In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import random
from random import randint
```

In [ ]:

```
def backtracking(f, grad_f, x, alpha = 1, c = 0.8, tau = 0.25):
    while f(x - alpha*grad_f(x)) > f(x) - c * alpha * np.linalg.norm(grad_f(x)) ** 2
        alpha = tau * alpha
        if alpha < 1e-3:
            break
    return alpha
```

In [ ]:

```
def GD(f, grad_f, x0, tolf, tolx, kmax, alpha= 1, bt= False):
    xk = x0
    f_vals = [f(xk)]
    grad_vals = [grad_f(xk)]
    err_vals = [np.linalg.norm(grad_f(xk))]
    x_vals = [xk]
    iteration = 0

    while iteration < kmax:
        x_prec = xk

        xk = xk - alpha * grad_f(xk)

        if bt:
            alpha = backtracking(f, grad_f, xk)

        x_vals.append(xk)
        f_vals.append(f(xk))
        grad_vals.append(grad_f(xk))
        err_vals.append(np.linalg.norm(grad_f(xk)))

        iteration+=1

        if np.linalg.norm(grad_f(xk)) < tolf * np.linalg.norm(grad_f(x0)):
            break

        if np.linalg.norm(xk - x_prec) < tolx * np.linalg.norm(x0):
            break

    return (x_vals, iteration, f_vals, grad_vals, err_vals)
```

In [ ]:

```
def f(x):
    return (2*(x[0])**2 + (x[1]-2)**2)
def grad_f(x):
    return np.array((4*x[0], 2*x[1] - 4))

def my_plot(xk_vals, k, f_vals, grad_valks, err_vals, f, title):
    xv = np.linspace(-10, 10, 100).T
    yv = np.linspace(-10, 10, 100).T

    xx,yy = np.meshgrid(xv, yv)

    zz = f([xx, yy])

    xk_vals = np.array(xk_vals)
    plt.plot(xk_vals[:,0], xk_vals[:,1], '--ro')

    plt.contour(xx, yy, zz)
    plt.title(title)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.grid()
    plt.show()

def my_plot_2D(xk_vals, k, f_vals, grad_valks, err_vals, f, title):
    x_vals = np.linspace(-3, 3, 100)
    y_vals = []
    for x in x_vals:
        y_vals.append(f([x]))
    plt.plot(x_vals, y_vals)
    plt.scatter(xk_vals, f_vals, c='green')
    plt.title(title)
    plt.show()

def plot_error(iters, errs, labels, title = "Error (2-norms of gradient)"):
    colors = []

    for i in range (len(iters)):
        colors.append('#%06X' % randint(0, 0xFFFFFF))

    colors = plt.get_cmap("tab20c")
    i= 0

    for item in zip(iters, errs, labels):
        plt.plot(item[0], item[1], c=colors(i/(len(iters)-1)), label = item[2])
        i+=1
    plt.title(title)

    plt.legend(loc="upper left")
    plt.show()
```

In [ ]:

```
def function_testing(f, grad_f, x0=0, title="", alpha = 1e-1, bt = False, oneD = Fal
    tolf = 1e-8
    tolx = 1e-8
    kmax = 300
    if not bt:
        xk_vals, k, f_vals, grad_vals, err_vals = GD(f, grad_f, x0, tolf, tolx, kmax
    else:
        alpha_bt = backtracking(f, grad_f, x0)
        xk_vals, k, f_vals, grad_vals, err_vals = GD(f, grad_f, x0, tolf, tolx, kmax
    if not oneD and not isMatr:
        if not bt:
            my_plot(xk_vals, k, f_vals, grad_vals, err_vals, f, title + " with alpha
        else:
            my_plot(xk_vals, k, f_vals, grad_vals, err_vals, f, title + " with backt
    elif not isMatr:
        if not bt:
            my_plot_2D(xk_vals, k, f_vals, grad_vals, err_vals, f, title + " with al
        else:
            my_plot_2D(xk_vals, k, f_vals, grad_vals, err_vals, f, title + " with ba
    print("Minimum Found =", xk_vals[k-1], "with", k, "iterations")

    to_ret = []
    if check_err:
        if not isMatr:
            for xk in xk_vals:
                to_ret.append(np.linalg.norm((xk - xtrue)))
        else:
            xtrue = np.array(xtrue)
            for xk in xk_vals:
                xk = np.array(xk)
                to_ret.append(np.linalg.norm((xk - xtrue)))

    if check_err:
        return np.arange(k+1), err_vals, to_ret
    return np.arange(k+1), err_vals
```

In [ ]:

```
lam = random.random()
def f1(x):
    return ((x[0] - 3)**2 + (x[1] - 1)**2)
def f2(x):
    return (10*(x[0] - 1)**2 + (x[1] - 2)**2)
def f3(x):
    x = np.array(x).T
    n = len(x)
    v = np.linspace(0, 1, n)
    A = np.vander(v)
    x_true = np.ones(n).T
    b= A @ x_true

    return ((np.linalg.norm((A @ x) - b)**2)/2)
def f4(x):
    n = len(x)
    v = np.linspace(0, 1, n)
    A = np.vander(v)
    x_true = np.ones(n).T
    b= A @ x_true

    return (((np.linalg.norm((A @ x) - b)**2)/2) + ((np.linalg.norm(x))**2)*lam/2)
def f5(x):
    return x[0]**4 + x[0]**3 - 2*(x[0]**2) - 2*x[0]
```

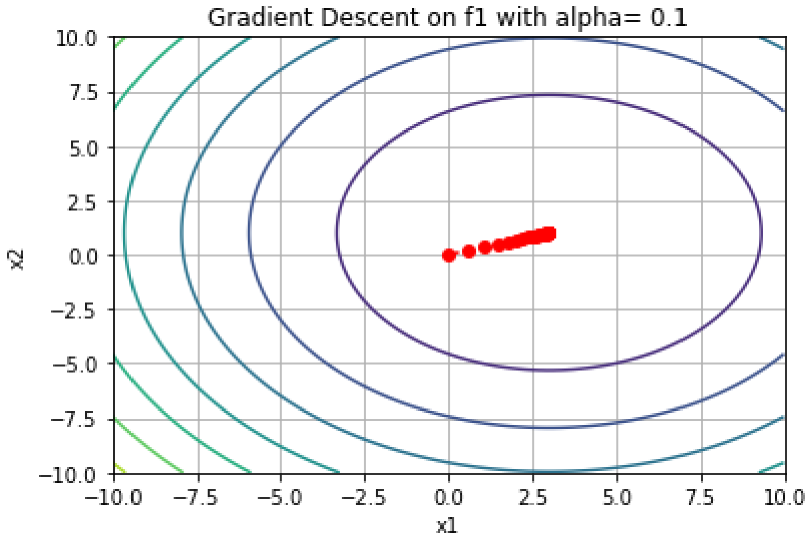
In [ ]:

```
def grad_f1(x):
    return np.array((2*x[0] - 6, 2*x[1] - 2))
def grad_f2(x):
    return np.array(20*(x[0] - 1), 2*(x[1] - 2))
def grad_f3(x):
    n = len(x)
    v = np.linspace(0,1,n)
    A = np.vander(v)
    x_true = np.ones(n).T
    b = A @ x_true
    return A.T@(A@x-b)
def grad_f4(x):
    return grad_f3(x) + lam*np.array(x)
def grad_f5(x):
    return np.array(4*x[0]**3 + 3*x[0]**2 - 4*x[0] - 2)
```

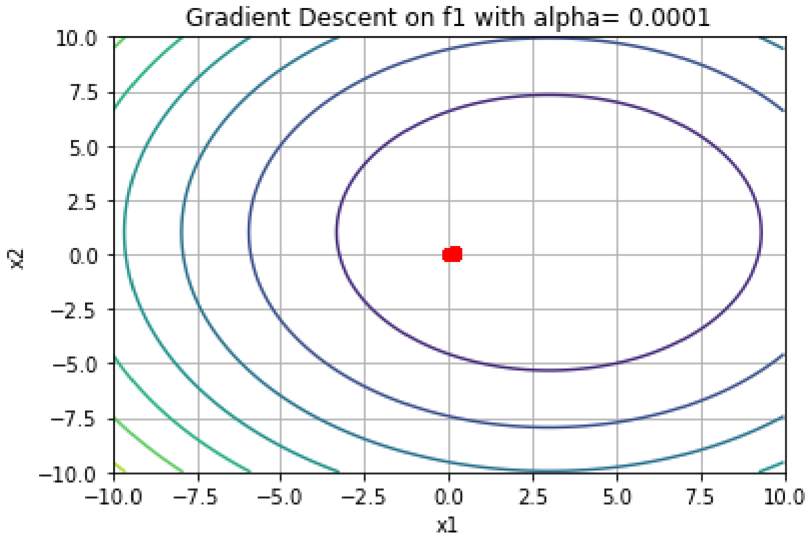
In [ ]:

