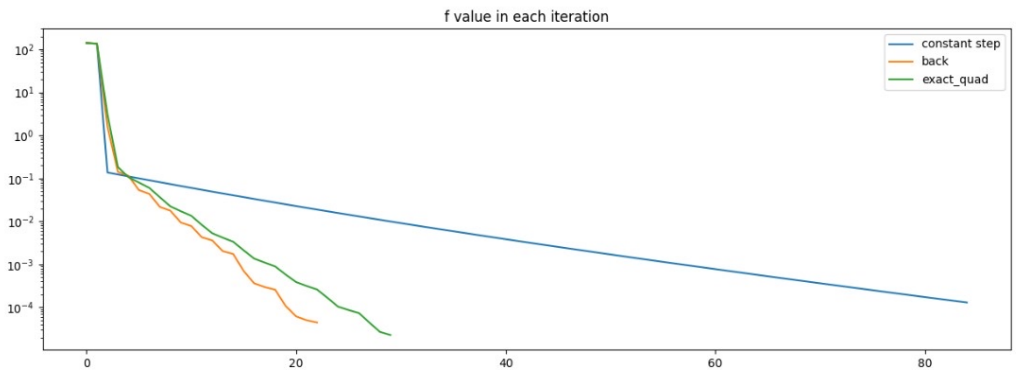
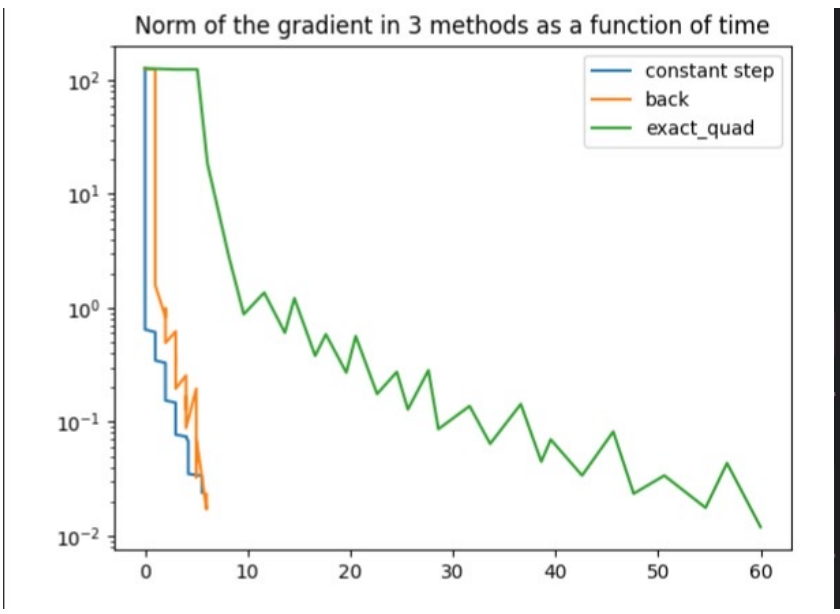
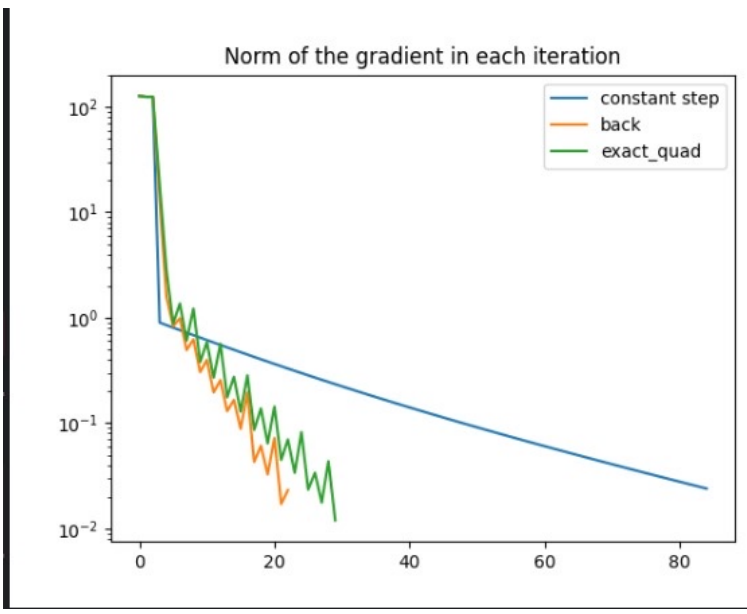
```python
def generic_grad(f, gf, lsearch, x0, eps):
    start_time = time.time() * 1000
    x, i = [x0], 0
    x.append(x[0] - lsearch(f, x[0], gf(x0)))
    fs, gs = [f(x[0]), f(x[1])], [np.linalg.norm(gf(x[0])), np.linalg.norm(gf(x[1]))]
    ts = [0, (time.time()*1000 - start_time)]
    while abs(f(x[i]) - f(x[i+1])) > eps:
        i += 1
        gk = gf(x[i])
        t = lsearch(f, x[i], gk)
        x.append(x[i] - t * gk)
        fs.append(f(x[i+1]))
        gs.append(np.linalg.norm(gk))
        ts.append(time.time()*1000 - start_time)
    return x[i], ts, fs, gs

def back(f, xk, gk, alpha, beta, s):
    t = s
    while f(xk - t * gk) >= f(xk) - alpha * t * (np.linalg.norm(gk) ** 2):
        t = beta * t
    return t


A = np.random.uniform( low: 0,  high: 1,  size: (20, 5))
AA = np.matmul(A.T, A)
f = lambda x: np.linalg.norm(np.matmul(A, x)) ** 2
gf = lambda x: 2 * np.matmul(AA, x)
s = 1/(2 * max(np.linalg.eigvals(A.T @ A)))

eps = 10**-5
x0 = np.ones((5,1))
x1, ts1, fs1, gs1 = generic_grad(f, gf, lambda f, xk, gk: const_step(f, xk, gk, s), x0, eps)
x2, ts2, fs2, gs2 = generic_grad(f, gf, lambda f, xk, gk: back(f, xk, gk,  alpha: 0.5,  beta: 0.5,  s: 1), x0, eps)
