CS7DS2 – Week 4 Assignment

Ayush Gupta – 21355081

Functions:
The functions used in the assignment are as follows:

$$f_1(x,y) = (x-1)^4 + 8(y-1)^2$$
$$f_2(x,y) = \max(x-1,0) + 8 \cdot |y-1|$$

Using SymPy, initializing x and y, define equations based on $f_1(x,y)$ and $f_2(x,y)$ given above and using the $diff()$ function to get the partial derivatives of the two functions.

$$\frac{\partial f_1}{\partial x} = 4(x-1)^3 \, , \frac{\partial f_1}{\partial y} = 16y - 16$$
$$\frac{\partial f_2}{\partial x} = \theta(x-1) \, , \frac{\partial f_2}{\partial y} = 8 \cdot sign(y-1)$$

Question (a)

(i) Polyak Step Size

In each iteration, a constant step is calculated by taking the value of the function for a given parameter value and dividing it by the sum of sum of the squares of the gradients(partial derivatives) of each parameter. A very small ϵ is added to the denominator to prevent division by zero.

```
for _ in range(150):
        count = 0
        for i in range(n):
            count = sum(df[i](x[i])**2)
        step = f(*x) / (count + epsilon)
        for i in range(n):
            x[i] -= step * df[i](x[i])
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(step)
```

(ii) RMSProp

In RMSProp, two empty arrays are initialised, one containing the step sizes and other containing the historical gradient sum for each parameter. Initial step size is set to α_0. After each iteration, parameter values are first subtracted by the product of initial step length and the parameter gradient. The squared beta parameter is updated by adding the current square of the parametric gradient to the sum. Step size gets finally updated by dividing $\alpha_0$by the √(sum of current squared gradients). As seen in (i), an ϵ is added to the denominator.

```
for _ in range(iterations):
        for i in range(n):
            x[i] -= alphas[i] * df[i](x[i])
            sums[i] = (beta * sums[i]) + ((1 - beta) * (df[i](x[i]) ** 2))
            alphas[i] = alpha0 / ((sums[i] ** 0.5) + epsilon)
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(deepcopy(alphas))
```

(iii) Heavy Ball

In Heavy Ball, prior to running any iterations, the value of z is set to 0. z represents a historical sum of square gradients. At each iteration as seen in RMSProp(ii), z is updated with β, controlling the weight of the historical sum and α parameter controlling the weight of the current sum. Each parameter is reduced by the product of the new step size, z, and its gradient.

```
for _ in range(iterations):
        z = (beta * z) + (α * f(*x) / (sum(df[j](x[j]) ** 2 for j in range(n)) + epsilon))
        for i in range(n):
            x[i] -= z * df[i](x[i])
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(z)
```

(iv) Adam

Adam is equivalent to the combination of RMSProp and Heavy Ball Algorithms. It initiates two arrays which contain the step size and the historical squared gradients for each parameter. The iteration counter, t, is set to 0 and increased by 1 in each iteration. β_1 determines the weight of the regular gradient while β_2 determines the weight of the square gradient and are added as seen in (ii). These sums are scaled by dividing them by $1 - \beta_1^t$. Finally each parameter is decreased by the product of α and scaled regular gradient sum divided by the square root of the scaled square gradient sum. An additional ϵ is added to the denominator.

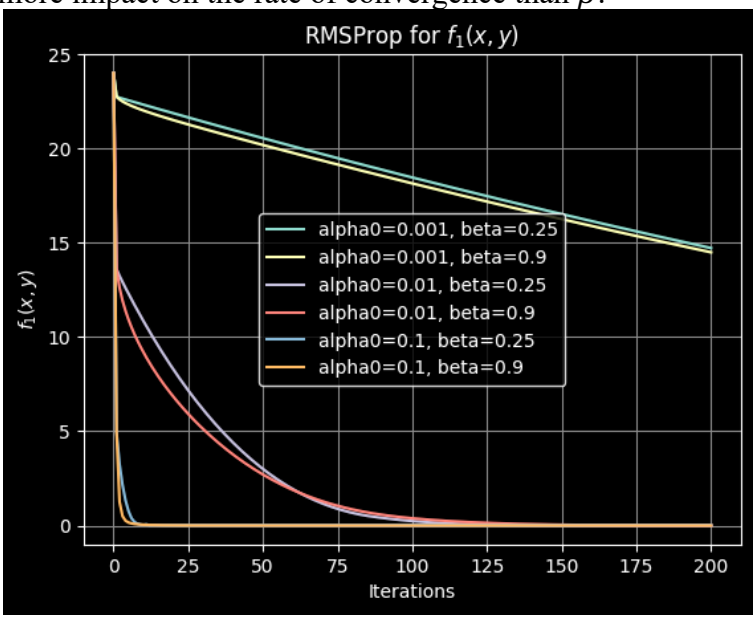
```
for _ in range(iterations):
        t += 1
        for i in range(n):
            ms[i] = (beta1 * ms[i]) + ((1 - beta1) * df[i](x[i]))
            vs[i] = (beta2 * vs[i]) + ((1 - beta2) * (df[i](x[i]) ** 2))
            m_hat = ms[i] / (1 - (beta1 ** t))
            v_hat = vs[i] / (1 - (beta2 ** t))
            step[i] = alpha * (m_hat / ((v_hat ** 0.5) + epsilon))
            x[i] -= step[i]
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(deepcopy(step))
```

