Benchmark Results

Alex Buckley

January 26, 2026

Abstract

This report evaluates the statistical characteristics and execution performance of ten standard benchmark functions using pseudo-random input populations. A population of 30 individuals was generated using an external Mersenne Twister implementation with a fixed seed, and scaled to match each function's defined input bounds. Fitness values were analyzed through summary statistics and distribution visualizations, revealing distinct behaviors across functions, such as high variability in Rosenbrock's Saddle and tightly clustered negative values in the Sine Envelope Sine Wave. Execution time was also measured, demonstrating that computational complexity of the function expressions strongly influences evaluation speed. The findings provide foundational insight into function landscapes and performance implications, which can guide selection and tuning of optimization algorithms in future work.

Contents

List of Figures

List of Tables

1 Overview

Standard benchmark functions provide a normalized method of evaluating optimization algorithms. The known characteristics of the various functions allows experiments which target and test specific aspects of optimization and convergence algorithms.

1.1 Purpose

In this benchmark, we analyze the properties and characteristics of a selection of 10 standard benchmark functions. Functions are input pseudo-random values within set bounds, producing fitness results. In this experiment, a population size of 30, as well as 30 dimensions. The pseudo-random inputs ar generated with an external Mersenne Twister implementation with a seed of 108664. The characteristics of each function's resulting fitness values help provide insight into the behavior and characteristics of each function. While convergence and optimization methods are not used in this experiment, understanding the behavior of functions helps to understand performance of optimization algorithms across differing functions.

Populations for each function in this experiment were generated with the same seed, however, the tested functions have different sensible domains. Thus, while the initial generated values are identical for each function, they are independently scaled to match the appropriate range of input values. Exact bounds used for each function can be seen in table 1.

Table 1: Population bounds for all experiments

Experiment	Lower Bound	Upper Bound
Schwefel	-512.0	512.0
DeJong_1	-100.0	100.0
Rosenbrocks_Saddle	-100.0	100.0
Rastrigin	-30.0	30.0
Griewangk	-500.0	500.0
Sine_Envelope_Sine_Wave	-30.0	30.0
Stretched_V_Sine_Wave	-30.0	30.0
Ackley_One	-32.0	32.0
Ackley_Two	-32.0	32.0
Egg_Holder	-500.0	500.0

2 Experiment

2.1 Benchmark Environment

As execution time makes up a portion of the experiment, Understanding the environment the experiment was run in is crucial. The hardware and software used to run the program can have aa very large effect on the execution times of the benchmarks.

2.1.1 Hardware Environment

In order to provide reliable results, various methods were used to provide consistency across program runs of the C++ benchmark program. The experiment was conducted with the laptop plugged in and fully charged, with energy-saving mode disabled. Additionally, the benchmark benchmarks were performed with all other apps closed and a CPU temperature validaated to be under 60° C. Furthermore, the `sudo renice -n -20 -p $$` command was executed in the terminal before launching the program. This sets the scheduling priority of the terminal to maximum, potentially reducing the amount of CPU time used by other processes during benchmarking.

2.1.2 Hardware Specifications

All benchmarks were run on a 2018 MacBook Pro running the Sonoma 14.6.1 operating system with a single 2.3 GHz Quad-Core Intel i5 processor. This processor has 4 physical cores, and 8 logical cores, as Hyper-Threading is enabled. Each physical core has access to two L1 caches of 32 KB each, one for instructions and one for data. Additionally, each core has a 256 KB L2 cache, while all cores share a 6 MB L3 cache. The device has 16 GB of LPDDR3 RAM spread across two equally sized banks. Additionally, the memory has a clock speed of 2133 MHz and does not support ECC (error-correcting code).

2.1.3 Timing Excution

This benchmark includes analysis of execution time of the tested functions. Each tested functions outputs 1 single time value alongside its fitness values. The result simply measures the wall-clock time to calculate all fitness values. That is, the combined evaluation time of all members of the population. Population generation and experiment initialization are not included in evaluation time.

2.2 Results

2.2.1 Fitness Values

table 2 summarizes the fitness distributions across all tested standard benchmark functions. Rosenbrock's Saddle shows extremely high values and variability, while Sine Envelope Sine Wave produces tightly clustered negative values with minimal spread. General trends in fitness across different benchmark functions help provide insight into behaviors.

Table 2: Summary statistics of fitness values across experiments

Experiment	Mean	Std	Median	Min	Max	Range	Time (ms)
Schwefel	1.2531e+04	1.2506e+03	1.2420e+04	8.9930e+03	1.4547e+04	5.5544e+03	0.02
DeJong_1	1.0346e+05	1.4695e+04	1.0619e+05	7.6623e+04	1.3204e+05	5.5418e+04	0.00
Rosenbrocks_Saddle	6.1694e+10	1.4346e+10	6.1598e+10	3.8417e+10	8.9215e+10	5.0798e+10	0.00
Rastrigin	9.6058e+03	1.3262e+03	9.8200e+03	7.2441e+03	1.2214e+04	4.9695e+03	0.02
Griewangk	6.4765e+02	9.1842e+01	6.6468e+02	4.7989e+02	8.2626e+02	3.4636e+02	0.03
Sine_Envelope_Sine_Wave	-2.0704e+01	1.0918e+00	-2.0491e+01	-2.2734e+01	-1.8064e+01	4.6698e+00	0.02
Stretched_V_Sine_Wave	9.7030e+01	9.3358e+00	9.6801e+01	8.0492e+01	1.1597e+02	3.5481e+01	0.06
Ackley_One	5.9683e+02	4.6201e+01	5.9257e+02	5.1819e+02	7.0543e+02	1.8724e+02	0.02
Ackley_Two	5.7666e+02	1.0485e+01	5.7552e+02	5.6012e+02	5.9512e+02	3.5003e+01	0.04
Egg_Holder	2.0242e+02	1.6582e+03	3.9818e+02	-3.8403e+03	4.0776e+03	7.9179e+03	0.03

2.2.2 Execution Time

Wall-clock execution time for evaluating standard benchmark functions with pseudo-random populations can be seen in fig. 1. Execution time seems to increase and decrease relative to the types of operations found within the function. For example, Stretched V Sine Wave took notably longer to execute than any other tested function. This function also contains 2 higher-index roots within the summation. This is a multi-instruction expression for thee system to calculate, likely with significant overhead compared to operations like adding, multiplication, and even square roots. Because the summation runs an iteration for each dimension, in this benchmark, Stretched V Sine Wave was required to calculate 60 higher-indexed root operations.

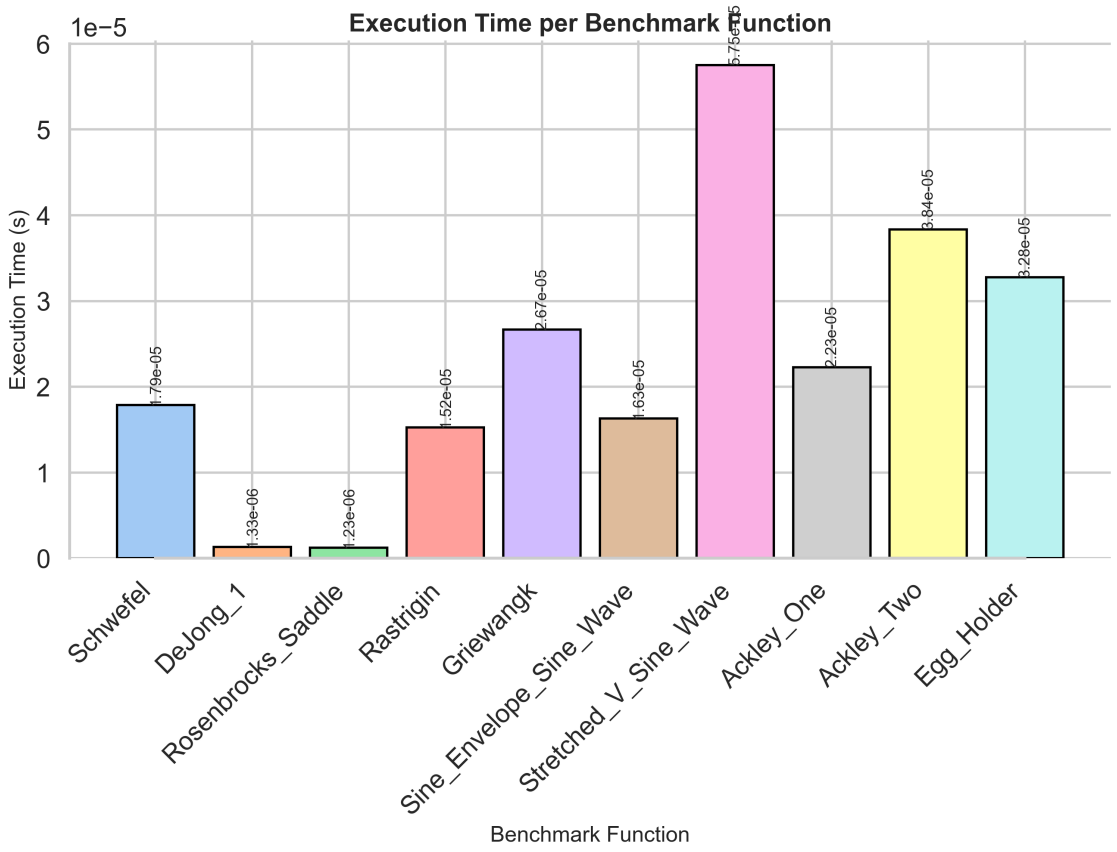

Figure 1: All Experiments Time

Ackley's Two was the second slowest function to evaluate. While this function does not contain any higher-index roots, it does have a fairly complicated summation body, requiring calculation of a square root, two floating-point exponent, and two trig functions. These operations are fairly expensive, and must all be calculated 30 times, once for each dimension.

On the other end of the spectrum, we observe the DeJong 1 Function to have the fastest evaluation time. This is unsurprising, as this function only requires a single multiplication operation for each summation loop. Trailing closely behind in evaluation speed is Rosenbrock's Saddle. While Rosenbrock's Saddle's summation loop is not quite as simple as DeJong 1, it still only requires a minimum of 4 multiplications and 3 additions/subtractions each summation loop (depending on exact implementation). This lack of more complicated operations is likely a large factor in the rapid execution speed. Overall, the number and type of operations iwthin each function's summation loop seems to be the primary factor in evaluation time of the function.

3 Function Characteristic

3.1 Ackley's One

$$f_8(\mathbf{x}) = \sum_{i=1}^{n-1} \left[\frac{1}{e^{0.2}} \sqrt{x_i^2 + x_{i+1}^2} + 3\left(\cos(2x_i) + \sin(2x_{i+1})\right) \right]$$

The Ackley function is a common benchmark in optimization literature [1]. Inputting pseudo-randomly generated data into the function consistently produced right-skewed data. The results Mean being larger

than the median demonstrates this, as shown in fig. 2. Additionally, fig. 3 visually demonstrates the skew towards higher fitness values.