x3, ts3, fs3, gs3 = generic_grad(f, gf, lambda f, xk, gk: exact_quad(f, xk, gk, AA), x0, eps)
# Create the graphs
plt.figure(figsize=(15, 5))

# f value in each iteration
plt.semilogy( *args: fs1, label='constant step')
plt.semilogy( *args: fs2, label='back')
plt.semilogy( *args: fs3, label='exact_quad')
plt.title('f value in each iteration')
plt.legend()
plt.show()
```

```python
def const_step(f, xk, gk, s):
    return s


1 usage  Gil Caplan
def exact_quad(f, xk, gk, A):
    try:
        np.linalg.cholesky(A)
    except np.linalg.LinAlgError:
        raise
    #return ((np.linalg.norm(gk) ** 2) / (2 * np.matmul(np.matmul(gk.T, A), gk)))
    t_values = np.linspace( start: 0.01,  stop: 1,  num: 100)
    return t_values[np.argmin([f(xk - t * gk) for t in t_values])]
```

```python
# Norm of the gradient in each iteration
plt.semilogy( *args: gs1, label='constant step')
plt.semilogy( *args: gs2, label='back')
plt.semilogy( *args: gs3, label='exact_quad')
plt.title('Norm of the gradient in each iteration')
plt.legend()
plt.show()

# Norm of the gradient in 3 methods as a function of time
plt.semilogy( *args: ts1, gs1, label='constant step')
plt.semilogy( *args: ts2, gs2, label='back')
plt.semilogy( *args: ts3, gs3, label='exact_quad')
plt.title('Norm of the gradient in 3 methods as a function of time')
plt.legend()
plt.show()
```







