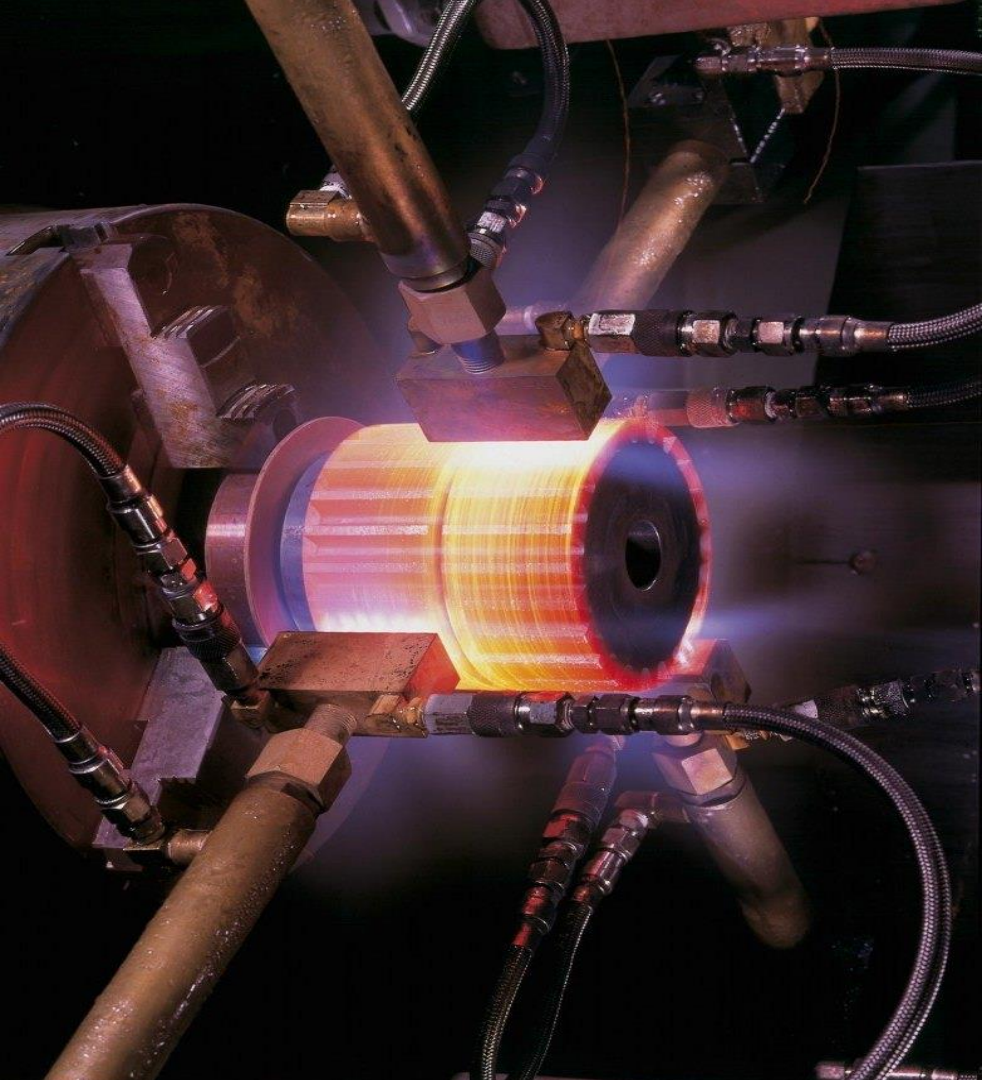


Development & Validation of Simulated Annealing Algorithm

Group 05

Group Members

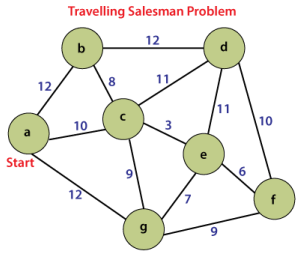
- S/18/813 – Tharindu Kariyawasam
- S/18/814 – Nipuni Kaushalya
- S/18/819 – Sanduni Ranasinghe
- S/18/822 – Ranali Piyatilake
- S/18/826 – Niroshan Sampath
- S/18/827 – Kumudumali Sandaleka
- S/18/836 – Chamod Wijesingha

Introduction to Simulated Annealing

Simulated Annealing is a probabilistic optimization technique inspired by the annealing process in metallurgy. It is used to find an approximate global optimum in a large and complex search space.

Why SA is Used for Optimization?

- **Versatility**: Effective for large-scale and complex optimization problems.
- **Global Search Capability**: Can escape local optima by accepting worse solutions with a certain probability.
- **Applicability**: Useful in various fields such as operations research, artificial intelligence, and engineering.



Advantages of SA

- **Global Optimization**: Capable of finding a global optimum in a complex landscape.
- **No Gradient Requirement**: Does not need derivative information, making it suitable for non-differentiable functions.
- **Flexibility**: Can be adapted to different types of problems and constraints.

Disadvantages of SA

- **Computational Intensity**: Requires many iterations, which can be time-consuming.
- **Parameter Sensitivity**: Performance heavily depends on parameters like initial temperature and cooling schedule.
- **Convergence Issues**: May require fine-tuning to balance exploration and exploitation.

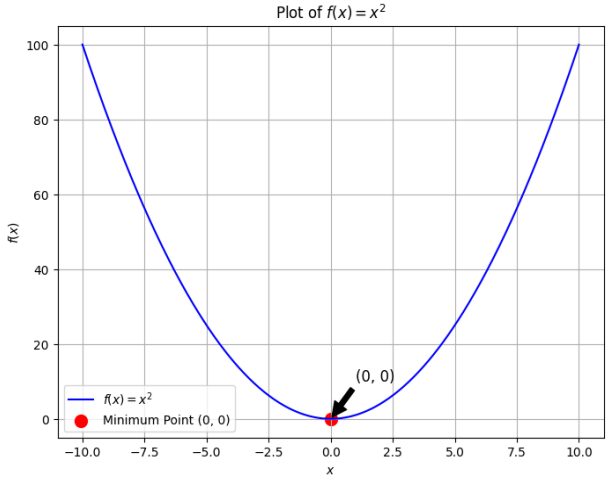
Pseudocode for the Metropolis Criterion

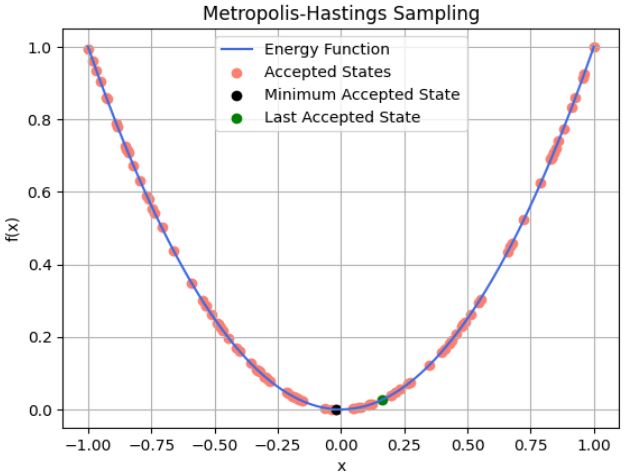
- Repeat
  - Perturbation: State i to state j
  - If $\Delta E_{ij} \leq 0$ then accept state j

    Else if $\exp\left(-\frac{\Delta E_{ij}}{c}\right) > \text{random}[0,1)$ then accept
  - If accept, update the state i with state j
- Until equilibrium is approached

c is the temperature

Graph for the Metropolis Criterion (One variable)



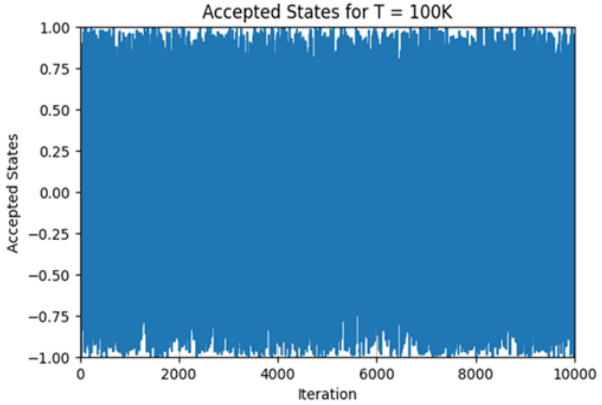


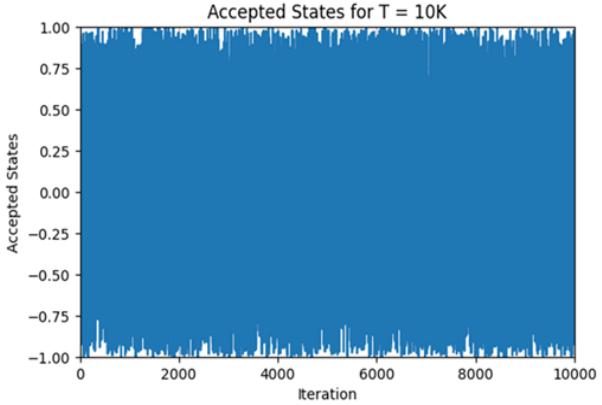
$$Enegy\ Function: f(x) = x^2$$

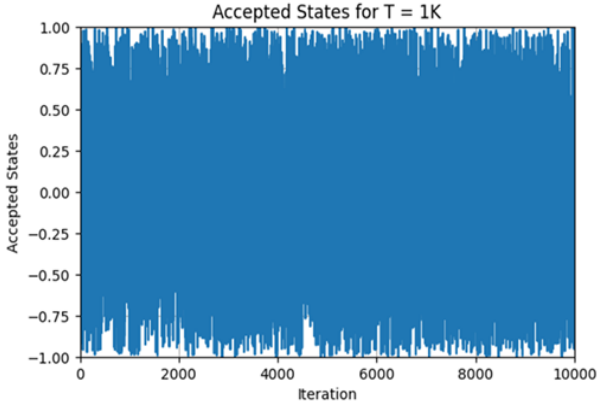
Minimum value of the accepted states list: -0.01913758287274714

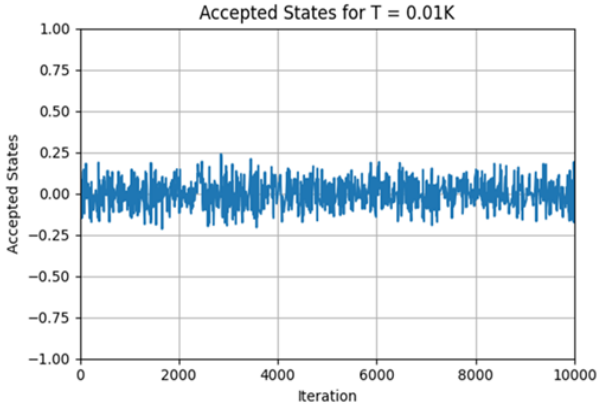
Last value of the accepted states list: 0.16464892860924252

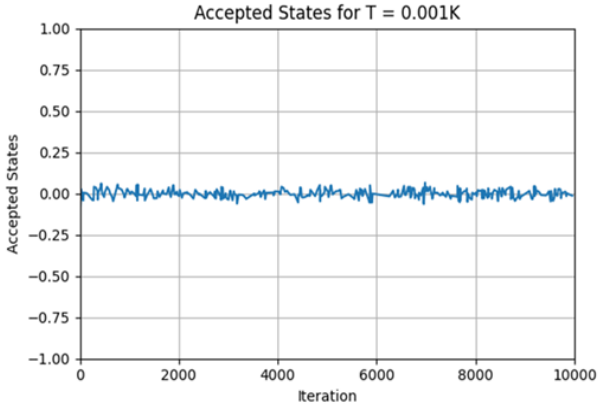
X-chains (Changing the initial temperature)

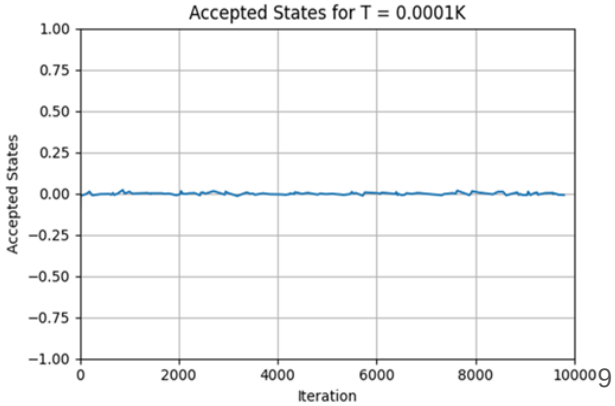








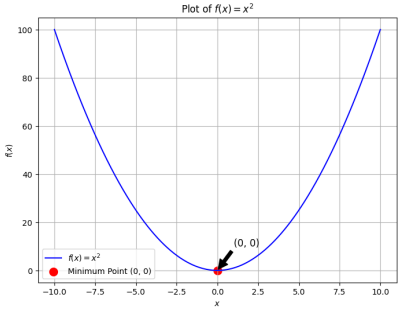




Introducing a Linear Cooling Schedule

Temperature Value = Temperature Value - Temperature Value * k

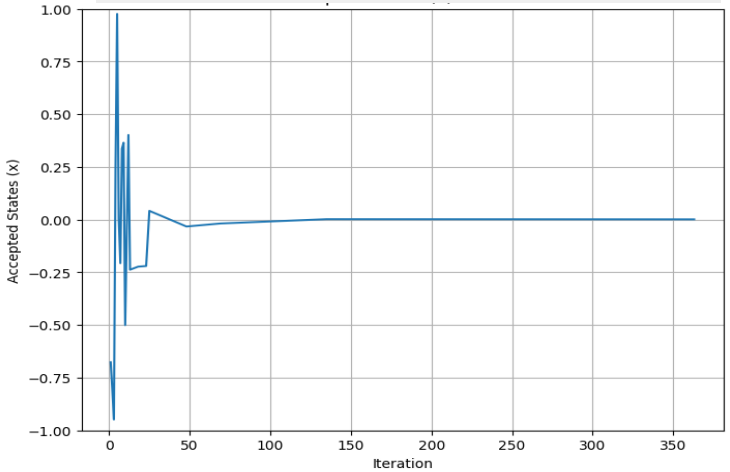
$k = cooling\ factor$



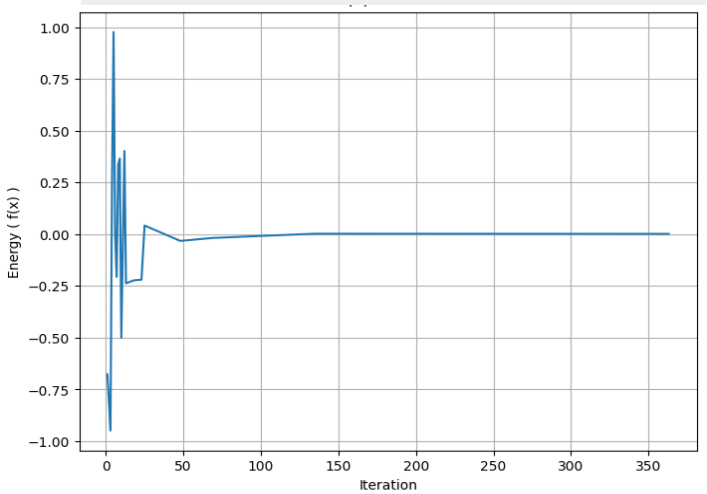
$f(x) = x^2$

- Setting temperature as a control parameter
- Temperature decreased by 50% at each iteration