```
iters = []
err_vals = []
labels = []
err_xtrue = []
xtrue = np.array([3,1]).T

el1, el2, el3 = function_testing(f1, grad_f1, x0 = np.array((0, 0)), title="Gradient
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-1")
```

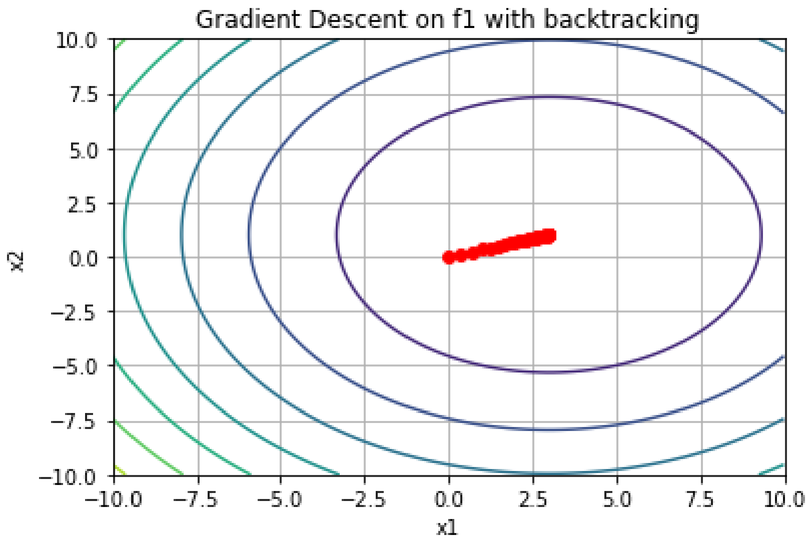
```
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f1, grad_f1, x0 = np.array((0, 0)), title="Gradient
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f1, grad_f1, x0 = np.array((0, 0)), title="Gradient
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking")
err_xtrue.append(el3)

plot_error(iters, err_vals, labels)
plot_error(iters, err_xtrue, labels, title="Error (distance from x_true)")
```



```
Minimum Found = [2.99999997 0.99999999] with 83 iterations
```

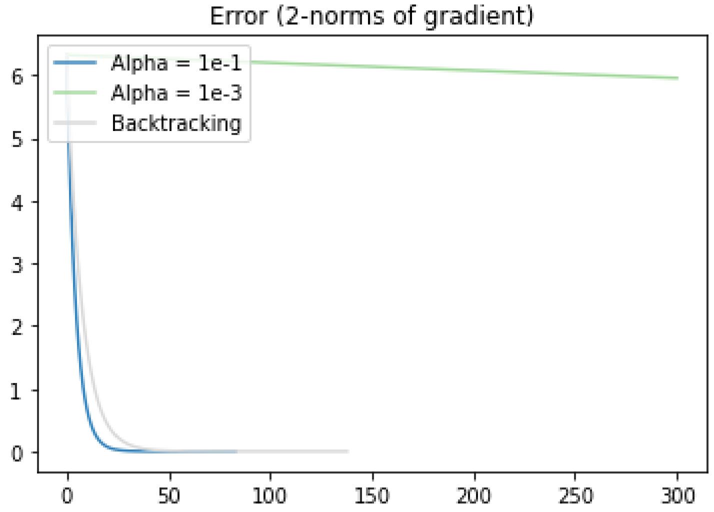


```
Minimum Found = [0.17415818 0.05805273] with 300 iterations
```

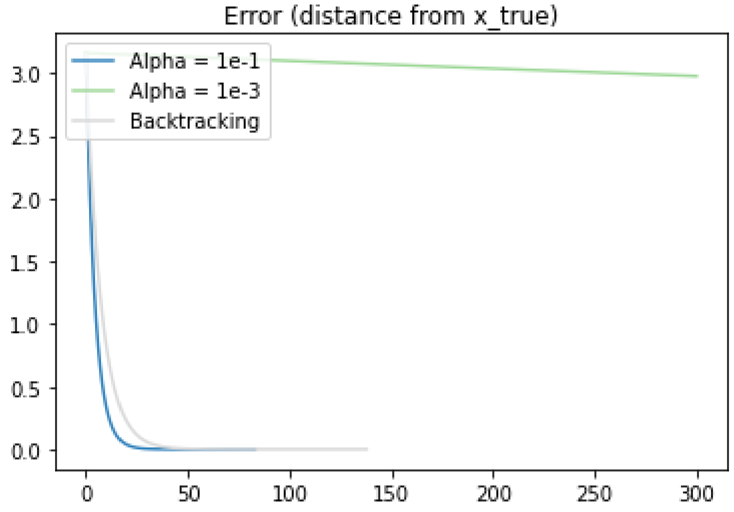


Gradient Descent on f1 with backtracking

```
Minimum Found = [2.99999997 0.99999999] with 138 iterations
```



Error (2-norms of gradient)

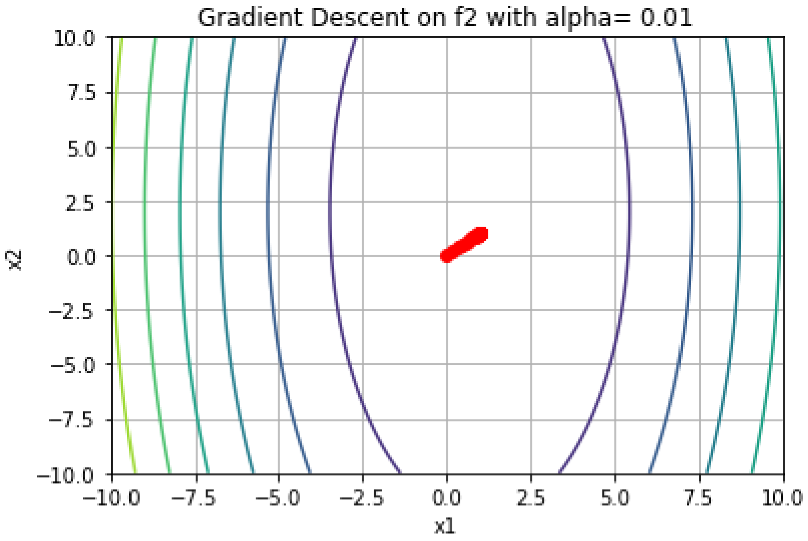


Error (distance from x_true)

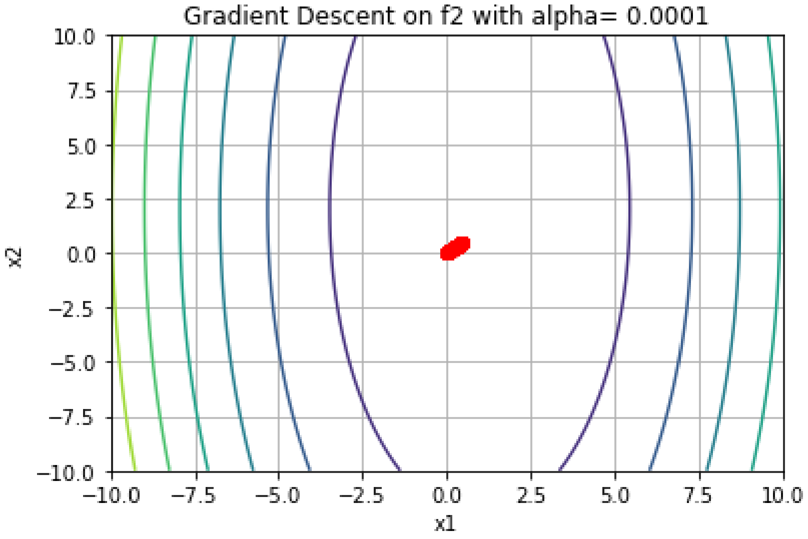
In [ ]:

```
iters = []
err_vals = []
labels = []
err_xtrue = []
xtrue = np.array([1,2]).T

el1, el2, el3 = function_testing(f2, grad_f2, x0 = np.array((0, 0)), title="Gradient
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-1")
err_xtrue.append(el3)
```

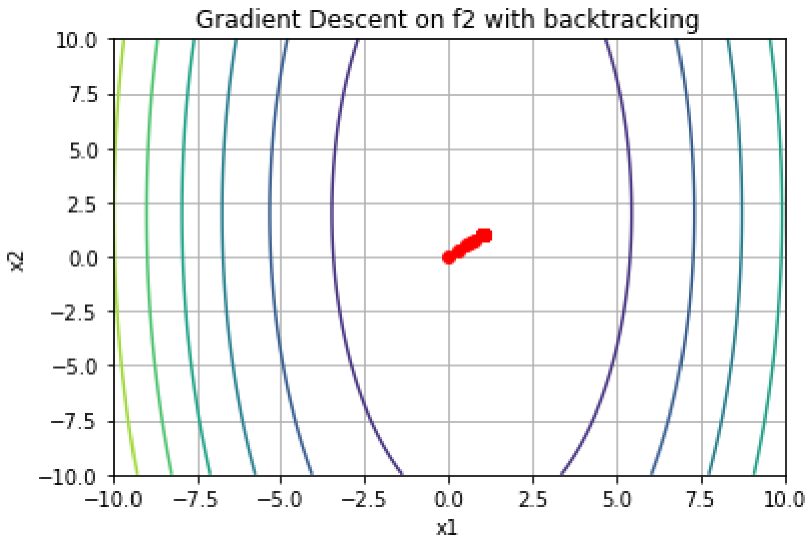
```
el1, el2, el3 = function_testing(f2, grad_f2, x0 = np.array((0, 0)), title="Gradient
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f2, grad_f2, x0 = np.array((0, 0)), title="Gradient
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking")
err_xtrue.append(el3)

