סעיף 7:

קוד:

```python
import numpy as np

a = np.sqrt(2)


def _stop_condition(current_delta):
    global a
    return current_delta <= (1e-50) / a


def _next_delta_1(current_delta):
    return (current_delta ** 2) / (2 * (1 + current_delta))


def _next_delta_2(current_delta):
    return current_delta / 2


def _next_delta_3(current_delta):
    return (current_delta ** 2) / 2


def _evaluations(update_function, delta_0):
    k = 0
    current_delta = delta_0
    while not _stop_condition(current_delta):
        k += 1
        current_delta = update_function(current_delta)
    return k


if __name__ == '__main__':
    delta_0 = a - 1
    print("1st expression reached bound in {} iterations".format(_evaluations(_next_delta_1, delta_0)))
    print("2nd expression reached bound in {} iterations".format(_evaluations(_next_delta_2, delta_0)))
    print("3rd expression reached bound in {} iterations".format(_evaluations(_next_delta_3, delta_0)))
```

פלט, מספר איטרציות לכל חסם:

```
1st expression reached bound in 7 iterations
2nd expression reached bound in 166 iterations
3rd expression reached bound in 7 iterations
```

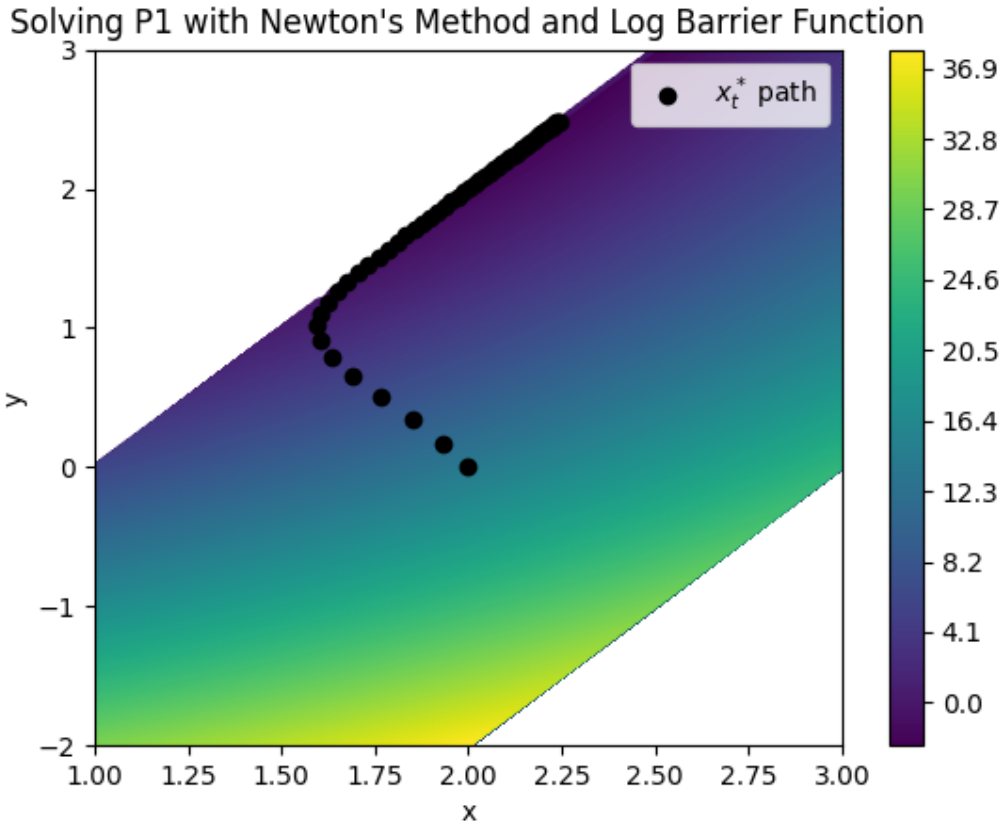
שאלה 5 סעיפים 2,3:

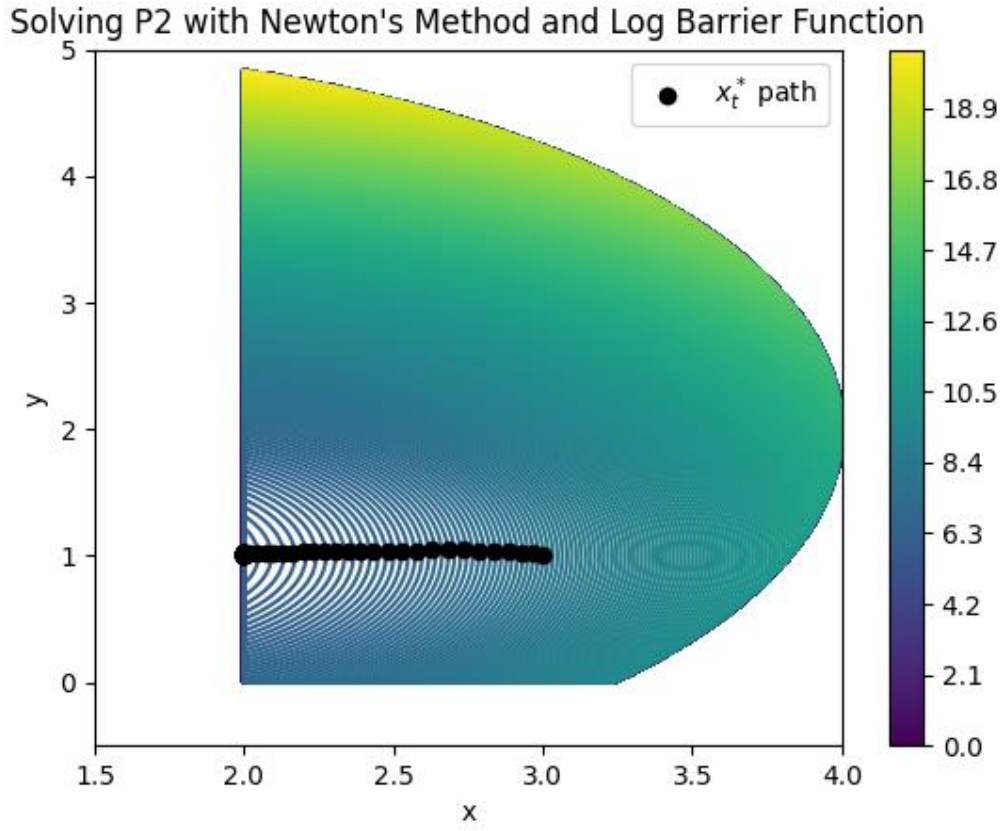
קוד:

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


class MinimizationProblem:
    def __init__(self, name: str, number_constraints):
        self.name: str = name
        self.number_constraints: int = number_constraints
        self.target_func = None
        self.dLdx = self.dLdy = self.ddLdxdy = self.ddLddx = self.ddLddy = None
        self.constrains = None
        self.starts_vector = None

    def gradient(self, x, y, t):
        return np.array([self.dLdx(x, y, t), self.dLdy(x, y, t)])

    def hessian(self, x, y, t):
        hess = np.array([[self.ddLddx(x, y, t), self.ddLdxdy(x, y, t)], [self.ddLdxdy(x, y, t), self.ddLddy(x, y, t)]])
        assert hess.shape == (2, 2)
        return hess


def solve_inner(current_x_vec, t, p:MinimizationProblem, num_iter=5):
    x_vec = current_x_vec
    for i in range(num_iter):
        x_vec = x_vec - 0.01 * np.linalg.inv(p.hessian(x_vec[0], x_vec[1], t)) @ p.gradient(x_vec[0], x_vec[1], t)
    return x_vec


def solve_outer(mu, t_0, epsilon, p:MinimizationProblem):
    x_vecs = [p.starts_vector]
    t = t_0
    while p.number_constraints / t > epsilon:
        next_vec = solve_inner(x_vecs[-1], t, p)
        x_vecs.append(next_vec)
        t *= mu
    return x_vecs
```

```python
def init_p1():
    p_1_problem = MinimizationProblem("P1", 2)
    p_1_problem.constrains = lambda x, y: abs(x - 0.5 * y - 2) <= 1
    p_1_problem.target_func = lambda x, y: (x ** 2) + (y ** 2) + 5 * x - 10 * y
    p_1_problem.dLdx = lambda x, y, t: t * (2 * x + 5) - 1 / (x - 0.5 * y - 3) - 1 / (x - 0.5 * y - 1)
    p_1_problem.ddLddx = lambda x, y, t: 2 * t + 1 / ((x - 0.5 * y - 3) ** 2) + 1 / ((x - 0.5 * y - 1) ** 2)
    p_1_problem.dLdy = lambda x, y, t: t * (2 * y - 10) + 0.5 / (x - 0.5 * y - 3) + 0.5 / (x - 0.5 * y - 1)
    p_1_problem.ddLddy = lambda x, y, t: 2 * t + 0.25 / ((x - 0.5 * y - 3) ** 2) + 0.25 / ((x - 0.5 * y - 1) ** 2)
    p_1_problem.ddLdxdy = lambda x, y, t: -0.5 / ((x - 0.5 * y - 3) ** 2) - 0.5 / ((x - 0.5 * y - 1) ** 2)
    p_1_problem.starts_vector = np.array([2, 0])
    return p_1_problem


def init_p2():
    p_2_problem = MinimizationProblem("P2", 3)
    p_2_problem.constrains = lambda x, y: ((x - 1) ** 2 + (y - 2) ** 2 <= 9) * (x >= 2) * (y >= 0)
    p_2_problem.target_func = lambda x, y: ((x - 3) ** 2) + (y ** 2) + 3 * x - 2 * y
    p_2_problem.dLdx = lambda x, y, t: t * (2 * x - 3) + 2 * (x - 1) / (-((x - 1) ** 2) - ((y - 2) ** 2) + 9) - 1 / (x - 2)
    p_2_problem.dLdy = lambda x, y, t: t * (2 * y - 2) + 2 * (y - 2) / (-((x - 1) ** 2) - ((y - 2) ** 2) + 9) - 1 / y
    p_2_problem.ddLdxdy = lambda x, y, t: 4 * (x - 1) * (y - 2) / ((-((x - 1) ** 2) - ((y - 2) ** 2) + 9) ** 2)
    p_2_problem.ddLddx = lambda x, y, t: 2 * t + 2 / (-((x - 1) ** 2) - ((y - 2) ** 2) + 9) + 4 * ((x - 1) ** 2) / (
            (-((x - 1) ** 2) - ((y - 2) ** 2) + 9) ** 2) \
                                                    + 1 / ((x - 2) ** 2)
    p_2_problem.ddLddy = lambda x, y, t: 2 * t + 2 / (-((x - 1) ** 2) - ((y - 2) ** 2) + 9) + 4 * ((y - 2) ** 2) / (
            (-((x - 1) ** 2) - ((y - 2) ** 2) + 9) ** 2) \
                                                    + 1 / (y ** 2)
    p_2_problem.starts_vector = np.array([3, 1])

    return p_2_problem


if __name__ == '__main__':
    p_1 = init_p1()
    p_2 = init_p2()

    mu = 1.5
    t0 = 1
    epsilon = 1e-10
    n = 500
    x_values_p1 = solve_outer(mu, t0, epsilon, p_1)
    x_values_p2 = solve_outer(mu, t0, epsilon, p_2)
    plot_cont(p_1, xlims=[1, 3], ylims=[-2, 3], x_values=x_values_p1, fig_path="p1.png")
    plot_cont(p_2, xlims=[1.5, 4], ylims=[-0.5, 5], x_values=x_values_p2, fig_path="p2.png")
```

פלט של הגרפים:


Solving P1 with Newton's Method and Log Barrier Function
x_t^* path
x
y
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3
2
1
0
−1
−2
36.9
32.8
28.7
24.6
20.5
16.4
12.3
8.2
4.1
0.0


Solving P2 with Newton's Method and Log Barrier Function
x_t^* path
x
y
1.5
2.0
2.5
3.0
3.5
4.0
5
4
3
2
1
0
18.9
16.8
14.7
12.6
10.5
8.4
6.3
4.2
2.1
0.0