1. (30 puntos) Optimización de funciones

Para las siguientes funciones:

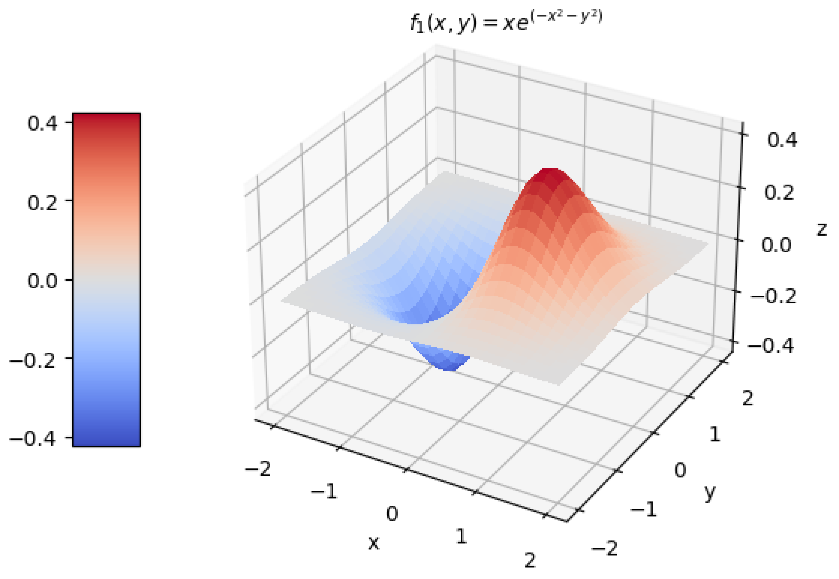
$$f_1(x, y) = xe^{(-x^2-y^2)} \tag{1}$$

1. Grafique tal función y distinga si las funciones son convexas o no, los puntos mínimos y regiones o puntos silla

In [26]:
```python
import math
import random
import torch
import sympy.core
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from sympy import diff, symbols, parse_expr, E, sympify, latex
from torch.autograd import grad
from IPython.display import display, Markdown
from enum import Enum
%matplotlib inline
```

In [27]:
```python
def f(x, y):
    return x * math.e ** (-x**2 - y**2)
```

In [15]:
```python
def plot_function():
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    plt.rcParams['legend.fontsize'] = 10
    linspace_x = torch.linspace(-2, 2, steps=30)
    linspace_y = torch.linspace(-2, 2, steps=30)
    X, Y = torch.meshgrid(linspace_x, linspace_y, indexing="xy")
    Z = f(X, Y)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ax.text2D(0.35, 0.95, r'$f_{1}(x,y)=xe^{(-x^{2}-y^{2})}$', transform=ax.transAxes)
    surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    fig.colorbar(surf, shrink=0.6, aspect=5, location='left')
    plt.show()

plot_function()
```



- La función $f_1(x, y) = xe^{(-x^2-y^2)}$ es no convexa.
- El punto mínimo se encuentra en la región azul, su valor en Z es aproximadamente -0.4.
- El punto máximo se encuentra en la región roja, su valor en Z es aproximadamente 0.4.
- Los puntos silla de la función se encuentran fuera del rango $x, y \in [-2, 2]$

2. (10 puntos) Implemente el algoritmo del descenso del gradiente con moméntum:

```
In [34]: #  Gradient descent with momentum implementation

         def gradient_descent_momentum(initial_position, epochs=5, momentum=0.1, alpha=0.05, epsilon=0.2, convergence=-0.42):
             agent = initial_position
             agent.requires_grad = True
             agents = [agent]
             inertia = 0
             message = ""
             for epoch in range(epochs):  #  range(epochs)
                 print("---------epoch-" + str(epoch) + "-------------\n")
                 print(f"Agent: {agent}\n")
                 function_eval = f(agent[:1], agent[1:])
                 gradient = grad(function_eval, agent, create_graph=True)[0]
                 agent = agent - ((momentum * inertia) + alpha * gradient)
                 theta = agent.detach()
                 agents.append(theta)
                 inertia = (momentum * inertia) + alpha * (1 - momentum) * gradient
                 print(f"Gradient: {gradient}")
                 print(f"New agent: {theta}")
                 print(f"Inertia: {inertia}\n")
                 if f(theta[:1], theta[1:]) <= convergence:
                     message = "Convergence reached !"
                     break
                 if torch.norm(gradient) < epsilon:
                     message = "Tolerance reached !"
                     break
             print(message)
             agents[0] = agents[0].detach()
             return agents, message
```

```
In [28]: # Code to generate the plot

         def plot(thetas, title):
             #  Contour plot
             linspace_x = torch.linspace(-2, 2, steps=30)
             linspace_y = torch.linspace(-2, 2, steps=30)
             X, Y = torch.meshgrid(linspace_x, linspace_y, indexing="xy")
             Z = f(X, Y)
             fig = plt.figure(figsize=plt.figaspect(0.4))
             ax = fig.add_subplot(1, 2, 1)
             cp = ax.contourf(X, Y, Z, cmap=cm.coolwarm)
             fig.colorbar(cp)  # Add a color bar to a plot
             ax.set_title(r'$f_{1}(x,y)=xe^{(-x^{2}-y^{2})}$')
             ax.set_xlabel('x')
             ax.set_ylabel('y')
             thetas = torch.stack(thetas)
             agents_x = thetas[:, 0]
             agents_y = thetas[:, 1]
             ax.scatter(agents_x, agents_y, s=40, lw=0, color='yellow', label=r'HELLO')
             plt.quiver(agents_x[:-1], agents_y[:-1], agents_x[1:]-agents_x[:-1], agents_y[1:]-agents_y[:-1], scale_units='xy',
                        angles='xy', scale=1)

             #  3D plot
             ax = fig.add_subplot(1, 2, 2, projection='3d')
             ax.set_xlabel('x')
             ax.set_ylabel('y')
             ax.set_zlabel('z')
             agents_z = f(agents_x, agents_y)
             ax.scatter(agents_x, agents_y, agents_z, s=80, lw=0, color='yellow', alpha=1)
             ax.plot_surface(X, Y, Z,  rstride=1, cstride=1, cmap=cm.coolwarm, edgecolor='none', alpha=0.4)
             ax.view_init(50, 100)
             ax.text2D(0.35, 0.95, r'$f_{1}(x,y)=xe^{(-x^{2}-y^{2})}$', transform=ax.transAxes)
             ax.quiver(agents_x[:-1], agents_y[:-1], agents_z[:-1], (agents_x[1:]-agents_x[:-1]), (agents_y[1:]-agents_y[:-1]),
                       (agents_z[1:]-agents_z[:-1]), length=1, alpha=1)
             plt.show()

             #  Results
             thetas_list = ""
             for theta in range(len(thetas)):
                 thetas_list += "epoch " + str(theta) + ": $\\theta_{" + str(theta) + "}=" + str(round(thetas[theta][0].item(), 3)) + "," + \
                                str(round(thetas[theta][1].item(), 3)) + "$ \n"
             minimums = ""
             for minimum in range(len(agents_z)):
                 minimums += "$f(\\theta_{" + str(minimum) + "})=" + str(round(agents_z[minimum].item(), 3)) + "$ \n"

             plt.figure(figsize=(1, 0.1))
             plt.text(0.2, 1, title, ha='center', va='baseline', size=12)
```

```python
    plt.axis('off')
    plt.show()
    plt.figure(figsize=(6, 0.1))
    plt.text(0, 1, thetas_list, ha='left', va='baseline', size=12)
    plt.text(1, 1, minimums, ha='left', va='baseline', size=12)
    plt.axis('off')
    plt.show()
```

a) Escoja un coeficiente de aprendizaje α que permita la convergencia y reporte los resultados para 10 corridas:

