**עבודה 3**

**שאלה 1**

1. התחום שלנו הוא וזהו תחום קמור. נשים לב כי פונקציה זו גזירה אינסוף פעמים.

נתבונן בנגזרת השנייה של הפונקציה:

קיבלנו כי לכל הפונקציה תהיה קמורה, ועבור נקבל כי פונקציה קבועה ולא קמורה.

ניקח את בסיס הלוג להיות e (ערן אמר לקחת אותו)תחום הגדרה של f הוא זהו תחום קמור, שכן לכל שתי נק' x,y בתחום מתקיים

מאחר והתחום שלנו מוגדר להיות , אזי גזירה לכל .   
סה"כ קיבלנו כי הפונקציה קמורה בכל תחום הגדרתה.

\*\* במידה ובסיס הלוג היה קטן מ-1, הפונקציה הייתה קעורה (כי ln של מספר בין 0 ל1 הוא שלילי), וכן לכל מספר גדול מ1 הפונקציה קמורה. אם הבסיס היה 1 הפונקציה לא מוגדרת.

1. ניקח את בסיס הלוג להיות e (ערן אמר לקחת אותו)

תחום הגדרה של f הוא זהו תחום קמור, שכן לכל שתי נק' x,y בתחום מתקיים

מאחר והתחום שלנו מוגדר להיות , אזי גזירה לכל .   
סה"כ קיבלנו כי הפונקציה קעורה בכל תחום הגדרתה.

במידה ובסיס הלוג היה קטן מ-1, הפונקציה הייתה קמורה (כי ln של מספר בין 0 ל1 הוא שלילי), וכן לכל מספר גדול מ1 הפונקציה קעורה. אם הבסיס היה 1 הפונקציה לא מוגדרת.

תחום ההגדרה של הפונקציה הינו וכן זהו תחום קמור.

*נוכיח עבור באמצעות ההגדרה האלטרנטיבית הראשונה כי f פונקציה קמורה.*

*תחילה נראה כי היא גזירה: קל לראות כי הנקודה הבעיייתית היחידה היא ב וכן לכל היא בהכרח גזירה לפי כללי גזירה ידועים. נראה כי f גזירה בנק' לפי הגדרה:*

*קיבלנו כי ולכן f גזירה בנק .*

*לכן הנגזרת של f הינה:   
כעת נוכיח כי לכל מתקיים : כלומר נרצה להוכיח כי  
 . נחקור את פונקציה*

* *אם : וכן זה יהיה שלילי כאשר x<y. כמו כן נשים לב כי הפונקציה מתאפסת בנק' וכן כאשר x>y הפונקציה עולה. קיבלנו כי יש לנו נק' מינימום כאשר ולכן לכל x.*
* *אם :  
   וכן זה יהיה שלילי כאשר . כמו כן נשים לב כי הפונקציה מתאפסת בנק' וכן כאשר x>y הפונקציה עולה. קיבלנו כי יש לנו נק' מינימום כאשר ולכן לכל x.*
* *אם : כלומר במקרה זה הפונקציה תמיד יורדת. בנק הפונקציה מתאפסת, ומאחר ובמקרה זה נקבל כי .*
* *אם : כלומר במקרה זה הפונקציה תמיד עולה. בנק' הפונקציה מתאפסת ומאחר ובמקרה זה נקבל כי .*

*במקרה ש-: נוכיח לפי ההגדרה של קמירות:  
יהיו .*

1. *.*

*פונקציה זו לא קמורה ולא קעורה.*

*עבור הנק' נקבל כי עבור נקבל כי .*

1. *.נשים לב כי התחום של הפונקציה הינו וזהו תחום קמור כפי שנלמד בכיתה.  
   כמו כן, הפונקציה גזירה פעמיים:*

*לפי המשפט השני בהגדרה האלטרניטיבית, f תהיה קמורה אם יתקיים כי וזה יתקיים אמ"מ A SPD.*

1. *: נניח כי f קמורה. יהיו . f קמורה ולכן מתקיים כי לכל מתקיים*

*נשים לב, כאשר אזי גם ואז נקבל מטור טיילור כי   
. נציב במשוואה הראשונה ונקבל כי   
נחסר משני האגפים ונקבל מאחר ן נוכל לחלק ב- מבלי לשנות את הכיוון של אי השוויון ונקבל: כלומר   
 כנדרש.*

