In [ ]:

```python
from utils import *
```

Black box optimization

Let $xxxxxxxx$ be your student Id. The three functions

$$f_{xxxxxxxx,i} : \mathbb{R}^3 \to \mathbb{R}$$

$i \in \{1,2,3\}$ are given in a black-box setting.

The archive hw3_executables.zip (contained in the HW3 dropbox) contains three (equivalent) command line executables hw3_nix, hw3_mac, and hw3_win.exe, running on three different OS families. Each executable expects five parameters on the standard input and, after a time consuming computation, outputs a real number to the standard output. Below is a sample call. Your student Id is the first parameter $xxxxxxxx$, the second parameter is an integer $i \in \{1,2,3\}$, and the remaining three parameters are real numbers.

Try to test the appropriate executable as fast as possible. Please report any problems should the above program not work as intended.

14. Find minima of functions

$$f_{xxxxxxx,1}, \quad f_{xxxxxxx,2}, \quad f_{xxxxxxxx,3}.$$

Use sufficiently high precision (ten significant digits or more).

These problems are in theory unconstrained, you are guaranteed that none of the calls results in an overflow if real parameters lie in $[-10, 10]$.

One step beyond: How would one use a gradient-descent based method in such a case. Which one is best suitable. Can you beat Nelder-Mead?

In [ ]:

```python
f1_x0 = np.array([1,1,1])
f2_x0 = np.array([1,1,0])
f3_x0 = np.array([0,1,1])
diameters = ([0.25])
v = 0.9
T = 20
```

In [ ]:

```python
### only run once to save the results

f1_steps = nelder_mead(black_box1, f1_x0, diameters, T)
np.save('results/f1_steps.npy', f1_steps)
f2_steps = nelder_mead(black_box2, f2_x0, diameters, T)
np.save('results/f2_steps.npy', f2_steps)
f3_steps = nelder_mead(black_box3, f3_x0, diameters, T)
np.save('results/f3_steps.npy', f3_steps)
```

In [ ]:

```python
f1_steps = np.load('results/f1_steps.npy')
f2_steps = np.load('results/f2_steps.npy')
f3_steps = np.load('results/f3_steps.npy')
```

```
optimizing_functions = ['Gradient descent', 'Polyak GD', 'Nesterov GD', 'AdaGrad', '

descent_steps_1 = {}
descent_steps_2 = {}
descent_steps_3 = {}

for of in optimizing_functions:
    descent_steps_1[of] = np.load('results/' + of + '_blackbox_1.npy')
    descent_steps_2[of] = np.load('results/' + of + '_blackbox_2.npy')
    descent_steps_3[of] = np.load('results/' + of + '_blackbox_3.npy')
```

In [ ]:

```
f1_min = f1_steps[-1][0]
gammas = np.array([[10e-2, 10e-3, 10e-3, 10e-2]])
```

