Group 01

EVOLUTION OF SIMULATED ANNEALING AND ITS APPLICATIONS

Content

Our Members

- S/18/806-Danushiyan
- S/18/809-Sathushana
- S/18/824-Thivajini
- S/18/840-Dinuka
- S/18/844-Shankara
- S/18/846-Janith
- S/18/SP/851-Kareendra

imperdiet. Fusce eu vestibulum augue, ac iaculis quam.

INTRODUCTION

What is simulated annealing and its application

Simulated annealing is a method for solving unconstrained and bound-constrained optimization problems. The method models the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy.

Applications

- Steel manufacturing
- Metal working
- Jewelry making
- Cooling system in car
- Production of semi-conductor
- Production of glass and ceramics

Boltzmann Distribution

$$f(c) = 4\pi c^2 * \left(\frac{m}{2\pi KT}\right)^{\frac{3}{2}} * \exp\left(\frac{-mc^2}{2KT}\right)$$

F(c) = The Probability density function of the speed c particles

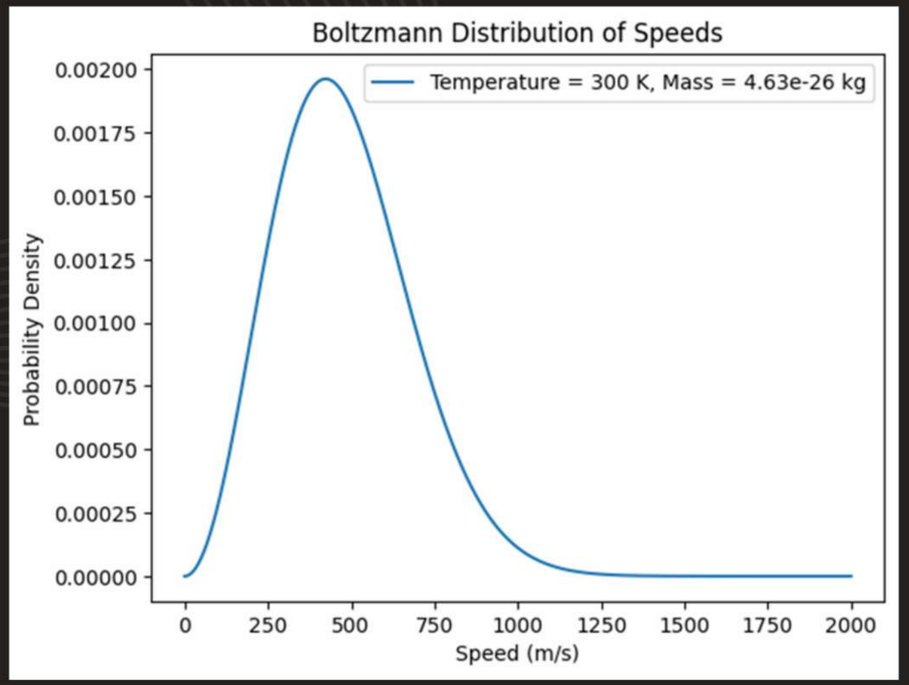
M = mass of the molecule

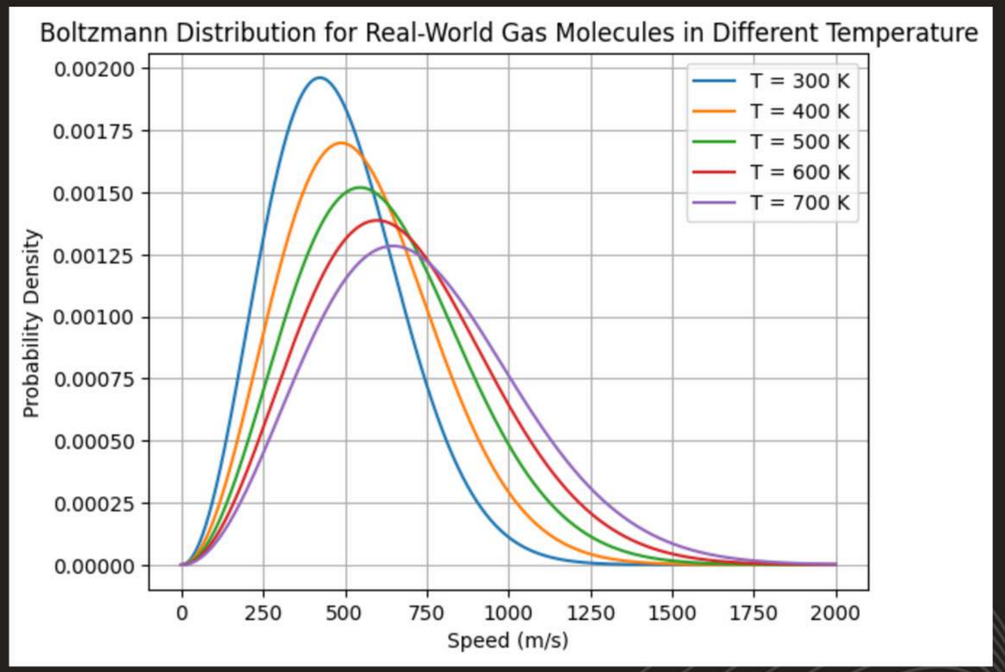
T = Temperature

K = Boltzmann constant

C = Speed of the molecule

Boltzmann distribution is a probability function used in statistical physics to characterize state of a system of particles, with respect to temperature and energy. The system can exist in several states, however, the chance of being in certain subset of states is higher than other.





At higher temperatures, the curve becomes broader and flatter, indicating that more molecules travel at very high speeds. From this plot, it's evident that at higher temperatures, there is a higher probability of achieving higher speeds, while at lower temperatures, there is a higher probability of lower speeds.

Pseudo Code For Cooling Process

Repeat

Perturbation : State i to state j

Calculate the energy change (Δ E)

Δ E = E (State j) – E (State i)

1. If Δ E < = 0
Then accept the new state as state j

2. Else
Then new state is accepted by Boltzmann factor using metropolis criterion i.e exp (-Δ E ij/T) > random [0,1) accept

3. If accept ,update the state i with state j
Until equilibrium is approached

Dry Run

Using Function:

$$y = x^2$$

Energy is analog to $y = f(x) = x^2; -1 \leq x \leq 1$

States (position of particles) are analog to $x - 1 \leq x \leq 1$

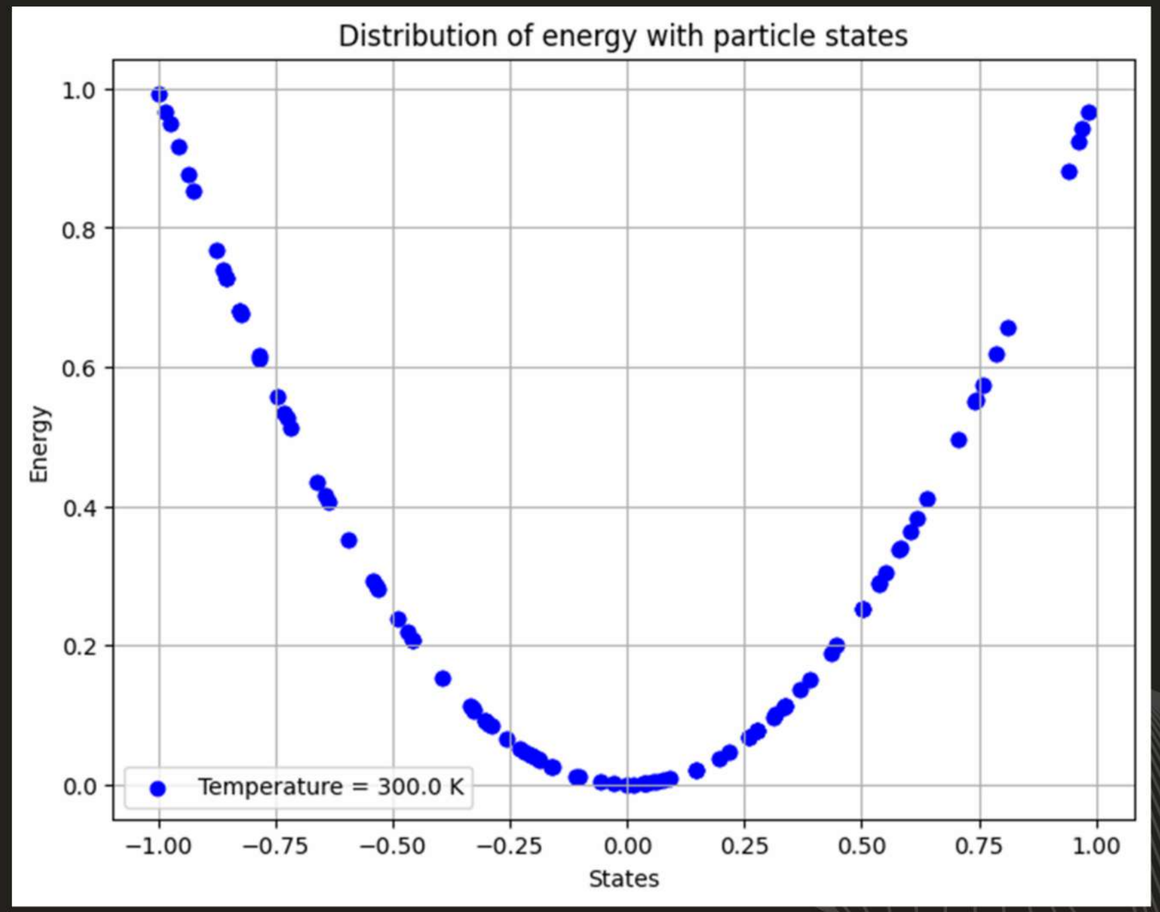
Starting point of iteration:

-0.8952728022981113

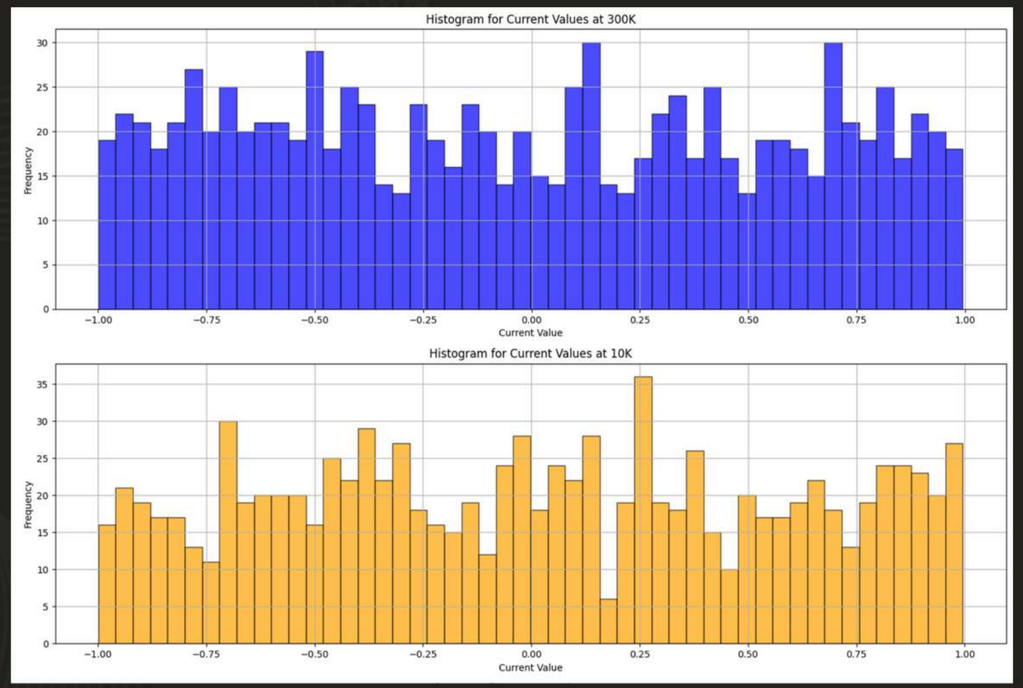
Minimum Value of current r values :

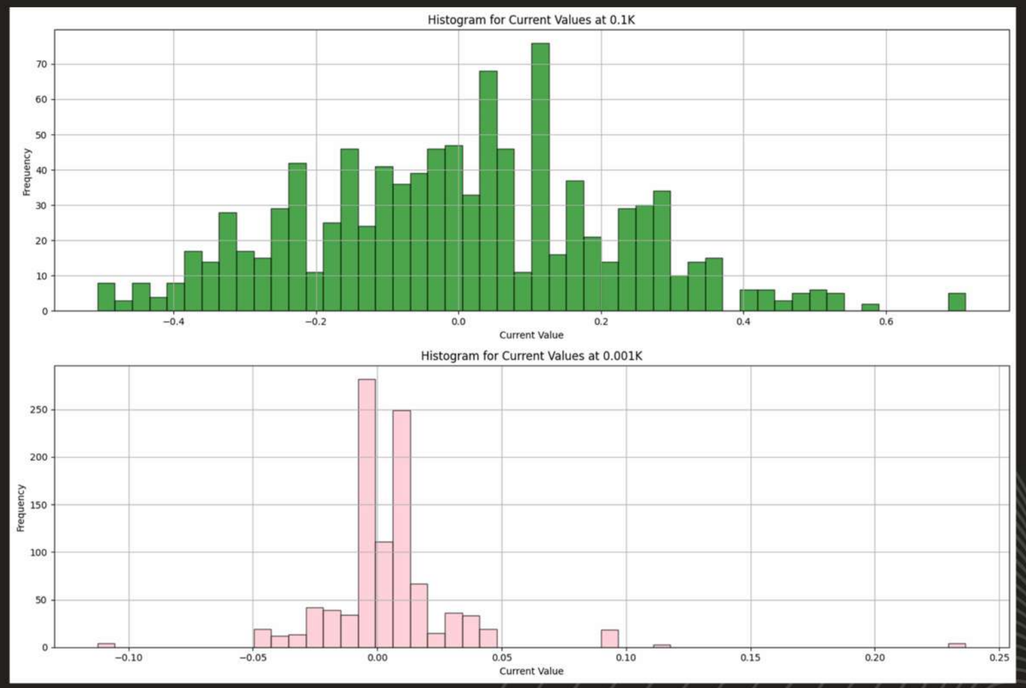
-0.9966588928415543

Distribution plot of Energy of particles with their states



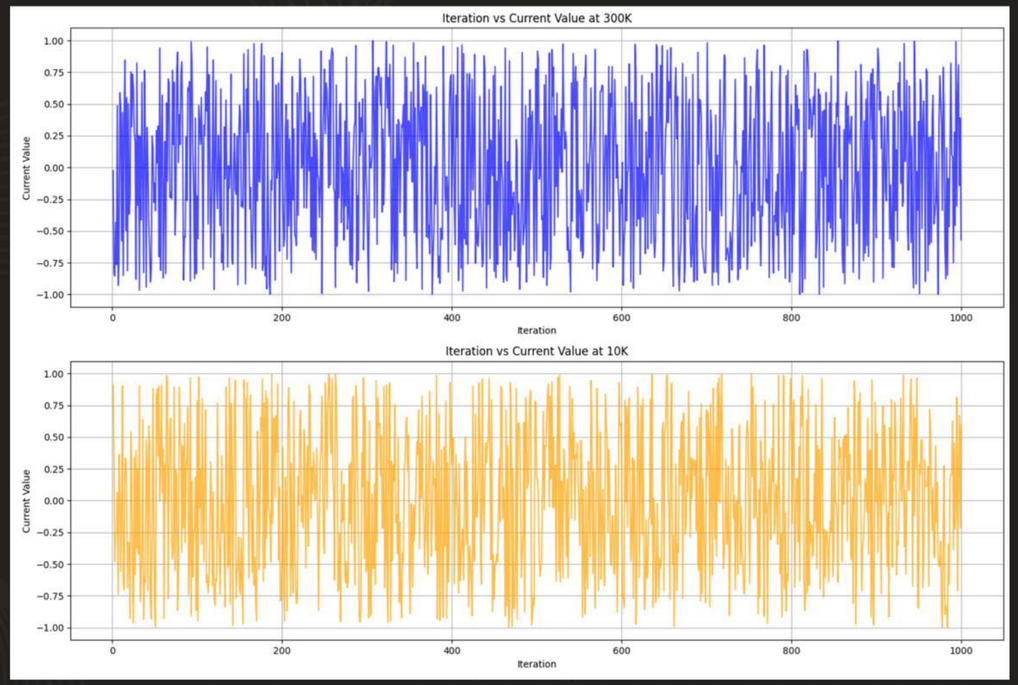
Distribution of x values change at Different Temperatures.

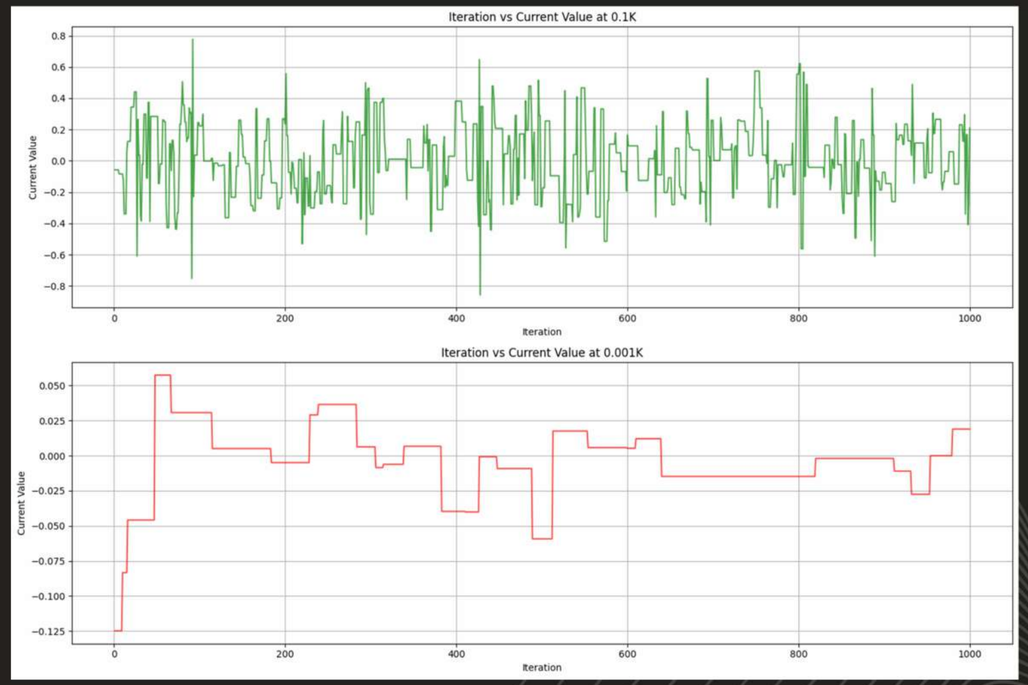




These graphs shows how the metropolis criterion changes when decreasing the temperature. In low temperatures the histogram represents the boltzmann distribution

X chain Changes at Different Temperatures.





In low temperatures the the function value starts to converge into the optimal solution(in this case, 0).

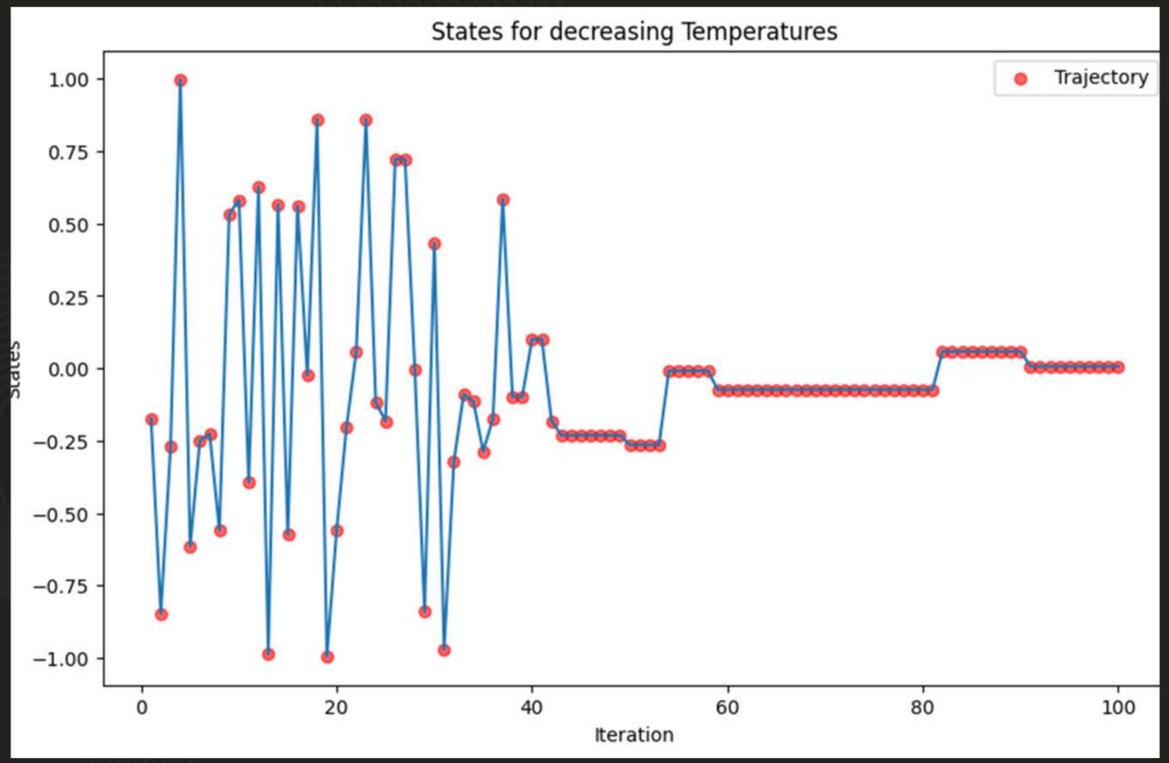
Simulated Annealing for Decreasing Temperatures

Add Temperature as a control parameter for the Simulated annealing Algorithm for the optimization of,

$$y = x^2$$

- Initially set a high value.
- Gradually decrease temperature by 20% at each iteration.

Plot of the current state vs iterations



- We can clearly see that as the temperature decreases, the width of the Accepted range of current state also decreases which is, the generated current state becomes closer to zero.

Function value also approached to zero

- Now we calculate the Accepted Rates for each iteration of temperature values

- Initially set a high temperature value and decrease gradually according to a cooling factor.

  Cooling Factor = 0.8 (decrease by20%)

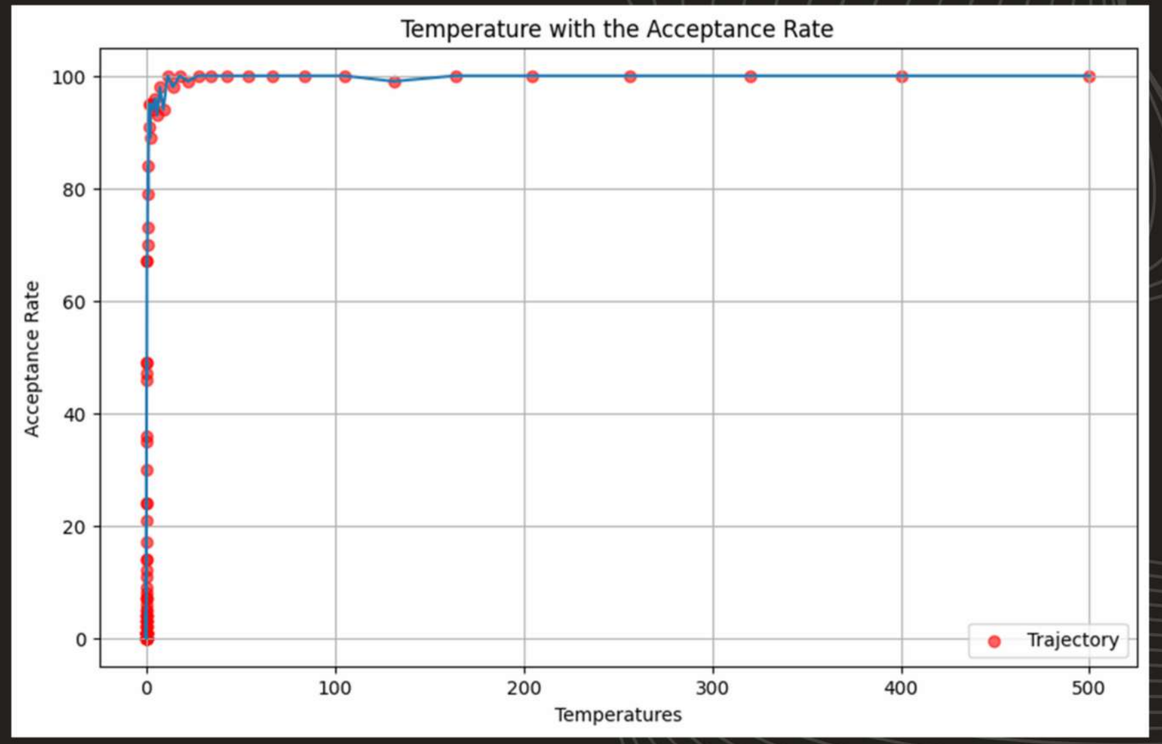
- Calculate the accepted current states for 100 iterations for each temperature.

