In [13]:
```
from numpy import arange
from numpy import meshgrid
from matplotlib import pyplot
from math import sqrt
from numpy import asarray
from numpy.random import rand
from numpy.random import seed
from numpy import arange
from numpy import meshgrid
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```

In [60]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the function to optimize
def f(x, y):
    return x**2 + y**2

# Define the gradient function
def grad_f(x, y):
    return np.array([2*x, 2*y])

# Define the RMSprop function
def rmsprop(x_init, y_init, learning_rate, gamma, num_iterations):
    # Initialize variables
    x = x_init
    y = y_init
    epsilon = 1e-8
    grad_squared_sum = np.zeros((2, 2))
    x_path = [x]
    y_path = [y]
    f_path = [f(x, y)]

    # Run RMSprop
    for i in range(num_iterations):
        grad = grad_f(x, y)
        grad_squared_sum = gamma * grad_squared_sum + (1 - gamma) * np.outer(gra
        x -= learning_rate * grad[0] / (np.sqrt(grad_squared_sum[0, 0]) + epsilo
        y -= learning_rate * grad[1] / (np.sqrt(grad_squared_sum[1, 1]) + epsilo
        x_path.append(x)
        y_path.append(y)
        f_path.append(f(x, y))

    return (x, y, f(x, y), x_path, y_path, f_path)
```

In [79]:
```
x_init = -4
y_init = 4
learning_rate = 0.1
gamma = 0.9
num_iterations = 50

# Run RMSprop
x_final, y_final, f_final, x_path, y_path, f_path = rmsprop(x_init, y_init, lear

# Print the optimized values
print("Optimized values:")
```

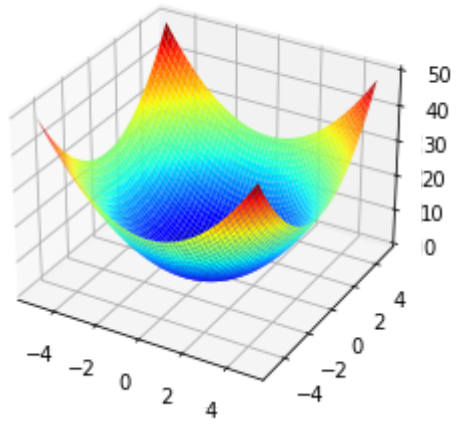
```
print("x =", x_final)
print("y =", y_final)
print("f(x, y) =", f_final)
# print("x_Path = ",type(x_path))
# print("y_Path = ",type(y_path))
# print("f_Path = ",type(f_path))

# Plot the optimization path
x_range = np.arange(-5, 5, 0.1)
y_range = np.arange(-5, 5, 0.1)
X, Y = np.meshgrid(x_range, y_range)
Z = f(X, Y)
```

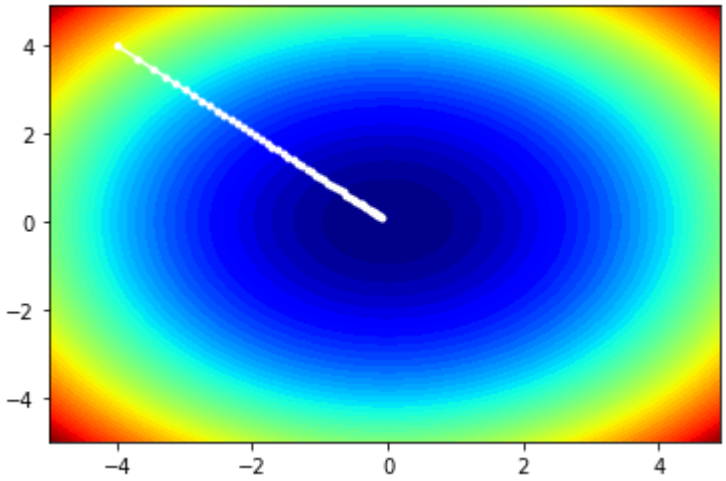
```
Optimized values:
x = -0.10352260641407904
y = 0.10352260641407904
f(x, y) = 0.021433860077528635
```

In [80]:
```
figure = pyplot.figure()
axis = figure.gca(projection='3d')
axis.plot_surface(X, Y, Z, cmap='jet')
# show the plot
pyplot.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_5620\4245877465.py:2: MatplotlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().
  axis = figure.gca(projection='3d')
```

