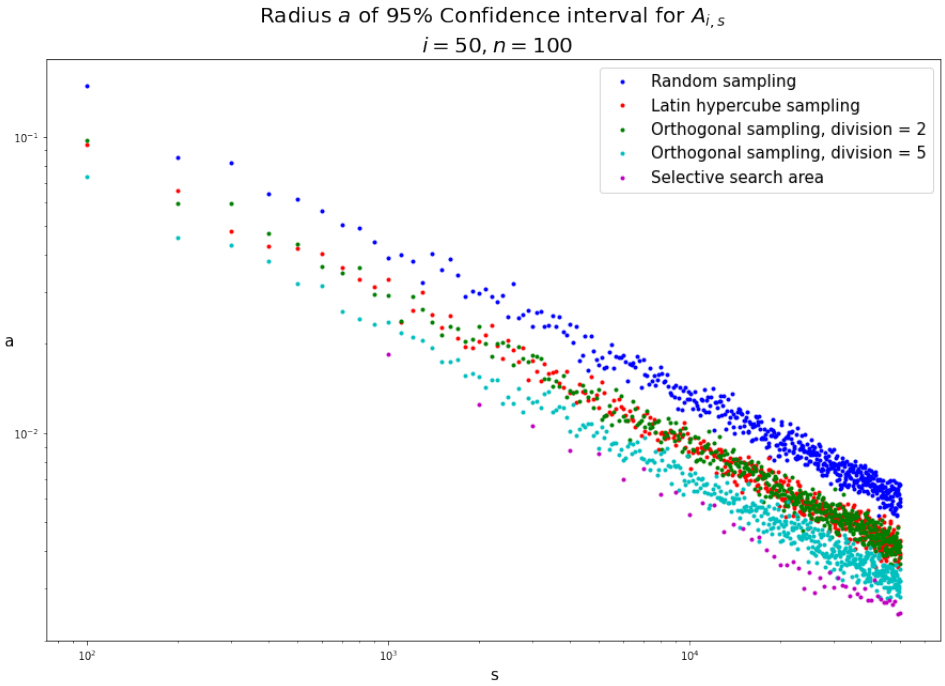


Figure 6: Radius a of the 95% confidence interval for the four sampling methods. $i = 50$, $n = 100$

picked a suitable value for i, the number of iterations for the Mandelbrot set, we set out to investigate the convergence of the area estimation as the number of samples s increased. We saw that all methods eventually converged to roughly the same value. However, our chosen i of 50, chosen for computational efficiency and our maximum s of $5 \cdot 10^4$, chosen because of time and computing power constraints were still relatively low. It is therefore likely the true area of the Mandelbrot set is quite different from this value.

However, the results are still very useful to compare the different Monte Carlo sampling methods. Since the same i was used to compare, the area that was investigated was identical even if it was an imprecise representation of the Mandelbrot set. This was done by analysis of the radius a of the 95% confidence interval, which could be seen as a measure for the accuracy of the results obtained by a specific method. Our results indicate that Latin hypercube sampling and orthogonal sampling are both effective at improving the convergence rate of the approximated area, especially when the number of divisions is very high for the orthogonal sampling method.

The newly proposed "selective search area" method is the most effective, as a higher portion of the sample points will be placed on the Mandelbrot set. An interesting topic for future research would be to combine this method with the orthogonal sampling method, which would distribute the points more evenly over the reduced search area.

References

Bogdan, Popa (2015). "Iterative function systems for natural image processing". In: *Proceedings of the 2015 16th International Carpathian Control Conference (ICCC)*, pp. 46–49. DOI: 10.1109/CarpathianCC.2015.7145043.

O'Neill, Melissa E. (2014). "PCG : A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation". In.