(א)

$$\underline{\text{MDS:}}\quad \min_{x} S(x) = \sum_{i<j} \left(\|x_i - x_j\|^2 - D_{ij}^2 \right)^2 = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\|x_i - x_j\|^2 - D_{ij}^2 \right)^2$$

נשים לב כאן שאנו מנסים למצוא $x_1, x_2, \ldots, x_N$ שמקיימים $\forall i: x_i \in \mathbb{R}^2$.

נסביר כעת למה בעיית ה-MDS משיקה לנו את הקירובים שבקורס המקורות שלנו במובן הזה שהמרחקים בין הנקודות שנמצא יהיו דומים למרחקים בין הנקודות המקוריות. כלומר במקום מטלה של "שונה" בין המרחק בין שני נקודות כלשהן שמנסים למצוא לבין המרחק האמיתי בין שני נקודות המקוריות. ז"א מציאה של הסכום של השונות הכללית אנו נמצא את הנקודות שעוברים לנו במעוזם הקירוב הכי טוב למרחקים האמיתיים. ניתן לחשוב על זה כמו MSE.

(ב)

נשים לב כי ניתן להפעיל שיטת הגרדיאנט כי $\|x_i - x_j\|^2$ - נורמה שאנו יודעים עם ביחס ל-x_i ועם ביחס ל-x_j.

המהלך $S(x^k)$ - חסומה מלמטה ע"י 0, ומה היא מונוטונית יורדת ולכן קיים לה גבול. כעת, לפי הגדרת קושי קיים N כך שלכל $N < k < m$:

$$\underbrace{|S(x^k) - S(x^m)|}_{\text{סדרה מונ' יורדת}} = S(x^k) - S(x^m) < \varepsilon$$

$\Rightarrow$ הגענו על ההגבלות מקיים.

②

$$\frac{\partial \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\|x_i-x_j\|^2-D_{ij}^2\right)^2\right)}{\partial x_k} = \text{נגזור לפי}$$

$$= \frac{\partial \frac{1}{2}\left[\sum_{i=1}^{N}\left(\|x_i-x_k\|^2-D_{ik}^2\right)^2+\sum_{j=1}^{N}\left(\|x_k-x_j\|^2-D_{kj}^2\right)^2\right]}{\partial x_k} =$$

$$= \frac{1}{2}\cdot 2\cdot 2\left[\sum_{i=1}^{N}\left(\|x_i-x_k\|^2-D_{ik}^2\right)\cdot\frac{\partial\|x_i-x_k\|^2}{\partial x_k}\right] \quad (*)$$

שניהם אפס המוזר"מ

$$(1) \quad = 2\cdot 2\left[\sum_{i=1}^{N}\left(\|x_i-x_k\|^2-D_{ik}^2\right)(x_i-x_k)\right]$$

*

① $\|x_i-x_k\|^2 = \|x_k-x_i\|^2$ (מכונת הנורמה)

② $\sum_{j=1}^{N}\left(\|x_k-x_j\|^2-D_{kj}^2\right) \overset{①}{=} \sum_{j=1}^{N}\left(\|x_j-x_k\|^2-D_{jk}^2\right) = \sum_{i=1}^{N}\left(\|x_i-x_k\|^2-D_{ik}^2\right)$

$D_{ij}=D_{ji}$ מכיוון ש-

מכפלה שווה סכימה

$$\Rightarrow (1) = 2\cdot 2\sum_{j=1}^{N}\left(\|x_j-x_k\|^2-D_{jk}^2\right)(x_j-x_k) =$$

$$= 4\sum_{j=1}^{N}\left(\|x_j-x_k\|^2-D_{jk}^2\right)(x_j-x_k)$$

$$\Rightarrow \forall k=1,\dots,N: \quad \frac{\partial \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\|x_i-x_j\|^2-D_{ij}^2\right)^2\right)}{\partial x_k} = 4\sum_{j=1}^{N}\left(\|x_j-x_k\|^2-D_{jk}^2\right)(x_j-x_k)$$

$$\Rightarrow \nabla S(x) = 4\begin{pmatrix} \sum_{j=1}^{N}\left(\|x_1-x_j\|^2-D_{1j}^2\right)(x_1-x_j) \\ \vdots \\ \sum_{j=1}^{N}\left(\|x_N-x_j\|^2-D_{Nj}^2\right)(x_N-x_j) \end{pmatrix}$$

ד)

$x_1 = x_2 = \dots = x_N = (c,c) \in \mathbb{R}^2$. $D_{ij} > 0$

$$\forall k = 1,\dots,N : \quad (\nabla S(x))_k = \sum_{j=1}^{N} \left(\|x_k - x_j\|^2 - D_{kj}^2 \right)(x_k - x_j) = 0$$

$(\nabla S(x))_k$ — כוח הכוח הגרביטציוני מכוכב ה-k-י

$$\Rightarrow \nabla S(x) = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = 0$$