Question (b)

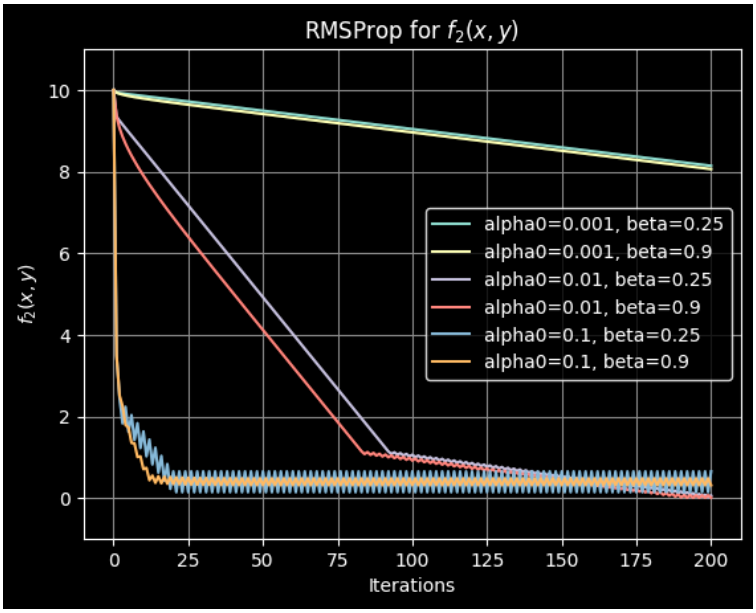
(i) RMSProp

In this exercise, parameter values of $\alpha_0 \in \{0.001, 0.01, 0.1\}$ and $\beta \in \{0.25, 0.9\}$ are investigated with $x_0 = 3$, $y_0 = 0$ and 200 iterations of the algorithm being run.

For the function 1, It can be observed that the function converges at around 100 iterations when $\alpha_0 = 0.01$ which appears to be the optimal learning rate. Additionally, it can be observed that for value of $\alpha_0 = 0.001$ the convergence rate is very slow and is not reaching convergence even after 200 iterations. It appears that α_0 has more impact on the rate of convergence than β.



For the 2nd function, when α_0= 0.1, the function converges very quickly, converging for β = 0.9 before β= 0.25, continuing to oscillate slightly above the minimum indefinitely. When $\alpha_0 = 0.001$ the function doesn't converge at all. And finally when $\alpha_0 = 0.01$ the function converges and reaches the minimum at around 150 iterations, reducing all the way until the 200th iteration. Again, the effect of α_0 is much greater than β in these curves.

