DAMA50 - 6th assignment

Quiz 1

In [5]:
```
# 1
from sympy import symbols, diff, solve

x = symbols('x')
f = x**4 - 2*x**3 + x - 2

f_prime = diff(f, x)
f_double_prime = diff(f_prime, x)

critical_points = solve(f_prime, x)

minimum_points = [point for point in critical_points if f_double_prime.subs(x, point) > 0 and f_prime.subs(x, point) != 0]
maximum_points = [point for point in critical_points if f_double_prime.subs(x, point) < 0]
saddle_points = [point for point in critical_points if f_double_prime.subs(x, point) > 0 and f_prime.subs(x, point) == 0]

minimum_points, maximum_points, saddle_points
```

Out[5]:
```
([1/2 - sqrt(3)/2, 1/2 + sqrt(3)/2], [1/2], [])
```

Quiz 2

In [10]:
```
# 2
from sympy import symbols, Matrix
from sage.all import *

x1, x2, x3 = symbols('x1 x2 x3')

f1 = 2*x1*x2 - x1**2 - x2**2
hessian_f1 = Matrix([[f1.diff(x1, 2), f1.diff(x1, x2)], [f1.diff(x2, x1), f1.diff(x2, 2)]])
is_convex_f1 = all(eigenvalue >= 0 for eigenvalue in hessian_f1.eigenvals())

f2 = 4*x1**2 + 3*x2**2 + 5*x3**2 + 6*x1*x2 + x1*x3 - 3*x1 - 2*x2 + 15
hessian_f2 = Matrix([[f2.diff(x1, 2), f2.diff(x1, x2), f2.diff(x1, x3)],
                     [f2.diff(x2, x1), f2.diff(x2, 2), f2.diff(x2, x3)],
                     [f2.diff(x3, x1), f2.diff(x3, x2), f2.diff(x3, 2)]])
is_convex_f2 = all(eigenvalue >= 0 for eigenvalue in hessian_f2.eigenvals())

is_convex_f1, is_convex_f2
```

Out[10]:
```
(False, False)
```

Quiz 3

In [13]:
```
# 3
from sympy import Matrix

H = Matrix([[2, 1], [1, 2]])
eigenvalues = H.eigenvals()

eigenvalues
```

Out[13]:
```
{3: 1, 1: 1}
```

DAMA50 - 6th assignment

Quiz 5

In [17]:
```python
# 5
from sympy import Symbol, exp, Rational

x = Symbol('x')
f = x * exp(-x**2)
f_prime = f.diff(x)
f_double_prime = f_prime.diff(x)

x0 = -2
f_prime_x0 = f_prime.subs(x, x0)
f_double_prime_x0 = f_double_prime.subs(x, x0)

x1 = x0 - f_prime_x0 / f_double_prime_x0

x1 = Rational(x1).simplify()

x1
```

Out[17]: $-\frac{47}{20}$

Quiz 6

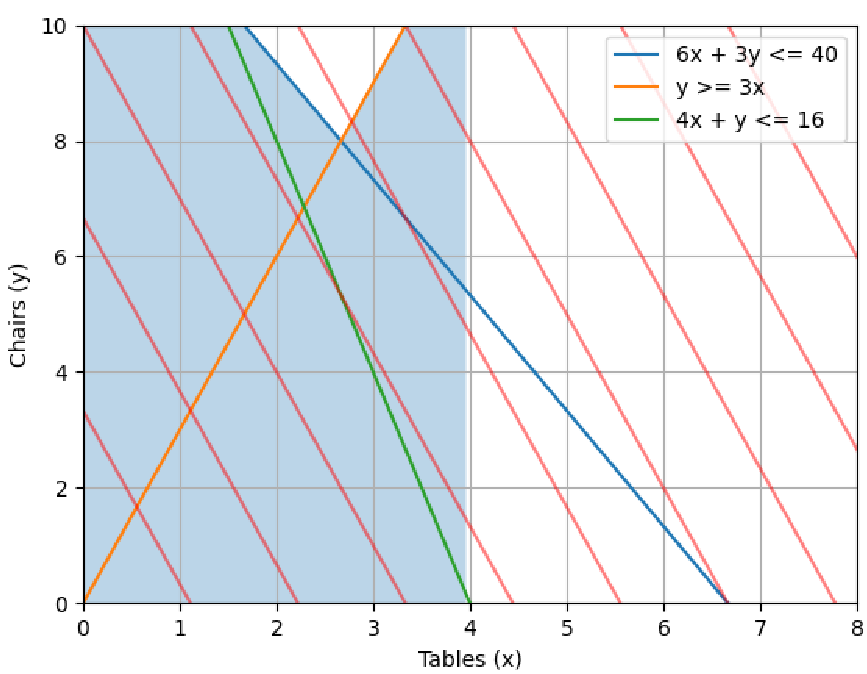
In [23]:
```python
# 6
c = [1, 1]
A = [[1, 2], [2, 1]]
b = [3, 3]
n = len(c)

direction = 1

c_dual = [-direction * b[i] for i in range(len(b))]
A_dual = [[-direction * A[i][j] for i in range(len(A))] for j in range(len(A[0]))]
b_dual = c

dual_solution = [0] * n
for i in range(n):
    if A_dual[i].count(0) == n - 1 and b_dual[i] > 0:
        dual_solution[i] = b_dual[i]

if sum(dual_solution) == 0:
    print("None of these")
else:
    objective_value = sum([c_dual[i] * dual_solution[i] for i in range(n)])
    objective_value
```

```
None of these
```