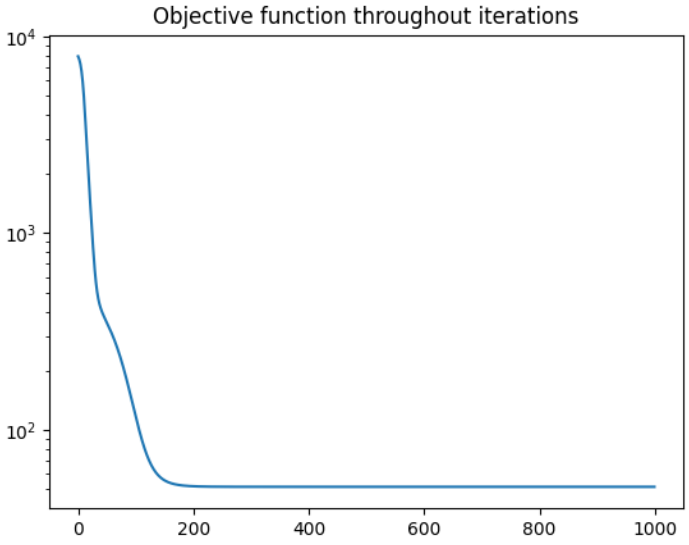
~~מכיוון ש $\nabla S(x) = 0$ בכל אזור "רחוק מהשפה" הבעיה:~~

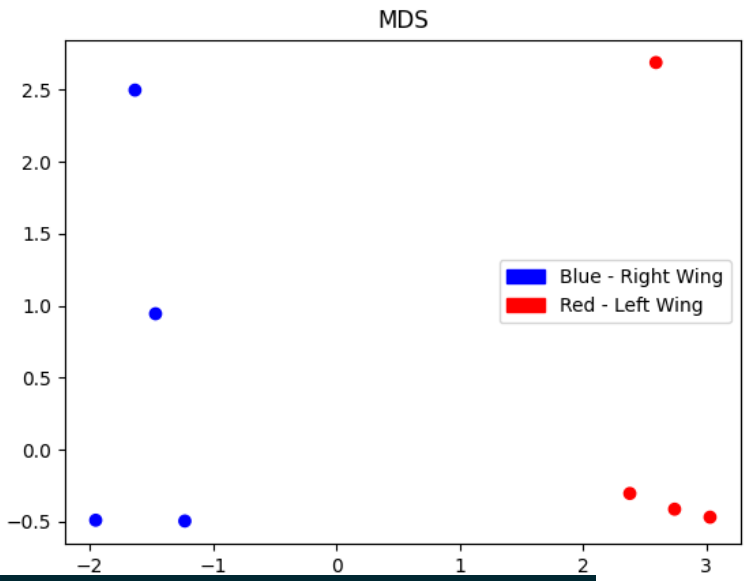
~~$\nabla S = 0 \Leftarrow x_k = x_{k+1} \; \forall k = 1,\dots,N-1$~~

מכיוון ש- $\nabla S(x) = 0$, אנו מתעלמים מזה נקודה מסוכנת וכך הנקודה שאלה מתאפסת שיטת הגרדיאנט ולכן לא "נגוע" מהנקודה אליה מתעלמים.

ה)

כן, לפי השרטוט אם מרכז המסה בכל איטרציה, אנו רואים שהאלגוריתם אכן מתכנס






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

O = np.array([
    [ 1,  0,  0,  0, -1, -1, -1, -1,  0],
    [ 0,  1, -1, -1, -1, -1, -1, -1, -1],
    [ 1,  1,  1,  1, -1, -1, -1, -1, -1],
    [ 0,  0,  1,  1, -1, -1, -1, -1, -1],
    [-1, -1, -1, -1,  1,  1,  1,  0,  0],
    [-1, -1,  1,  1,  1,  1,  0,  1,  1],
    [-1, -1,  1,  1,  1,  0,  1,  1,  0],
    [-1, -1,  1,  1,  0,  1,  1,  0,  0]
])

def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y)**2))

def grad_step(X, D):
    n = D.shape[0]
    grad = np.zeros((n, 2))

    for k in range(n):
        for j in range(n):
            if k != j:
                grad[k] += (euclidean_distance(X[k], X[j]) ** 2 - D[k, j]) * (X[k] - X[j])

    return grad

def obj_func(X, D):
    n = D.shape[0]
    obj = 0

    for i in range(n):
        for j in range(n):
            obj += (euclidean_distance(X[i], X[j]) ** 2 - D[i, j]) ** 2

    return obj / 2


def MDS(O: np.ndarray, t=0.001, max_iter=1000):
    f = []
    n = O.shape[0]
    # Sample n 2 dimensional points
    X = np.random.uniform(0, 1, (n, 2))
    # Calculate the dissimilarity matrix
    D = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            D[i, j] = euclidean_distance(O[i], O[j]) ** 2