Plot of the Acceptance Rates vs Temperatures

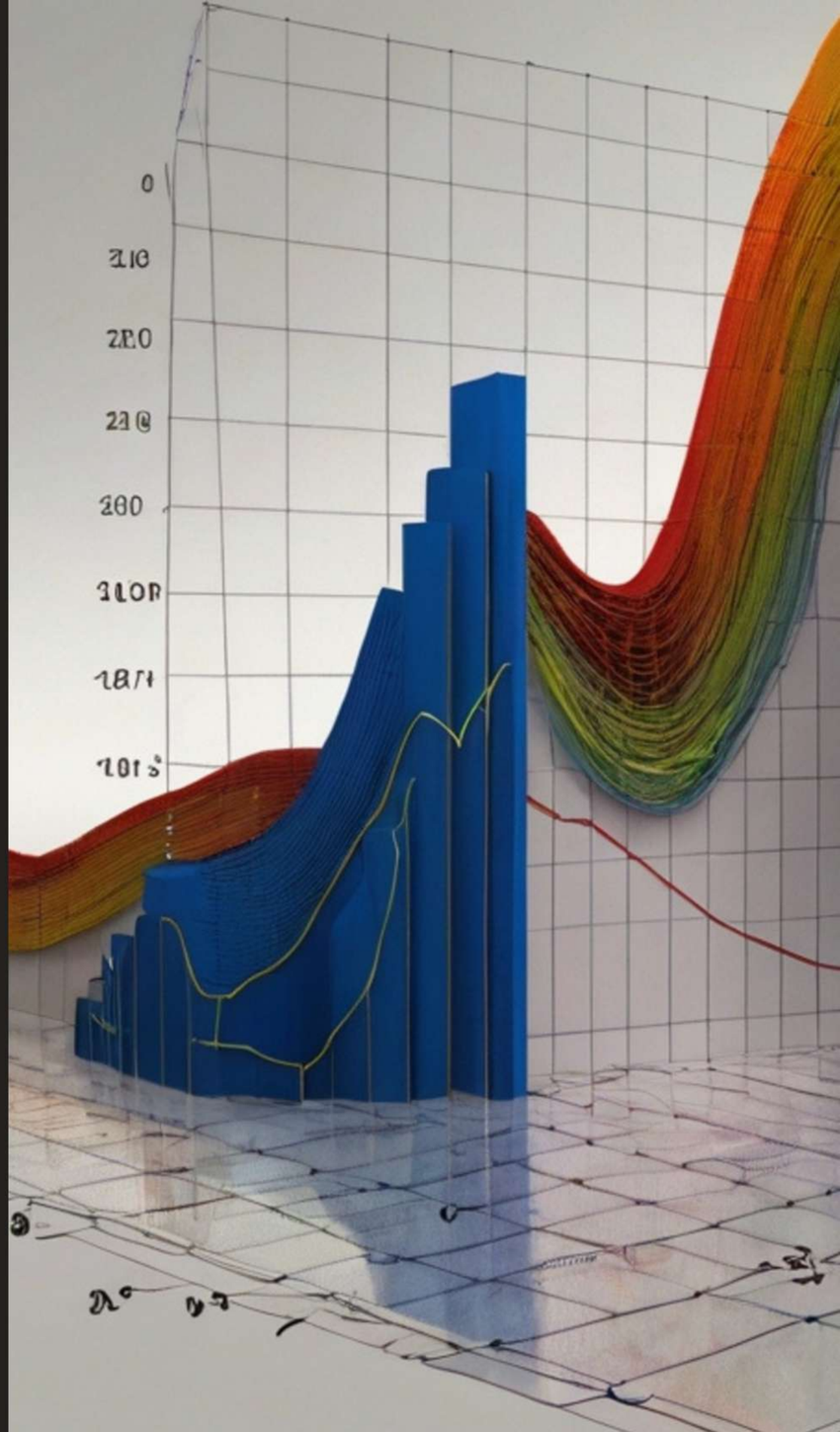
- We can clearly see that, as the temperature decreases, the Acceptance Rate also decreases.

- Which means, as the temperature is decreasing, the Boltzmann factor decreases, thus the number of accepted currents states starts to decrease.

- So accepted rate starts to decrease.



Simulated Annealing for Two variable Function



Simulated Annealing for Two Variables

The Function

$$f(x,y) = (x-3)^2 + (y+2)^2 + 5$$

We choose a two variable function to test our algorithm and show that we get the expected results.

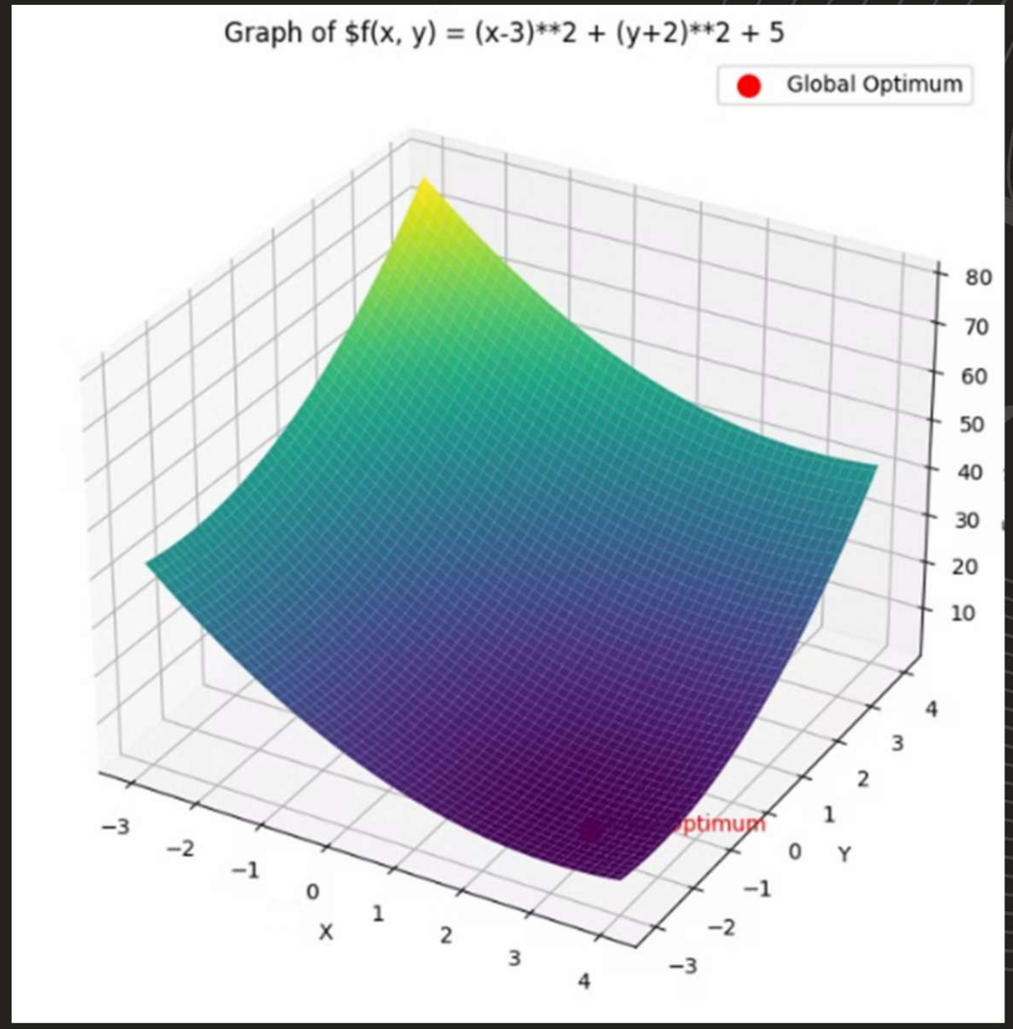
The Global Minimum of the Function

Global Optimum Point:

x = 3.0101010101010095

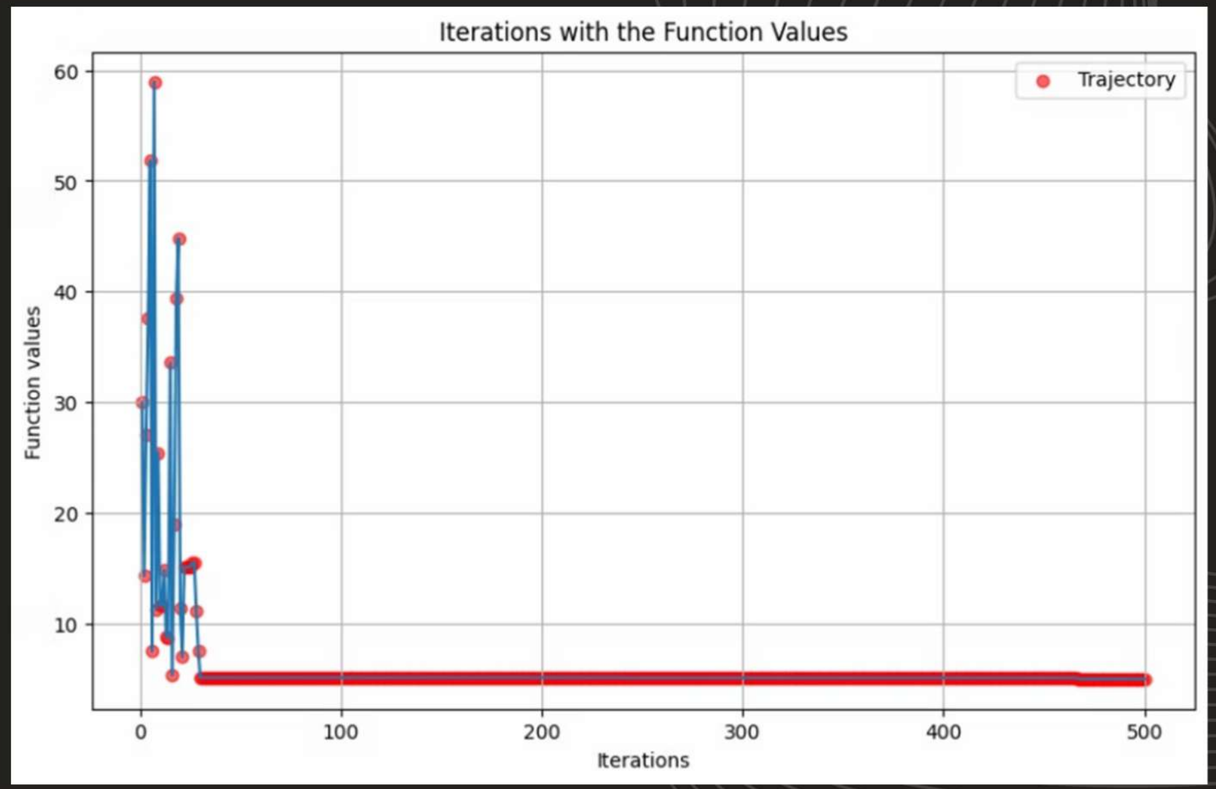
y = -2.0101010101010104

z = 5.000204060810121 5

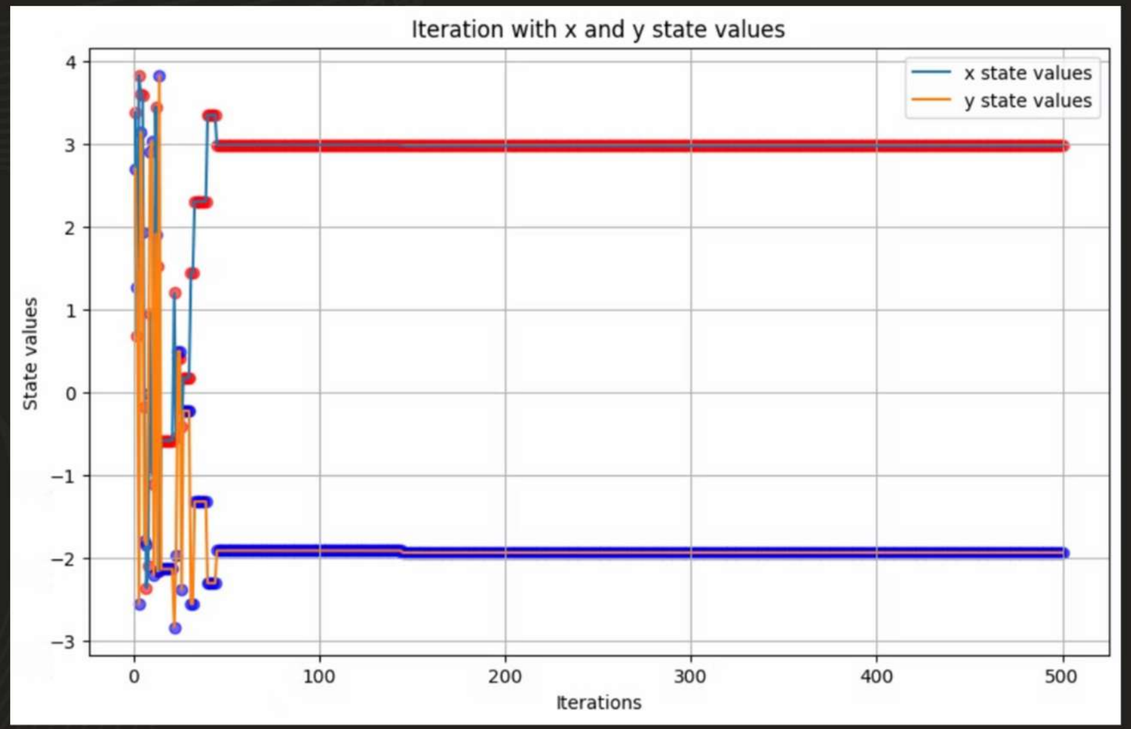


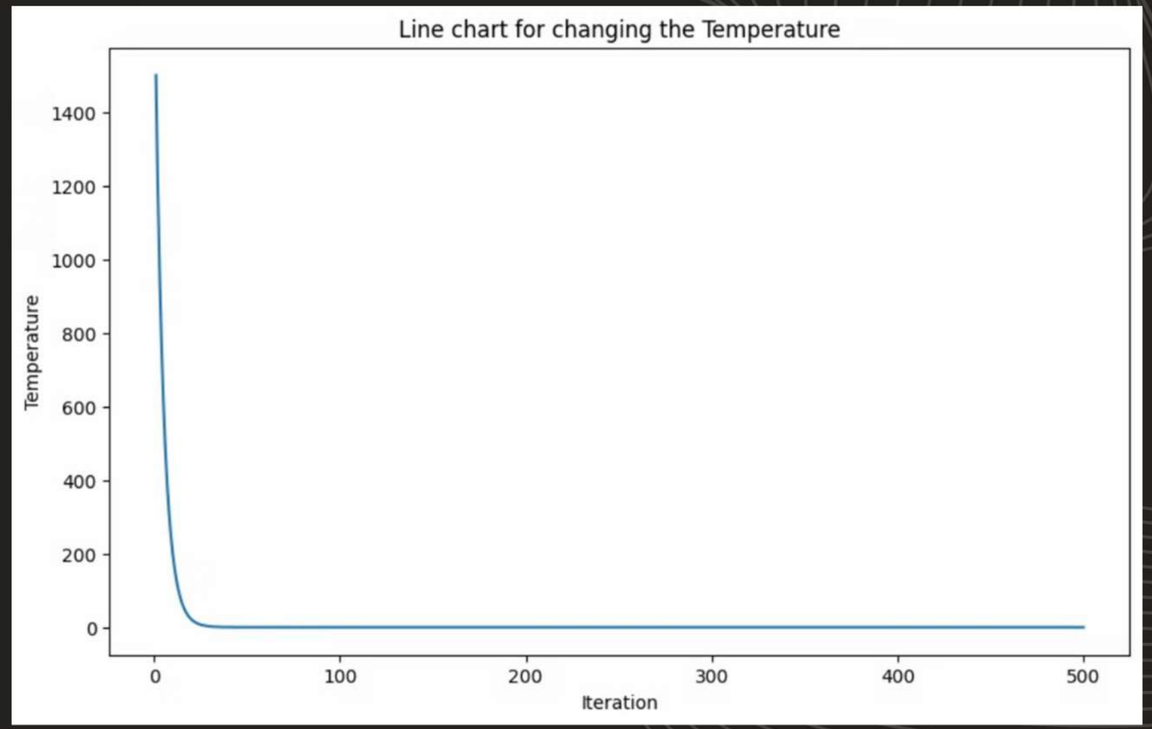
The Simulation Results

- Initial temperature was set to 1500K
- We have done 500 iterations
- by decreasing the temperature by 20%

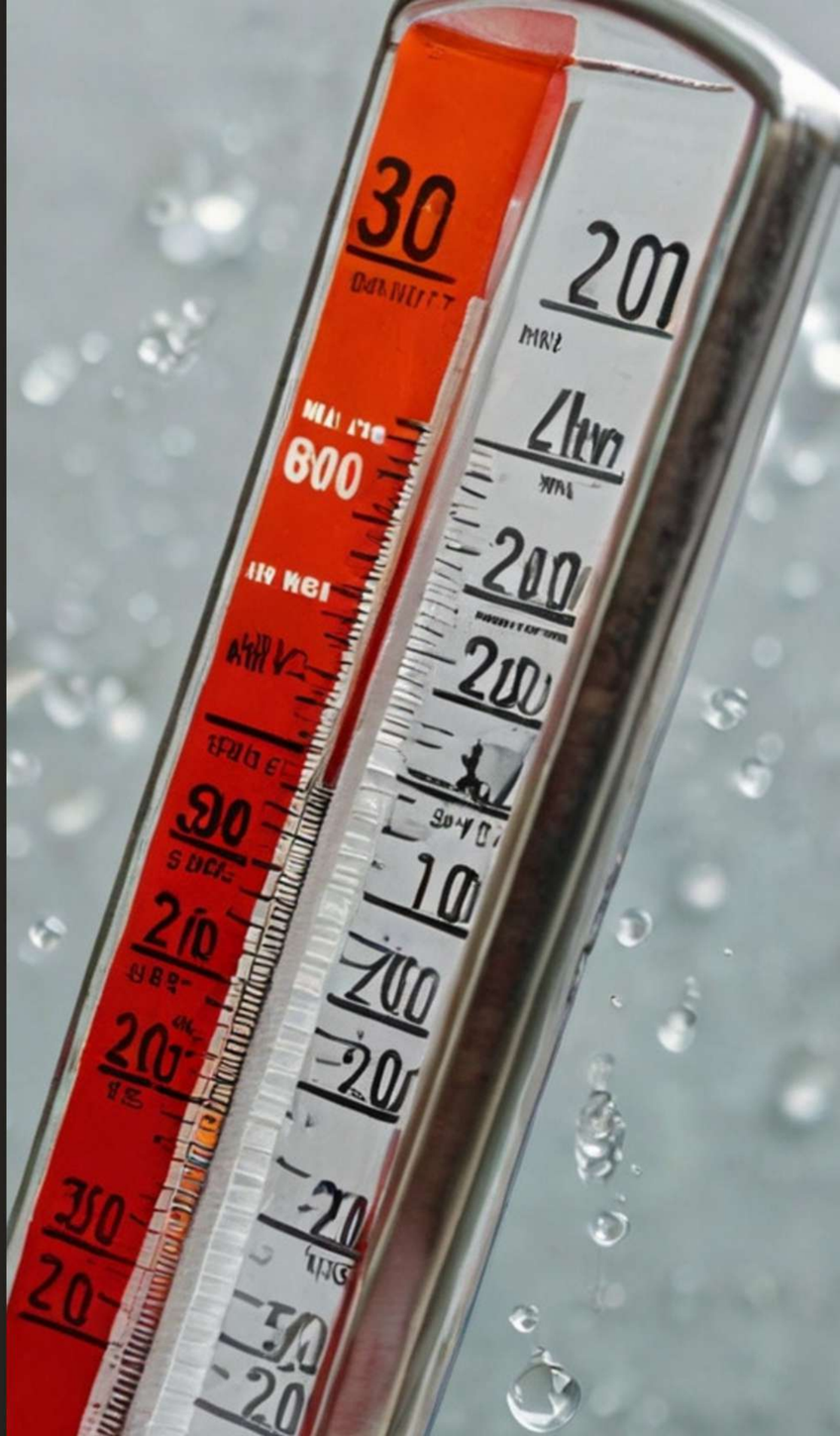


The Simulation Results





Cooling Schedules



Cooling Schedules.

Exponential schedule

$$T(k+1) = \alpha\, T(k)$$

$$\alpha = 0.95$$

Linear schedule

$$T(k+1) = T(k) - \Delta T$$

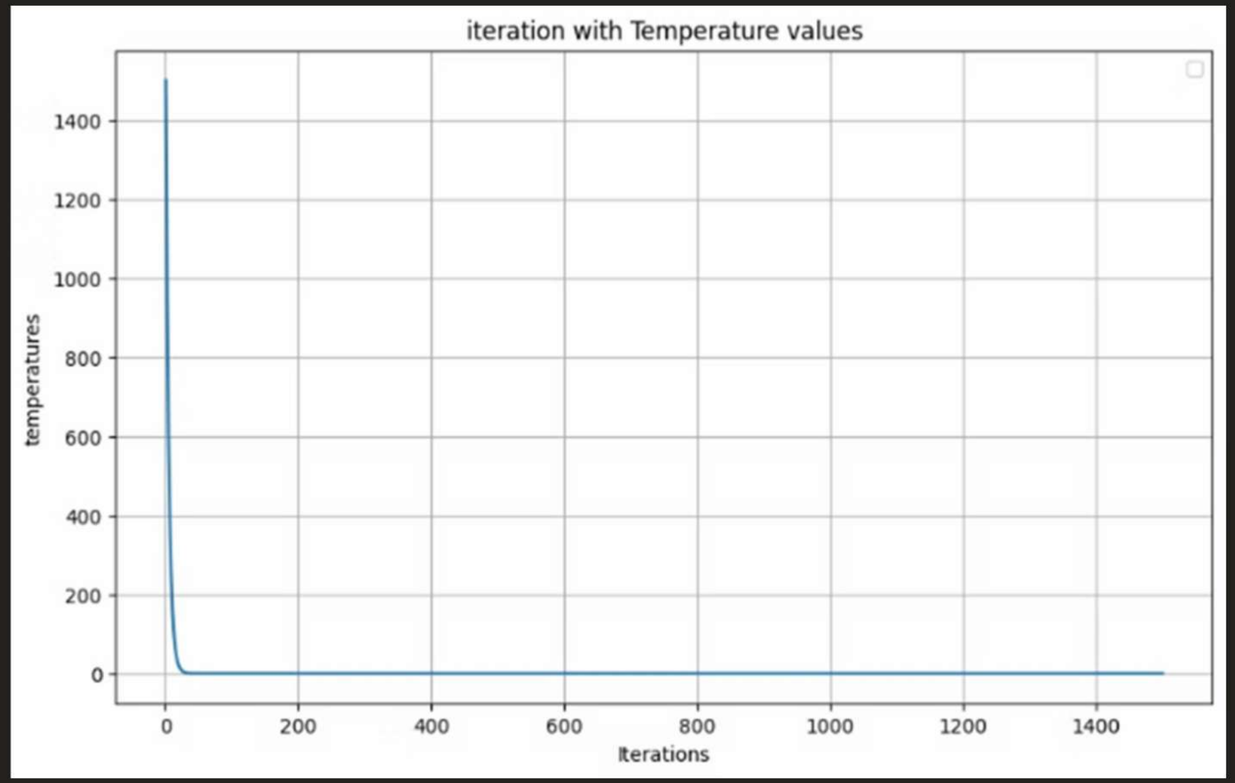
at L trials

With a **fixed ΔT**

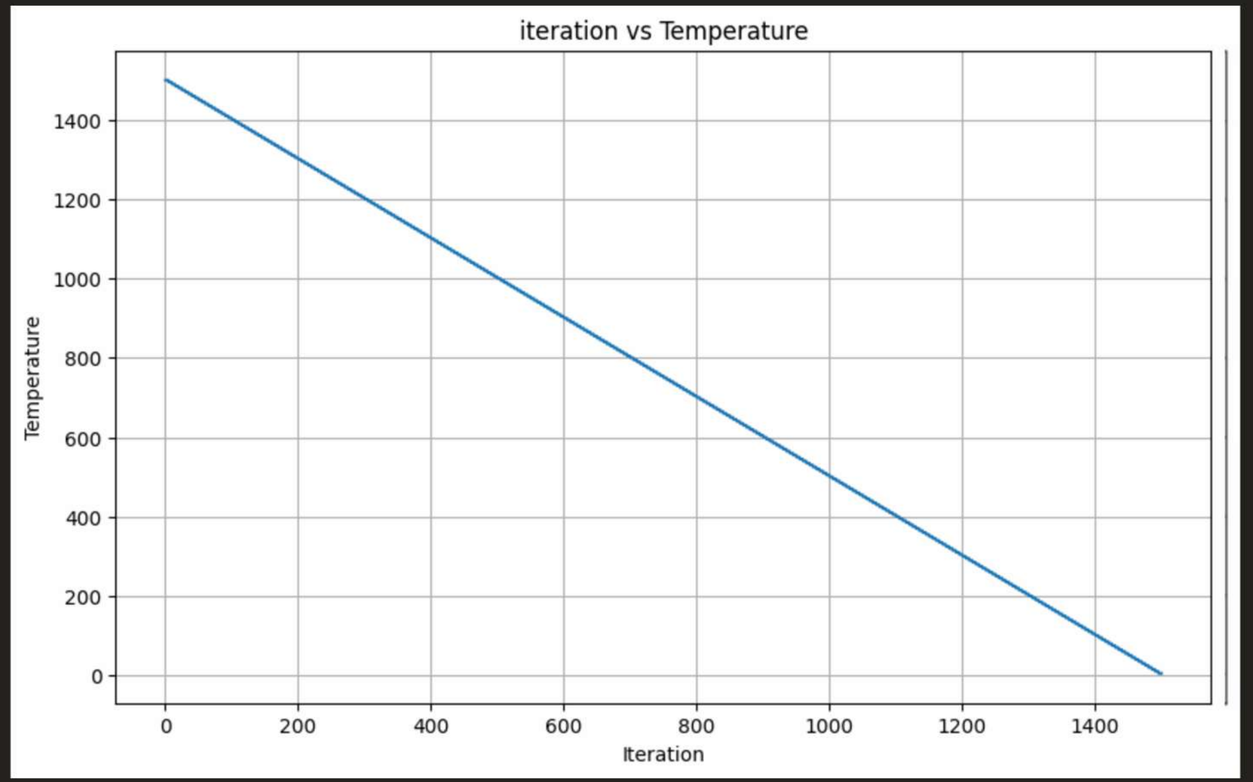
https://www.researchgate.net/publication/238690391_Simulated_annealing_overview