plot_error(iters, err_vals, labels)
plot_error(iters, err_xtrue, labels)
```



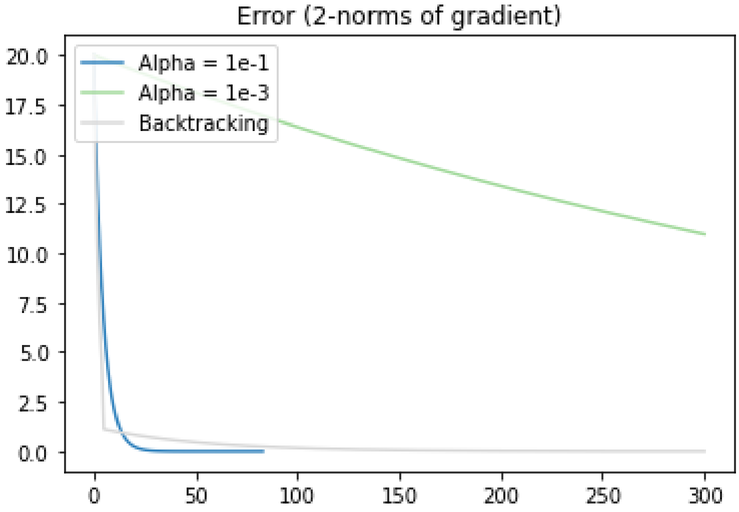
```
Minimum Found = [0.99999999 0.99999999] with 83 iterations
```

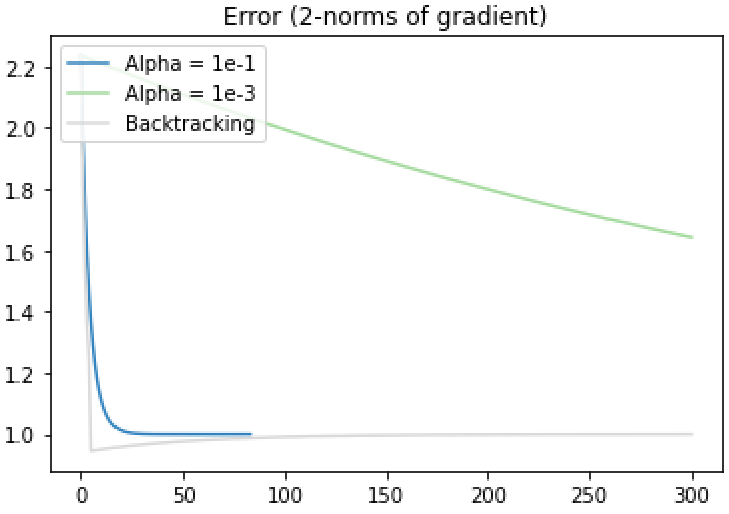


```
Minimum Found = [0.45041883 0.45041883] with 300 iterations
```



```
Minimum Found = [1.00016926 1.00016926] with 300 iterations
```



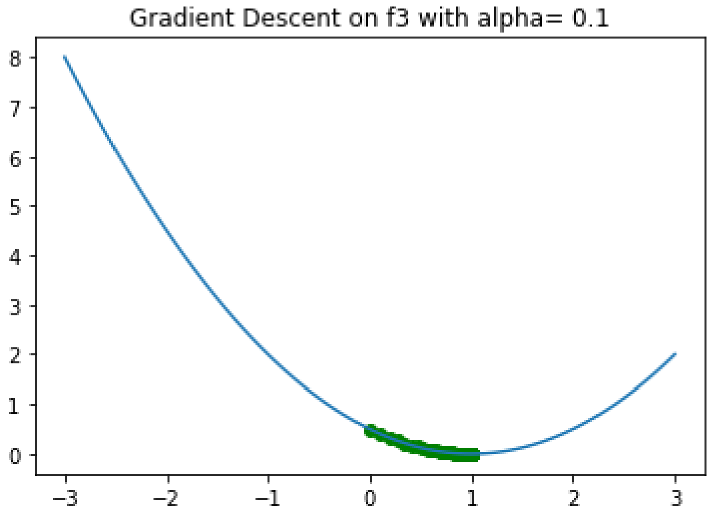


In [ ]:

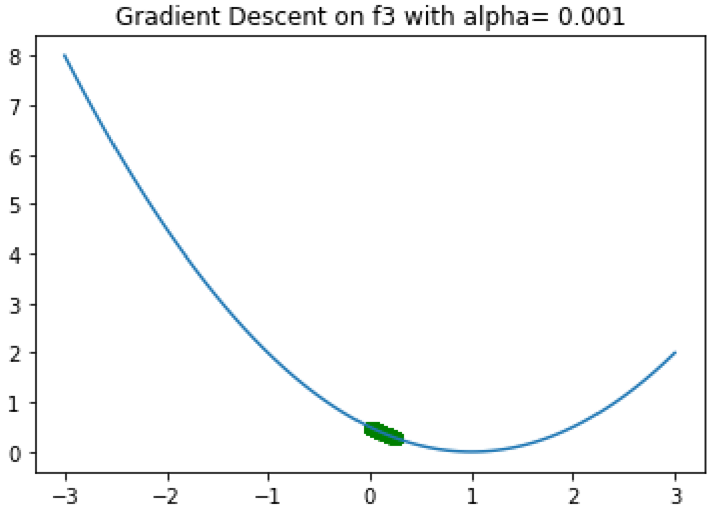
```
iters = []
err_vals = []
labels = []
err_xtrue = []

print("n = 1")
xtrue = np.ones(1).T

el1, el2, el3 = function_testing(f3, grad_f3, x0 = [0], title="Gradient Descent on f
iters.append(el1)
err_vals.append(el2)
```

```
labels.append("Alpha = 1e-1")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f3, grad_f3, x0 = [0], title="Gradient Descent on f
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f3, grad_f3, x0 = [0], title="Gradient Descent on f
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking")
err_xtrue.append(el3)

plot_error(iters, err_vals, labels)
plot_error(iters, err_xtrue, labels, title="Error (distance from x_true)")
```

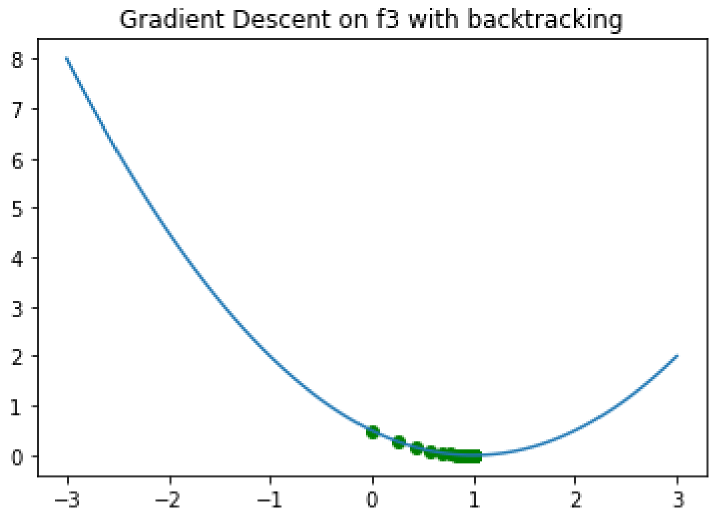
```
n = 1
```



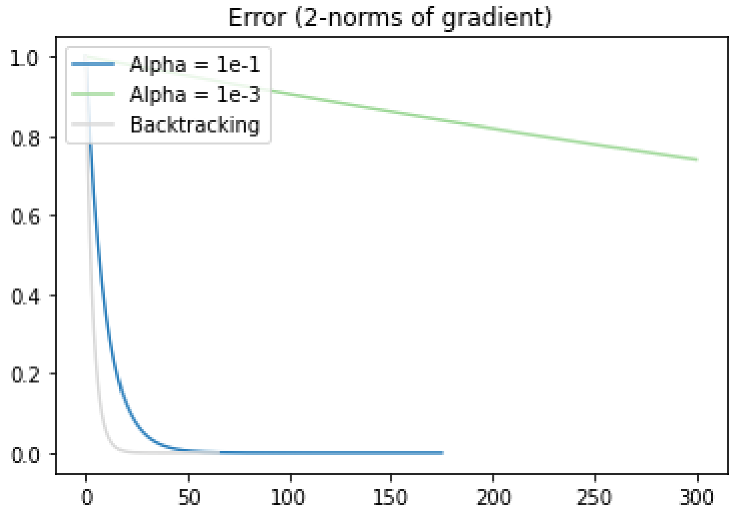
```
Minimum Found = [0.99999999] with 175 iterations
```

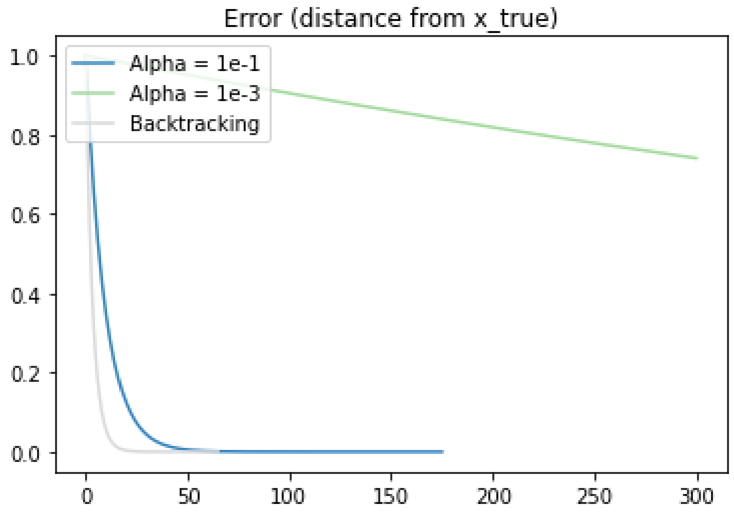