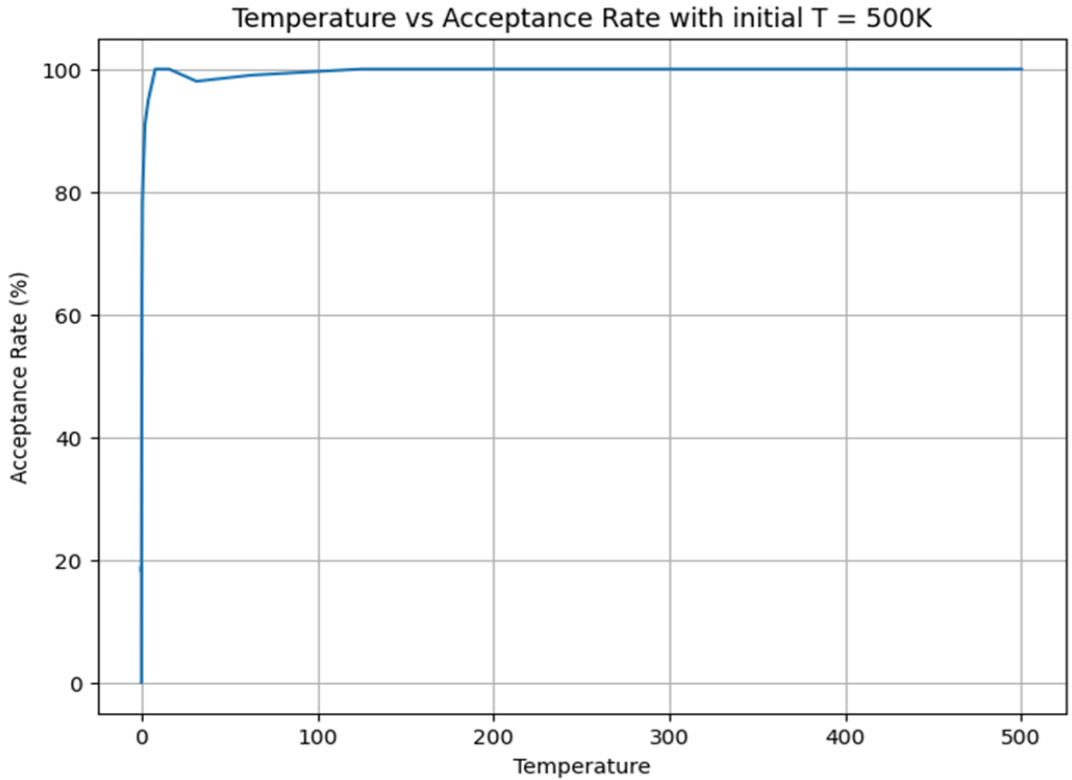
Iteration vs x values plot
(Initial temperature = 5000K)



Iteration vs f(x) values plot
(Initial temperature = 5000K)

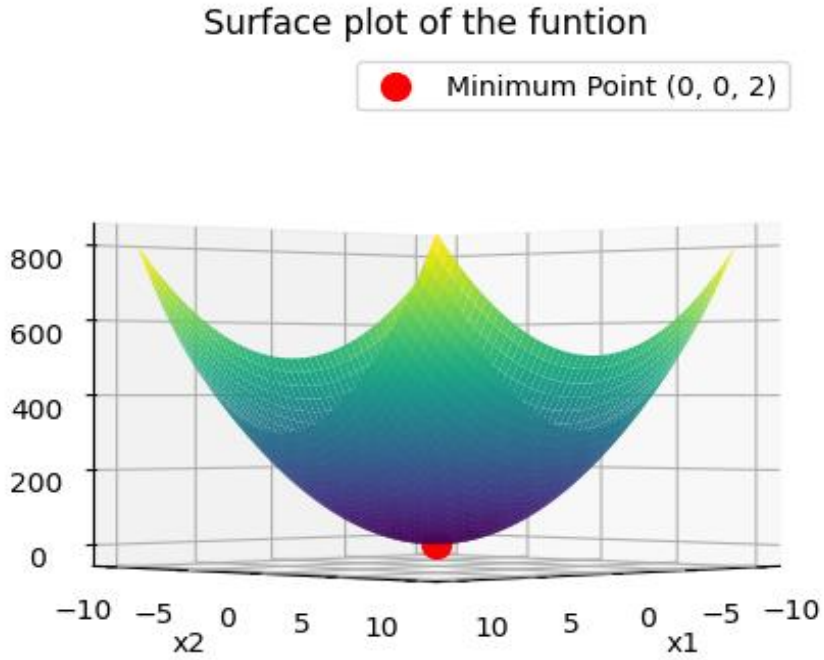


Variation of Acceptance Rate Against Temperature

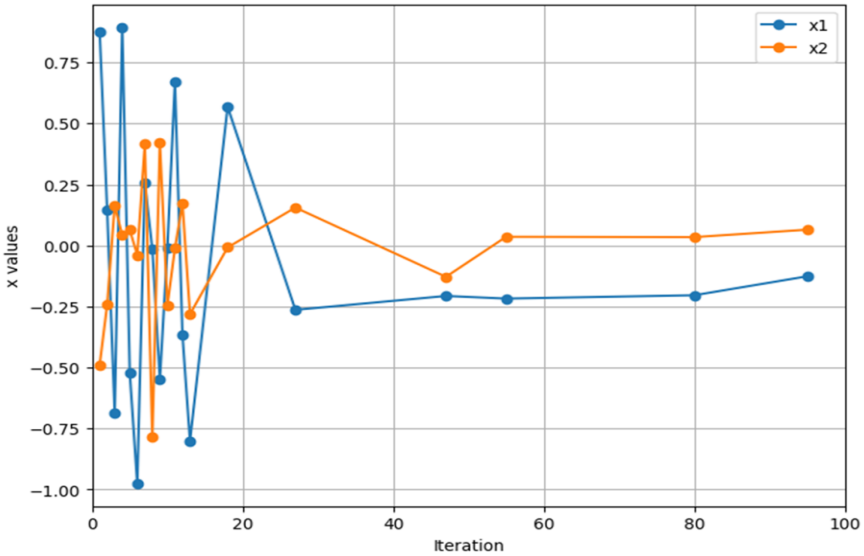


Introducing a Dual Variable Function

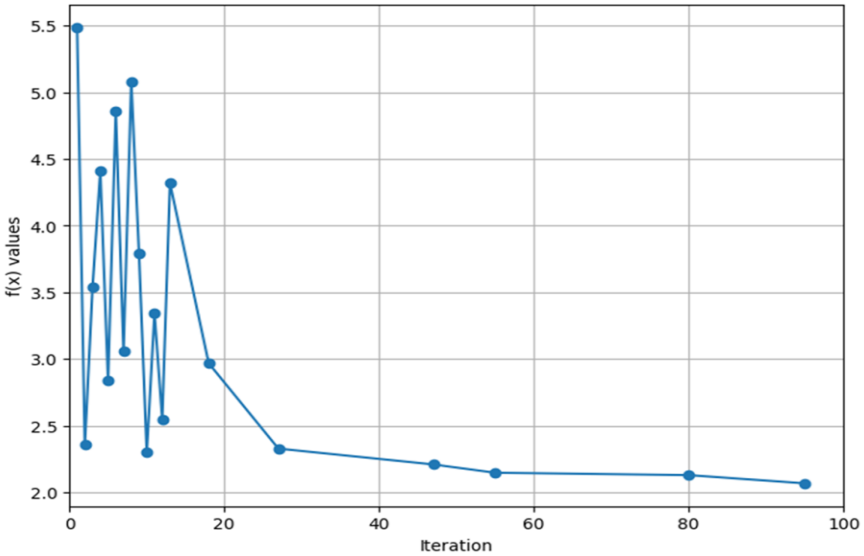
$$f(x_1, x_2) = 3{x_1}^2 + 5{x_2}^2 + 2$$



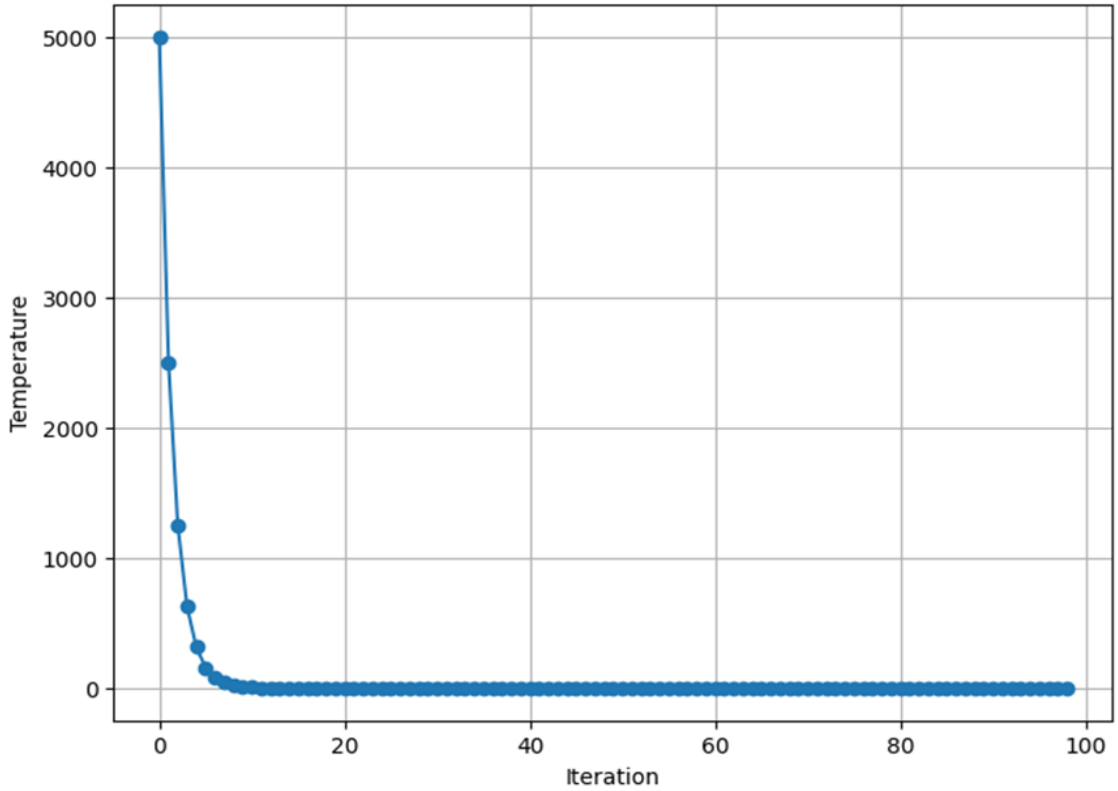
Iteration vs x values plot
(Initial temperature = 5000K)



Iteration vs f(x) values plot
(Initial temperature = 5000K)



Iteration vs Temperature plot



Introducing a New Cooling Schedule

$$T_{new} = T_{initial} \cdot e^{-ki}$$

$k = Cooling\ rate$
$i = Current\ iteration$

Previous Schedule
(Linear)

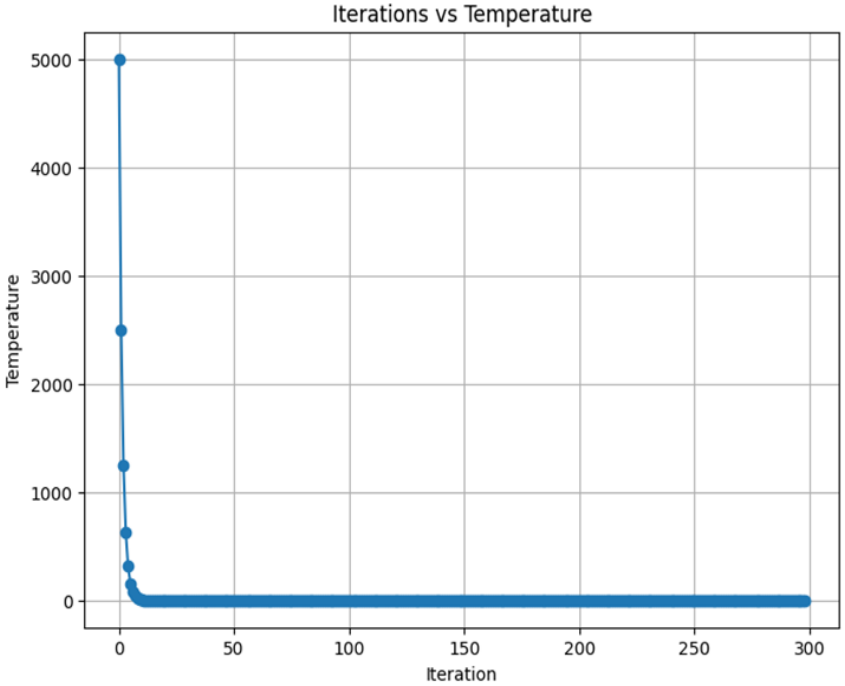
$$T_{new} = T_{current} - T_{current} \cdot k$$

New Schedule
(Exponential)