DAMA50 - 6th assignment

Problem 7

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 8, 100)
y1 = (40 - 6 * x) / 3
y2 = 3 * x
y3 = 16 - 4 * x

plt.plot(x, y1, label="6x + 3y <= 40")
plt.plot(x, y2, label="y >= 3x")
plt.plot(x, y3, label="4x + y <= 16")
plt.fill_between(x, np.maximum(np.maximum(y1, y2), y3), 0, where=(y2 >= 0) & (y3 >= 0), alpha=0.3)
P = lambda x, y: 30 * x + 10 * y
X, Y = np.meshgrid(np.linspace(0, 10, 100), np.linspace(0, 10, 100))
Z = P(X, Y)
plt.contour(X, Y, Z, levels=np.linspace(0, 300, 10), colors='red', alpha=0.5)
plt.xlabel('Tables (x)')
plt.ylabel('Chairs (y)')
plt.legend()
plt.xlim(0, 8)
plt.ylim(0, 10)
plt.grid(True)

plt.show()
```



DAMA50 - 6th assignment

Problem 8

(Used Python)

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.sin(x[0]) * np.exp((1 - np.cos(x[1]))**2) + np.cos(x[1]) * np.exp((1 - np.sin(x[0]))**2) + (x[0] - x[1])**2

def f_grad(x):
    df_dx0 = np.cos(x[0]) * np.exp((1 - np.sin(x[1]))**2) * (2 * (1 - np.sin(x[1])) * np.cos(x[0]) - (x[0] - x[1]))
    df_dx1 = -np.sin(x[1]) * np.exp((1 - np.cos(x[1]))**2) * (2 * (1 - np.cos(x[1])) * np.sin(x[0]) - (x[0] - x[1]))
    return np.array([df_dx0, df_dx1], dtype=float)
```

```python
# a
x0 = np.array([-1.0, 2.0], dtype=float)
alpha = 0.01
max_iter = 1000

for _ in range(max_iter):
    gradient = f_grad(x0)
    x0 -= alpha * gradient

res_with_deriv = x0
print("Algorithm with derivatives:")
print("Local minimum:", res_with_deriv)
x0 = np.array([-1.0, 2.0], dtype=float)
delta = 0.01
tolerance = 1e-6
max_iter = 1000

for _ in range(max_iter):
    x = np.array([x0[0] - delta, x0[0], x0[0] + delta], dtype=float)
    y = np.array([x0[1] - delta, x0[1], x0[1] + delta], dtype=float)
    values = np.array([[f([xi, yi]) for yi in y] for xi in x])
    idx = np.unravel_index(np.argmin(values, axis=None), values.shape)
    x0 = np.array([x[idx[0]], y[idx[1]]], dtype=float)
    if np.max(np.abs(values[idx])) < tolerance:
        break

res_without_deriv = x0
print("Algorithm without derivatives:")
print("Local minimum:", res_without_deriv)
```