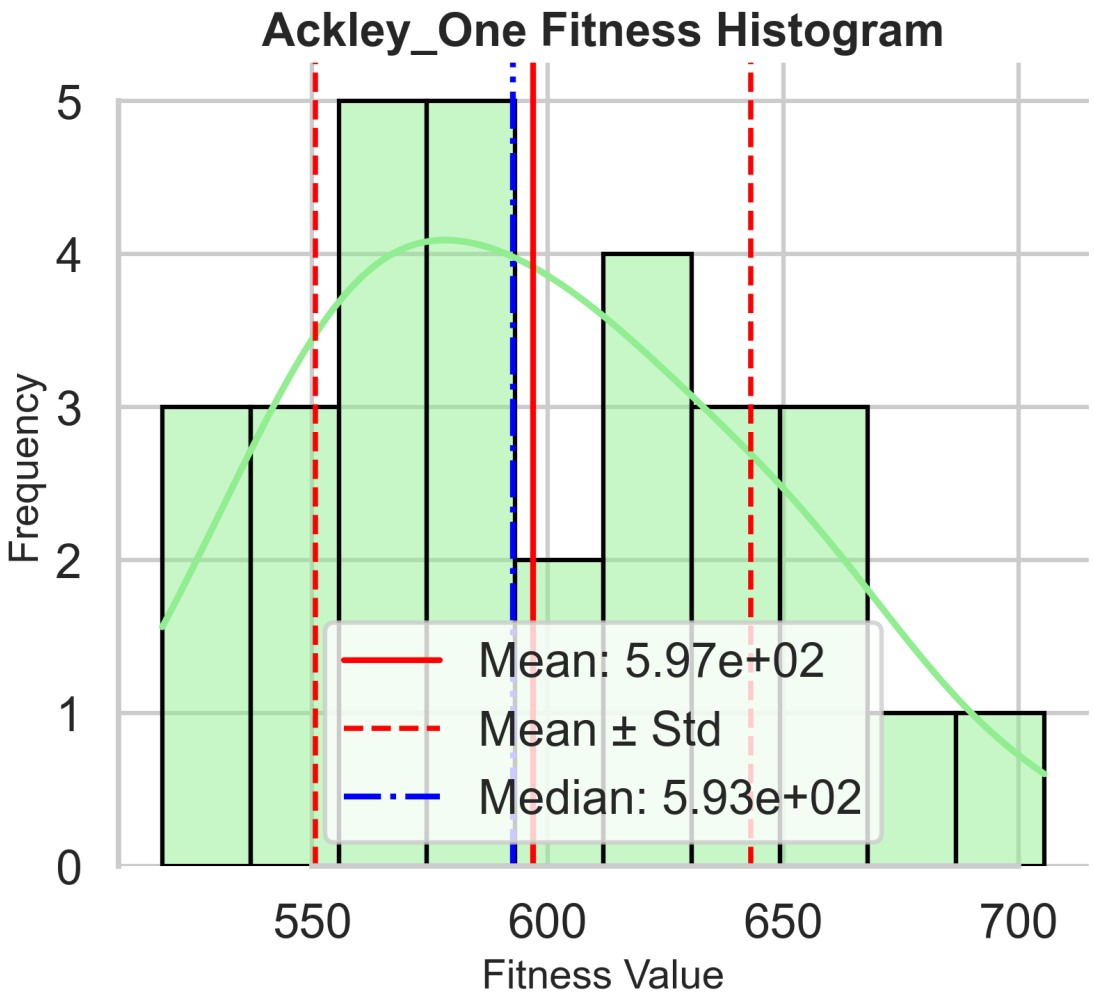

Figure 2: Ackley One Histogram

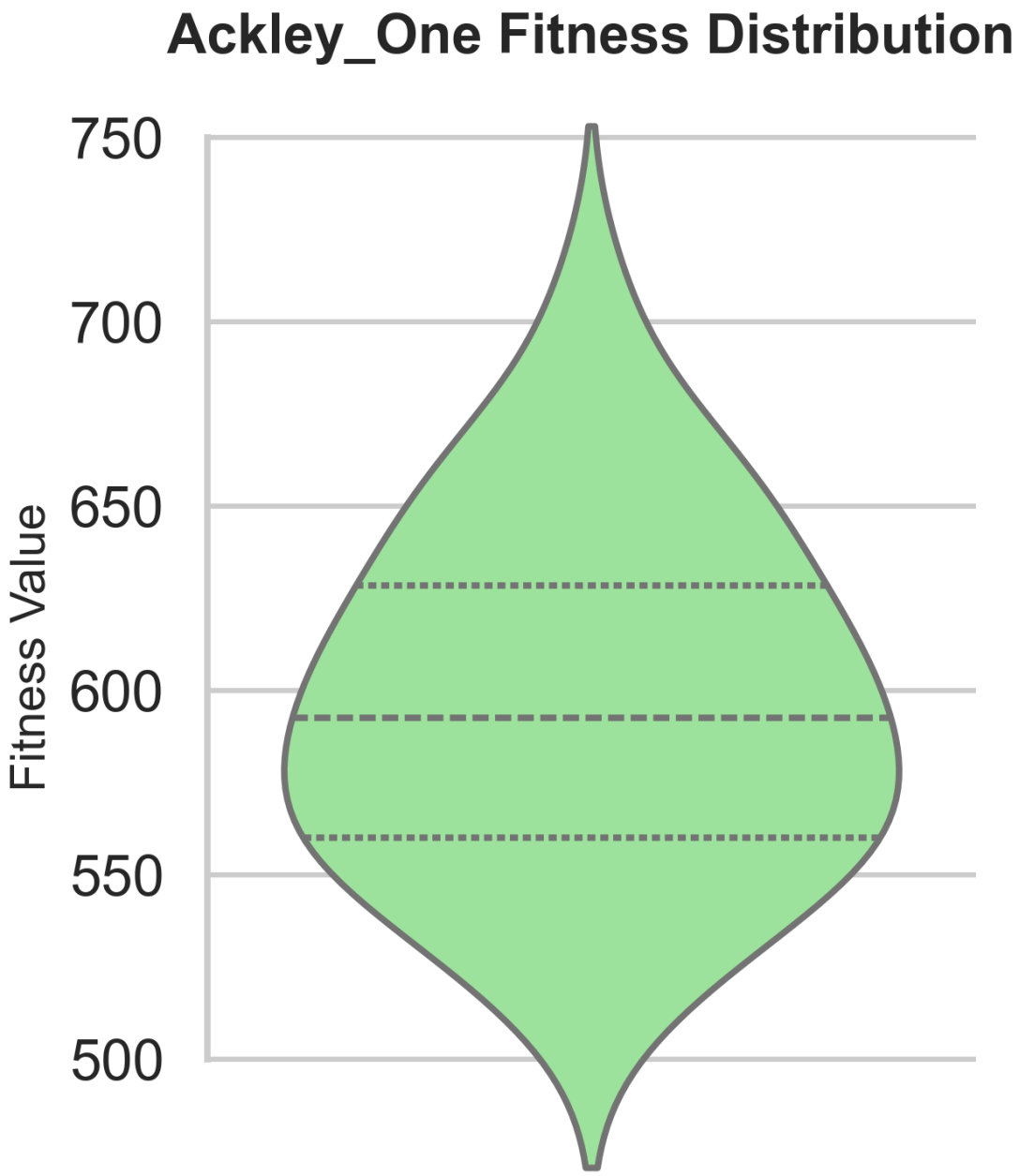

Figure 3: Ackley One Violin

3.2 Ackley's Two

$$f_9(\mathbf{x}) = \sum_{i=1}^{n-1} \left[20 + e - 20e^{-0.2\sqrt{\frac{x_i^2 + x_{i+1}^2}{2}}} - e^{0.5[\cos(2\pi x_i) + \cos(2\pi x_{i+1})]} \right]$$

fig. 4 demonstrates the notably flat distribution of fitness values, indicting the fitness values tend to vary significantly from the median. We can see that as fitness shrinks or grows from the center, frequency tends

to remain fairly stable until exceeding the lower and upper quartiles. The data is not heavily skewed, but fig. 5 shows a slight skew towards higher fitness values. For example, we see a flatter curve above the median then below. Additionally, the 3rd quartile covers a much wider range of fitness values then the 2nd quartile.