1. La tolerancia fijada para la convergencia en términos de la magnitud del gradiente.

2. La cantidad de iteraciones necesarias para converger.

3. El punto de convergencia.

4. Escoga una de las corridas y en una gráfica muestre los puntos probados (visitados) por el algoritmo.

In [50]:
```python
#  %%capture
#  Uncomment previous line to disable prints

#  GDM Execution
alpha = 0.25 # Learning rate alpha
gamma = 0.5 # Momentum coefficient
epsilon = 0.11  # Tolerance epsilon
convergence = -0.42 # Convergence point
epochs = 20 # Iteration epochs
runs = 10 # Runs
results = [] # Gradient descent results

for run in range(runs):
    print("\n----------------------")
    print(f"Run #{run + 1}\n")
    point_x = random.uniform(-1, 1)
    point_y = random.uniform(-1, 1)
    init_position = torch.Tensor([point_x, point_y])
    thetas, message = gradient_descent_momentum(init_position, epochs=epochs, alpha=alpha, momentum=gamma,
                                                epsilon=epsilon, convergence=convergence)
    results.append((thetas, message))

#  Randomly choose a run and plot
run = random.randint(0, runs - 1)

#  Example execution:
#  init_position = torch.Tensor([0.6, -0.1])
#  thetas, message = gradient_descent_momentum(init_position, epochs=epochs, alpha=alpha, momentum=0, epsilon=epsilon, convergence=co
#  title = "GDM with: $\\alpha=" + str(alpha) + ", \\gamma=" + str(gamma) + ", \\epsilon=" + str(epsilon) + "$" + ", convergence=$" +
#        ", Stopped due to: " + message
```

```
----------------------
Run #1

---------epoch-0-------------

Agent: tensor([ 0.5197, -0.3662], requires_grad=True)

Gradient: tensor([0.3069, 0.2541], grad_fn=<AddBackward0>)
New agent: tensor([ 0.4430, -0.4297])
Inertia: tensor([0.0384, 0.0318], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([ 0.4430, -0.4297], grad_fn=<SubBackward0>)

Gradient: tensor([0.4150, 0.2601], grad_fn=<AddBackward0>)
New agent: tensor([ 0.3201, -0.5106])
Inertia: tensor([0.0711, 0.0484], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([ 0.3201, -0.5106], grad_fn=<SubBackward0>)

Gradient: tensor([0.5530, 0.2273], grad_fn=<AddBackward0>)
New agent: tensor([ 0.1463, -0.5916])
Inertia: tensor([0.1047, 0.0526], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([ 0.1463, -0.5916], grad_fn=<SubBackward0>)

Gradient: tensor([0.6602, 0.1194], grad_fn=<AddBackward0>)
New agent: tensor([-0.0711, -0.6478])
Inertia: tensor([0.1349, 0.0412], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.0711, -0.6478], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.6474, -0.0602], grad_fn=<AddBackward0>)
New agent: tensor([-0.3003, -0.6533])
Inertia: tensor([0.1484, 0.0131], grad_fn=<AddBackward0>)

---------epoch-5-------------

Agent: tensor([-0.3003, -0.6533], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.4887, -0.2340], grad_fn=<AddBackward0>)
New agent: tensor([-0.4967, -0.6014])
Inertia: tensor([ 0.1353, -0.0227], grad_fn=<AddBackward0>)

---------epoch-6-------------

Agent: tensor([-0.4967, -0.6014], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.2757, -0.3251], grad_fn=<AddBackward0>)
New agent: tensor([-0.6333, -0.5087])
Inertia: tensor([ 0.1021, -0.0520], grad_fn=<AddBackward0>)

---------epoch-7-------------

Agent: tensor([-0.6333, -0.5087], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.1023, -0.3331], grad_fn=<AddBackward0>)
New agent: tensor([-0.7099, -0.3995])
Inertia: tensor([ 0.0638, -0.0676], grad_fn=<AddBackward0>)

---------epoch-8-------------

Agent: tensor([-0.7099, -0.3995], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0041, -0.2921], grad_fn=<AddBackward0>)
New agent: tensor([-0.7408, -0.2926])
Inertia: tensor([ 0.0314, -0.0703], grad_fn=<AddBackward0>)

---------epoch-9-------------

Agent: tensor([-0.7408, -0.2926], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0517, -0.2299], grad_fn=<AddBackward0>)
New agent: tensor([-0.7436, -0.2000])
Inertia: tensor([ 0.0092, -0.0639], grad_fn=<AddBackward0>)
```

```
---------epoch-10-------------

Agent: tensor([-0.7436, -0.2000], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0585, -0.1644], grad_fn=<AddBackward0>)
New agent: tensor([-0.7336, -0.1269])
Inertia: tensor([-0.0027, -0.0525], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #2

---------epoch-0-------------

Agent: tensor([0.4274, 0.6008], requires_grad=True)

Gradient: tensor([ 0.3686, -0.2982], grad_fn=<AddBackward0>)
New agent: tensor([0.3352, 0.6753])
Inertia: tensor([ 0.0461, -0.0373], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([0.3352, 0.6753], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.4391, -0.2564], grad_fn=<AddBackward0>)
New agent: tensor([0.2024, 0.7581])
Inertia: tensor([ 0.0779, -0.0507], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([0.2024, 0.7581], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.4960, -0.1658], grad_fn=<AddBackward0>)
New agent: tensor([0.0394, 0.8249])
Inertia: tensor([ 0.1010, -0.0461], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([0.0394, 0.8249], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.5041, -0.0329], grad_fn=<AddBackward0>)
New agent: tensor([-0.1371,  0.8561])
Inertia: tensor([ 0.1135, -0.0271], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.1371,  0.8561], grad_fn=<SubBackward0>)

Gradient: tensor([0.4538, 0.1107], grad_fn=<AddBackward0>)
New agent: tensor([-0.3073,  0.8420])
Inertia: tensor([0.1135, 0.0003], grad_fn=<AddBackward0>)

---------epoch-5-------------

Agent: tensor([-0.3073,  0.8420], grad_fn=<SubBackward0>)

Gradient: tensor([0.3632, 0.2317], grad_fn=<AddBackward0>)
New agent: tensor([-0.4548,  0.7840])
Inertia: tensor([0.1021, 0.0291], grad_fn=<AddBackward0>)

---------epoch-6-------------

Agent: tensor([-0.4548,  0.7840], grad_fn=<SubBackward0>)

Gradient: tensor([0.2578, 0.3136], grad_fn=<AddBackward0>)
New agent: tensor([-0.5703,  0.6910])
Inertia: tensor([0.0833, 0.0537], grad_fn=<AddBackward0>)

---------epoch-7-------------

Agent: tensor([-0.5703,  0.6910], grad_fn=<SubBackward0>)

Gradient: tensor([0.1566, 0.3532], grad_fn=<AddBackward0>)
New agent: tensor([-0.6511,  0.5759])
Inertia: tensor([0.0612, 0.0710], grad_fn=<AddBackward0>)

---------epoch-8-------------

Agent: tensor([-0.6511,  0.5759], grad_fn=<SubBackward0>)
```

```
Gradient: tensor([0.0714, 0.3523], grad_fn=<AddBackward0>)
New agent: tensor([-0.6996,  0.4523])
Inertia: tensor([0.0395, 0.0795], grad_fn=<AddBackward0>)

---------epoch-9-------------

Agent: tensor([-0.6996,  0.4523], grad_fn=<SubBackward0>)

Gradient: tensor([0.0105, 0.3161], grad_fn=<AddBackward0>)
New agent: tensor([-0.7220,  0.3335])
Inertia: tensor([0.0211, 0.0793], grad_fn=<AddBackward0>)

---------epoch-10-------------

Agent: tensor([-0.7220,  0.3335], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0226,  0.2558], grad_fn=<AddBackward0>)
New agent: tensor([-0.7269,  0.2299])
Inertia: tensor([0.0077, 0.0716], grad_fn=<AddBackward0>)

---------epoch-11-------------

Agent: tensor([-0.7269,  0.2299], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0317,  0.1869], grad_fn=<AddBackward0>)
New agent: tensor([-0.7228,  0.1473])
Inertia: tensor([-0.0001,  0.0592], grad_fn=<AddBackward0>)

---------epoch-12-------------

Agent: tensor([-0.7228,  0.1473], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0261,  0.1236], grad_fn=<AddBackward0>)
New agent: tensor([-0.7162,  0.0868])
Inertia: tensor([-0.0033,  0.0450], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #3

---------epoch-0-------------

Agent: tensor([0.4148, 0.0323], requires_grad=True)

Gradient: tensor([ 0.5516, -0.0225], grad_fn=<AddBackward0>)
New agent: tensor([0.2769, 0.0379])
Inertia: tensor([ 0.0689, -0.0028], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([0.2769, 0.0379], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.7830, -0.0194], grad_fn=<AddBackward0>)
New agent: tensor([0.0467, 0.0442])
Inertia: tensor([ 0.1323, -0.0038], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([0.0467, 0.0442], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.9915, -0.0041], grad_fn=<AddBackward0>)
New agent: tensor([-0.2673,  0.0471])
Inertia: tensor([ 0.1901, -0.0024], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([-0.2673,  0.0471], grad_fn=<SubBackward0>)

Gradient: tensor([0.7962, 0.0234], grad_fn=<AddBackward0>)
New agent: tensor([-0.5614,  0.0425])
Inertia: tensor([0.1946, 0.0017], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.5614,  0.0425], grad_fn=<SubBackward0>)

Gradient: tensor([0.2692, 0.0347], grad_fn=<AddBackward0>)
New agent: tensor([-0.7260,  0.0329])
Inertia: tensor([0.1309, 0.0052], grad_fn=<AddBackward0>)

Convergence reached !
```