DAMA50 - 6th assignment

```python
# b
x0 = np.array([3.0, 4.0], dtype=float)

for _ in range(max_iter):
    gradient = f_grad(x0)
    x0 -= alpha * gradient

res_with_deriv = x0
print("Algorithm with derivatives:")
print("Local minimum:", res_with_deriv)
x0 = np.array([3.0, 4.0], dtype=float)

for _ in range(max_iter):
    x = np.array([x0[0] - delta, x0[0], x0[0] + delta], dtype=float)
    y = np.array([x0[1] - delta, x0[1], x0[1] + delta], dtype=float)
    values = np.array([[f([xi, yi]) for yi in y] for xi in x])
    idx = np.unravel_index(np.argmin(values, axis=None), values.shape)
    x0 = np.array([x[idx[0]], y[idx[1]]], dtype=float)
    if np.max(np.abs(values[idx])) < tolerance:
        break

res_without_deriv = x0
print("Algorithm without derivatives:")
print("Local minimum:", res_without_deriv)
```

```python
# c
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = f([X, Y])

plt.contour(X, Y, Z, 50, cmap='viridis')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Contour Plot of f(x, y)')
plt.show()
```

DAMA50 - 6th assignment

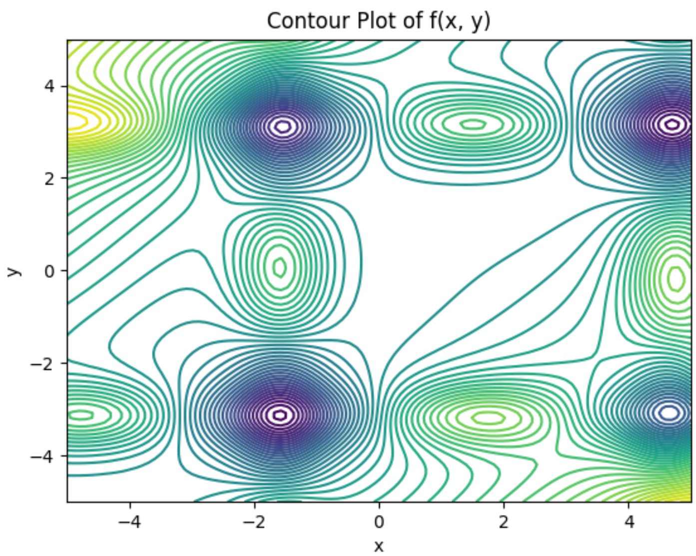
```python
# d
bounds = np.array([[-5, 0], [-5, 2]], dtype=float)
x0 = np.array([-5.0, -3.0], dtype=float)
x = np.linspace(bounds[0, 0], bounds[0, 1], 100)
y = np.linspace(bounds[1, 0], bounds[1, 1], 100)
X, Y = np.meshgrid(x, y)
Z = f([X, Y])
valid_points = np.logical_and((X + 5)**2 + (Y + 5)**2 < 25, X > Y)
X_valid = X[valid_points]
Y_valid = Y[valid_points]
Z_valid = Z[valid_points]
idx = np.unravel_index(np.argmin(Z_valid, axis=None), Z_valid.shape)
res_constrained = np.array([X_valid[idx], Y_valid[idx]], dtype=float)
print("Constrained local minimum:", res_constrained)
```

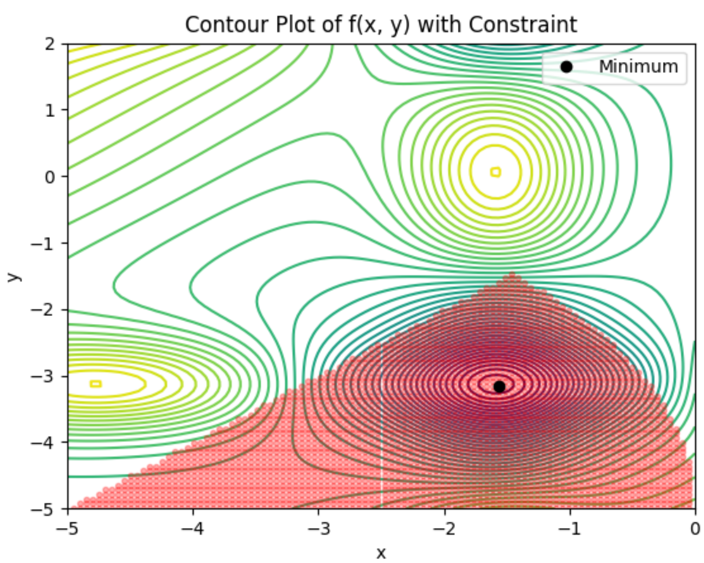
```python
# e
plt.contour(X, Y, Z, 50, cmap='viridis')
plt.plot(X_valid, Y_valid, 'r.', alpha=0.3)
plt.plot(res_constrained[0], res_constrained[1], 'ko', label='Minimum')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Contour Plot of f(x, y) with Constraint')
plt.legend()
plt.show()
```

DAMA50 - 6th assignment

Results:

```
Algorithm with derivatives:
Local minimum: [-1.57079633  3.14159265]
Algorithm without derivatives:
Local minimum: [-1.54  3.11]
Algorithm with derivatives:
Local minimum: [2.11087609 3.39335103]
Algorithm without derivatives:
Local minimum: [4.7  3.15]
Constrained local minimum: [-1.56565657 -3.16161616]
```





DAMA50 - 6th assignment

Problem 9

(Used Python)

```python
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, diff, solve, hessian

x, y = symbols('x y')
f = (x + y + 3*x*y) / (1 + x**2 + y**2)
```

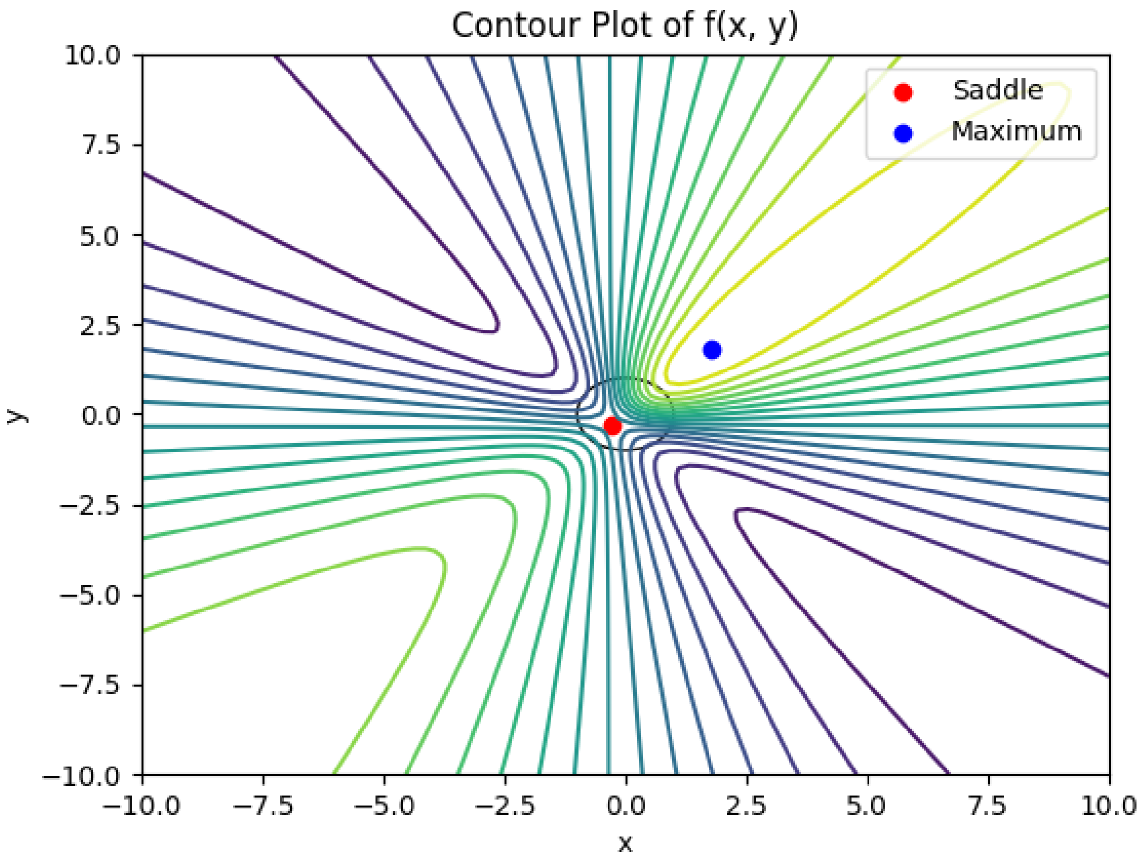
```python
# a
df_dx = diff(f, x)
df_dy = diff(f, y)
stationary_points = solve([df_dx, df_dy], x, y)
```

```python
# b
hessian_matrix = hessian(f, (x, y))
stationary_points_data = []
for point in stationary_points:
    hessian_subs = np.array([[float(hessian_matrix[i, j].subs([(x, point[0]), (y, point[1])])) for j in range(hessian_matrix.shape[1])] for i in range(hessian_matrix.shape[0])])
    eigenvals = np.linalg.eigvals(hessian_subs)
    point_type = 'Minimum' if np.all(eigenvals > 0) else 'Maximum' if np.all(eigenvals < 0) else 'Saddle'
    value = float(f.subs([(x, point[0]), (y, point[1])]))
    stationary_points_data.append((point, point_type, value))
```

```python
# c
f_func = lambda x, y: (x + y + 3*x*y) / (1 + x**2 + y**2)
x_vals = np.linspace(-10, 10, 100)
y_vals = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = f_func(X, Y)

plt.contour(X, Y, Z, levels=20, cmap='viridis')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Contour Plot of f(x, y)')

for point, point_type, value in stationary_points_data:
    color = 'blue' if point_type != 'Saddle' else 'red'
    plt.scatter(point[0], point[1], color=color, label=point_type)

constraint = plt.Circle((0, 0), 1, color='black', fill=False)
plt.gca().add_patch(constraint)

plt.legend()
plt.show()
```

DAMA50 - 6th assignment

Results:



DAMA50 - 6th assignment

Problem 10

10) $F(x,y) = \frac{2x^3}{3} - 2xy - 5x + 2y^2 + 4y + 5$

$$\frac{dF}{dx} = \frac{d[(2x^3/3) - 2xy - 5x + 2y^2 + 4y + 5]}{dx}$$

$$\frac{dF}{dx} = 2x^2 - 2y - 5 = 0 \quad (1)$$

$$\frac{dF}{dy} = \frac{d[(2x^3/3) - 2xy - 5x - 2y^2 + 4y + 5]}{dy}$$

$$\frac{dF}{dy} = -2x + 4y + 4 = 0 \quad (2)$$

$$(2) = -2x = -4y - 4 \Rightarrow x = 2y + 2 \quad (3)$$

$$(1) \overset{(3)}{\Rightarrow} 2(2y+2)^2 - 2y - 5 = 0 \Rightarrow 8y^2 + 16y + 8 - 2y - 5 = 0 \Rightarrow$$

$$\Rightarrow 8y^2 + 14y + 3 = 0 \qquad y = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$= \frac{-14 \pm \sqrt{14^2 - 4 \cdot 8 \cdot 3}}{2 \cdot 8}$$

$$= \frac{-14 \pm \sqrt{100}}{16}$$

$$= \frac{-14 \pm 10}{16}$$

$$y_1 = -\frac{1}{4} \qquad y_2 = -\frac{3}{2}$$

DAMA50 - 6th assignment

(3) • Για $y = -\frac{1}{4}$

$\rightarrow x = 2(-1/4) + 2 = 3/2$

• Για $y = -\frac{3}{2}$

$\rightarrow x = 2(-3/2) + 2 = -1$

Critical points: $(3/2, -1/4)$ & $(-1, -3/2)$

$$\frac{\partial^2 F}{\partial x^2} = \frac{\partial[2x^2 - 2y - 5]}{\partial x} = 4x$$

$$\frac{\partial^2 F}{\partial y^2} = \frac{\partial[-2x + 4y + 4]}{\partial y} = 4$$

$$\frac{\partial^2 F}{\partial x \partial y} = \frac{\partial[-2x + 4y + 4]}{\partial x} = -2$$

$$H = \frac{\partial^2 F}{\partial x^2} \cdot \frac{\partial^2 F}{\partial y^2} - \left(\frac{\partial^2 F}{\partial x \partial y}\right)^2 = 4x \cdot 4 - (-2)^2 = 16x - 4$$

• For $(3/2, -1/4)$: $H = 16(3/2) - 4 = 20$

$H > 0$ & $\frac{\partial^2 F}{\partial x^2} = 4x > 0 \Rightarrow$ local minimum

• For $(-1, -3/2)$

$H = 16 \cdot (-1) - 4 = -20$

$H < 0 \Rightarrow$ saddle point