comparison of the two shedules

Exponential schedule

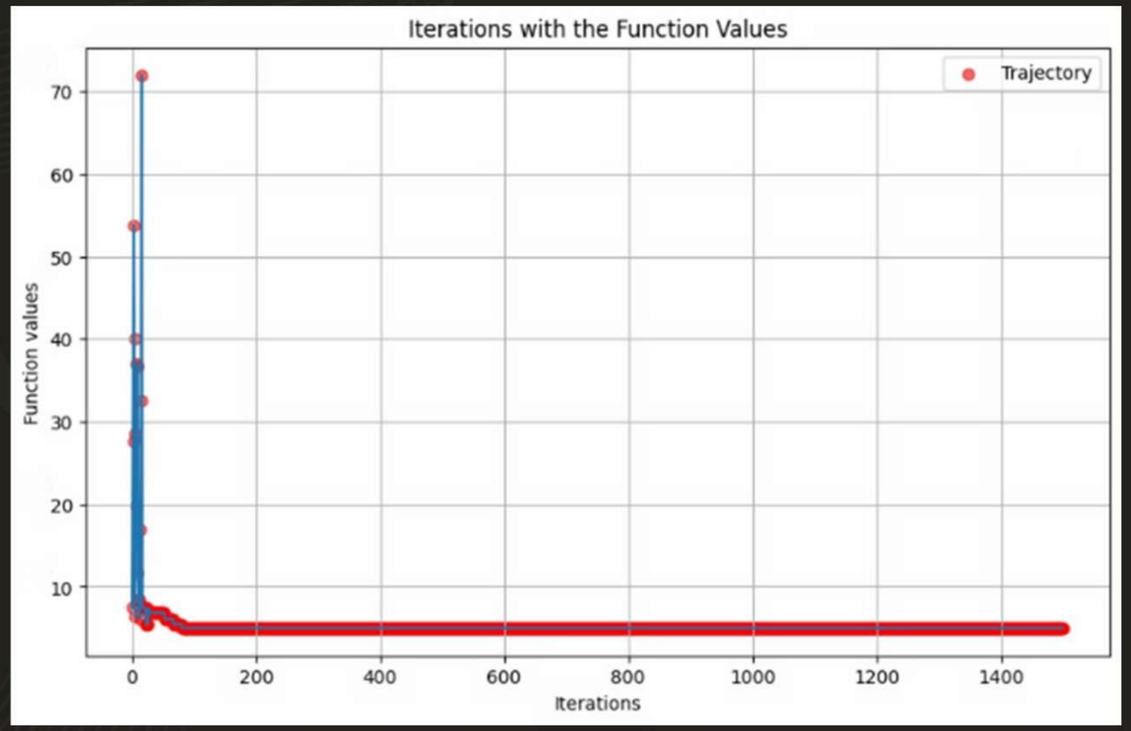


Linear schedule

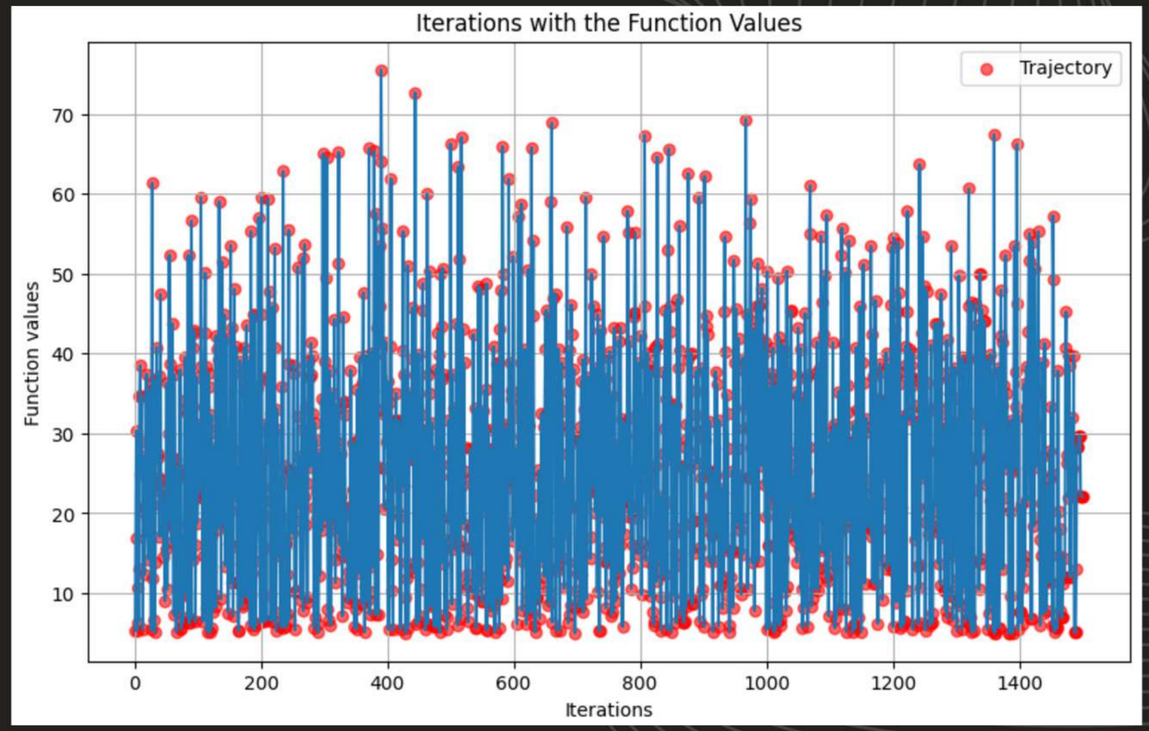


The Simulation Results

Exponential schedule



Linear schedule

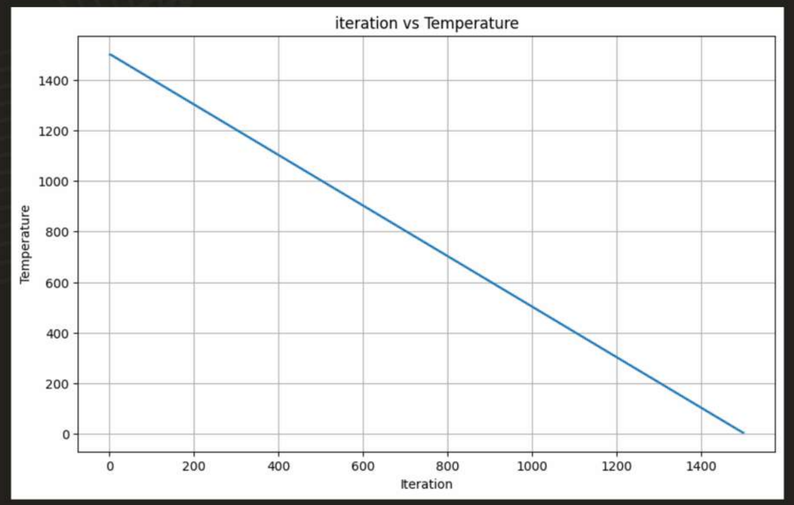


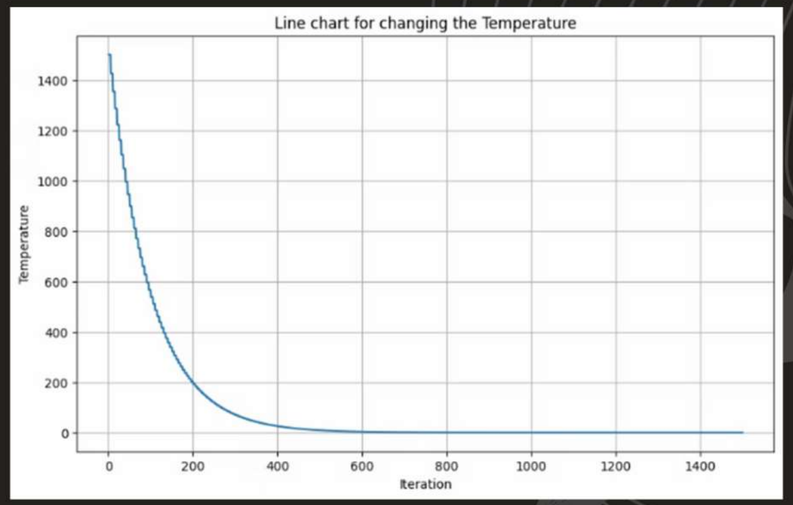
Alterations to the schedule

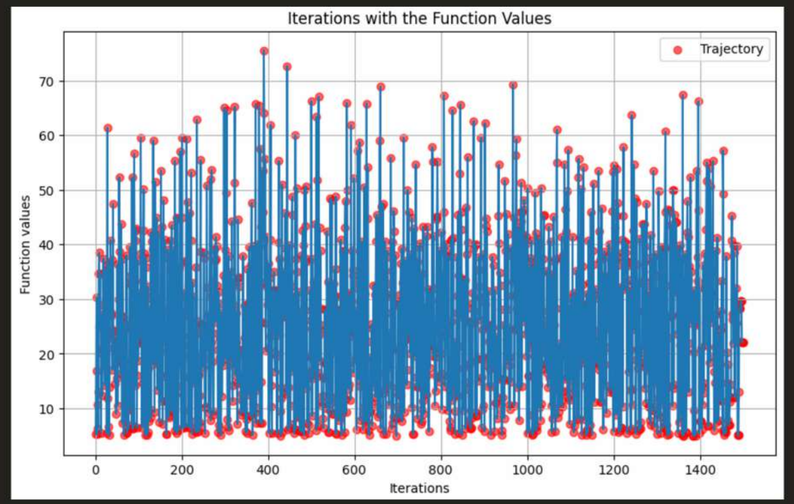
```
L = 5
delta_T = 5 # Decrement Value
if i % L == 0:
    temp = temp - delta_T
```

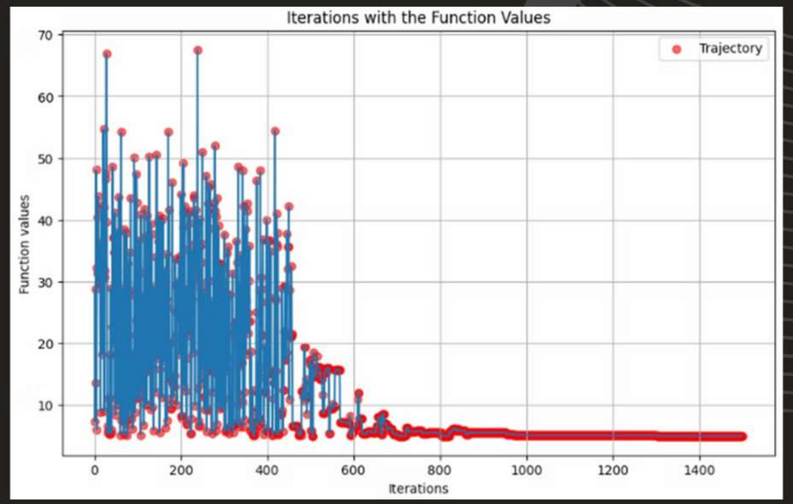
```
# Decrease the temperature
L = 5
delta_T = 0.05 * temp # Decrement Value
if i % L == 0:
    temp = temp - delta_T
```

Altercations to the shedule







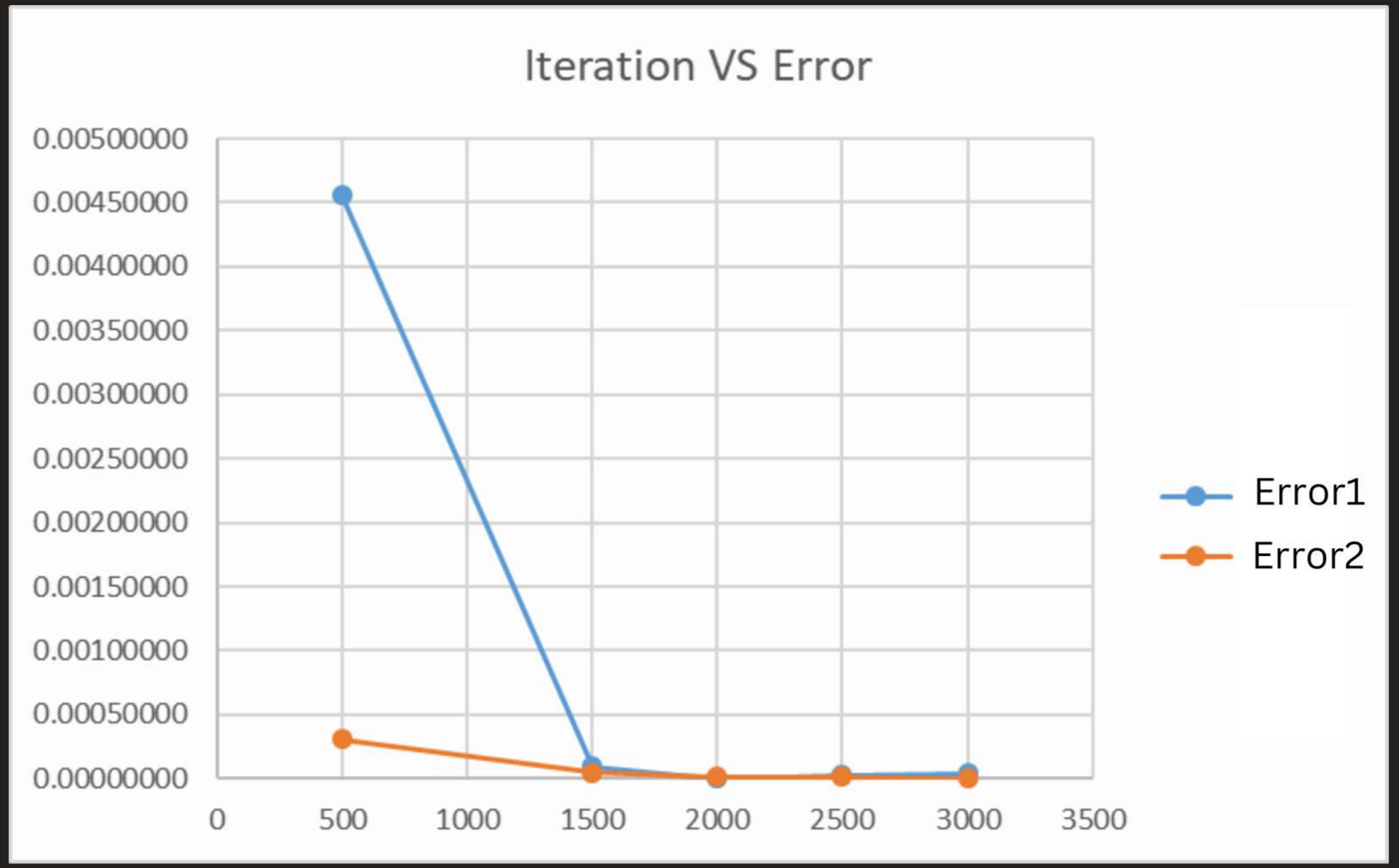


ERROR Comparisons

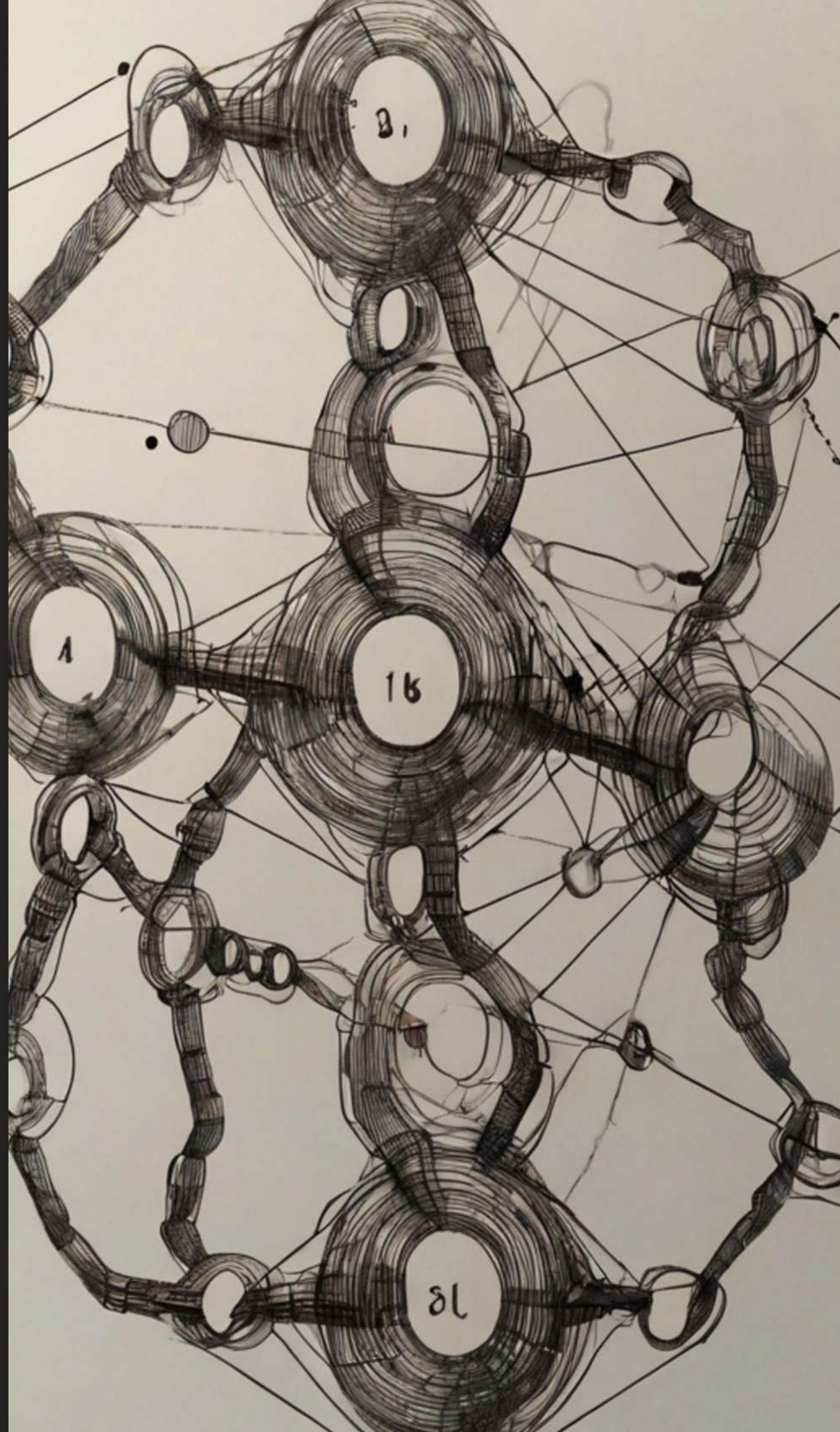
$$Error = \left[f(x)_y - f(x)\right]^2$$

ITERATIONS	Exponential cooling Shedule (alpha = 0.95)	Linear cooling shedule	error 1	error 2
500	5.067757399	5.01775322	0.00456345	0.00030797
1500	5.010063789	5.007227785	0.00009721	0.00004933
2000	5.000618402	5.003690734	0.00000017	0.00001216

ERROR Comparisons



Introduction to Markov Chain



What is a Markov Chain?

A Markov chain is a mathematical system that transitions from one state to another within a finite or countable number of possible states.

In simulated annealing, a Markov chain is used to explore the solution space more effectively.

It provides a structured yet flexible mechanism to navigate through the possible solutions, enhancing the likelihood of finding the optimal solution by balancing exploration and exploitation.

Modified Algorithm

1: Initialization: $i \leftarrow i_{\text{start}}, k \leftarrow 0, c_k \leftarrow c_0, L_k \leftarrow L_0$
2: **repeat**
3: **for** $i = 0$ to L_k **do**
4: Generate a solution j using uniformly distribution over S
5: **if** $f(j) \leq f(i)$ **then**
6: $i \leftarrow j$
7: **else**
8: **if** $e^{\frac{f(i)-f(j)}{c_k}} >$ Unif(0,1) **then**
9: $i \leftarrow j$
10: **end if**
11: **end if**
12: **end for**
13: $k \leftarrow k + 1$
14: calculate c_k
15: calculate L_k
16: **until** stop criterion

Introducing Markov Chain to our Algorithm

Now we set the Markov chain length as 50 for each iteration of temperature to reach equlibrium state.

cooling rate = 0.05
Initial temperature = 1500K
Iterations = 500
Markov Chain Length = 50

$$f(x,y) = (x-3)^2 + (y+2)^2 + 5$$

Optimal solution before:
x = 3.1292237321842062
y = -2.0535007633756712
f(x, y) = 5.01775322

```
# Decrease the temperature
L = 5
delta_T = 0.05 * temp # Decrement Value
if i % L == 0:
    temp = temp - delta_T
```

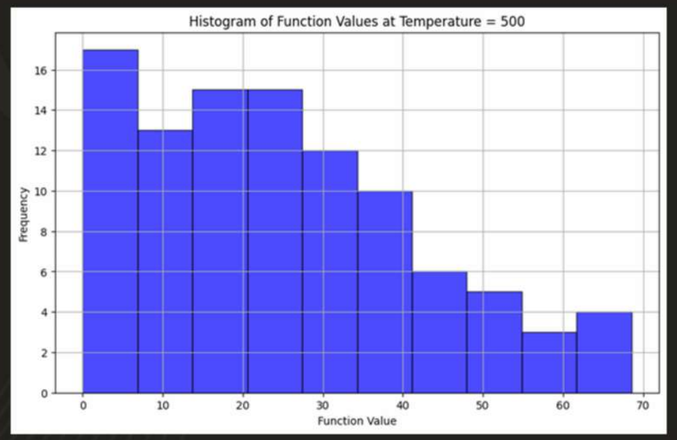
New Solution:

x = 3.012449423914802
y = -2.014115868937141
f(x,y) = 5.000354245911660 6

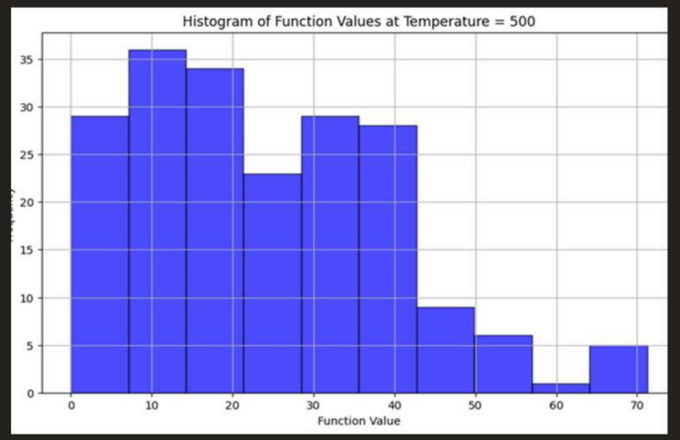
Markov chain for different Lengths

- When we plot the distribution of function values after introducing markov chain trails it should follow boltzman distribution .

