List 01 - Result

Tags: #fleeting #literature #permanent
Description:
Theme:

github : https://github.com/viniciusrondon/PO-245---Neural-Net-and-LLM---Prof.-Mauri.git

ID: 20250813100331

Description

Apply the steepest descent algorithm to the function, using step size

$$\lambda = 0.1,$$

and with the initial point given by:

$$f(x,y) = xye^{-(x^2+y^2)}, \quad x = 0.3, \quad y = 1.2$$

Solve the exercise step-by-step, showing the calculations **up to the second iteration**.

Tasks to perform:

a) Plot the function and the initial point.
b) Plot the gradient vector in the direction of the minimum, starting from the initial point.
c) Create a Python routine considering the `initial error value` equal to

$$error = 1 \quad (\text{eps} = 1),$$

and run the loop while

$$\text{error} > 0.00001.$$

d) After running the routine from item (c), plot the function and the minimum point found.

Additional points will be awarded for solutions that show the "path" to the local minimum.

---

a) Plot the function and the initial point.

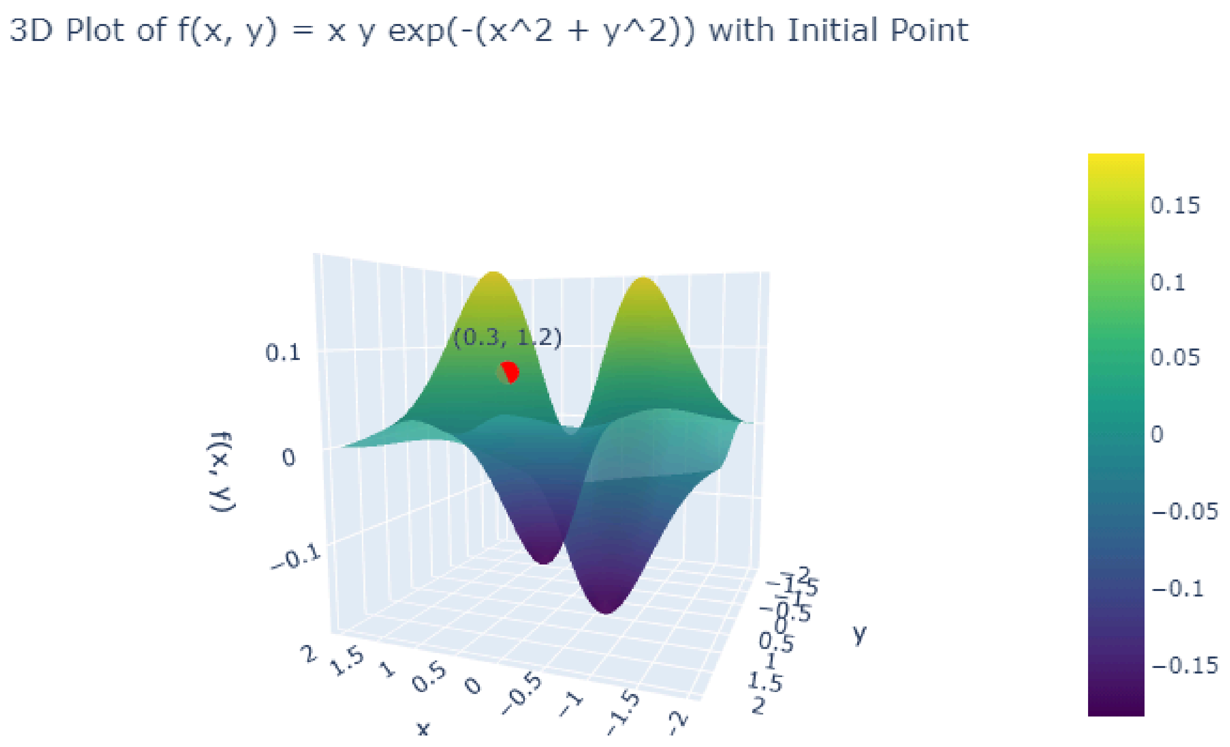

Figure 01: Illusterates the $f(x, y)$ surface and the red dot represent the initial point $(x_0 = 0.3 \quad y_0 = 1.2)$

```
# Define the function
def f(x, y):
    return x * y * np.exp(-(x**2 + y**2))

# Create a grid of x and y values
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

# Create the surface plot
surface = go.Surface(x=X, y=Y, z=Z, colorscale='Viridis', opacity=0.8,
name='f(x, y)')


# Plot the initial point
x0, y0 = 0.3, 1.2
z0 = f(x0, y0)

point = go.Scatter3d(
    x=[x0],
```

```python
    y=[y0],
    z=[z0],
    mode='markers+text',
    marker=dict(size=7, color='red'),
    text=['(0.3, 1.2)'],
    textposition='top center',
    name='Initial Point'
)


# Layout
layout = go.Layout(
    title='3D Plot of f(x, y) = x y exp(-(x^2 + y^2)) with Initial Point',
    scene=dict(
        xaxis_title='x',
        yaxis_title='y',
        zaxis_title='f(x, y)'
    ),
    width=700,
    height=500
)

fig = go.Figure(data=[surface, point], layout=layout)
fig.show()
```

---

b) Plot the gradient vector in the direction of the minimum, starting from the initial point.