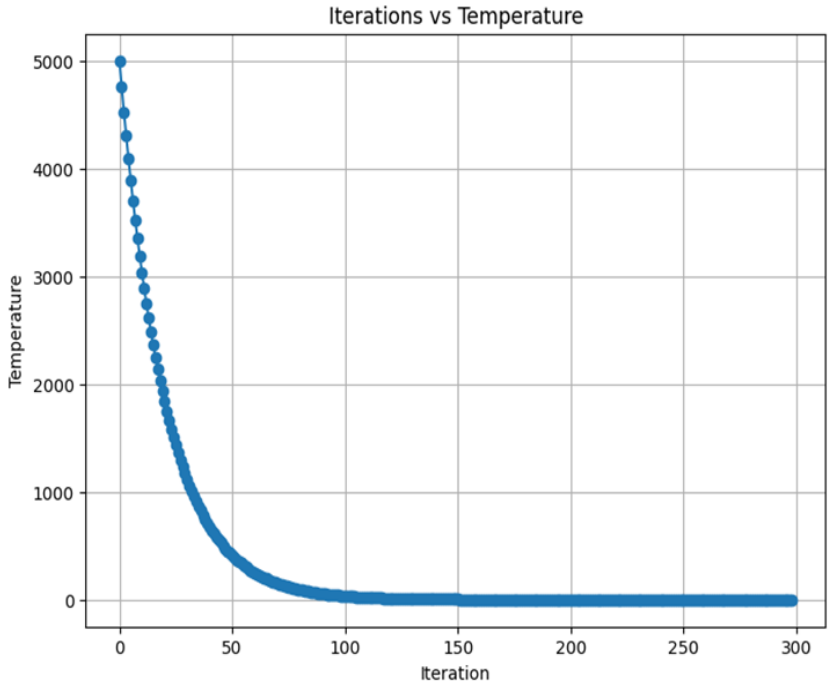
$$T_{new} = T_{initial} \cdot e^{-ki}$$

Iteration vs Temperature Plots

Linear Schedule

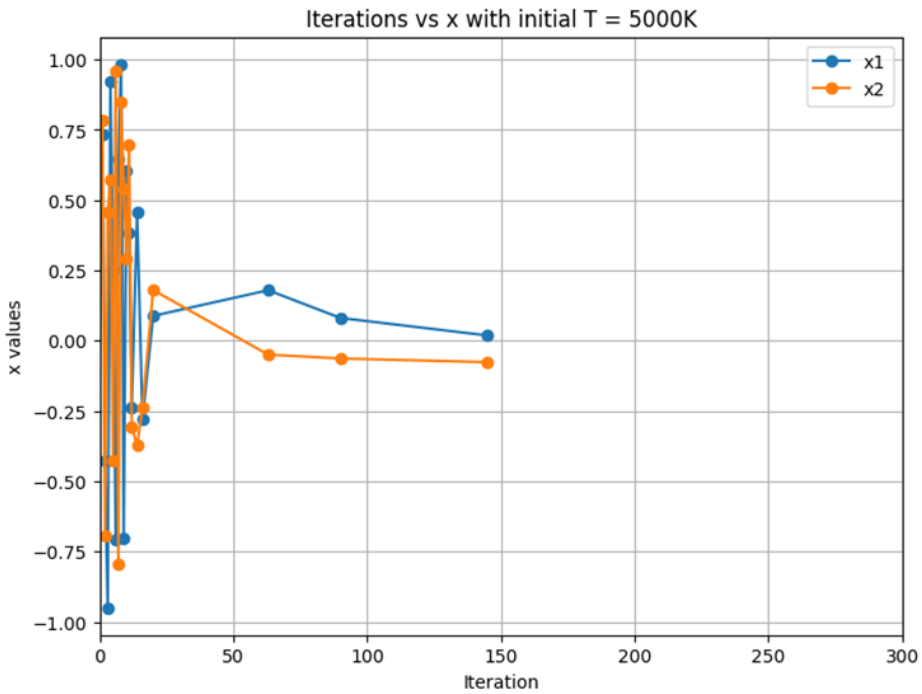


Exponential Schedule

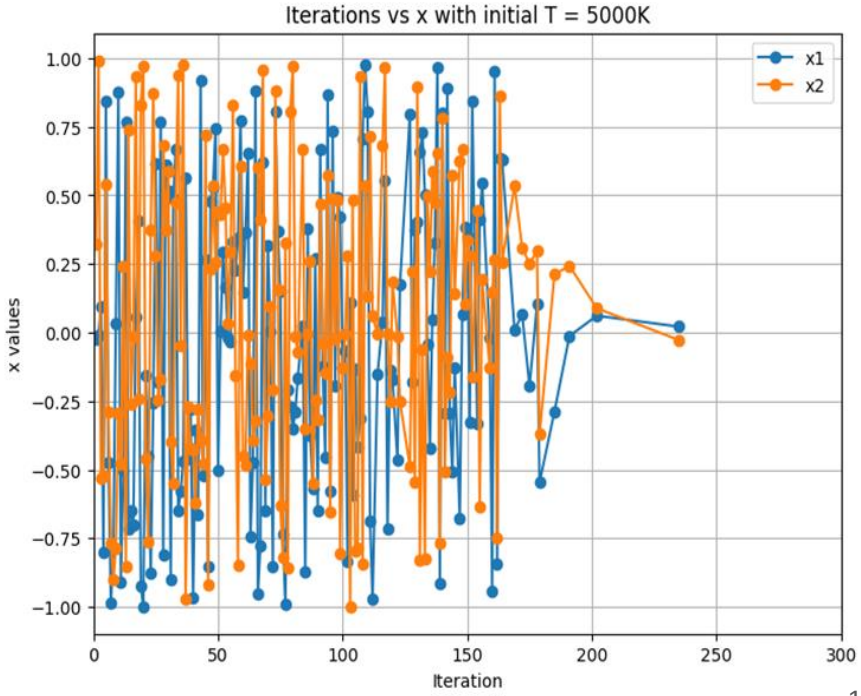


Iteration vs x Plots

Linear Schedule

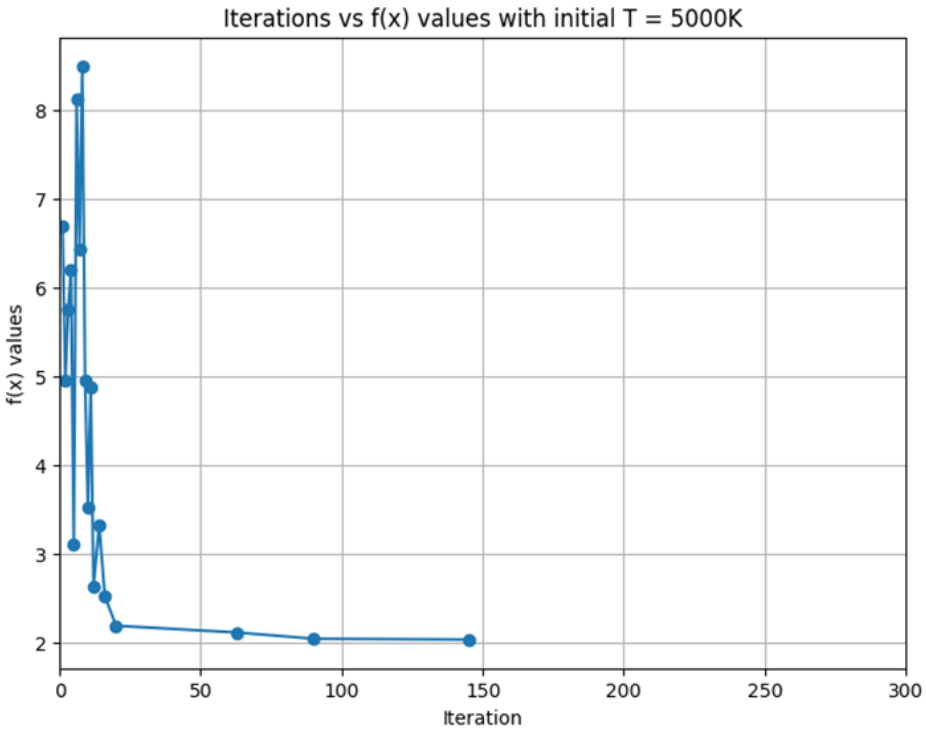


Exponential Schedule

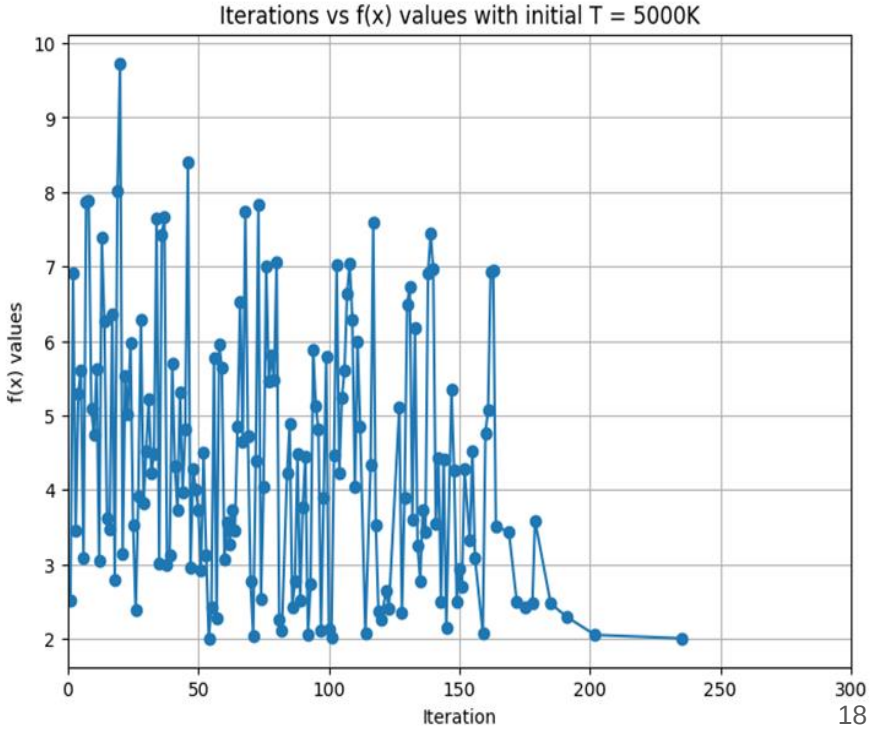


Iteration vs f(x) Values Plots

Linear Schedule



Exponential Schedule



Metropolis Criterion to Simulated Annealing Algorithm

Algorithm 1 Basic Simulated Annealing Algorithm [1–3]

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

Cooling schedule :

$$T_{new} = T_{initial} \cdot e^{-ki}$$

$i = iteration\ number\ (outer\ loop)$
$L_k = markov\ chain\ length$
$T = temperature$

Increment of Markov chain length:

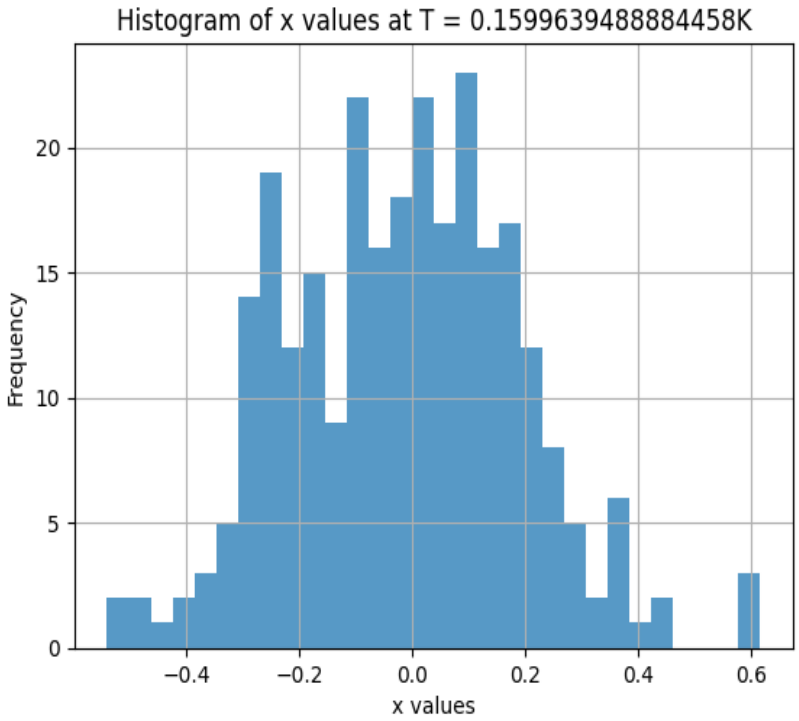
$$L_k = L_0(1.01)^i$$

$$i = -\frac{1}{k}\ln(\frac{T_{new}}{T_{initial}})$$

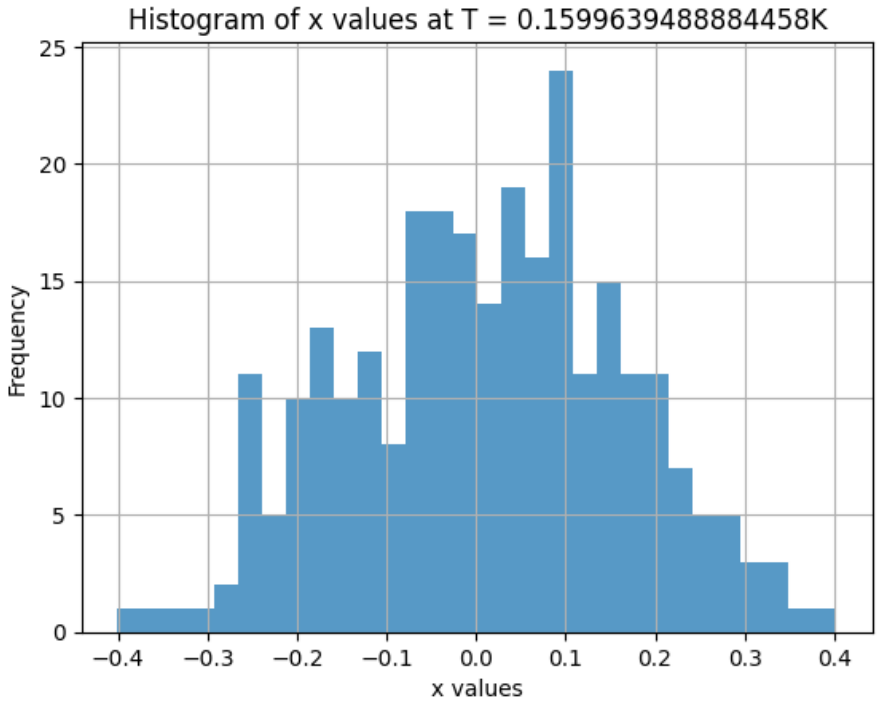
$$L_k = L_0 * 1.01^{\{-\frac{1}{k}\ln(\frac{T_{new}}{T_{initial}})\}}$$

X Chain of Inner Loop at 0.1599K

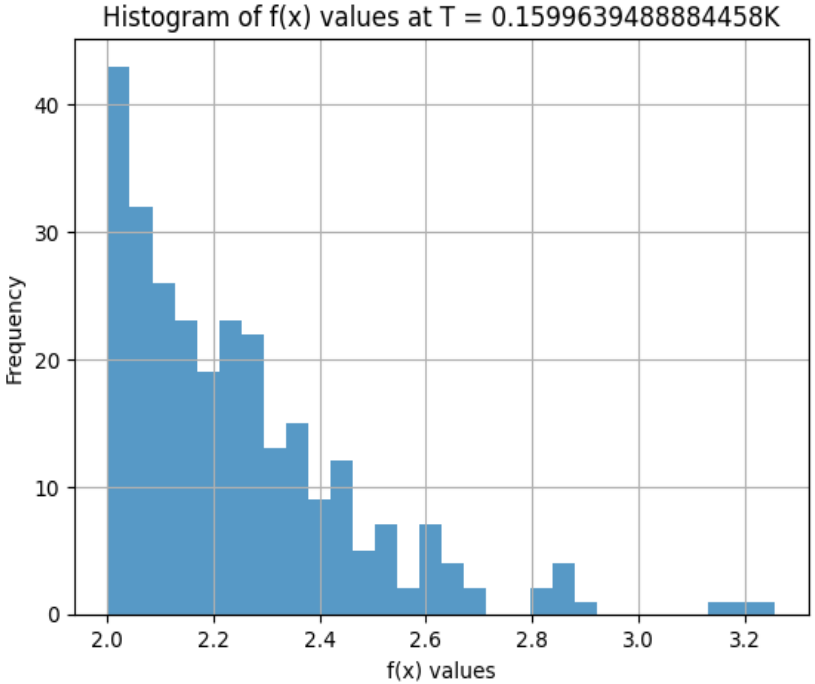
For x_1 :