```
Minimum Found = [0.25855152] with 300 iterations
```



```
Minimum Found = [0.99999999] with 65 iterations
```



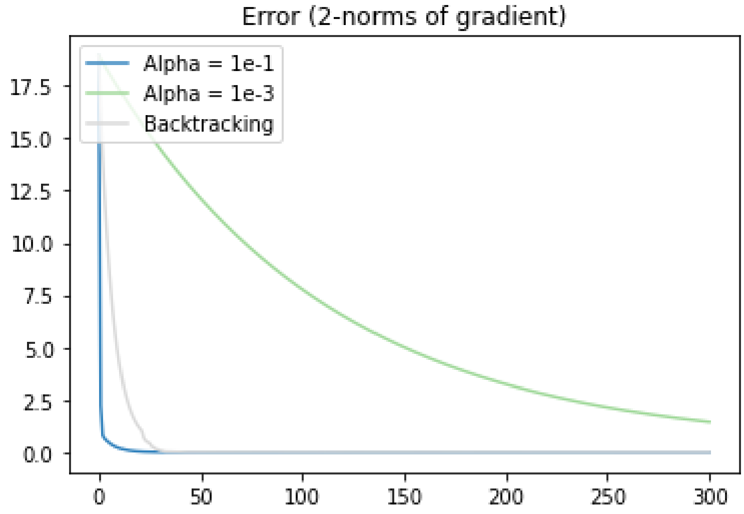


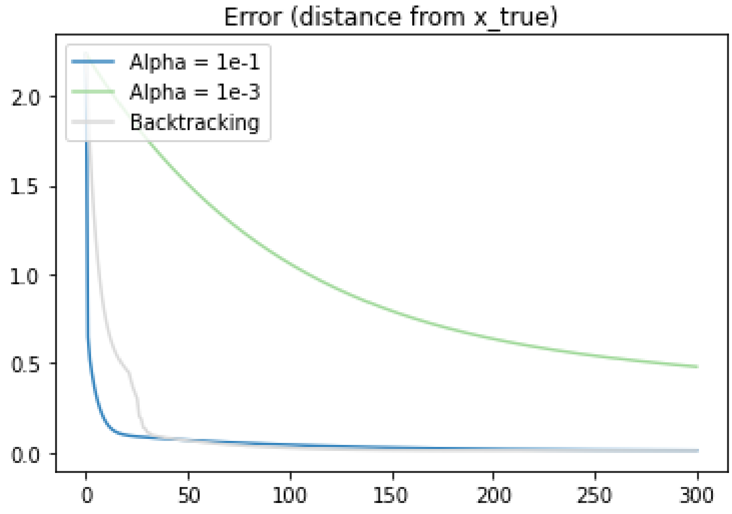
In [ ]:

```
iters = []
err_vals = []
labels = []
err_xtrue = []

print("n = 5")
xtrue = np.ones(5).T
x0 = np.zeros(5)

el1, el2, el3 = function_testing(f3, grad_f3, x0 = x0, title="Gradient Descent on f3
iters.append(el1)
err_vals.append(el2)
```

```
labels.append("Alpha = 1e-1")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f3, grad_f3, x0 = x0, title="Gradient Descent on f3
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f3, grad_f3, x0 = x0, title="Gradient Descent on f3
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking")
err_xtrue.append(el3)

plot_error(iters, err_vals, labels)
plot_error(iters, err_xtrue, labels, title="Error (distance from x_true)")
```

```
n = 5
Minimum Found = [0.99163637 1.00089691 1.00736081 1.00026101 0.99911969] with 300 it
erations
Minimum Found = [0.70041381 0.74416544 0.81164195 0.92495403 1.18560809] with 300 it
erations
Minimum Found = [0.99477693 1.00214716 1.0061874  0.99660827 1.00014983] with 300 it
erations
```


Error (2-norms of gradient)

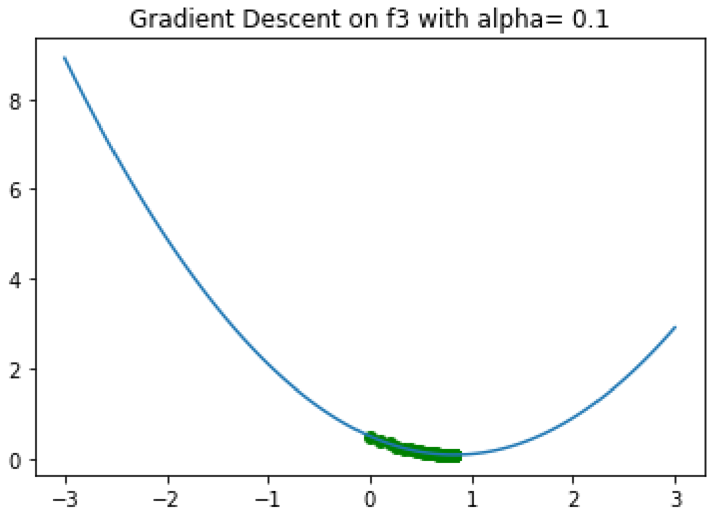

Error (distance from x_true)

In [ ]:

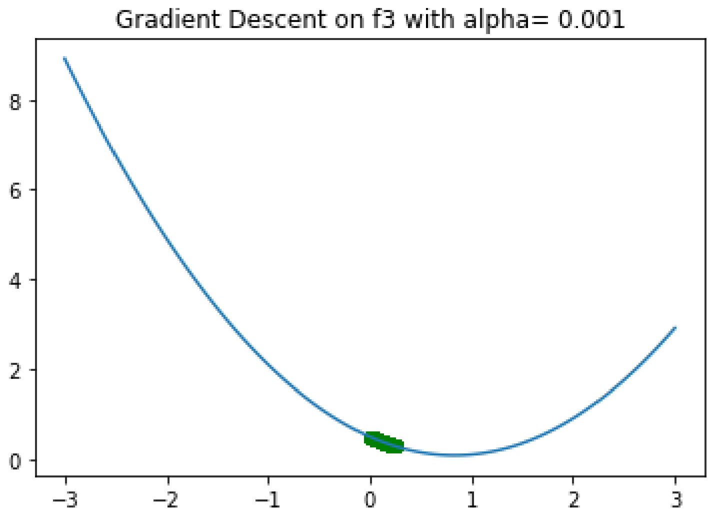
```
iters = []
err_vals = []
labels = []
err_xtrue = []
```

```
print("n = 1")
xtrue = np.ones(1).T

el1, el2, el3 = function_testing(f4, grad_f4, x0 = [0], title="Gradient Descent on f
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-1")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f4, grad_f4, x0 = [0], title="Gradient Descent on f
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f4, grad_f4, x0 = [0], title="Gradient Descent on f
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking")
err_xtrue.append(el3)

