Xiangyu Zheng 21331025 CS7CS2 Optimisation for Machine Learning week4 assignment

(a) (i) Polyak step size

According to the function of Polyak step size α: $\alpha = \frac{\mathrm{f(x)} - f^*}{\nabla f(x)^T \nabla f(x)}$

1. calculate the numerator, suppose f* = 0, use the function lambdify from sympy, If it is a unary function, get the value of f(x). If it is a binary function, get the value of f(x,y).

```
numerator = getF1(x) - 0
numerator = getF2(x, y) - 0x
```

2. calculate the denominator. It is the squared addition of the derivatives of each parameter.

$$\nabla f(x)^T \nabla f(x) = \sum_{i=1}^{n} \frac{af}{ax_i}(x)^2$$

Here we take the unary function and the binary function as an example. Add ϵ to prevent the denominator from being 0. Then calculate the step size α:

```
alpha = numerator / (getDf1(x) ** 2 + epsilon)
```

binary function, Use lambdify in sympy to bring in x and y to get the partial derivatives of x and y:

```
dx, dy = getDf2(x, y)
alpha = numerator / (dx ** 2 + dy ** 2 + epsilon)
```

3. Use the calculated step size to calculate the descending step size alpha of x and y by multiplying the partial derivative of x and the partial derivative of y respectively. If it is a unary function, only the derivative of x is calculated

```
x = x - alpha * dx
y = y - alpha * dy
```

4. Record the number of iterations i, and stop the iteration when the upper limit is reached, otherwise repeat the above steps.

```
i = 0
while i < maxIters:
```

(ii) RMSProp

1. Calculate the iteration step size according to the formula, and pass in a fixed α0 value and β, where the numerator is α0 and the denominator is the square of the partial derivative value with weight β. The denominator is increased by ϵ to prevent 0.

$$\alpha_t = \frac{\alpha_0}{\sqrt{(1-\beta)\beta^t \frac{df}{dx}(x_0)^2 + (1-\beta)\beta^{t-1} \frac{df}{dx}(x_1)^2 + \cdots + (1-\beta)\frac{df}{dx}(x_{t-1})^2 + \epsilon}}$$

```
sum = Beta * sum + (1 - Beta) * (dx ** 2)
```

If the multivariate function calculates the sum of squares of the weighted partial derivatives respectively, here is the second parameter y of the binary function as an example.

```
sum2 = Beta * sum2 + (1 - Beta) * (dy ** 2)
```

2.Then calculate the iteration step alpha separately:

```
alpha = alpha0 / (math.sqrt(sum) + epsilon)
alpha2 = alpha0 / (math.sqrt(sum2) + epsilon)
```

3. Calculate the new points separately, use the old point minus the step size alpha and multiply the partial derivative value

```
x = x - alpha * dx
y = y - alpha2 * dy
```

4. Use i to record the number of iterations, and stop repeating the above steps when the upper limit is reached

```
i = 0
while i < maxIters:
```

(iii) Heavy Ball

1.Pass in β to set the weight for the sum of the past partial derivatives, and pass in α to set the weight for the current partial derivatives. Add the two to get a new step size:

```
z = Beta * z + alpha * dx
```

If it is a multivariate function, the step size of the parameter is calculated separately. Here is an example of the second parameter of the binary function:

```
z2 = Beta * z2 + alpha * dy
```

2.Calculate a new point based on the new step size obtained:

```
x = x - z
y = y - z2
```

3. Record the current iteration number reaches the upper limit and stop the iteration

```
i = 0
while i < maxIters:
```

(iv) Adam

1.Use the lambdify function in sympy to get the partial derivatives of x and y

```
dx, dy = getDf2(x, y)
```

Calculate m and v respectively according to the formula and incoming β1 and β2, where m is the sum of the weighted partial derivatives. v is the sum of the squares of the weighted partial derivatives:

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla f(x_t)$$

$$v_{t+1} = \beta_2 v_t + (1-\beta_2)[\frac{\partial f}{\partial x_1}(x_t)^2, \frac{\partial f}{\partial x_2}(x_t)^2, \ldots\ldots, \frac{\partial f}{\partial x_n}(x_t)^2]$$

