In [1]:
```python
import numpy as np
```

In [2]:
```python
precision = 4
tol = 1e-12  # Default

f = lambda x: (x[0] - 3) ** 2 + (x[1] + 1) ** 2
x_0 = np.array([0, 0])


def tolist_round(x, precision=precision):
    return np.round(x, precision).tolist()
```

Practice Problem

1

In [3]:
```python
def test_direction_for_one_variable_search(f, x_0, step_size=0.1, tol=tol, outpu
    x_0 = np.array(x_0, dtype=np.float64)
    best_point = x_0
    best_result = f(x_0)

    if output:
        print(f"Original point: {tolist_round(x_0)}, function value: {f(x_0)}")
        print("-" * 10 + "Start searching" + "-" * 10)

    for dim in range(len(x_0)):
        for sign in [-1, 1]:
            test_point = np.array(x_0)
            test_point[dim] += sign * step_size
            test_value = f(test_point)

            if output:
                print(
                    f"Current point: {tolist_round(test_point)}, function value:
                )

            if test_value - best_result < -tol:
                best_result = test_value
                best_point = test_point.copy()

    if output:
        print("-" * 10 + "-End searching-" + "-" * 10)
        print(
            f"Improved point: {tolist_round(best_point)}, function value: {best_
        )
        print(f"Vector between two points: {tolist_round(best_point - x_0)}")

    return best_point - x_0


test_direction_for_one_variable_search(f, x_0)
pass
```

```
Original point: [0.0, 0.0], function value: 10.0
----------Start searching----------
Current point: [-0.1, 0.0], function value: 10.610000000000001
Current point: [0.1, 0.0], function value: 9.41
Current point: [0.0, -0.1], function value: 9.81
Current point: [0.0, 0.1], function value: 10.21
-----------End searching-----------
Improved point: [0.1, 0.0], function value: 9.41
Vector between two points: [0.1, 0.0]
```

2

In [4]:

```
def optimize(f, x_0, delta, alpha=1, tol=tol, output=True):
    x_0 = np.array(x_0, dtype=np.float64)
    best_point = x_0
    best_result = f(x_0)
    iteration = 1

    if output:
        print(
            f"Strat from: {tolist_round(x_0)}, function value: {f(x_0)}, vector:
        )
        print("-" * 10 + "Start optimizing" + "-" * 10)

    while True:
        test_point = best_point + alpha * delta
        test_value = f(test_point)

        if output:
            print(
                f"Iteration No.{iteration}: {tolist_round(test_point)}, function
            )

        if test_value - best_result > tol:
            if output:
                print("-" * 10 + "-End optimizing-" + "-" * 10)
                print(
                    f"Can not be more optimized with the given vector,"
                    f"stop at {tolist_round(best_point)}, function value: {best_
                )

            return best_point, best_result

        best_point = test_point
        best_result = test_value
        iteration += 1


optimize(f, x_0, test_direction_for_one_variable_search(f, x_0, 0.1, output=Fals
pass
```

```
Strat from: [0.0, 0.0], function value: 10.0, vector: [0.1, 0.0]
----------Start optimizing----------
Iteration No.1: [0.1, 0.0], function value: 9.41
Iteration No.2: [0.2, 0.0], function value: 8.84
Iteration No.3: [0.3, 0.0], function value: 8.290000000000001
Iteration No.4: [0.4, 0.0], function value: 7.760000000000001
Iteration No.5: [0.5, 0.0], function value: 7.25
Iteration No.6: [0.6, 0.0], function value: 6.76
Iteration No.7: [0.7, 0.0], function value: 6.289999999999999
Iteration No.8: [0.8, 0.0], function value: 5.840000000000001
Iteration No.9: [0.9, 0.0], function value: 5.41
Iteration No.10: [1.0, 0.0], function value: 5.0
Iteration No.11: [1.1, 0.0], function value: 4.61
Iteration No.12: [1.2, 0.0], function value: 4.24
Iteration No.13: [1.3, 0.0], function value: 3.8899999999999997
Iteration No.14: [1.4, 0.0], function value: 3.5599999999999996
Iteration No.15: [1.5, 0.0], function value: 3.2499999999999996
Iteration No.16: [1.6, 0.0], function value: 2.959999999999999
Iteration No.17: [1.7, 0.0], function value: 2.689999999999999
Iteration No.18: [1.8, 0.0], function value: 2.4399999999999986
Iteration No.19: [1.9, 0.0], function value: 2.2099999999999986
Iteration No.20: [2.0, 0.0], function value: 1.9999999999999991
Iteration No.21: [2.1, 0.0], function value: 1.8099999999999992
Iteration No.22: [2.2, 0.0], function value: 1.639999999999999
Iteration No.23: [2.3, 0.0], function value: 1.4899999999999989
Iteration No.24: [2.4, 0.0], function value: 1.359999999999999
Iteration No.25: [2.5, 0.0], function value: 1.2499999999999991
Iteration No.26: [2.6, 0.0], function value: 1.1599999999999993
Iteration No.27: [2.7, 0.0], function value: 1.0899999999999994
Iteration No.28: [2.8, 0.0], function value: 1.0399999999999996
Iteration No.29: [2.9, 0.0], function value: 1.0099999999999998
Iteration No.30: [3.0, 0.0], function value: 1.0
Iteration No.31: [3.1, 0.0], function value: 1.0100000000000002
-----------End optimizing-----------
Can not be more optimized with the given vector,stop at [3.0, 0.0], function valu
e: 1.0
```