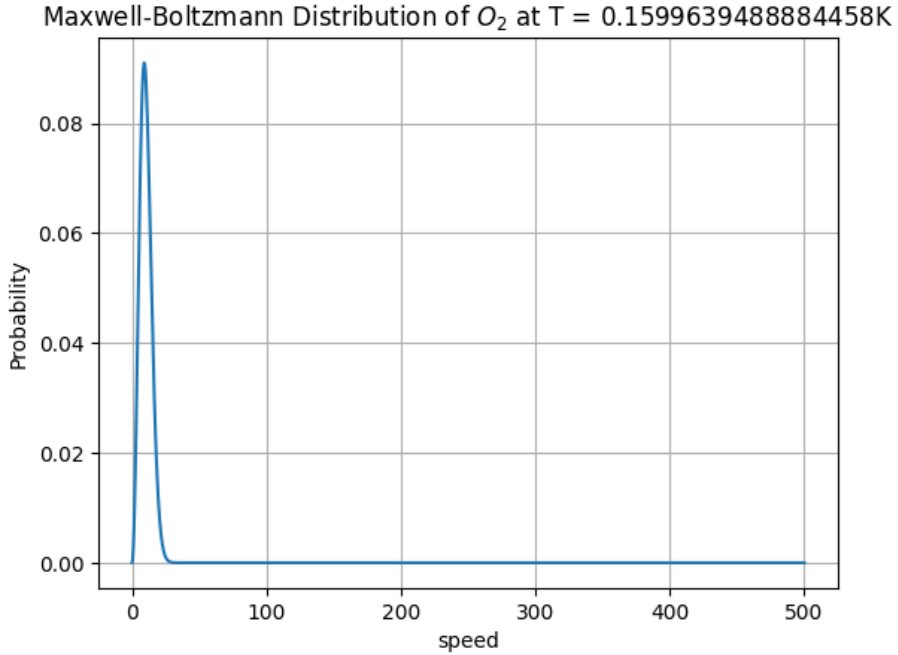


For x_2 :



F(x) values of inner loop at 0.1599K






```
Optimal Solution = 2.000093733172636
Minimum points = (0.004743140031492343, -0.002290896792140229)
Error = 8.785907652420836e-09
```

Validating Developed SA Algorithm Using Benchmark Functions

Benchmark Functions

- Benchmark functions are standard test functions used to evaluate and compare the performance of optimization algorithms.
- Importance of Benchmarking
  - ✓ Standardization: Ensures consistency in evaluating different optimization techniques.
  - ✓ Comparability: Allows for direct comparison between various algorithms.
  - ✓ Validation: Helps validate the effectiveness and reliability of an algorithm.

Why Benchmark Functions ?

- **Performance Evaluation**: Helps in assessing the convergence and solution quality of SA.
- **Parameter Tuning**: Assists in fine-tuning parameters like temperature schedule, cooling rate, etc.
- **Algorithm Improvement**: Identifies strengths and weaknesses, guiding further enhancements.

Multimodal Benchmark Function

Price 2 Function [67]
(Continuous, Differentiable, Non – Separable, Non – Scalable, Multimodal)

$$\boldsymbol{f_{95}(x) = 1 + sin^2(x_1) + sin^2(x_2) - 0.1\, e^{-x_1^2 - x_2^2}}$$

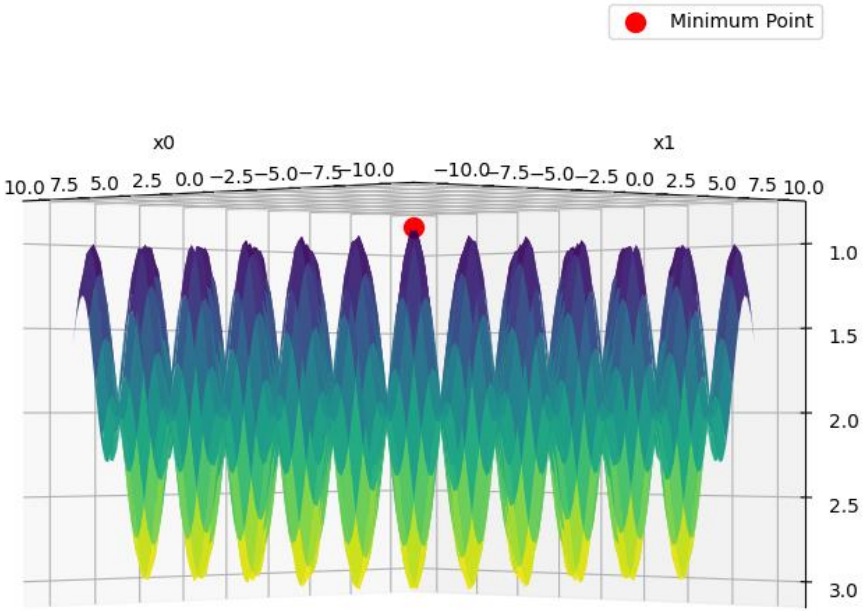
$$Subject\ to \quad -10 \leq x \leq 10$$

$$The\ global\ minimum\ is\ located\ at\ x^* = f(0,0), \qquad f(x^*) = 0.9$$

Multimodal Benchmark Function

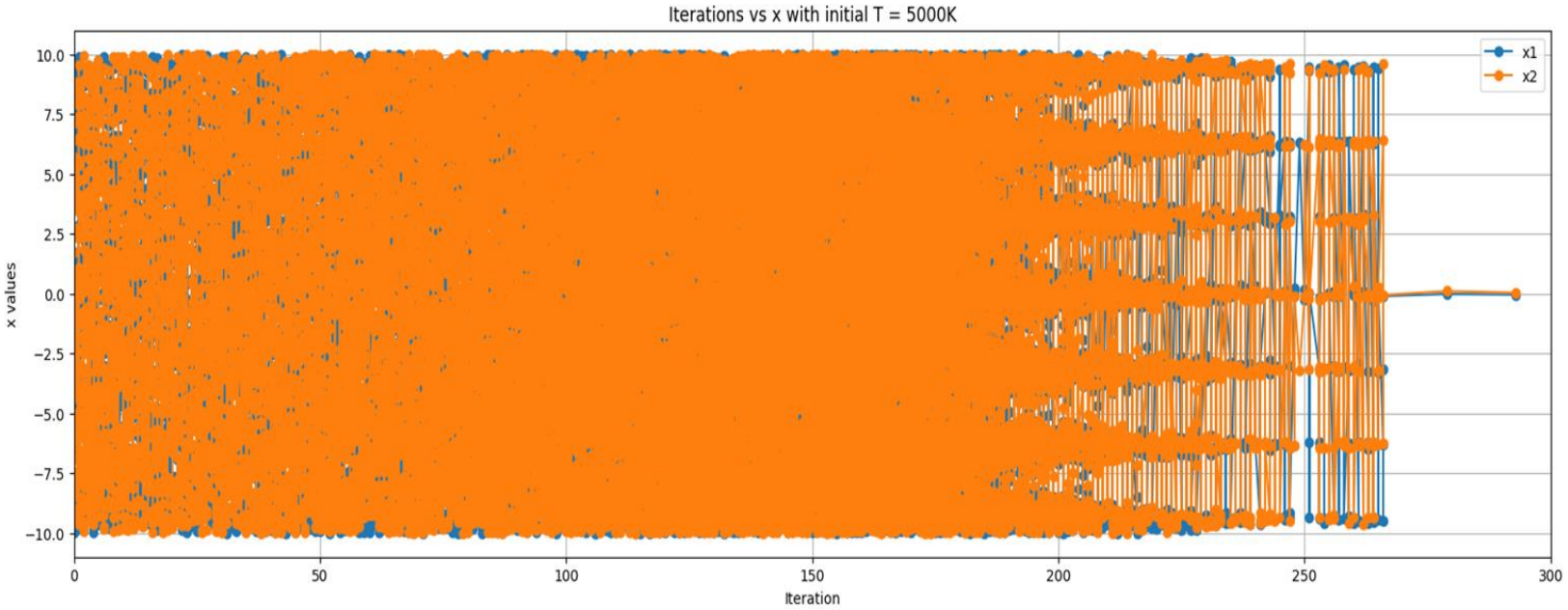
Surface Plot of the Function

$$f_{95}(x) = 1 + sin^2(x_1) + sin^2(x_2) - 0.1\, e^{-x_1^2 - x_2^2}$$

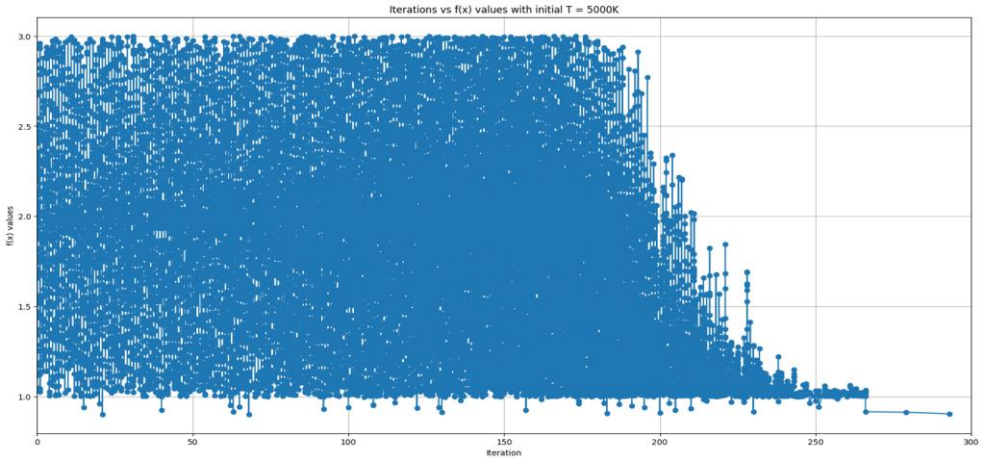



```
Minimum point: [0. 0.]
Function value at minimum point: 0.9
```

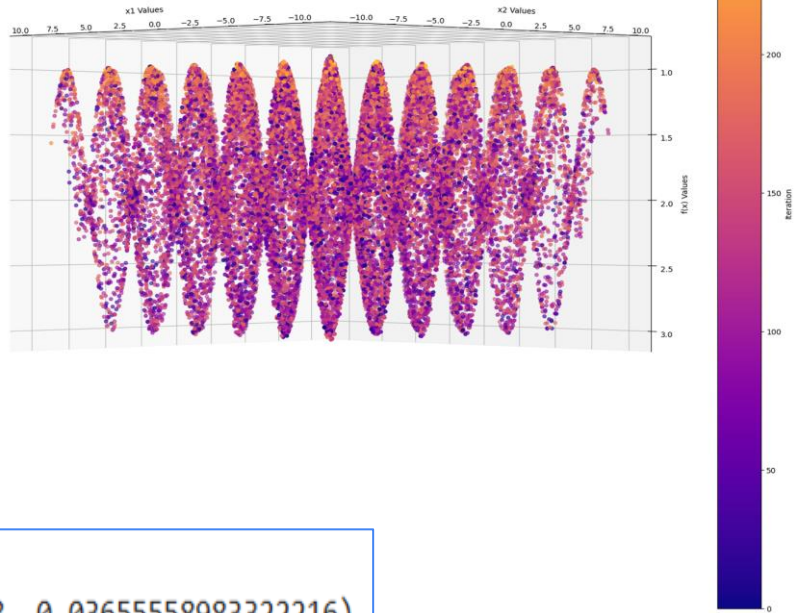
Iteration vs x values plot
(Initial temperature = 5000K)



Iteration vs f(x) values plot
(Initial temperature = 5000K)

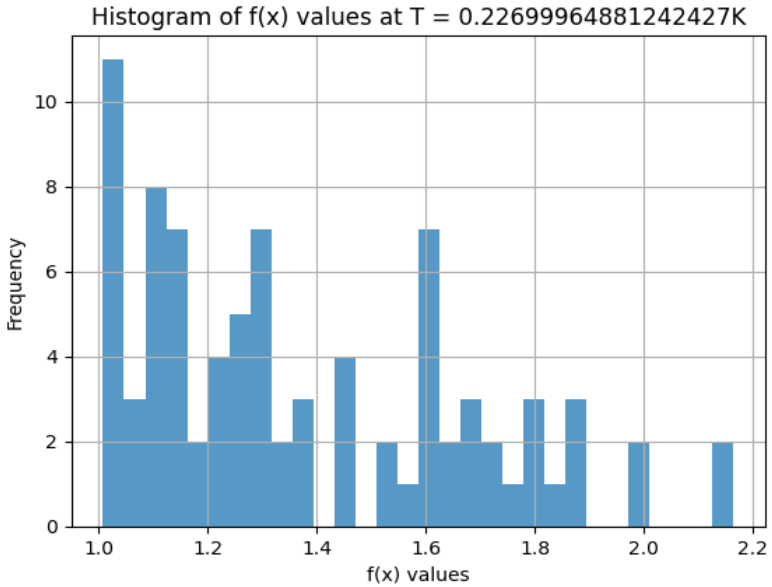


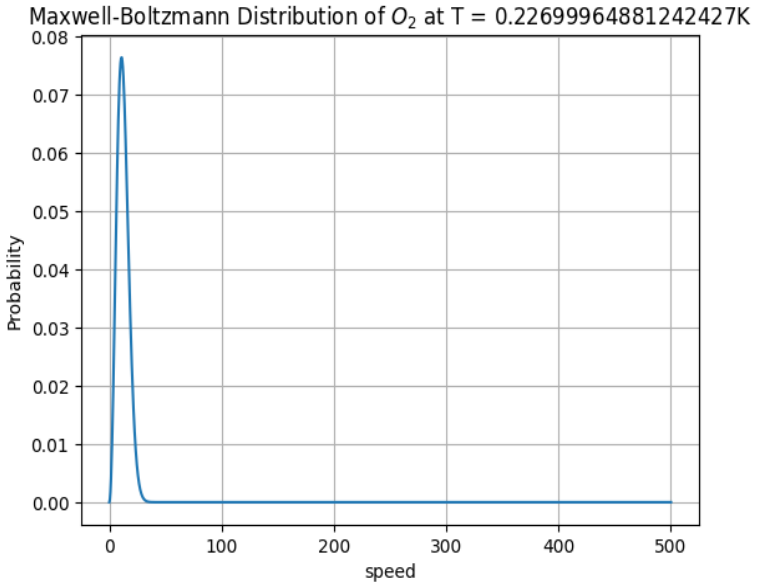
3D scatterplot for
iteration vs f(x) values




```
Optimal Solution = 0.9035364384866527
Minimum points = (-0.04336721639244878, 0.03655558983322216)
Error = 1.2506397169877998e-05
```

Inner Loop at T = 0.227K





Unimodal Benchmark Function

Ackley 2 Function [1]
(Continuous, Differentiable, Non – Separable, Non – Scalable, Unimodal)