*: יהיו וכן*

*מאי השוויון הנתון נקבל כי : , .*

*נכפול את אי השוויון הראשון ב ואת אי השוויון השני ב, נחבר ביניהם, ונקבל:*

*כעת נציב את t ונקבל כנדרש.*

**שאלה 2**

1. ניסינו עבור עם

lam = 100

GTG = np.transpose(G) @ G

M = np.eye(n) + lam/2 \* GTG

x\_min = LA.inv(M) @ y

plt.figure(); plt.plot(x,x\_min, label = r'recover using ${{\scrl}\_2 } norm,\lambda=100$'); plt.plot(x,f, label = "clean"); plt.legend() ;plt.show()

lam = 80

GTG = np.transpose(G) @ G

M = np.eye(n) + lam/2 \* GTG

x\_min = LA.inv(M) @ y

plt.figure(); plt.plot(x,x\_min, label = r'recover using ${{\scrl}\_2 } norm,\lambda=80$'); plt.plot(x,f, label = "clean"); plt.legend() ;plt.show()

*הגרפים שקיבלנו הם:*

Chart, line chart

Description automatically generated

1. **Chart, line chart

   Description automatically generated*הקוד:***

def IRLS( y, w, G , epsilon, lamda, maxIter):

for i in range(maxIter):

x = LA.inv(np.eye(n) + lamda/2 \* (np.transpose(G) @ w @ G)) @ y

for j in range (len(w)):

w[j][j] = 1/ (abs(G[j] @ x)+ epsilon)

return x

x\_IRLS = IRLS( y, np.eye(n-1), G, 0.001, 1, 10)

*plt.figure(); plt.plot(x,x\_IRLS, label = "recover using IRLS"); plt.plot(x,f, label = "clean"); plt.legend() ;plt.show()*

Chart, line chart

Description automatically generated

**שאלה 3**

1. נגדיר
2. *נגדיר*

*נרצה לגזור ולהשוות לאפס. נשים לב כי נגזור כל חלק בנפרד, ולכן נוכל להשתמש בביטוי שקיבלנו בסעיף הקודם עבור החלק השמאלי של המשוואה:*

1. *קיבלנו כי*

*אנו יודעים כי היא SPD, והוספת מטריצה אלכסונית, משאירה אותה SPD. על כן, ההופכית שלה גם SPD. לכן נקבל כי d הוא כיוון מנוגד לכיוון הירידה, ולמדנו שזהו כיוון ירידה.*

**שאלה 4**

def LGobj(X, y ,w):  
 c1, c2 = generate\_c(y)  
 XTw = np.transpose(X) @ w  
 return (- 1/ len(w)) \* ((c1 @ vec\_log(vec\_sigmoid(XTw))) + (c2 @ vec\_log(comp\_sigmoid(XTw))))  
  
def vec\_sigmoid(v):  
 sig\_vec = []  
 for i in range(len(v)):  
 sig\_vec.append(1 / (1 + np.exp(-v[i])))  
 return np.asarray(sig\_vec)  
  
def comp\_sigmoid(v):  
 sig\_vec = []  
 for i in range(len(v)):  
 sig\_vec.append(1- (1 / (1 + np.exp(-v[i]))))  
 return np.asarray(sig\_vec)  
  
def vec\_log(v):  
 log\_vec =[]  
 for i in range(len(v)):  
 if v[i] > 0:  
 log\_vec.append(math.log(v[i]))  
 else:  
 log\_vec.append(v[i])  
 return np.asarray(log\_vec)  
  
def gradientLogistic(X,y,w):  
 c1, c2 = generate\_c(y)  
 XTw = np.transpose(X) @ w  
 sig\_XTw = vec\_sigmoid(XTw)  
 return 1/len(w) \* (X @ (sig\_XTw - c1))  
  
def hessianLogistic(X, w):  
 XTw = np.transpose(X) @ w  
 D = vec\_sigmoid(XTw)\*comp\_sigmoid(XTw)  
 D = np.diag(D)  
 return 1 / len(w) \* (X @ D @ np.transpose(X))  
  
def generate\_c(y):  
 c1\_label = y[0]  
 c1=[1]  
 for i in range(1,len(y)):  
 if y[i] == c1\_label :  
 c1.append(1)  
 else:  
 c1.append(0)  
  
 c2 = []  
 for i in range(len(y)):  
 if c1[i] == 1:  
 c2.append(0)  
 else:  
 c2.append(1)  
  
 return np.asarray(c1), np.asarray(c2)

def LG(X, y, w):  
 f\_w = LGobj(X, y, w)  
 gradient = gradientLogistic(X, y, w)  
 hessian = hessianLogistic(X, w)  
 return f\_w, gradient, hessian