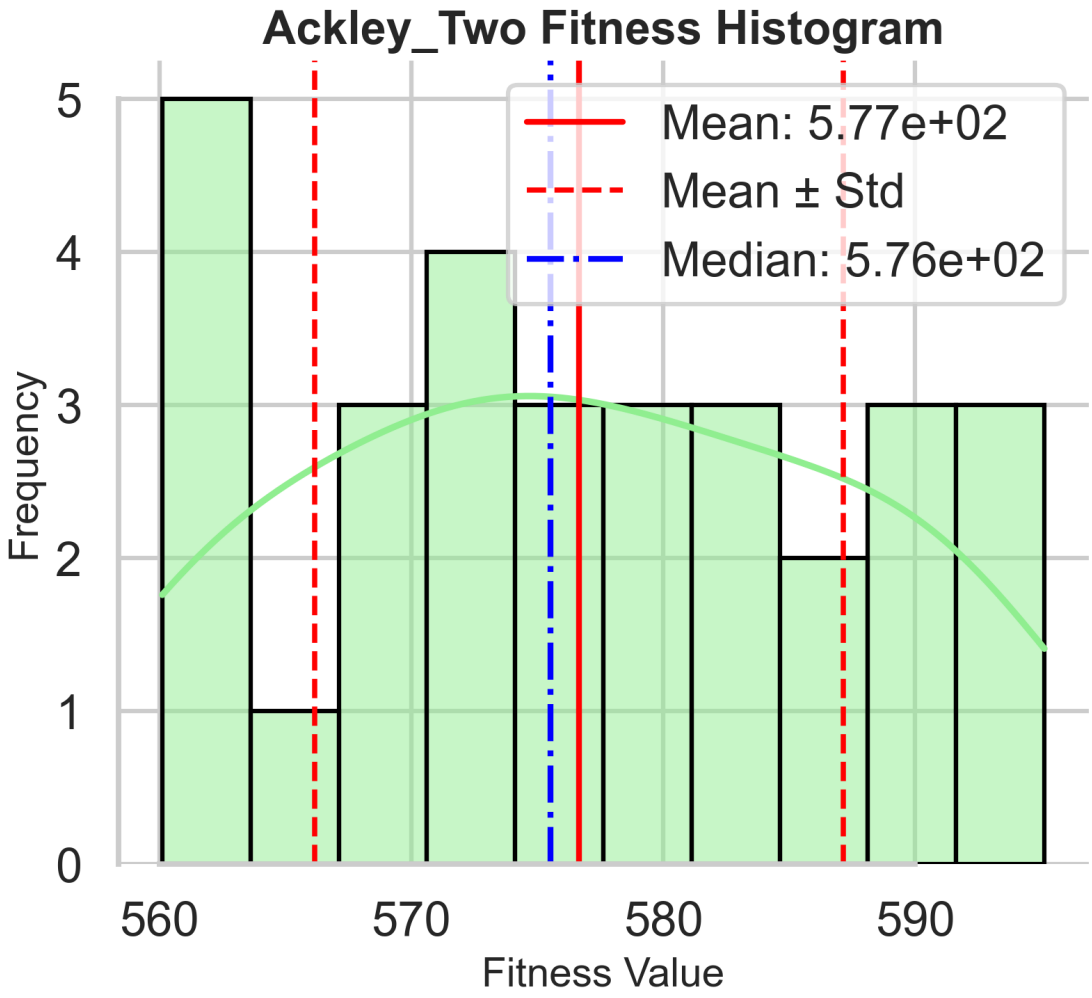

Figure 4: Ackley Two Histogram

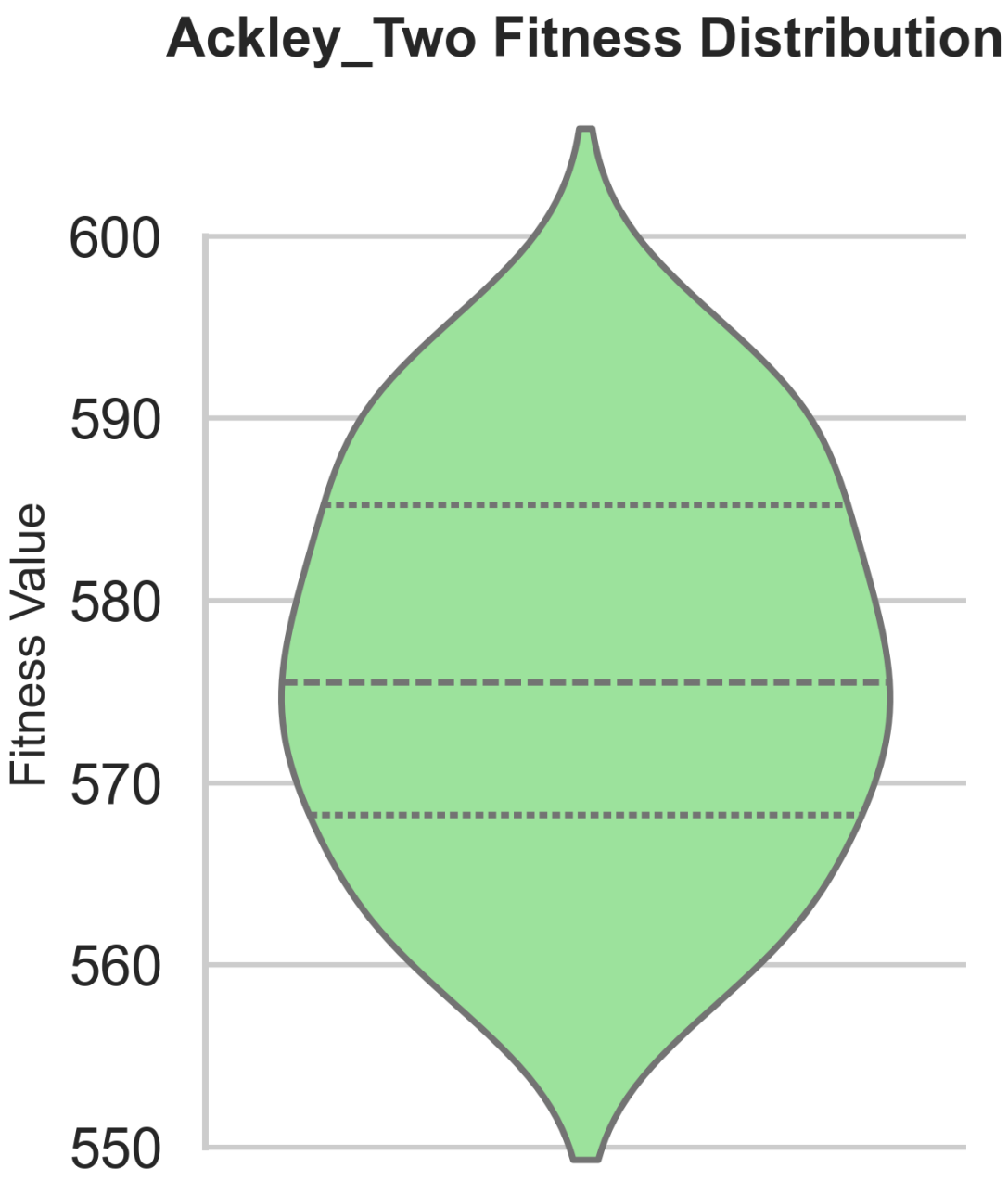

Figure 5: Ackley Two Violin

3.3 DeJong One

$$f_2(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

DeJong One provides a very simple convex function to test convergence on [2]. fig. 6 shows a strong skew to the left among fitness values. This is reinforced by the significant variation in distance from mean to the first and 3rd quartiles seen in fig. 7. The DeJong 1 function shows a fairly wide spread, having the 2nd highest

range out of any of the tested functions, as seen in table 2. Additionally, fig. 1 shows that it is the second fastest function to execute, likely due to its simplicity. Despite the function's simplicity, fitness results hav the second largest standard deviation out of any of the tested functions.

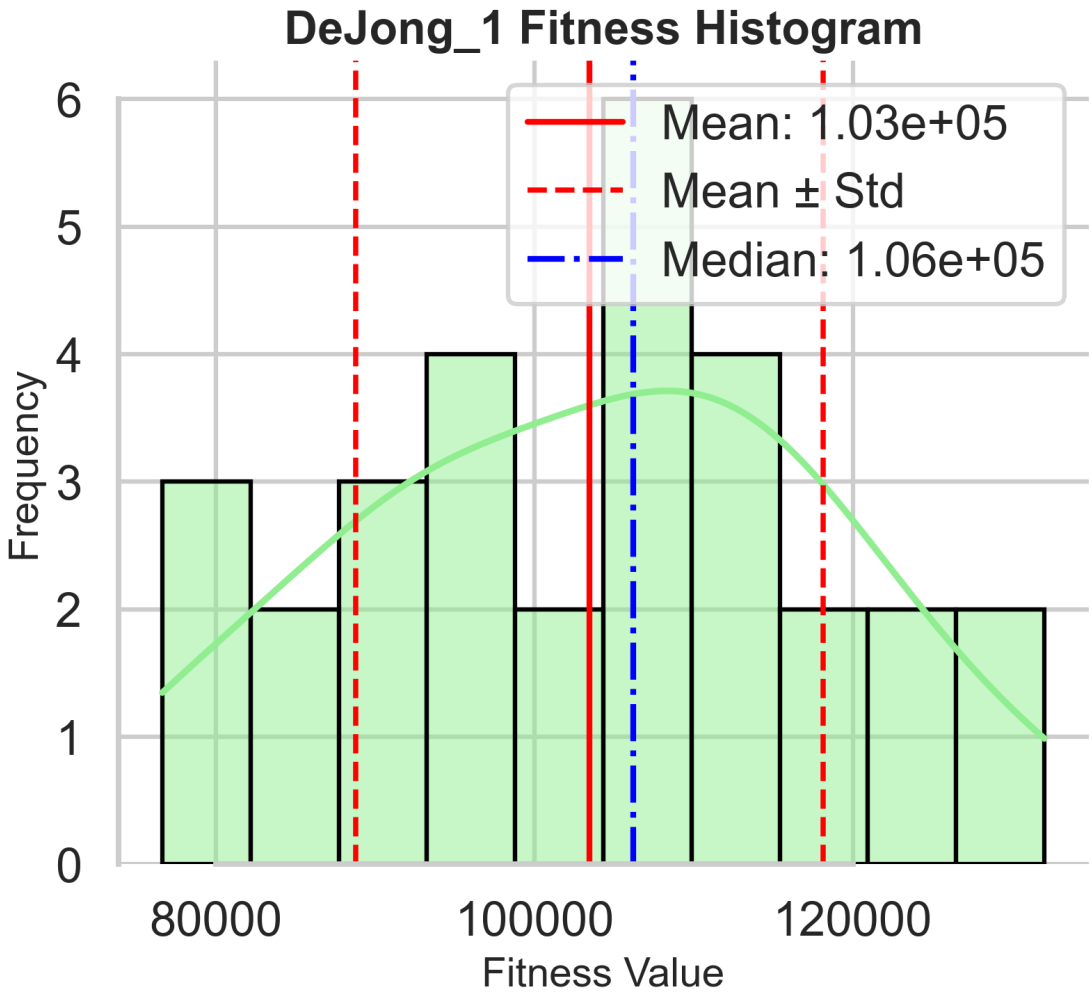

Figure 6: Dejong 1 Histogram

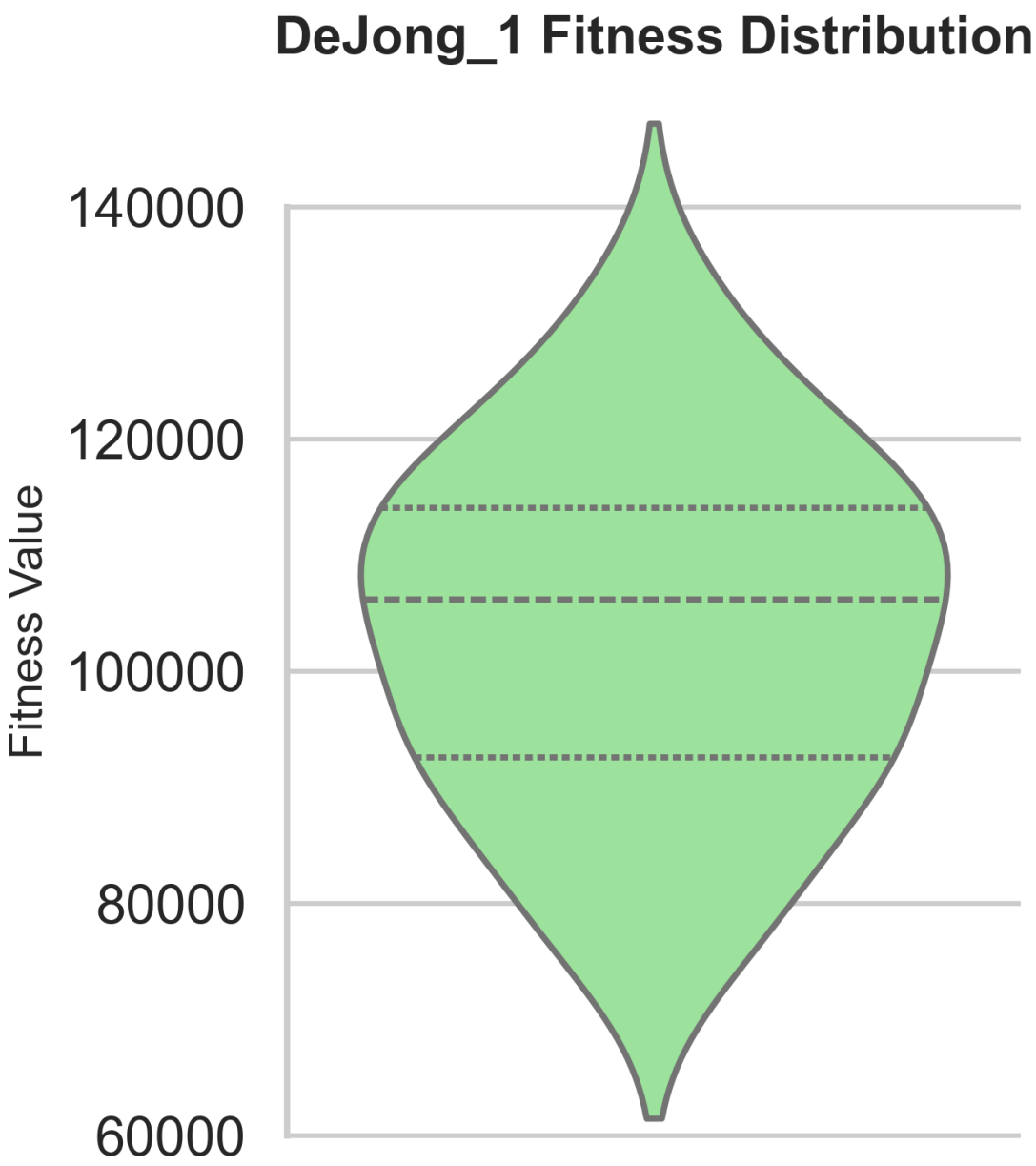

Figure 7: Dejong 1 Violin

3.4 Egg Holder

$$f_{10}(\mathbf{x}) = \sum_{i=1}^{n-1} \left[-x_i \sin\left(\sqrt{|x_i - x_{i+1} - 47|}\right) - (x_{i+1} + 47) \sin\left(\sqrt{\left|x_{i+1} + 47 + \frac{x_i}{2}\right|}\right) \right]$$

The most notable characteristic of th Egg Holder Function's fitness values is their leptokurtic nature. fig. 9 demonstrates the tight grouping fitness values have around the mean, with the lower and upper quartiles being especially centralized. Additionally, a slight tail-skew can be seen in fig. 8.