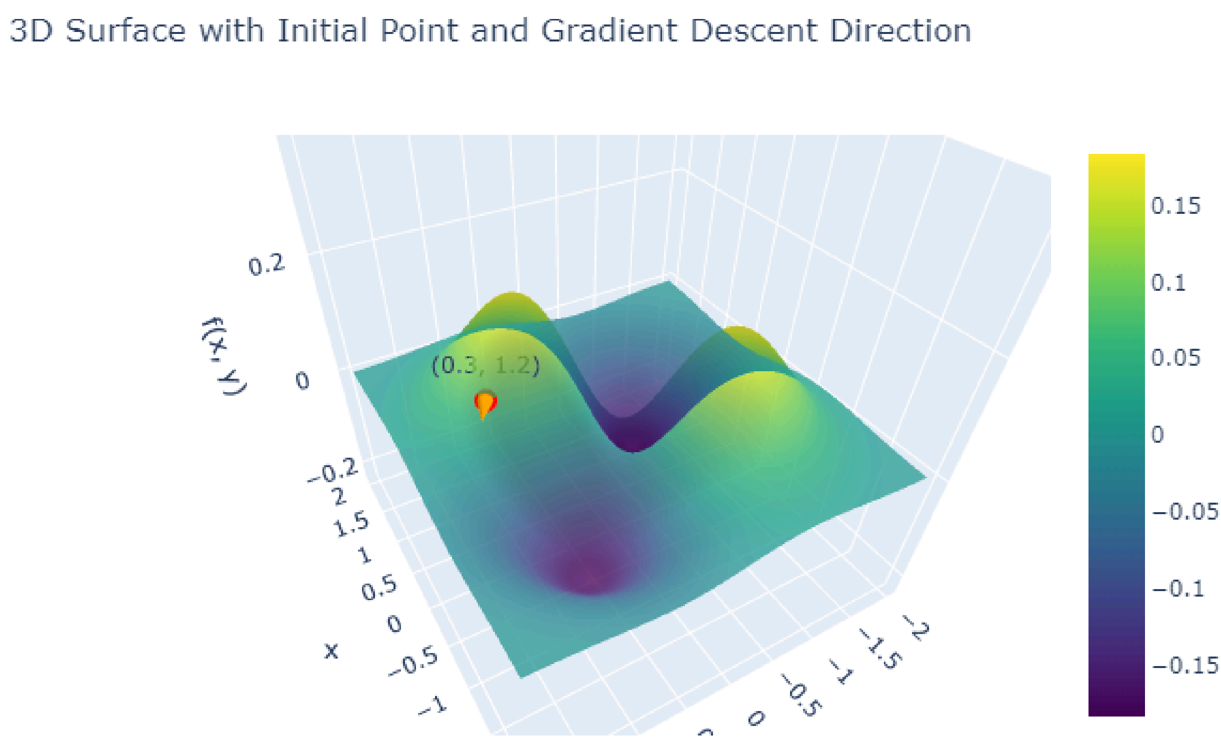

Figure 02: Function Surface with Gradient Descend Vector

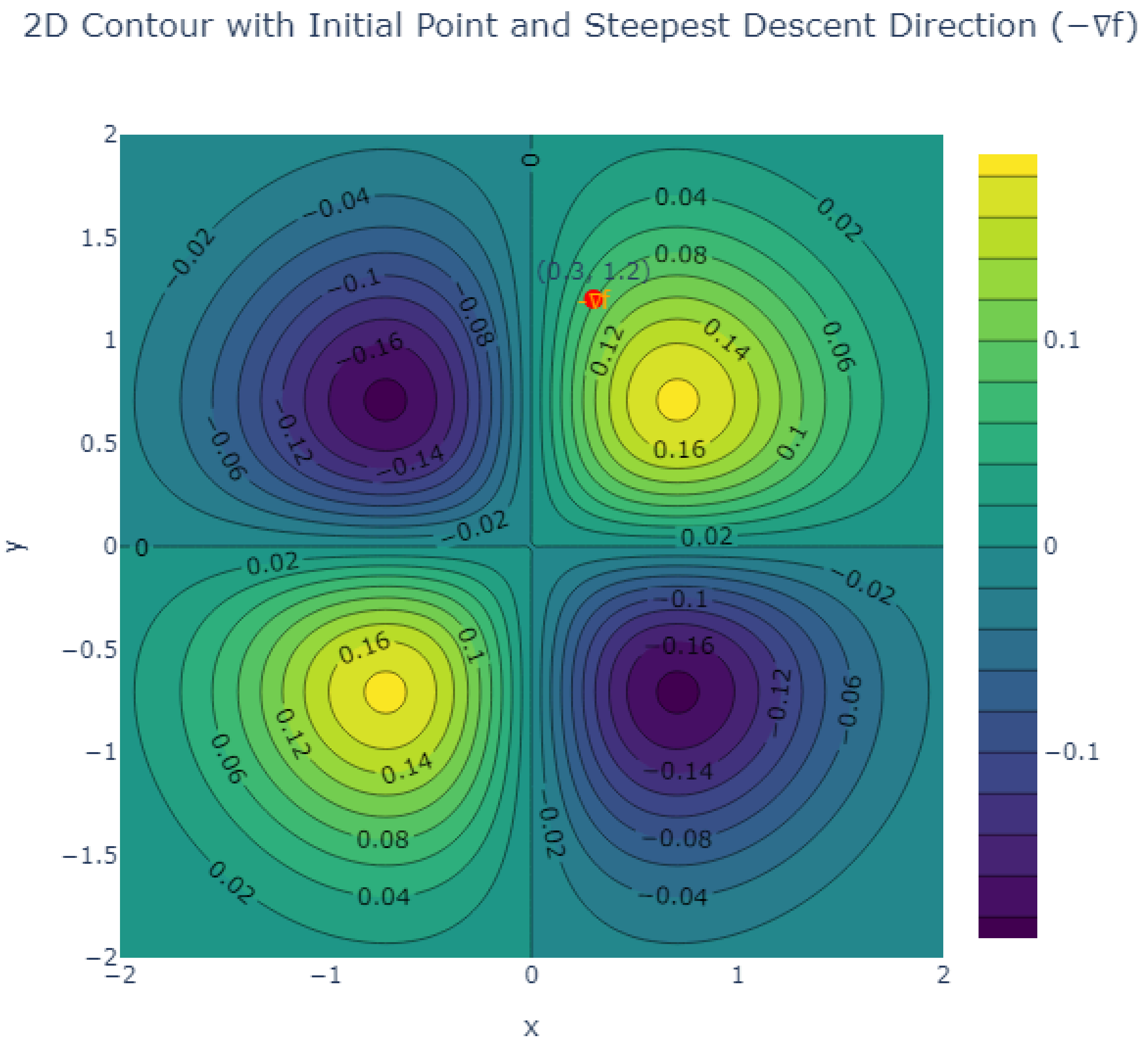

Figure 03: 2D contour plot and the Steepest Descend Direction based on initial point $(x_0 = 0.3 \quad y_0 = 1.2)$