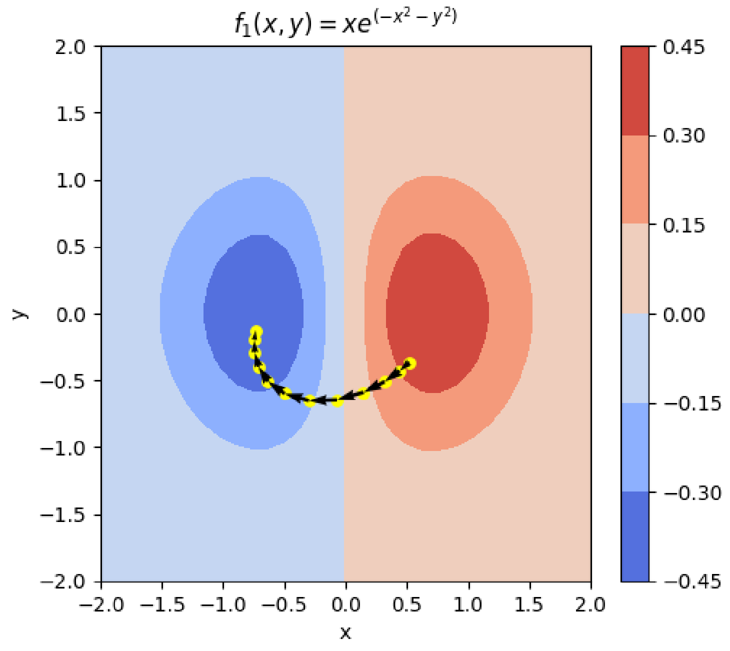
```
----------------------
Run #4

---------epoch-0-------------

Agent: tensor([-0.3961,  0.0969], requires_grad=True)

Gradient: tensor([0.5811, 0.0650], grad_fn=<AddBackward0>)
New agent: tensor([-0.5414,  0.0807])
Inertia: tensor([0.0726, 0.0081], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.5414,  0.0807], grad_fn=<SubBackward0>)

Gradient: tensor([0.3067, 0.0647], grad_fn=<AddBackward0>)
New agent: tensor([-0.6544,  0.0604])
Inertia: tensor([0.0747, 0.0122], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #5

---------epoch-0-------------

Agent: tensor([0.1768, 0.2303], requires_grad=True)

Gradient: tensor([ 0.8617, -0.0749], grad_fn=<AddBackward0>)
New agent: tensor([-0.0386,  0.2490])
Inertia: tensor([ 0.1077, -0.0094], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.0386,  0.2490], grad_fn=<SubBackward0>)

Gradient: tensor([0.9357, 0.0180], grad_fn=<AddBackward0>)
New agent: tensor([-0.3264,  0.2491])
Inertia: tensor([ 0.1708, -0.0024], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([-0.3264,  0.2491], grad_fn=<SubBackward0>)

Gradient: tensor([0.6649, 0.1374], grad_fn=<AddBackward0>)
New agent: tensor([-0.5780,  0.2160])
Inertia: tensor([0.1685, 0.0160], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([-0.5780,  0.2160], grad_fn=<SubBackward0>)

Gradient: tensor([0.2268, 0.1706], grad_fn=<AddBackward0>)
New agent: tensor([-0.7189,  0.1654])
Inertia: tensor([0.1126, 0.0293], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.7189,  0.1654], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0196,  0.1380], grad_fn=<AddBackward0>)
New agent: tensor([-0.7703,  0.1162])
Inertia: tensor([0.0539, 0.0319], grad_fn=<AddBackward0>)

---------epoch-5-------------

Agent: tensor([-0.7703,  0.1162], grad_fn=<SubBackward0>)

Gradient: tensor([-0.1019,  0.0976], grad_fn=<AddBackward0>)
New agent: tensor([-0.7718,  0.0759])
Inertia: tensor([0.0142, 0.0281], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #6

---------epoch-0-------------

Agent: tensor([-0.8058,  0.4263], requires_grad=True)
```

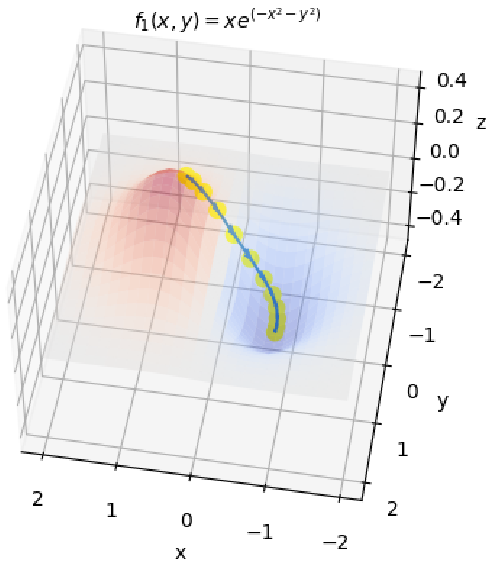
```
Gradient: tensor([-0.1300,  0.2993], grad_fn=<AddBackward0>)
New agent: tensor([-0.7732,  0.3515])
Inertia: tensor([-0.0163,  0.0374], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.7732,  0.3515], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0952,  0.2642], grad_fn=<AddBackward0>)
New agent: tensor([-0.7413,  0.2667])
Inertia: tensor([-0.0200,  0.0517], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([-0.7413,  0.2667], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0533,  0.2126], grad_fn=<AddBackward0>)
New agent: tensor([-0.7180,  0.1877])
Inertia: tensor([-0.0167,  0.0524], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([-0.7180,  0.1877], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0179,  0.1554], grad_fn=<AddBackward0>)
New agent: tensor([-0.7052,  0.1227])
Inertia: tensor([-0.0106,  0.0456], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #7

---------epoch-0-------------

Agent: tensor([-0.5923, -0.9204], requires_grad=True)

Gradient: tensor([ 0.0901, -0.3291], grad_fn=<AddBackward0>)
New agent: tensor([-0.6148, -0.8381])
Inertia: tensor([ 0.0113, -0.0411], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.6148, -0.8381], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0828, -0.3498], grad_fn=<AddBackward0>)
New agent: tensor([-0.6411, -0.7301])
Inertia: tensor([ 0.0160, -0.0643], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([-0.6411, -0.7301], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0692, -0.3642], grad_fn=<AddBackward0>)
New agent: tensor([-0.6664, -0.6069])
Inertia: tensor([ 0.0166, -0.0777], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([-0.6664, -0.6069], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0496, -0.3590], grad_fn=<AddBackward0>)
New agent: tensor([-0.6871, -0.4783])
Inertia: tensor([ 0.0145, -0.0837], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.6871, -0.4783], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0276, -0.3261], grad_fn=<AddBackward0>)
New agent: tensor([-0.7013, -0.3549])
Inertia: tensor([ 0.0107, -0.0826], grad_fn=<AddBackward0>)

---------epoch-5-------------

Agent: tensor([-0.7013, -0.3549], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0088, -0.2684], grad_fn=<AddBackward0>)
New agent: tensor([-0.7089, -0.2465])
Inertia: tensor([ 0.0065, -0.0749], grad_fn=<AddBackward0>)

---------epoch-6-------------
```

```
Agent: tensor([-0.7089, -0.2465], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0028, -0.1990], grad_fn=<AddBackward0>)
New agent: tensor([-0.7114, -0.1593])
Inertia: tensor([ 0.0029, -0.0623], grad_fn=<AddBackward0>)

---------epoch-7-------------

Agent: tensor([-0.7114, -0.1593], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0071, -0.1333], grad_fn=<AddBackward0>)
New agent: tensor([-0.7110, -0.0949])
Inertia: tensor([ 0.0005, -0.0478], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #8

---------epoch-0-------------

Agent: tensor([-0.3779, -0.3535], requires_grad=True)

Gradient: tensor([ 0.5466, -0.2044], grad_fn=<AddBackward0>)
New agent: tensor([-0.5145, -0.3024])
Inertia: tensor([ 0.0683, -0.0256], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.5145, -0.3024], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.3295, -0.2180], grad_fn=<AddBackward0>)
New agent: tensor([-0.6311, -0.2352])
Inertia: tensor([ 0.0753, -0.0400], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([-0.6311, -0.2352], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.1293, -0.1886], grad_fn=<AddBackward0>)
New agent: tensor([-0.7011, -0.1680])
Inertia: tensor([ 0.0538, -0.0436], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([-0.7011, -0.1680], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0101, -0.1401], grad_fn=<AddBackward0>)
New agent: tensor([-0.7305, -0.1112])
Inertia: tensor([ 0.0282, -0.0393], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #9

---------epoch-0-------------

Agent: tensor([-0.5143, -0.4933], requires_grad=True)

Gradient: tensor([ 0.2834, -0.3053], grad_fn=<AddBackward0>)
New agent: tensor([-0.5852, -0.4169])
Inertia: tensor([ 0.0354, -0.0382], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.5852, -0.4169], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.1881, -0.2912], grad_fn=<AddBackward0>)
New agent: tensor([-0.6499, -0.3250])
Inertia: tensor([ 0.0412, -0.0555], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([-0.6499, -0.3250], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0916, -0.2492], grad_fn=<AddBackward0>)
New agent: tensor([-0.6934, -0.2350])
Inertia: tensor([ 0.0321, -0.0589], grad_fn=<AddBackward0>)

---------epoch-3-------------
```

```
Agent: tensor([-0.6934, -0.2350], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.0225, -0.1907], grad_fn=<AddBackward0>)
New agent: tensor([-0.7150, -0.1579])
Inertia: tensor([ 0.0188, -0.0533], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.7150, -0.1579], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0132, -0.1321], grad_fn=<AddBackward0>)
New agent: tensor([-0.7212, -0.0982])
Inertia: tensor([ 0.0078, -0.0431], grad_fn=<AddBackward0>)

Convergence reached !

----------------------
Run #10

---------epoch-0-------------

Agent: tensor([-0.9125,  0.1362], requires_grad=True)

Gradient: tensor([-0.2840,  0.1061], grad_fn=<AddBackward0>)
New agent: tensor([-0.8415,  0.1097])
Inertia: tensor([-0.0355,  0.0133], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([-0.8415,  0.1097], grad_fn=<SubBackward0>)

Gradient: tensor([-0.2026,  0.0899], grad_fn=<AddBackward0>)
New agent: tensor([-0.7731,  0.0806])
Inertia: tensor([-0.0431,  0.0179], grad_fn=<AddBackward0>)

Convergence reached !
```

