```
import time
import random
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

matplotlib.rcParams['figure.dpi'] = 200
font = {'family': 'times new roman',
        'weight': 'bold',
        'size': 18}
matplotlib.rc('font', **font)
plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (6, 6)

# ----------------------------------------------------------------------
-------
# Config
initial_temperature = 100
cooling = 0.9
number_variables = 2
upper_bounds = [4, 4]
lower_bounds = [-4, -4]


def objective_function(X):
    x = X[0]
    y = X[1]
    value = 3 * (1 - x) ** 2 * math.exp(-x ** 2 - (y + 1) ** 2) - 10 * (x
/ 5 - x ** 3 - y ** 5) * math.exp(
        -x ** 2 - y ** 2) - (1 / 3) * math.exp(-(x + 1) ** 2 - y ** 2)
    return value


initial_solution = np.array([2., 0.])
current_solution = initial_solution
best_solution = initial_solution
best_fitness = objective_function(best_solution)
current_temperature = initial_temperature  # current temperature
start = time.time()
no_attempts = 9999  # number of attempts in each level of temperature
record_best_fitness = []
record_best_solution = []
record_p = []
is_accept = False

for j in range(no_attempts):
    # make a random move
    prev_solution = current_solution
    for k in range(number_variables):
        current_solution[k] = current_solution[k] + 0.5 *
(random.uniform(-1, 1))
```

```
        current_solution[k] = max(min(current_solution[k],
upper_bounds[k]), lower_bounds[k])

    current_fitness = objective_function(current_solution)
    E = current_fitness - best_fitness

    # ============================================
    #
    # if the current move leads to a less objective value (E<0), accept
it with probability function: math.exp(E / current_temperature)
    # if the current move leads to a larger objective value, accept it.
    #
    # ============================================
    if E < 0:
        p =math.exp(E/current_temperature)

        if random.random()<p:
            is_accept =True
        else:
          is_accept =False

    else:
        is_accept =True

    # update accepted moves so far
    if is_accept is True:
        best_solution = current_solution
        best_fitness = objective_function(best_solution)
        print('iteration: {}, best_solution: {}, best_fitness:
{}'.format(j, best_solution, best_fitness))
        record_best_fitness.append(best_fitness)
        record_best_solution.append([best_solution[0], best_solution[1]])
    else:
        current_solution = prev_solution

    current_temperature = current_temperature * cooling

plt.axis([lower_bounds[0], upper_bounds[0], lower_bounds[1],
upper_bounds[1]])
plt.plot(np.array(record_best_solution)[:, 0],
np.array(record_best_solution)[:, 1], '.-')
plt.plot(np.array(record_best_solution)[:, 0][0],
np.array(record_best_solution)[:, 1][0], c='blue', marker=".",
         markersize=30)
plt.plot(np.array(record_best_solution)[:, 0][-1],
np.array(record_best_solution)[:, 1][-1], c='red', marker=".",
         markersize=30)
plt.scatter(0.00593309, 1.5836536, facecolors='none', edgecolors='black',
linewidth=3, marker="o", s=300)
plt.show()
```