3

1. Step 1: Start from an initial point $x_0^1 = (x_0, y_0)$ and a fixed variation $\pm\delta$ (for instance $\delta = 0.1$), and move parallel to Ox_1. Find the values of f at $(x_0 + \delta, y_0)$ and $(x_0 - \delta, y_0)$ and compare. Save the best value and call the new point x_1^1.
2. Step 2: Start from the new point and move parallel to Ox_2 with a variation $\pm\delta$. If there is some improvement the new point is x_2^1. It is supposed that at least one of the two values gives a better result than the initial point x_0^1.
3. Step 3: The accelaration is introduced here. Define the vector formed by the two points x_0^1 and x_2^1, called v and define a new point x_0^2

$$x_0^2 = x_0^1 + \alpha v = x_0^1 + \alpha(x_2^1 - x_0^1)$$

with recommended factor for $\alpha = 2$. Search in that direction until you stop seeing improvement.
4. Step 4: When you stop seeing improvement in the direction of the vector v repeat steps 1 and 2. When step 1 gives back the original point, you move to step 5:
5. Step 5: Decrease the incremental interval. If you started by adding and subtracting 0.1, now you might add and subtract 0.01. Then repeat everything until you have the desired level of accuracy. The stopping criterion can be determined by the user and could be $||\delta|| < \epsilon$, $\epsilon > 0$ a desired accuracy level.

In [5]:
```
def test_direction_for_hooke_jeeves(f, x_0, step_size=0.1, tol=tol, output=True)
    x_0 = np.array(x_0, dtype=np.float64)
    best_point = x_0
    best_result = f(x_0)

    if output:
        print(
            f"Original point: {tolist_round(x_0)}, function value: {f(x_0)}"
        )
        print("-" * 10 + "Start searching" + "-" * 10)

    for dim in range(len(x_0)):
        test_points = [best_point.copy(), best_point.copy()]
        test_points[0][dim] += step_size
        test_points[1][dim] -= step_size
        test_values = [f(test_points[0]), f(test_points[1])]

        if min(test_values) - best_result < -tol:
            if test_values[0] < test_values[1]:
                best_point = test_points[0]
                best_result = test_values[0]
            else:
                best_point = test_points[1]
                best_result = test_values[1]

            if output:
                print(
                    f"Current dimension: {dim+1}, better point: {tolist_round(be
                )

        else:
            if output:
                print(f"Current dimension: {dim+1}, no better point")
```

```
    if output:
        print("-" * 10 + "-End searching-" + "-" * 10)
        print(
            f"Improved point: {tolist_round(best_point)}, function value: {best_
        )
        print(f"Vector between two points: {tolist_round(best_point - x_0)}")

    return best_point - x_0


test_direction_for_hooke_jeeves(f, x_0)
pass
```

```
Original point: [0.0, 0.0], function value: 10.0
----------Start searching----------
Current dimension: 1, better point: [0.1, 0.0]
Current dimension: 2, better point: [0.1, -0.1]
-----------End searching-----------
Improved point: [0.1, -0.1], function value: 9.22
Vector between two points: [0.1, -0.1]
```