    for _ in range(max_iter):
        grad = grad_step(X, D)
        X -= t * grad
        f.append(obj_func(X, D))

    return X, f


# Plot the results
colors = ['b', 'b', 'b', 'b', 'r', 'r', 'r', 'r']
X, f = MDS(O)
plt.scatter(X[:, 0], X[:, 1], c=colors)
plt.title('MDS')
blue_patch = mpatches.Patch(color='blue', label='Blue - Right Wing')
red_patch = mpatches.Patch(color='red', label='Red - Left Wing')
plt.legend(handles=[blue_patch, red_patch])
plt.show()

plt.semilogy(f)
plt.title('Objective function throughout iterations')
plt.show()
```

3) א)

$f(x,y)=x^2+y^4-y^2$

$\nabla f(x,y)=\begin{pmatrix}2x\\4y^3-2y\end{pmatrix} \Rightarrow \begin{matrix}2x=0 \Rightarrow x=0\\ 2y(2y^2-1)=2y(y+\frac{1}{\sqrt2})(y-\frac{1}{\sqrt2}) \Rightarrow y=0,\pm\frac{1}{\sqrt2}\end{matrix}$

$\nabla^2 f(x,y)=\begin{pmatrix}2&0\\0&12y^2-2\end{pmatrix}\succ 0 \Rightarrow$ הנק' מינימום לוקלי עבור $(0,\pm\frac{1}{\sqrt2})$

$\begin{pmatrix}2&0\\0&-2\end{pmatrix}$ עבור $(0,0)$ מטריצת ההסיין לא מוגדרת.

ב) $\{(x^k,y^k)\}_{k\geq0}$ עבור $x^0\in\mathbb{R}$, $y^0=0$, $\{t^k\}_{k\geq0}$

$\nabla f=\begin{pmatrix}2x\\4y^3-2y\end{pmatrix}$

נק' ראשונה $(x^0,0)$: $x^0\in\mathbb{R}$

נעקוב אחרי האלגוריתם gradient descent עם קווי מקדימה.

שלב ראשון:

נסתכל על קווי מקדימה: $\min_{t\geq0} f\left(\begin{pmatrix}x^0\\0\end{pmatrix}-t\begin{pmatrix}2x_0\\0\end{pmatrix}\right)=\min_{t\geq0} f\left(x^0(1-2t),y^0\right)$

נקבל מינימום כאשר $f(x,y)=x^2$ כי הנחנו בסעיף קודם שזה מינימום. לכן עבור $1-2t=0$ נקבל מינימום של f (נבחר) $t=\frac12$

שלב שני, עבור (x^1,y^1), $t=\frac12$: $(x^1,y^1)=(x^0,0)-\frac12\cdot(2x_0,0)$

$\Rightarrow (x^1,y^1)=(x^0,0)-(x_0,0)=(0,0)$

עכשיו קיבלנו דיוק וכן האלגוריתם עוצר ונקבל $(x^k,y^k)\to(0,0)$

לא מתכנס כי יכול לא יתקע במינימום לוקלי.

ב) שני הפיתרונות לא מתכנסים את המינימום על פי
הגרפים שהגרילו לישיבה ומכאן שני השני שהם ראש ניתן
לראות שהכאוס עשה כל מיני הפרעות בצד כמותר
כתישמרה על היקף.

לפי ניתן לעורך את הזה.

I

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


5 usages  new *
def f(x, y): return x ** 2 + y**4 - y**2
2 usages  new *
def gf(x, y): return (2 * x, 4 * y ** 3 - 2 * y)

mul = lambda x, t: (x[0] * t, x[1] * t) if type(x) is not int else x * t
min = lambda x, y: (x[0] - y[0], x[1] - y[1])
plus = lambda x, y: (x[0] + y[0], x[1] + y[1])
```