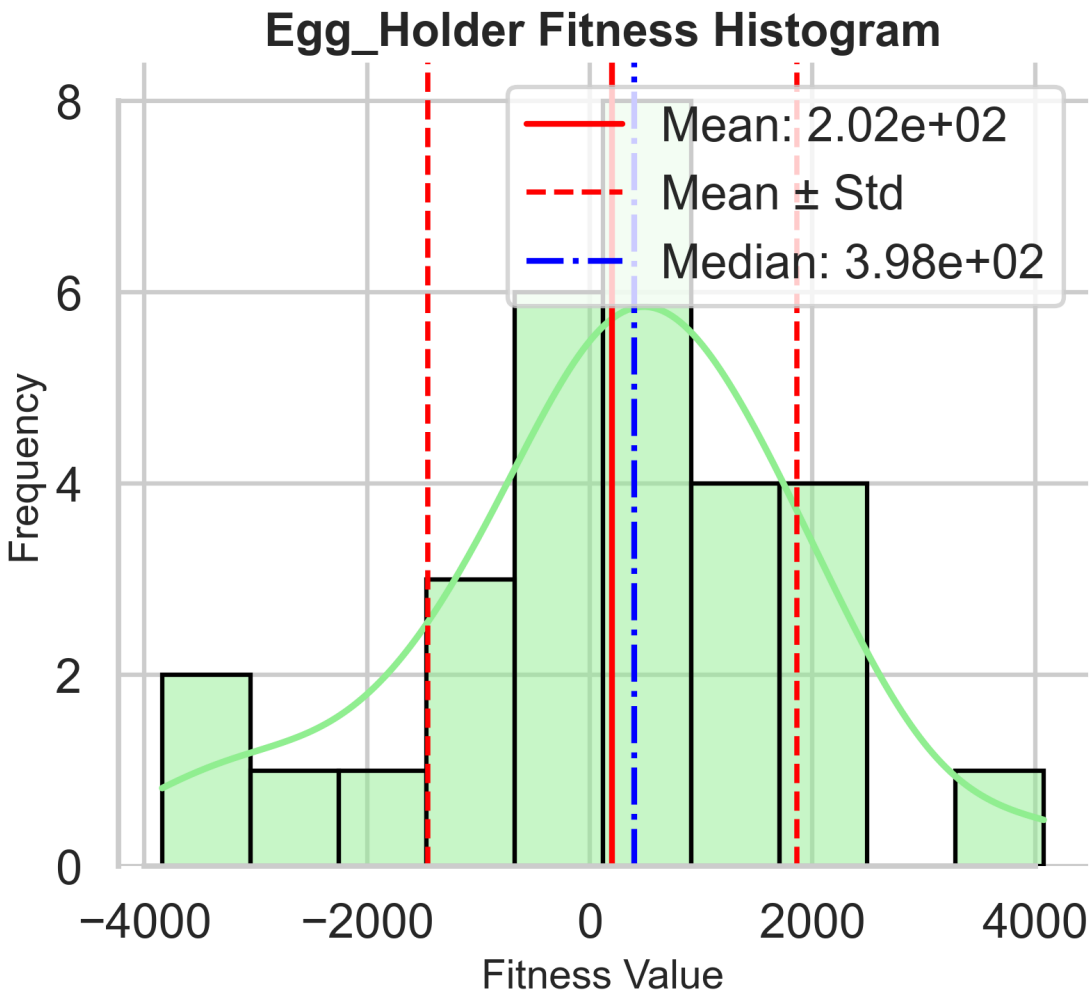

Figure 8: Egg Holder Histogram

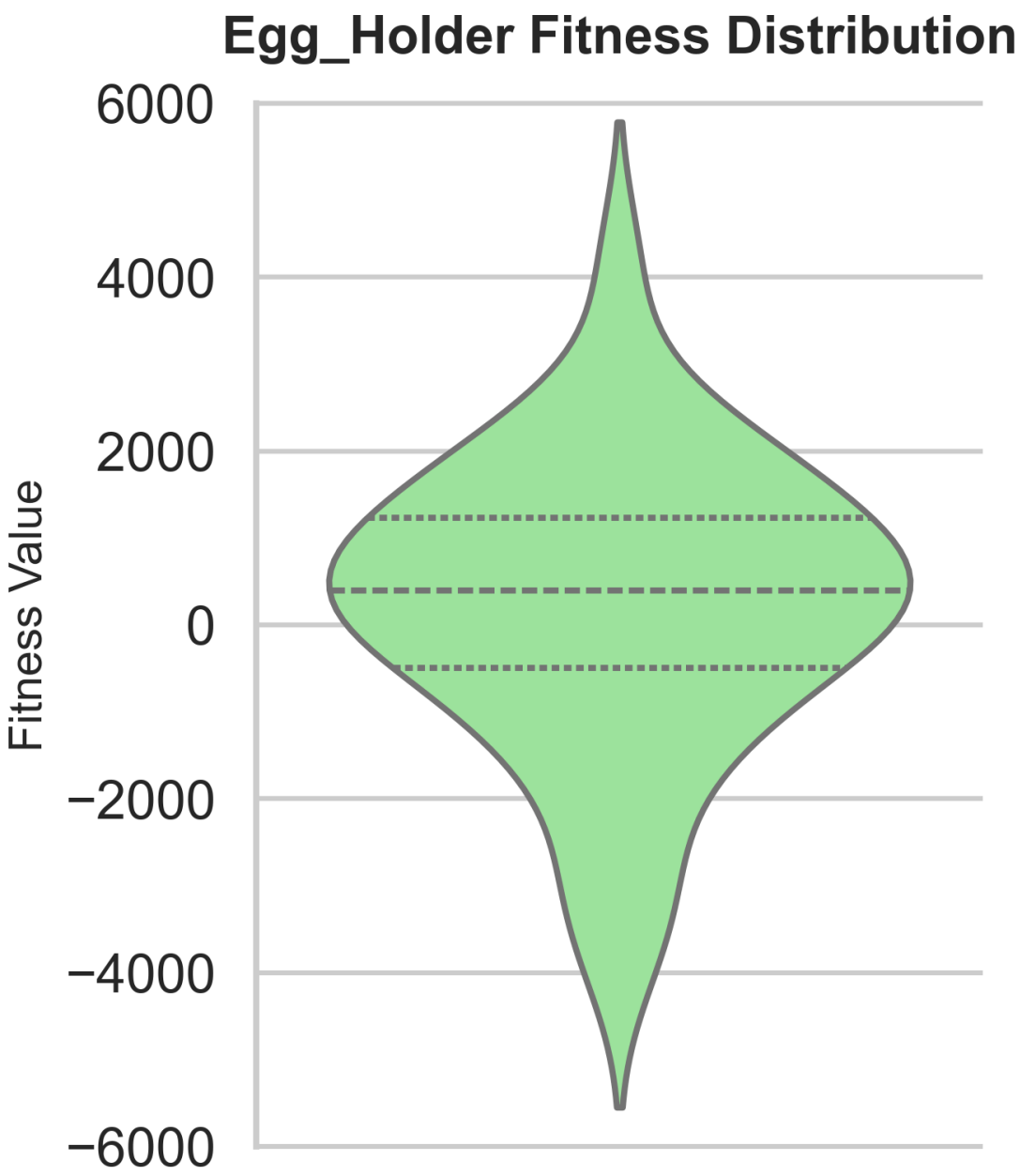

Figure 9: Egg Holder Violin

3.5 Griewangk

$$f_5(\mathbf{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

The left-skewed and high kurtosis of fitness values for the Griewangk function is demonstrated by the distribution curve seen in fig. 10. The significantly larger gap between mean and the lower quartile versus between mean and upper quartile seen in fig. 11 further display the Griewangk function's tendency to produce left-skewed fitness values. When compared to other tested standard benchmark functions, Griewangk

produces somewhat average results for standard deviation and range of fitness values, as well as execution time, as seen in table 2.