X = np.random.rand(10,10)  
y = np.random.choice([0, 1], size=10)  
w = np.random.rand(10)  
d = np.random.rand(10)  
  
  
def test\_grad(X,y,w , epsilon, d, maxIter):  
  
 loss\_with\_grad = []  
 loss = []  
 epsilons = []  
 DF = []  
 DF\_grad = []  
 for i in range(maxIter):  
 epsilons.append(epsilon)  
 f\_eps, grad\_eps, hess\_eps = LG(X, y, w + epsilon \* d)  
 f, grad, hess = LG(X, y, w)  
 loss.append(abs(f\_eps-f))  
 loss\_with\_grad.append(abs(f\_eps-f-(epsilon \* d @ grad)))  
 epsilon = 0.5 \* epsilon  
 if len(loss) >1:  
 DF.append(loss[i-1] / loss[i])  
 DF\_grad.append((loss\_with\_grad[i-1]/loss\_with\_grad[i]))  
  
 return loss, loss\_with\_grad, epsilons, DF, DF\_grad  
  
def test\_hess(X,y,w , epsilon, d, maxIter):  
 loss\_with\_hess = []  
 loss = []  
 epsilons = []  
 DF =[]  
 DF\_hess = []  
 for i in range(maxIter):  
 epsilons.append(epsilon)  
 f\_eps, grad\_eps, hess\_eps = LG(X, y, w + epsilon \* d)  
 f, grad, hess = LG(X, y, w)  
 loss.append(LA.norm(grad\_eps-grad))  
 loss\_with\_hess.append(LA.norm(grad\_eps-grad-(hess @ (epsilon \* d))))  
 epsilon = 0.5 \* epsilon  
 if len(loss) >1:  
 DF.append(loss[i-1] / loss[i])  
 DF\_hess.append((loss\_with\_hess[i-1]/loss\_with\_hess[i]))  
  
 return loss, loss\_with\_hess, epsilons, DF, DF\_hess  
  
  
  
loss , loss\_grad, epsilons, DF, DF\_grad = test\_grad(X, y, w, 0.1, d, 20)  
iterations = np.arange(19)  
iterations2 = np.arange(20)  
loss2 , loss\_hess, epsilons, DF2, DF\_hess = test\_hess(X, y, w, 0.1, d, 20)  
  
plt.figure(); plt.semilogy(iterations2,loss, label = "gradient test : loss"); plt.semilogy(iterations2,loss\_grad, label = "loss\_grad");  
  
  
plt.semilogy(iterations2,loss2, label = "hessian test : loss"); plt.semilogy(iterations2,loss\_hess, label = "loss\_hess"); plt.legend() ;plt.show()  
plt.figure(); plt.plot(iterations,DF, label = "gradient test : loss"); plt.plot(iterations,DF\_grad, label = "loss\_grad");  
  
plt.plot(iterations,DF2, label = "hessian test : loss"); plt.plot(iterations,DF\_hess, label = "loss\_hess"); plt.legend() ;plt.show()

Chart, line chart

Description automatically generated

Chart

Description automatically generated

def Armijio(X,y,x, gradx, d, a, b, c, maxiter):  
 for i in range(maxiter):  
 obj\_a = LGobj(X, y, x + a\*d)  
 if obj\_a < (LGobj(X, y, x) + c\*a\*np.inner(gradx,d)):  
 return a  
 else:  
 a = b\*a  
 return a  
  
def gradient\_descent(X, y, w,maxIter, a0, beta, c, epsilon):  
 f\_values = [LGobj(X, y, w)]  
 for i in range(maxIter):  
 grad = gradientLogistic(X,y,w)  
 d = -grad  
 alpha = Armijio(X,y,w, grad, d, a0,beta, c, 100)

wPrev = w  
 w = w + alpha \* d  
 w = np.clip(w,-1,1)  
 f\_values.append(LGobj(X,y,w))

if LA.norm(w- wPrev) / LA.norm(w) < epsilon:  
 break  
 return w, f\_values  
  
def exactNewton(X, y, w,maxIter, a0, beta, c, epsilon):  
 f\_values = [LGobj(X, y, w)]  
 for i in range(maxIter):  
 grad = gradientLogistic(X,y,w)  
 hess = hessianLogistic(X,w) + np.eye(len(w))\* 0.001 d = -1 \* LA.inv(hess) @ grad  
 alpha = Armijio(X,y,w, grad, d, a0,beta, c, 100)

wPrev = w  
 w = w + alpha \* d

w = np.clip(w,-1,1)  
 f\_values.append(LGobj(X,y,w))  
 if LA.norm(w- wPrev) / LA.norm(w) < epsilon:  
 break  
 return w, f\_values  
  
def filterTwoDigits(num1, num2, x, y):  
 x\_res=[]  
 y\_res = []  
 for i in range(len(x)):  
 if y[i] == num1 or y[i] == num2:  
 x\_res.append(x[i])  
 y\_res.append(y[i])  
 return x\_res ,y\_res  
  
def imageToVector(data):  
 xNew = copy.deepcopy(data)  
 for i in range(len(xNew)):  
 tmpMatrix = xNew[i]  
 tmp = np.ndarray.tolist(tmpMatrix[0])  
 for j in range(1, len(tmpMatrix)):  
 list = np.ndarray.tolist(tmpMatrix[j])  
 tmp.extend(list)  
 xNew[i] = np.asarray(tmp)  
 return xNew  
  