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

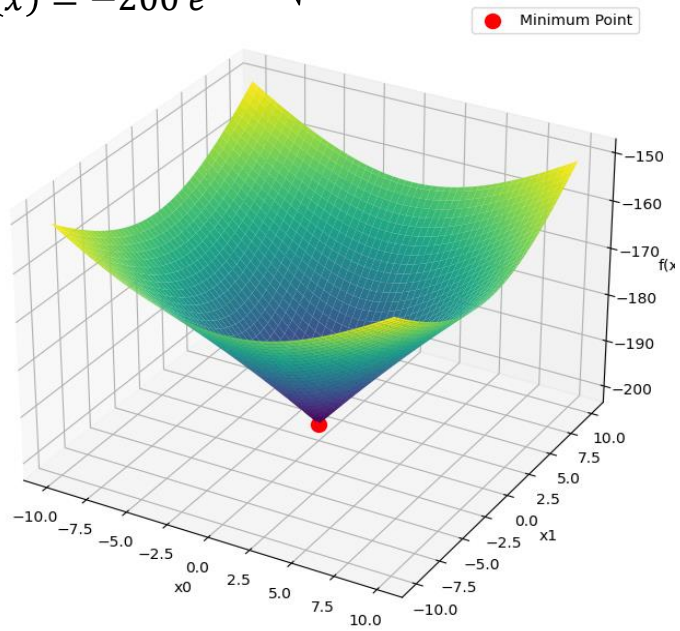
$$Subject\ to\ \ -32 \leq x \leq 32$$

$$The\ global\ minimum\ is\ located\ at\ x^* = (0,0), \qquad f(x^*) = -200$$

Unimodal Benchmark Function

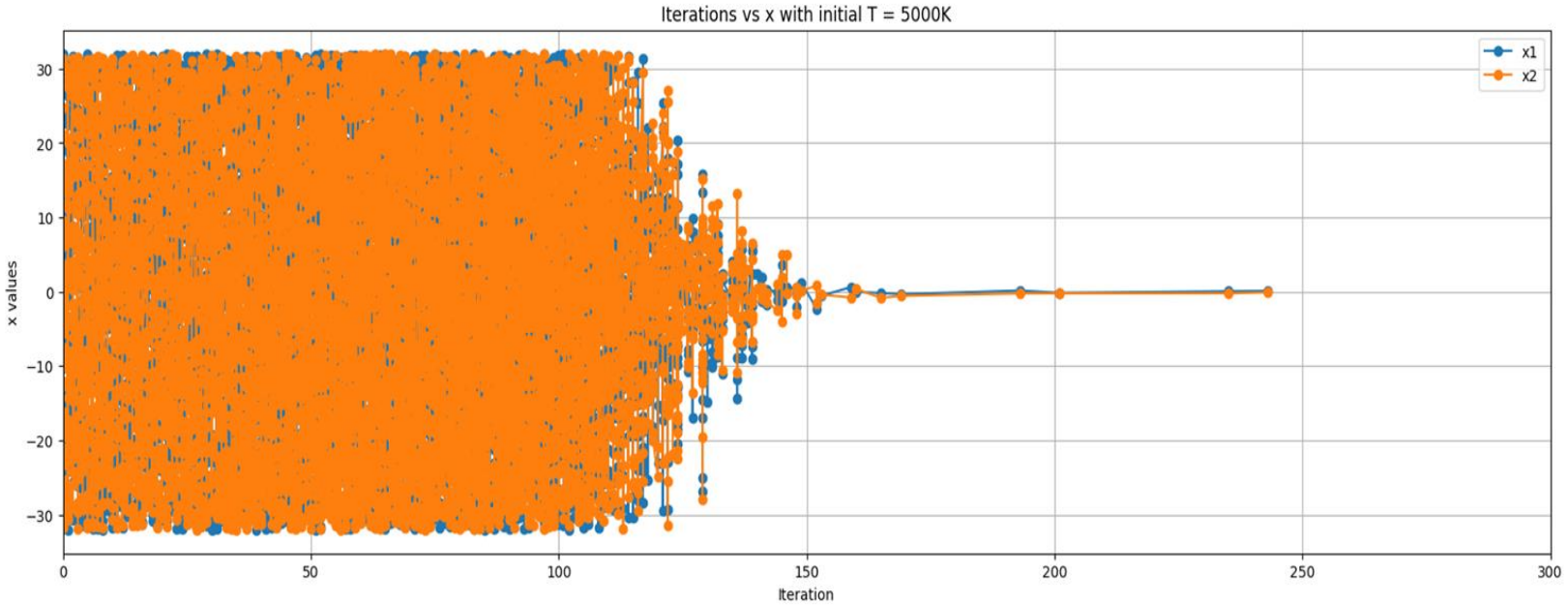
Surface Plot of the Function

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

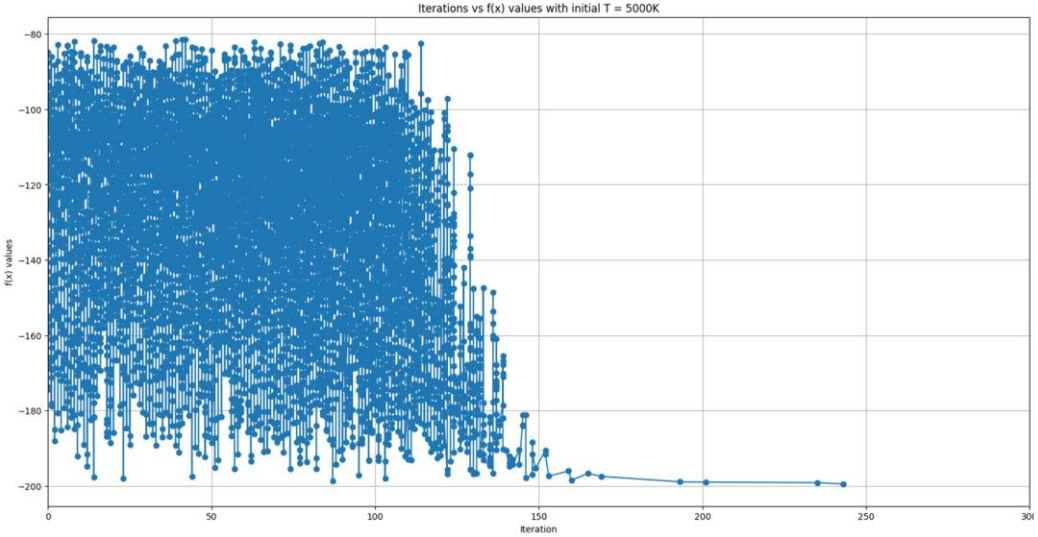



```
Minimum point: [0. 0.]
Function value at minimum point: -200.0
```

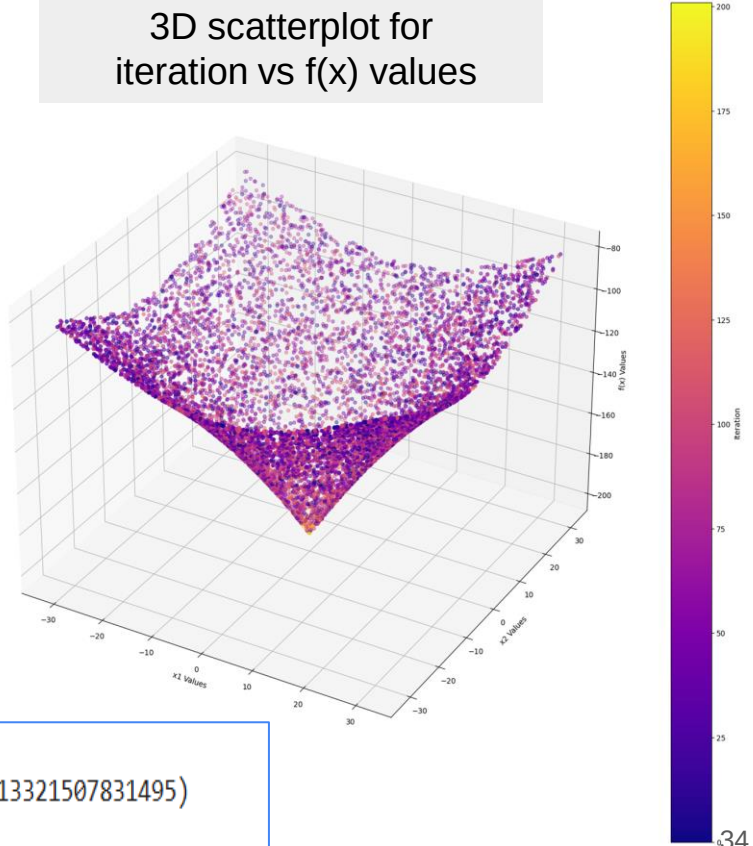
Iteration vs x values plot
(Initial temperature = 5000K)



Iteration vs f(x) values plot
(Initial temperature = 5000K)



3D scatterplot for
iteration vs f(x) values




```
Optimal Solution = -199.44988290250936
Minimum points = (0.11054679189005157, -0.08213321507831495)
Error = 0.302628820951528
```

Unimodal Benchmark Function (2)

Rotated Ellipse 2 Function [66]
(Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal)

$$\boldsymbol{f_{108}(x) = x_1^2 - x_1 x_2 + x_2^2}$$

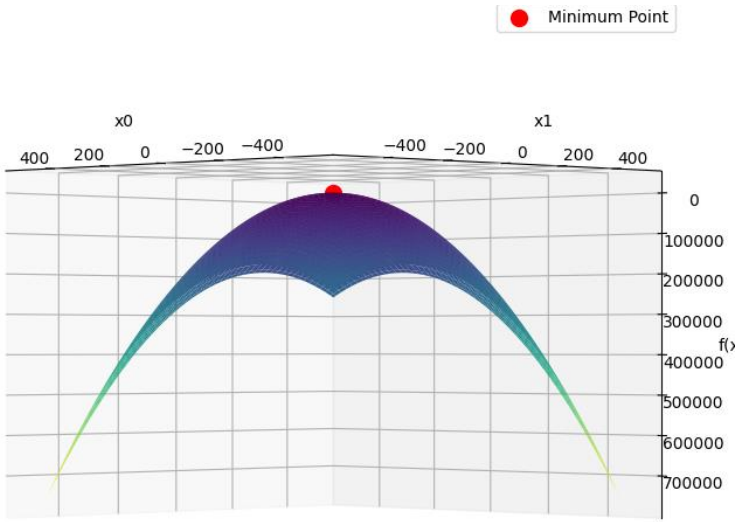
$$Subject\ to\ \ -500 \leq x \leq 500$$

$$The\ global\ minimum\ is\ located\ at\ x^* = (0,0), f(x^*) = 0$$

Unimodal Benchmark Function (2)

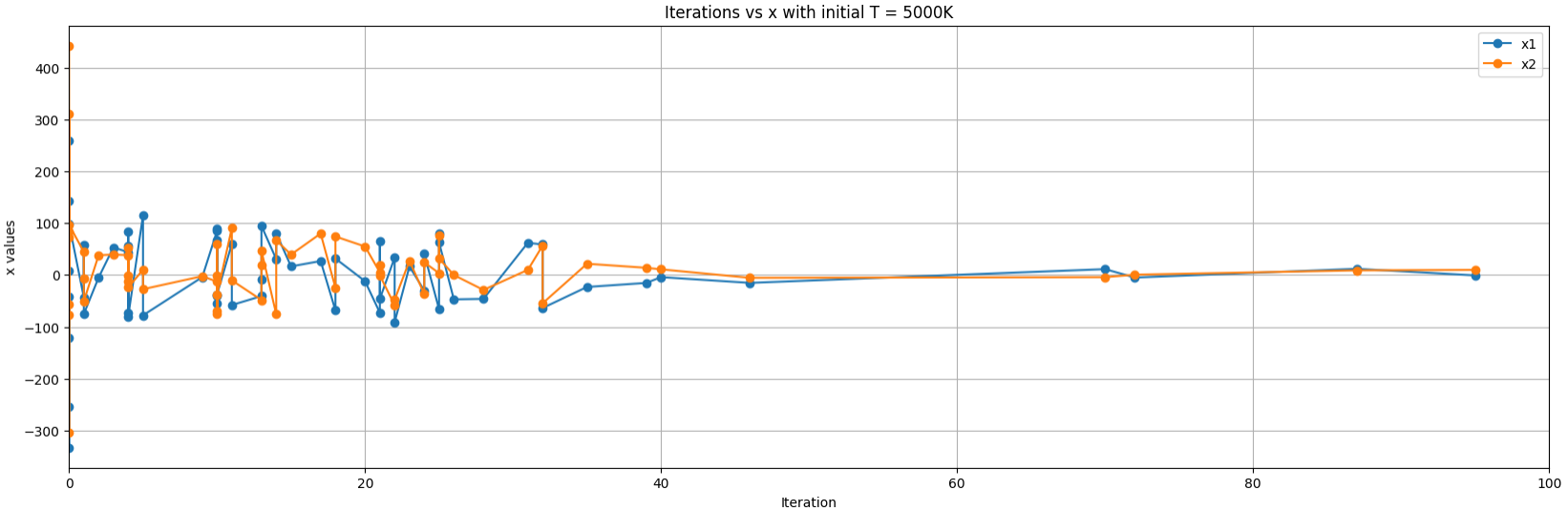
Surface Plot of the Function

$$f_{108}(x) = x_1^2 - x_1 x_2 + x_2^2$$

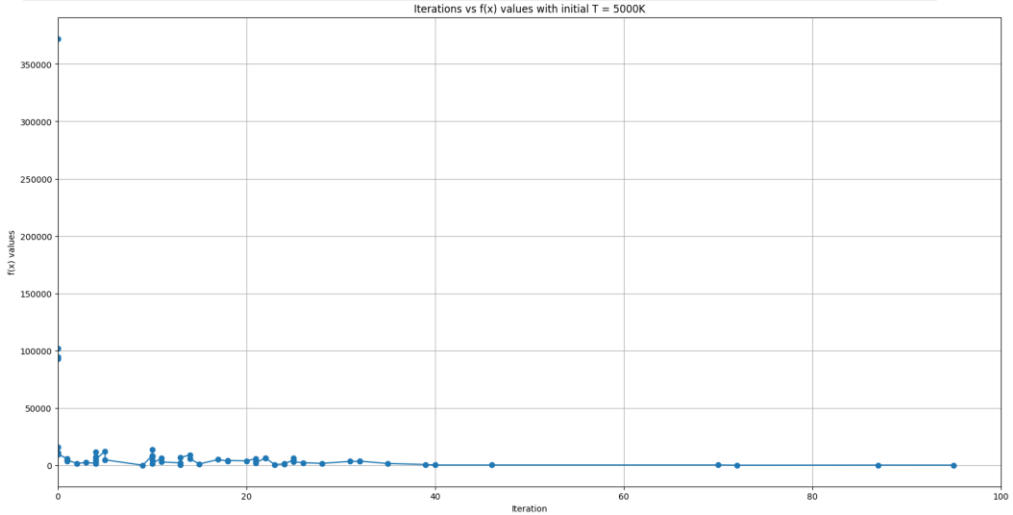



```
Minimum point: [0. 0.]
Function value at minimum point: 0.0
```