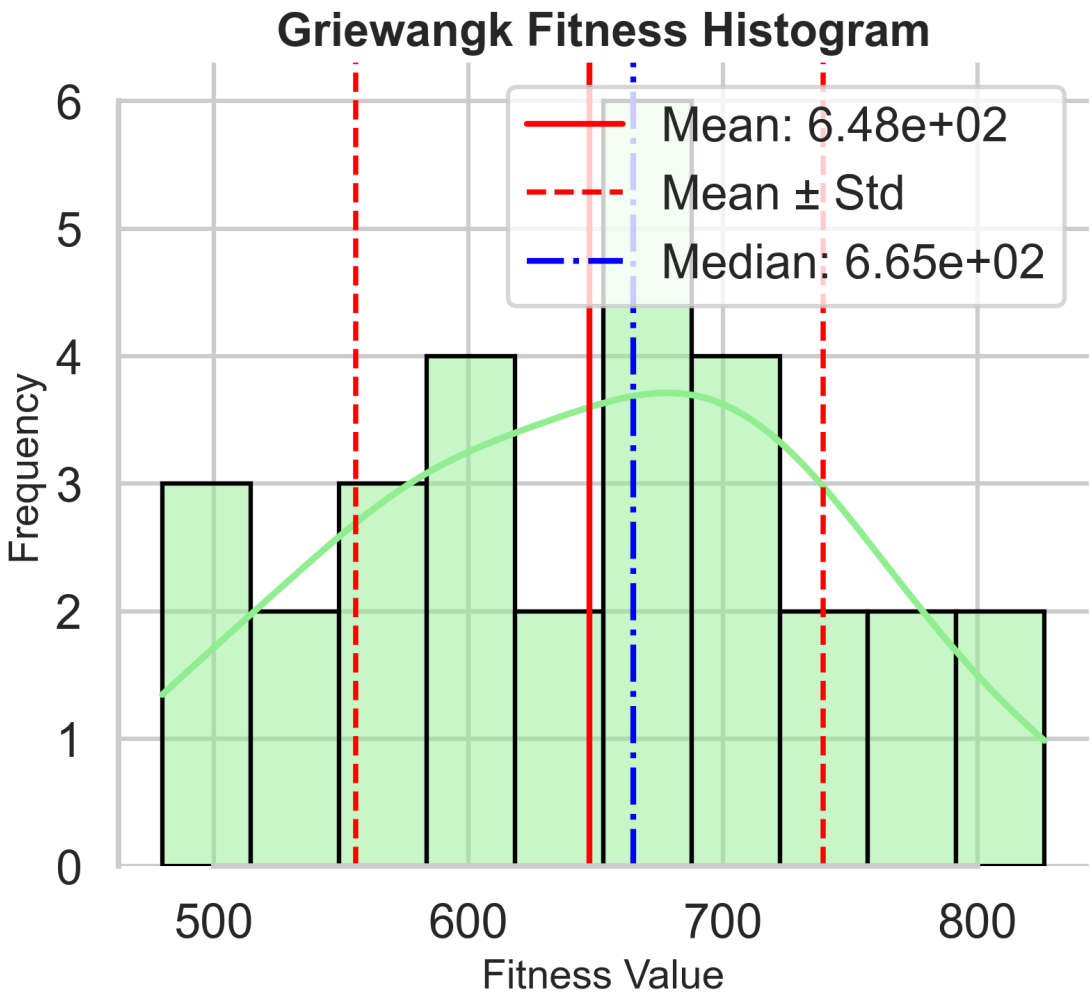

Figure 10: Griewangk Histogram

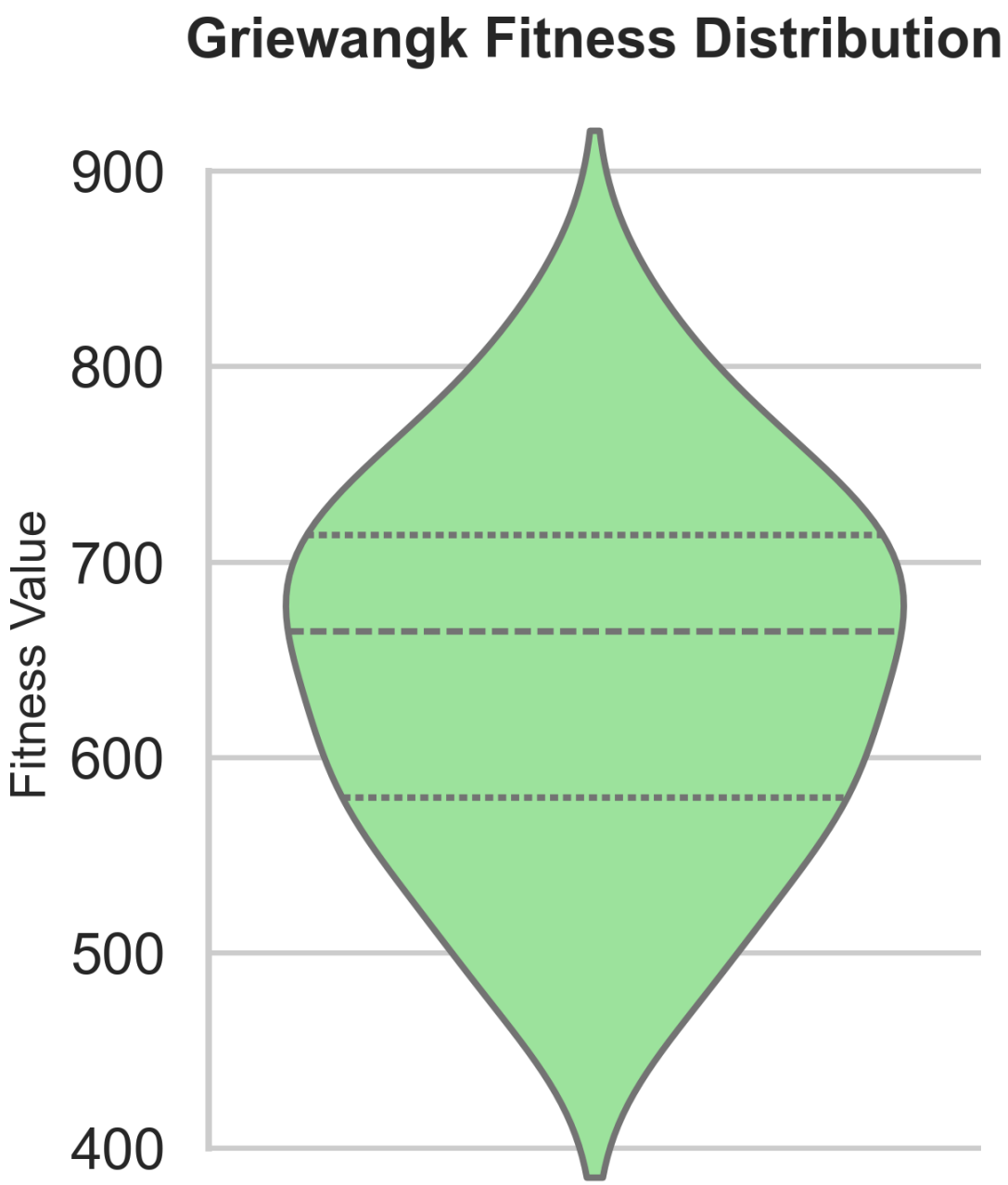

Figure 11: Griewangk Violin

3.6 Rastrigin

$$f_4(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

Similar to many of the tested standard benchmark functions, the Rastrigin function tends to produce fitness values skewed to the left. This can be seen in the flatter left side of the distribution curve in fig. 12 and the

comparatively large gap between mean and the lower-quartile in fig. 13. Additionally, fig. 13 demonstrates the significantly higher occurrences of fitness values near the lower end of produced fitness range compared to the higher end. This is demonstrated by the more pointed top of the violin plot compared to the more rounded bottom.

Compared to other tested standard benchmark functions, the Rastrigin function experienced fast execution. fig. 1 shows only two functions executing more quickly. However, table 2 shows that the gap between the fastest executing functions and Rastrigin is large, with DeJong 1 and Rosenbrock's saddle executing in less than 0.005 ms, while Rastrigin took approximately 0.02 ms to executes.

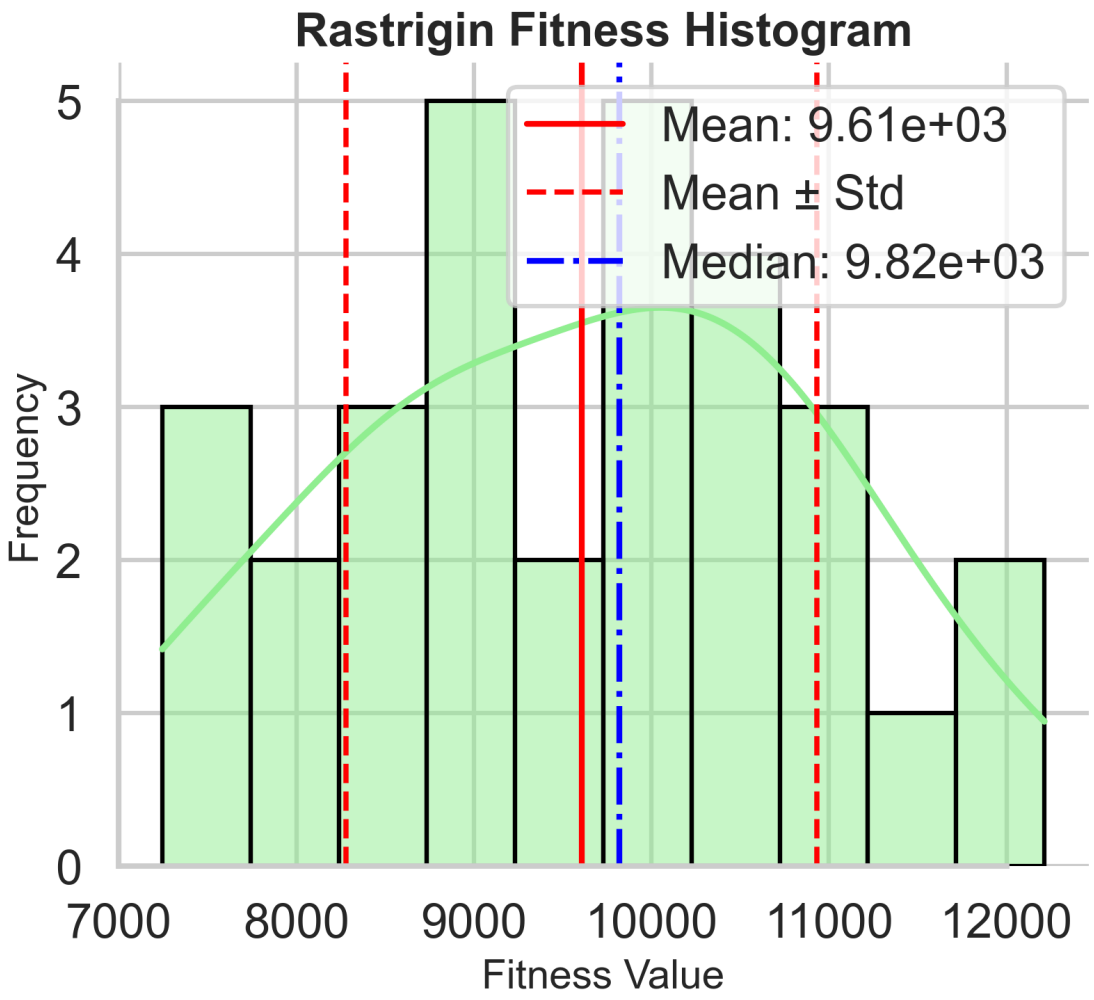

Figure 12: Rastrigin Histogram

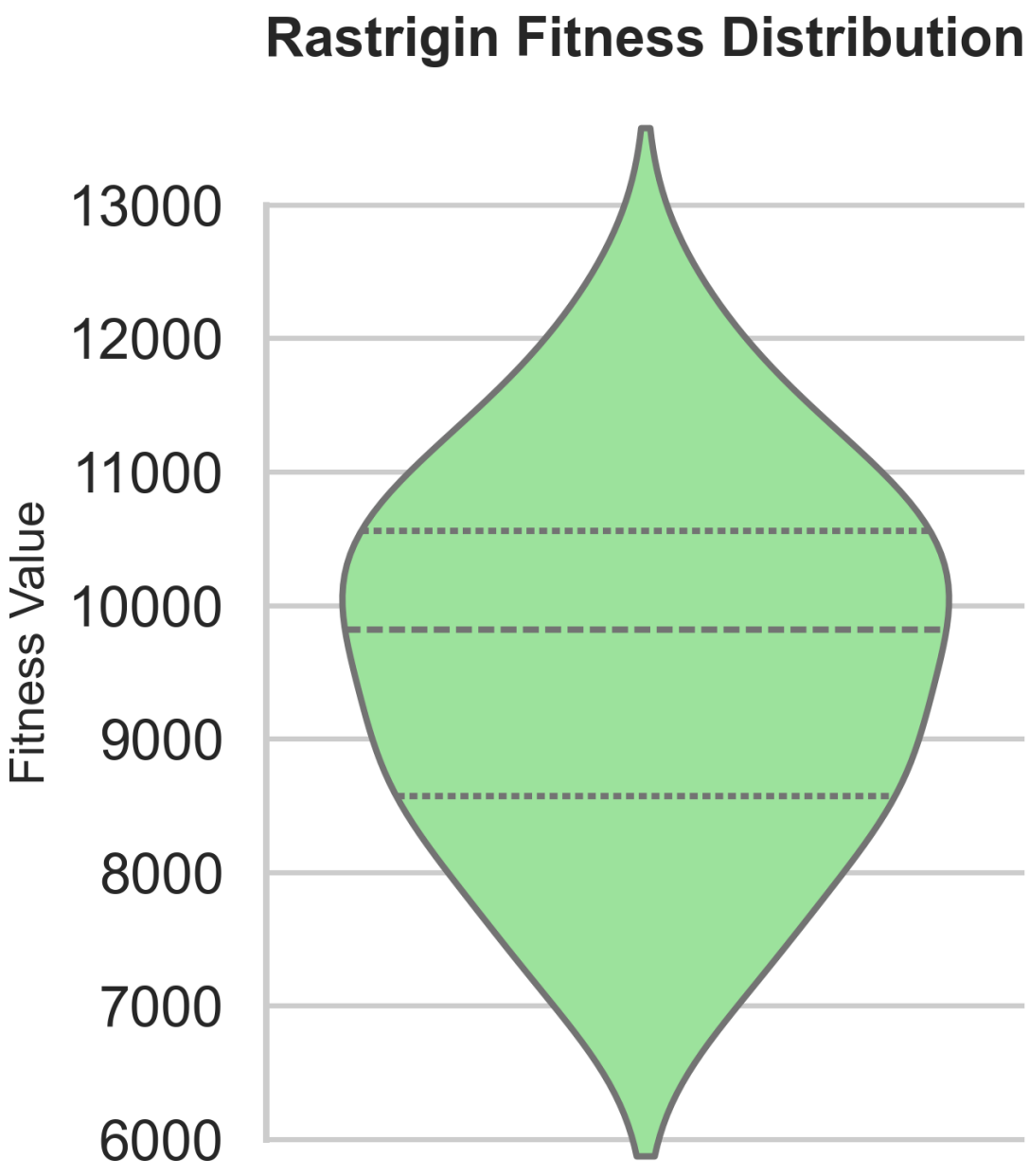

Figure 13: Rastrigin Violin

3.7 Rosenbrock's Saddle

$$f_3(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100\left(x_i^2 - x_{i+1}\right)^2 + (1 - x_i)^2\right]$$

Rosenbrock's Saddle is useful for testing algorithms due to its curved valley and local minima structure [3]. Fitness values produced by Rosenbrock's Saddle demonstrate the aptness of the function name, with the distribution curve seen in fig. 14 showing a clear saddle-like distribution shape. Frequency immediately begins to decrease as fitness decreases from the distributions peak, until about $5e^{10}$, where frequency stabilizes.

fig. 15 shows the secondary distribution peak, or 'saddle', exists near the lower quartile. This function behavior causes existence of local minima and maximums, making this function a very good choice for testing how well convergence or optimization algorithms handle local minima and maximums.

fig. 1 shows Rosenbrock's saddle as the second fastest function to execute, slightly trailing DeJong 1. This is likely due to the relative simplicity of the Rosenbrock's Saddle equation. While not as straightforward as DeJong 1, Rosenbrock's Saddle is still free of trigonometry functions, square roots, and floating-point divisions. Every mathematical operation in this function can be programmed as addition or multiplication, allowing for rapid calculation, as these operations have significantly less overhead than something like $\cos x$ or $\sqrt{x}$.