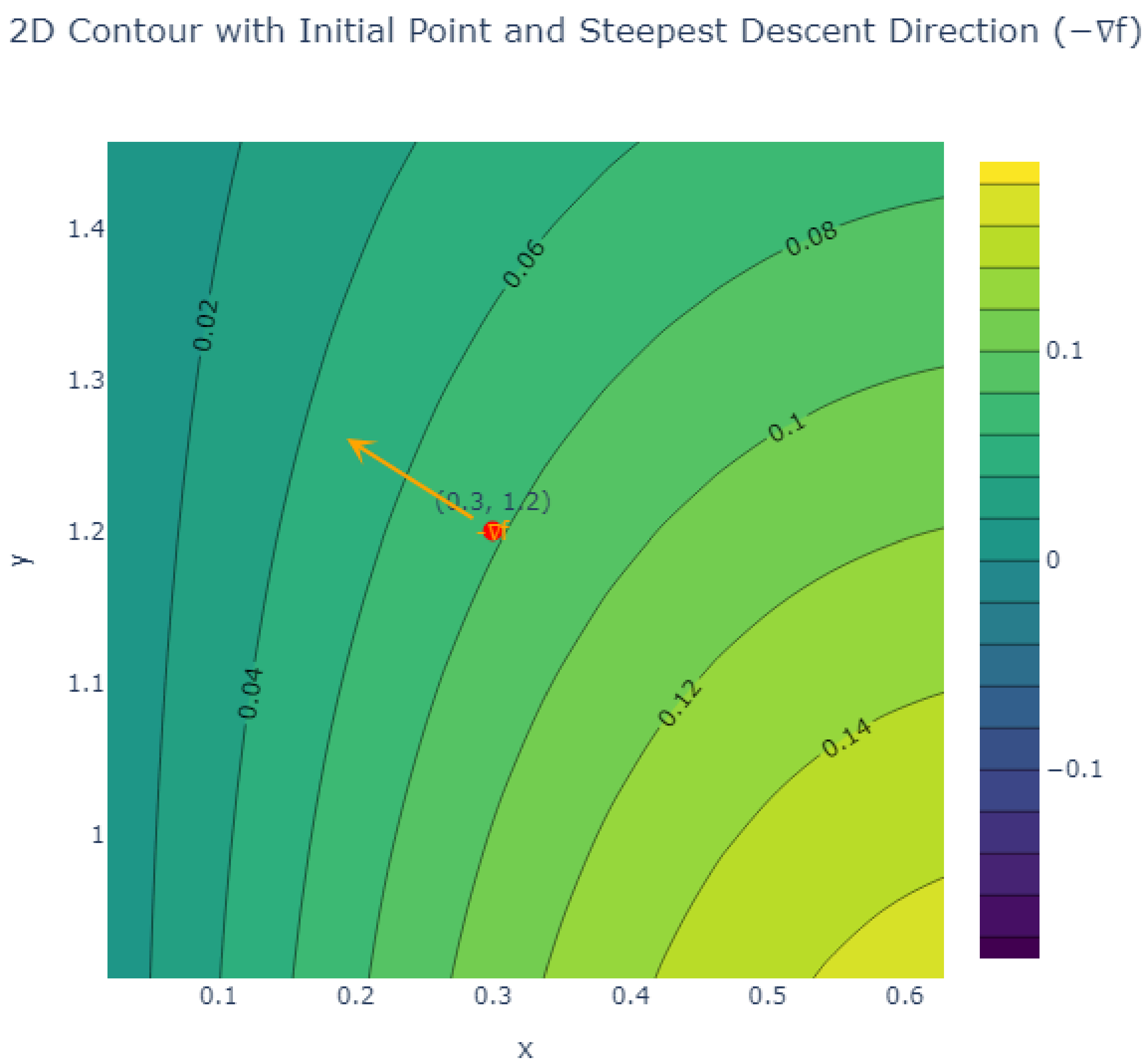

Figure 04: Gradient Vector

```
# Plotly 3D surface and 2D contour with gradient descent direction

import numpy as np
import plotly.graph_objs as go

# Function and gradient
def f(x, y):
    return x * y * np.exp(-(x**2 + y**2))

def grad_f(x, y):
    r2 = x**2 + y**2
    ex = np.exp(-r2)
    dfx = y * ex * (1 - 2*x**2)
    dfy = x * ex * (1 - 2*y**2)
    return np.array([dfx, dfy])

# Initial point
```

```
x0, y0 = 0.3, 1.2
g0 = grad_f(x0, y0)
desc_dir0 = -g0  # steepest descent direction

# Plot 1: 3D Surface with initial point and gradient vector (Plotly) =====
xs = np.linspace(-2, 2, 100)
ys = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(xs, ys)
Z = f(X, Y)

# Surface
surface = go.Surface(x=X, y=Y, z=Z, colorscale='Viridis', opacity=0.8,
name='f(x, y)')

# Initial point
z0 = f(x0, y0)
point = go.Scatter3d(
    x=[x0],
    y=[y0],
    z=[z0],
    mode='markers+text',
    marker=dict(size=7, color='red'),
    text=['(%.1f, %.1f)' % (x0, y0)],
    textposition='top center',
    name='Initial Point'
)

# Gradient descent vector (as an arrow)
scale3d = 0.5  # scale for visibility
arrow3d = go.Cone(
    x=[x0],
    y=[y0],
    z=[z0],
    u=[scale3d * desc_dir0[0]],
    v=[scale3d * desc_dir0[1]],
    w=[0],  # show in x-y plane
    sizemode="absolute",
    sizeref=0.3,
    anchor="tail",
    showscale=False,
    colorscale=[[0, 'orange'], [1, 'orange']],
    name='Steepest Descent'
)

layout3d = go.Layout(
    title='3D Surface with Initial Point and Gradient Descent Direction',
    scene=dict(
        xaxis_title='x',
        yaxis_title='y',
        zaxis_title='f(x, y)'
```

```python
    ),
    width=700,
    height=500,
    showlegend=False
)

fig3d = go.Figure(data=[surface, point, arrow3d], layout=layout3d)
fig3d.show()

# Plot 2: 2D Contour with gradient arrow (Plotly) =====
import plotly.figure_factory as ff

# Contour
contour = go.Contour(
    x=xs,
