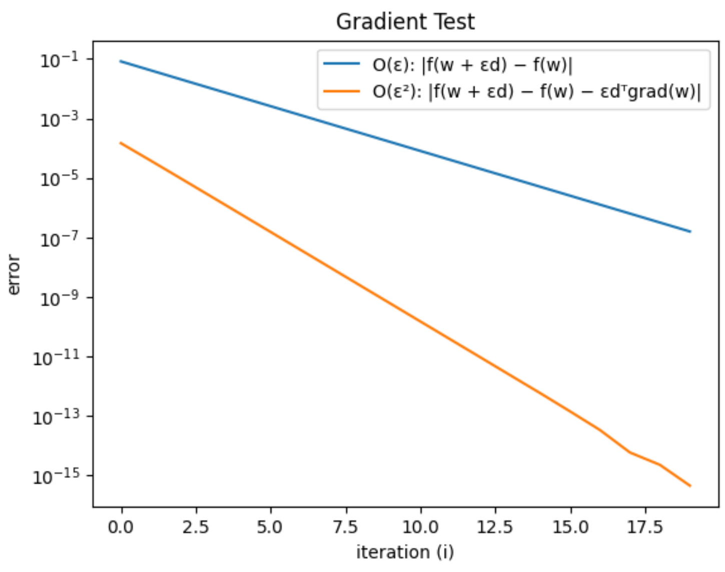
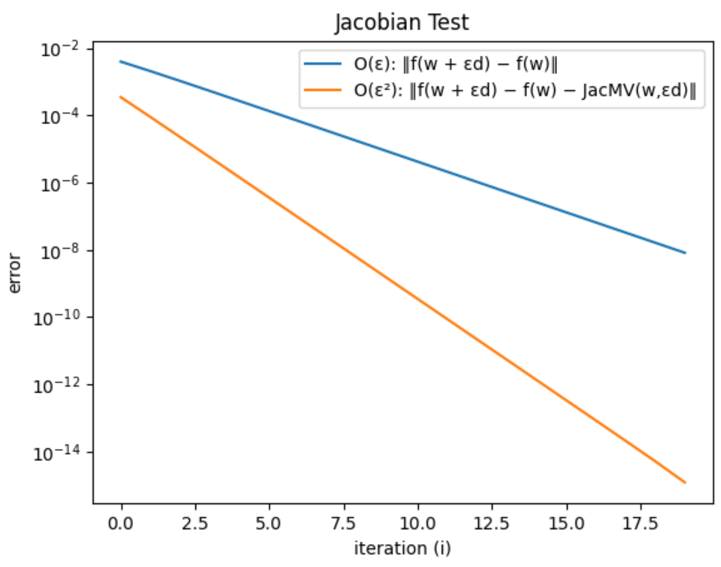
שאלה 4
סעיף א':

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def calc_obj_grad_hess_4a(X, y):
    c1 = y
    c2 = np.ones(y.shape[0]) - c1
    m = X.shape[1]
    Xt = np.transpose(X)

    def objective(w):
        Xtw = Xt @ w
        sig_Xtw = sigmoid(Xtw)
        t1 = np.transpose(c1) @ np.log(sig_Xtw)
        t2 = np.transpose(c2) @ np.log(1 - sig_Xtw)
        return (-1 / m) * (t1 + t2)  # F(w)=(-1/m) (c1^⊤*log(σ(X^⊤w))
+c2^⊤*log(1-σ(X^⊤w)))

    def gradient(w):
        Xtw = Xt @ w
        sig_Xtw = sigmoid(Xtw)
        return (1 / m) * X @ (sig_Xtw - c1)

    def hessian(w):
        Xtw = Xt @ w
        sig_Xtw = sigmoid(Xtw)
        return (1 / m) * (X @ np.diag((sig_Xtw * (1 - sig_Xtw))) @ Xt)