In addition to rapid execution, Rosenbrock's Saddle produces a wider range of fitness values than other tested functions. table 2 shows the range of Rosenbrock's saddle as 6 orders of magnitude greater than the range of any other tested function. The standard deviation of fitness values is similarly 6 orders of magnitude greater than any other tested function's standard deviation. Additionally, the resulting fitness values are consistently larger than those of other tested functions. Even with the huge range compared to other functions, The minimum fitness value observed from Rosenbrock's Saddle experiment is 5 orders of magnitude greater than the maximum fitness value observed from any other function.

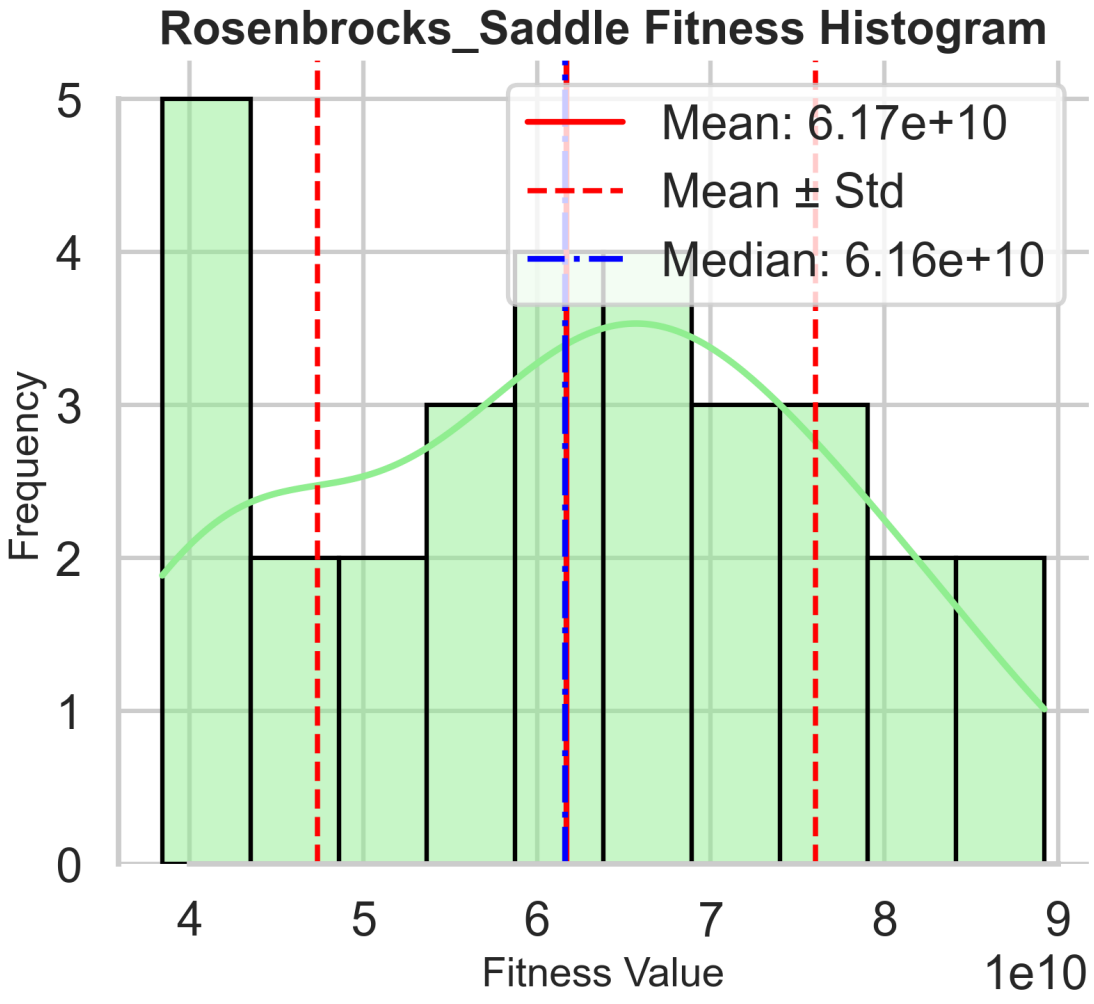

Figure 14: Rosenbrocks Saddle Histogram

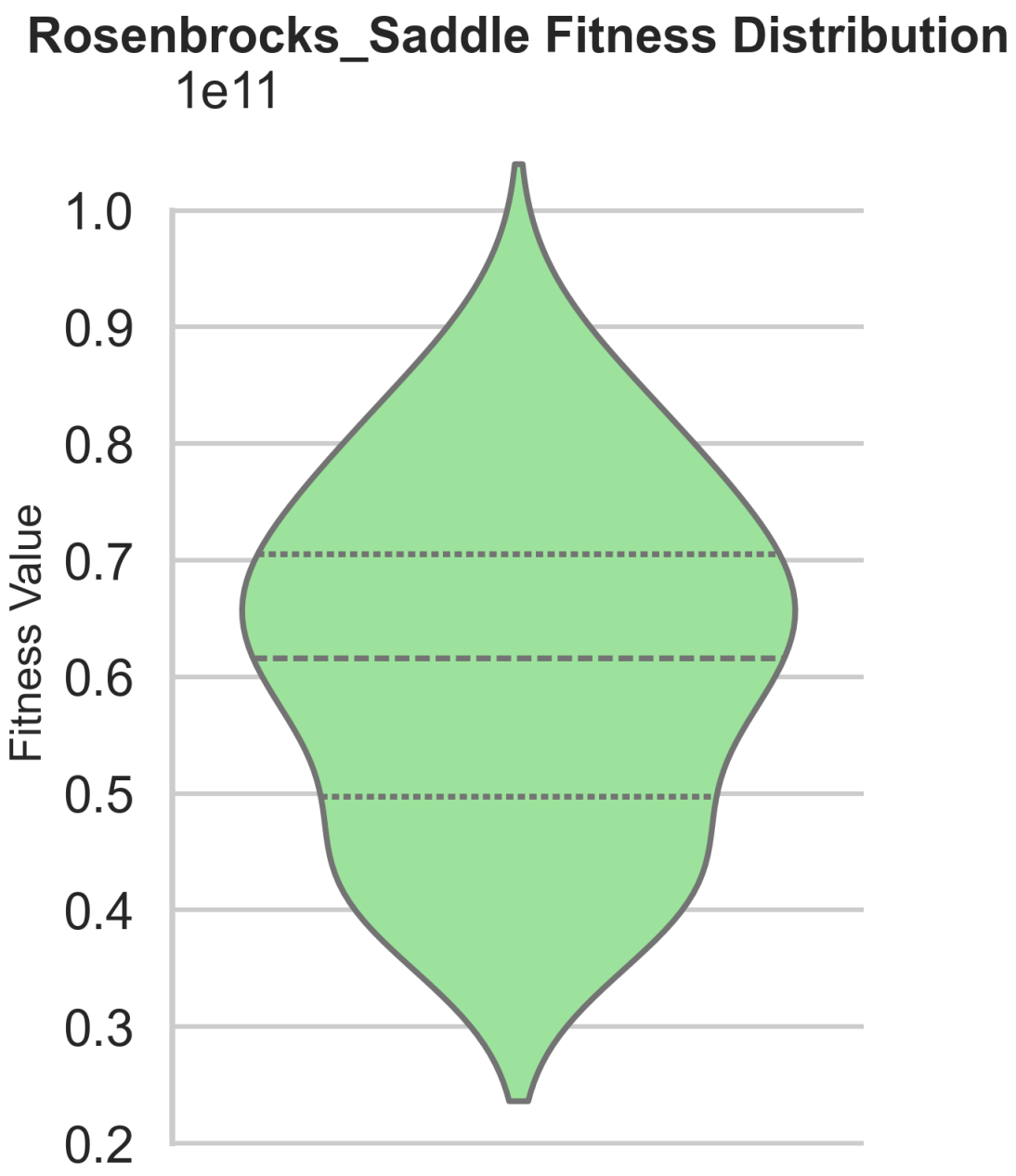

Figure 15: Rosenbrocks Saddle Violin

3.8 Schwefel

$$f_1(\mathbf{x}) = 418.9829\,n - \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right)$$

The Schwefel function has a complex nature, containing many local minima [4]. This makes it very useful in analyzing how well an optimization algorithm handles local minima when searching for the global minima. Fitness values observed from the Schwefel function appear to have fairly low kurtosis. fig. 16 shows frequency quickly dropping as fitness values stray from the median. This drop off is especially noticeable on the lower

end of fitness values. However, once sufficiently far from the median fitness value in either direction, frequency experiences a local maxima. The intensity of this maxima is greater on the higher end. While the frequency spike in the lower end is not as significant, it is notably further from the median than on the higher end.

fig. 17 further demonstrates the unique distribution of fitness values observed in this function, showing that distribution drops of fairly steadily when fitness values are greater than the median, with a spike just outside of the upper quartile. As fitness value decreases, frequency drops rapidly. That is, however, until far outside the lower-quartile, where frequency spikes. The elongated tail seen in fig. 17 demonstrates the unusual nature of this lower-quartile behavior.