In [52]:
```
#  Plot separated so that console prints are not shown alongside !!
thetas, message = results[run]
title = "GDM with: $\\alpha=" + str(alpha) + ", \\gamma=" + str(gamma) + ", \\epsilon=" + str(epsilon) + "$" +\
        ", convergence=$" + str(convergence) + "$" + ", Stopped due to: " + message
plot(thetas, title)
```





GDM with: $\alpha = 0.25, \gamma = 0.5, \epsilon = 0.11$, convergence$=-0.42$, Stopped due to: Convergence reached !

epoch 0: $\theta_0 = 0.52, -0.366$
epoch 1: $\theta_1 = 0.443, -0.43$
epoch 2: $\theta_2 = 0.32, -0.511$
epoch 3: $\theta_3 = 0.146, -0.592$
epoch 4: $\theta_4 = -0.071, -0.648$
epoch 5: $\theta_5 = -0.3, -0.653$
epoch 6: $\theta_6 = -0.497, -0.601$
epoch 7: $\theta_7 = -0.633, -0.509$
epoch 8: $\theta_8 = -0.71, -0.399$
epoch 9: $\theta_9 = -0.741, -0.293$
epoch 10: $\theta_{10} = -0.744, -0.2$
epoch 11: $\theta_{11} = -0.734, -0.127$

$f(\theta_0) = 0.347$
$f(\theta_1) = 0.303$
$f(\theta_2) = 0.223$
$f(\theta_3) = 0.101$
$f(\theta_4) = -0.046$
$f(\theta_5) = -0.179$
$f(\theta_6) = -0.27$
$f(\theta_7) = -0.327$
$f(\theta_8) = -0.366$
$f(\theta_9) = -0.393$
$f(\theta_{10}) = -0.411$
$f(\theta_{11}) = -0.421$

b) Escoja un α relativamente grande respecto al valor seleccionado. ¿Qué sucede? ¿Permite un α muy grande la convergencia?
Para esta superficie, con los parámetros default programados y un punto inicial cercano a los rangos en $Y = [0.5, -0.5]$ y en $X = [-1.0, 0.5]$, el algoritmo permite un α muy grande. Al probar con varios valores de α, se llegó a que un α de 1.2 es lo suficientemente grande para una ejecución estable respetando los rangos y los otros parámetros anteriormente mencionados. En el ejemplo abajo se puede observar una ejecución con un valor de $\alpha = 1.2$. Con un α elevado, el algoritmo llega más rápido al mínimo, sin embargo, corre el riesgo de avanzar de más y omitir el mínimo dependiendo de los otros parámetros del algoritmo y el punto inicial. En este caso, se permite un α grande para la convergencia, se llega mucho más rápido al punto de convergencia default, sin importar donde se ubique el punto inicial según los parámetros y los rangos indicados anteriormente.

c) ¿Qué sucede si escoge un α muy pequeño?

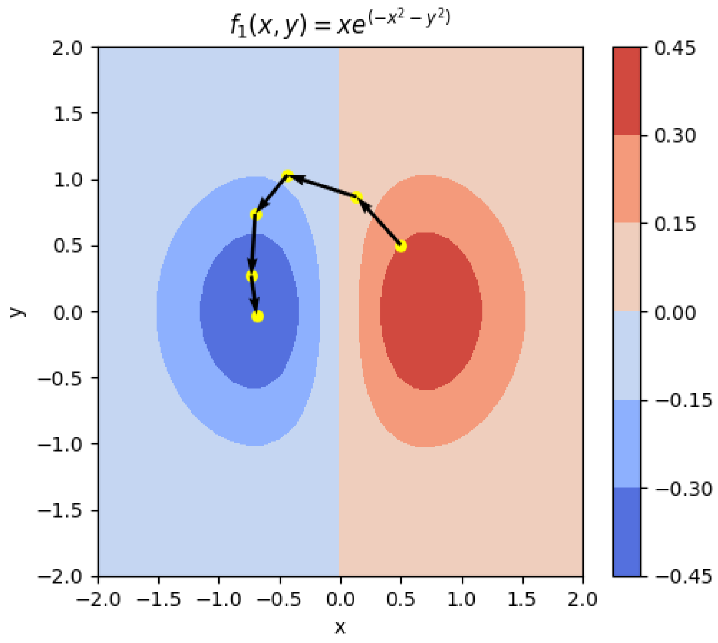
Para esta superficie con un α pequeño, al compararlo con un alpha grande, se llega al mínimo en una cantidad mayor de epochs. Véase por ejemplo una ejecución con un α de 0.05, punto inicial $x = 0.5, y = -0.23$ y otros parámetros default, fue necesario 32 epochs para llegar a un mínimo de -0.42, en cambio con un α de 0.8 se llegó a un mínimo de -0.428 en 5 epochs. A pesar de esto, al usar un α pequeño, se sigue una ruta más segura hacia el mínimo en lugar de un α grande que lo puede llegar a omitir.

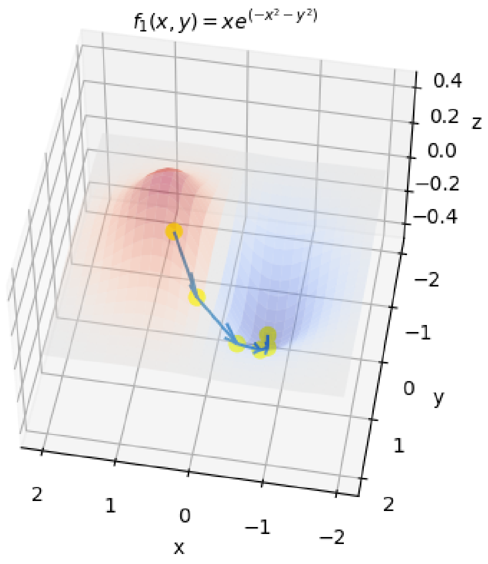
d) ¿Cómo puede el algoritmo de descenso de gradiente evitar quedar atrapado en mínimos locales o puntos silla?

Se puede utilizar un coeficiente de momentum grande, puede ser que llegue cerca de un mínimo local o punto silla pero el momentum lo saca de ahí y lo hace buscar otros mínimos menores en donde no se quede atascado. Del mismo modo, también depende en donde se encuentre el punto inicial, por ejemplo, con esta superficie, si el mismo se encuentra en una posición $x > 1.2$ nunca llegará al mínimo sin importar los valores de los otros parámetros.