  
def calculateError(w, w\_k):  
 ans = []  
 for i in range(len(w\_k)):  
 ans.append(abs(w\_k[i]-w))  
 return ans;  
  
x\_train = [x\_train[i] for i in range(30000)]  
  
def runAndPlotGD(digit1, digit2):  
 zero\_one\_images\_train, zero\_one\_labels\_train = filterTwoDigits(digit1, digit2, x\_train, y\_train)  
 zero\_one\_images\_train = imageToVector(zero\_one\_images\_train)  
 zero\_one\_images\_test, zero\_one\_labels\_test = filterTwoDigits(digit1, digit2, x\_test, y\_test)  
 zero\_one\_images\_test = imageToVector(zero\_one\_images\_test)  
  
 for i in range(len(zero\_one\_images\_train)):  
 zero\_one\_images\_train[i] = (zero\_one\_images\_train[i]) / 255

for i in range(len(zero\_one\_images\_test)):  
 zero\_one\_images\_test[i] = (zero\_one\_images\_test[i]) / 255  
 zero\_one\_images\_train = np.transpose(np.asarray(zero\_one\_images\_train))  
 zero\_one\_labels\_train = np.asarray(zero\_one\_labels\_train)  
 zero\_one\_images\_test = np.transpose(np.asarray(zero\_one\_images\_test))  
 zero\_one\_labels\_test = np.asarray(zero\_one\_labels\_test)  
  
 w1 = np.zeros(784)  
 bet = 0.5  
 c = 1e-4  
 wStar, w\_k = gradient\_descent(zero\_one\_images\_train, zero\_one\_labels\_train, w1, 100, 0.25, bet, c, 0.0001)  
 error\_train = calculateError(w\_k[len(w\_k) - 1], w\_k)  
 wStar\_test, w\_k\_test = gradient\_descent(zero\_one\_images\_test, zero\_one\_labels\_test,w1, 100*,* *0.25,* bet, c, 0.0001)  
 error\_test = calculateError(w\_k\_test[len(w\_k) - 1], w\_k\_test)  
  
 plt.semilogy(error\_train, label='train '+str(digit1)+',' + str(digit2))  
 plt.semilogy(error\_test, label='test '+str(digit1)+',' + str(digit2))  
 plt.xlabel("iteration")  
 plt.ylabel(r'$|f(w^{(k)}-f(w^\*)|$')  
 plt.legend()  
 plt.show()  
  
def runAndPlotNewton(digit1, digit2):  
 zero\_one\_images\_train, zero\_one\_labels\_train = filterTwoDigits(digit1, digit2, x\_train, y\_train)  
 zero\_one\_images\_train = imageToVector(zero\_one\_images\_train)  
 zero\_one\_images\_test, zero\_one\_labels\_test = filterTwoDigits(digit1, digit2, x\_test, y\_test)  
 zero\_one\_images\_test = imageToVector(zero\_one\_images\_test)  
  
 for i in range(len(zero\_one\_images\_train)):  
 zero\_one\_images\_train[i] = (zero\_one\_images\_train[i]) / 255  
  
 for i in range(len(zero\_one\_images\_test)):  
 zero\_one\_images\_test[i] = (zero\_one\_images\_test[i]) / 255  
 zero\_one\_images\_train = np.transpose(np.asarray(zero\_one\_images\_train))  
 zero\_one\_labels\_train = np.asarray(zero\_one\_labels\_train)  
 zero\_one\_images\_test = np.transpose(np.asarray(zero\_one\_images\_test))  
 zero\_one\_labels\_test = np.asarray(zero\_one\_labels\_test)  
  
 w1 = np.zeros(784)  
 bet = 0.5  
 c = 1e-4  
 wStar, w\_k = exactNewton(zero\_one\_images\_train, zero\_one\_labels\_train, w1, 100, 1, bet, c, 0.0001)  
 error\_train = calculateError(w\_k[len(w\_k) - 1], w\_k)  
 wStar\_test, w\_k\_test = exactNewton(zero\_one\_images\_test, zero\_one\_labels\_test, w1, 100, 1, bet, c, 0.0001)  
 error\_test = calculateError(w\_k\_test[len(w\_k) - 1], w\_k\_test)  
  
 plt.semilogy(error\_train, label='train ' + str(digit1) + ',' + str(digit2))  
 plt.semilogy(error\_test, label='test ' + str(digit1) + ',' + str(digit2))  
 plt.xlabel("iteration")  
 plt.ylabel(r'$|f(w^{(k)}-f(w^\*)|$')

plt.legend()  
 plt.show()  
  
runAndPlotGD(0,1)  
runAndPlotGD(8,9)  
  
runAndPlotNewton(0,1)  
runAndPlotNewton(8,9)

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generated