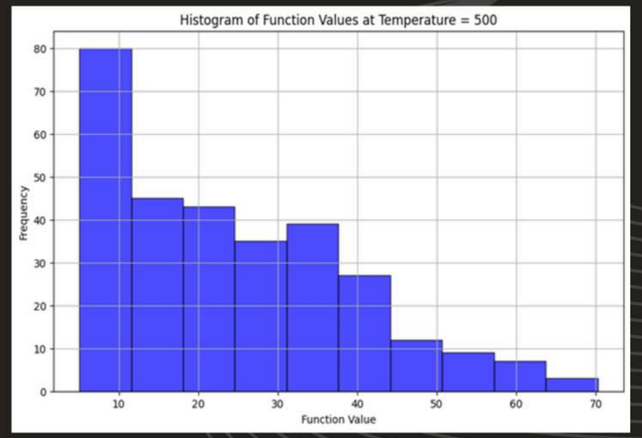
Markov chain length = 100



Markov chain length = 200



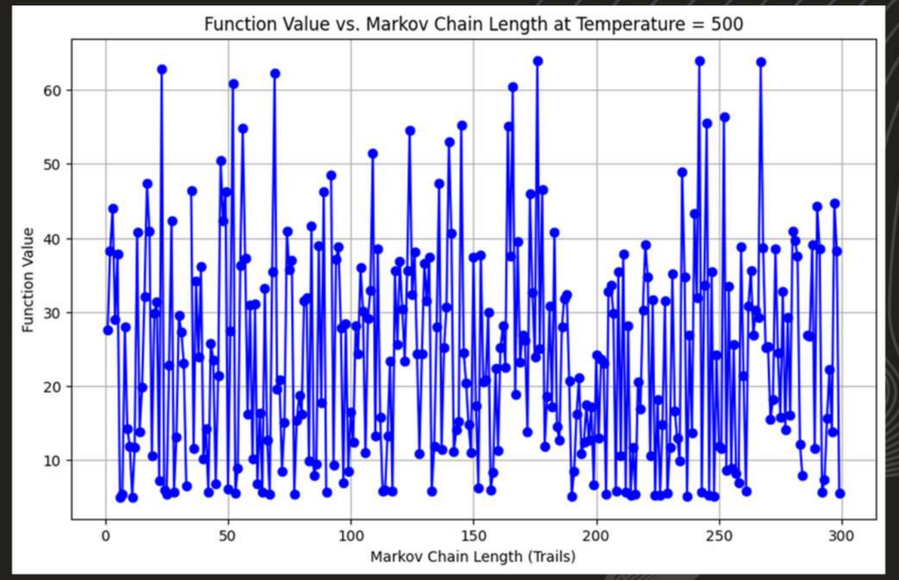
Markov chain length = 300

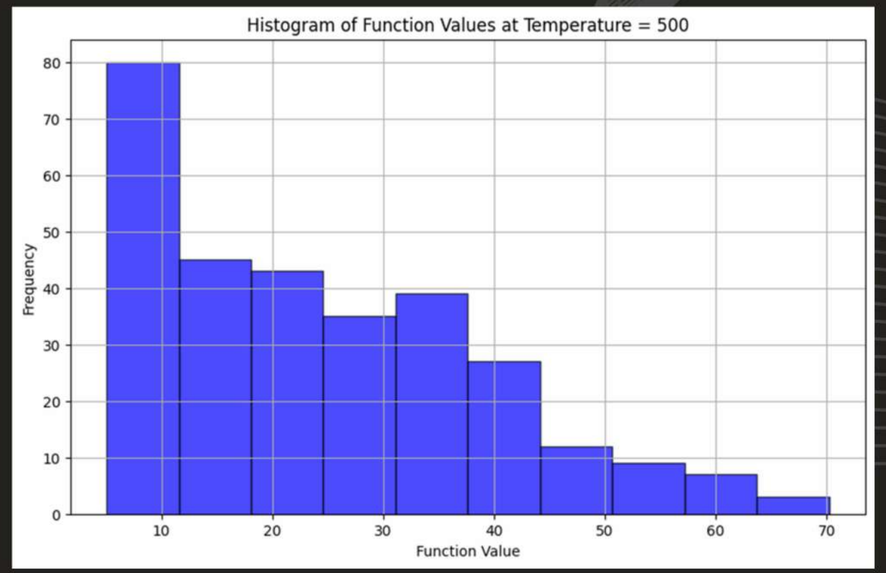


Markov chain for different temperatures

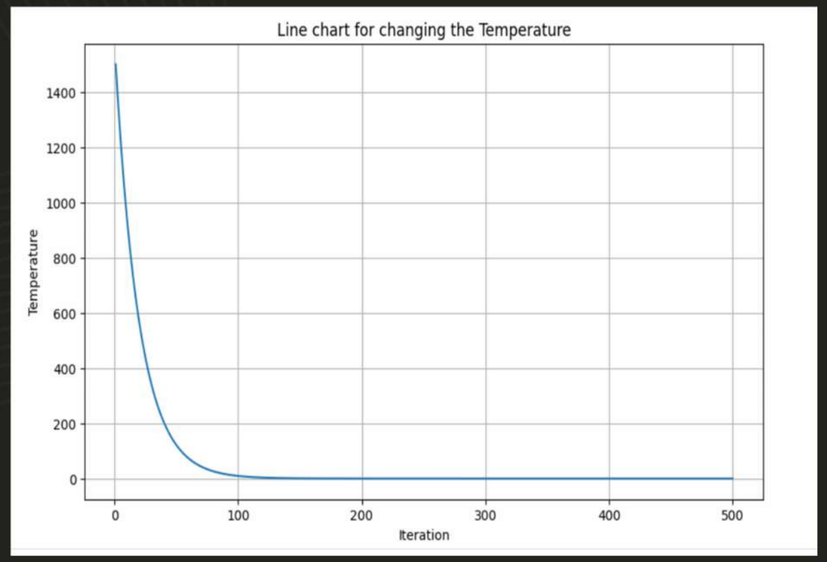
Now we check how the function value behaves in each Markov Chain at certain temperatures.

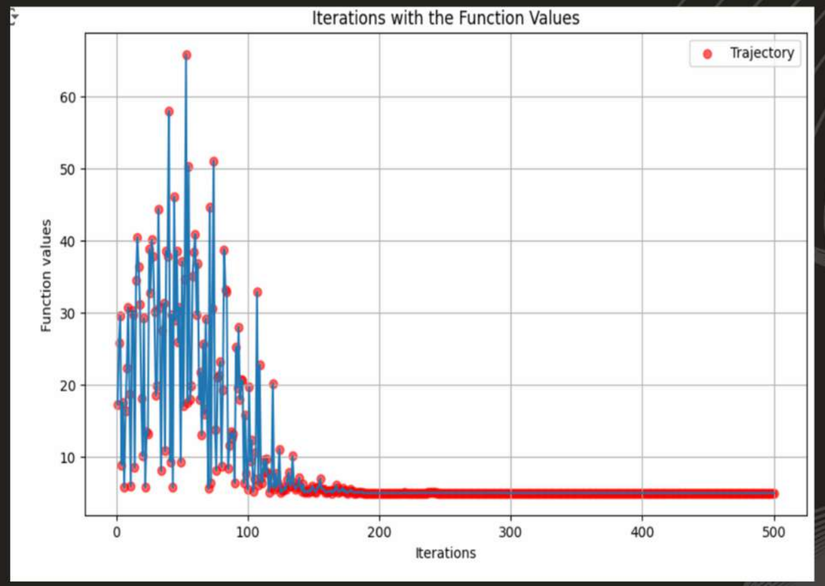
As expected this histogram starts to represent a boltzmann distribution.

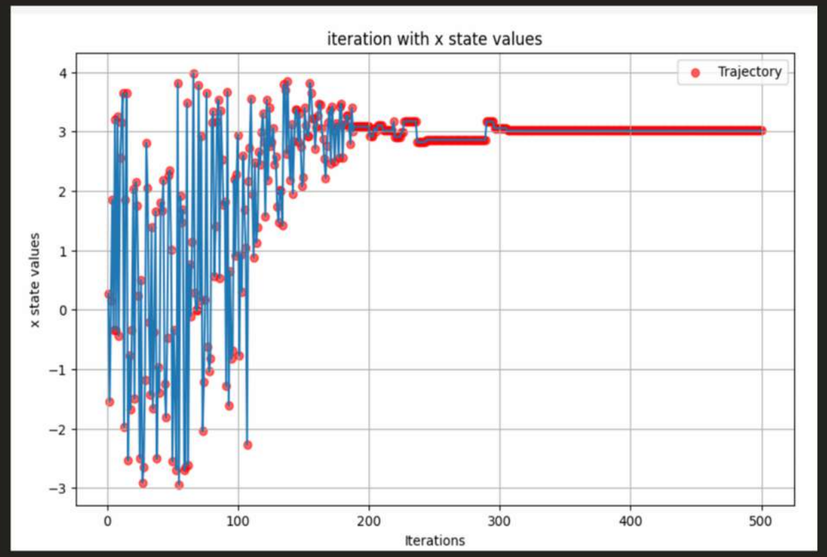


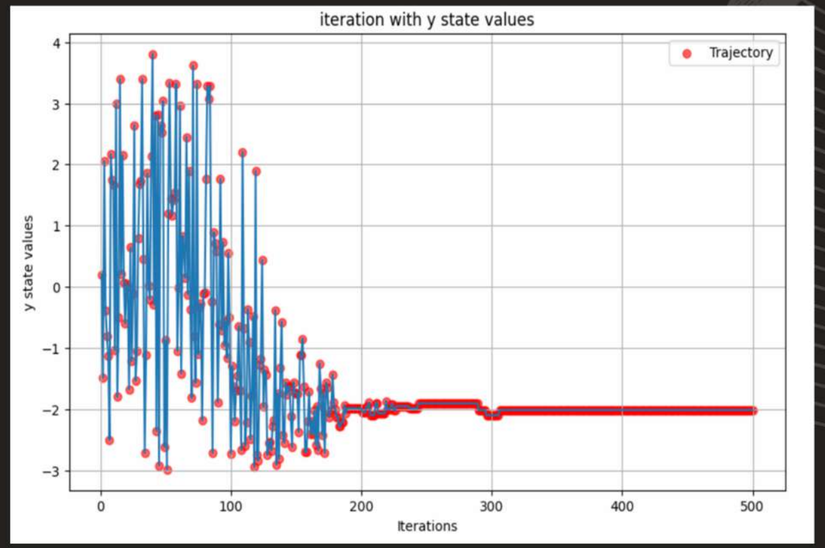


Final output using SA algorithm with markov chain









Bench Mark Functions

Why we choose Benchmark functions

- Main purpose of using the test function is to evaluate the behavior and performance of the algorithm.
- For this purpose, the researchers compiled a rich set of benchmark functions for unconstrained optimization problems with diverse properties in terms of modality, separability, and valley landscape.
- Test of reliability, efficiency and validation of optimization algorithms is frequently carried out by using a chosen set of common standard benchmarks or test functions.

Types of bench mark functions

A benchmark function is a standard test function used to evaluate and compare the performance of optimization algorithms

- Unimodal Functions:
  - Have a single global optimum
  - Example: Sphere Function
- Multimodal Functions:
  - Have multiple local optima
  - Example: Rastrigin Function
- Composite Functions:
  - Combine multiple functions to create complex landscapes
  - Example: Schwefel Function

Advantages

1. **Standardization** : Benchmark problems provide a standardized set of test cases. Researchers can compare different optimization algorithms fairly using the same benchmarks.
2. **Complexity** : Normal functions may not adequately represent real-world optimization challenges. Benchmarks mimic complex landscapes with multiple local optima, allowing evaluation under realistic conditions.
3. **Known Optima** : Benchmarks have known global optima, making it easier assess an algorithm's convergence. Real-world problems rarely have known optimal solutions.
4. **Diversity** : Benchmarks cover various problem types (unimodal, multimodal, noisy). This diversity helps evaluate algorithm robustness across scenarios.

Out of 170 bench mark functions we selected a few

3 Benchmark Test Functions for Global Optimization

Now, we present a collection of 175 unconstrained optimization test problems which can be used to validate the performance of optimization algorithms. The dimensions, problem domain size and optimal solution are denoted by D, $Lb \leq \mathbf{x}_i \leq Ub$ and $f(\mathbf{x}^*) = f(x_1, ...x_n)$, respectively. The symbols Lb and Ub represent lower, upper bound of the variables, respectively. It is worth noting that in several cases, the optimal solution vectors and their corresponding solutions are known only as numerical approximations.

1. **Ackley 1 Function** [9](Continuous, Differentiable, Non-separable, Scalable, Multimodal)

$$f_1(x) = -20e^{-0.02\sqrt{D^{-1}\sum_{i=1}^{D} x_i^2}} - e^{D^{-1}\sum_{i=1}^{D}\cos(2\pi x_i)} + 20 + e$$

subject to $-35 \leq x_i \leq 35$. The global minima is located at origin $\mathbf{x}^* = (0, \cdots, 0)$, $f(\mathbf{x}^*) = 0$.

2. **Ackley 2 Function** [1] (Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal)

$$f_2(x) = -200e^{-0.02\sqrt{x_1^2+x_2^2}}$$

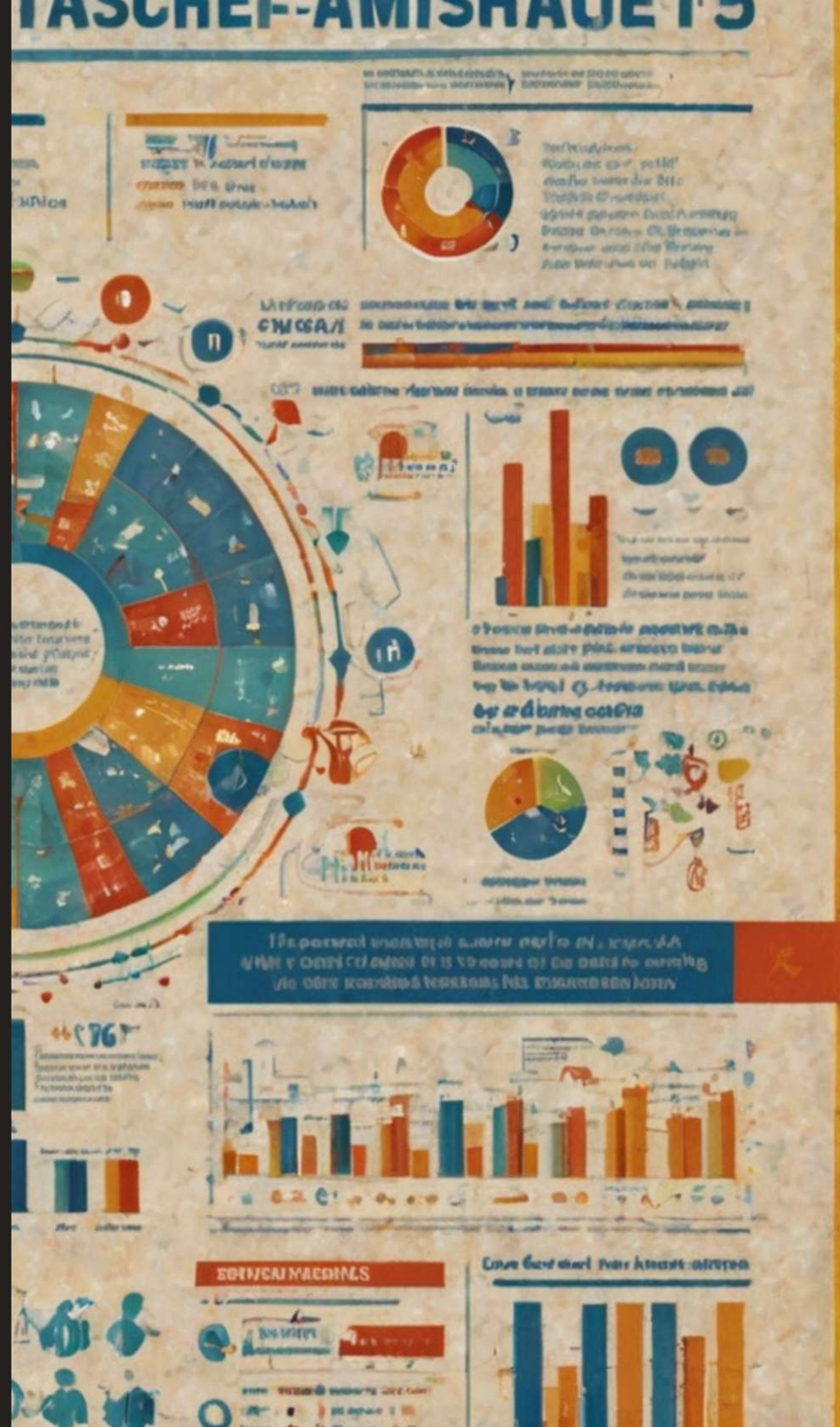
subject to $-32 \leq x_i \leq 32$. The global minimum is located at origin $\mathbf{x}^* = (0, 0)$, $f(\mathbf{x}^*) = -200$.

3. **Ackley 3 Function** [1] (Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal)

$$f_3(x) = 200e^{-0.02\sqrt{x_1^2+x_2^2}} + 5e^{\cos(3x_1)+\sin(3x_2)}$$

subject to $-32 \leq x_i \leq 32$. The global minimum is located at $\mathbf{x}^* = (0, \approx -0.4)$, $f(\mathbf{x}^*) \approx -219.1418$.

THE RESULTS



UniModal Function

Ackley 2 Function :

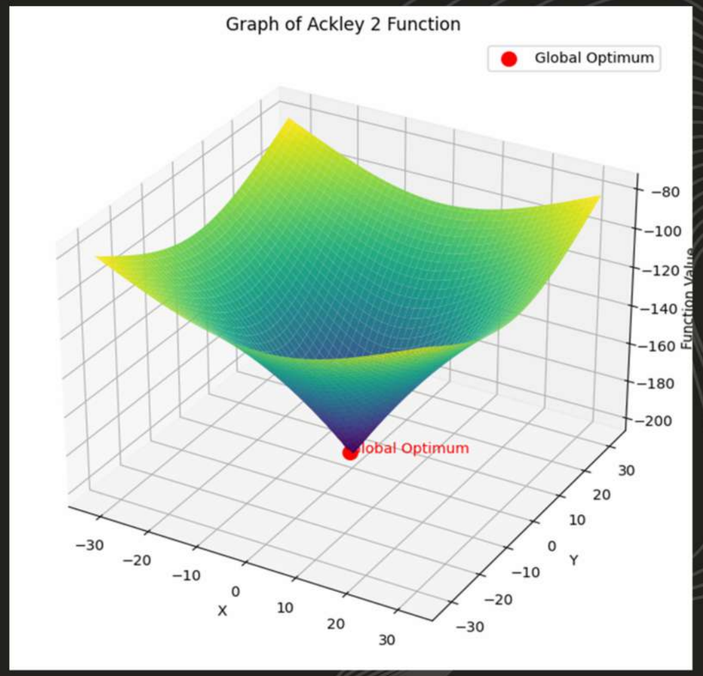
(Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal)

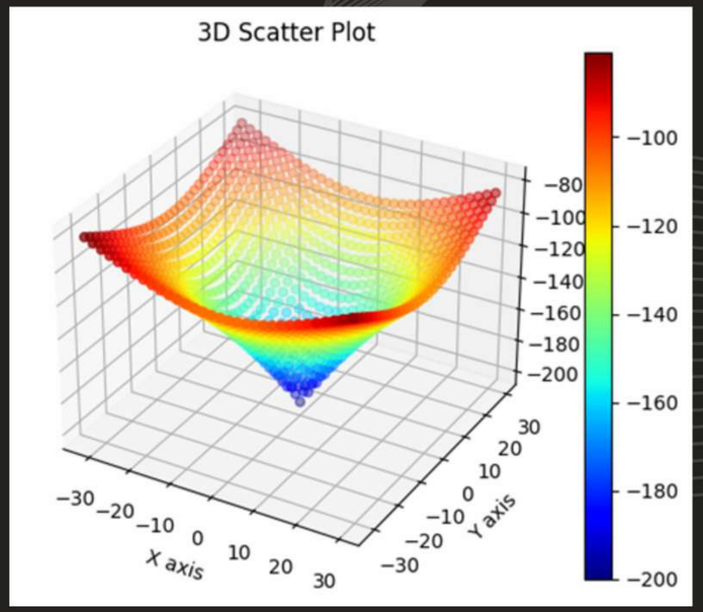
$$f(x_1, x_2) = -200e^{-0.02\sqrt{x_1^2+x_2^2}}$$

subject to, $x_i \in [-32, 32]$ for $i = 1, 2$.

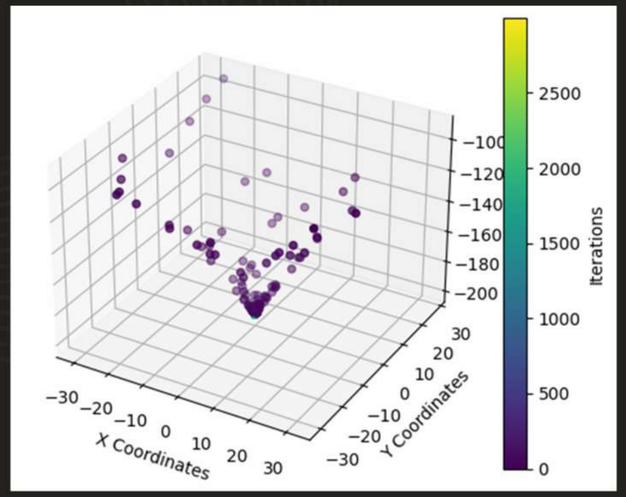
The global minimum is located at origin,

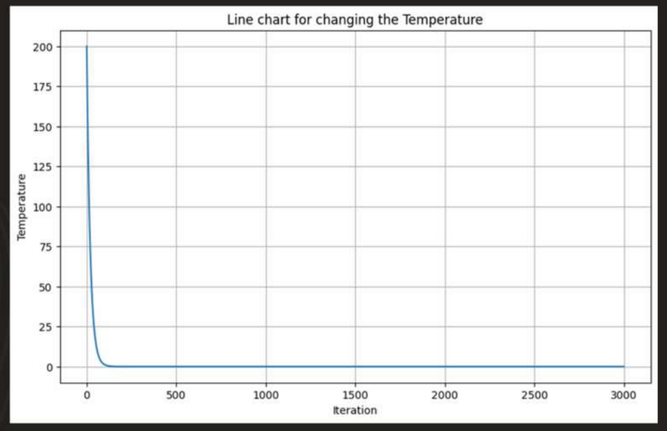
$x^* = (0,0)$ $f(x, y) = -200$

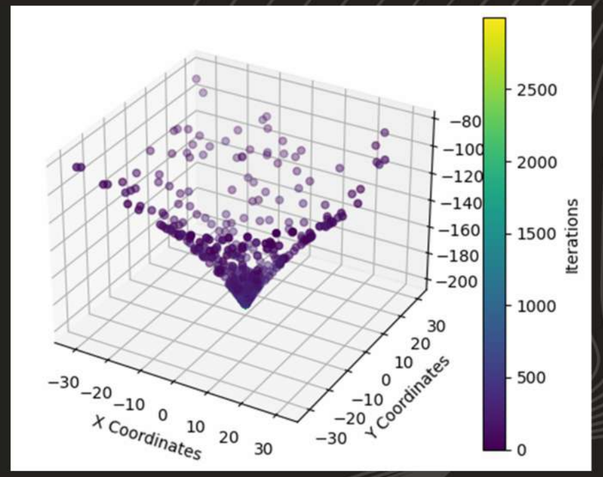


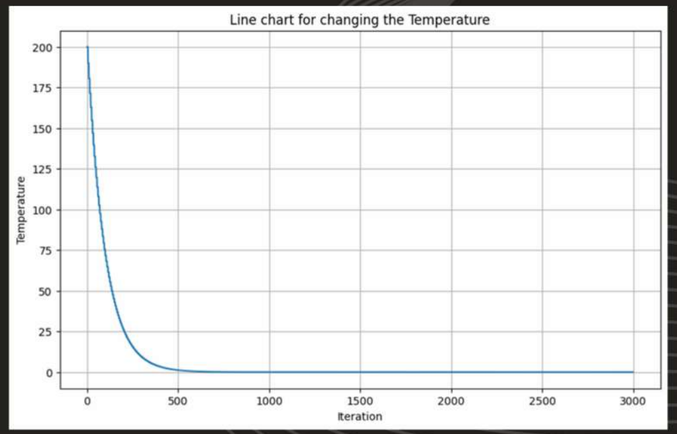


Former Cooling Schedule Vs. New Schedule







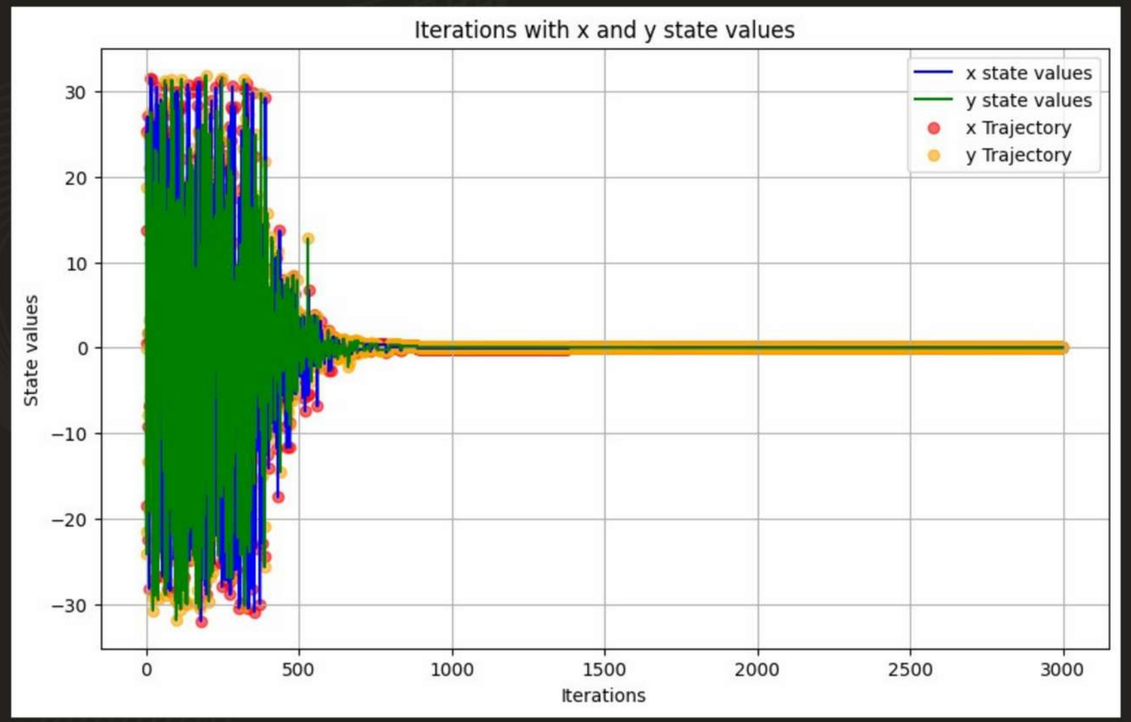


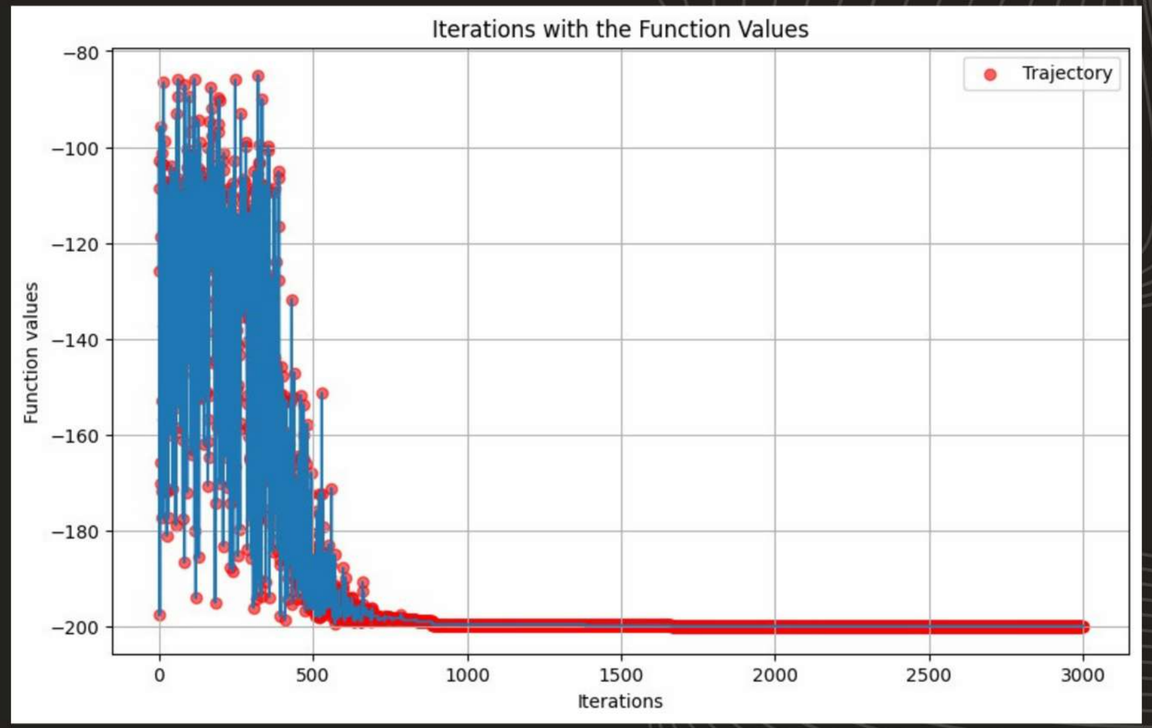

```
#earlier schedule
temp = temp*0.95
```

```
Best Solution: (0.0006778854153779434, -0.020838515811583136)
Best Value: -199.9166192303517
```

```
# Decrease temperature once every 5 iterations
if (i + 1) % 5 == 0:
  delta_T = temp * 0.05
  temp -= delta_T
```