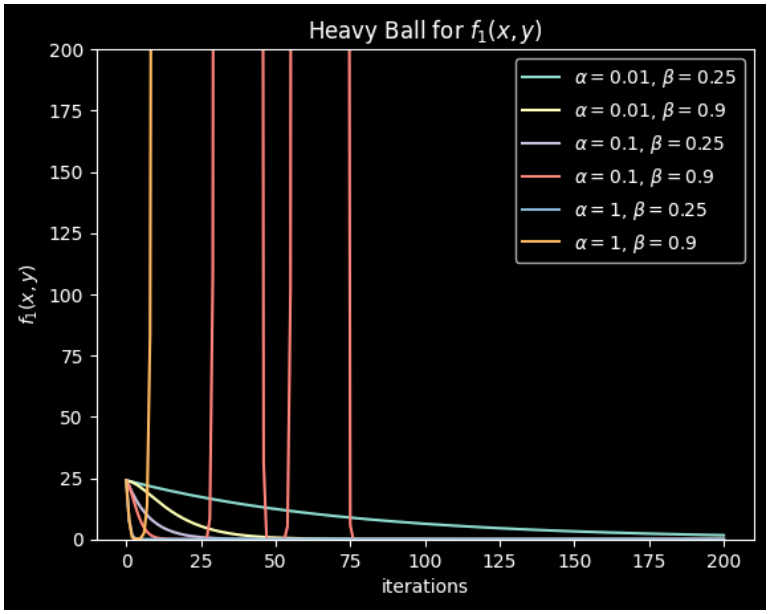


Hence, for both the functions, the parameter of $\beta = 0.9$ and $\alpha_0 = 0.01$ produce the best results.

(ii) Heavy Ball

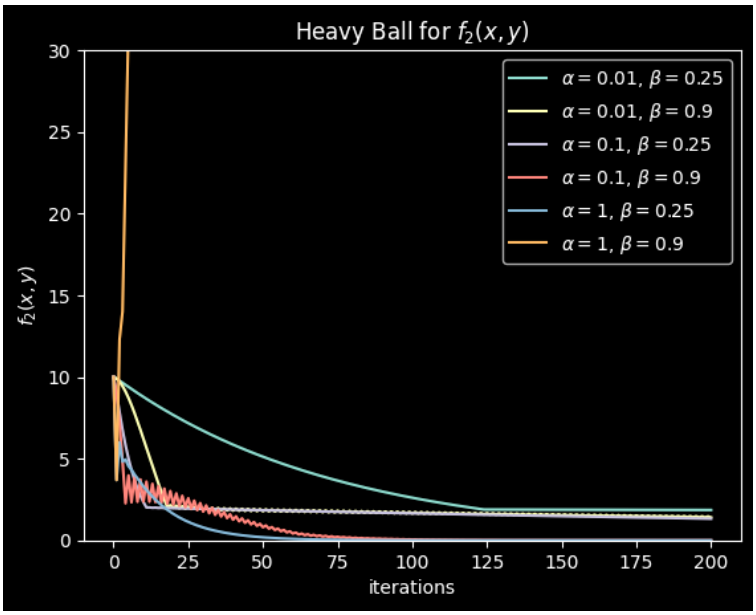
This exercises use the values of $\alpha_0 \in \{0.01, 0.1, 1\}$ and $\beta \in \{0.25, 0.9\}$ along with $x_0 = 3$ and $y_0 = 0$ and the algorithm is run for 200 iterations

For the first function, we observe that for the same value of $\alpha = 0.01$ and different values of $\beta = 0.25$ and $\beta = 0.90$ it can be observed that the function converges too slowly for $\beta = 0.25$ and reaches convergence at iteration 50 for $\beta = 0.90$. When $\alpha = 0.1$, the function converges to the minimum very quickly but it can be seen when beta $= 0.9$ the function starts to diverge for some iterations and return to the minimum. When $\alpha = 1$ both the functions converge quickly but again it can be observed that for beta $= 0.9$, the function diverges again and seems to not converge in the 200 iterations.



For the Second function, when α = 0.01 and beta = 0.25, the function converges very slowly and doesn't reach the minimum and when beta = 0.9 the function converges quickly at the value of 20

iterations. When α = 0.1 and beta = 0.25 the function converges on the minimum at around 15 iterations but when beta = 0.9 the function takes almost 90 iterations to converge. It can be observed that for α = 0.1 and beta = 0.9 the function oscillates before reaching its minima. When α = 1 and beta = 0.9 the function diverges very quickly and when beta = 0.25 the function converges very quickly. Similar trend is observed in function 1 for the value of α = 1.