    y=ys,
    z=Z,
    ncontours=20,
    colorscale='Viridis',
    showscale=True,
    contours=dict(showlabels = True)
)

# Initial point
point2d = go.Scatter(
    x=[x0],
    y=[y0],
    mode='markers+text',
    marker=dict(size=10, color='red'),
    text=['(%.1f, %.1f)' % (x0, y0)],
    textposition='top center',
    name='Initial Point'
)

# Arrow for gradient descent direction
scale2d = 0.5
arrow2d = go.layout.Annotation(
    x=x0 + scale2d * desc_dir0[0],
    y=y0 + scale2d * desc_dir0[1],
    ax=x0,
    ay=y0,
    xref='x',
    yref='y',
    axref='x',
    ayref='y',
    showarrow=True,
    arrowhead=3,
    arrowsize=1.2,
    arrowwidth=2,
    arrowcolor='orange',
```

```
    text='-∇f',
    font=dict(color='orange')
)

layout2d = go.Layout(
    title="2D Contour with Initial Point and Steepest Descent Direction (-∇f)",
    xaxis_title="x",
    yaxis_title="y",
    width=600,
    height=600,
    showlegend=False,
    annotations=[arrow2d]
)

fig2d = go.Figure(data=[contour, point2d], layout=layout2d)
fig2d.show()

# Also print numerical gradient and direction magnitude for reference
print("Gradient at initial point:", g0)
print("Norm of gradient:", np.linalg.norm(g0))
```

Compute the gradient of :

$$f(x,y) = x\,y\,e^{-(x^2+y^2)}. \tag{01}$$

Intuition

product rule for $xy \cdot (\cdot)$ and **chain rule** for the exponential.

Partial derivatives

Let $r^2 = x^2 + y^2$. Then

$$\frac{\partial}{\partial x} e^{-r^2} = e^{-r^2} \cdot (-2x), \qquad \frac{\partial}{\partial y} e^{-r^2} = e^{-r^2} \cdot (-2y). \tag{02}$$

- For x:

$$\frac{\partial f}{\partial x} = \underbrace{y}_{\partial(xy)/\partial x} e^{-r^2} + xy \cdot e^{-r^2}(-2x) = e^{-r^2}\left(y - 2x^2 y\right) = y\,e^{-r^2}\,(1 - 2x^2). \tag{03}$$

- For y:

$$\frac{\partial f}{\partial y} = \underbrace{x}_{\partial(xy)/\partial y} e^{-r^2} + xy \cdot e^{-r^2}(-2y) = e^{-r^2}\left(x - 2xy^2\right) = x\,e^{-r^2}\,(1 - 2y^2). \tag{04}$$