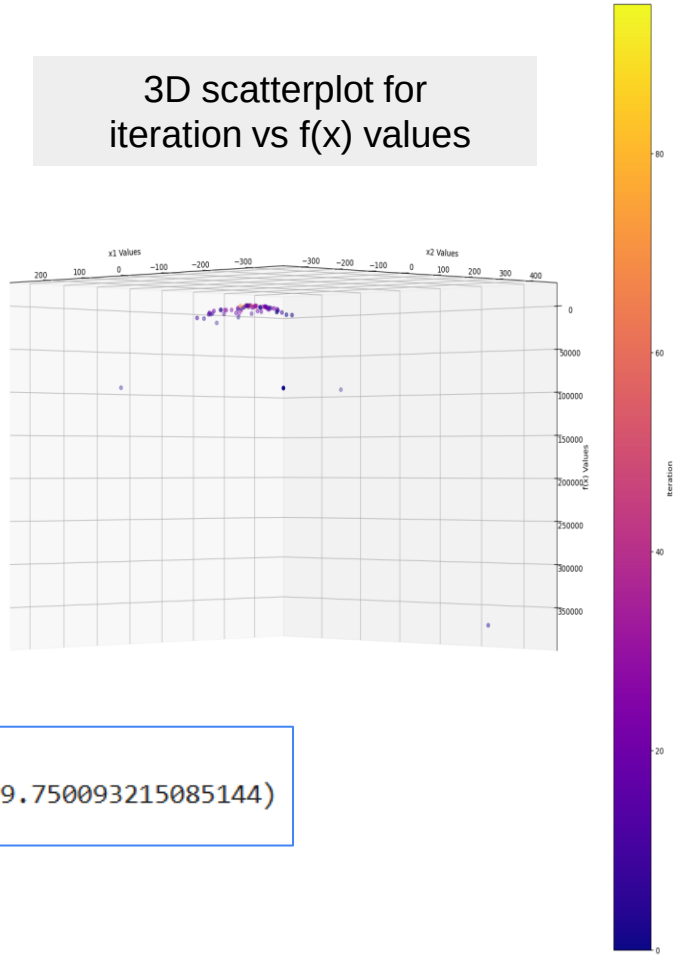
Iteration vs x values plot
(Initial temperature = 5000K)



Iteration vs f(x) values plot
(Initial temperature = 5000K)



3D scatterplot for iteration vs f(x) values



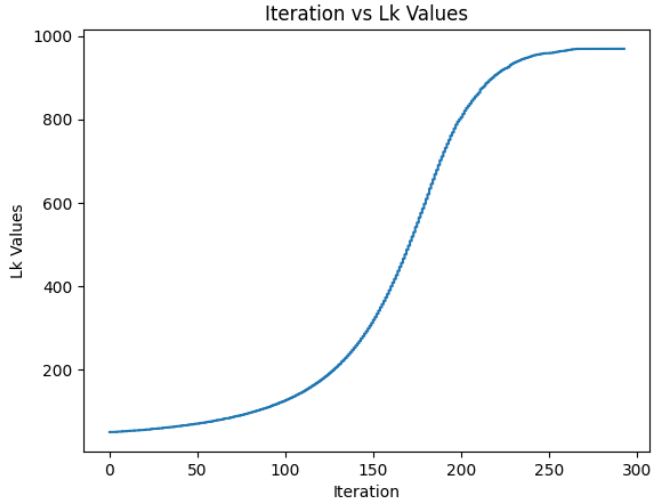

```
Optimal Solution = 105.03232078587006
Minimum points = (-0.9330582906779341, 9.750093215085144)
Error = 11031.78840966591
```

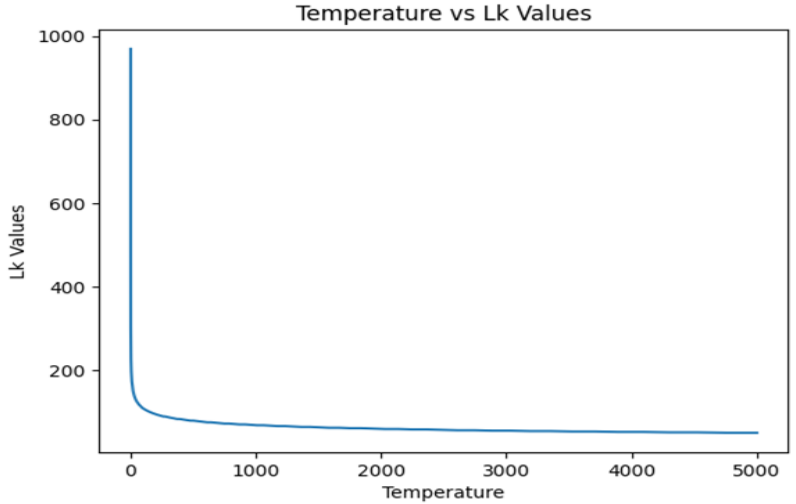
Comparative Analysis

1 - Varying L_k vs Constant L_k
(for Unimodal and Multimodal function)

Multimodal Function

For a varying L_k value

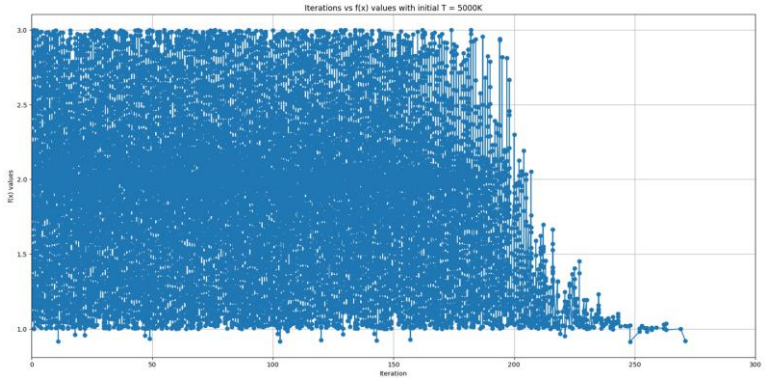




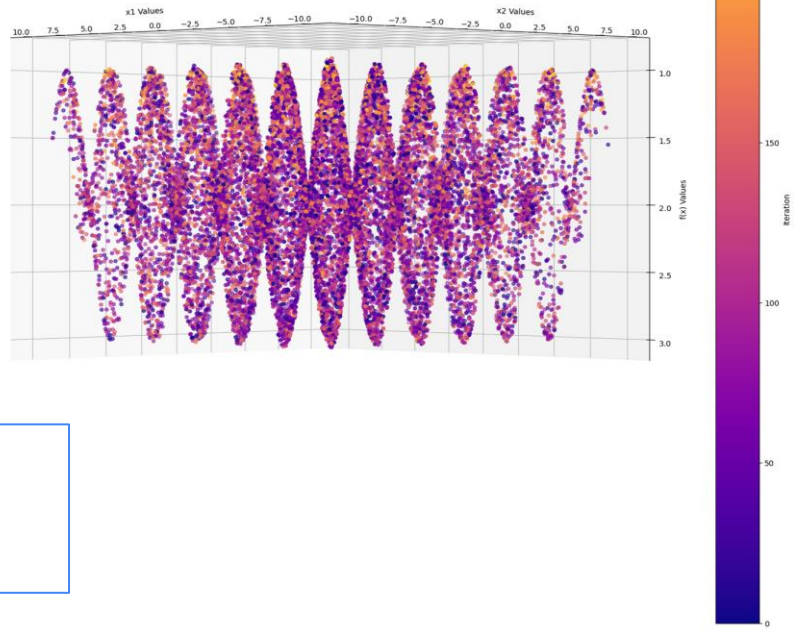

```
Optimal Solution = 0.9035364384866527
Minimum points = (-0.04336721639244878, 0.03655558983322216)
Error = 1.2506397169877998e-05
```

For a constant L_k value (L_k = 50)

Iteration vs f(x) values plot



3D scatterplot for iteration vs f(x) values

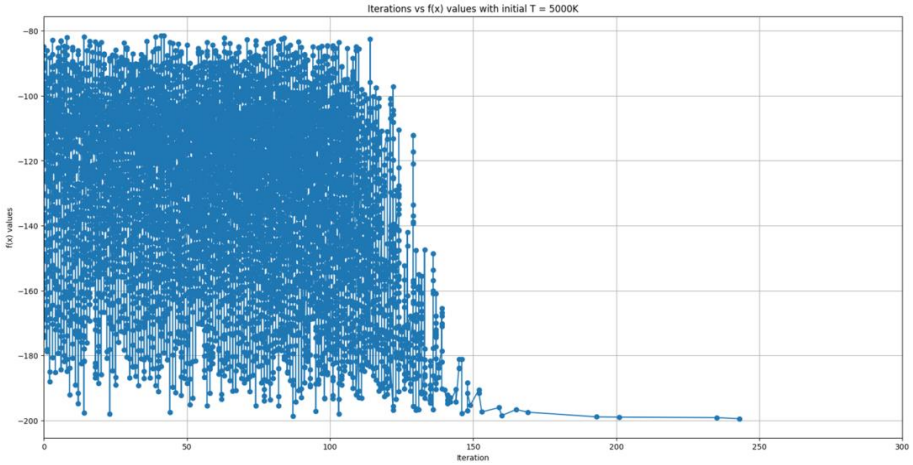



```
Optimal Solution = 0.9195322662971305
Minimum points = (-0.09299332187339182, -0.09576684309869066)
Error = 0.0003815094267020181
```

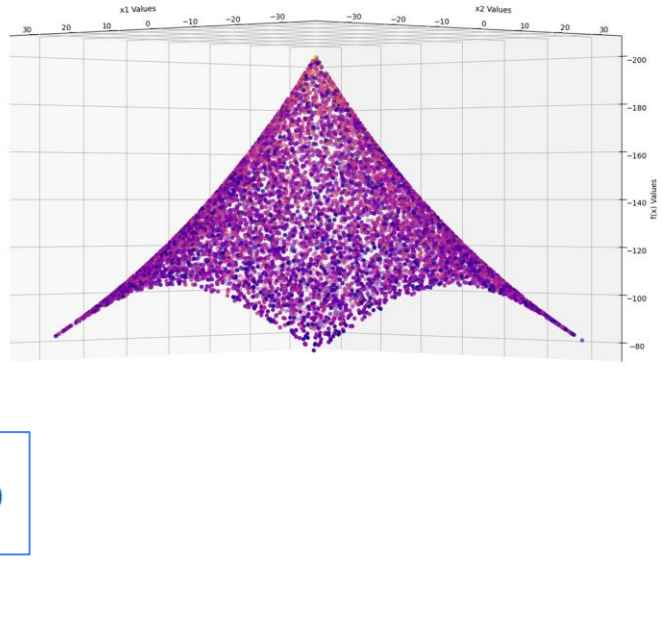
Unimodal Function

For a varying L_k value

Iteration vs f(x) values plot



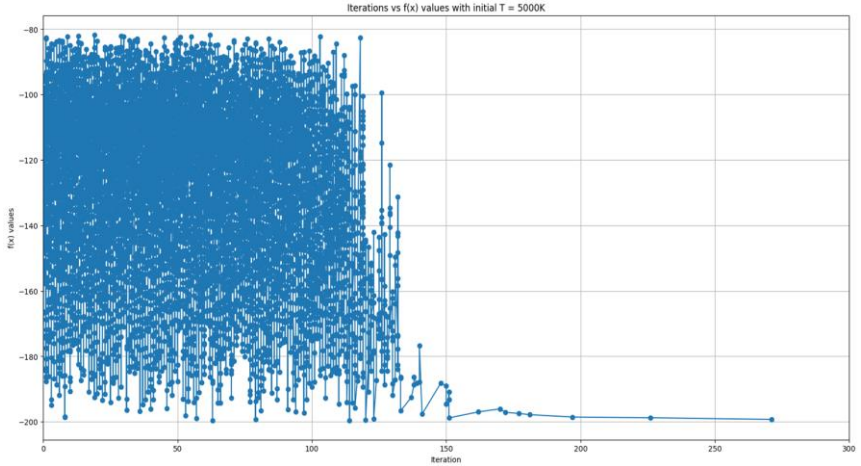
3D scatterplot for iteration vs f(x) values



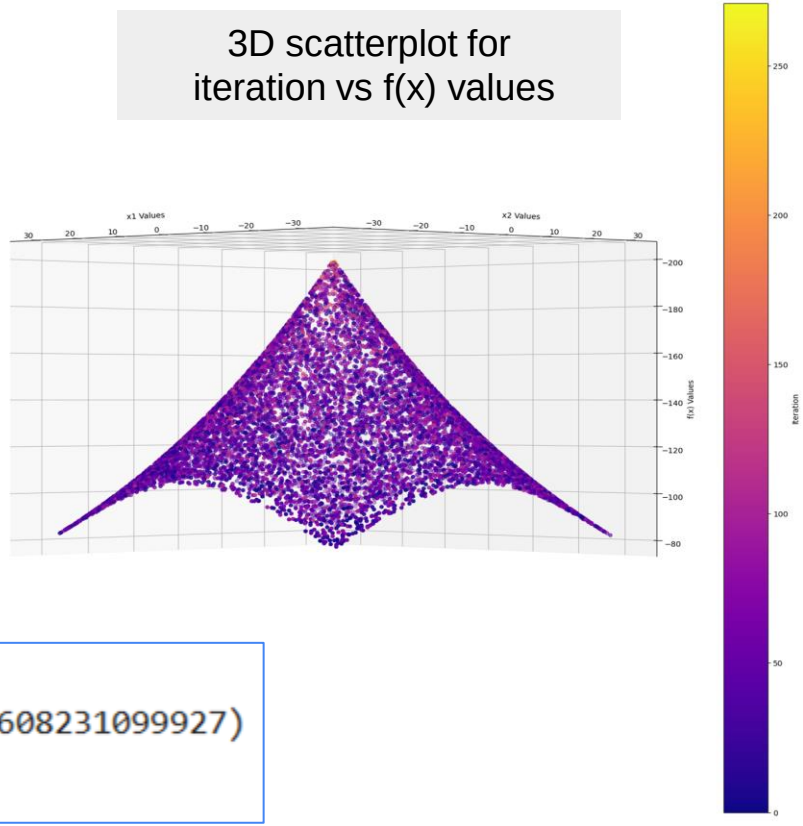

```
Optimal Solution = -199.44988290250936
Minimum points = (0.11054679189005157, -0.08213321507831495)
Error = 0.302628820951528
```

For a constant L_k value (L_k = 50)

Iteration vs f(x) values plot



3D scatterplot for iteration vs f(x) values




```
Optimal Solution = -199.27628033582124
Minimum points = (-0.16666877815666936, -0.07124608231099927)
Error = 0.5237701523190158
```

Observations

For 100 runs	**Case 1**	**Case 2**
	Unimodal Function	Multimodal Function
Lk varying	Mean Optimal Value = -199.5073789718 Mean Error = 0.31468536962744315	Mean Optimal Value = 0.904805252383 Mean Error = 4.228787492020182e-05
Lk constant (Lk=50)	Mean Optimal Value = -198.6896461195 Mean Error = 2.128397503254249	Mean Optimal Value = 0.94551495287 Mean Error = 0.003430238850707838