In [39]:
```
#  Point B
alpha = 1.2
init_position = torch.Tensor([0.5, 0.5])
thetas, message = gradient_descent_momentum(init_position, alpha=alpha)
title = "GDM with: $\\alpha=" + str(alpha) + "$ and default programmed parameters"
plot(thetas, title)
```

```
---------epoch-0-------------

Agent: tensor([0.5000, 0.5000], requires_grad=True)

Gradient: tensor([ 0.3033, -0.3033], grad_fn=<AddBackward0>)
New agent: tensor([0.1361, 0.8639])
Inertia: tensor([ 0.3275, -0.3275], grad_fn=<AddBackward0>)

---------epoch-1-------------

Agent: tensor([0.1361, 0.8639], grad_fn=<SubBackward0>)

Gradient: tensor([ 0.4482, -0.1094], grad_fn=<AddBackward0>)
New agent: tensor([-0.4345,  1.0280])
Inertia: tensor([ 0.5168, -0.1509], grad_fn=<AddBackward0>)

---------epoch-2-------------

Agent: tensor([-0.4345,  1.0280], grad_fn=<SubBackward0>)

Gradient: tensor([0.1792, 0.2571], grad_fn=<AddBackward0>)
New agent: tensor([-0.7011,  0.7346])
Inertia: tensor([0.2452, 0.2625], grad_fn=<AddBackward0>)

---------epoch-3-------------

Agent: tensor([-0.7011,  0.7346], grad_fn=<SubBackward0>)

Gradient: tensor([0.0060, 0.3673], grad_fn=<AddBackward0>)
New agent: tensor([-0.7329,  0.2676])
Inertia: tensor([0.0310, 0.4229], grad_fn=<AddBackward0>)

---------epoch-4-------------

Agent: tensor([-0.7329,  0.2676], grad_fn=<SubBackward0>)

Gradient: tensor([-0.0403,  0.2134], grad_fn=<AddBackward0>)
New agent: tensor([-0.6875, -0.0308])
Inertia: tensor([-0.0405,  0.2727], grad_fn=<AddBackward0>)

Convergence reached !
```





GDM with: $\alpha = 1.2$ and default programmed parameters

epoch 0: $\theta_0 = 0.5, 0.5$	$f(\theta_0) = 0.303$
epoch 1: $\theta_1 = 0.136, 0.864$	$f(\theta_1) = 0.063$
epoch 2: $\theta_2 = -0.434, 1.028$	$f(\theta_2) = -0.125$
epoch 3: $\theta_3 = -0.701, 0.735$	$f(\theta_3) = -0.25$
epoch 4: $\theta_4 = -0.733, 0.268$	$f(\theta_4) = -0.399$
epoch 5: $\theta_5 = -0.688, -0.031$	$f(\theta_5) = -0.428$

3. (20 puntos) Implemente el algoritmo de Newton-Raphson:

In [29]:
```python
# Code to generate the visual construction of the hessian matrix of any function.
# This is used for visual purposes only !!

def visual_function(expression: str = ""):
    if not expression:
        x = symbols('x')
        y = symbols('y')
        return x * E ** (-x**2 - y**2)
    return parse_expr(expression)


def visual_hessian_matrix(expression: sympy.core.Expr, variables: list = None):
    if variables is None:
        variables = ["x", "y"]
    hess_matrix = np.empty((len(variables), len(variables)), dtype=sympy.core.Expr)
    matrix_col_row = 0
    derivatives = []
    for variable in variables:
        print("Building column #" + str(matrix_col_row + 1) + " and row #" + str(matrix_col_row + 1))
        first_derivative = sympify(diff(expression, variable))
        der = "df/d" + variable + "=" + str(first_derivative)
        derivatives.append(der)
        print(der)
        variable_index = variables.index(variable)
        column = []
        for second_variable in variables[variable_index:]:
            second_derivative = sympify(diff(first_derivative, second_variable))
            der = "df/d" + second_variable + "d" + variable + "=" + str(second_derivative)
            derivatives.append(der)
            print(der)
            column.append(second_derivative)
        hess_matrix[matrix_col_row:, matrix_col_row] = column
        row = []
        for second_variable in variables[variable_index+1:]:
            derivative_second_var = sympify(diff(expression, second_variable))
            der = "df/d" + second_variable + "=" + str(derivative_second_var)
            derivatives.append(der)
            print(der)
            second_derivative = sympify(diff(derivative_second_var, variable))
            der = "df/d" + variable + "d" + second_variable + "=" + str(second_derivative)
            derivatives.append(der)
            print(der)
            row.append(second_derivative)
        hess_matrix[matrix_col_row, matrix_col_row + 1:] = row
        matrix_col_row += 1
    return hess_matrix, derivatives


def derivatives_to_latex(derivatives):
    latex_derivatives = []
    for der in derivatives:
        start = "$" + der[0: der.find("=")]
        func = sympify(der[der.find("=") + 1:])
        latex_derivative = start + "=" + latex(func) + "$\n"
        latex_derivatives.append(latex_derivative)
    return latex_derivatives


def matrix_to_latex(matrix):
    latex_matrix = r'$H=\begin{pmatrix}'
    for row in matrix:
        element_latex = ""
        for element in row:
            element_latex += latex(element) + " & "
        element_latex = element_latex[:len(element_latex) - 3] + r'\\'
        latex_matrix += element_latex
    latex_matrix += r'\end{pmatrix}$'
    return latex_matrix


def step_by_step_hessian():
    h_matrix, all_derivatives = visual_hessian_matrix(visual_function())
    all_derivatives = derivatives_to_latex(all_derivatives)
    latex_h_matrix = matrix_to_latex(h_matrix)
    print(f"Hessian Matrix: {h_matrix}" + "\n")
    print(f"Hessian Matrix in Latex: {latex_h_matrix}\n")
    return latex_h_matrix, all_derivatives
```

In [30]:
```python
# Newton-Raphson implementation
```

```python
def hessian_matrix(gradient, agent, visualize=True):
    if visualize:
        print(f"First Derivative: {gradient}")
    dimensions = agent.shape[0]
    hess_matrix = torch.zeros(dimensions, dimensions)
    for dimension in range(dimensions):
        second_derivative = grad(gradient[dimension], agent, create_graph=True)[0]
        if visualize:
            print(f"Second derivative on dimension {dimension}: {second_derivative}")
        hess_matrix[dimension:] = second_derivative
    if visualize:
        print(f"Hessian Matrix: {hess_matrix}\n")
    return hess_matrix


class Point(Enum):
    LOCAL_MIN = "local minimum"
    LOCAL_MAX = "local maximum"
    SADDLE_POINT = "saddle point"
    NO_CONCLUSION = "no conclusion"


def point_status_by_determinant(hess_matrix):
    determinant = hess_matrix[0][0] * hess_matrix[1][1] - hess_matrix[0][1]**2
    if determinant > 0 and hess_matrix[0][0] > 0:
        return Point.LOCAL_MIN
    if determinant > 0 and hess_matrix[0][0] < 0:
        return Point.LOCAL_MAX
    if determinant < 0:
        return Point.SADDLE_POINT
    return Point.NO_CONCLUSION


def newton_raphson(initial_position, function, epochs=5, damping_factor=0.4, convergence=-0.42, epsilon=0.2,
                   run_with_fix=True, stop_at_saddle=False):
    agent = initial_position
    agent.requires_grad = True
    agents = [agent]
    latex_h_matrix, all_derivatives = step_by_step_hessian()
    exit_message = ""
    for epoch in range(epochs):
        print("---------epoch-" + str(epoch) + "-------------\n")
        print(f"Agent: {agent}\n")
        function_eval = function(agent[:1], agent[1:])
        gradient = grad(function_eval, agent, create_graph=True)[0]
        hess_matrix = hessian_matrix(gradient, agent)
        point_by_determinant = point_status_by_determinant(hess_matrix)
        print(f"The point ({round(agent[:1].item(), 3)}, {round(agent[1:].item(), 3)}) is: {point_by_determinant.value}\n")
        if (stop_at_saddle and point_by_determinant == Point.SADDLE_POINT) or point_by_determinant == Point.NO_CONCLUSION:
            exit_message = "Saddle point found ! Stopping..." if point_by_determinant == Point.SADDLE_POINT else \
                "Stopped due to non conclusive point !"
            break
        inverse_hess_matrix = torch.nan_to_num(torch.inverse(hess_matrix))
        hess_gradient = torch.mm(inverse_hess_matrix, gradient.view(gradient.shape[0], 1))
        new_agent = agent.view(agent.shape[0], 1) - damping_factor * hess_gradient
        new_agent = new_agent.view(agent.shape[0])
        if function(new_agent[:1], new_agent[1:]) > function_eval and run_with_fix:  # This is the fix !!
            # If it goes up then force it to go down
            agent = agent.view(agent.shape[0], 1) - damping_factor * torch.abs(hess_gradient)
            agent = agent.view(agent.shape[0])
        else:
            agent = new_agent
        new_agent_status = point_status_by_determinant(hessian_matrix(grad(function(agent[:1], agent[1:]), agent,
                                                                            create_graph=True)[0], agent, visualize=False))
        theta = agent.detach()
        agents.append(theta)
        print(f"Inverse Hessian Matrix: {inverse_hess_matrix}\n")
        print(f"Gradient: {hess_gradient}")
        print(f"New agent: {theta}")
        print(f"The new agent is a: {new_agent_status.value}\n")
        if new_agent_status.value == Point.NO_CONCLUSION:
            exit_message = "Stopped due to non conclusive point !"
            break
        if f(theta[:1], theta[1:]) <= convergence:
            exit_message = "Convergence reached ! "
            break
        if torch.norm(gradient) < epsilon:
            exit_message = "Tolerance reached !"
            break
    print(exit_message)
    agents[0] = agents[0].detach()
    return agents, latex_h_matrix, all_derivatives, exit_message
```