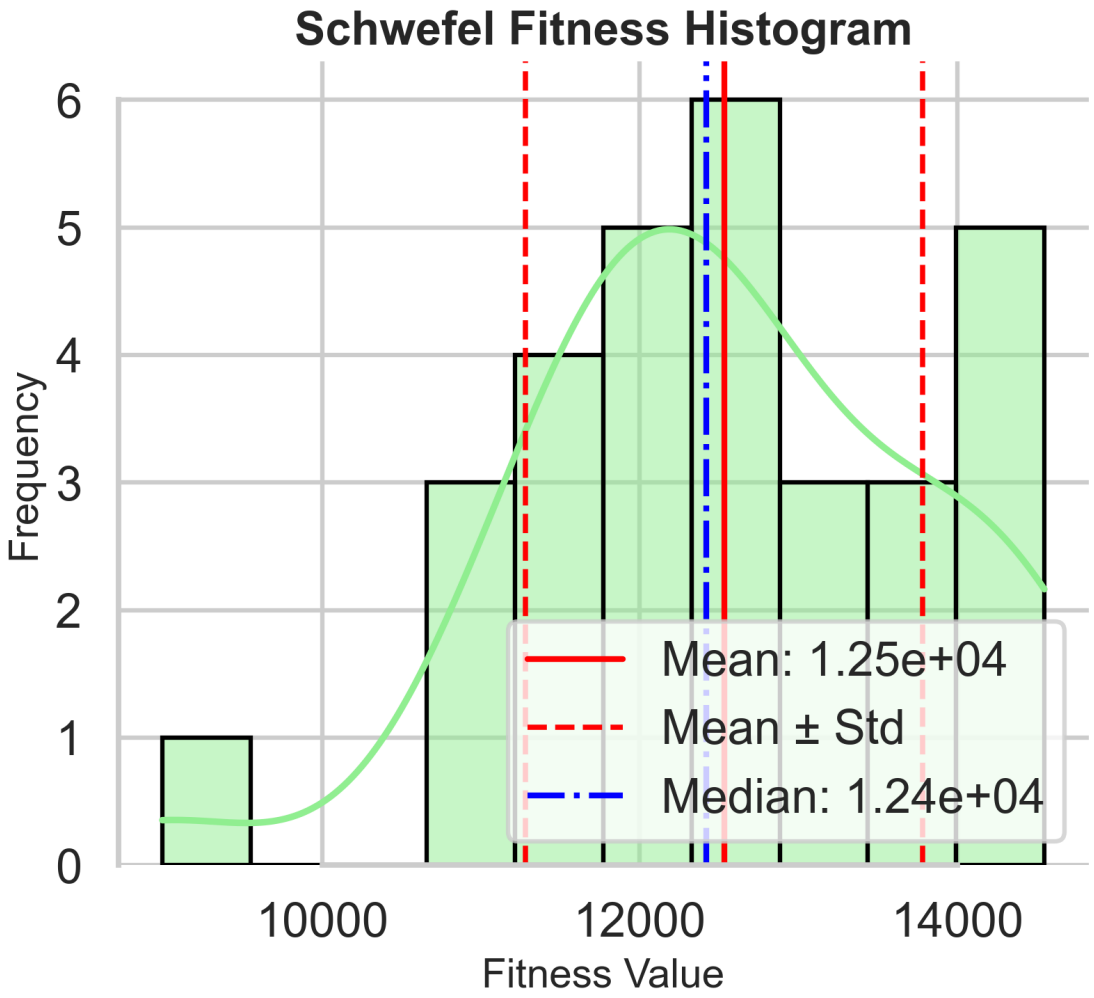

Figure 16: Schwefel Histogram

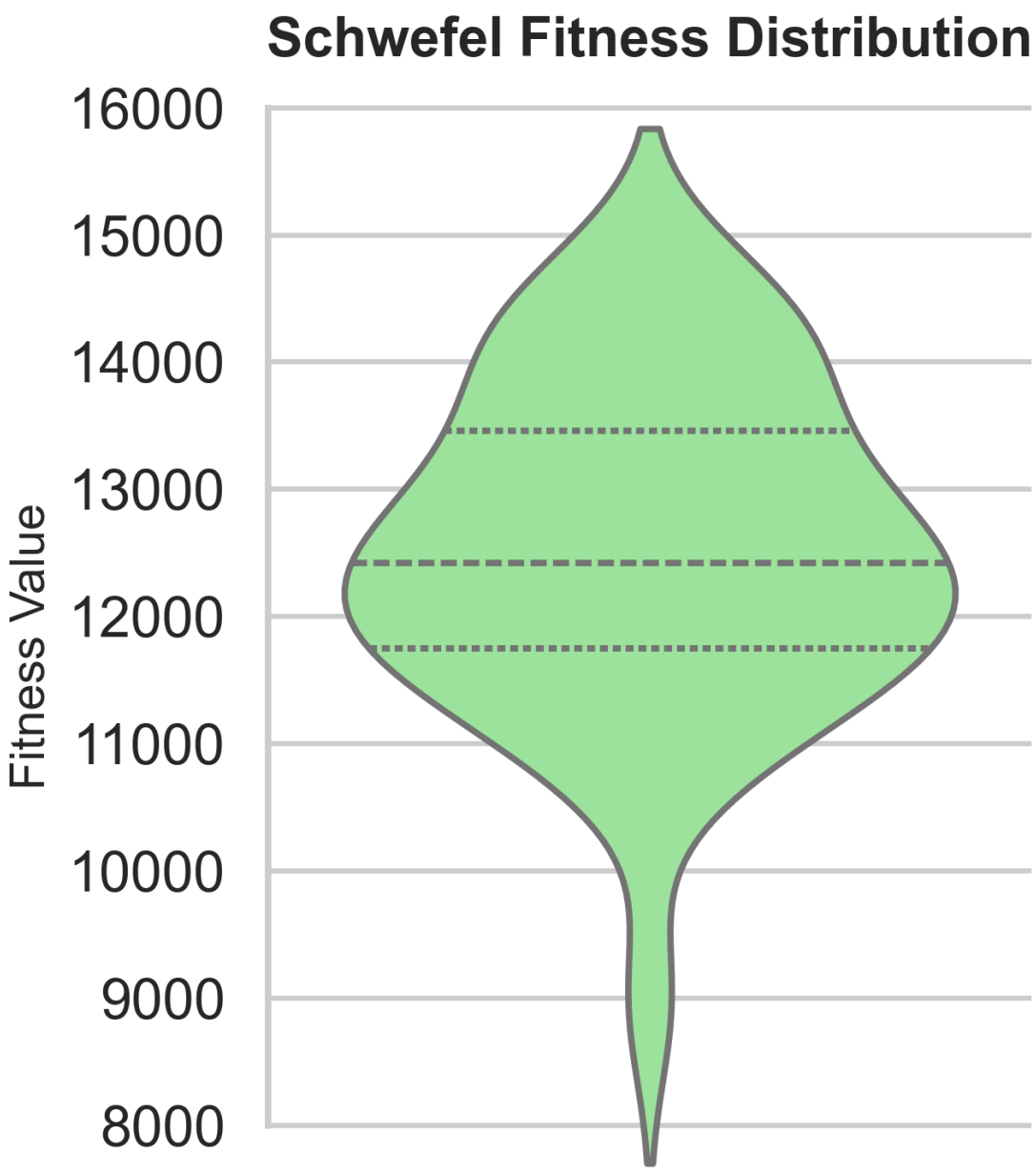

Figure 17: Schwefel Violin

3.9 Sine Envelope Sine Wave

$$f_6(\mathbf{x}) = -\sum_{i=1}^{n-1} 0.5 + \frac{\sin^2\left(x_i^2 + x_{i+1}^2 - 0.5\right)}{\left[1 + 0.001\left(x_i^2 + x_{i+1}^2\right)\right]^2}$$

The Sine Envelope Sine Wave function produced tightly-grouped fitness values compared to the other tested functions producing the smallest range of fitness values. For example, table 2 shows the range of the Ackley's Two function, which has the second smallest range, as being over 7 times larger than Sine Envelope Sine

Wave's range. This function also produced the smallest standard deviation of fitness values out of all tested functions, with the small range in fitness values likely playing a role in this. Additionally, this is the only tested benchmark function to produce only negative fitness values.

fig. 18 shows the tail-skewed nature of fitness values, with frequency dropping off faster when fitness is greater than the median. The smooth and gradual decrease of frequency as fitness decreases from mean can be seen in fig. 19. However, as fitness increases from the mean, we observe a short and rapid drop in frequency until well past the upper quartile, where frequency then dwindles more casually. This is further reinforced by the very small step from the mean to the upper quartile compared to the mean to the lower quartile.