Achieved Optimal values





Multi Modal Function

Adjiman Function

(Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

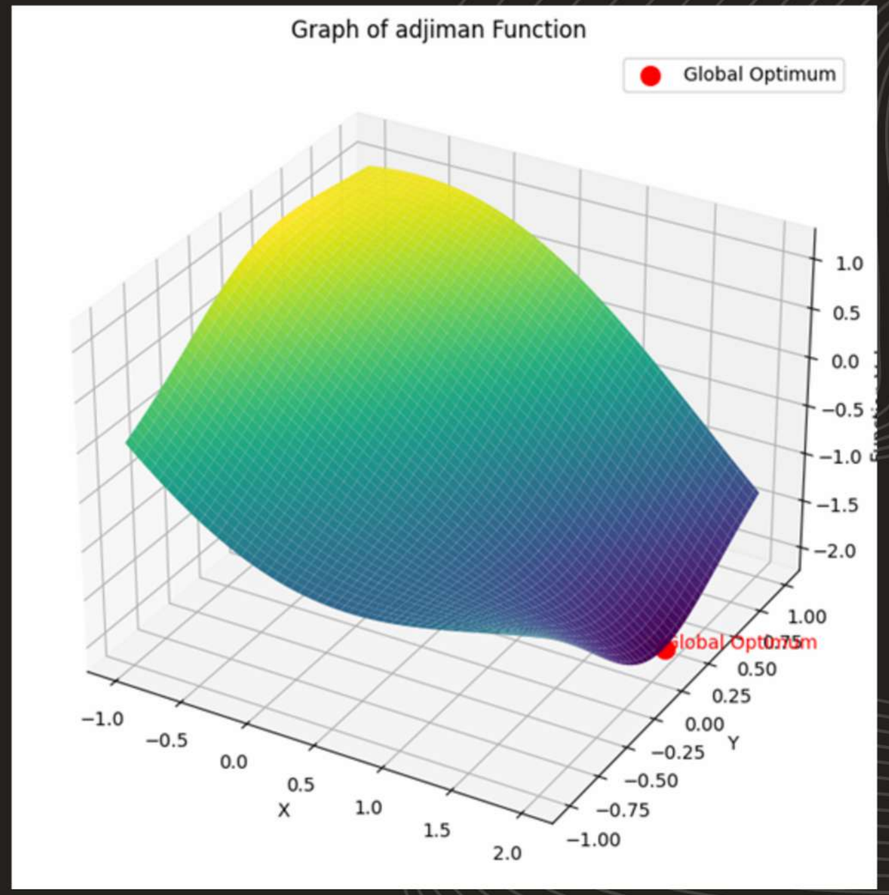
$$f_5(x) = \cos(x_1)\sin(x_2) - \frac{x_1}{(x_2^2+1)}$$

Subject to

$$-1 \leq x_1 \leq 2,\ -1 \leq x_2 \leq 1.$$

The global minimum is located at

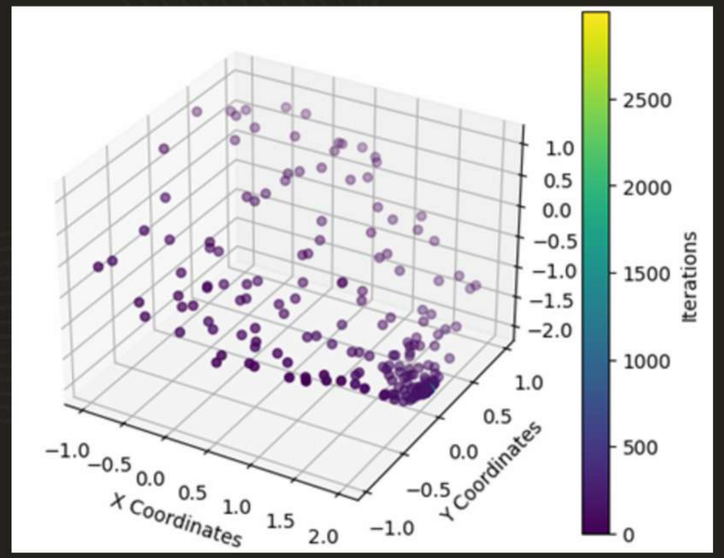
$$x^* = (2, 0.10578) \qquad f(x,y) = -2.02181$$

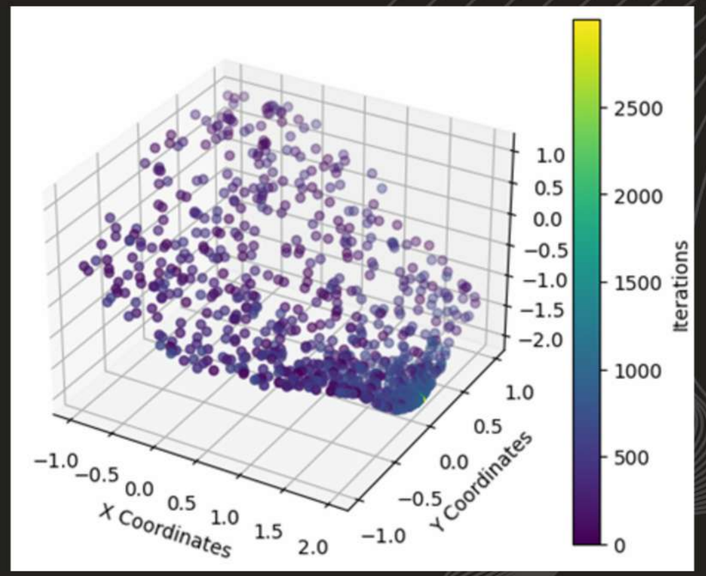


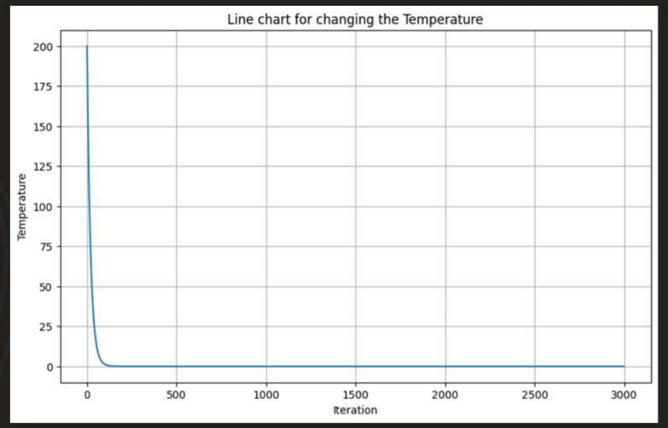
our solution

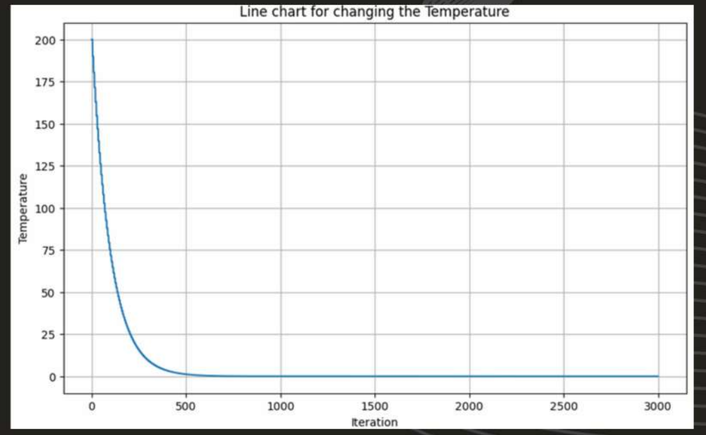
```
Best Solution: (1.999689132690134, 0.110514870852441 72)
Best Value: -2.0214262268840 52
```

Former cooling Schedule Vs. New Schedule










```
Best Solution: (1.9999669758948477, 0.09783202299373683)
Best Value: -2.021651195634284
```

References

- Busetti, F. (2001). Simulated annealing overview. ResearchGate. https://www.researchgate.net/publication/238690391_Simulated_annealing_overview

- Momin Jamil and Xin-She Yang, A literature survey of benchmark functions for global optimization problems, Int. Journal of Mathematical Modelling and Numerical Optimisation, Vol. 4, No. 2, pp. 150-194 (2013). DOI: 10.1504/IJMMNO.2013.055204

- Henderson, D., Jacobson, S. H., & Johnson, A. W. (2006). The theory and practice of simulated annealing. In Kluwer Academic Publishers eBooks (pp. 287-319). https://doi.org/10.1007/0-306-48056-5_10

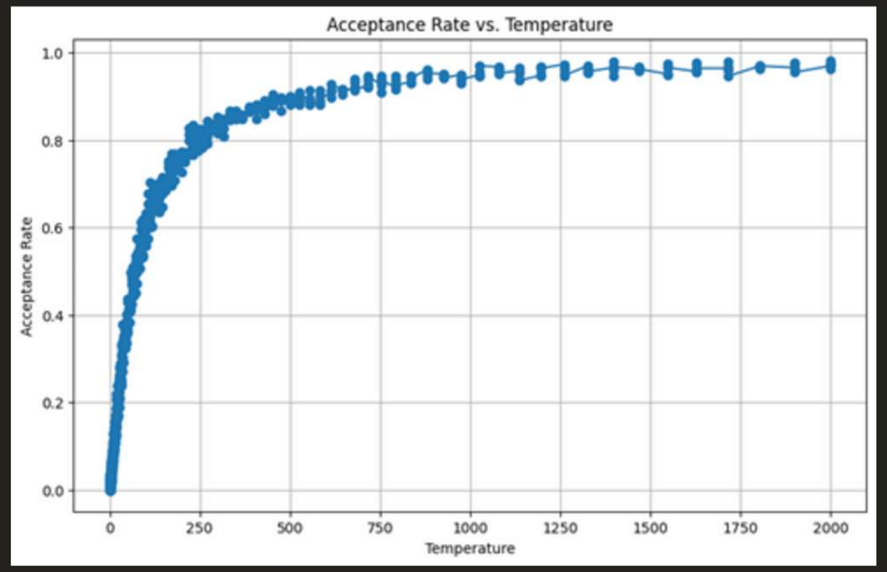
- Wang, Z.; Wang, Y.; Xu, H.; Xie, H. Application of Simulated Annealing Algorithm in Core Flow Distribution Optimization. Energies 2022, 15, 8242. https://doi.org/10.3390/en15218242

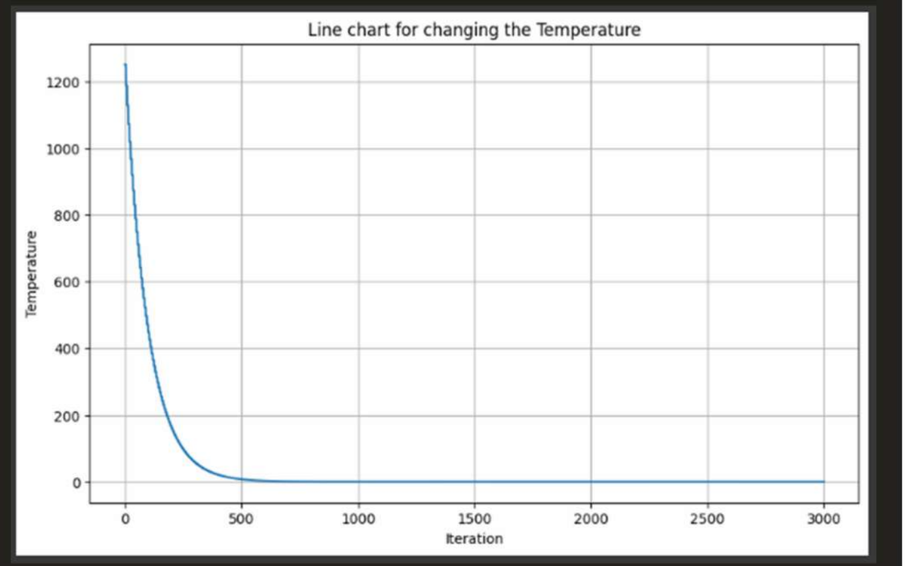
THANK YOU

Benchmark function collab codes

65	https://colab.research.google.com/drive/1QcoHQzNCjS804654TGT4MWTxoWiD22DR?usp=sharing
45	https://colab.research.google.com/drive/1VABsrIqSubN_ezNcyQyqTml4crrH-T-h?usp=sharing
29	https://colab.research.google.com/drive/1xrXqdir1cuj5eB9BCmyAx655j5FrELeu?usp=sharing
33	https://colab.research.google.com/drive/1zbAQuREuOmAjso24L2gxTnuU_LKpHr9v?usp=sharing
26	https://colab.research.google.com/drive/1ZdcyRrf8Td9ubQhFqqnVI-PW1x27JaAQ?usp=sharing
30	https://colab.research.google.com/drive/1V1TQP0ASrWOv8vSdDsAHp2JPKwl8bUlb?usp=sharing
5	https://colab.research.google.com/drive/1Osyf-vfORVLfm26qBg-Vd01p9aaIfpdS?usp=sharing
10	https://colab.research.google.com/drive/1nLxP1_cMDdYWRgAPQc-HuRefBYy-SE_T?usp=sharing
129	https://colab.research.google.com/drive/1Z3q1BSNwUZWgPlyi0GNZp1Z4gvZoogq5?usp=sharing
107	https://colab.research.google.com/drive/1yTB20UgA0zYNdFNXbmQwm34v8V49EmNB?usp=sharing
116	https://colab.research.google.com/drive/1Yo2ToTltew8JpEFyJtzaTJevw-rBCT5P?usp=sharing
52	https://colab.research.google.com/drive/1QOR4hfy0hR2vqdijxPaFjAVeUUrF5n_U?usp=sharing

Initial Temperature 750K



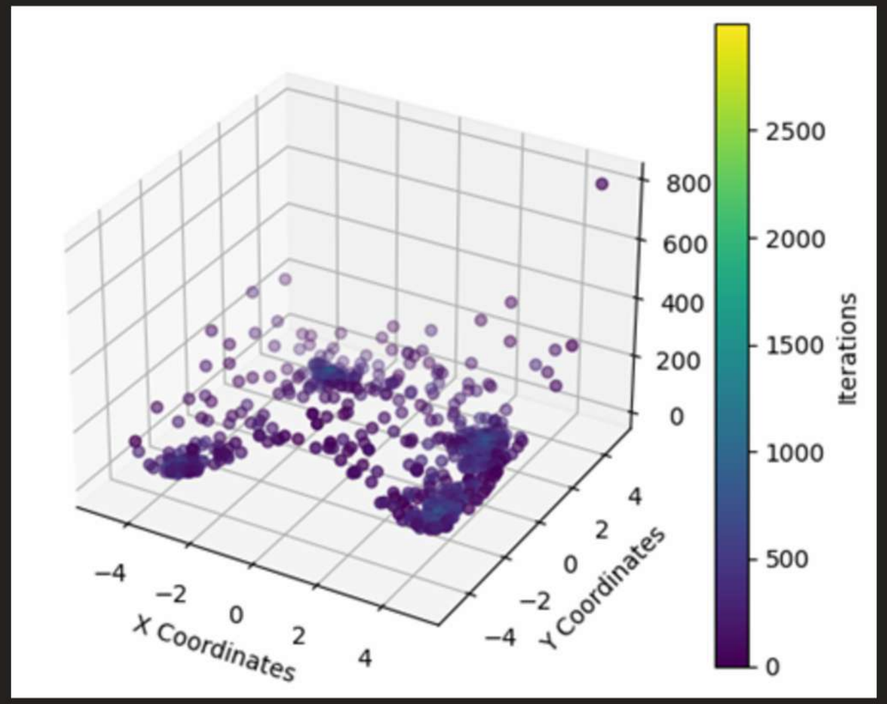


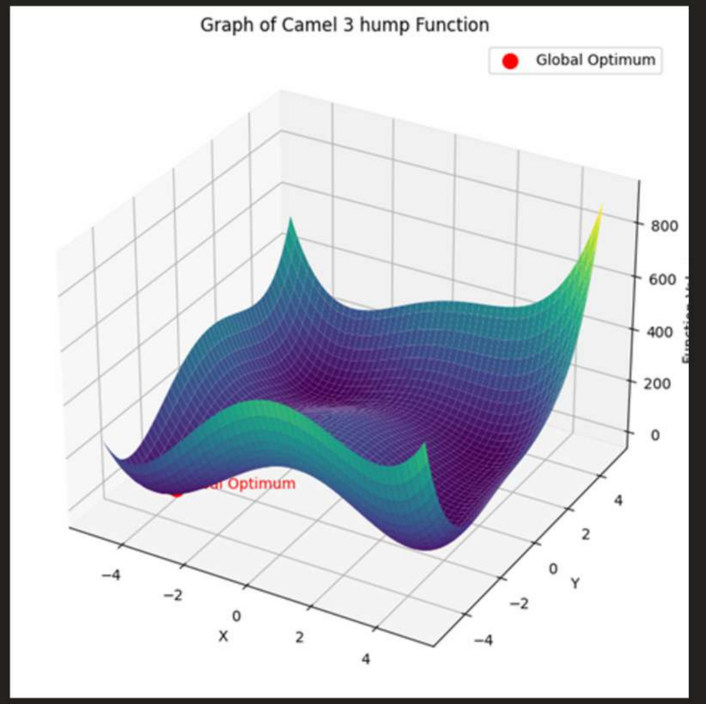
65. **Himmelblau Function** [45] (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

$$f_{65}(\mathbf{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

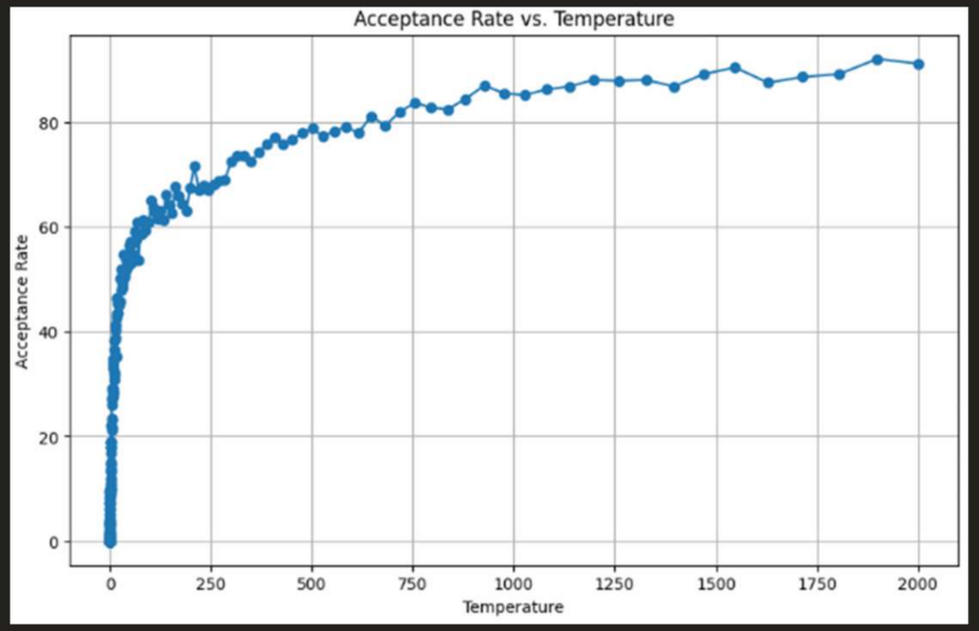
subject to $-5 \leq x_i \leq 5$. The global minimum is located at $\mathbf{x}^* = f(3, 2),\ f(\mathbf{x}^*) = 0.$

```
Best Solution: (-2.8047207833047683, 3.132321120159954)
Best Value: 4.657766700271174e-05
```





Initial Temperature 750K

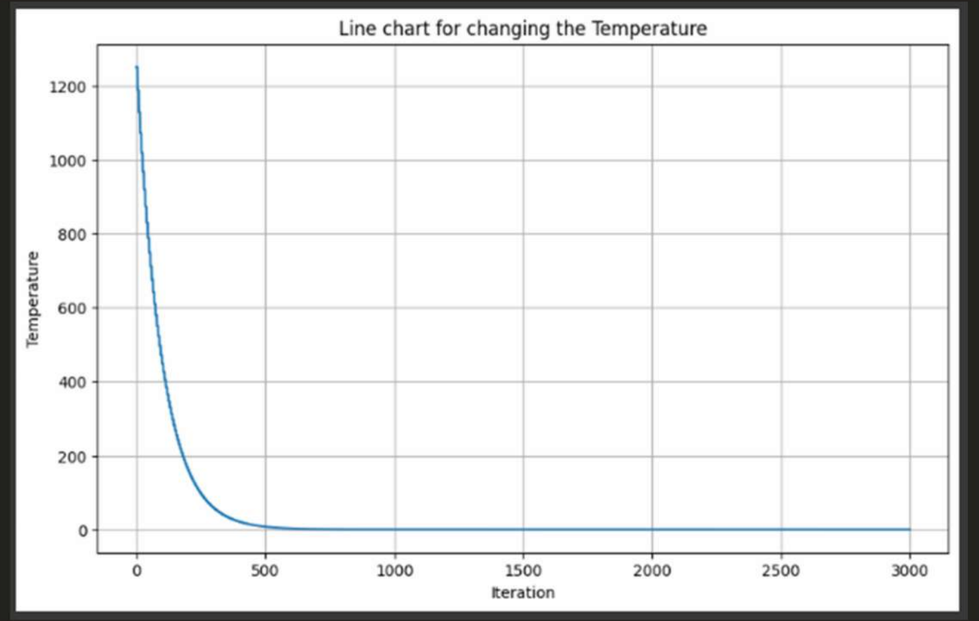


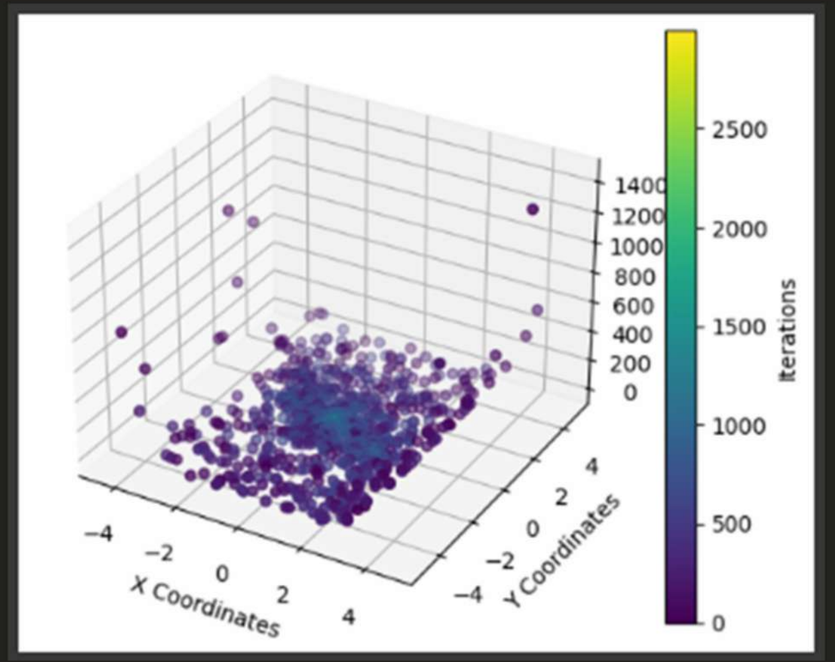
29. **Camel Function – Three Hump** [15] (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

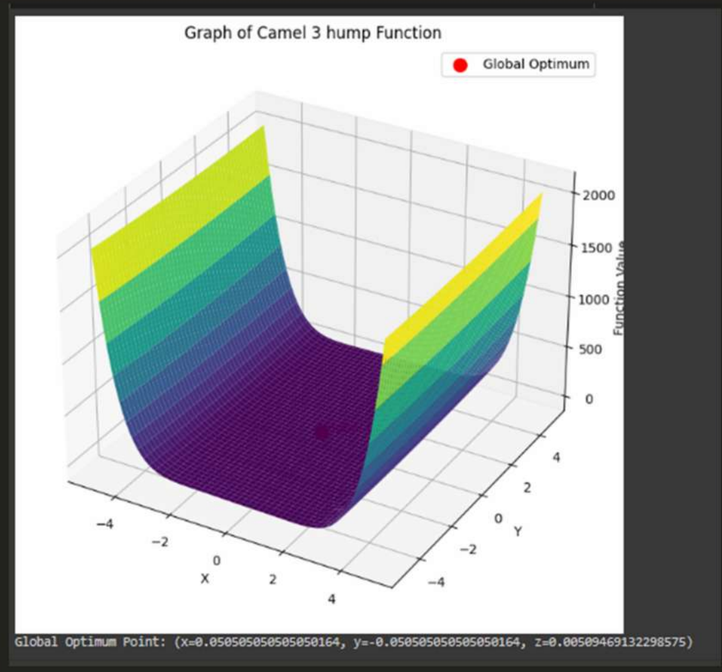
$$f_{29}(\mathbf{x}) = 2x_1^2 - 1.05x_1^4 + x_1^6/6 + x_1x_2 + x_2^2$$

subject to $-5 \leq x_i \leq 5$. The global minima is located at $\mathbf{x}^* = f(0,0)$, $f(\mathbf{x}^*) = 0$.

