Exercise # 6

Consider the following unconstrained nonlinear optimization problem.

$$Min\ f(x, y) = x^2 + y^2 + z^2 - 6x + 4y - 2z$$
$$where\ x, y, z\ are\ integers\ from\ -10\ to\ 10.$$

1. **Write the program code in python (or in MATLAB or in R) that implements HARMONY SEARCH ALGORITHM to solve the problem.**

Below is the provided code implementing the HARMONY SEARCH ALGORITHM in Python.

```python
import random
import numpy as np

# Objective function
def objective_function(x, y, z):
    """Calculate the objective function value."""
    return x**2 + y**2 + z**2 - 6*x + 4*y - 2*z

# Harmony Search Algorithm
def harmony_search(iterations, harmony_memory_size, pitch_adjust_rate,
bandwidth, search_space):
    """
    Harmony Search Algorithm for unconstrained nonlinear optimization.

    Parameters:
    - iterations: Number of iterations.
    - harmony_memory_size: Size of the harmony memory.
    - pitch_adjust_rate: Probability of adjusting pitch.
    - bandwidth: Pitch adjustment bandwidth.
    - search_space: Tuple defining the search space limits.

    Returns:
    - best_solution: Best solution found.
      - best_value: Objective function value corresponding to the best
solution.
    """
    # Initialize harmony memory
    harmony_memory = []
    for _ in range(harmony_memory_size):
        solution = [random.randint(search_space[0], search_space[1]) for _
in range(3)]
        harmony_memory.append(solution)

    # Main loop
    for _ in range(iterations):
        # Generate a new harmony
        new_harmony = []
        for i in range(3):
            if random.uniform(0, 1) < pitch_adjust_rate:
                        new_harmony.append(random.randint(search_space[0],
search_space[1]))
            else:
                random_index = random.randint(0, harmony_memory_size - 1)
                # Ensure the new harmony is an integer
                       new_value  =  int(harmony_memory[random_index][i]  +
random.uniform(-bandwidth, bandwidth))
                   new_harmony.append(np.clip(new_value,  search_space[0],
search_space[1]))

        # Update harmony memory
        new_harmony_value = objective_function(*new_harmony)
              worst_index  =  np.argmax([objective_function(*h)  for  h  in
harmony_memory])
                                            if      new_harmony_value      <
objective_function(*harmony_memory[worst_index]):
            harmony_memory[worst_index] = new_harmony

    # Return the best solution found
              best_solution    =    min(harmony_memory,    key=lambda    x:
objective_function(*x))
    return best_solution, objective_function(*best_solution)

# Parameters
iterations = 1000
harmony_memory_size = 20
pitch_adjust_rate = 0.5
bandwidth = 1.0
```

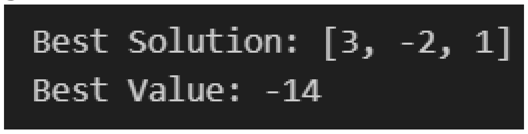
```
search_space = (-10, 10)

# Run Harmony Search Algorithm
best_solution,        best_value        =        harmony_search(iterations,
harmony_memory_size, pitch_adjust_rate, bandwidth, search_space)

# Display the result
print("Best Solution:", best_solution)
print("Best Value:", best_value)
```

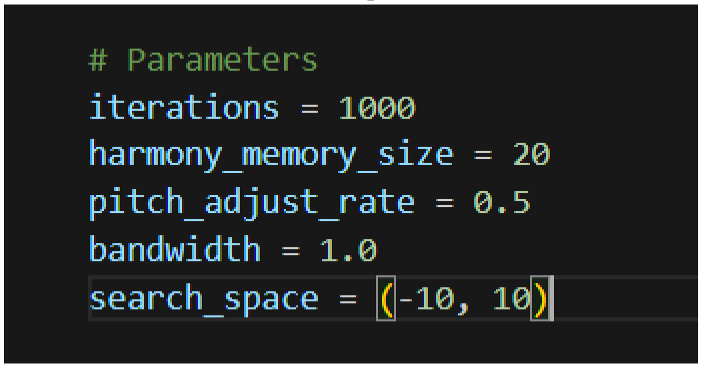
2. **Give your final answer. Indicate the stopping criterion, the definition of neighborhood that you used, and values of your parameters.**

The output that we got is shown below,



```
Best Solution: [3, -2, 1]
Best Value: -14
```

The stopping criterion used in this code is the max number of iterations, which is 1000. The definition of neighborhood here is the search space of the variable which is the set of all integers from -10 to 10. The values of the parameters used are shown below.



```
# Parameters
iterations = 1000
harmony_memory_size = 20
pitch_adjust_rate = 0.5
bandwidth = 1.0
search_space = (-10, 10)
```