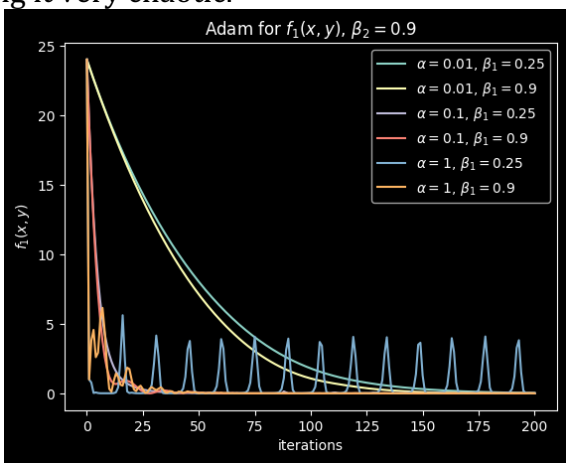


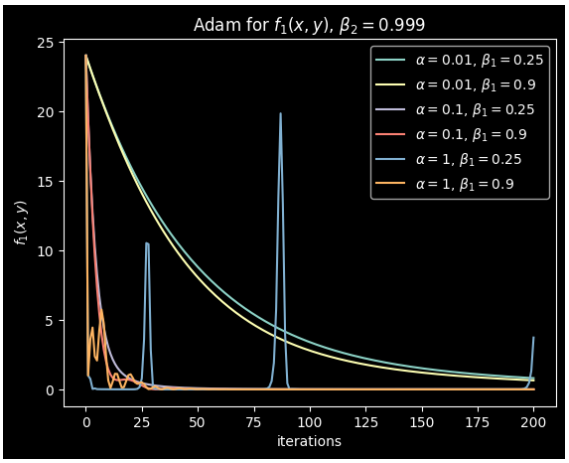
It can be observed that the parameter of α_0= 1 and beta = 0.25 creates very good results, converging very quickly and performing better than RMSProp Algorithm

(iii) Adam

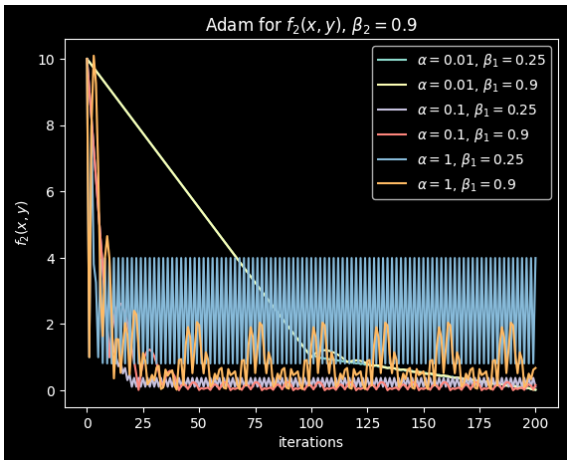
This exercises use the values of $\alpha_0 \in \{0.01, 0.1, 1\}$, $\beta_1 \in \{0.25, 0.9\}$ and $\beta_2 \in \{0.9, 0.999\}$ along with $x_0 = 3$ and $y_0 = 0$ and the algorithm is run for 200 iterations

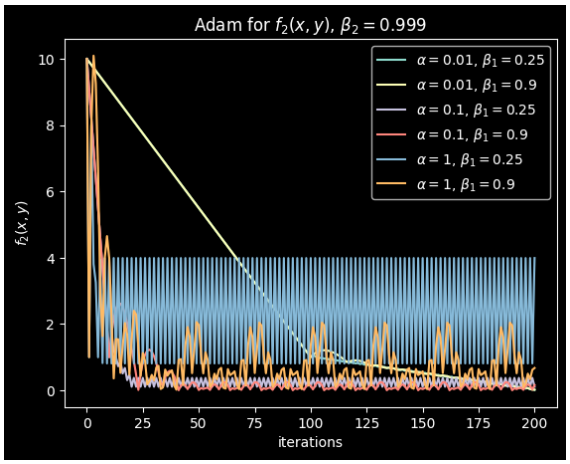
For the first function, we observe that for the same value of α = 0.01, β_1 and β_2 take long to converge and $\beta_2 = 0.999$ not converging even in 200 iterations while $\beta_2 = 0.9$ barely reaches the minimum value in this time. When α_0= 0.1, in all the cases the function reaches the minimum in less than 50 iterations, where $\beta_1 = 0.9$ reaches the minimum earliest. When α= 1 and β_1= 0.9 the function converges quickly with a little bit of erratic movement. When $\beta_2 = 0.999$ for α_0= 1, the function converges quickly but diverges a couple of times at 25 iterations and 80 iterations before returning to the minimum again. For $\beta_2 = 0.9$, the function diverges every 20 iterations, making it very chaotic.