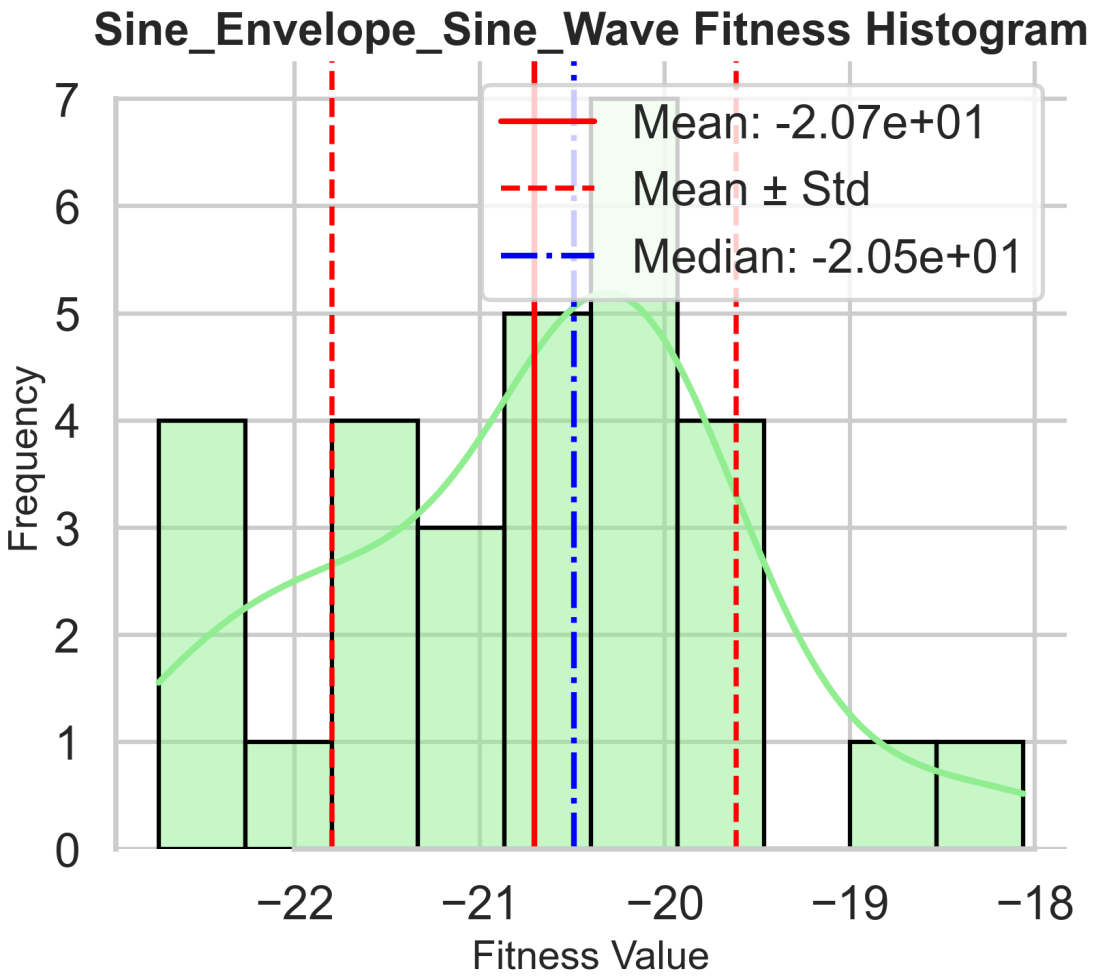

Figure 18: Sine Envelope Sine Wave Histogram

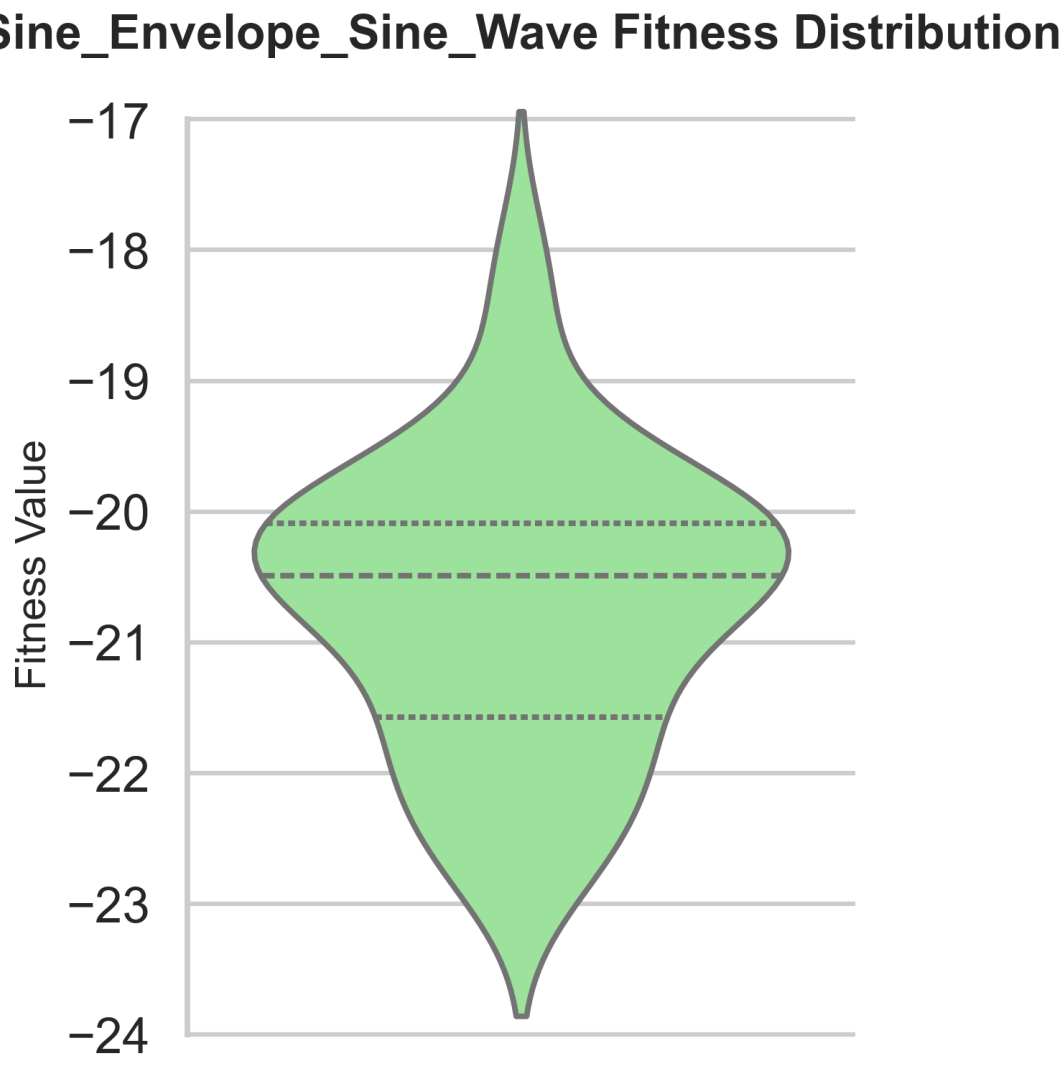

Figure 19: Sine Envelope Sine Wave Violin

3.10 Stretched V Sine Wave

$$f_7(\mathbf{x}) = \sum_{i=1}^{n-1} \left[\left(x_i^2 + x_{i+1}^2\right)^{1/4} \sin^2\left(50\left(x_i^2 + x_{i+1}^2\right)^{1/10}\right) + 1 \right]$$

One notable characteristic of fitness values produced by the Sttretched V Sine Wave function is fairly high kurtosis, as seen in the flattened distribution curve in fig. 20. As fitness values decrease from the median, frequency decreases slowly. Alternately, as fitness increases, frequency experiences a slower initial drop off, reducing more drastically after the upper quartile. This is demonstrated by the concave shape at the top of the violin plot compared to the convex shape at the bottom, as seen in fig. 21. This further indicates the delayed but then more rapid drop off of frequency as fitness increases versus decreases.

table 2 Demonstrates the relatively small range of fitness values produced by the Stretched V Sine Wave function. Additionally, this function produced fitness values with the second lowest standard deviation out of any tested function. Additionally, fig. 1 shows this function's slow evaluation. Execution time for the

Stretched V Sine Wave function is notably slower than any of the other tested functions. This is likely due to the multiple higher-index radicals that must be calculated for each summation.

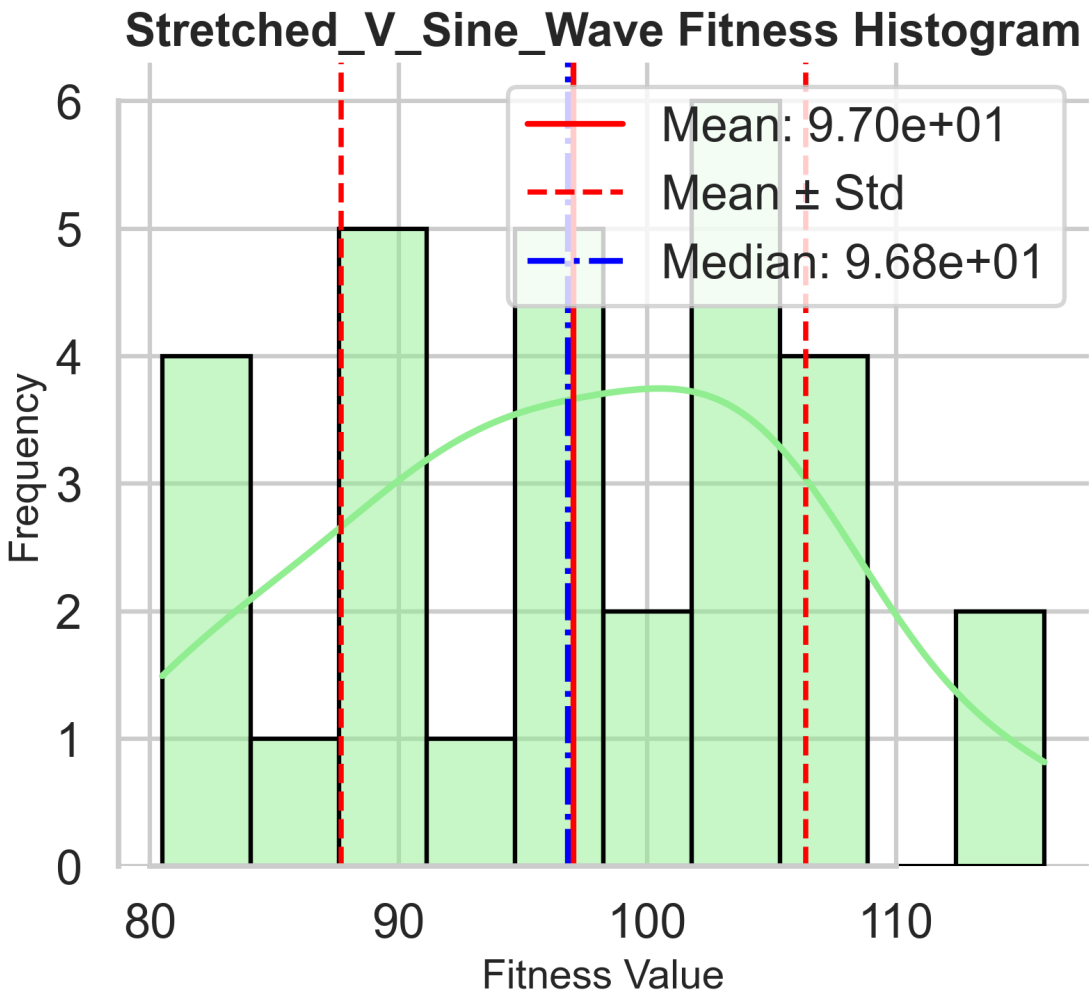

Figure 20: Stretched V Sine Wave Histogram

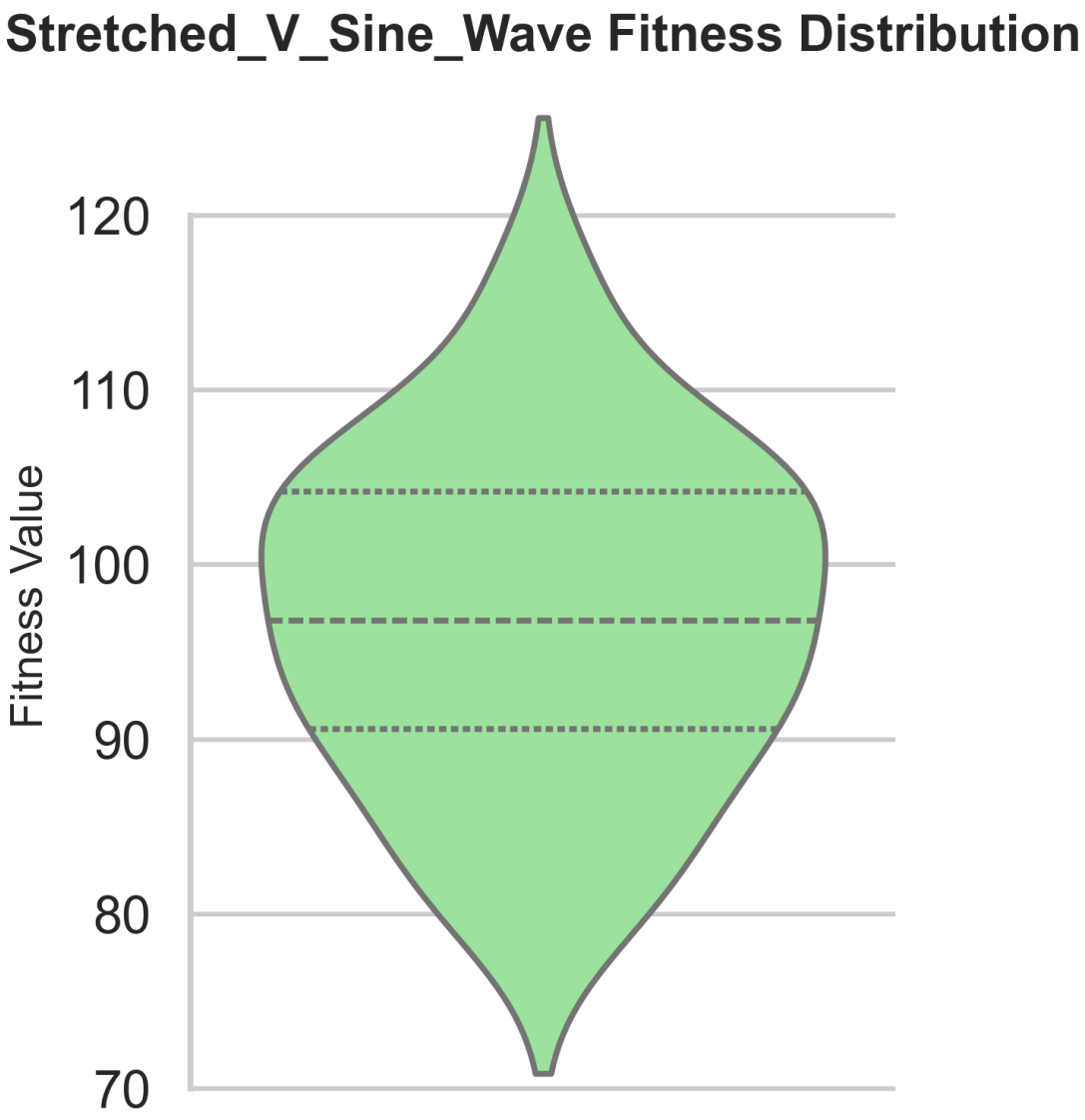

Figure 21: Stretched V Sine Wave Violin

References

[1] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, 1987.

[2] Marcin Molga and Czeslaw Smutnicki. Test functions for optimization needs. `https://robertmarks.org/Classes/ENGR5358/Papers/functions.pdf`, 2005. Section 2.1 (De Jong's function). Accessed: 2026-01-26.

[3] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3), 1960.

[4] Sonja Surjanovic and Derek Bingham. Schwefel function. `https://www.sfu.ca/~ssurjano/schwef.html`, 2026. Accessed: 2026-01-26.