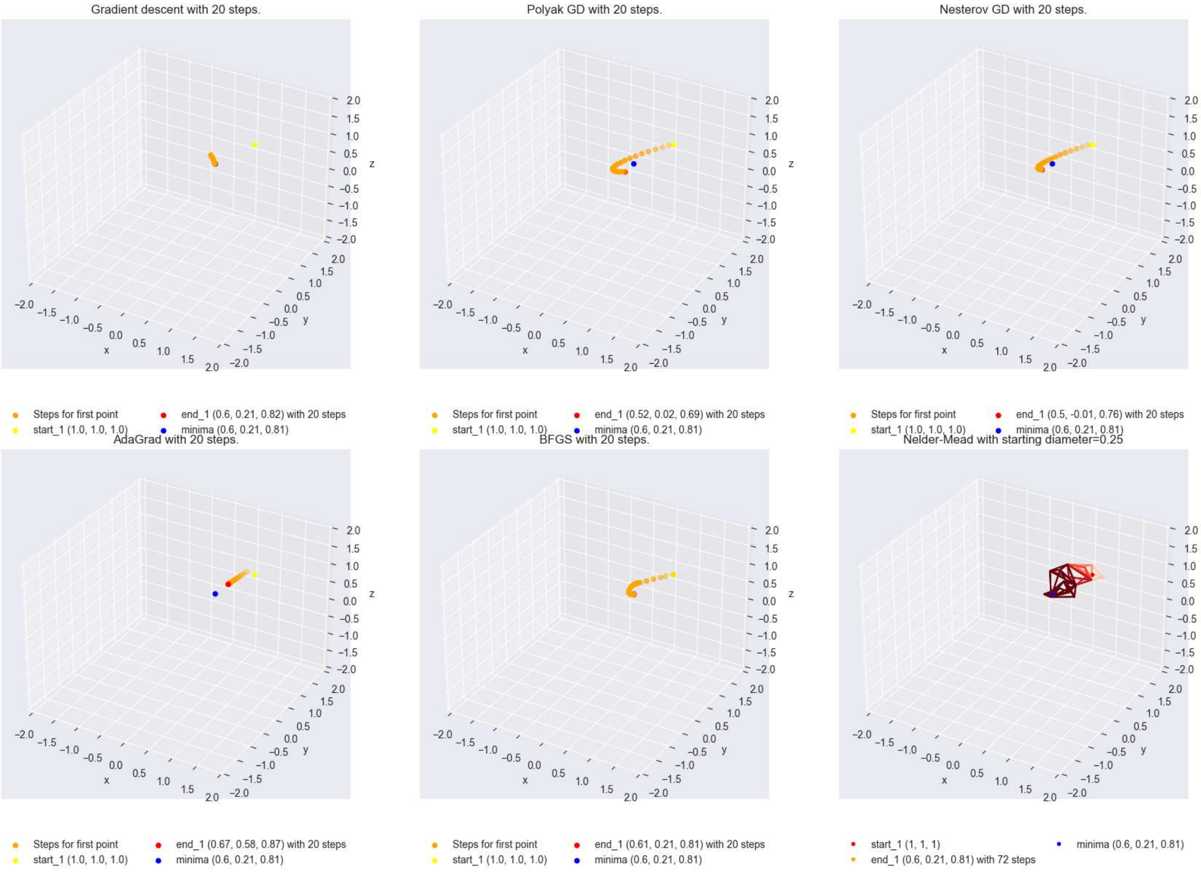
In [ ]:

```
print(f"Lowest value of f1 obtained by Nelder-Mead method: {black_box1(f1_min)} on p
plot_all(f1_x0, None, f1_min, gammas, v, T, None, optimizing_functions, black_box1,
```

```
Lowest value of f1 obtained by Nelder-Mead method: 0.602081360021159 on point: [0.60
200122 0.20809675 0.81359698]

Difference between final function value and obtained nelder-mead minimum value:

Gradient descent for starting point (1.0, 1.0, 1.0): 1.6452288980062235e-05
Polyak GD for starting point (1.0, 1.0, 1.0): 0.05954874519446507
Nesterov GD for starting point (1.0, 1.0, 1.0): 0.06567454693738706
AdaGrad for starting point (1.0, 1.0, 1.0): 0.17716383203017005
BFGS for starting point (1.0, 1.0, 1.0): 5.4478349330033815e-05
```



Gradient descent with 20 steps.

Polyak GD with 20 steps.

Nesterov GD with 20 steps.

AdaGrad with 20 steps.