Gradient (compact form)

$$\boxed{\nabla f(x,y) = \big(\, y\, e^{-(x^2+y^2)}(1-2x^2),\ \ x\, e^{-(x^2+y^2)}(1-2y^2)\,\big)} \tag{05}$$

---

c) Create a Python routine considering the `initial error value` equal to

$$error = 1 \quad (\text{eps} = 1),$$

and run the loop while

$$\text{error} > 0.00001.$$

iter	x	y	f(x,y)	‖grad‖	eps (=‖Δθ‖)
0	0.300000	1.200000	0.077953	0.245589	NaN
1	0.278693	1.212213	0.071911	0.246436	0.024559
2	0.256898	1.223715	0.065831	0.246968	0.024644
3	0.234655	1.234447	0.059729	0.247230	0.024697
4	0.212004	1.244355	0.053617	0.247267	0.024723
5	0.188987	1.253390	0.047505	0.247124	0.024727
6	0.165646	1.261508	0.041402	0.246848	0.024712
7	0.142024	1.268672	0.035314	0.246479	0.024685
8	0.118163	1.274848	0.029244	0.246058	0.024648
9	0.094104	1.280011	0.023196	0.245619	0.024606

Table 01: Gradient iterations

Gradient Norm in the Table

At each iteration, was calculated

$$\|grad\| = \|\nabla f(x_k, y_k)\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{07}$$

This is a measure of **how steep the slope** is at the current point.

- If the gradient norm is large → we are far from a minimum (steep slope).
- If the gradient norm is close to zero → we are near a stationary point (possible min/max/saddle).

`eps` in the Table

$$\text{eps} = \|(x_{k+1}, y_{k+1}) - (x_k, y_k)\| \tag{08}$$

This is the **magnitude of the update step** (how much the point moved in one iteration).

It should start **larger** at the beginning and **shrink** as approaching the minimum.

`tol`

`tol = 1e-5` is the **tolerance threshold** for stopping.

- Continue iterating **while** the parameter change (`eps`) is **bigger** than the tolerance.
- Once `eps` becomes smaller than `1e-5`, the loop ends — this means our updates are so tiny that we consider the algorithm converged.

```
# Step 4 — Python routine: run steepest descent until error <= 1e-5, track
path, and plot (using plotly)
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px

# Function and gradient
def f(x, y):
    return x*y*np.exp(-(x**2 + y**2))

def grad_f(x, y):
    r2 = x**2 + y**2
    ex = np.exp(-r2)
    dfx = y * ex * (1 - 2*x**2)
    dfy = x * ex * (1 - 2*y**2)
    return np.array([dfx, dfy])

# Parameters
lam = 0.1           # fixed step size
tol = 1e-5          # stopping tolerance
x, y = 0.3, 1.2     # initial point
eps = 1.0           # initial error as requested
max_iter = 1_000_000 # robust cap

# Storage for path
xs = [x]
ys = [y]
fs = [f(x, y)]
grad_norms = [np.linalg.norm(grad_f(x, y))]
errs = []

# Iteration
k = 0
while eps > tol and k < max_iter:
    g = grad_f(x, y)
```

```
    step = lam * g
    x_new = x - step[0]
    y_new = y - step[1]
    eps = np.linalg.norm([x_new - x, y_new - y])  # error = parameter change
magnitude
    x, y = x_new, y_new
    xs.append(x)
    ys.append(y)
    fs.append(f(x, y))
    grad_norms.append(np.linalg.norm(grad_f(x, y)))
    errs.append(eps)
    k += 1

final_point = (x, y)
final_value = f(x, y)
final_grad_norm = np.linalg.norm(grad_f(x, y))

# Build iteration table
data = {
    "iter": np.arange(len(xs)),
    "x": xs,
    "y": ys,
    "f(x,y)": fs,
}
# pad errs to match length (first iter has no eps defined)
eps_series = [np.nan] + errs
grad_series = grad_norms
data["||grad||"] = grad_series
data["eps (=||Δθ||)"] = eps_series
df = pd.DataFrame(data)

# Show a preview table to the user (using display if available, else print)
try:
    from IPython.display import display
    display(df.head(10))
except ImportError:
    print(df.head(10))

# ===== Plot A: Contour with full path (Plotly) =====
xs_grid = np.linspace(-2, 2, 300)
ys_grid = np.linspace(-2, 2, 300)
X, Y = np.meshgrid(xs_grid, ys_grid)
Z = f(X, Y)

# Create contour plot
contour = go.Contour(
    z=Z,
    x=xs_grid,
    y=ys_grid,
    ncontours=30,
```

```
    colorscale='Viridis',
    showscale=True,
    contours_coloring='lines',
    line_width=1,
    name='Contours'
)

# Path trace
path_trace = go.Scatter(
    x=xs,
    y=ys,
    mode='lines+markers',
    marker=dict(size=4, color='red'),
    line=dict(color='red', width=2),
    name='Descent Path'
)

# Start and end points
start_point = go.Scatter(
    x=[xs[0]],
    y=[ys[0]],
    mode='markers',
    marker=dict(size=10, color='blue', symbol='circle'),
    name='Start'
)

end_point = go.Scatter(
    x=[xs[-1]],
    y=[ys[-1]],
    mode='markers',
    marker=dict(size=12, color='green', symbol='star'),
    name='End'
)

fig_contour = go.Figure(data=[contour, path_trace, start_point, end_point])
fig_contour.update_layout(
    title="Steepest Descent Path on Contours of f(x,y)",
    xaxis_title="x",
    yaxis_title="y",
    width=700,
    height=700,
    legend=dict(x=0.01, y=0.99)
)
fig_contour.show()

# ===== Plot B: 3D surface with final point (Plotly) =====
surface = go.Surface(
    x=xs_grid,
    y=ys_grid,
    z=Z,
```

```
    colorscale='Viridis',
    opacity=0.9,
    showscale=True,
    name='f(x,y)'
)

# Path in 3D
path3d = go.Scatter3d(
    x=xs,
    y=ys,
    z=fs,
    mode='lines+markers',
    marker=dict(size=3, color='red'),
    line=dict(color='red', width=3),
    name='Descent Path'
)


# Final point in 3D
final3d = go.Scatter3d(
    x=[xs[-1]],
    y=[ys[-1]],
    z=[fs[-1]],
    mode='markers',
    marker=dict(size=8, color='green', symbol='diamond'),
    name='Found Minimum'
)


fig_surface = go.Figure(data=[surface, path3d, final3d])
fig_surface.update_layout(
    title="Surface: f(x,y) with Found Minimum Marked",
    scene=dict(
        xaxis_title='x',
        yaxis_title='y',
        zaxis_title='f(x,y)',
        aspectmode='cube'
    ),
    width=800,
    height=600,
    legend=dict(x=0.01, y=0.99)
)

fig_surface.show()
final_point, final_value, final_grad_norm, k
```