plot_error(iters, err_vals, labels)
plot_error(iters, err_xtrue, labels, title="Error (distance from x_true)")
```

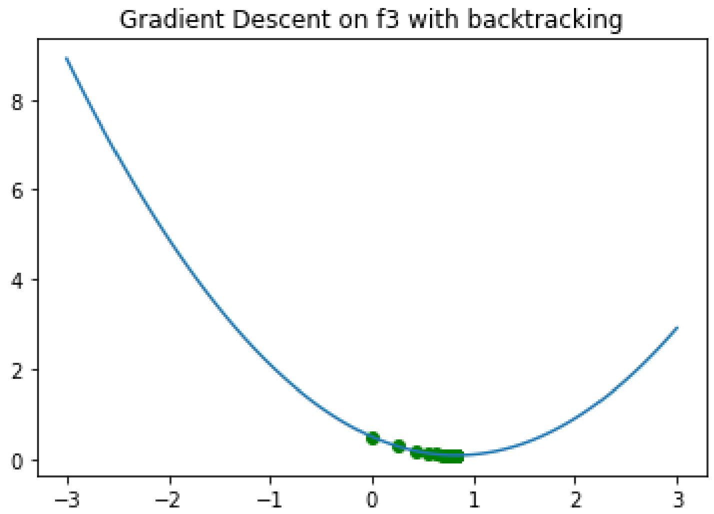
```
n = 1
```



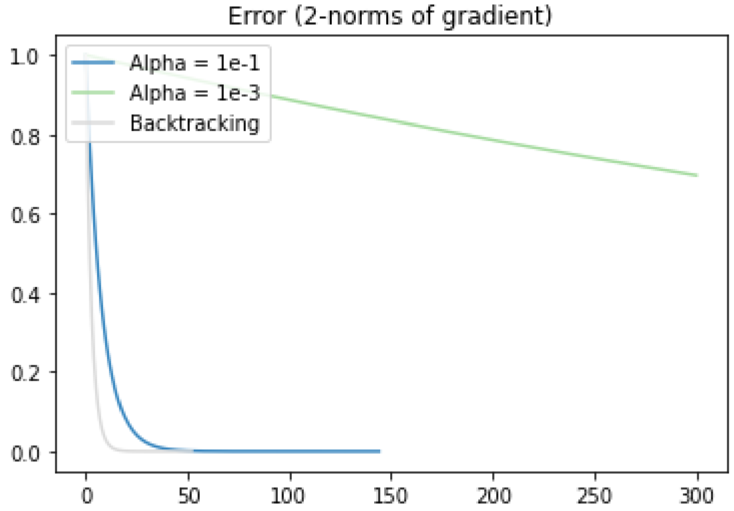
```
Minimum Found = [0.83147589] with 144 iterations
```

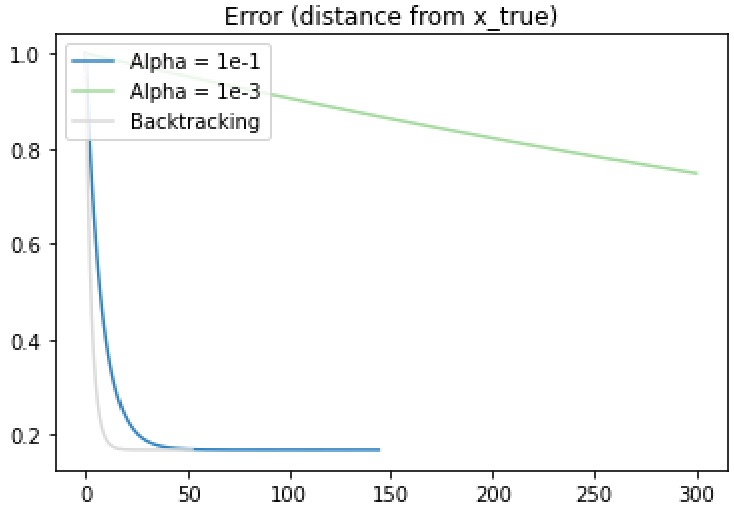


```
Minimum Found = [0.25126923] with 300 iterations
```


Gradient Descent on f3 with backtracking

```
Minimum Found = [0.83147589] with 52 iterations
```


Error (2-norms of gradient)

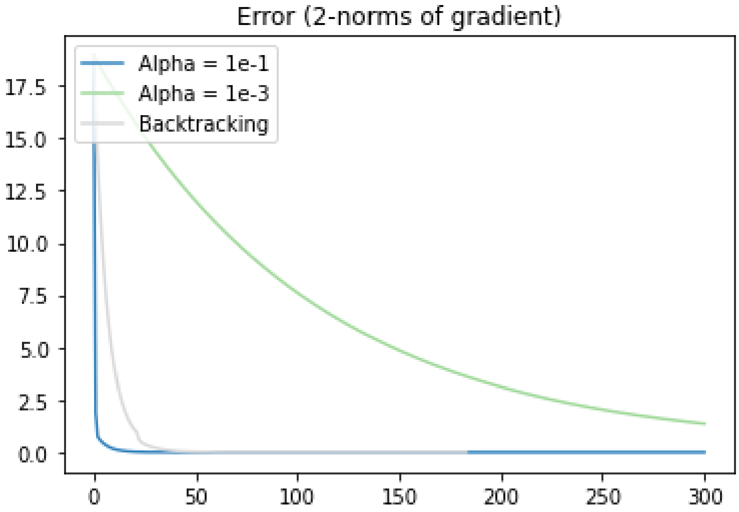

Error (distance from x_true)

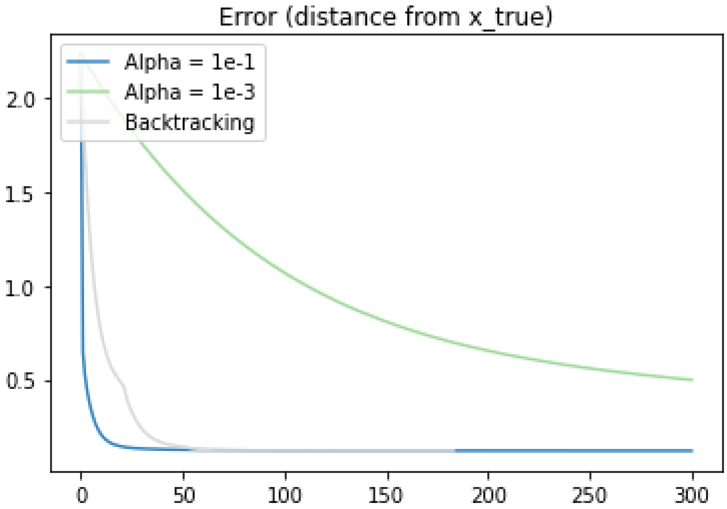
In [ ]:

```
iters = []
err_vals = []
labels = []
err_xtrue = []

print("n = 5")
xtrue = np.ones(5).T
x0 = np.zeros(5)

el1, el2, el3 = function_testing(f4, grad_f4, x0 = x0, title="Gradient Descent on f3
iters.append(el1)
err_vals.append(el2)
```

```
labels.append("Alpha = 1e-1")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f4, grad_f4, x0 = x0, title="Gradient Descent on f3
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3")
err_xtrue.append(el3)

el1, el2, el3 = function_testing(f4, grad_f4, x0 = x0, title="Gradient Descent on f3
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking")
err_xtrue.append(el3)

plot_error(iters, err_vals, labels)
plot_error(iters, err_xtrue, labels, title="Error (distance from x_true)")
```

```
n = 5
Minimum Found = [0.90459675 0.93318309 0.97062456 1.01608673 1.00932355] with 300 it
erations
Minimum Found = [0.6867671  0.72980327 0.7962228  0.90788947 1.16602198] with 300 it
erations
Minimum Found = [0.90459852 0.9331839  0.97062402 1.0160844  1.00932417] with 183 it
erations
```

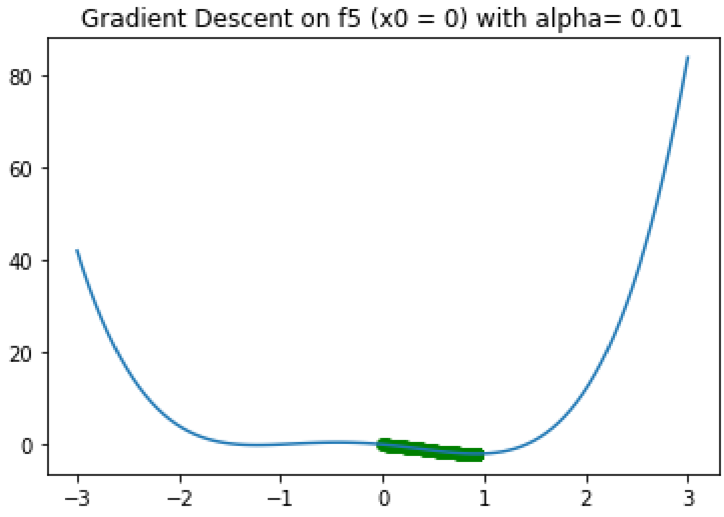




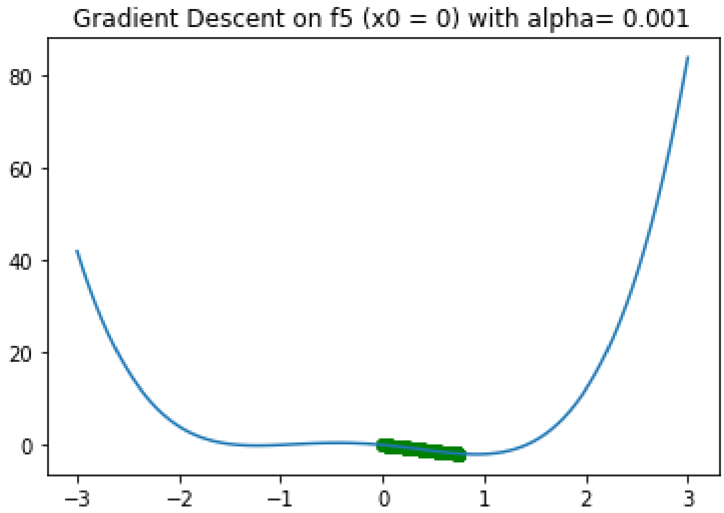
In [ ]:

```
iters = []
err_vals = []
labels = []
```

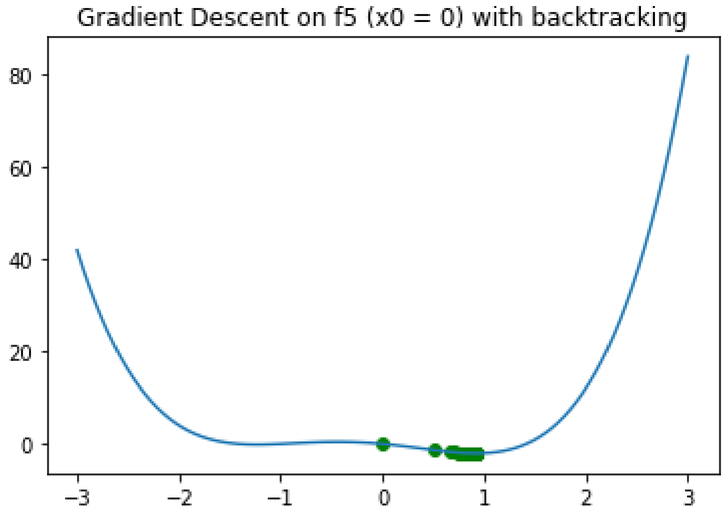
```
el1, el2 = function_testing(f5, grad_f5, x0 = [0], title="Gradient Descent on f5 (x0
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-1 (x0 = 0)")

el1, el2 = function_testing(f5, grad_f5, x0 = [0], title="Gradient Descent on f5 (x0
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3 (x0 = 0)")

el1, el2 = function_testing(f5, grad_f5, x0 = [0], title="Gradient Descent on f5 (x0
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking (x0 = 0)")

plot_error(iters, err_vals, labels)
iters = []
err_vals = []
labels = []

el1, el2 = function_testing(f5, grad_f5, x0 = [-3], title="Gradient Descent on f5 (x
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-1 (x0 = -3)")

el1, el2 = function_testing(f5, grad_f5, x0 = [-3], title="Gradient Descent on f5 (x
iters.append(el1)
err_vals.append(el2)
labels.append("Alpha = 1e-3 (x0 = -3)")

el1, el2 = function_testing(f5, grad_f5, x0 = [-3], title="Gradient Descent on f5 (x
iters.append(el1)
err_vals.append(el2)
labels.append("Backtracking (x0 = -3)")

plot_error(iters, err_vals, labels)
```



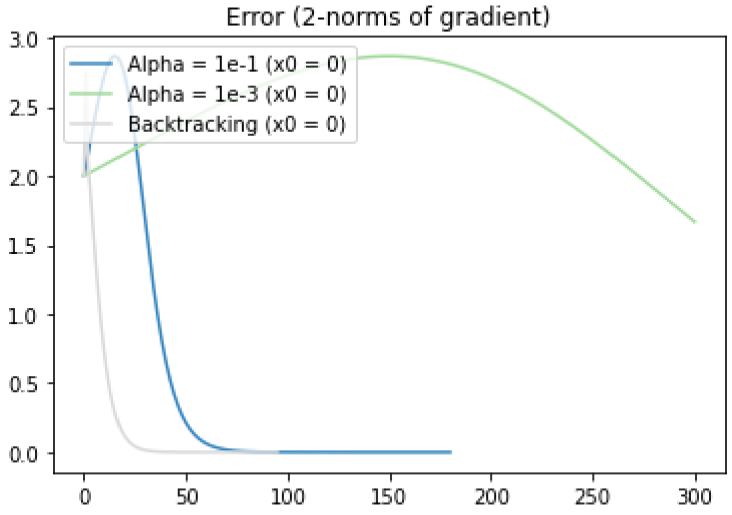
```
Minimum Found = [0.92222479] with 180 iterations
```

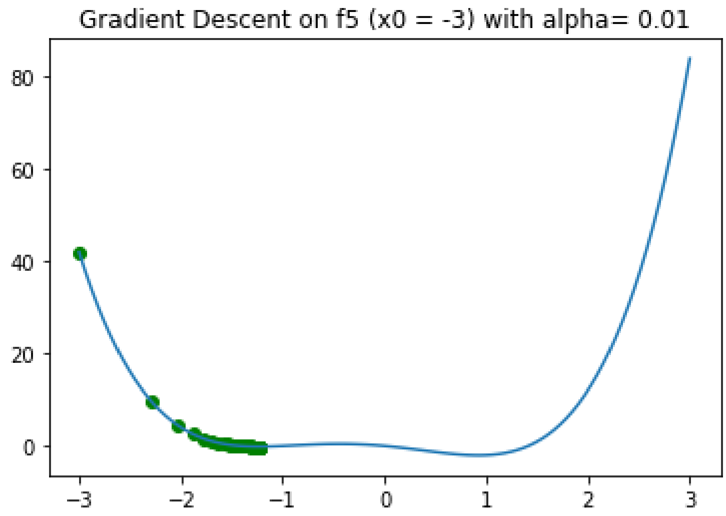


```
Minimum Found = [0.74222729] with 300 iterations
```

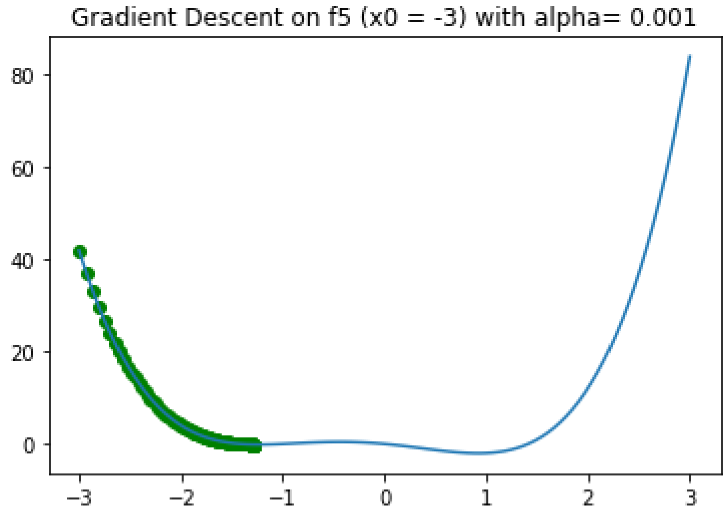


```
Minimum Found = [0.9222248] with 95 iterations
```

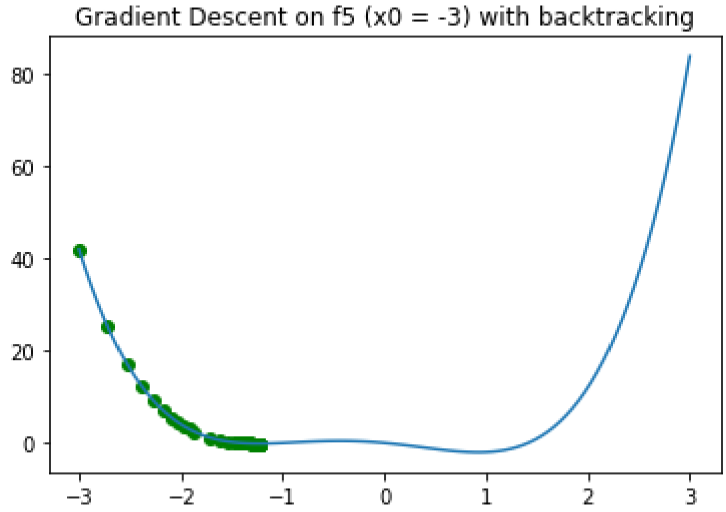




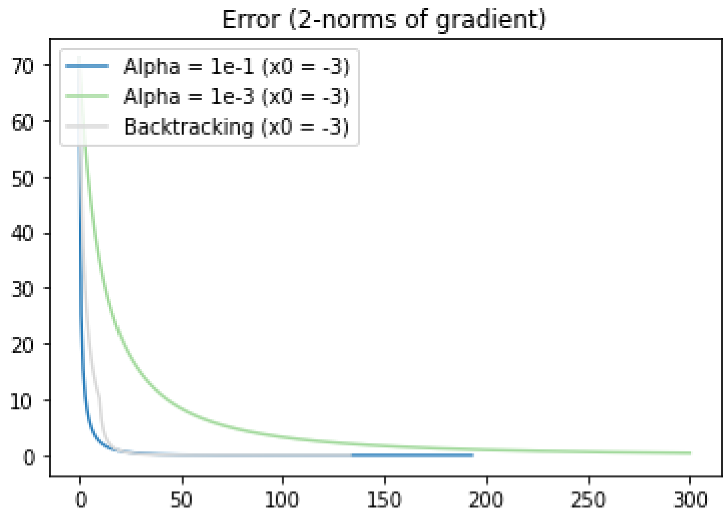
```
Minimum Found = [-1.23224005] with 193 iterations
```



```
Minimum Found = [-1.28699186] with 300 iterations
```



```
Minimum Found = [-1.2322399] with 133 iterations
```



Stochastic Gradient Descent

In [ ]:

```
# Utils

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

def f(w, xhat):
    return sigmoid(xhat.T @ w)

def grad_f(w, xhat):
    return (sigmoid(xhat.T @ w) * (1 - sigmoid(xhat.T @ w)) * xhat.T)

def MSE(f_w_x, y):
    return np.linalg.norm((f_w_x-y))**2

def grad_MSE(grad_f_w_x, f_w_x, y):
    return grad_f_w_x.T * (f_w_x - y)

def ell(w, X, Y):
    d, N = X.shape

    mse_sum = 0
    for i in range(0, N):
        mse_sum+=MSE(f(w, X[:, i]), Y[i])

    return mse_sum / N

def grad_ell(w, X, Y):
    d, N = X.shape

    grad_mse_sum = 0
    for i in range(0, N):
        grad_mse_sum += grad_MSE(np.array(grad_f(w, X[:, i])), f(w, X[:, i]), Y[i])

    return grad_mse_sum / N
```

In [ ]:

```
def x_split(X, Y, N_train):
    d, N = X.shape

    idx = np.arange(N)
    np.random.shuffle(idx)

    train_idx = idx[:N_train]
    test_idx = idx[N_train:]

    Xtrain = X[:, train_idx]
    Ytrain = Y[train_idx]

    Xtest = X[:, test_idx]
    Ytest = Y[test_idx]

    return Xtrain, Xtest, Ytrain, Ytest

def get_digits(X, Y, chosen_numbers):
    I = [idx for idx, elem in enumerate(Y) if elem in chosen_numbers]
    X_def = X[:, I]
    Y_def = Y[I]

    return X_def, Y_def
```

In [ ]:

```
def SGD(l, grad_l, w0, D, batch_size, n_epochs):
    alpha = 1e-3


    X, Y = D
    X_backup = X
    Y_backup = Y
    d, N = X.shape

    Xhat = np.concatenate((np.ones((1,N)), X), axis=0)

    n_batch_per_epoch = N // batch_size
    w_vals = [w0]
    f_vals = [l(w0, Xhat, Y)]
    grad_f_vals = [grad_l(w0, Xhat, Y)]
    err_vals = [np.linalg.norm(grad_l(w0, Xhat, Y))]

    for epoch in range(n_epochs):
        idx = np.arange(N)
        np.random.shuffle(idx)

        for k in range(n_batch_per_epoch):
            batch_indices = idx[k * batch_size : (k + 1) * batch_size]

            Mx = Xhat[:, batch_indices]
            My = Y[batch_indices]
            M = (Mx, My)


            w = w0 - alpha * grad_l(w0, Mx, My)
            w_vals.append(w)

            w0 = w

        X = X_backup
        Y = Y_backup
        f_vals.append(l(w, Xhat, Y))
        grad_f_vals.append(grad_l(w, Xhat, Y))
        err_vals.append(np.linalg.norm(grad_l(w, Xhat, Y)))

    return w, f_vals, grad_f_vals, err_vals
```

In [ ]:

```
import pandas as pd
```

In [ ]:

```
data = pd.read_csv("./data.csv")
data = np.array(data)
```

In [ ]:

```
X = data[:, 1:].T
Y = data[:, 0]

chosen_digits = [6, 9]
```

In [ ]:

```
X_set, Y_set = get_digits(X, Y, chosen_digits)
```

In [ ]:

```
d, N = X_set.shape
N_train = int(N/3*2)
Y_set[Y_set == chosen_digits[0]] = 0
Y_set[Y_set == chosen_digits[1]] = 1

X_train, X_test, Y_train, Y_test = x_split(X_set, Y_set, N_train)

D = (X_train, Y_train)
```

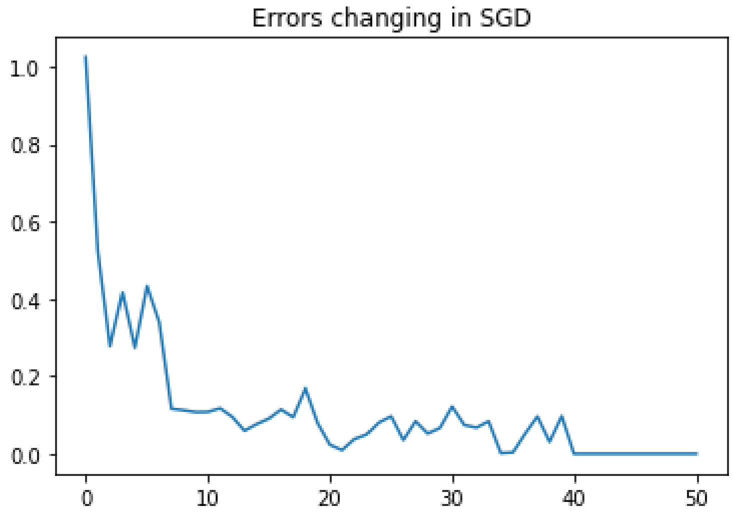
In [ ]:

```
d, N = X_train.shape
w0 = np.random.normal(0, 0.1, d+1)
batch_size = 15
n_epochs = 50
w, f_vals, grad_vals, err_vals = SGD(ell, grad_ell, w0, D, batch_size, n_epochs)
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_16980/718066766.py:4: RuntimeWarning: ov
erflow encountered in exp
  return (1 / (1 + np.exp(-z)))
```

In [ ]:

```
x_plot = np.arange(n_epochs+1)
plt.plot(x_plot, err_vals)
plt.title("Errors changing in SGD")
plt.show()
```



In [ ]:

```
def acc(app, Y, chosen_indeces):
    tot = 0
    for i in range(len(Y)):
        if (int(app[i]) == Y[i]):
            tot+=1
    return tot, tot/len(Y)

def predict(w, X, threshold = 0.5):
    d, N = X.shape
    app = np.zeros(N)
    for i in range(N):
        result = f(w, X[:, i])
        if (result >= threshold):
            app[i] = 1
        else:
            app[i] = 0
    return app
```

In [ ]:

```
Xt = X_train.copy()
Yt = Y_train.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Train Set:", int(tot))
print("Total entries on Train Set:", int(Yt.shape[0]))
print("Accuracy on Train Set:", round(avg*100, 2))

Xt = X_test.copy()
Yt = Y_test.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Test Set:", int(tot))
print("Total entries on Test Set:", int(Yt.shape[0]))
print("Accuracy on Test Set:", round(avg*100, 2))
```

```
Matches on Train Set: 5511
Total entries on Train Set: 5550
Accuracy on Train Set: 99.3
Matches on Test Set: 2751
Total entries on Test Set: 2775
Accuracy on Test Set: 99.14
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_16980/718066766.py:4: RuntimeWarning: ov
erflow encountered in exp
  return (1 / (1 + np.exp(-z)))
```

In [ ]:

```
def GD_2(l, grad_l, w0, D, tolf = 1e-9, tolx= 1e-9, kmax = 50, alpha = 1e-1):
    X, Y = D
    d, N = X.shape

    Xhat = np.concatenate((np.ones((1,N)), X), axis=0)

    w_vals = [w0]
    f_vals = [l(w0, Xhat, Y)]
    grad_f_vals = [grad_l(w0, Xhat, Y)]
    err_vals = [np.linalg.norm(grad_l(w0, Xhat, Y))]

    iterations = 0

    while iterations < kmax:
        w = w_vals[-1] - alpha * grad_l(w_vals[-1], Xhat, Y)

        w_vals.append(w)
        f_vals.append(l(w, Xhat, Y))
        grad_f_vals.append(grad_l(w, Xhat, Y))
        err_vals.append(np.linalg.norm(grad_l(w, Xhat, Y)))

        iterations+=1

        if err_vals[-1] < tolf * err_vals[0]:
            break

        if np.linalg.norm(w_vals[-1] - w_vals[-2]) < tolx * np.linalg.norm(w_vals[0]
            break

    return (w, f_vals, grad_f_vals, err_vals, iterations)
```

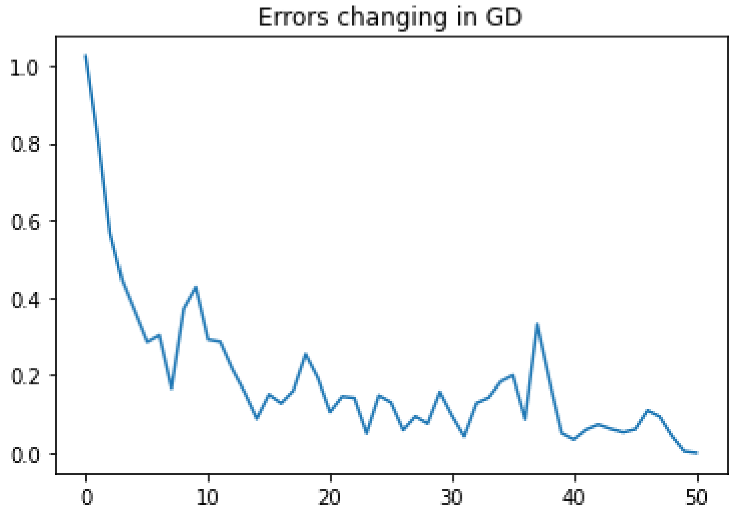
In [ ]:

```
w_gd, f_vals_gd, grad_vals_gd, err_vals_gd, iterations_gd = GD_2(ell, grad_ell, w0,
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_16980/718066766.py:4: RuntimeWarning: ov
erflow encountered in exp
  return (1 / (1 + np.exp(-z)))
```

In [ ]:

```
x_plot = np.arange(len(err_vals_gd))
plt.plot(x_plot, err_vals_gd)
plt.title("Errors changing in GD")
plt.show()
```



In [ ]:

```
Xt = X_train.copy()
Yt = Y_train.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w_gd, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Train Set:", int(tot))
print("Total entries on Train Set:", int(Yt.shape[0]))
print("Accuracy on Train Set:", round(avg*100, 2))

Xt = X_test.copy()
Yt = Y_test.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w_gd, Xthat)
```

```
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Test Set:", int(tot))
print("Total entries on Test Set:", int(Yt.shape[0]))
print("Accuracy on Test Set:", round(avg*100, 2))
```

```
Matches on Train Set: 5454
Total entries on Train Set: 5550
Accuracy on Train Set: 98.27
Matches on Test Set: 2714
Total entries on Test Set: 2775
Accuracy on Test Set: 97.8
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_16980/718066766.py:4: RuntimeWarning: ov
erflow encountered in exp
  return (1 / (1 + np.exp(-z)))
```

In [ ]:

```
X = data[:, 1:].T
Y = data[:, 0]

chosen_digits = [1, 5]
```

In [ ]:

```
X_set, Y_set = get_digits(X, Y, chosen_digits)
```

In [ ]:

```
d, N = X_set.shape
N_train = int(N/4*3)
Y_set[Y_set == chosen_digits[0]] = 0
Y_set[Y_set == chosen_digits[1]] = 1

X_train, X_test, Y_train, Y_test = x_split(X_set, Y_set, N_train)

D = (X_train, Y_train)
```

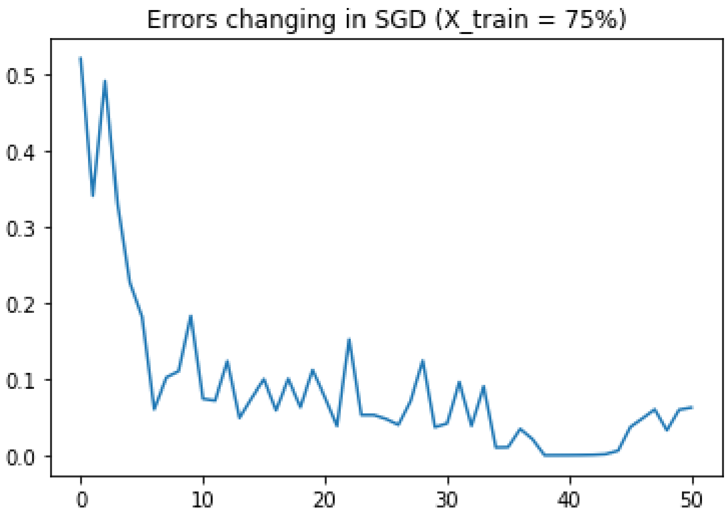
In [ ]:

```
d, N = X_train.shape
w0 = np.random.normal(0, 0.1, d+1)
batch_size = 15
n_epochs = 50
w, f_vals, grad_vals, err_vals = SGD(ell, grad_ell, w0, D, batch_size, n_epochs)
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_16980/718066766.py:4: RuntimeWarning: ov
erflow encountered in exp
  return (1 / (1 + np.exp(-z)))
```

In [ ]:

```
x_plot = np.arange(n_epochs+1)
plt.plot(x_plot, err_vals)
plt.title("Errors changing in SGD (X_train = 75%)")
plt.show()
```



In [ ]:

```
Xt = X_train.copy()
Yt = Y_train.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Train Set:", int(tot))
print("Total entries on Train Set:", int(Yt.shape[0]))
print("Accuracy on Train Set:", round(avg*100, 2))

Xt = X_test.copy()
Yt = Y_test.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Test Set:", int(tot))
print("Total entries on Test Set:", int(Yt.shape[0]))
print("Accuracy on Test Set:", round(avg*100, 2))
```

```
Matches on Train Set: 6274
Total entries on Train Set: 6359
Accuracy on Train Set: 98.66
Matches on Test Set: 2086
Total entries on Test Set: 2120
Accuracy on Test Set: 98.4
```

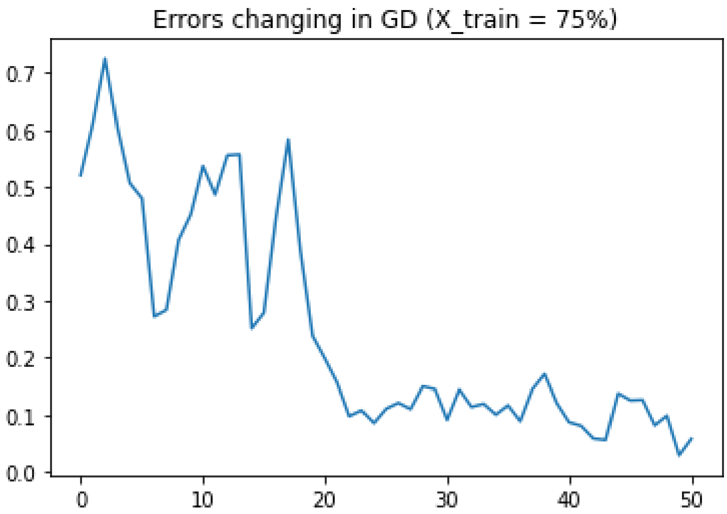
In [ ]:

```
w_gd, f_vals_gd, grad_vals_gd, err_vals_gd, iterations_gd = GD_2(ell, grad_ell, w0,
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_16980/718066766.py:4: RuntimeWarning: ov
erflow encountered in exp
  return (1 / (1 + np.exp(-z)))
```

In [ ]:

```
x_plot = np.arange(len(err_vals_gd))
plt.plot(x_plot, err_vals_gd)
plt.title("Errors changing in GD (X_train = 75%)")
plt.show()
```



In [ ]:

```
Xt = X_train.copy()
Yt = Y_train.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w_gd, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Train Set:", int(tot))
print("Total entries on Train Set:", int(Yt.shape[0]))
print("Accuracy on Train Set:", round(avg*100, 2))

Xt = X_test.copy()
Yt = Y_test.copy()

d, N = Xt.shape
Xthat = np.concatenate((np.ones((1,N)), Xt), axis=0)
app = predict(w_gd, Xthat)
tot, avg = acc(app, Yt, chosen_digits)
print("Matches on Test Set:", int(tot))
print("Total entries on Test Set:", int(Yt.shape[0]))
print("Accuracy on Test Set:", round(avg*100, 2))
```

```
Matches on Train Set: 6229
Total entries on Train Set: 6359
Accuracy on Train Set: 97.96
Matches on Test Set: 2070
Total entries on Test Set: 2120
Accuracy on Test Set: 97.64
```

Comparison with PCA and LDA

In [ ]:

```
import scipy.sparse.linalg
```

In [ ]:

```
def PCA(X, k):
    c_X = np.mean(X, axis = 1)
    c_X = np.reshape(c_X, (len(c_X), 1))

    X_c = X - c_X
    U, _, _ = np.linalg.svd(X_c, full_matrices = False)

    U_k = U[:, :k]

    Z_k = U_k.T @ X
    return Z_k, U_k.T
```

In [ ]:

```
def c_k(X, Y, k):
    I = (Y == k)
    tmp_X = X[:, I]
    return np.mean(tmp_X, axis = 1)
```

In [ ]:

```
Z_pca, P_pca = PCA(X_train, 2)
```

In [ ]:

```
def classify(Z, Y, P, x, chosen_numbers = chosen_digits):
    z = P @ x
    cs = [c_k(Z, Y, c) for c in chosen_numbers]
    ds = [np.linalg.norm((z - c)) for c in cs]

    idx = np.argmin(ds)
    return idx
```

In [ ]:

```
hitting_train = {"pca_train": 0}
for index, elem in enumerate(X_train.T):
    true_digit = Y_train[index]
    if classify(Z_pca, Y_train, P_pca, elem) == true_digit:
        hitting_train["pca_train"]+=1

hitting_test = {"pca_test": 0}
for index, elem in enumerate(X_test.T):
    true_digit = Y_test[index]
    if classify(Z_pca, Y_train, P_pca, elem) == true_digit:
        hitting_test["pca_test"]+=1
```

In [ ]:

```
accuracy_train = {"pca_train": 0}
for idx in accuracy_train.keys():
    accuracy_train[idx] = hitting_train[idx]/len(X_train.T)*100
accuracy_test = {"pca_test": 0}
for idx in accuracy_test.keys():
    accuracy_test[idx] = hitting_test[idx]/len(X_test.T)*100
```

In [ ]:

```
print(f"Hitting values on train are = {hitting_train}")
print(f"Accuracy values on train are = {accuracy_train}")
```

```
Hitting values on train are = {'pca_train': 2837}
Accuracy values on train are = {'pca_train': 44.61393300833464}
```

In [ ]:

```
print(f"Hitting values on test are = {hitting_test}")
print(f"Accuracy values on test are = {accuracy_test}")
```

```
Hitting values on test are = {'pca_test': 958}
Accuracy values on test are = {'pca_test': 45.18867924528302}
```