Simulated annealing

```
import numpy as np

from scipy.optimize import dual_annealing


def queens_max(position):

    # This function calculates the number of pairs of queens that are not attacking each other

    position = np.round(position).astype(int) # Round and convert to integers for queen positions

    n = len(position)

    queen_not_attacking = 0


    for i in range(n - 1):

        no_attack_on_j = 0

        for j in range(i + 1, n):

            # Check if queens are on the same row or on the same diagonal

            if position[i] != position[j] and abs(position[i] - position[j]) != (j - i):

                no_attack_on_j += 1

        if no_attack_on_j == n - 1 - i:

            queen_not_attacking += 1

    if queen_not_attacking == n - 1:

        queen_not_attacking += 1

    return -queen_not_attacking  # Negative because we want to maximize this value


# Bounds for each queen's position (0 to 7 for an 8x8 chessboard)

bounds = [(0, 8) for _ in range(8)]
```

```
# Use dual_annealing for simulated annealing optimization

result = dual_annealing(queens_max, bounds)


# Display the results

best_position = np.round(result.x).astype(int)

best_objective = -result.fun # Flip sign to get the number of non-attacking queens


print('The best position found is:', best_position)

print('The number of queens that are not attacking each other is:', best_objective)
```

```
The best position found is: [0 8 5 2 6 3 7 4]
The number of queens that are not attacking each other is: 8
```