In [6]:

```
def hooke_jeeves(f, x_0, step_size=0.1, alpha=2, tol=tol, output=True):
    x_0 = np.array(x_0, dtype=np.float64)
    best_point = x_0
    best_result = f(x_0)
    iteration = 1

    if output:
        print(
            f"Starting Hooke-Jeeves optimization from: {tolist_round(x_0)}, func
        )
        print("-" * 10 + "Start Hooke-Jeeves optimization" + "-" * 10)

    while True:
        # Step 1 & 2 & 4'
        direction = test_direction_for_hooke_jeeves(
            f, best_point, step_size, tol=tol, output=False
        )

        # Step 5
        if (
            np.linalg.norm(direction) < tol
            # Avoid "side-to-side hopping" between two points at the same elevat
            or f(best_point + alpha * direction) - best_result > -tol
        ):
            step_size /= 10
            if step_size < tol:
                if output:
                    print(f"No better direction, step size smaller than toleranc

                break

            if output:
                print(f"No better direction, try reducing step size to {step_siz

            continue

        # Step 3 & 4'
        best_point, best_result = optimize(f, best_point, direction, alpha, tol,
```

```python
        if output:
            print(
                f"Iteration {iteration}: "
                f"improved point: {tolist_round(best_point)}, function value: {b
            )

        iteration += 1

    if output:
        print("-" * 10 + "-End Hooke-Jeeves optimization-" + "-" * 10)
        print(
            f"Search method completed at {tolist_round(best_point)} with functic
        )

    return tolist_round(best_point), best_result


hooke_jeeves(f, x_0, step_size=0.1, alpha=2)
pass
```

```
Starting Hooke-Jeeves optimization from: [0.0, 0.0], function value: 10.0
----------Start Hooke-Jeeves optimization----------
Iteration 1: improved point: [2.0, -2.0], function value: 2.0
Iteration 2: improved point: [3.0, -1.0], function value: 4.388038785291878e-30
No better direction, try reducing step size to 0.01
No better direction, try reducing step size to 0.001
No better direction, try reducing step size to 0.0001
No better direction, try reducing step size to 1e-05
No better direction, try reducing step size to 1.0000000000000002e-06
No better direction, try reducing step size to 1.0000000000000002e-07
No better direction, try reducing step size to 1.0000000000000002e-08
No better direction, try reducing step size to 1.0000000000000003e-09
No better direction, try reducing step size to 1.0000000000000003e-10
No better direction, try reducing step size to 1.0000000000000003e-11
No better direction, try reducing step size to 1.0000000000000002e-12
No better direction, step size smaller than tolerance
-----------End Hooke-Jeeves optimization-----------
Search method completed at [3.0, -1.0] with function value: 4.388038785291878e-30
```

Some tests

In [7]:
```python
fs = [
    # f0
    lambda x: (x[0] - 1) ** 2 + (x[1] - 1) ** 2 + (x[2] - 1) ** 2,
    # f1
    lambda x: (x[0] + 2) ** 2 + (x[1] - 2) ** 2 + (x[2] + 2) ** 2 + (x[3] - 2) *
    # f2
    lambda x: (x[0] - 1) ** 2
    + (x[1] - 2.1) ** 4
    + (x[2] - 3.22) ** 4
    + (x[3] - 4.333) ** 2
    + (x[4] - 5.4444) ** 4
    + (x[5] - 6.1234) ** 4
    + (x[6] - 7.5678) ** 4,
]
```

Different functions

In [8]:
```python
hooke_jeeves(fs[0], np.zeros(3), step_size=1, alpha=2)
pass
```

```
Starting Hooke-Jeeves optimization from: [0.0, 0.0, 0.0], function value: 3.0
----------Start Hooke-Jeeves optimization----------
No better direction, try reducing step size to 0.1
Iteration 1: improved point: [1.0, 1.0, 1.0], function value: 0.0
No better direction, try reducing step size to 0.01
No better direction, try reducing step size to 0.001
No better direction, try reducing step size to 0.0001
No better direction, try reducing step size to 1e-05
No better direction, try reducing step size to 1.0000000000000002e-06
No better direction, try reducing step size to 1.0000000000000002e-07
No better direction, try reducing step size to 1.0000000000000002e-08
No better direction, try reducing step size to 1.0000000000000003e-09
No better direction, try reducing step size to 1.0000000000000003e-10
No better direction, try reducing step size to 1.0000000000000003e-11
No better direction, try reducing step size to 1.0000000000000002e-12
No better direction, step size smaller than tolerance
-----------End Hooke-Jeeves optimization-----------
Search method completed at [1.0, 1.0, 1.0] with function value: 0.0
```