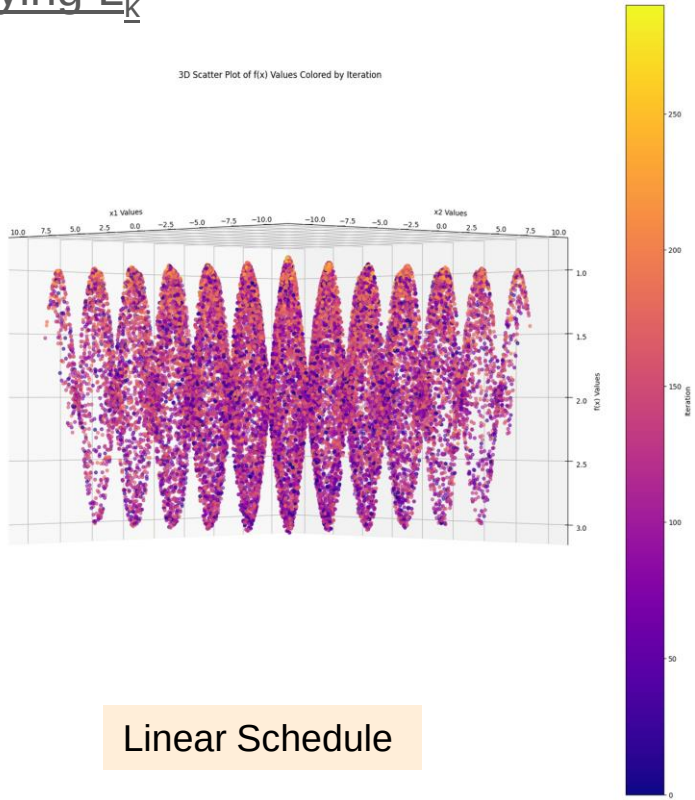
2 - Linear vs Exponential Cooling Schedule (for Multimodal function & Unimodal function)

Observations - For Multimodal Function

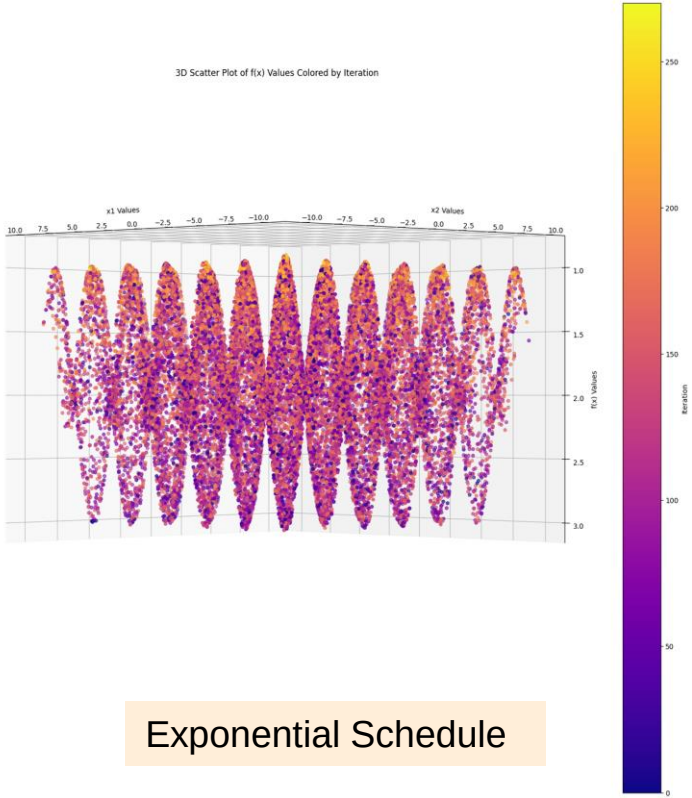
For 100 runs		**Linear Schedule**	**Exponential Schedule**
Case 1	Varying L_k	Mean Optimal Value = 0.903733180203 Mean Error = 3.049632504119059e-05	Mean Optimal Value = 0.904805252383 Mean Error = 4.228787492020182e-05
Case 2	Constant L_k	Mean Optimal Value = 0.94521226607 Mean Error = 0.0033228358221415633	Mean Optimal Value = 0.94551495287 Mean Error = 0.003430238850707838

Linear vs Exponential - 3D Scatterplots

Varying L_k

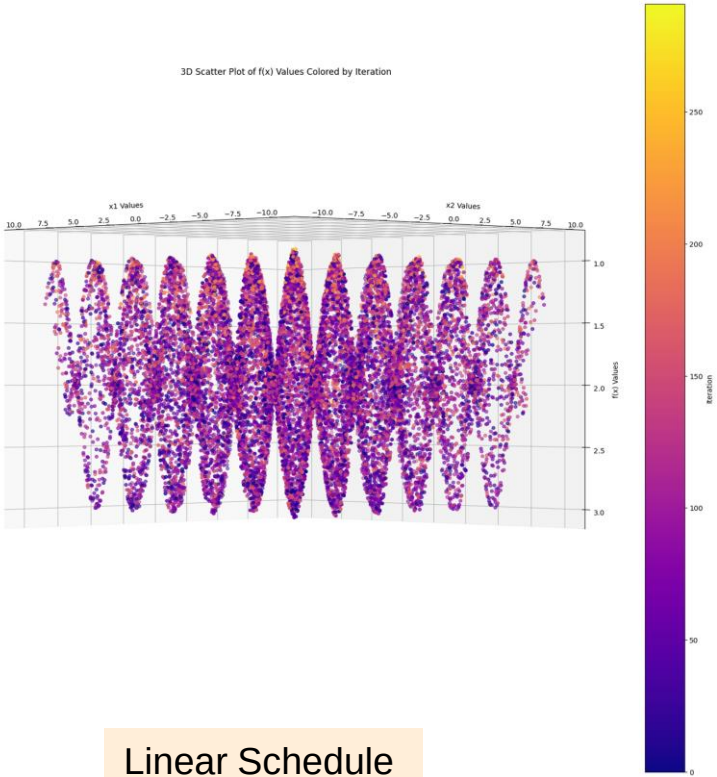


Linear Schedule

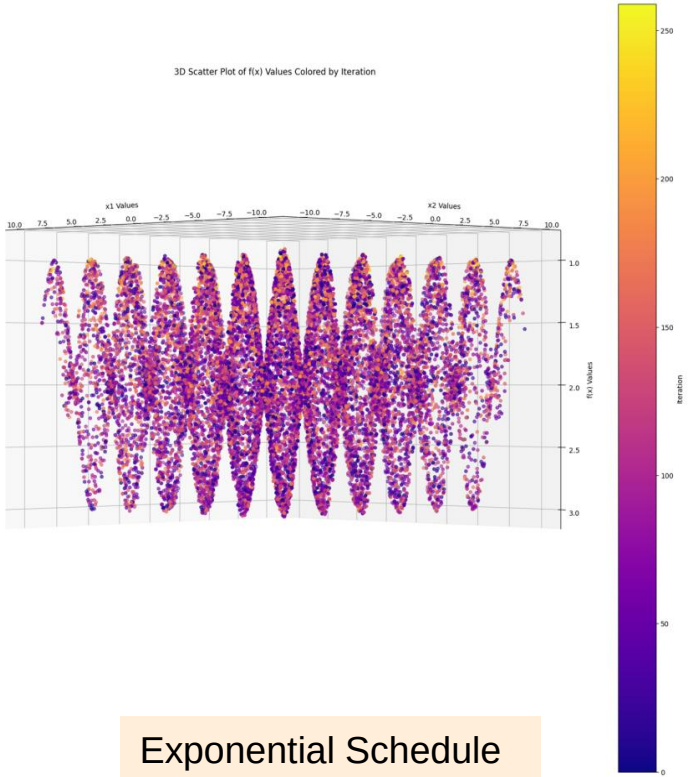


Exponential Schedule

Constant L_k



Linear Schedule



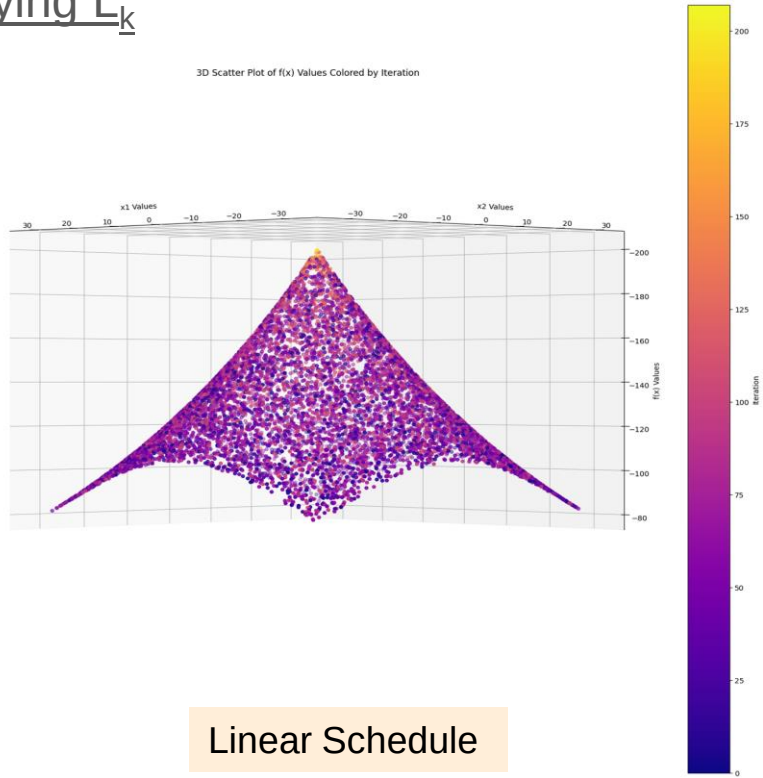
Exponential Schedule

Observations - For Unimodal Function

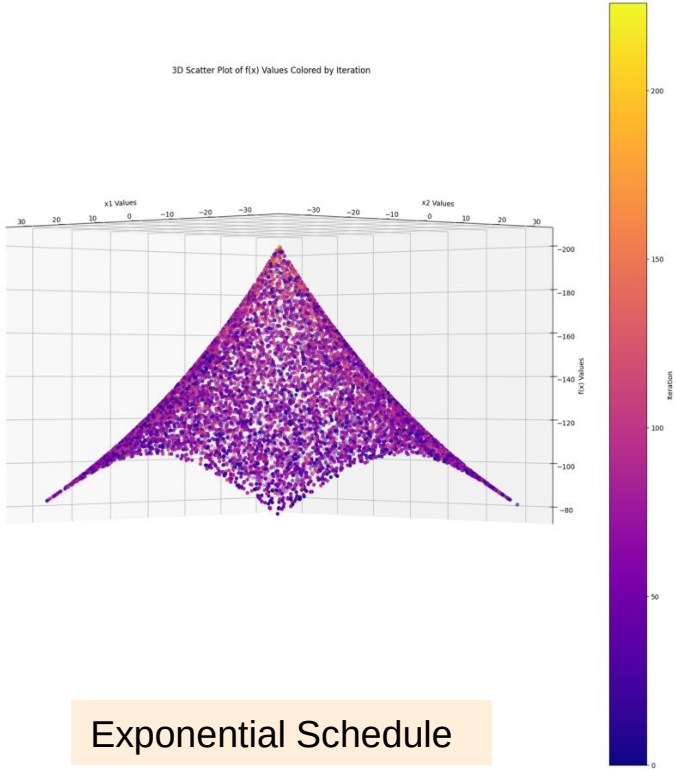
For 100 runs		**Linear Schedule**	**Exponential Schedule**
Case 1	Varying L_k	Mean Optimal Value = -199.5181847 Mean Error = 0.3064314058284057	Mean Optimal Value = -199.5073789718 Mean Error = 0.3146853696274431 5
Case 2	Constant L_k	Mean Optimal Value = -198.4343832 Mean Error = 3.0073503291217665	Mean Optimal Value = -198.689646119 Mean Error = 2.128397503254249

Linear vs Exponential – 3D Scatterplots

Varying L_k

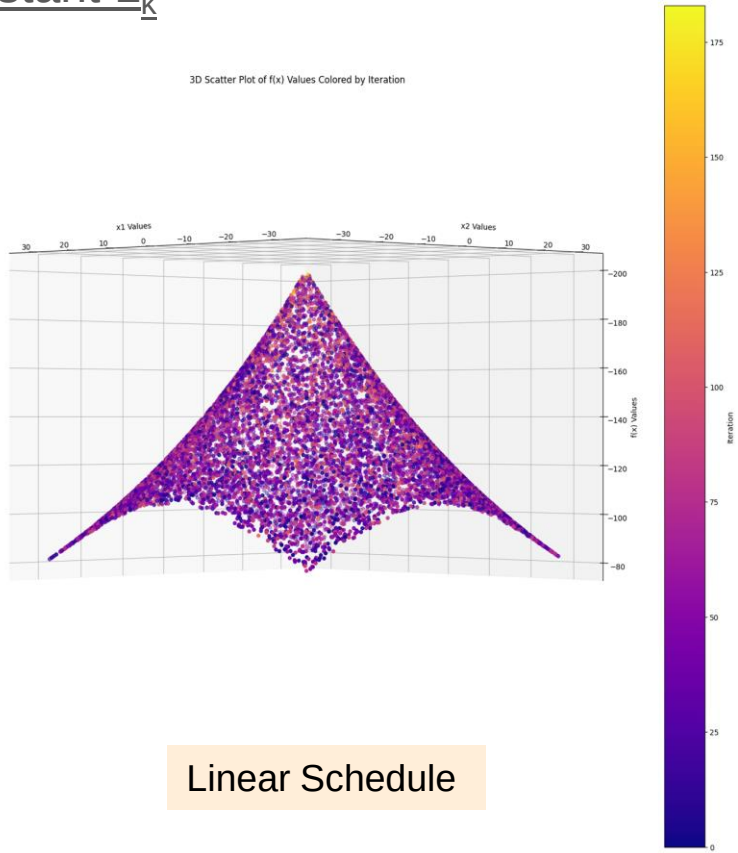


Linear Schedule

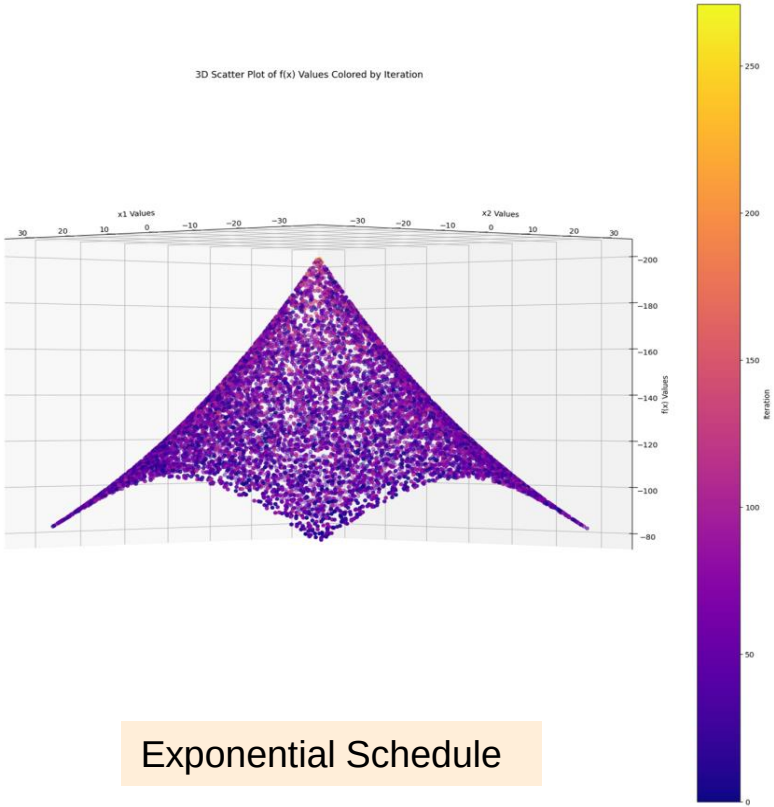


Exponential Schedule

Constant L_k



Linear Schedule



Exponential Schedule

Appendix

```
temp = 5000
k = 0.05
no_of_iterations = 100
L0 = 50  # Initial Markov chain length
```