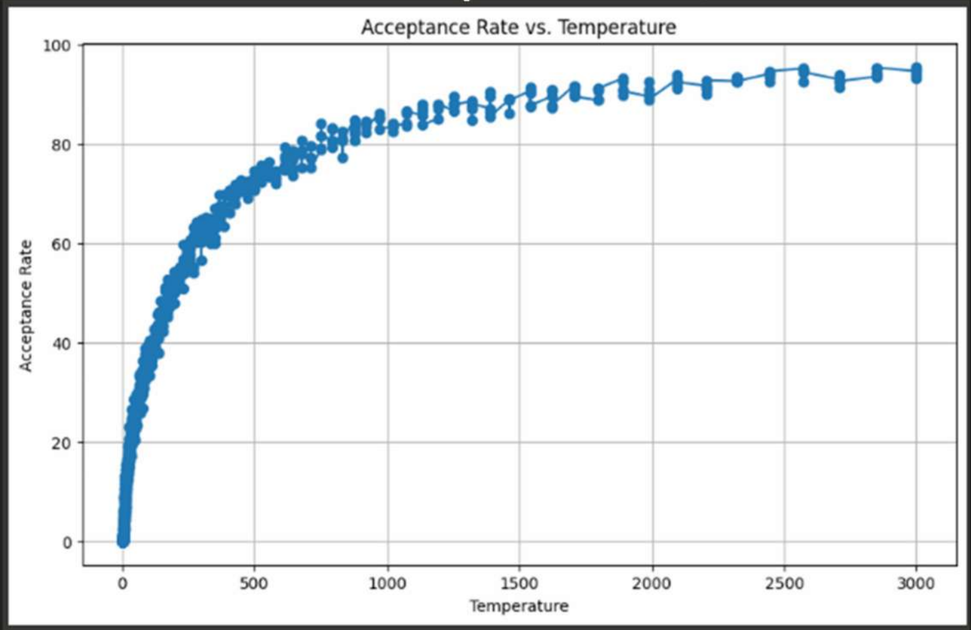
```
Best Solution: (0.0036871040689865353, -0.0007803392334686521)
Best Value: 2.492101613017011e-05
```

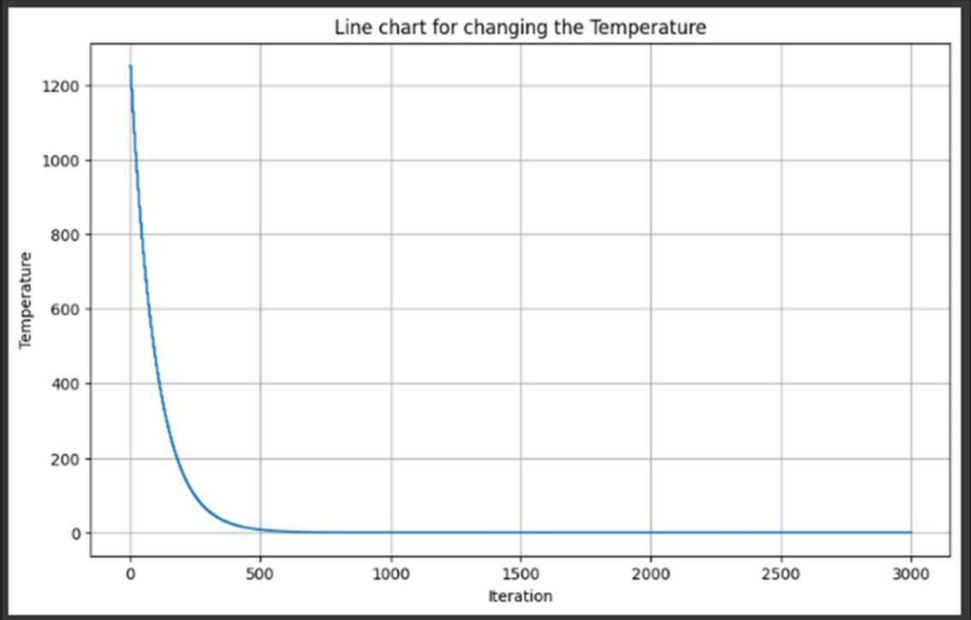






Initial Temperature 1000K



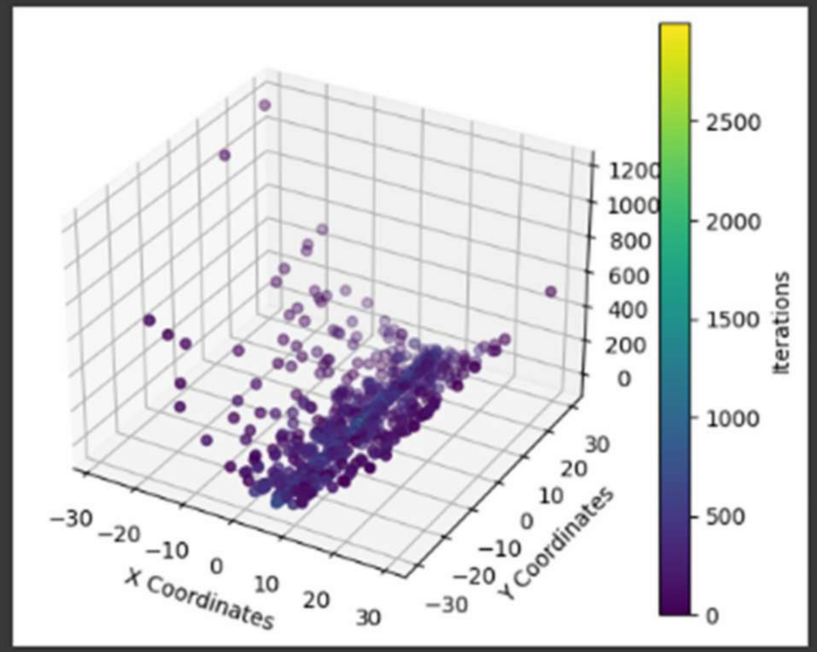


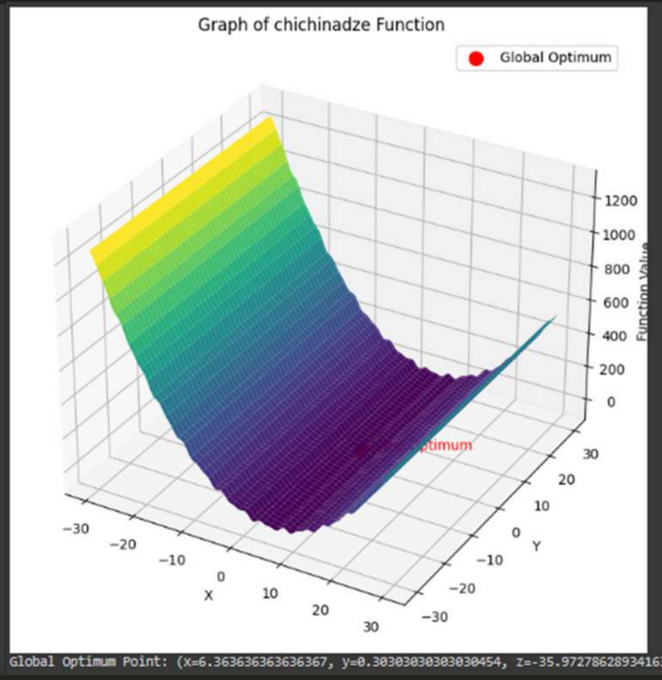
33. **Chichinadze Function** (Continuous, Differentiable, Separable, Non-Scalable, Multimodal)

$$f_{33}(\mathbf{x}) = x_1^2 - 12x_1 + 11 + 10\cos(\pi x_1/2) + 8\sin(5\pi x_1/2) - (1/5)^{0.5}\exp(-0.5(x_2 - 0.5)^2)$$

subject to $-30 \leq x_i \leq 30$. The global minimum is located at $\mathbf{x}^* = f(5.90133, 0.5)$, $f(\mathbf{x}^*) = -43.3159$.

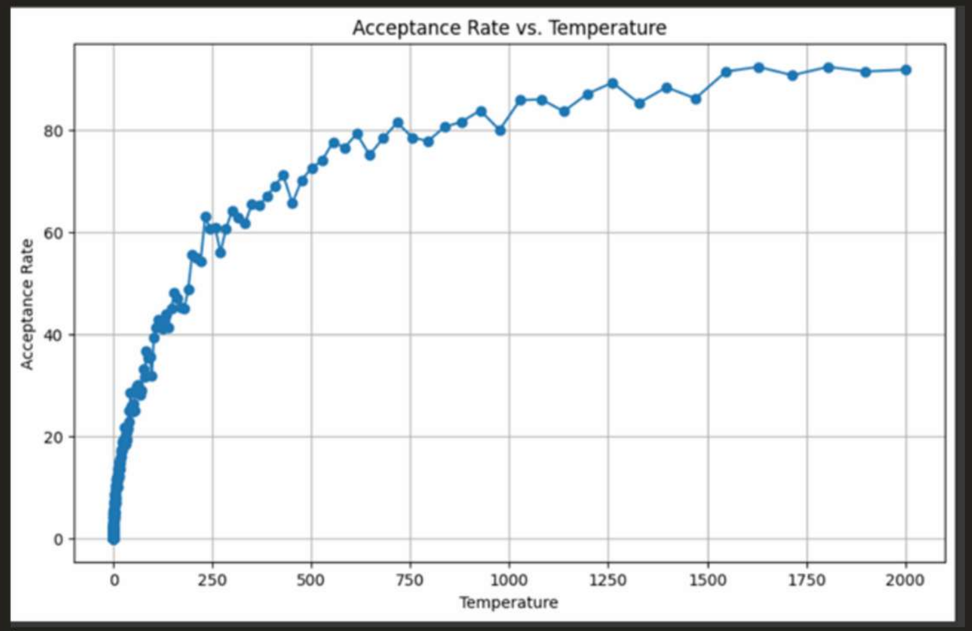
```
Best Solution: (6.189963719556019, 0.50226210791309)
Best Value: -42.94438343346988
```





Global Optimum Point: (x=6.363636363636367, y=0.30303030303030454, z=-35.972786289341634)

Initial Temperature 1500K

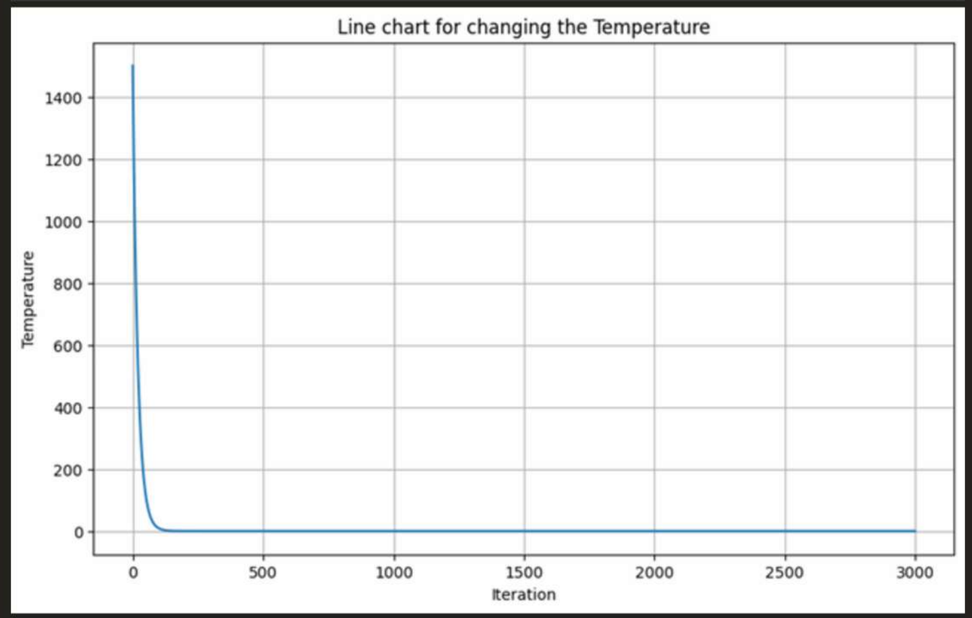


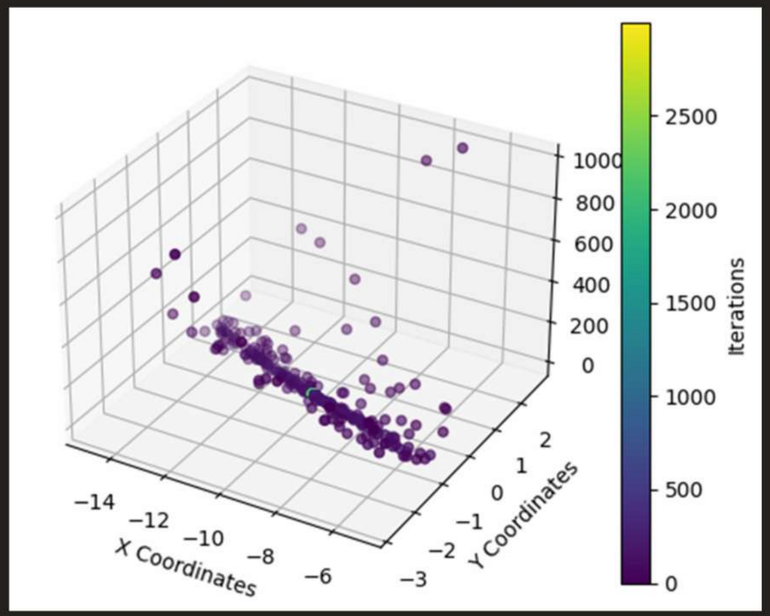
26. **Bukin 2 Function** (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

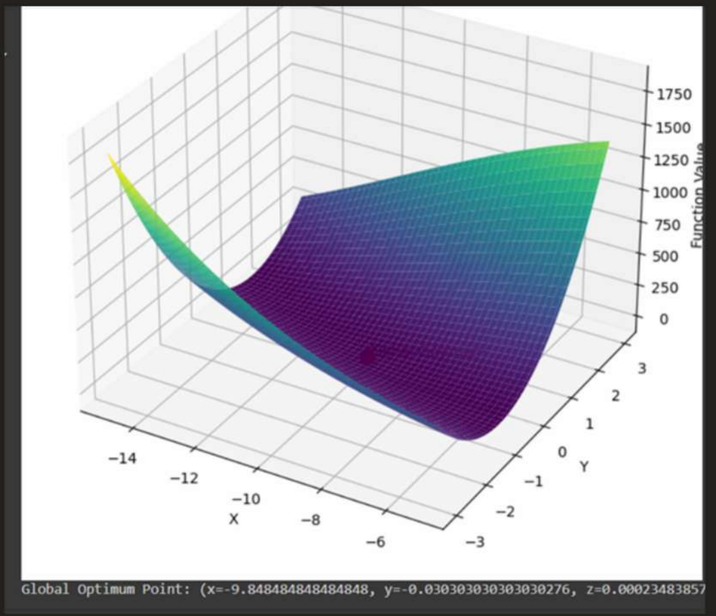
$$f_{26}(\mathbf{x}) = 100(x_2 - 0.01x_1^2 + 1) + 0.01(x_1 + 10)^2$$

subject to $-15 \leq x_1 \leq -5$ and $-3 \leq x_2 \leq -3$. The global minimum is located at $\mathbf{x}^* = f(-10, 0)$, $f(\mathbf{x}^*) = 0$.

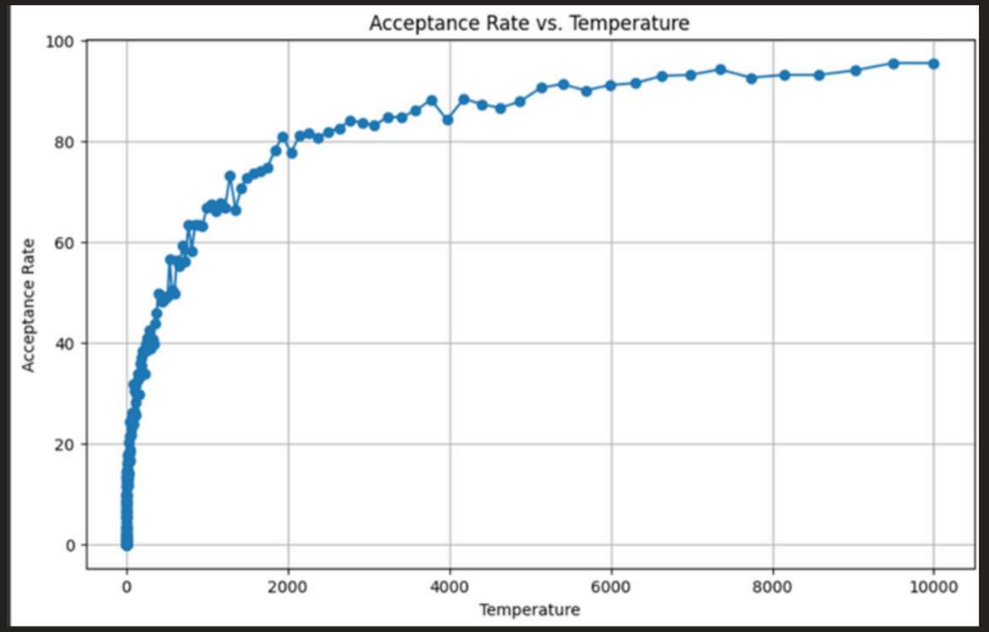
```
Best Solution: (-9.998639775775 77, -0.0004286857990427606)
Best Value: 1.0571547293113338e-06
```







Initial Temperature 200K

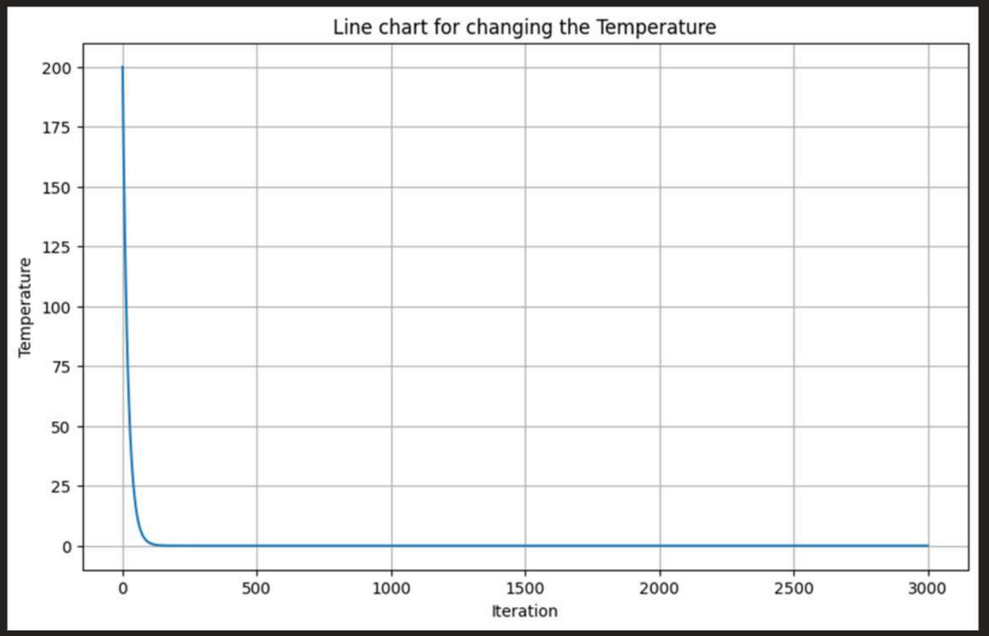


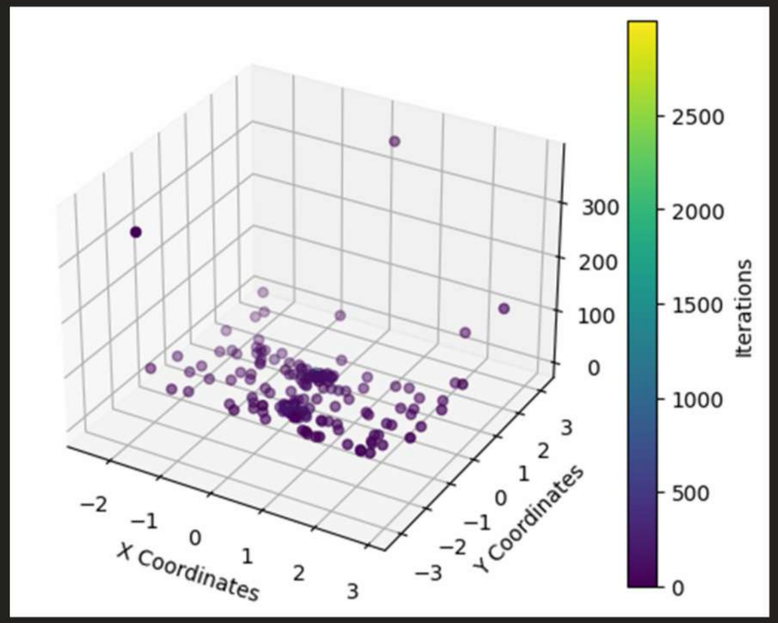
30. **Camel Function – Six Hump** [15] (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

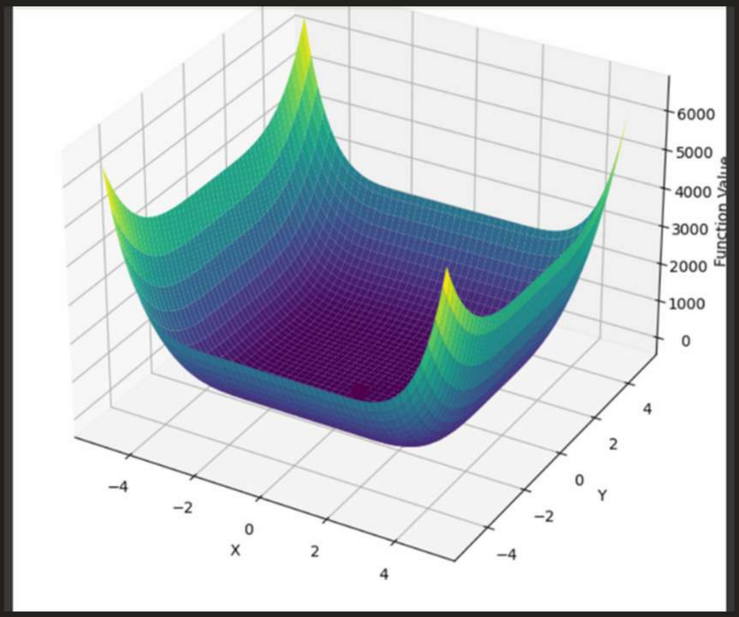
$$f_{30}(\mathbf{x}) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + (4x_2^2 - 4)x_2^2$$

subject to $-5 \leq x_i \leq 5$. The two global minima are located at $\mathbf{x}^* = f(\{-0.0898, 0.7126\}, \{0.0898, -0.7126, 0\})$, $f(\mathbf{x}^*) = -1.0316$.