For the second function, the results of $\beta_2 = 0.999$ and $\beta_2 = 0.9$ are very similar. When α= 0.01 the function converges very slowly to the minimum. When α_0= 1, the function begins to oscillate indefinitely. When α_0= 0.1 the function converges to the minimum value very quickly and then begins to oscillate for both values of β_1. The probable cause of the oscillation found in the second function can be attributed to the nature of the $sign$ function making result oscillate to get the minimum when $\alpha_0$is large.



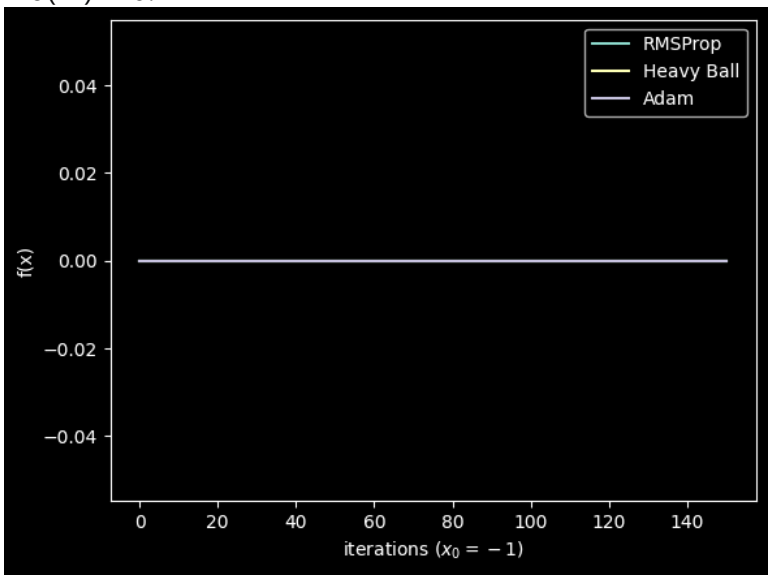


The optimal parameters for the first function can be $\alpha_0 = 1$, $\beta_1 = 0.25$ and $\beta_2 = 0.999$ cautiously since the function seems to diverge at 2 different points and can lead to some discrepancies. No optimal parameters for the second function prevail due to the oscillating nature of all the parameter's convergence.

Question (c)

For this part of the question, the following parameters from question(b) were chosen and run for 150 iterations: $\alpha_0 = 0.01$, $\beta_1 = 0.9$ for RMSProp; $\alpha_0 = 1$, $\beta_1 = 0.25$ for Heavy Ball; $\alpha_0 = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ for Adam.

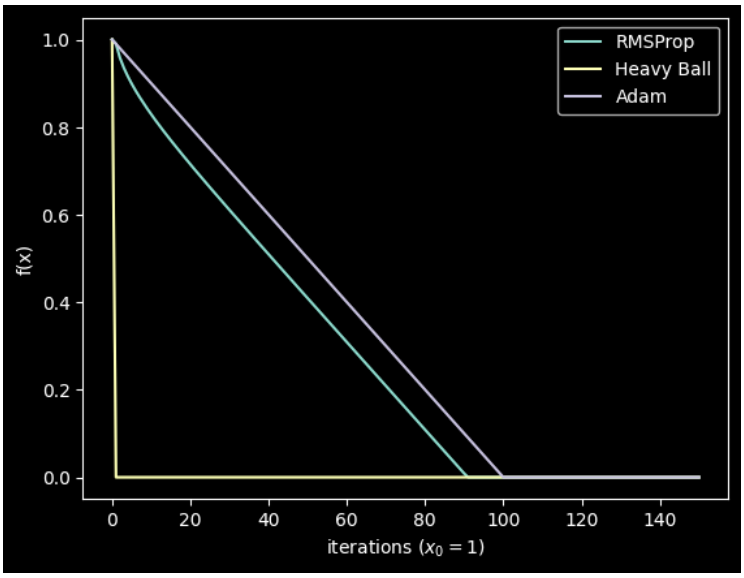
(i) When $x_0 = -1$ all algorithms converge on the minimum which is 0, in a single iteration. This is due to all values of the function $\max(-1,0) = 0$ is already minimum and value of the gradient θ(-1) = 0.