    return objective, gradient, hessian
```

סעיף ב':

מבחני גרדיאנט וג'קוביאן עבור הפונקציות מסעיף א'. ניתן לראות כי המבחנים עברו בהצלחה 😎





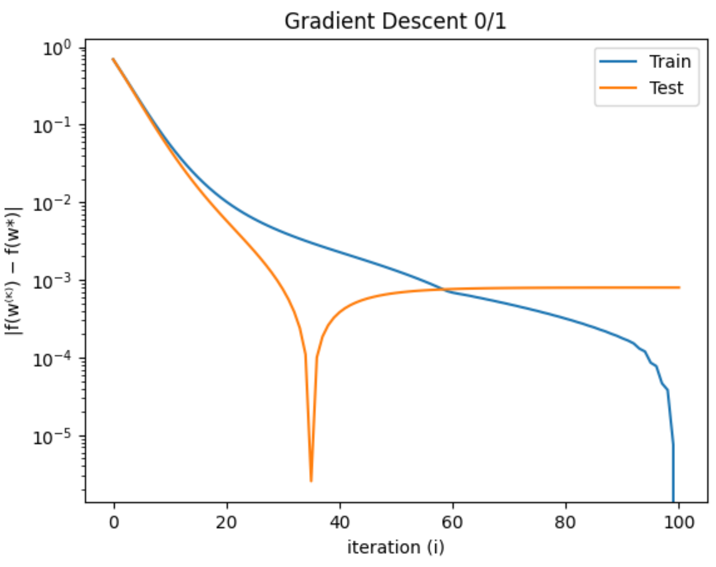
קוד עבור סעיף ב':

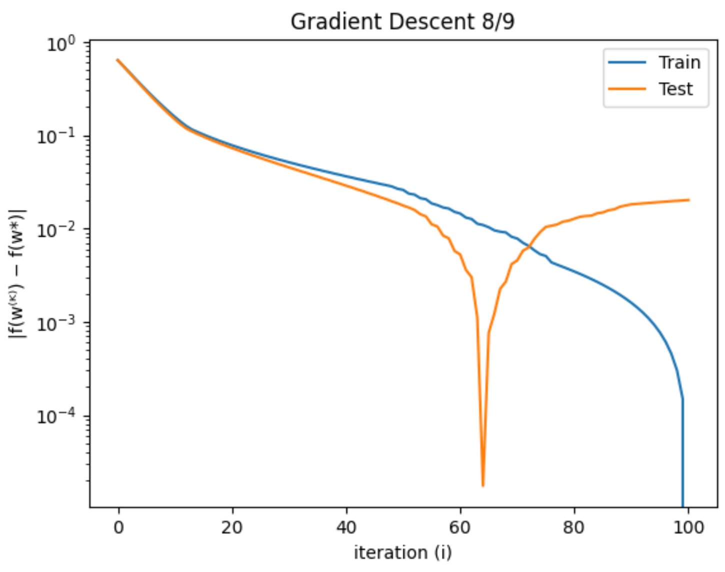
```
def gradient_test(F, grad, n,
                  epsilon = 0.1):

    w = np.random.rand(n)
    d = np.random.rand(n)
    d = d / np.linalg.norm(d)
    F0 = F(w)
    g0 = grad(w)
    y0 = np.zeros(n)
    y1 = np.zeros(n)
    for i in range(0, n):
        epsi = epsilon * (0.5 ** i)
        Fi = F(w + epsi * d)
        F1 = F0 + epsi * np.dot(g0, d)
        y0[i] = abs(Fi - F0)
        y1[i] = abs(Fi - F1)
        print(i, "\t", y0[i], "\t", y1[i])

    plt.semilogy(y0)
    plt.semilogy(y1)
    plt.legend(("O(ε): |f(w + εd) - f(w)|”,
                "O(ε²): |f(w + εd) - f(w) -
εdᵀgrad(w)|"))
    plt.title('Gradient Test')
    plt.xlabel('iteration (i)')
    plt.ylabel('error')
    plt.show()
    return 0
```