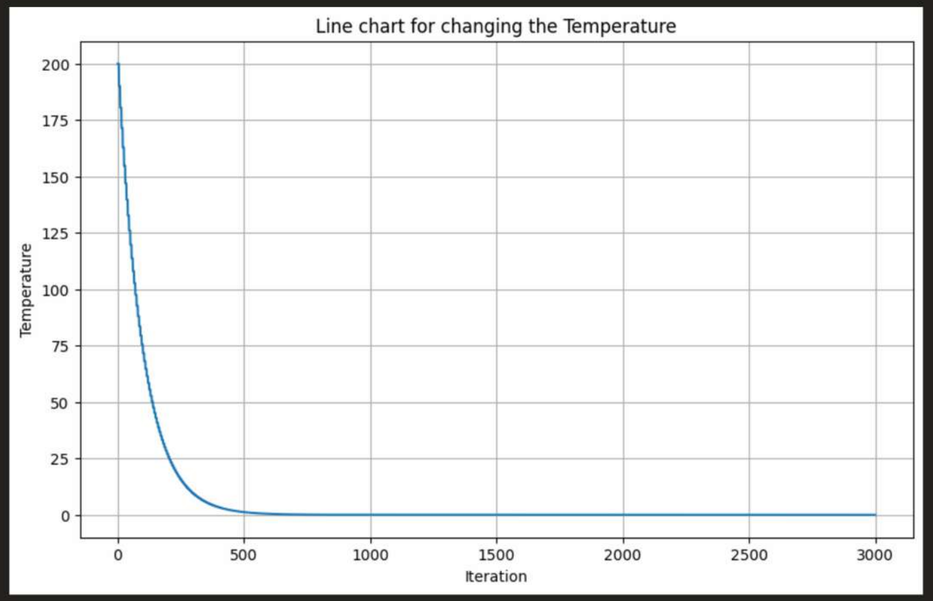
```
Best Solution: (0.092598925030134, -0.713080017286126)
Best Value: -1.0315985350582528
```

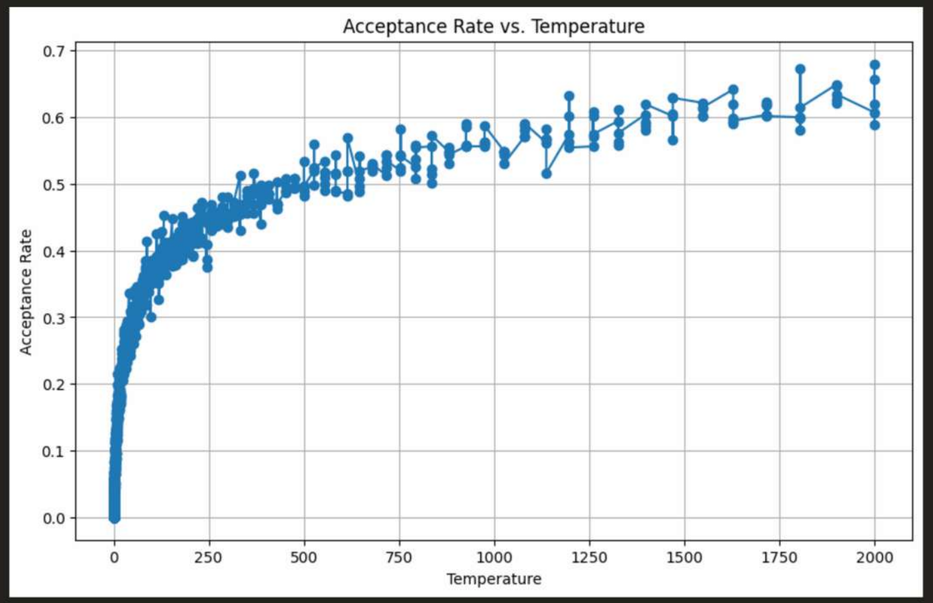






Initial Temperature 200K



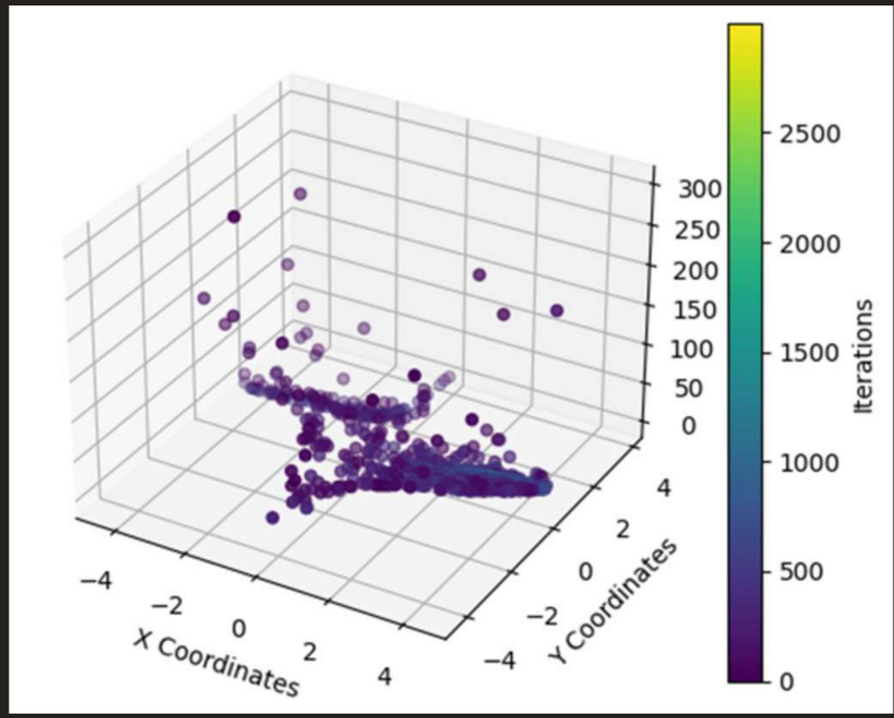


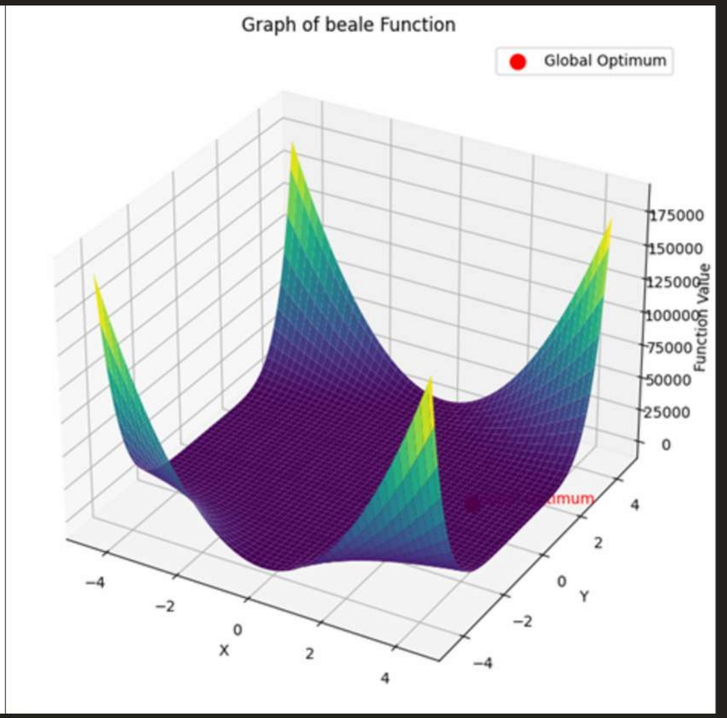
10. **Beale Function** (Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal)

$$f_{10}(\mathbf{x}) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$$

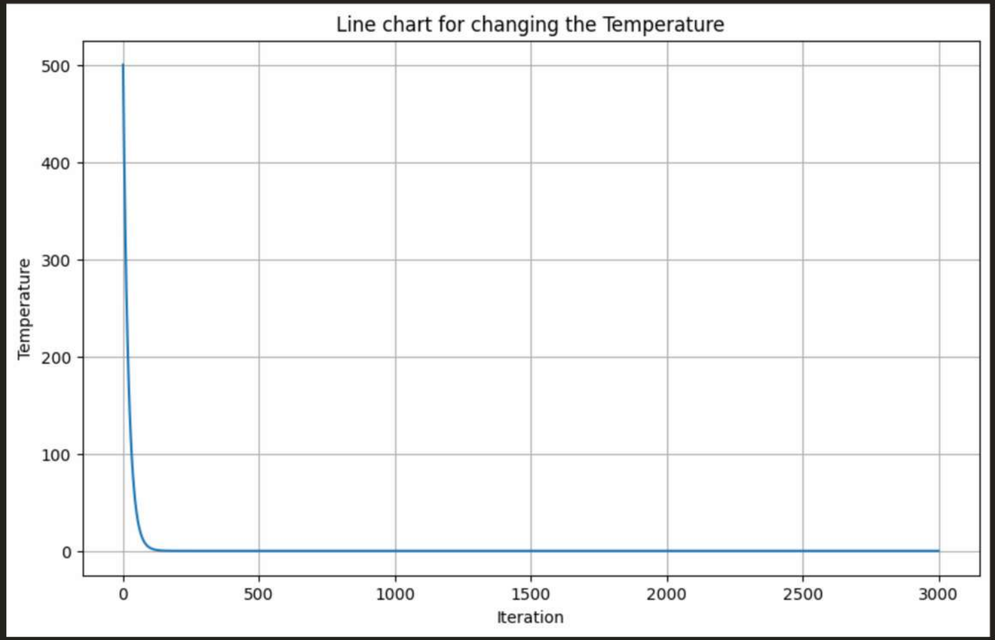
subject to $-4.5 \leq x_i \leq 4.5$. The global minimum is located at $\mathbf{x}^* = (3, 0.5)$, $f(\mathbf{x}^*) = 0$.

```
Best Solution: (3.023405588903123, 0.5046259372746942)
Best Value: 0.00011535832549013372
```





Initial Temperature 500K

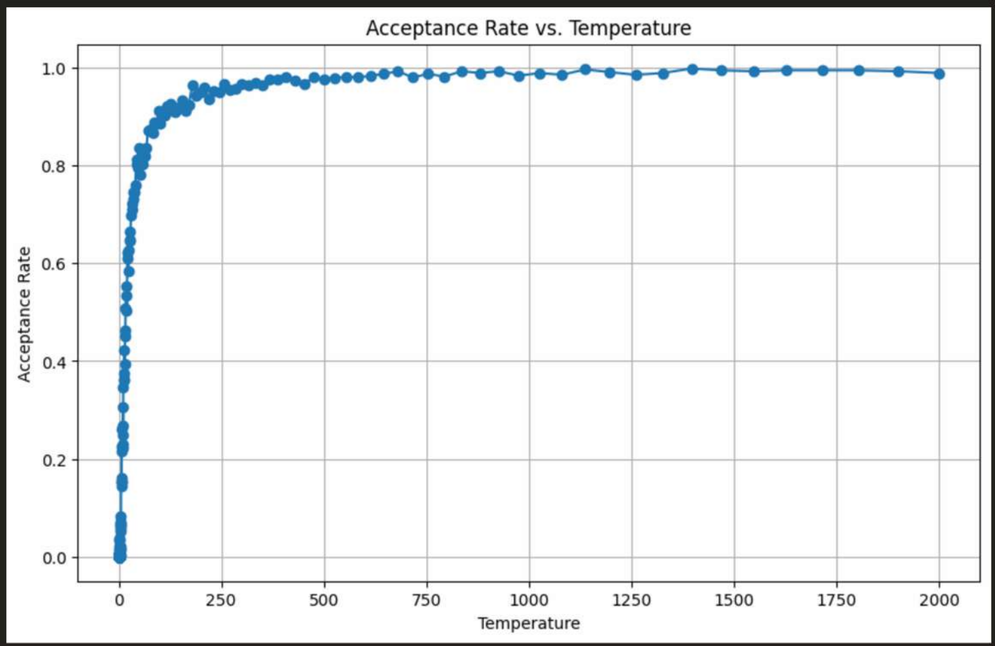


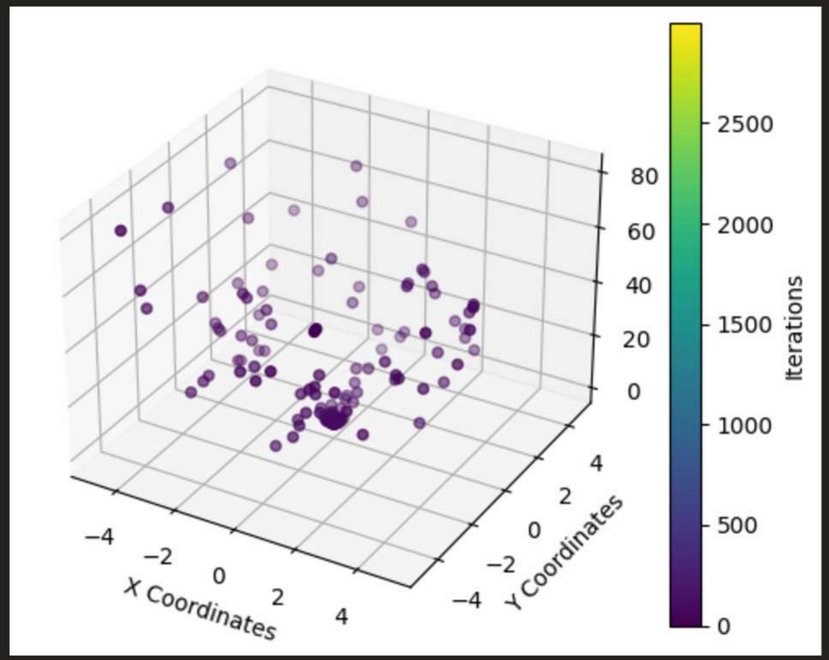
52. **Egg Crate Function** (Continuous, Separable, Non-Scalable)

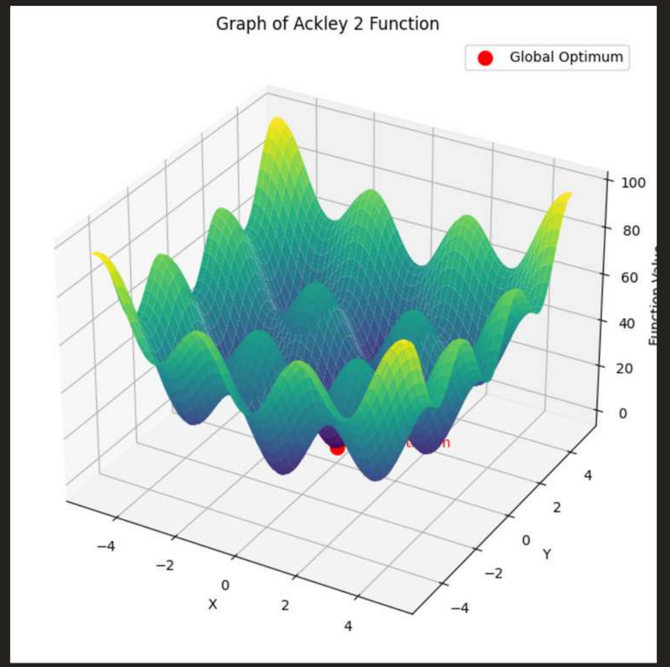
$$f_{52}(\mathbf{x}) = x_1^2 + x_2^2 + 25(\sin^2(x_1) + \sin^2(x_2))$$

subject to $-5 \leq x_i \leq 5$. The global minimum is located at $\mathbf{x}^* = f(0,0)$, $f(\mathbf{x}^*) = 0$.

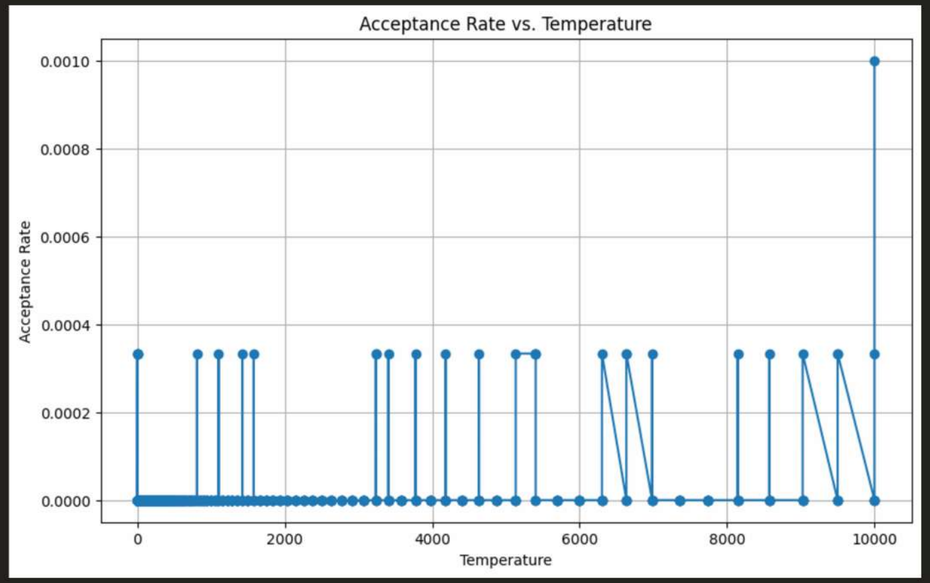
```
Best Solution: (0.0026990847474861 47, -0.0001056682154505495)
Best Value: 0.00018970138812467294
```

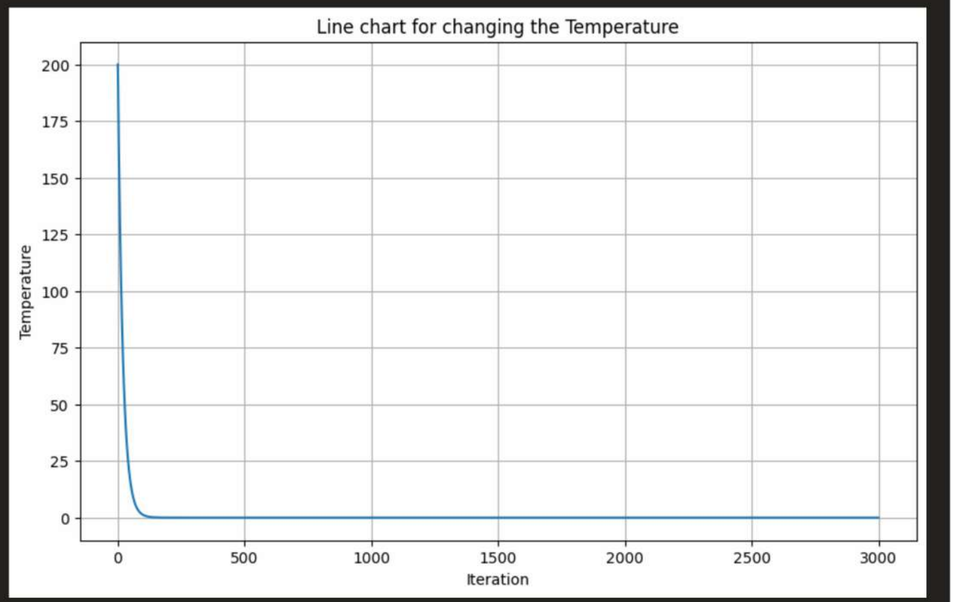






Initial Temperature 200K



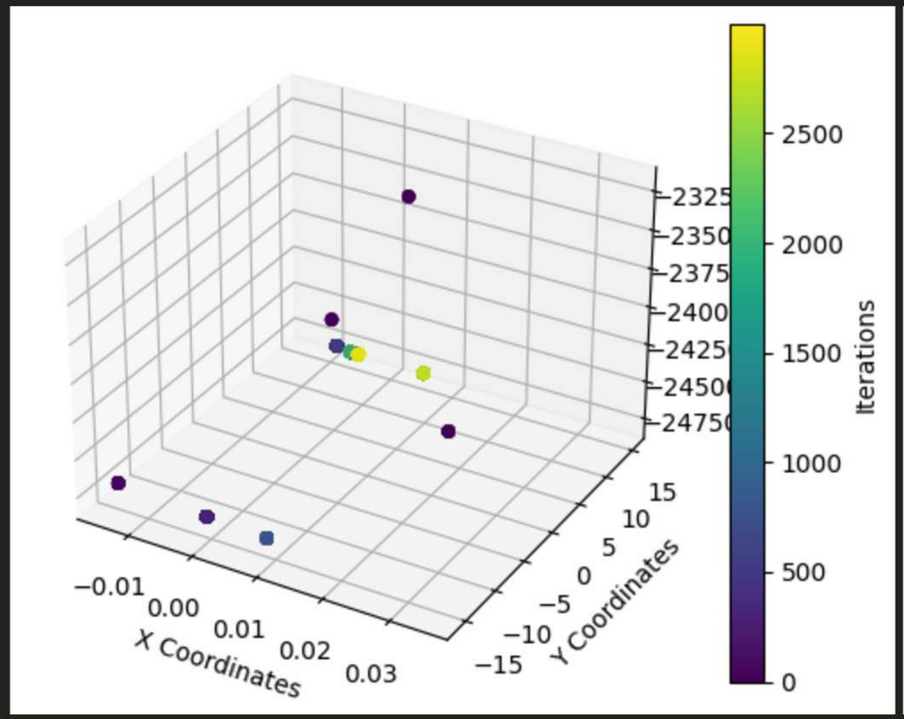


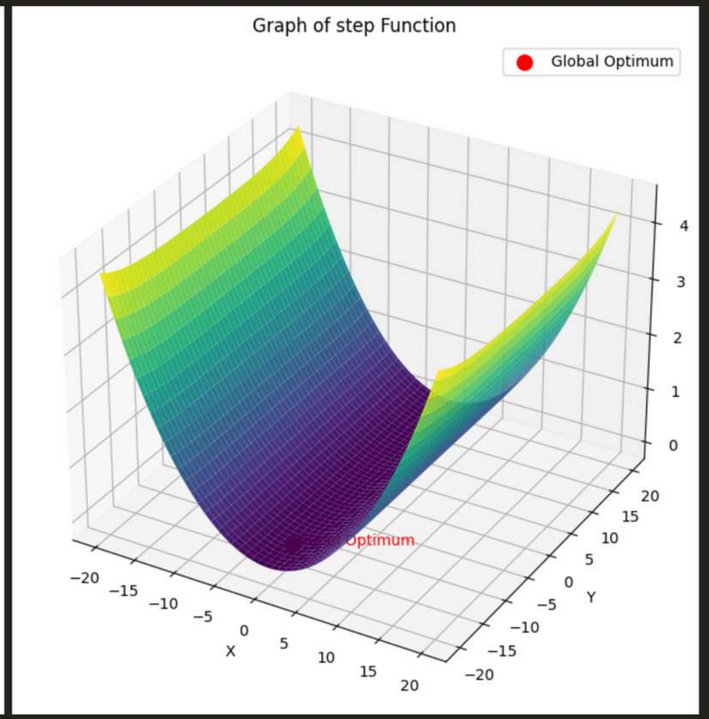
45. **Deckkers-Aarts Function** [4] (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

$$f_{45}(\mathbf{x}) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4$$

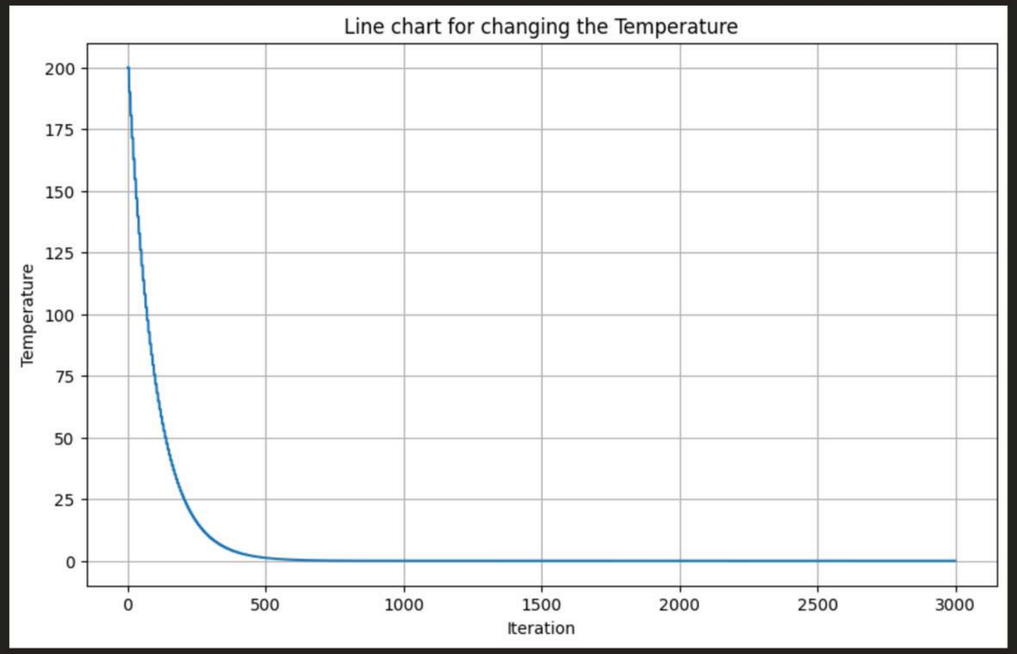
subject to $-20 \leq x_i \leq 20$. The two global minima are located at $\mathbf{x}^* = f(0, \pm 15)$ $f(\mathbf{x}^*) = -24777$.

```
Best Solution: (-0.005306801670712957, 14.947403060997303)
Best Value: -24773.692797709642
```