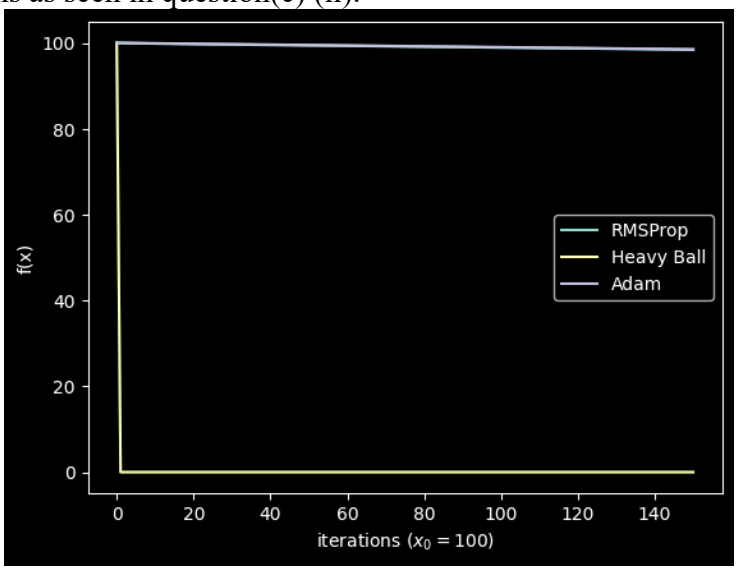
(ii) When $x_0 = +1$ all algorithms converge on the minimum. Heavy ball converges immediately in 1-2 iterations, RMSProp in about 90 iterations and finally Adam in about 100 iterations.

For Heavy Ball, the value of z is $\frac{1}{1+\epsilon}$. This value is then multiplied by the gradient θ(1) = 1 and decreased from x_0. Since epsilon is infinitesimally small, the function evaluates to z = 1 and hence result = 0 after a single iteration.

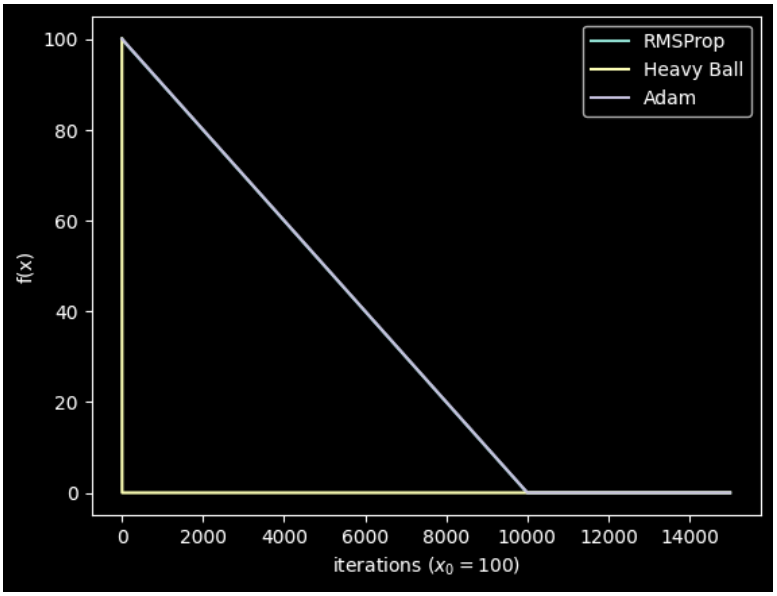
It can be observed that RMSProp and Adam both converge at a constant rate. This can be attributed to the gradient, $\theta(x)$, remaining constant when x>0. RMSProp converges quicker in the beginning due to RMSProp starting with a custom α_0.



(iii) When $x_0 = +100$ only the Heavy Ball Algorithm converges in very few iterations for the same reasons as seen in question(c) (ii).



For RMSProp and Adam in this example, the change in value of x has no effect on the rate due to value of $f(x)$ never being present in the step size calculation. Since, only the gradient has an effect on the rate of convergence and it remaining constant for x>0. It can be observed from the below curve that RMSProp and Adam converge after about 10,000 iterations.