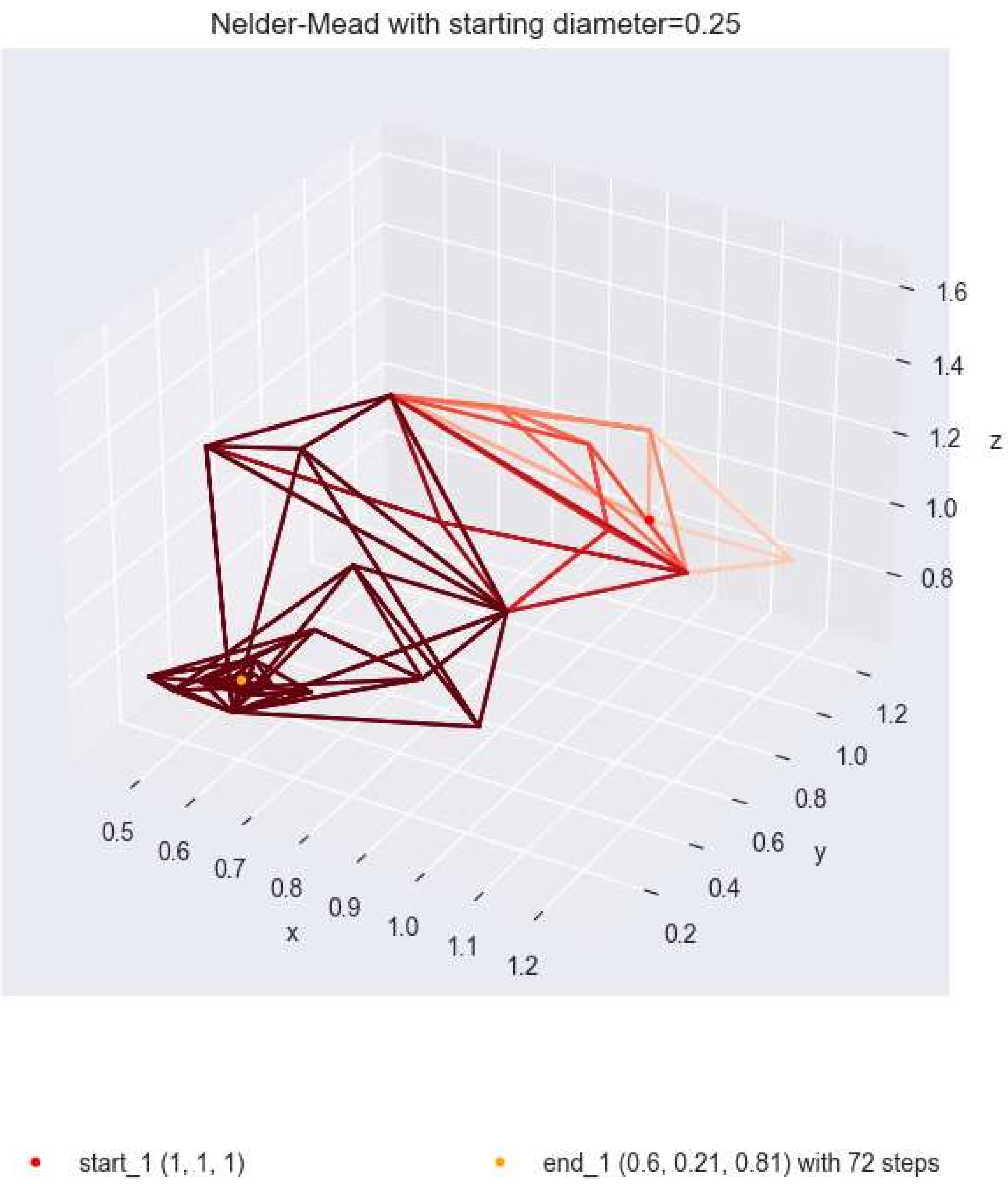
BFGS with 20 steps.

Nelder-Mead with starting diameter=0.25

We can observe that normal gradient descent and BFGS came really close to Nelder-Mead method in only 20 steps. The difference was only in 5 significant digits. \ A close up of Nelder-Mead method with 72 steps:

In [ ]:

```
plot_nelder_mead_3D(f1_x0, None, None, black_box1, diameters, T, None, steps=f1_step
```



In [ ]:

```
f2_min = f2_steps[-1][0]
gammas = np.array([[10e-4, 10e-4, 10e-4, 10e-4]])
```

In [ ]:

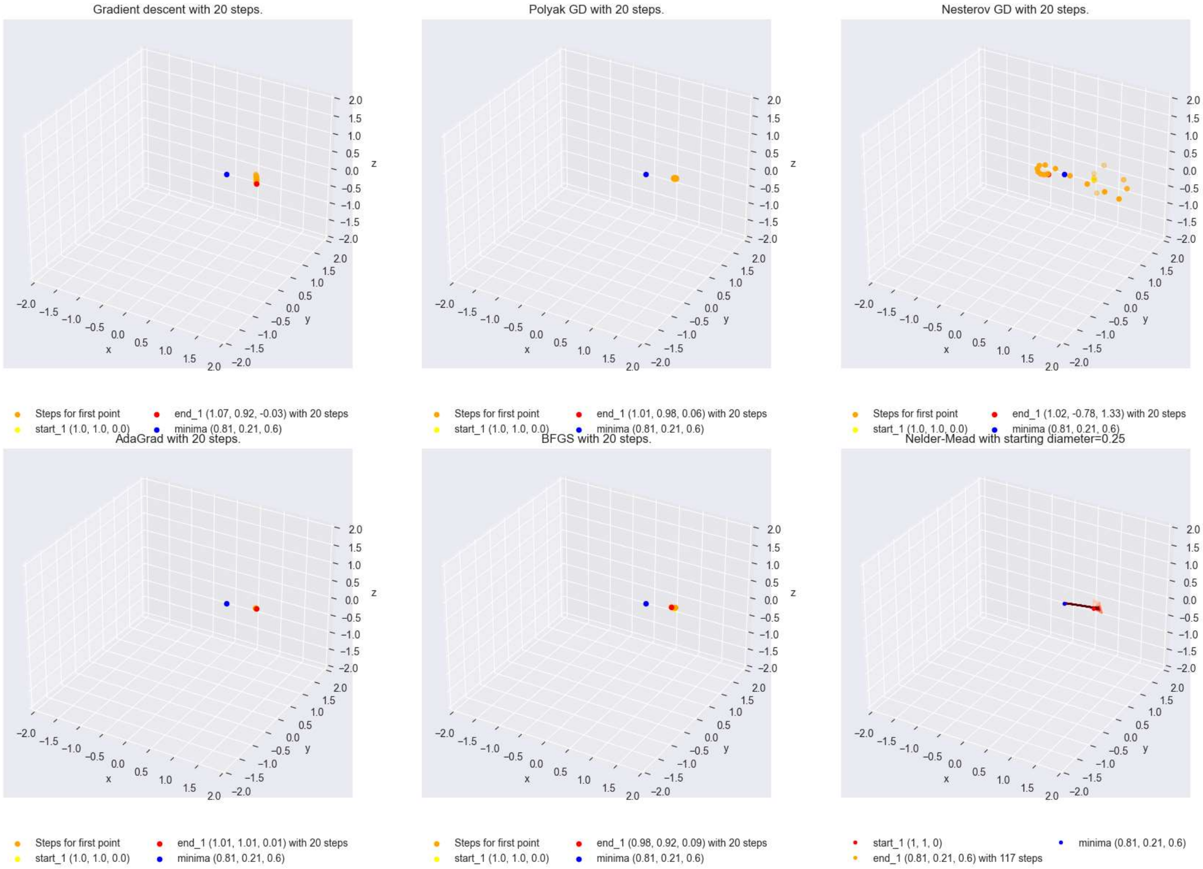
```
print(f"Lowest value of f2 obtained by Nelder-Mead method: {black_box2(f2_min)} on p
plot_all(f2_x0, None, f2_min, gammas, v, T, None, optimizing_functions, black_box2,
```

```
Lowest value of f2 obtained by Nelder-Mead method: 0.602081360038181 on point: [0.81
360036 0.20810135 0.60199933]

Difference between final function value and obtained nelder-mead minimum value:

Gradient descent for starting point (1.0, 1.0, 0.0): 13.752654786352918
Polyak GD for starting point (1.0, 1.0, 0.0): 0.24154756616684603
Nesterov GD for starting point (1.0, 1.0, 0.0): 1.075571534137009
```

```
AdaGrad for starting point (1.0, 1.0, 0.0): 0.38883934705451695
BFGS for starting point (1.0, 1.0, 0.0): 0.16122740690121395
```



Gradient descent with 20 steps.

Polyak GD with 20 steps.

Nesterov GD with 20 steps.

AdaGrad with 20 steps.