Se realizó un ajuste para forzar la búsqueda del mínimo en caso de que se busque el máximo, el mismo se encuentra en la línea 59 de arriba. Este ajuste funciona mejor para posiciones de x y y positivas, por ejemplo $[0.5, 0.2]$. También se implementó el "Damping Factor" que es una especie de learning rate para que el punto no se mueva muy largo.
En la generación de las 10 corridas, se colocaron los valores de los puntos iniciales en $x = [-1, 1], y = [-0.2, 0.2]$, ya que, en esos rangos, el algoritmo funciona mejor y no tiende a "dispararse" tanto. Los mismos valores fueron seleccionados después de muchas pruebas.

Solo para fines demostrativos, se implementó una funcionalidad utilizando sympy para desplegar la construcción de la matriz Hessiana paso a paso, también funciona para otras funciones multivariables, no solo la que se utiliza en el ejercicio.

B) Reporte los resultados para 10 corridas:

1. La tolerancia fijada para la convergencia en términos de la magnitud del gradiente.

2. La cantidad de iteraciones necesarias para converger.

3. El punto de convergencia.

4. Escoga una de las corridas y en una gráfica muestre los puntos probados (visitados) por el algoritmo.

In [35]:
```
#  %%capture
#  Uncomment previous line disable prints
#  Newton-Raphson execution

convergence = -0.43 # Convergence point
damping_factor = 1  # Similar to learning rate - To deactivate set to 1
epsilon = 0.2 # Tolerance epsilon
epochs = 6 # Iteration epochs
run_with_fix = False # Execute with the manual fix
stop_at_saddle = False # Stop when saddle point is found
results = [] # Newton-Raphson results
runs = 10

#  Running with recommended ranges for initial points x = [-1, 1] y = [-0.2, 0.2]
for run in range(runs):
    print("\n---------------------")
    print(f"Run #{run + 1}\n")
    point_x = random.uniform(-1, 1)
    point_y = random.uniform(-0.2, 0.2)
    init_position = torch.Tensor([point_x, point_y])
    thetas, latex_hess_matrix, visual_derivatives, exit_message = newton_raphson(init_position, f, convergence=convergence,
                                                                                damping_factor=damping_factor,
                                                                                epsilon=epsilon, epochs=epochs,
                                                                                run_with_fix=run_with_fix,
                                                                                stop_at_saddle=stop_at_saddle)
    results.append((thetas, latex_hess_matrix, visual_derivatives, exit_message))

#  Randomly choose a run and plot
run = random.randint(0, runs - 1)

#  Example execution with fix and damping factor:
#  init_position = torch.Tensor([0.5, 0.2])
#  thetas, latex_hess_matrix, visual_derivatives, exit_message = newton_raphson(init_position, f, convergence=convergence, damping_f
#  title = "Newton-Raphson with: damping factor=$" + "0.6" + ", \\epsilon=" + str(epsilon) + \
#          "$, convergence=$" + str(convergence) + "$, run with fix=" + "True" + ", stopped due to: " + exit_message
```

```
----------------------
Run #1

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([-0.3881,  0.0908], requires_grad=True)

First Derivative: tensor([0.5961, 0.0602], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.7870, -0.1083], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.1083,  0.6512], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.7870, -0.1083],
        [-0.1083,  0.6512]], grad_fn=<CopySlices>)

The point (-0.388, 0.091) is: local minimum

Inverse Hessian Matrix: tensor([[0.5653, 0.0940],
        [0.0940, 1.5512]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[0.3426],
        [0.1494]], grad_fn=<MmBackward0>)
New agent: tensor([-0.7307, -0.0585])
The new agent is a: local minimum

---------epoch-1-------------

Agent: tensor([-0.7307, -0.0585], grad_fn=<ViewBackward0>)

First Derivative: tensor([-0.0397, -0.0500], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.6498, -0.0046], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0046,  0.8480], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.6498, -0.0046],
        [-0.0046,  0.8480]], grad_fn=<CopySlices>)

The point (-0.731, -0.059) is: local minimum

Inverse Hessian Matrix: tensor([[0.6062, 0.0033],
        [0.0033, 1.1792]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.0242],
        [-0.0590]], grad_fn=<MmBackward0>)
New agent: tensor([-7.0651e-01,  5.3607e-04])
The new agent is a: local minimum

Tolerance reached !

----------------------
Run #2

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([ 0.8021, -0.1640], requires_grad=True)

First Derivative: tensor([-0.1467,  0.1346], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.4059, -0.0481], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0481, -0.7765], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.4059, -0.0481],
        [-0.0481, -0.7765]], grad_fn=<CopySlices>)

The point (0.802, -0.164) is: local maximum

Inverse Hessian Matrix: tensor([[-0.7128,  0.0442],
        [ 0.0442, -1.2906]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[ 0.1105],
        [-0.1802]], grad_fn=<MmBackward0>)
New agent: tensor([0.6916, 0.0162])
The new agent is a: local maximum

Tolerance reached !

----------------------
Run #3

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([ 0.9193, -0.1238], requires_grad=True)

First Derivative: tensor([-0.2919,  0.0962], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.0187, -0.0723], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0723, -0.7539], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.0187, -0.0723],
        [-0.0723, -0.7539]], grad_fn=<CopySlices>)

The point (0.919, -0.124) is: local maximum

Inverse Hessian Matrix: tensor([[-0.9883,  0.0947],
        [ 0.0947, -1.3355]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[ 0.2976],
        [-0.1562]], grad_fn=<MmBackward0>)
New agent: tensor([0.6216, 0.0324])
The new agent is a: local maximum

---------epoch-1-------------

Agent: tensor([0.6216, 0.0324], grad_fn=<ViewBackward0>)

First Derivative: tensor([ 0.1542, -0.0274], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.8794, -0.0100], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0100, -0.8421], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.8794, -0.0100],
        [-0.0100, -0.8421]], grad_fn=<CopySlices>)

The point (0.622, 0.032) is: local maximum

Inverse Hessian Matrix: tensor([[-0.5321,  0.0063],
        [ 0.0063, -1.1876]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.0822],
        [ 0.0335]], grad_fn=<MmBackward0>)
```

```
New agent: tensor([ 0.7038, -0.0010])
The new agent is a: local maximum

Tolerance reached !

----------------------
Run #4

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([0.3635, 0.1867], requires_grad=True)

First Derivative: tensor([ 0.6226, -0.1148], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.6830, -0.2325], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.2325, -0.5723], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.6830, -0.2325],
        [-0.2325, -0.5723]], grad_fn=<CopySlices>)

The point (0.363, 0.187) is: local maximum

Inverse Hessian Matrix: tensor([[-0.6295,  0.2557],
        [ 0.2557, -1.8512]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.4213],
        [ 0.3718]], grad_fn=<MmBackward0>)
New agent: tensor([ 0.7848, -0.1851])
The new agent is a: local maximum

---------epoch-1-------------

Agent: tensor([ 0.7848, -0.1851], grad_fn=<ViewBackward0>)

First Derivative: tensor([-0.1210,  0.1517], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.4486, -0.0448], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0448, -0.7631], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.4486, -0.0448],
        [-0.0448, -0.7631]], grad_fn=<CopySlices>)

The point (0.785, -0.185) is: local maximum

Inverse Hessian Matrix: tensor([[-0.6916,  0.0406],
        [ 0.0406, -1.3128]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[ 0.0898],
        [-0.2040]], grad_fn=<MmBackward0>)
New agent: tensor([0.6950, 0.0189])
The new agent is a: local maximum

Tolerance reached !

----------------------
Run #5

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([-0.8272,  0.0287], requires_grad=True)

First Derivative: tensor([-0.1858,  0.0239], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([1.3604, 0.0107], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([0.0107, 0.8325], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[1.3604, 0.0107],
        [0.0107, 0.8325]], grad_fn=<CopySlices>)

The point (-0.827, 0.029) is: local minimum

Inverse Hessian Matrix: tensor([[ 0.7351, -0.0094],
        [-0.0094,  1.2013]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.1368],
        [ 0.0305]], grad_fn=<MmBackward0>)
New agent: tensor([-0.6904, -0.0018])
The new agent is a: local minimum

Tolerance reached !

----------------------
Run #6

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([-0.8343,  0.1523], requires_grad=True)