```
def jacobian_test(grad, hess, n,
                  epsilon = 0.1):

    w = np.random.rand(n)
    d = np.random.rand(n)
    d = d / np.linalg.norm(d)
    F0 = grad(w)
    y0 = np.zeros(n)
    y1 = np.zeros(n)

    for i in range(0, n):
        epsi = epsilon * (0.5 ** i)
        Fi = grad(w + epsi * d)
        F1 = F0 + np.transpose(
                  hess(w)) @ (epsi * d)
                  # H(f(x))=J(∇f(x))T

        y0[i] = np.linalg.norm(Fi - F0)
        y1[i] = np.linalg.norm(Fi - F1)
        print(i, "\t", y0[i], "\t", y1[i])

    plt.semilogy(y0)
    plt.semilogy(y1)
    plt.legend(("O(ε): ‖f(w + εd) - f(w)‖”,
        "O(ε²):‖f(w + εd) - f(w) - JacMV(w,εd)‖"))
    plt.title('Jacobian Test')
    plt.xlabel('iteration (i)')
    plt.ylabel('error')
    plt.show()
    return 0
```

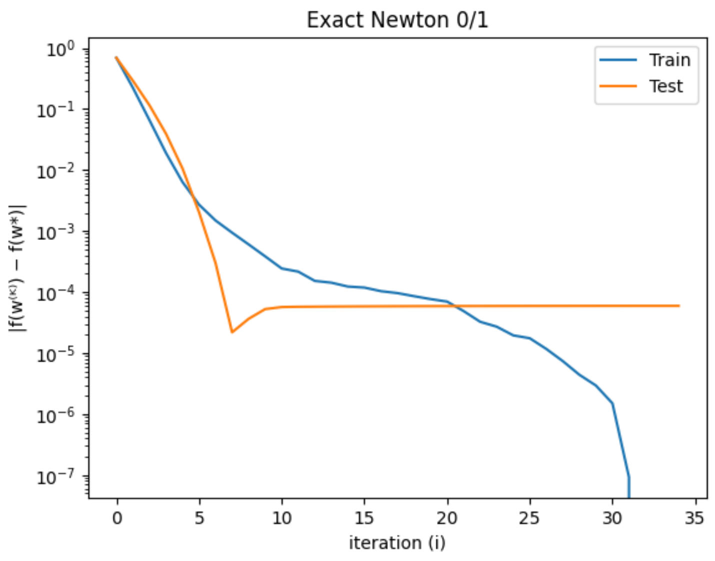
<u>סעיף ג':</u>

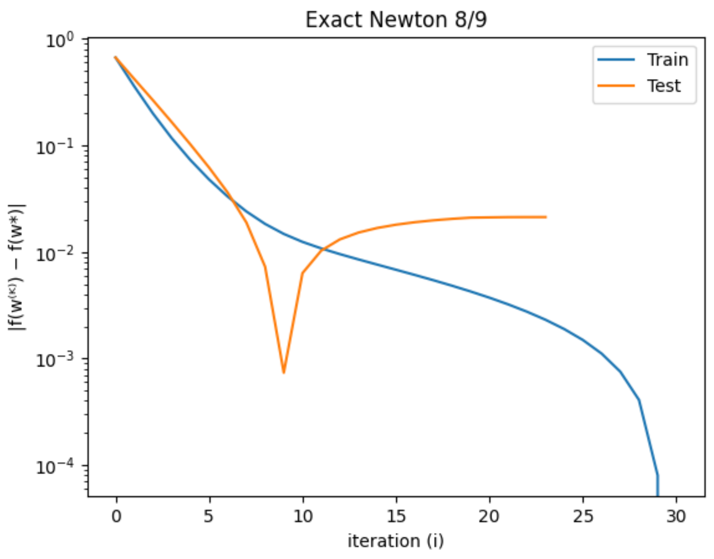
<u>Gradient Descent</u>





<u>Exact Newton</u>





ניתן לראות את הoverfitting שהתקבל עבור הTest. כמו כן ניתן להבחין כי בשיטת Exact Newton הגענו להתכנסות לאחר כ-30 איטרציות בשני המקרים לעומת כ-100 בשיטת Gradient Descent.

קוד עבור סעיף ג':