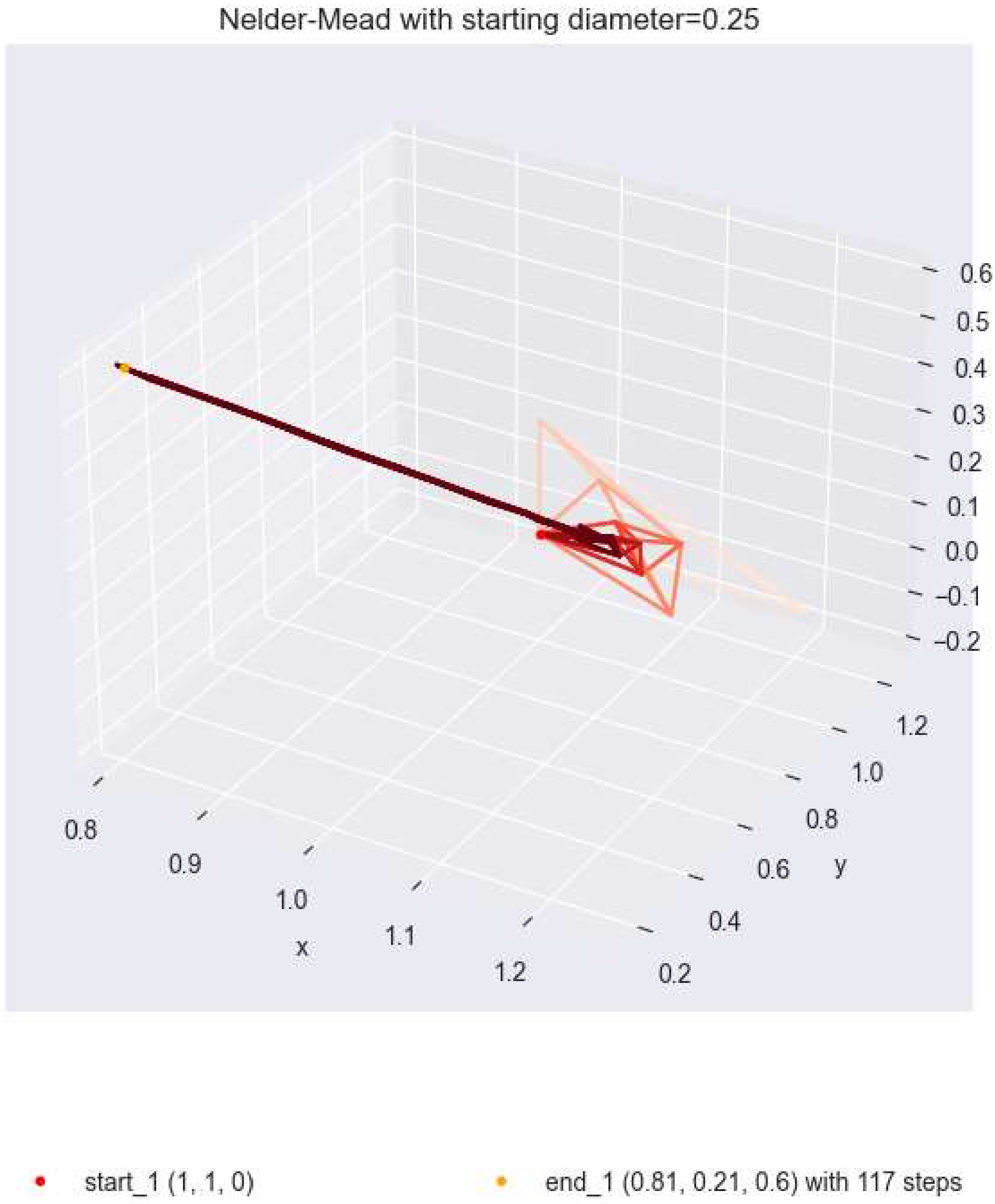
BFGS with 20 steps.

Nelder-Mead with starting diameter=0.25

For this function and starting point the descent methods struggled and didn't achieve a lot in 20 steps. The closest one was BFGS (if looking at the difference between function values at the end points) which was still far from it. \ A close up of Nelder-Mead method with 117 steps:

In [ ]:

```
plot_nelder_mead_3D(f2_x0, None, None, black_box2, diameters, T, None, steps=f2_step
```



In [ ]:

```
f3_min = f3_steps[-1][0]
gammas = np.array([[10e-1, 10e-1, 10e-1, 10e-2]])
```