In [9]:
```python
hooke_jeeves(fs[1], np.zeros(4), step_size=1, alpha=2)
pass
```

```
Starting Hooke-Jeeves optimization from: [0.0, 0.0, 0.0, 0.0], function value: 1
6.0
----------Start Hooke-Jeeves optimization----------
Iteration 1: improved point: [-2.0, 2.0, -2.0, 2.0], function value: 0.0
No better direction, try reducing step size to 0.1
No better direction, try reducing step size to 0.01
No better direction, try reducing step size to 0.001
No better direction, try reducing step size to 0.0001
No better direction, try reducing step size to 1e-05
No better direction, try reducing step size to 1.0000000000000002e-06
No better direction, try reducing step size to 1.0000000000000002e-07
No better direction, try reducing step size to 1.0000000000000002e-08
No better direction, try reducing step size to 1.0000000000000003e-09
No better direction, try reducing step size to 1.0000000000000003e-10
No better direction, try reducing step size to 1.0000000000000003e-11
No better direction, try reducing step size to 1.0000000000000002e-12
No better direction, step size smaller than tolerance
-----------End Hooke-Jeeves optimization-----------
Search method completed at [-2.0, 2.0, -2.0, 2.0] with function value: 0.0
```

Working as expected!

Different tolerances

In [10]:
```python
hooke_jeeves(fs[2], np.zeros(7), step_size=0.1, tol=1e-4, alpha=2)
pass
```

```
Starting Hooke-Jeeves optimization from: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], fun
ction value: 5711.332604988733
----------Start Hooke-Jeeves optimization----------
Iteration 1: improved point: [4.8, 4.8, 4.8, 4.8, 4.8, 4.8, 4.8], function value:
135.9605765399642
Iteration 2: improved point: [3.0, 3.0, 3.0, 3.0, 6.6, 6.6, 6.6], function value:
9.14753964447621
Iteration 3: improved point: [2.0, 2.0, 4.0, 4.0, 5.6, 5.6, 7.6], function value:
1.5567740906202143
Iteration 4: improved point: [1.4, 2.0, 3.4, 4.6, 5.0, 6.2, 7.6], function value:
0.27147710057219737
Iteration 5: improved point: [1.0, 2.0, 3.0, 4.2, 5.4, 6.2, 7.6], function value:
0.020170949557786307
Iteration 6: improved point: [1.0, 2.0, 3.2, 4.4, 5.4, 6.2, 7.6], function value:
0.004628549557789739
No better direction, try reducing step size to 0.01
Iteration 7: improved point: [1.0, 2.0, 3.2, 4.34, 5.4, 6.2, 7.6], function valu
e: 0.00018854955778888408
No better direction, try reducing step size to 0.001
No better direction, try reducing step size to 0.0001
No better direction, step size smaller than tolerance
-----------End Hooke-Jeeves optimization-----------
Search method completed at [1.0, 2.0, 3.2, 4.34, 5.4, 6.2, 7.6] with function val
ue: 0.00018854955778888408
```