First Derivative: tensor([-0.1910,  0.1238], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([1.3070, 0.0582], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([0.0582, 0.7751], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[1.3070, 0.0582],
        [0.0582, 0.7751]], grad_fn=<CopySlices>)

The point (-0.834, 0.152) is: local minimum

Inverse Hessian Matrix: tensor([[ 0.7677, -0.0576],
        [-0.0576,  1.2945]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.1537],
        [ 0.1712]], grad_fn=<MmBackward0>)
New agent: tensor([-0.6805, -0.0189])
The new agent is a: local minimum

---------epoch-1-------------

Agent: tensor([-0.6805, -0.0189], grad_fn=<ViewBackward0>)

First Derivative: tensor([ 0.0464, -0.0162], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([1.7756e+00, 1.7576e-03], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([0.0018, 0.8556], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[1.7756e+00, 1.7576e-03],
        [1.7576e-03, 8.5562e-01]], grad_fn=<CopySlices>)

The point (-0.681, -0.019) is: local minimum
```

```
Inverse Hessian Matrix: tensor([[ 5.6319e-01, -1.1569e-03],
        [-1.1569e-03,  1.1688e+00]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[ 0.0261],
        [-0.0190]], grad_fn=<MmBackward0>)
New agent: tensor([-7.0668e-01,  6.7310e-05])
The new agent is a: local minimum

Tolerance reached !

----------------------
Run #7

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([0.5937, 0.0246], requires_grad=True)

First Derivative: tensor([ 0.2072, -0.0205], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.9144, -0.0102], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0102, -0.8332], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.9144, -0.0102],
        [-0.0102, -0.8332]], grad_fn=<CopySlices>)

The point (0.594, 0.025) is: local maximum

Inverse Hessian Matrix: tensor([[-0.5224,  0.0064],
        [ 0.0064, -1.2003]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.1084],
        [ 0.0260]], grad_fn=<MmBackward0>)
New agent: tensor([ 0.7021, -0.0014])
The new agent is a: local maximum

---------epoch-1-------------

Agent: tensor([ 0.7021, -0.0014], grad_fn=<ViewBackward0>)

First Derivative: tensor([0.0086, 0.0012], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([-1.7275e+00,  2.3356e-05], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([ 2.3356e-05, -8.5772e-01], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[-1.7275e+00,  2.3356e-05],
        [ 2.3356e-05, -8.5772e-01]], grad_fn=<CopySlices>)

The point (0.702, -0.001) is: local maximum

Inverse Hessian Matrix: tensor([[-5.7886e-01, -1.5763e-05],
        [-1.5763e-05, -1.1659e+00]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.0050],
        [-0.0014]], grad_fn=<MmBackward0>)
New agent: tensor([7.0709e-01, 1.4063e-07])
The new agent is a: local maximum

Tolerance reached !

----------------------
Run #8

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
```

```
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([-0.9462,  0.1650], requires_grad=True)

First Derivative: tensor([-0.3143,  0.1241], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([0.9099, 0.1037], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([0.1037, 0.7113], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[0.9099, 0.1037],
        [0.1037, 0.7113]], grad_fn=<CopySlices>)

The point (-0.946, 0.165) is: local minimum

Inverse Hessian Matrix: tensor([[ 1.1176, -0.1630],
        [-0.1630,  1.4296]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.3714],
        [ 0.2287]], grad_fn=<MmBackward0>)
New agent: tensor([-0.5747, -0.0637])
The new agent is a: local minimum

---------epoch-1-------------

Agent: tensor([-0.5747, -0.0637], grad_fn=<ViewBackward0>)

First Derivative: tensor([ 0.2429, -0.0524], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([1.9248, 0.0309], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([0.0309, 0.8161], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[1.9248, 0.0309],
        [0.0309, 0.8161]], grad_fn=<CopySlices>)

The point (-0.575, -0.064) is: local minimum

Inverse Hessian Matrix: tensor([[ 0.5199, -0.0197],
        [-0.0197,  1.2261]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[ 0.1273],
        [-0.0690]], grad_fn=<MmBackward0>)
New agent: tensor([-0.7020,  0.0053])
The new agent is a: local minimum

---------epoch-2-------------

Agent: tensor([-0.7020,  0.0053], grad_fn=<ViewBackward0>)

First Derivative: tensor([0.0087, 0.0046], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.7276e+00, -9.3148e-05], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-9.3148e-05,  8.5765e-01], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.7276e+00, -9.3148e-05],
        [-9.3148e-05,  8.5765e-01]], grad_fn=<CopySlices>)

The point (-0.702, 0.005) is: local minimum

Inverse Hessian Matrix: tensor([[5.7883e-01, 6.2866e-05],
        [6.2866e-05, 1.1660e+00]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[0.0050],
        [0.0053]], grad_fn=<MmBackward0>)
New agent: tensor([-7.0709e-01, -8.5356e-07])
The new agent is a: local minimum

Tolerance reached !

----------------------
Run #9

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
```

```
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([-0.4339,  0.1135], requires_grad=True)

First Derivative: tensor([0.5098, 0.0805], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.8618, -0.1157], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.1157,  0.6914], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.8618, -0.1157],
        [-0.1157,  0.6914]], grad_fn=<CopySlices>)

The point (-0.434, 0.113) is: local minimum

Inverse Hessian Matrix: tensor([[0.5427, 0.0908],
        [0.0908, 1.4615]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[0.2840],
        [0.1640]], grad_fn=<MmBackward0>)
New agent: tensor([-0.7179, -0.0505])
The new agent is a: local minimum

---------epoch-1-------------

Agent: tensor([-0.7179, -0.0505], grad_fn=<ViewBackward0>)

First Derivative: tensor([-0.0184, -0.0432], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.6843, -0.0019], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0019,  0.8510], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.6843, -0.0019],
        [-0.0019,  0.8510]], grad_fn=<CopySlices>)

The point (-0.718, -0.051) is: local minimum

Inverse Hessian Matrix: tensor([[0.5937, 0.0013],
        [0.0013, 1.1751]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.0110],
        [-0.0508]], grad_fn=<MmBackward0>)
New agent: tensor([-7.0696e-01,  2.8352e-04])
The new agent is a: local minimum

Tolerance reached !

----------------------
Run #10

Building column #1 and row #1
df/dx=-2*x**2*exp(-x**2 - y**2) + exp(-x**2 - y**2)
df/dxdx=4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
df/dydx=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dxdy=4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
Building column #2 and row #2
df/dy=-2*x*y*exp(-x**2 - y**2)
df/dydy=4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)
Hessian Matrix: [[4*x**3*exp(-x**2 - y**2) - 6*x*exp(-x**2 - y**2)
  4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)]
 [4*x**2*y*exp(-x**2 - y**2) - 2*y*exp(-x**2 - y**2)
  4*x*y**2*exp(-x**2 - y**2) - 2*x*exp(-x**2 - y**2)]]

Hessian Matrix in Latex: $H=\begin{pmatrix}4 x^{3} e^{- x^{2} - y^{2}} - 6 x e^{- x^{2} - y^{2}} & 4 x^{2} y e^{- x^{2} - y^{2}} -
2 y e^{- x^{2} - y^{2}}\\4 x^{2} y e^{- x^{2} - y^{2}} - 2 y e^{- x^{2} - y^{2}} & 4 x y^{2} e^{- x^{2} - y^{2}} - 2 x e^{- x^{2} -
y^{2}}\\\end{pmatrix}$

---------epoch-0-------------

Agent: tensor([-0.3797,  0.0742], requires_grad=True)

First Derivative: tensor([0.6127, 0.0485], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.7730, -0.0909], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0909,  0.6467], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.7730, -0.0909],
        [-0.0909,  0.6467]], grad_fn=<CopySlices>)

The point (-0.38, 0.074) is: local minimum

Inverse Hessian Matrix: tensor([[0.5681, 0.0799],
        [0.0799, 1.5576]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[0.3520],
        [0.1245]], grad_fn=<MmBackward0>)
New agent: tensor([-0.7317, -0.0503])
The new agent is a: local minimum

---------epoch-1-------------

Agent: tensor([-0.7317, -0.0503], grad_fn=<ViewBackward0>)

First Derivative: tensor([-0.0413, -0.0430], grad_fn=<AddBackward0>)
Second derivative on dimension 0: tensor([ 1.6488, -0.0042], grad_fn=<AddBackward0>)
Second derivative on dimension 1: tensor([-0.0042,  0.8503], grad_fn=<AddBackward0>)
Hessian Matrix: tensor([[ 1.6488, -0.0042],
        [-0.0042,  0.8503]], grad_fn=<CopySlices>)

The point (-0.732, -0.05) is: local minimum

Inverse Hessian Matrix: tensor([[0.6065, 0.0030],
        [0.0030, 1.1761]], grad_fn=<NanToNumBackward0>)

Gradient: tensor([[-0.0252],
        [-0.0507]], grad_fn=<MmBackward0>)
New agent: tensor([-7.0651e-01,  3.7859e-04])
The new agent is a: local minimum

Tolerance reached !
```