Appendix :

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy

def calculate_derivatives():
    x, y = sp.symbols('x y', real=True)
    # The First Function

    f1 = (1 * ((x - 1) ** 4)) + (8 * ((y - 1) ** 2))
    df1_x = sp.diff(f1, x)
    df1_y = sp.diff(f1, y)
    print(f1)
    print(df1_x)
    print(df1_y)
    # The second Function

    f2 = sp.Max(x - 1, 0) + (8 * sp.Abs(y - 1))
    df2_x = sp.diff(f2, x)
    df2_y = sp.diff(f2, y)
    print(f2)
    print(df2_x)
    print(df2_y)

def polyak(df,x0,f,epsilon = 1e-8):
    # standard code
    x = deepcopy(x0)
    n = len(df)
    x_list, f_list, step_list = [deepcopy(x)], [f(*x)],[]

    #report
    for _ in range(150):
        count = 0
        for i in range(n):
            count = sum(df[i](x[i])**2)
        step = f(*x) / (count + epsilon)
        for i in range(n):
            x[i] -= step * df[i](x[i])
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(step)
    return x_list, f_list, step_list

def RMSprop(f, df, x0, parameters, iterations=100):
    alpha0, beta = parameters
    x = deepcopy(x0)
    n = len(df)
    x_list, f_list ,step_list = [deepcopy(x)], [f(*x)], [[alpha0] * n]
    # defining the constant parameter of epsilon to account for division by zero
    epsilon = 1e-8
    sums = [0] * n
    alphas = [alpha0] * n
    for _ in range(iterations):
```

```
        for i in range(n):
            x[i] -= alphas[i] * df[i](x[i])
            sums[i] = (beta * sums[i]) + ((1 - beta) * (df[i](x[i]) ** 2))
            alphas[i] = alpha0 / ((sums[i] ** 0.5) + epsilon)
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(deepcopy(alphas))
    return x_list, f_list ,step_list

def HeavyBall(f, df, x0, parameters, iterations=100):
    alpha, beta = parameters
    x = deepcopy(x0)
    n = len(df)
    x_list, f_list, step_list = [deepcopy(x)], [f(*x)], [0]
    # defining the constant parameter of epsilon to account for division by zero
    epsilon = 1e-8
    z = 0
    for _ in range(iterations):

        z = (beta * z) + (alpha * f(*x) / (sum(df[j](x[j]) ** 2 for j in range(n)) +
epsilon))
        for i in range(n):
            x[i] -= z * df[i](x[i])
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(z)
    return x_list, f_list, step_list

def Adam(f,df,x0,parameters,iterations=100):
    alpha,beta1,beta2 = parameters
    x = deepcopy(x0)
    n = len(df)
    x_list, f_list, step_list = [deepcopy(x)], [f(*x)], [[0] * n]
    # defining the constant parameter of epsilon to account for division by zero
    epsilon = 1e-8
    ms = [0] * n
    vs = [0] * n
    step = [0] * n
    t = 0
    for _ in range(iterations):
        t += 1
        for i in range(n):
            ms[i] = (beta1 * ms[i]) + ((1 - beta1) * df[i](x[i]))
            vs[i] = (beta2 * vs[i]) + ((1 - beta2) * (df[i](x[i]) ** 2))
            m_hat = ms[i] / (1 - (beta1 ** t))
            v_hat = vs[i] / (1 - (beta2 ** t))
            step[i] = alpha * (m_hat / ((v_hat ** 0.5) + epsilon))
            x[i] -= step[i]
        x_list.append(deepcopy(x))
        f_list.append(f(*x))
        step_list.append(deepcopy(step))
    return x_list, f_list, step_list

def B1(f, df, x, fnum):
    alpha0s = [0.001, 0.01, 0.1]
    betas = [0.25, 0.9]
    iterations = 200
    iters = list(range(iterations + 1))
```

```
    plt.figure(facecolor='black')
    plt.xlabel('Iterations', color='white')
    plt.ylabel(f'$f_{fnum}(x,y)$', color='white')
    plt.title(f'RMSProp for $f_{fnum}(x,y)$', color='white')
    plt.grid(color='gray')
    plt.tick_params(axis='x', colors='white')
    plt.tick_params(axis='y', colors='white')

    all_values = []

    for alpha0 in alpha0s:
        for beta in betas:
            xs, values, steps = RMSprop(f, df, x, [alpha0, beta], iterations=iterations)
            plt.plot(iters, values, label=f'alpha0={alpha0}, beta={beta}')
            all_values.extend(values)


    plt.ylim([min(all_values) - 1, max(all_values) + 1])


    plt.legend(facecolor='black', edgecolor='white', fontsize='medium')

    plt.show()