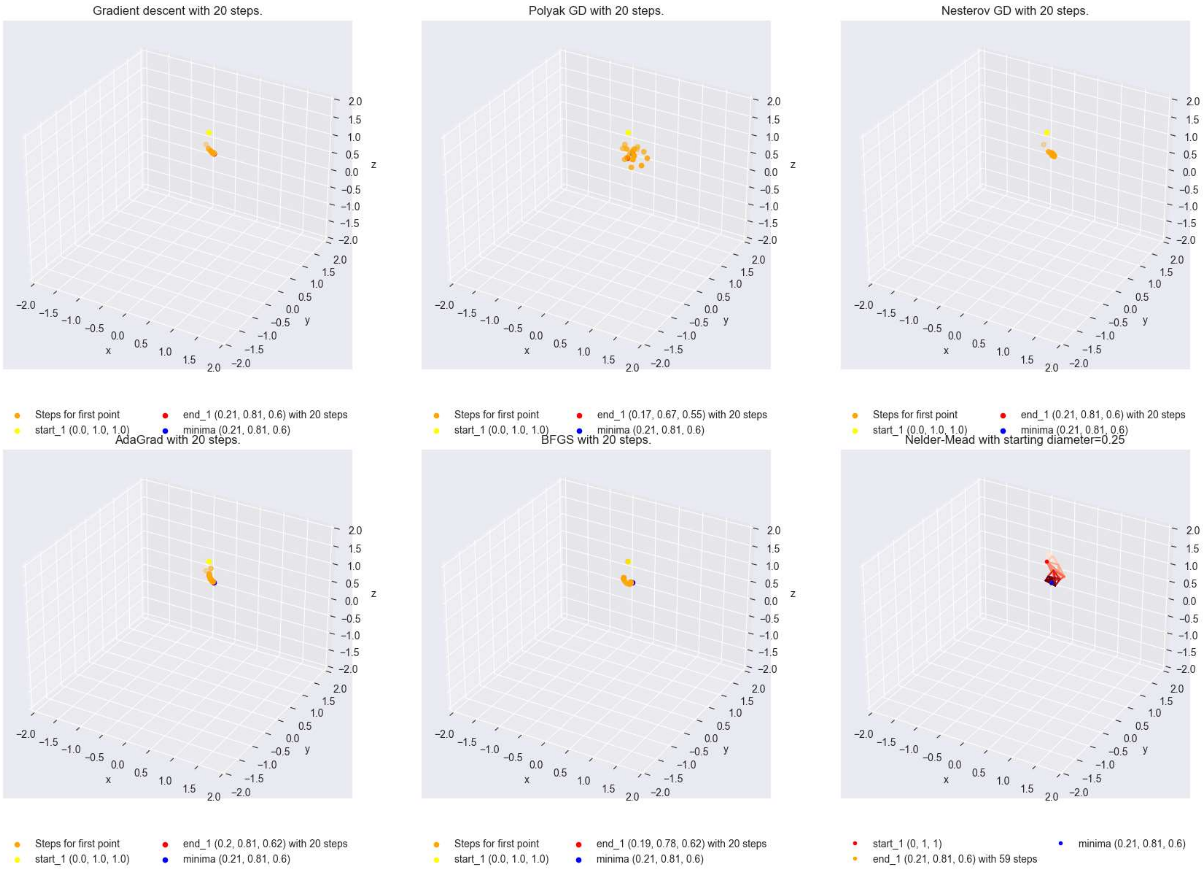
In [ ]:

```
print(f"Lowest value of f3 obtained by Nelder-Mead method: {black_box3(f3_min)} on p
plot_all(f3_x0, None, f3_min, gammas, v, T, None, optimizing_functions, black_box3,
```

```
Lowest value of f3 obtained by Nelder-Mead method: 0.602081360076319 on point: [0.20
81124  0.81359906 0.60198568]

Difference between final function value and obtained nelder-mead minimum value:

Gradient descent for starting point (0.0, 1.0, 1.0): 8.253210337372252e-10
Polyak GD for starting point (0.0, 1.0, 1.0): 0.0073296052608220474
Nesterov GD for starting point (0.0, 1.0, 1.0): 2.9741884299827603e-07
AdaGrad for starting point (0.0, 1.0, 1.0): 7.623175634607904e-05
BFGS for starting point (0.0, 1.0, 1.0): 0.00039013963085698933
```



Gradient descent with 20 steps.

Polyak GD with 20 steps.

Nesterov GD with 20 steps.

AdaGrad with 20 steps.

BFGS with 20 steps.

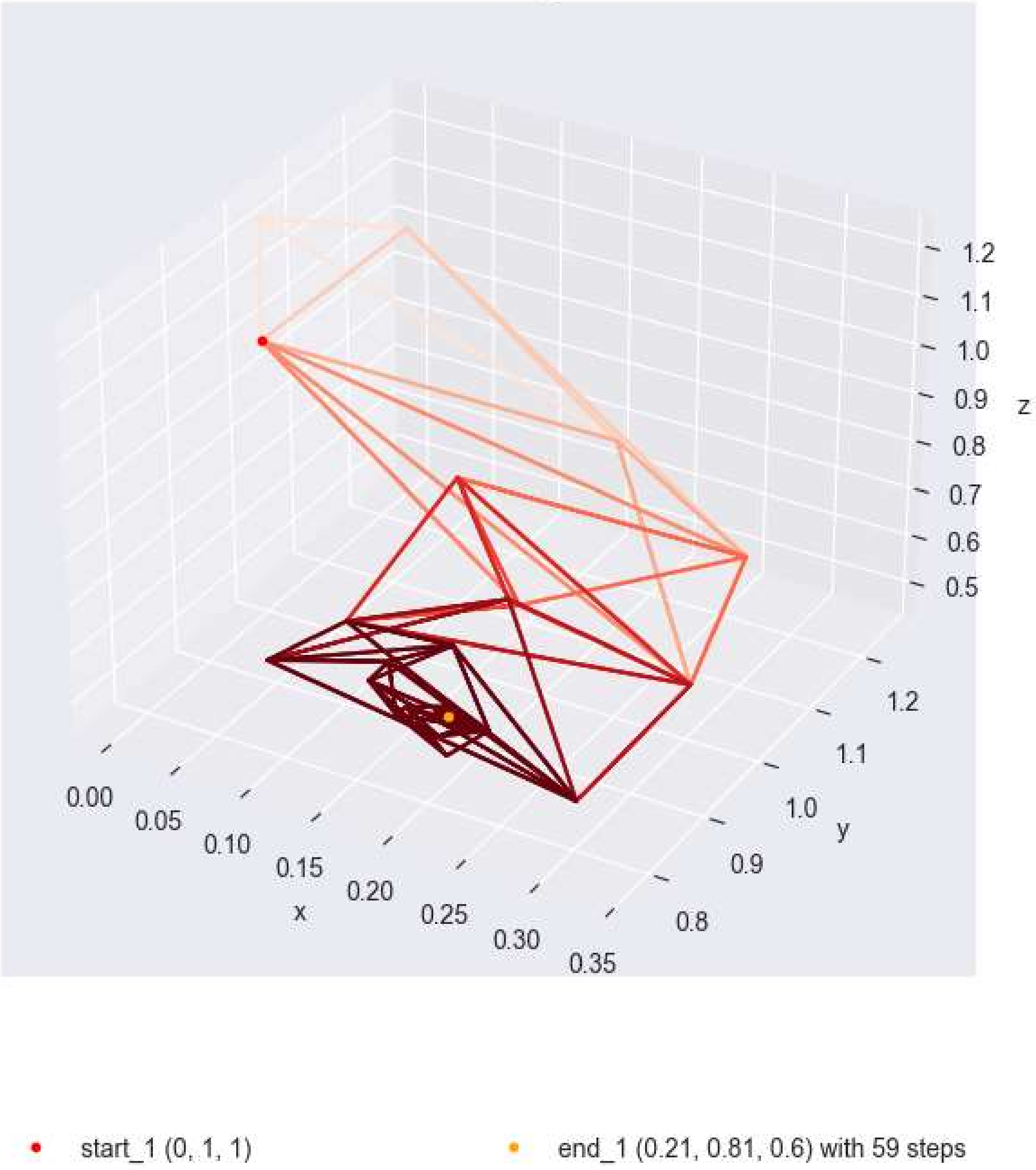
Nelder-Mead with starting diameter=0.25

Descent methods outperformed Nelder-Mead for this function and starting point. Normal gradient descent reached similar minima as Nelder-Mead in only 20 steps. \ A close up of Nelder-Mead method with 59 steps:

In [ ]:

```
plot_nelder_mead_3D(f3_x0, None, None, black_box3, diameters, T, None, steps=f3_step
```

Nelder-Mead with starting diameter=0.25


z
x
y
● start_1 (0, 1, 1)
● end_1 (0.21, 0.81, 0.6) with 59 steps