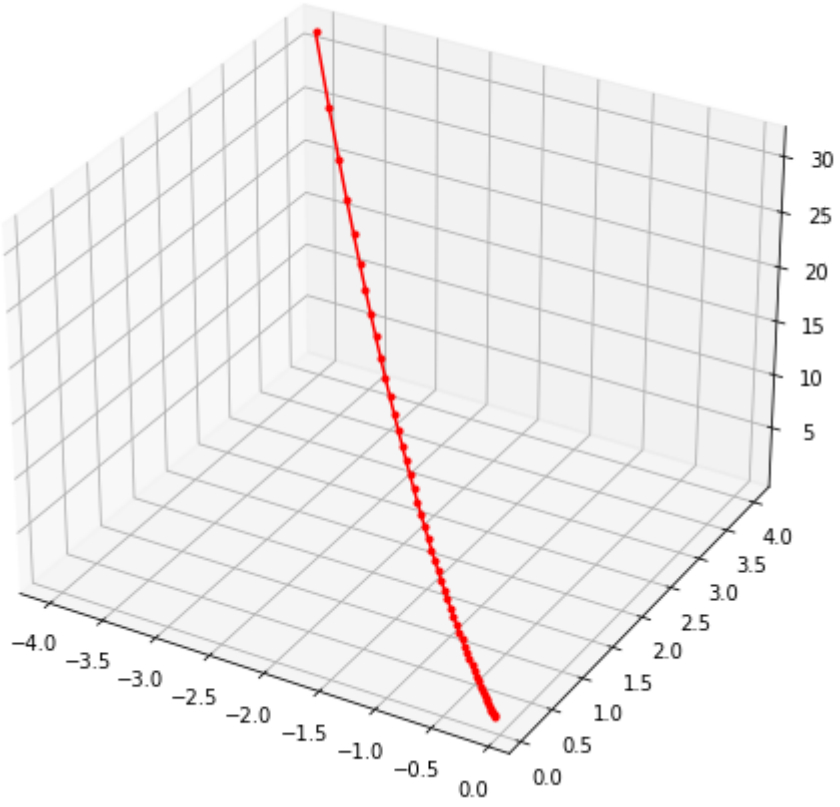


In [81]:
```
pyplot.contourf(X, Y, Z, levels=50, cmap='jet')
pyplot.plot(x_path,y_path, '.-', color='w')
pyplot.show()
```



In [82]:
```
# print(score)
figure =  pyplot.figure(figsize=(10, 8))
axis = figure.gca(projection='3d')
# axis.plot_surface(x, y, results, cmap='jet')
# solutions = asarray(solutions)
pyplot.plot(x_path,y_path,f_path, '.-', color='r')
pyplot.show()
```

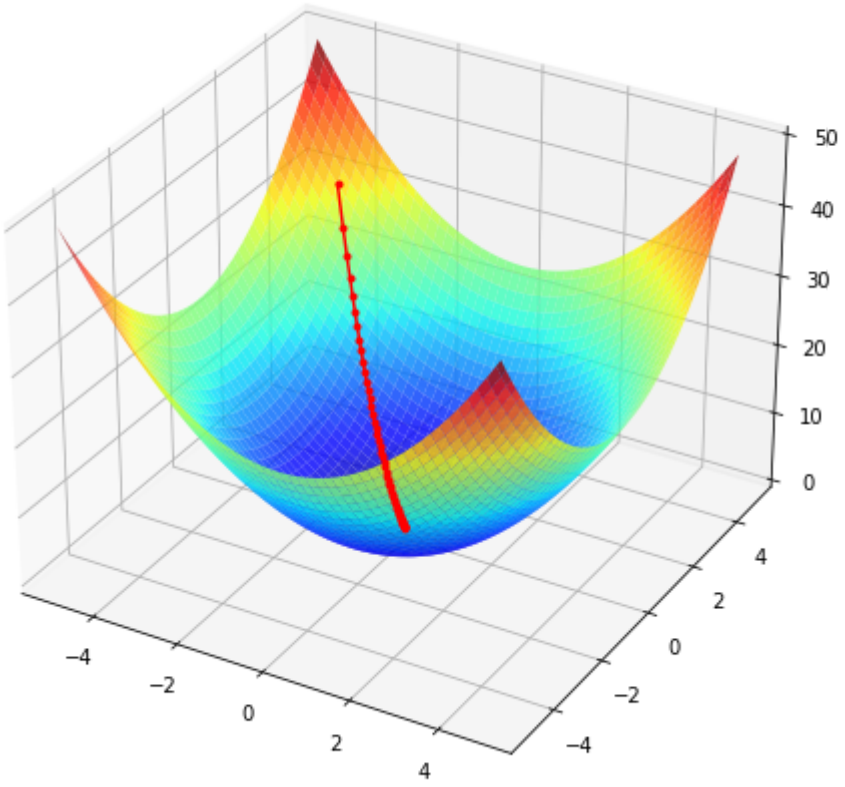
```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_5620\2492270181.py:3: MatplotlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().
  axis = figure.gca(projection='3d')
```



In [83]:
```
figure =  pyplot.figure(figsize=(10, 8))
axis = figure.gca(projection='3d')
```

```
axis.plot_surface(X, Y,Z, cmap='jet', alpha=0.8)
# solutions = asarray(solutions)
pyplot.plot(x_path,y_path,f_path, '.-', color='r', zorder=10)
pyplot.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_5620\3187885447.py:2: MatplotlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().
  axis = figure.gca(projection='3d')
```



In [ ]:

In [ ]: