תרגיל בית מס 6

**מגישים:**

**כפיר עשור, 205913270**

**אדוה שולץ, 305497877**

***שאלה 1:***

***שאלה 2:***

1. *נרשום פונקציה המממשת את השיטה ההיברידית כפי שנלמדה:*

import numpy as np  
import numpy.linalg as la  
import time  
  
  
def hybrid\_newton(f, gf, hf, lsearch, x0, eps):  
 T = time.time()  
 ts = np.array(time.time() - T)  
 fs = np.array([f(\*x0) + eps + 1, f(\*x0)])  
 gs = np.array([la.norm(gf(\*x0))])  
 newton = []  
 next\_x = np.array(x0)  
 while abs(fs[-1] - fs[-2]) >= eps:  
 xk = next\_x  
 gk = np.array(gf(\*xk)).T[0]  
 hk = np.array(hf(\*xk)).T[0]  
 try:  
 la.cholesky(hk)  
 newton.append(1)  
 dk = -la.inv(hk) \* gk  
 direction = [dk, 'newton']  
 except:  
 newton.append(0)  
 dk = -gk  
 direction = [dk, 'grad']  
 step\_size = lsearch(xk, gk, direction)  
 next\_x = xk - step\_size \* gk  
 fs = np.append(fs, f(\*next\_x))  
 gs = np.append(gs, la.norm(gf(\*next\_x)))  
 ts = np.append(ts, time.time() - T)  
 fs = fs[1:]  
 return next\_x, fs, gs, ts, newton

1. *נבנה פונקציה המממשת חיפוש עקיבה לאחור עבור איטרציית גרדיאנט ומחזירה גודל צעד 1 עבור שיטת ניוטון (שיטה טהורה):*

def hybrid\_back(f, alpha, beta, s):  
 def search(x1, x2, x3):  
 if x3[1] == 'newton':  
 return 1  
  
 t = s  
 while (f(\*x1) - f(\*(x1 - t \* x2)))[0][0] < alpha \* t \* pow(la.norm(x2), 2):  
 t = beta \* t  
 return t  
  
 return lambda x1, x2, x3: search(x1, x2, x3)

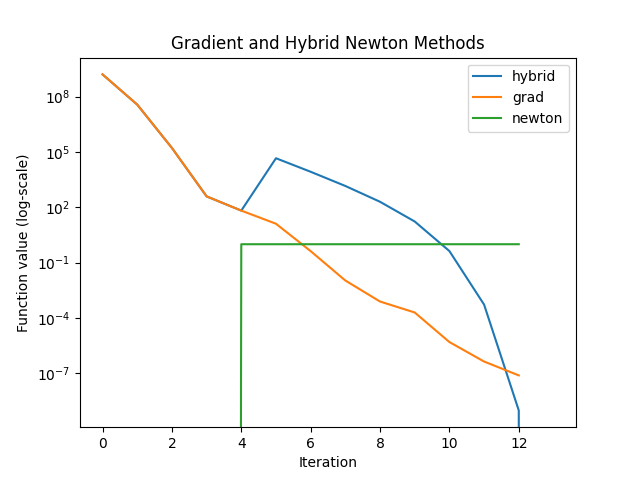
1. *נפעיל את השיטה כפי שנתבקשנו בתרגיל.*

alpha, beta, s = 1/4, 1/2, 1  
f = lambda x, y: pow(x, 4) + pow(y, 4) - 36 \* x \* y  
x0 = np.array([200, 0])  
eps = 1e-6  
gf = lambda x, y: np.array([4 \* pow(x, 3) - 36 \* y, 4 \* pow(y, 3) - 36 \* x])  
hf = lambda x, y: np.array([[12 \* x\*\*2, -36], [-36, 12 \* y\*\*2]])  
  
x\_grad, fs\_grad, gs\_grad, ts\_grad = generic\_grad(f, gf, back(alpha, beta, s), x0, eps)  
x\_hybrid, fs\_hybrid, gs\_hybrid, ts\_hybrid, newton =\  
 hybrid\_newton(f, gf, hf, hybrid\_back(f, alpha, beta, s), x0, eps)  
  
plt.semilogy(fs\_hybrid + 162, label='hybrid')  
plt.semilogy(fs\_grad + 162, label='grad')  
plt.semilogy(newton, label='newton')  
plt.legend()  
plt.title('Gradient and Hybrid Newton Methods')  
plt.xlabel('Iteration')  
plt.ylabel('Function value (log-scale)')  
plt.show()

*כאשר האלגוריתם של שיטת הגרדיאנט עם חיפוש קווי עוקב בוצע בעזרת תרגיל בית 4 עם מעט התאמות, ולכן השתמשנו בקוד הבא:*

def generic\_grad(f, gf, lsearch, x0, eps):  
 T = time.time()  
 ts = np.array(time.time() - T, dtype=float)  
 fs = np.array([f(\*x0) + eps + 1, f(\*x0)], dtype=float)  
 gs = np.array([la.norm(gf(\*x0))], dtype=float)  
 next\_x = np.array(x0, dtype=float)  
 while abs(fs[-1] - fs[-2]) >= eps:  
 xk = next\_x  
 gk = gf(\*xk)  
 step\_size = lsearch(f, xk, gk)  
 next\_x = xk - step\_size \* gk  
 fs = np.append(fs, f(\*next\_x))  
 gs = np.append(gs, la.norm(gf(\*next\_x)))  
 ts = np.append(ts, time.time() - T)  
 fs = fs[1:]  
 return next\_x, fs, gs, ts  
  
  
def back(alpha, beta, s):  
 if s <= 0:  
 print('The step size should be positive!')  
 exit()  
 if alpha <= 0 or alpha >= 1:  
 print('The parameter alpha should be positive and less then one!')  
 exit()  
 if beta <= 0 or beta >= 1:  
 print('The parameter beta should be positive and less then one!')  
 exit()  
  
 def search(x1, x2, x3):  
 t = s  
 while (x1(\*x2) - x1(\*(x2 - t \* x3))) < alpha \* t \* pow(la.norm(x3), 2):  
 t = beta \* t  
 return t  
  
 return lambda x1, x2, x3: search(x1, x2, x3)

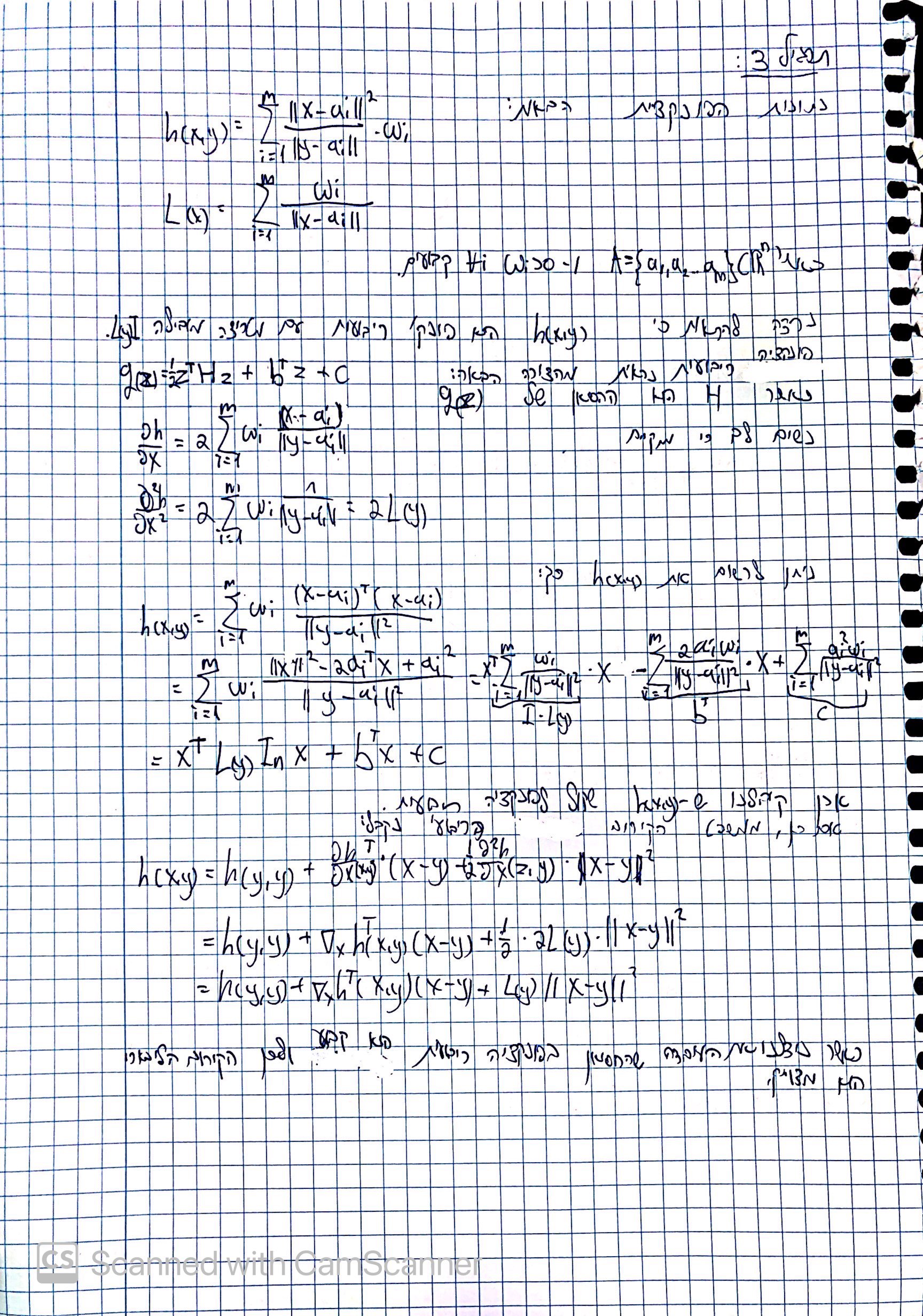
*נקבל את התוצאה הבאה:*

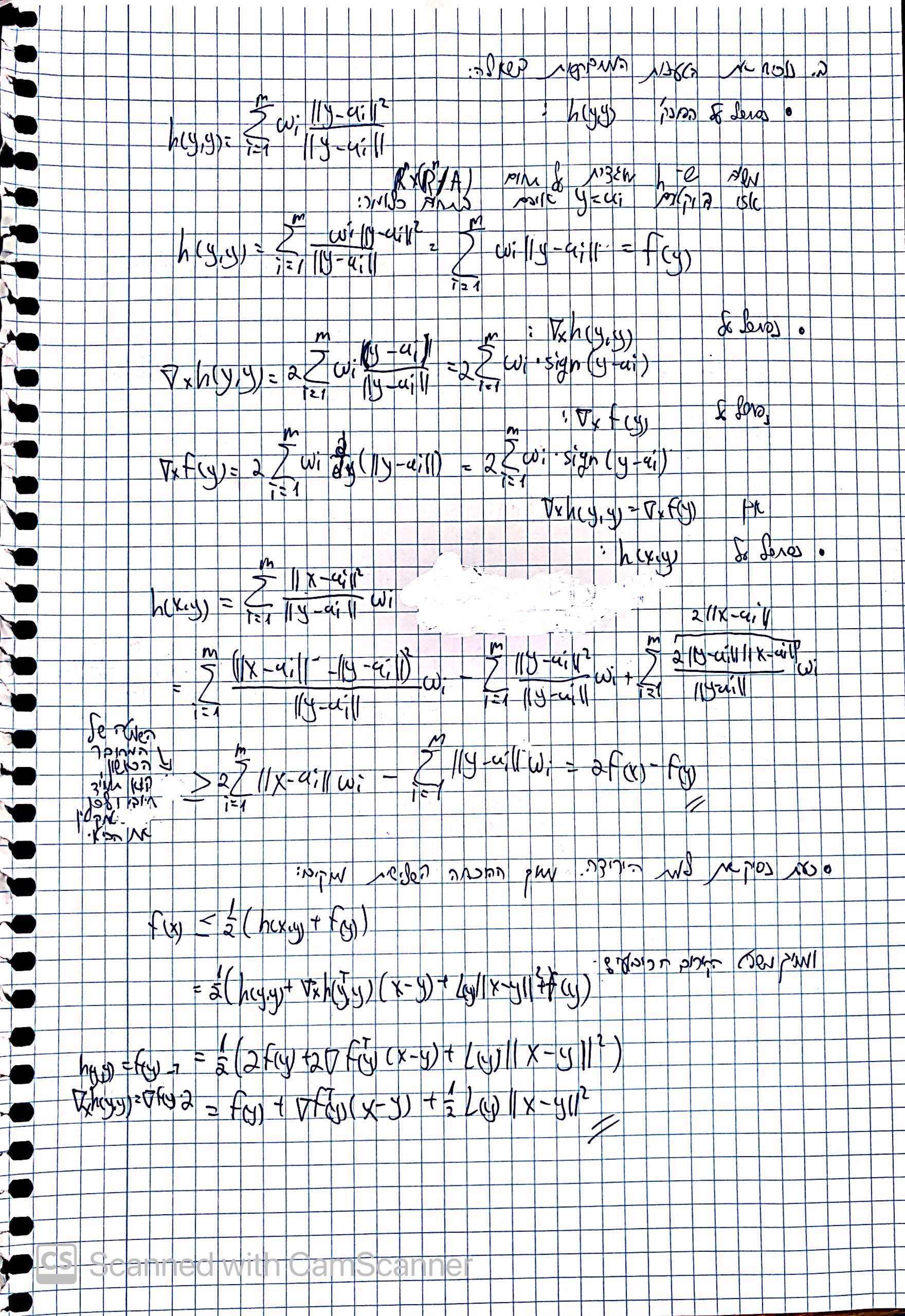
**

*על פי הגרף שהתקבל, ניתן לראות שבאיטרציה הראשונה של שיטת ניוטון הייתה עלייה בערך הפונקציה. לכן אלגוריתם זה אינו אלגוריתם ירידה, שכן הוא לא מבטיח ירידה של ערך הפונקציה בכל איטרציה, כפי שניתן לראות בגרף.*

*ניתן לבצע התאמה כך שבמידה וערך הפונקציה יורד בעזרת שיטת ניוטון אז נשתמש בה ואחרת נשתמש בשיטת הגרדיאנט.*

***שאלה 3****:*

******

**