II

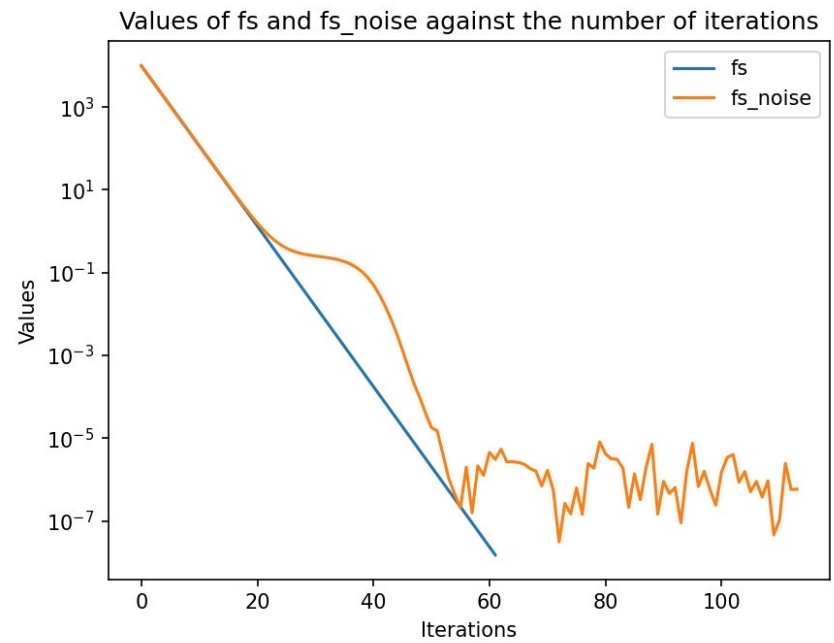
```python
def gradient_descent(x0, eps, mu=0, sigma=0, bias=0., r=False):
    x, i, t = [x0], 0, 0.1
    gk = gf(x[0][0], x[0][1])
    beta = np.random.normal(mu, sigma, size=2) if r else (0,0)
    x.append(plus(min(x[i], mul(gk, t)), beta))
    fs = [f(x[0][0], x[0][1])+0.25, f(x[1][0], x[1][1]) + bias]
    while abs(f(x[i][0], x[i][1]) - f(x[i + 1][0], x[i+1][1])) > eps:
        i += 1
        gk = gf(x[i][0], x[i][1])
        beta = np.random.normal(mu, sigma, size=2) if r else (0,0)
        x.append(plus(min(x[i], mul(gk, t)), beta))
        fs.append(f(x[i + 1][0], x[i+1][1]) + bias)
    return x, fs
```

```python
1 usage  Gil Caplan *
def ex3(mu, sigma, x0, eps):
    x, fs = gradient_descent(x0, eps)
    x_noise, fs_noise = gradient_descent(x0, eps, mu, sigma, bias: 0.25, r: True)
    return x, x_noise, fs, fs_noise
```

```python
[x, x_noise, fs, fs_noise] = ex3( mu: 0,  sigma: 0.0005,  x0: (100,0), 10**-8)
```

III

```python
iterations = range(len(fs))
iterations_noise = range(len(fs_noise))
plt.figure()
plt.semilogy( *args: iterations, fs, label='fs')
plt.semilogy( *args: iterations_noise, fs_noise, label='fs_noise')
plt.xlabel('Iterations')
plt.ylabel('Values')
plt.title('Values of fs and fs_noise against the number of iterations')
plt.legend()
plt.show()
```



4.

$f: \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$, $\nabla f(x) \neq 0$

~~[illegible]~~

$f'(x;d) = \nabla f(x)^T d$

$\Rightarrow \min_{d \in \mathbb{R}^n} \{ f'(x;d) \mid \|d\| = 1 \} = \min_{d \in \mathbb{R}^n} \{ \nabla f(x)^T d \mid \|d\| = 1 \}$

לפי אי-שוויון קושי-שוורץ:

$$\nabla f(x)^T d \geq -\|d\| \, \|\nabla f(x)\| = -\|\nabla f(x)\|$$

$\|d\| = 1$ מתקיים בכיוון הנגדי של הגרדיאנט

לכן, $\|\nabla f(x)\|$ הוא חסם תחתון לביטוי $f'(x;d)$.

זה נכון עבור $d = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$ מכיוון ש:

① $\|d\| = 1$

② $$f'(x;d) = \nabla f(x)^T d = -\frac{\nabla f(x)^T \nabla f(x)}{\|\nabla f(x)\|} =$$

$$= -\frac{\|\nabla f(x)\|^2}{\|\nabla f(x)\|} = -\|\nabla f(x)\|$$

לכן מצאנו d שעבורו הביטוי הגדול מקבל מינימום:

$$d^* = -\frac{\nabla f(x)}{\|\nabla f(x)\|} \in \arg\min_{d \in \mathbb{R}^n} \{ f'(x;d) \mid \|d\| = 1 \}$$

כלומר d^* - כיוון הירידה המקסימלי.