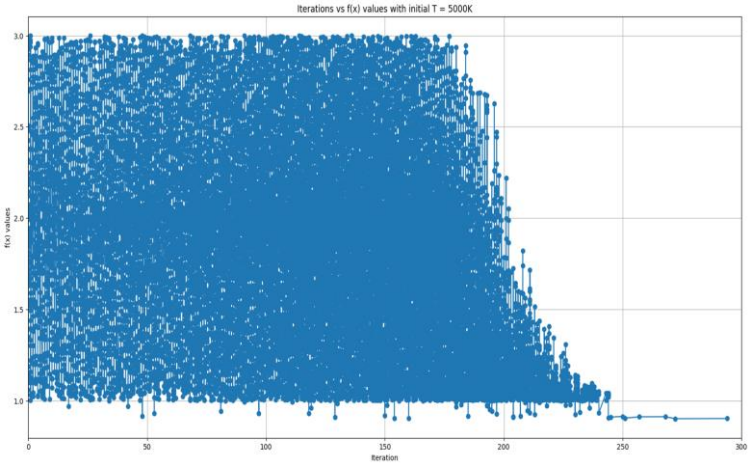
```
def simulated_annealing_algo(temperature_value, no_of_iterations, L0, k, a, b):
    init_temp = temperature_value
    state_i = np.random.uniform(a, b, 2)
    accepted_states_x1 = []
    accepted_states_x2 = []
    iteration = []
    temperature = []
    f_x_values = []
    inner_temp = []
    Lk_values = []
    Lk = L0  # Initialize Markov chain length

    for i in range(no_of_iterations):
        for p in range(Lk):  # Perform Lk transitions at each temperature level
            state_j = np.random.uniform(a, b, 2)
            energy_diff = f(state_j) - f(state_i)

            if energy_diff <= 0 or math.exp(-energy_diff / temperature_value) > np.random.uniform(0, 1):
              # Accept state_j
                state_i = state_j
                accepted_states_x1.append(state_i[0])
                accepted_states_x2.append(state_i[1])
                f_x_values.append(f(state_i))
                iteration.append(i)
                inner_temp.append(temperature_value)

        # Record the temperature and
        temperature.append(temperature_value)
        Lk_values.append(Lk)

        # Update temperature and Markov chain length for next iteration
        temperature_value = init_temp*math.exp(-k*i)                    #exponential Cooling Schedule
        #temperature_value = temperature_value - temperature_value*k    #Linear Cooling Schedule

        #Update Lk value
        Lk = L0 * (1.01 ** np.round(math.log(temperature_value/init_temp)/-k))
        Lk = int(Lk)  # Convert to integer if necessary

    return accepted_states_x1, accepted_states_x2, iteration, temperature, f_x_values, inner_temp, Lk_values
```

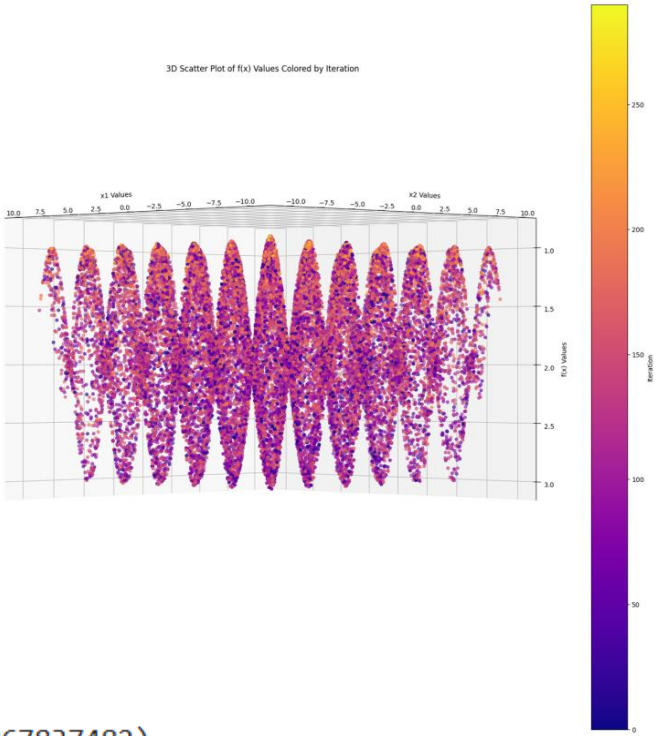
https://drive.google.com/drive/folders/1Ov6lBy0wta2BM_dY4_1NRwt8HpAZsiwY?usp=sharing

Thank you !!!

Multimodal Function

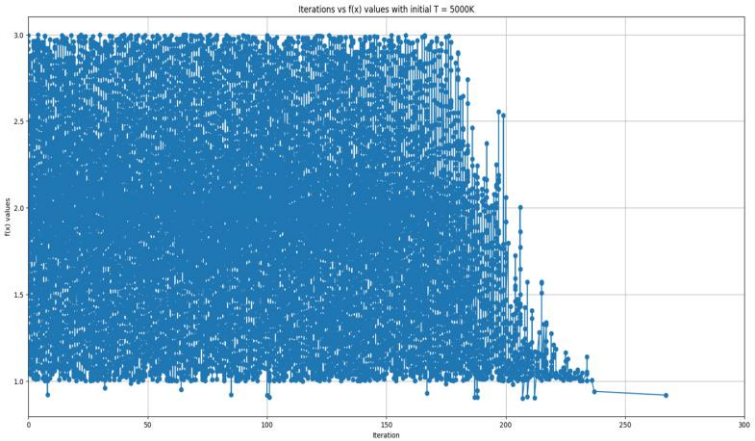
Linear schedule - Lk varying

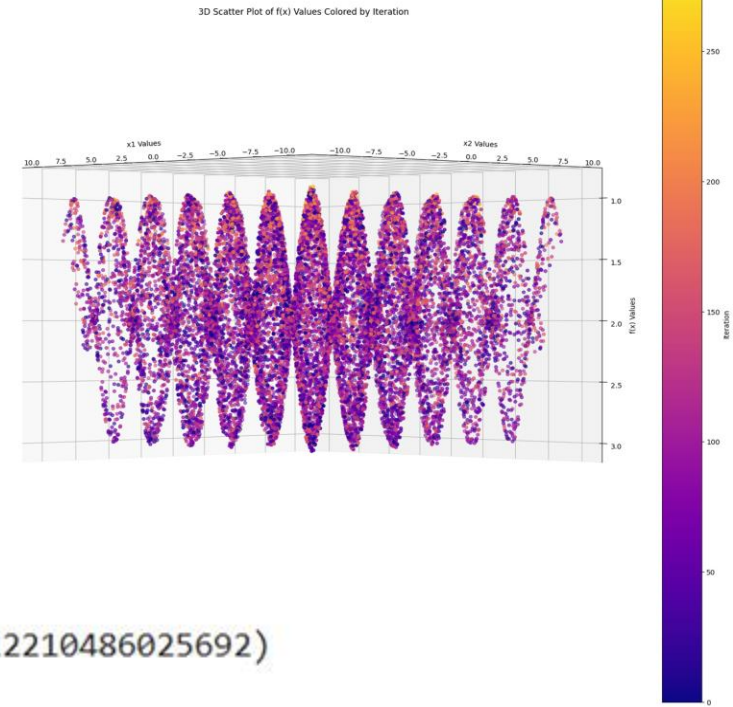





```
Optimal Solution = 0.9017548311485822
Minimum points = (-0.025990475575959238, 0.030336558867837482)
Error = 3.0794323600343736e-06
```

Linear schedule - constant L_k

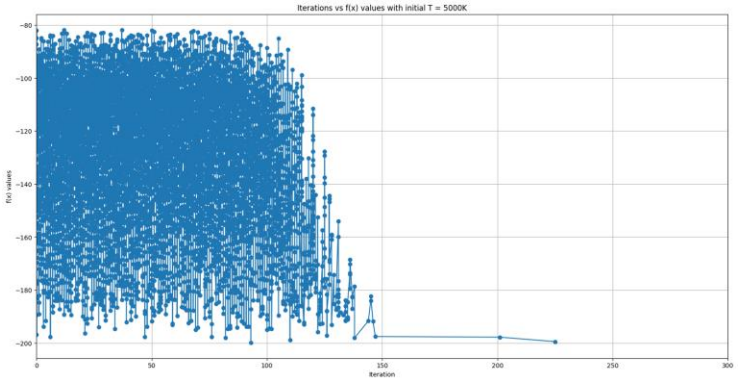


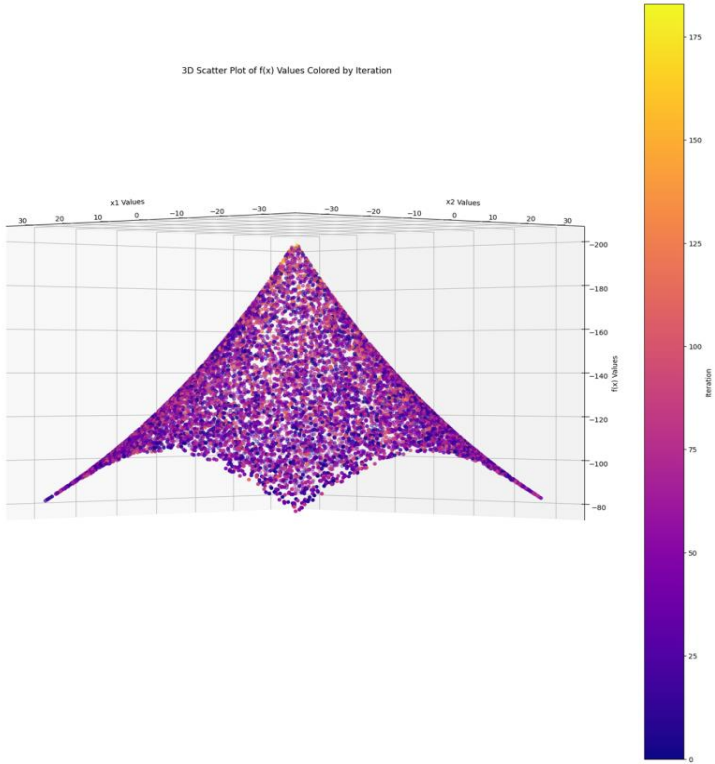



```
Optimal Solution = 0.9181548223144294
Minimum points = (-0.1137750848933905, 0.06022210486025692)
Error = 0.0003295975732685012
```

Unimodal Function

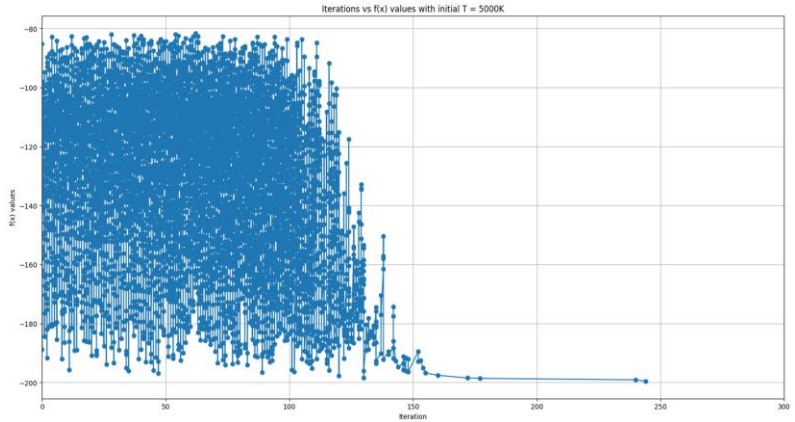
Linear Schedule – constant L_k

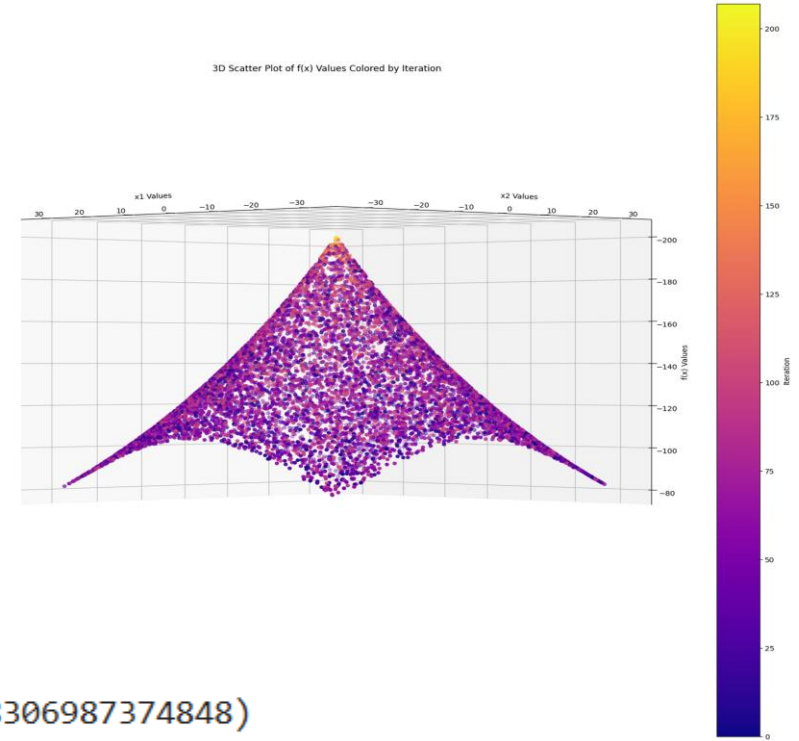





```
Optimal Solution = -199.4877751724607
Minimum points = (-0.044020591514204455, -0.12042706004793757)
Error = 0.26237427394767926
```

Linear Schedule – Varying L_k






```
Optimal Solution = -199.5478460323059
Minimum points = (-0.04418750996718046, 0.10418306987374848)
Error = 0.2044432105015188
```