def B2(f, df, x, fnum):
    alphas = [0.01,0.1,1]
    betas = [0.25,0.9]
    iterations = 200
    iters = list(range(iterations + 1))
    legend = []
    contour_data = []
    for alpha in alphas:
        for beta in betas:
            xs, values, steps = HeavyBall(f, df, x, [alpha, beta], iterations=iterations)
            legend.append(f'$\\alpha={alpha},\\, \\beta={beta}$')
            print(f'alpha={alpha}, beta={beta}: final_value={values[-1]}')
            plt.plot(iters, values)
    plt.xlabel('iterations')
    plt.ylabel(f'$f_{fnum}(x,y)$')
    plt.title(f'Heavy Ball for $f_{fnum}(x,y)$')
    if fnum == 1: plt.ylim([0, 200])
    else: plt.ylim([0, 30])
    plt.legend(legend)
    plt.show()

def B3(f, df, x, fnum):
    alphas = [0.01, 0.1, 1]
    beta1s = [0.25, 0.9]
    beta2s = [0.9, 0.999]
    iterations = 200
    iters = list(range(iterations + 1))
    legend = []
    for beta2 in beta2s:
        for alpha in alphas:
            for beta1 in beta1s:
                xs, values, steps = Adam(f, df, x, [alpha, beta1, beta2],
iterations=iterations)
                legend.append(f'$\\alpha={alpha},\\, \\beta_1={beta1}$')
```

```
                print(f'alpha={alpha}, beta1={beta1}, beta2={beta2}: final_value={values[-
1]}')
                plt.figure(1)
                plt.plot(iters, values)
                stepsx = [step[0] for step in steps]
                stepsy = [step[1] for step in steps]
                plt.figure(2)
                plt.plot(iters, stepsx)
                plt.figure(3)
                plt.plot(iters, stepsy)
        plt.figure(1)
        plt.xlabel('iterations')
        plt.ylabel(f'$f_{fnum}(x,y)$')
        plt.title(f'Adam for $f_{fnum}(x,y),\\, \\beta_2={beta2}$')
        plt.legend(legend)
        plt.figure(2)
        plt.xlabel('iterations')
        plt.ylabel('step $x$')
        plt.title(f'Step size of $x$ for $f_{fnum},\\, \\beta_2={beta2}$')
        plt.figure(3)
        plt.xlabel('iterations')
        plt.ylabel('step $y$')
        plt.title(f'Step size of $y$ for $f_{fnum},\\, \\beta_2={beta2}$')
        plt.show()

def Question_B():
    f1 = lambda x, y: 1*(x - 1)**4 + 8*(y - 1)**2
    df1_x = lambda x: 4*(x - 1)**3
    df1_y = lambda y: 16*y - 16
    f2 = lambda x, y: 8*abs(y - 1) + max(0, x - 1)
    df2_x = lambda x: np.heaviside(x - 1, 0)
    df2_y = lambda y: 8*np.sign(y - 1)
    print('(b)(i) f1')
    B1(f1, [df1_x, df1_y], [3, 0], 1)
    print('(b)(i) f2')
    B1(f2, [df2_x, df2_y], [3, 0], 2)
    print('(b)(ii) f1')
    B2(f1, [df1_x, df1_y], [3, 0], 1)
    print('(b)(ii) f2')
    B2(f2, [df2_x, df2_y], [3, 0], 2)
    print('(b)(iii) f1')
    B3(f1, [df1_x, df1_y], [3, 0], 1)
    print('(b)(iii) f2')
    B3(f2, [df2_x, df2_y], [3, 0], 2)


def Question_C():
    f = lambda x: max(x, 0)
    df = lambda x: np.heaviside(x, 0)
    num_iters = 150
    #can be changed to 15000 for last answer
    iters = list(range(num_iters + 1))
    for x0 in [-1, 1, 100]:
        _, values, _ = RMSprop(f, [df], [x0], [0.01, 0.9], iterations=num_iters)
        print(f'RMSProp (x0={x0}): {values[-1]}')
        plt.plot(iters, values)
        _, values, _ = HeavyBall(f, [df], [x0], [1, 0.25], iterations=num_iters)
        print(f'Heavy Ball (x0={x0}): {values[-1]}')
        plt.plot(iters, values)
```

```
        _, values, _ = Adam(f, [df], [x0], [0.01, 0.9, 0.999], iterations=num_iters)
        print(f'Adam (x0={x0}): {values[-1]}')
        plt.plot(iters, values)
        plt.xlabel(f'iterations ($x_0={x0}$)')
        plt.ylabel('f(x)')
        plt.legend(['RMSProp', 'Heavy Ball', 'Adam'])
        plt.show()


calculate_derivatives()
Question_B()
Question_C()
```