In [11]:

```python
hooke_jeeves(fs[2], np.zeros(7), step_size=0.1, tol=1e-10, alpha=2)
pass
```

```
Starting Hooke-Jeeves optimization from: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], fun
ction value: 5711.332604988733
----------Start Hooke-Jeeves optimization----------
Iteration 1: improved point: [4.8, 4.8, 4.8, 4.8, 4.8, 4.8, 4.8], function value:
135.9605765399642
Iteration 2: improved point: [3.0, 3.0, 3.0, 3.0, 6.6, 6.6, 6.6], function value:
9.14753964447621
Iteration 3: improved point: [2.0, 2.0, 4.0, 4.0, 5.6, 5.6, 7.6], function value:
1.5567740906202143
Iteration 4: improved point: [1.4, 2.6, 3.4, 4.6, 5.0, 6.2, 7.6], function value:
0.33387710057220116
Iteration 5: improved point: [1.0, 2.2, 3.0, 4.2, 5.4, 5.8, 7.6], function value:
0.031075079004186663
Iteration 6: improved point: [1.0, 2.0, 3.2, 4.4, 5.4, 6.0, 7.6], function value:
0.004825999880989767
No better direction, try reducing step size to 0.01
Iteration 7: improved point: [1.0, 2.06, 3.26, 4.34, 5.46, 6.06, 7.54], function
value: 7.09333683489248e-05
No better direction, try reducing step size to 0.001
Iteration 8: improved point: [1.0, 2.068, 3.252, 4.332, 5.452, 6.068, 7.548], fun
ction value: 1.2673926684789707e-05
Iteration 9: improved point: [1.0, 2.07, 3.25, 4.334, 5.45, 6.07, 7.55], function
value: 1.0852765468814001e-05
Iteration 10: improved point: [1.0, 2.072, 3.248, 4.332, 5.448, 6.072, 7.552], fu
nction value: 9.271752732788659e-06
Iteration 11: improved point: [1.0, 2.074, 3.246, 4.334, 5.446, 6.074, 7.554], fu
nction value: 7.905582876813145e-06
Iteration 12: improved point: [1.0, 2.076, 3.244, 4.332, 5.446, 6.076, 7.556], fu
nction value: 6.7308768287879715e-06
Iteration 13: improved point: [1.0, 2.078, 3.242, 4.334, 5.446, 6.078, 7.558], fu
nction value: 5.726122780812608e-06
Iteration 14: improved point: [1.0, 2.08, 3.24, 4.332, 5.446, 6.08, 7.56], functi
on value: 4.87150633278757e-06
Iteration 15: improved point: [1.0, 2.082, 3.238, 4.334, 5.446, 6.082, 7.562], fu
nction value: 4.148749084812322e-06
Iteration 16: improved point: [1.0, 2.084, 3.236, 4.332, 5.446, 6.084, 7.564], fu
nction value: 3.5411086367873826e-06
Iteration 17: improved point: [1.0, 2.086, 3.234, 4.334, 5.446, 6.086, 7.566], fu
nction value: 3.0333785888122206e-06
Iteration 18: improved point: [1.0, 2.088, 3.232, 4.332, 5.446, 6.088, 7.566], fu
nction value: 2.6118990367873546e-06
Iteration 19: improved point: [1.0, 2.09, 3.23, 4.334, 5.446, 6.09, 7.566], funct
ion value: 2.2644911648122536e-06
Iteration 20: improved point: [1.0, 2.092, 3.228, 4.332, 5.446, 6.092, 7.566], fu
nction value: 1.9803261727874365e-06
Iteration 21: improved point: [1.0, 2.094, 3.226, 4.334, 5.446, 6.094, 7.566], fu
nction value: 1.7497272608123742e-06
Iteration 22: improved point: [1.0, 2.096, 3.224, 4.332, 5.446, 6.096, 7.566], fu
nction value: 1.5641696287875854e-06
Iteration 23: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.098, 7.566], fu
nction value: 1.4162804768125407e-06
Iteration 24: improved point: [1.0, 2.098, 3.222, 4.332, 5.446, 6.1, 7.566], func
tion value: 1.29987100478776e-06
Iteration 25: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.102, 7.566], fu
nction value: 1.2097764128127142e-06
Iteration 26: improved point: [1.0, 2.098, 3.222, 4.332, 5.446, 6.104, 7.566], fu
nction value: 1.1416959007879246e-06
Iteration 27: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.106, 7.566], fu
nction value: 1.0917126688128656e-06
Iteration 28: improved point: [1.0, 2.098, 3.222, 4.332, 5.446, 6.108, 7.566], fu
```

```
nction value: 1.0562939167880595e-06
Iteration 29: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.11, 7.566], fun
ction value: 1.0322908448129815e-06
Iteration 30: improved point: [1.0, 2.098, 3.222, 4.332, 5.446, 6.112, 7.566], fu
nction value: 1.016938652788155e-06
Iteration 31: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.114, 7.566], fu
nction value: 1.0078565408130564e-06
Iteration 32: improved point: [1.0, 2.098, 3.222, 4.332, 5.446, 6.116, 7.566], fu
nction value: 1.00304770878821e-06
Iteration 33: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.118, 7.566], fu
nction value: 1.000899356813093e-06
Iteration 34: improved point: [1.0, 2.098, 3.222, 4.332, 5.446, 6.12, 7.566], fun
ction value: 1.000182684788231e-06
Iteration 35: improved point: [1.0, 2.098, 3.222, 4.334, 5.446, 6.122, 7.566], fu
nction value: 1.0000528928131025e-06
No better direction, try reducing step size to 0.0001
Iteration 36: improved point: [1.0, 2.098, 3.222, 4.333, 5.446, 6.122, 7.566], fu
nction value: 5.289279999997604e-11
No better direction, try reducing step size to 1e-05
No better direction, try reducing step size to 1.0000000000000002e-06
No better direction, try reducing step size to 1.0000000000000002e-07
No better direction, try reducing step size to 1.0000000000000002e-08
No better direction, try reducing step size to 1.0000000000000003e-09
No better direction, try reducing step size to 1.0000000000000003e-10
No better direction, step size smaller than tolerance
-----------End Hooke-Jeeves optimization-----------
Search method completed at [1.0, 2.098, 3.222, 4.333, 5.446, 6.122, 7.566] with f
unction value: 5.289279999997604e-11
```