d) After running the routine from item (c), plot the function and the minimum point found.

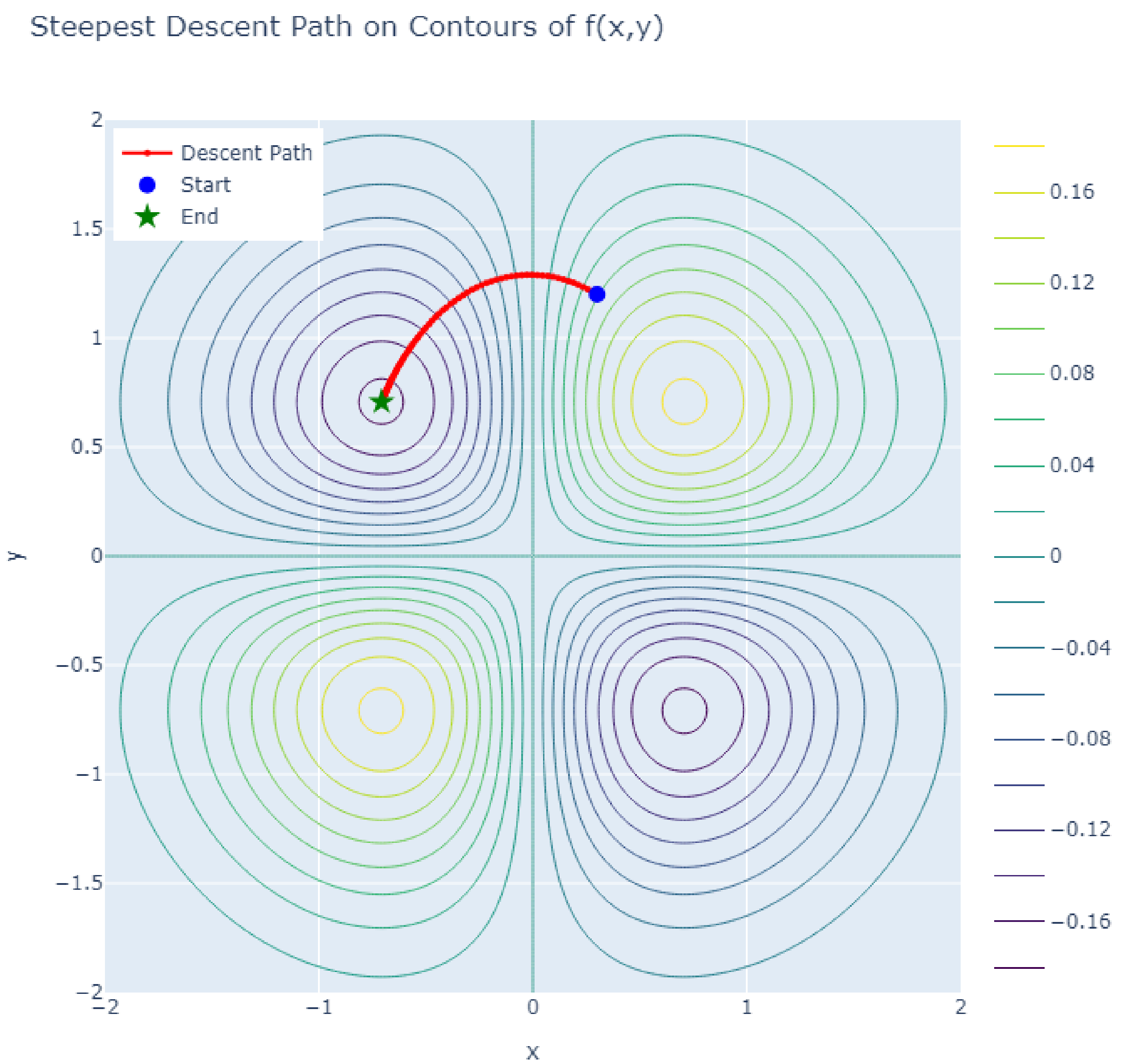

Figure 05: 2D Contour Plot illustrating the Steepest Descend Path

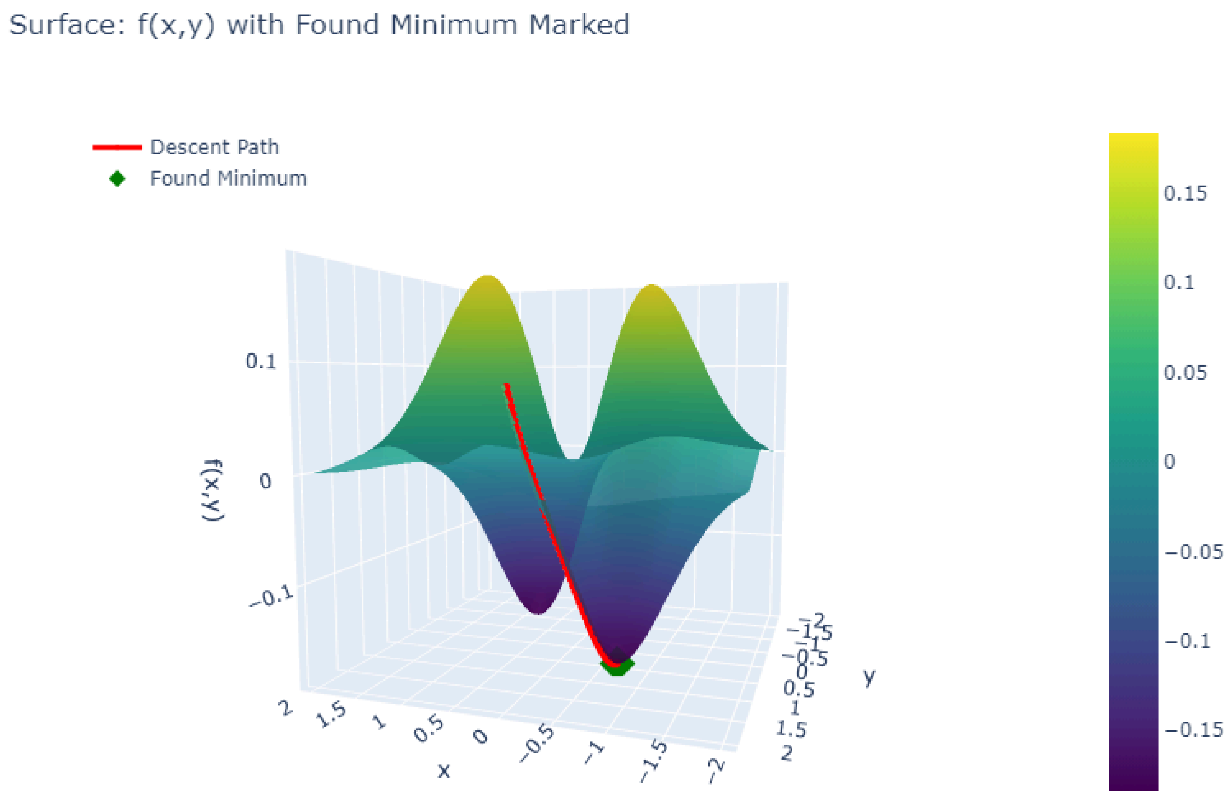

Figure 05: Surface with looking for the Local Minimun

With $\lambda = 0.1$ starting at $(x_0, y_0) = (0.3,\ 1.2)$.

$$\nabla f(x,y) = \Big(y\, e^{-(x^2+y^2)}(1-2x^2),\ x\, e^{-(x^2+y^2)}(1-2y^2)\Big), \quad (x_{k+1}, y_{k+1}) = (x_k, y_k) - \lambda\, \nabla f(x_k, y_k$$

Iteration 0 $\to$ 1

- $r_0^2 = x_0^2 + y_0^2 = 0.09 + 1.44 = 1.53$, hence $e^{-r_0^2} = e^{-1.53} \approx 0.2165356673$.
- Gradient at (x_0, y_0):

$$\frac{\partial f}{\partial x}(x_0,y_0) = y_0\, e^{-r_0^2}(1-2x_0^2) = 1.2 \cdot 0.2165356673 \cdot (1 - 2\cdot 0.09) \approx \mathbf{0.2130710966},$$
$$\frac{\partial f}{\partial y}(x_0,y_0) = x_0\, e^{-r_0^2}(1-2y_0^2) = 0.3 \cdot 0.2165356673 \cdot (1 - 2\cdot 1.44) \approx \mathbf{-0.1221261164}. \quad (\text{:}$$

  Gradient norm $\|\nabla f(x_0, y_0)\| \approx 0.2455892516$.