Initial Temperature 200K

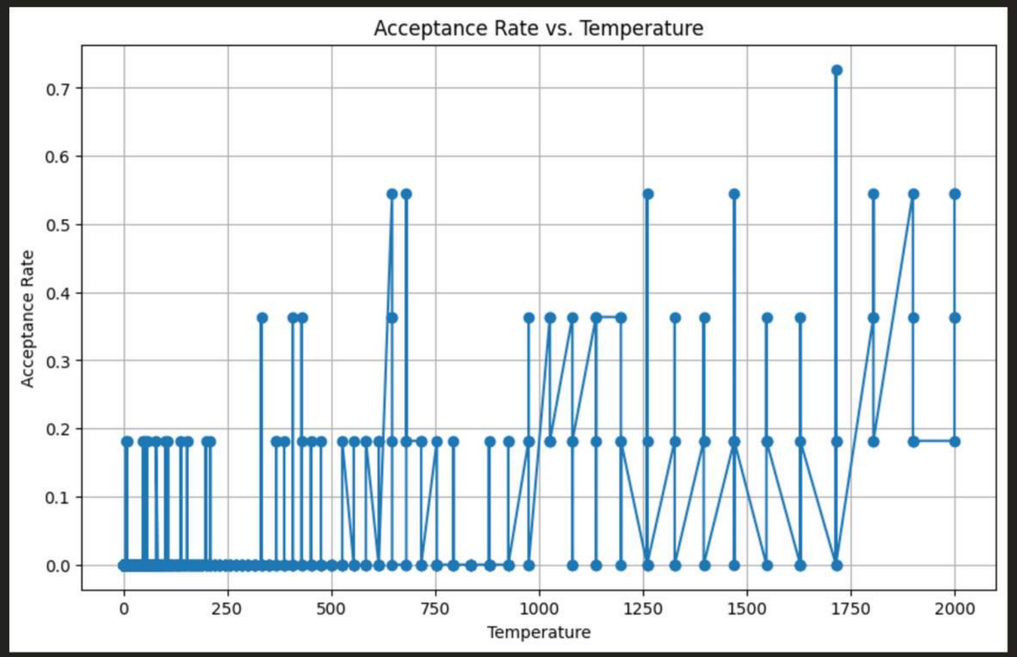


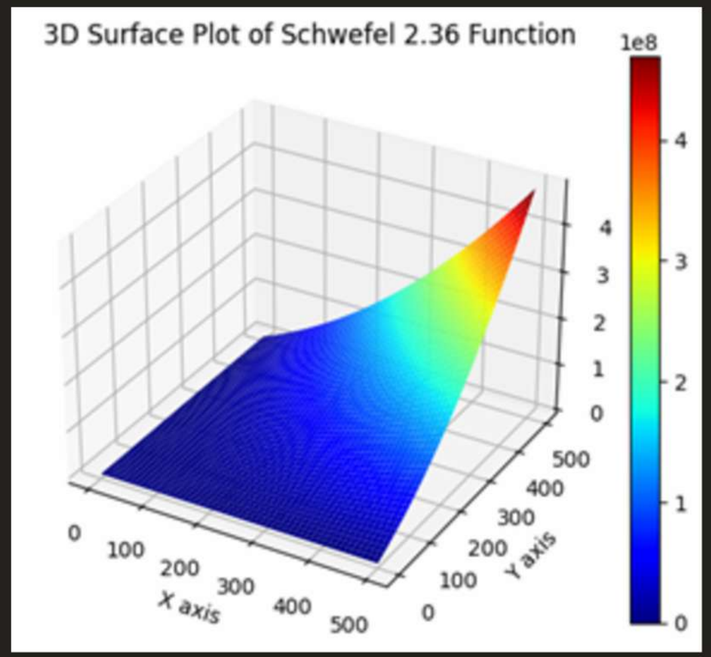
129. **Schwefel 2.36 Function** [77] (Continuous, Differentiable, Separable, Scalable, Multimodal)

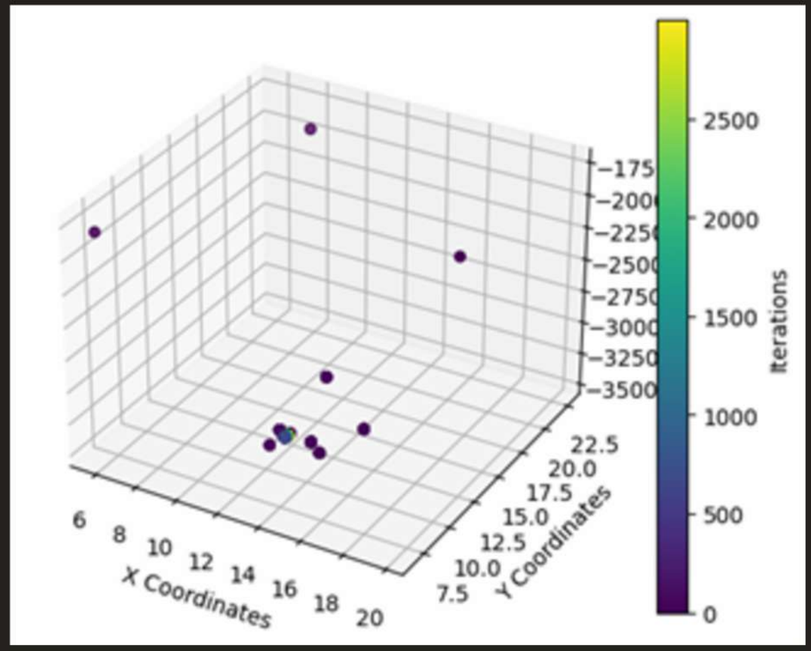
$$f_{129}(\mathbf{x}) = -x_1 x_2 (72 - 2x_1 - 2x_2)$$

subject to $0 \leq x_i \leq 500$. The global minimum is located at $\mathbf{x}^* = f(12, \cdots, 12)$, $f(\mathbf{x}^*) = -3456$.

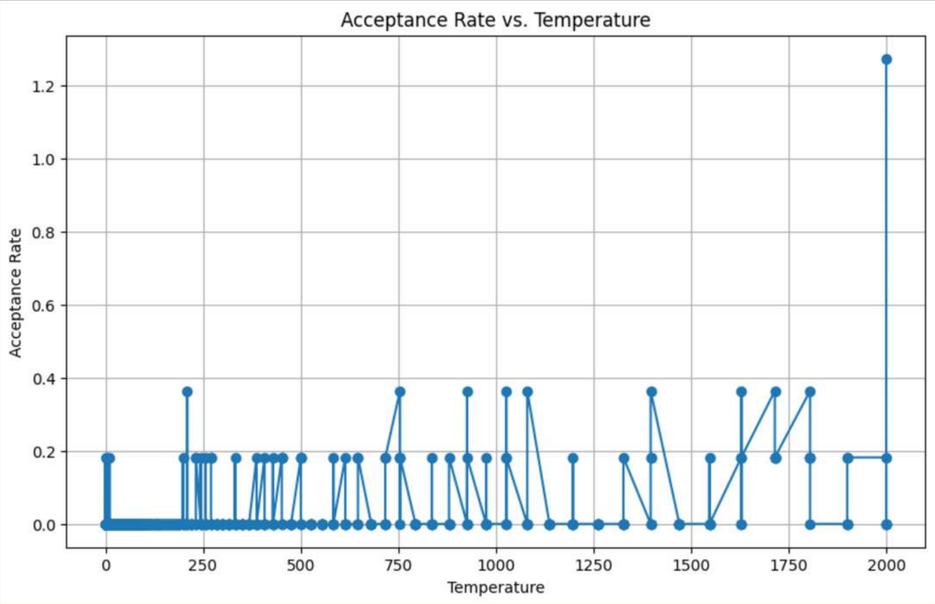
```
Best Solution: (12.007832576409417, 12.026455515647461)
Best Value: -3455.9767427881775
```







Initial Temperature 200K

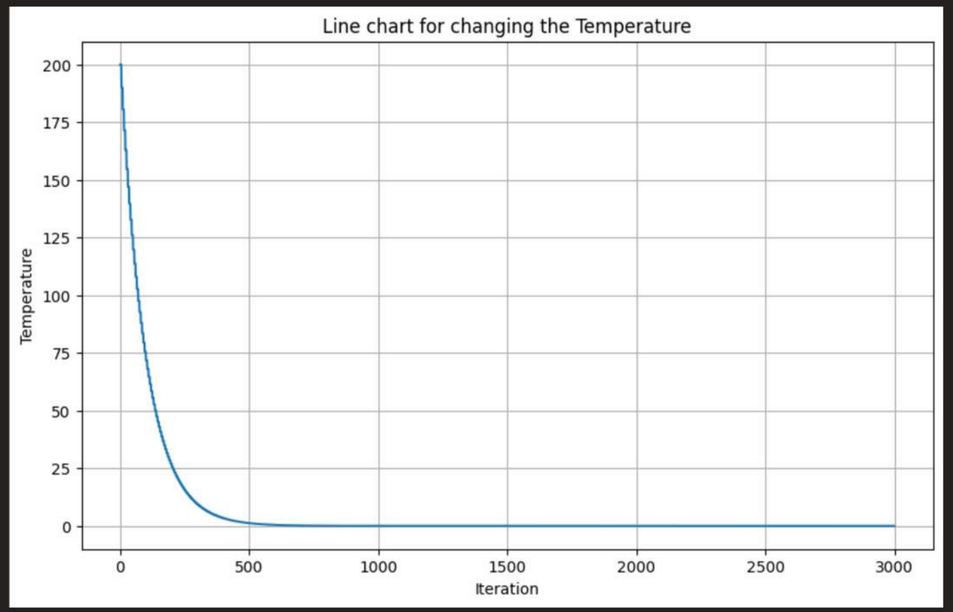


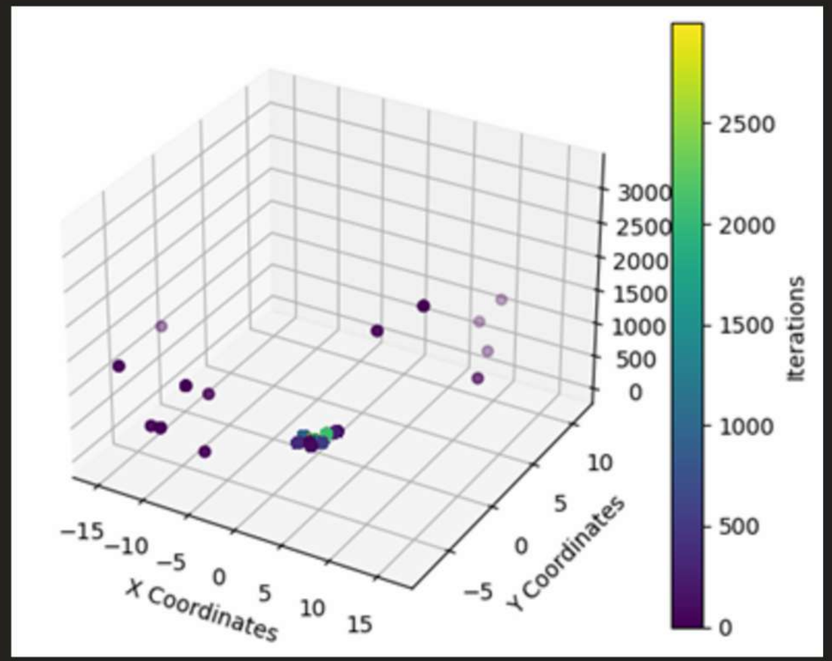
107. **Rotated Ellipse Function** (Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal)

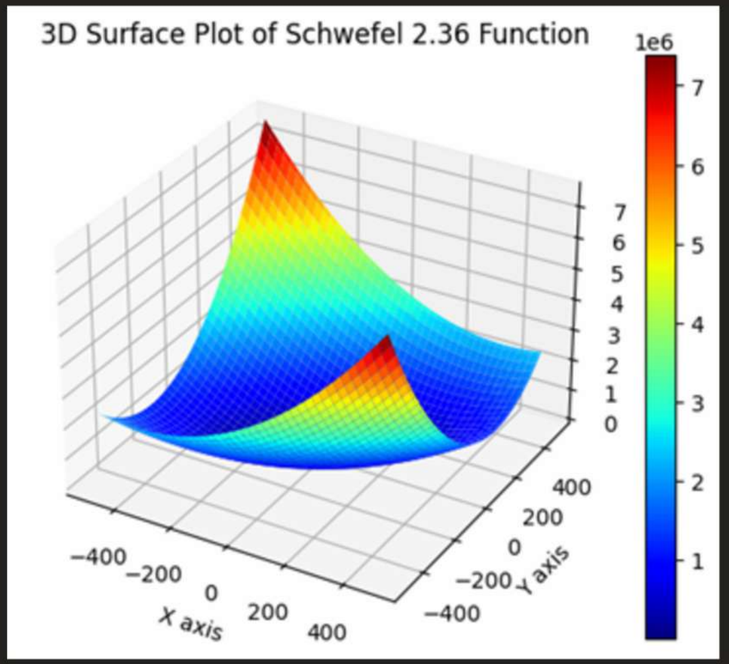
$$f_{107}(\mathbf{x}) = 7x_1^2 - 6\sqrt{3}x_1x_2 + 13x_2^2$$

subject to $-500 \leq x_i \leq 500$. The global minimum is located at $\mathbf{x}^* = f(0,0)$, $f(\mathbf{x}^*) = 0$.

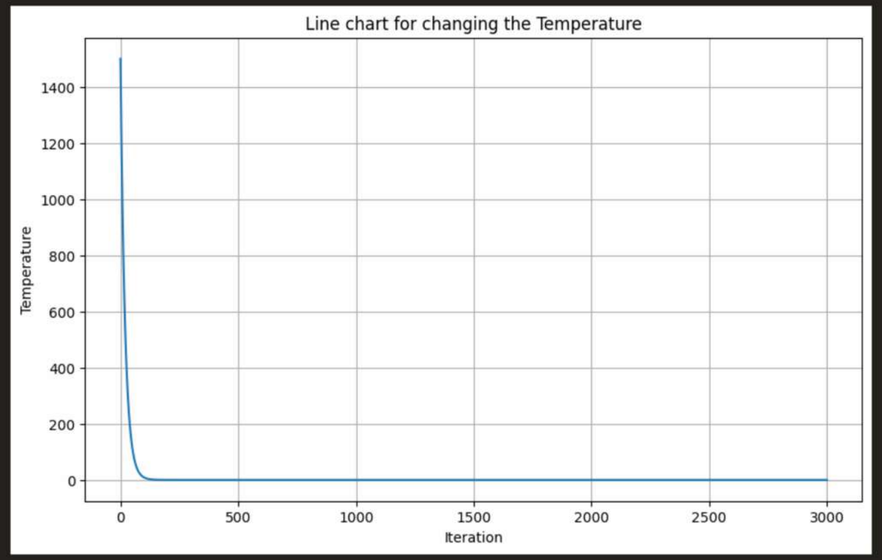
```
Best Solution: (0.17406384294139343, 0.36974782842389686)
Best Value: 1.3205165735858735
```







Initial Temperature 1500K

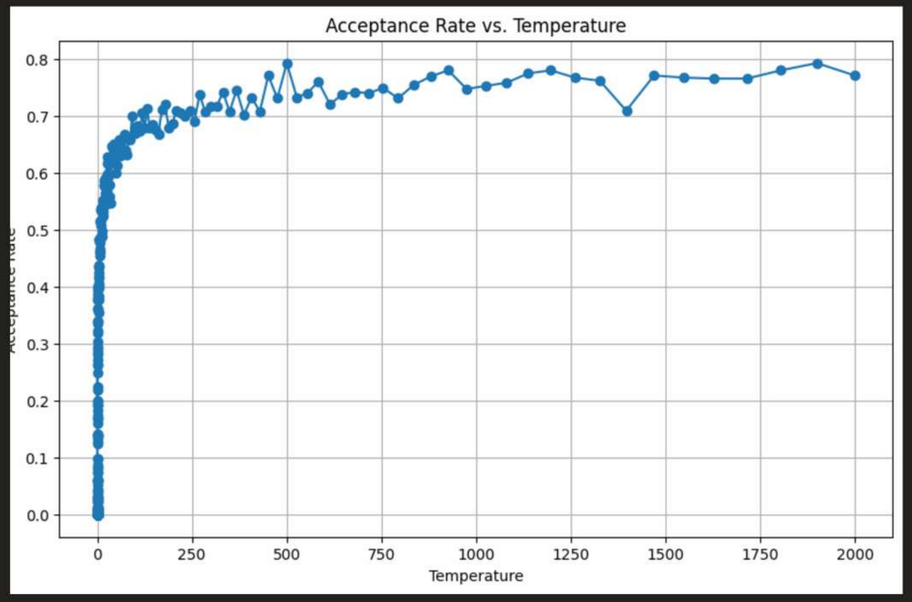


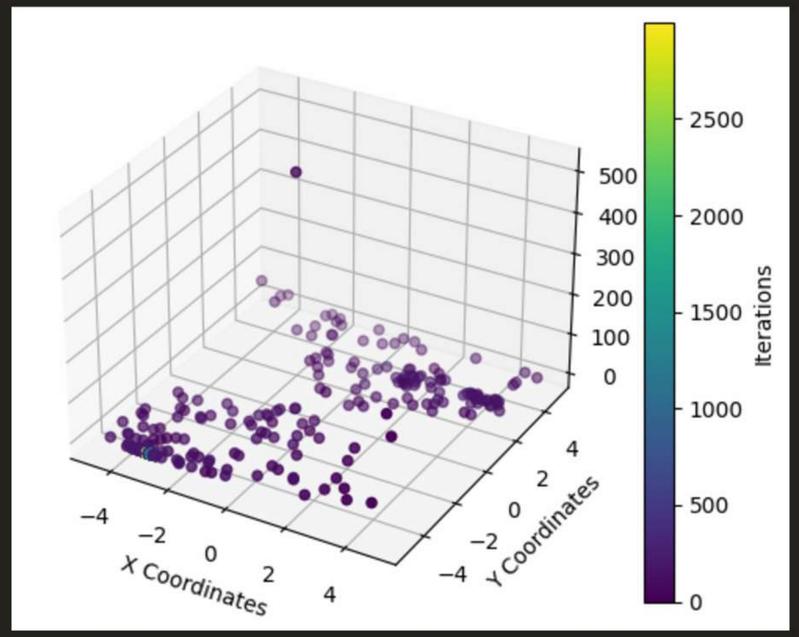
116. **Schmidt Vetters Function** [50] (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal)

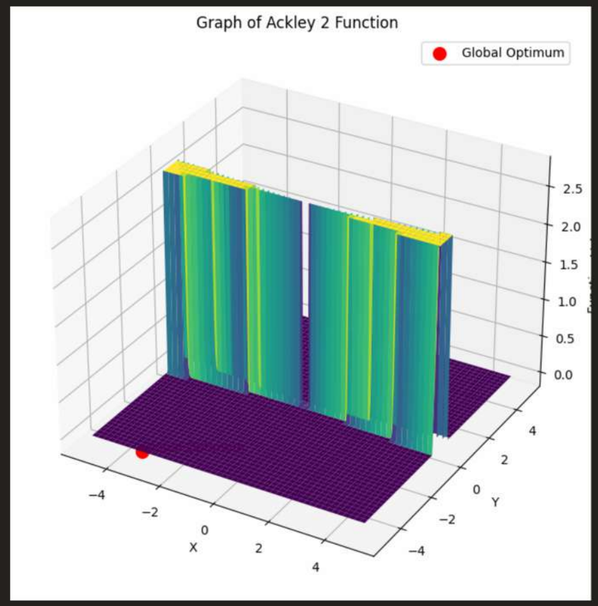
$$f_{116}(\mathbf{x}) = \frac{1}{1+(x_1-x_2)^2} + \sin\left(\frac{\pi x_2 + x_3}{2}\right) + e^{\left(\frac{x_1+x_2}{x_2}-2\right)^2}$$

The global minimum is located at $\mathbf{x}^* = f(0.78547, 0.78547, 0.78547),\ f(\mathbf{x}^*) = 3.$

```
Best Solution: (-3.138984808066697, -4.999970442627864)
Best Value: 0.4487899086640892
```







```python
import numpy as np
import random
import math
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde

# Arrays for storing temperatures, function values, iterations, and states
temperatures = []
function_values = []
iterations = []
states = []

# Initial parameters
L_0 = 550 # Fixed Markov chain length

# Define the Ackley 2 function
def ackley2(x, y):
    return -200 * math.exp(-0.02 * math.sqrt(x**2 + y**2))
# Define the camel_three_hump function
def camel_three_hump(x1, x2):

    term1 = x1**2
    term2 = x2**2
    term3 = -2*x1*x2
    return term1 + term2 + term3
# Define the chichinadze_function Function function
def chichinadze_function(x, y):
    term1 = x**2 - 12 * x + 11
    term2 = 10 * np.cos(np.pi * x / 2)
    term3 = 8 * np.sin(5 * np.pi * x / 2)
    term4 = ((1 / 5)**0.5) * (np.exp(-0.5 * (y - 0.5)**2))
    return term1 + term2 + term3 - term4
```

```python
def simulate(temp, func, bounds):
    current_x = random.uniform(bounds[0], bounds[1])
    current_y = random.uniform(bounds[0], bounds[1])
    best_solution = (current_x, current_y)
    best_value = func(current_x, current_y)
    L_i = L_0

    data_dimensions = 3
    simulation_data = np.empty((3000, L_0, data_dimensions))

    for i in range(3000):
        for j in range(L_i):
            new_x = random.uniform(bounds[0], bounds[1])
            new_y = random.uniform(bounds[0], bounds[1])
            delta_E = func(new_x, new_y) - func(current_x, current_y)

            if delta_E <= 0 or random.uniform(0, 1) < math.exp(-delta_E / temp):
                current_x = new_x
                current_y = new_y



            simulation_data[i, j, 0] = temp
            simulation_data[i, j, 1] = j
            simulation_data[i, j, 2] = func(current_x, current_y)

        best_solution = (current_x,current_y)
        best_value = func(current_x,current_y)

        states.append(best_solution)
        function_values.append(best_value)
        temperatures.append(temp)
        iterations.append(i)

        #earlier schedule
        #temp = temp*0.95

        # Decrease temperature once every 5 iterations
        if (i + 1) % 5 == 0:
          delta_T = temp * 0.05
          temp -= delta_T
```

```python
[ ]  # Extract x and y coordinates from the states array
     x_coords = [state[0] for state in states]
     y_coords = [state[1] for state in states]

     # Extract the function values
     z_coords = function_values

     # Extract the iterations
     iterations_array = iterations

     # Create a 3D scatter plot
     fig = plt.figure()
     ax = fig.add_subplot(111, projection='3d')

     # Plot the points with color scheme based on iterations
     sc = ax.scatter(x_coords, y_coords, z_coords, c=iterations_array, cmap='viridis')

     # Add color bar to show iteration color scheme
     plt.colorbar(sc, label='Iterations')

     # Set labels
     ax.set_xlabel('X Coordinates')
     ax.set_ylabel('Y Coordinates')
     ax.set_zlabel('Function Value (z)')

     # Show the plot
     plt.show()
```

```python
    return best_solution, best_value, states, function_values, temperatures, iterations, simulation_data
```

```python
[ ]  ## Example usage
     temp =1000 # Initial temperature
     bounds = (-32, 32)  # Bounds for the Ackley function
     best_solution, best_value, states, function_values, temperatures, iterations, simulation_data = simulate(temp, ackley2, bounds)

     print(f"Best Solution: {best_solution}")
     print(f"Best Value: {best_value}")
```

Code for the Final Algorithm

Code for the Final Algorithm

```python
# Define the Ackley 2 function
def ackley2(x, y):
    return -200 * np.exp(-0.02 * np.sqrt(x**2 + y**2))
def camel_three_hump(x1, x2):

    term1 = 2 * x1**2
    term2 = -1.05 * x1**4
    term3 = x1**6 / 6
    term4 = x1 * x2
    term5 = x2**2

    return term1 + term2 + term3 + term4 + term5
def chichinadze_function(x, y):
    term1 = x**2 - 12 * x + 11
    term2 = 10 * np.cos(np.pi * x / 2)
    term3 = 8 * np.sin(5 * np.pi * x / 2)
    term4 = ((1 / 5)**0.5) * (np.exp(-0.5 * (y - 0.5)**2))
    return term1 + term2 + term3 - term4

# Generate data points
x = np.linspace(-30, 30, 100)
y = np.linspace(-30, 30, 100)
X, Y = np.meshgrid(x, y)
Z = chichinadze_function(X, Y)

# Find global optimum point
global_optimum = np.unravel_index(np.argmin(Z), Z.shape)
global_x = X[global_optimum]
global_y = Y[global_optimum]
global_z = Z[global_optimum]

# Plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

# Annotate global optimum point
ax.scatter(global_x, global_y, global_z, color='red', s=100, label='Global Optimum')
ax.text(global_x, global_y, global_z, 'Global Optimum', color='red')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Function Value')
ax.set_title('Graph of chichinadze Function')

plt.legend()
plt.show()

print("Global Optimum Point: (x={}, y={}, z={})".format(global_x, global_y, global_z))
```