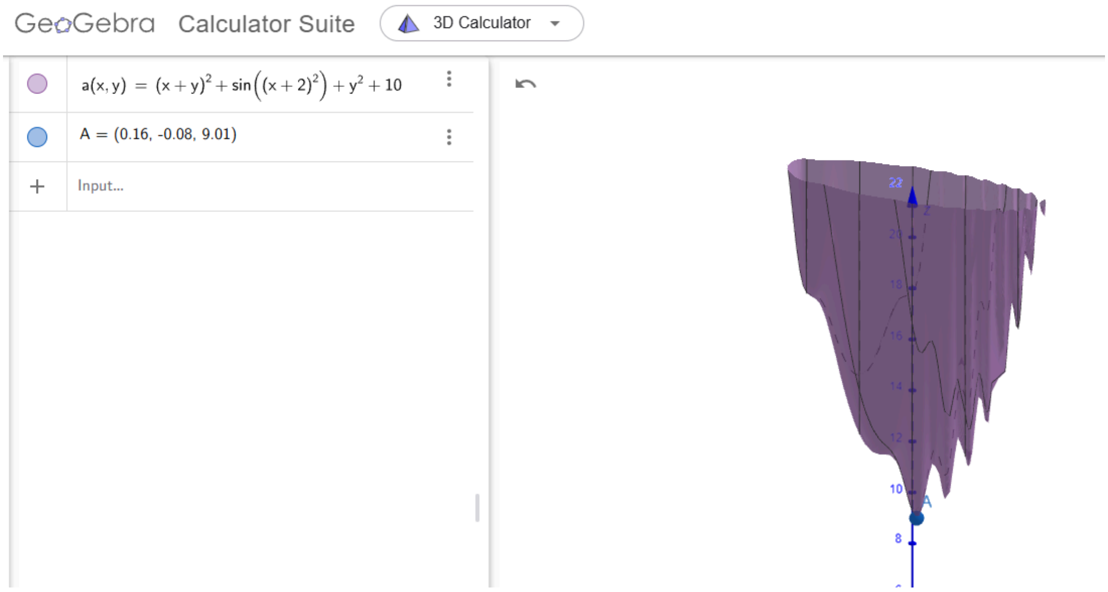
Greater accuracy!

4

In [12]:
```python
f4 = lambda x: (x[0] + x[1]) ** 2 + np.sin((x[0] + 2) ** 2) + x[1] ** 2 + 10

hooke_jeeves(f4, np.zeros(2), step_size=1, tol=1e-6, alpha=2)
pass
```

```
Starting Hooke-Jeeves optimization from: [0.0, 0.0], function value: 9.2431975046
92072
----------Start Hooke-Jeeves optimization----------
No better direction, try reducing step size to 0.1
Iteration 1: improved point: [0.2, 0.0], function value: 9.048131242689088
No better direction, try reducing step size to 0.01
Iteration 2: improved point: [0.16, -0.04], function value: 9.01709440466418
Iteration 3: improved point: [0.16, -0.08], function value: 9.01389440466418
No better direction, try reducing step size to 0.001
Iteration 4: improved point: [0.162, -0.08], function value: 9.013851431554384
No better direction, try reducing step size to 0.0001
No better direction, try reducing step size to 1e-05
No better direction, try reducing step size to 1.0000000000000002e-06
No better direction, step size smaller than tolerance
-----------End Hooke-Jeeves optimization-----------
Search method completed at [0.162, -0.08] with function value: 9.013851431554384
```



Seems to be correct!