```
def GD(f, grad, w, maxIter, eps=0.001):
    history = [f(w)]
    for k in range(0, maxIter):
        print(f"Iter {k}: {history[len(history) - 1]}")
        d = -grad(w)
        d = d / np.linalg.norm(d)
        alpha = armijo_step(w, f, grad, d)
        w = w + alpha * d
        history.append(f(w))
        l = len(history) - 1
        if np.linalg.norm(history[l] - history[l-1])/np.linalg.norm(history[l-1])<eps:
            break
    return w, history
```

```
def EN(f, grad, hess, w, maxIter, eps=0.001):
    history = [f(w)]
    for k in range(0, maxIter):
        print(f"Iter {k}: {history[len(history) - 1]}")
        hw=hess(w)
        if np.linalg.det(hw)==0:
            hw = hw+(np.abs(np.min(np.linalg.eigvals(hw)))+0.01)*np.eye(hw.shape[0])
        d = -np.linalg.inv(hw) @ grad(w)
        d = d / np.linalg.norm(d)
        alpha = armijo_step(w, f, grad, d, 1)
        w = w + alpha * d
        history.append(f(w))
        l=len(history) - 1
        if np.linalg.norm(history[l] - history[l-1])/np.linalg.norm(history[l-1])<eps:
            break
    return w, history
```

חישוב גודל הצעד בשיטת Armijo:

```python
def armijo_step(w, f, grad, d, alpha=0.1):
    betta = 0.25
    c = 0.1
    for i in range(10):
        phi = f(w + alpha * d)
        cond = f(w) + c * alpha * np.dot(grad(w), d)
        if phi <= cond:
            return alpha
        alpha = betta * alpha
    return -1
```

קריאה לשיטת Gradient Descent:

```python
def steepest_descent(dig1, dig2):  # 9 is good(=1), 8 is not good(=0)
    (X_train, Y_train), (X_test, Y_test) = mnist_dataloader.load_data()
    X_train, Y_train = np.asarray(X_train), np.asarray(Y_train)
    train_filter = np.where((Y_train == dig1) | (Y_train == dig2), True, False)
    X_train, Y_train = X_train[train_filter], Y_train[train_filter]
    y = np.where(Y_train == dig2, 1, 0)
    xt = np.transpose(X_train.reshape(len(X_train), 784)) / 256
    f, g, h = calc_obj_grad_hess_4a(xt, y)
    w = np.zeros(784)
    x1, train_hist = GD(f, g, w, 100)

    X_test, Y_test = np.asarray(X_test), np.asarray(Y_test)
    test_filter = np.where((Y_test == dig1) | (Y_test == dig2), True, False)
    X_test, Y_test = X_test[test_filter], Y_test[test_filter]
    ytest = np.where(Y_test == dig2, 1, 0)
    xtest = np.transpose(X_test.reshape(len(X_test), 784)) / 256
    f_test, g_test, h_test = calc_obj_grad_hess_4a(xtest, ytest)
    x2, test_hist = GD(f_test, g_test, w, 100)

    plt.semilogy(np.abs(train_hist - train_hist[len(train_hist) - 1]))
    plt.semilogy(np.abs(test_hist - train_hist[len(train_hist) - 1]))
    plt.legend(("Train", "Test"))
    plt.title(f'Gradient Descent {dig1}/{dig2}')
    plt.xlabel('iteration (i)')
    plt.ylabel('|f(w⁽ᴷ⁾) - f(w*)|')
    plt.show()
    return 0
```

(הקריאה לשיטת Exact Newton והדפסת תוצאותיה נעשו באופן דומה ואף זהה)