In [36]:
```python
# Plot separated so that console prints are not shown alongside !!

#  Comment if using example execution
thetas, latex_hess_matrix, visual_derivatives, exit_message = results[run]

# Visualization of Hessian Matrix - Rerun if not being shown !!!!
display(Markdown("Hessian Matrix construction: "))
for visual_derivative in visual_derivatives:
    display(Markdown(visual_derivative))
display(Markdown(latex_hess_matrix))

# Plot Newton-Raphson

#  Comment if using example execution
title = "Newton-Raphson with: damping factor=$" + str(damping_factor) + ", \\epsilon=" + str(epsilon) + \
        "$, convergence=$" + str(convergence) + "$, run with fix=" + str(run_with_fix) + ", stopped due to: " + exit_message
plot(thetas, title)
```

Hessian Matrix construction:

$df/dx = -2x^2e^{-x^2-y^2} + e^{-x^2-y^2}$

$df/dxdx = 4x^3e^{-x^2-y^2} - 6xe^{-x^2-y^2}$

$df/dydx = 4x^2ye^{-x^2-y^2} - 2ye^{-x^2-y^2}$

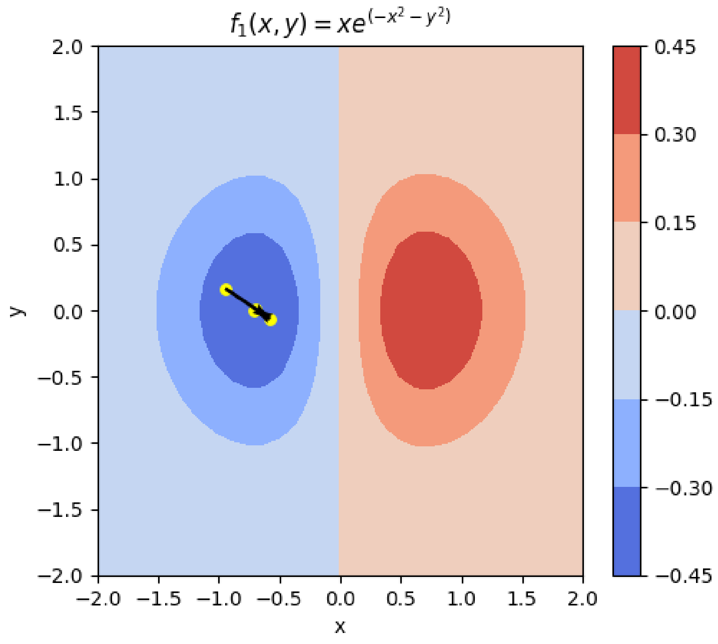
$df/dy = -2xye^{-x^2-y^2}$

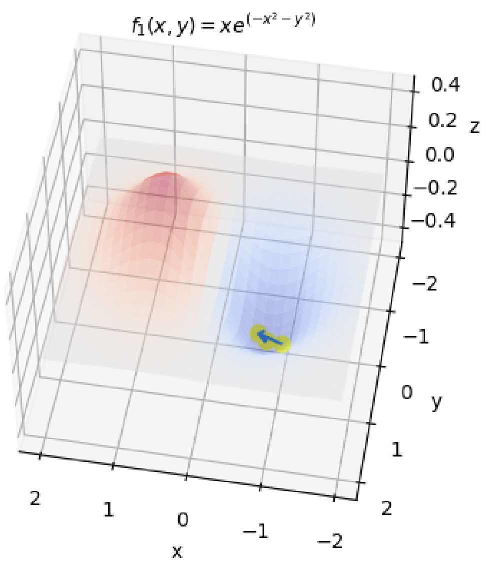
$df/dxdy = 4x^2ye^{-x^2-y^2} - 2ye^{-x^2-y^2}$

$df/dy = -2xye^{-x^2-y^2}$

$df/dydy = 4xy^2e^{-x^2-y^2} - 2xe^{-x^2-y^2}$

$$H = \begin{pmatrix} 4x^3e^{-x^2-y^2} - 6xe^{-x^2-y^2} & 4x^2ye^{-x^2-y^2} - 2ye^{-x^2-y^2} \\ 4x^2ye^{-x^2-y^2} - 2ye^{-x^2-y^2} & 4xy^2e^{-x^2-y^2} - 2xe^{-x^2-y^2} \end{pmatrix}$$





Newton-Raphson with: damping factor=1, $\varepsilon = 0.2$, convergence=−0.43, run with fix=False, stopped due to: Tolerance reached !

epoch 0: $\theta_0 = -0.946, 0.165$	$f(\theta_0) = -0.376$
epoch 1: $\theta_1 = -0.575, -0.064$	$f(\theta_1) = -0.411$
epoch 2: $\theta_2 = -0.702, 0.005$	$f(\theta_2) = -0.429$
epoch 3: $\theta_3 = -0.707, -0.0$	$f(\theta_3) = -0.429$

4. Investigue y reporte las principales ventajas y desventajas, usando los resultados obtenidos, del algoritmo Newton Raphson respecto al algoritmo del descenso del gradiente con moméntum, citando adecuadamente las referencias.

Una gran desventaja del algoritmo de Newton-Raphson es que presenta una mayor tendencia a divergir, esto sucede cuando la matriz Hessiana no es positiva definitiva, osea que todos sus valores sean positivos [1]. A pesar de eso, en algunos casos se puede continuar calculando el punto óptimo estimado y se puede modificar la dirección para asegurar el descenso, sin embargo, Fletcher [2] menciona que el punto estacionario de la función cuadrática de aproximación (la aproximación de Taylor) no es un punto de minimización y la relevancia de buscar en esa dirección es cuestionable; también alude que el método básico de Newton, tal como está, no es adecuado para un algoritmo de propósito general, ya que la Hessiana puede no ser positiva definitiva cuando $x(t)$ está lejos de la solución.

Lo mencionado anteriormente fue evidente en los experimentos, especialmente aquellos donde la posición inicial estaba fuera de los rangos $x = [-1, 1], y = [-0.2, 0.2]$, por eso, en el arreglo que se implementó al algoritmo, cuando detectamos que el nuevo punto es mayor al de la iteración anterior, modificamos la Hessiana para que sea positiva definitiva, con esto se le dá mejor dirección al algoritmo.

Otra desventaja del Newton-Raphson es el costo computacional para calcular la Hessiana, menciona LeCun et al. [1] que, uno de los principales inconvenientes es que se debe almacenar e invertir una matriz NxN Hessiana, lo que requiere $O(N^3)$ iteraciones.

Una ventaja del algoritmo de descenso del gradiente con momentum, además no presentar las desventajas mencionadas anteriormente, como menciona LeCun et al. [1], puede aumentar la velocidad cuando la superficie de costo es altamente no esférica porque amortigua el tamaño de los pasos a lo largo de las direcciones de alta curvatura, lo que produce una mayor tasa de aprendizaje efectiva a lo largo de las direcciones de baja curvatura. En los experimentos realizados, esto resulta ser cierto pero depende mucho del coeficiente del momentum y del learning rate, ya que hemos visto instancias que con o sin momentum se llega a la convergencia en la misma cantidad de epochs.

Referencias:

[1] Y. LeCun, L. Bottou, G. B. Orr, and K. Müller, “Efficient backprop”, Lecture Notes in Computer Science, pp. 9–50, 1998. DOI: 10.1007/3-540-49430-8 2.

[2] R. Fletcher, “Newton-like methods”, in Practical methods of optimization. 2nd edition. John Wiley amp; Sons, 1987.