- Update with $\lambda = 0.1$:

$$\boxed{\begin{aligned} x_1 &= x_0 - 0.1\,(\partial f/\partial x) \approx 0.3 - 0.1(0.2130710966) = \mathbf{0.2786928903}, \\ y_1 &= y_0 - 0.1\,(\partial f/\partial y) \approx 1.2 - 0.1(-0.1221261164) = \mathbf{1.2122126116}. \end{aligned}} \qquad (11)$$

- Function values:

$$f(x_0,y_0) = x_0 y_0 e^{-r_0^2} \approx \mathbf{0.0779528402}, \qquad f(x_1,y_1) \approx \mathbf{0.0719109520}\ (\downarrow). \qquad (12)$$

Iteration 1 → 2

- Compute $r_1^2 = x_1^2 + y_1^2$ and then $\nabla f(x_1, y_1)$:

$$\nabla f\big|_{(x_1,y_1)} \approx \big(\mathbf{0.2179472317},\ -\mathbf{0.1150206636}\big), \quad \|\nabla f(x_1, y_1)\| \approx 0.2464360949. \quad (13)$$

- Update:

$$\boxed{\begin{aligned} x_2 &= x_1 - 0.1 \cdot 0.2179472317 \approx \mathbf{0.2568981672}, \\ y_2 &= y_1 - 0.1 \cdot (-0.1150206636) \approx \mathbf{1.2237146780}. \end{aligned}} \tag{14}$$

- Function value:

$$f(x_2, y_2) \approx \mathbf{0.0658313744}\ (\downarrow). \tag{16}$$

So after two iterations with fixed step $\lambda = 0.1$, we have:

$$(x_0, y_0) = (0.3, 1.2)\ \rightarrow\ (x_1, y_1) \approx (0.2786928903,\ 1.2122126116)\ \rightarrow\ (x_2, y_2) \approx (0.2568981672$$

Go deep into **gradient norm**

$$n = \frac{\nabla f(x, y)}{\|\nabla f(x, y)\|} \tag{18}$$

Given a scalar field $\mathbb{R}^n \to \mathbb{R}$, its **gradient** is:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ \dots, \ \frac{\partial f}{\partial x_n} \right). \tag{19}$$

The gradient vector **points in the direction of greatest increase** of f at that point.

Gradient norm:

$$\|\nabla f(x)\| = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i} \right)^2} \tag{20}$$

This is simply the **length** (Euclidean magnitude) of that vector.

- **Large gradient norm** → steep slope (function changes rapidly if moving in gradient's direction).
- **Small gradient norm** → nearly flat region (candidate for a local min, max, or saddle point).
- **Optimization stopping criteria**: Many algorithms stop when $\|\nabla f\| \le \mathrm{tol}$ because that means there's almost no slope left to descend.

Steepest Descent

The gradient gives both *direction* and *magnitude* of steepest ascent.

If divided by its norm, you remove the magnitude — leaving a **pure direction vector** with length 1.

$$n = \frac{\nabla f(x, y)}{\|\nabla f(x, y)\|}$$

In steepest descent, the **direction** we move is $-\nabla f$ (opposite to ascent). If wanted to normalize it, we could write:

$$\mathbf{d}_k = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \tag{21}$$

and then move:

$$d_{kx_{k+1}} = x_k + \lambda \mathbf{d}_k \tag{22}$$

where λ is the fixed step length in **units of distance**, not scaled by slope steepness.

In our current implementation, we don't normalize — instead, we scale by the gradient magnitude directly in:

$$x_{k+1} = x_k - \lambda \nabla f(x_k) \tag{23}$$

This means that bigger slopes → bigger steps, smaller slopes → smaller steps (without needing normalization).

What the gradient actually tells

The gradient $\nabla f(x)$ **always points in the direction of *steepest increase*** of f from that point.

This is a fact from multivariable calculus:

- Moving a tiny bit in the gradient's direction will make f increase the fastest.
- Moving in the **opposite** direction $-\nabla f(x)$ will make ff decrease the fastest.

Geometric reason:
From the Taylor approximation:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^\top \Delta x \tag{24}$$

The change in ff for a small step Δx is essentially the **dot product** between the gradient and the step.

- If $Delta x$ is aligned with ∇f → dot product is **positive** → f increases.
- If Δx is aligned with $-\nabla f$ → dot product is **negative** → f decreases.

In gradient descent:

$$x_{k+1} = x_k - \lambda \nabla f(x_k) \tag{25}$$

- The “minus” sign means we go **opposite** to steepest ascent.
- λ is the **learning rate** — controlling how far we step in that direction.
- If λ is small enough, the first-order approximation above ensures we **always** go downhill locally.

Can the gradient ever "go up" instead?

Yes — if:

1. λ **is too large**:

- It might overshoot the minimum and start climbing up the other side.
- That’s why step size control (learning rate tuning) is crucial.

2. **Function isn’t convex**:

- The gradient always points to local steepest descent, but in non-convex landscapes, that might lead to a **local** minimum or saddle, not the **global** one.

Citações Importantes

- "Citação 1" (Página X) - Contexto:
- "Citação 2" (Página Y) - Contexto:

Comentários Pessoais

- **Reflexões**:
- **Críticas**:
- **Aplicações Práticas**:

Conexões

- **Relacionadas diretamente**:
  - Nota Relacionada 1
  - Nota Relacionada 2
- **Contexto mais amplo**:
  - Nota de Categoria 1
  - Nota de Categoria 2