If it is a multivariate function, the m and v of the respective parameters are calculated respectively, here is the binary function x and y as an example:

```
mt = Beta1 * mt + (1 - Beta1) * dx
mt2 = Beta1 * mt2 + (1 - Beta1) * dy
vt = Beta2 * vt + (1 - Beta2) * (dx ** 2)
vt2 = Beta2 * vt2 + (1 - Beta2) * (dy ** 2)
```

2. Calculate $\widehat{m}$ and $\hat{v}$ according to the formula, if it is a multivariate function, calculate the parameters respectively:

$$\widehat{m} = \frac{m_{t+1}}{(1-\beta_1^t)}, \hat{v} = \frac{v_{t+1}}{(1-\beta_2^t)}$$

```
me = mt / (1 - Beta1 ** (i + 1))
me2 = mt2 / (1 - Beta1 ** (i + 1))
ve = vt / (1 - Beta2 ** (i + 1))
ve2 = vt2 / (1 - Beta2 ** (i + 1))
```

3. Get a new point for each parameter according to a and the step size in the formula:

$$x_{t+1} = x_t - \alpha[\frac{\widehat{m_1}}{\sqrt{\widehat{v_1}}-\epsilon}, \frac{\widehat{m_2}}{\sqrt{\widehat{v_2}}-\epsilon}, \ldots\ldots, \frac{\widehat{m_n}}{\sqrt{\widehat{v_n}}-\epsilon}]$$

```
x = x - alpha * (me / (math.sqrt(ve) + epsilon))
y = y - alpha * (me2 / (math.sqrt(ve2) + epsilon))
```

4. Repeat the above steps until the end of the iteration

```
i = 0
while i < maxIters:
```

(b)

Here are the two equations I downloaded:

Function1: 1*(x-5)^4+3*(y-0)^2

Function2: Max(x-5,0)+3*|y-0|

(i) α and β in RMSProp.

Function1 1*(x-5)^4+3*(y-0)^2

All experiments are performed at the starting point x=1 and y=1

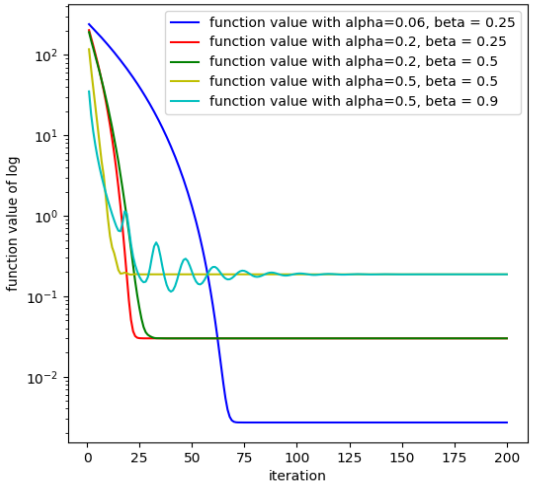

Figure 1. The log of the function value varies with different alpha and beta values

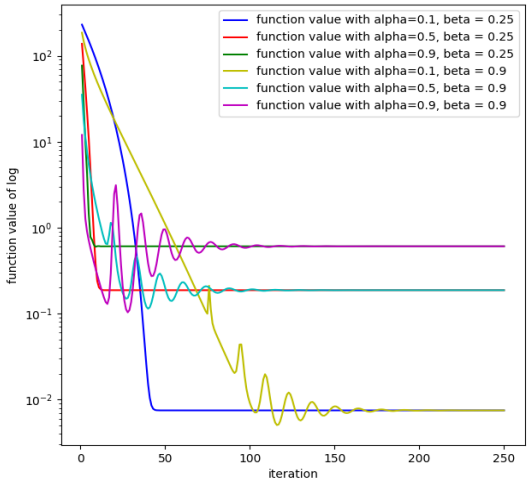

figure 2 The log of the function value varies with different alpha and beta =0.25 and beta=0.9

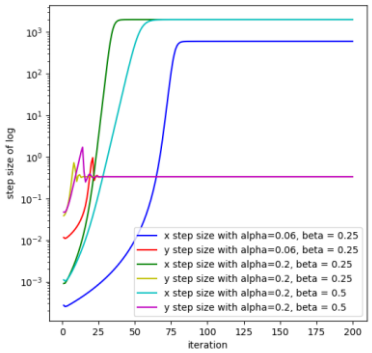

figure 3 The log value of the step size of x and y changes with different alpha and beta values

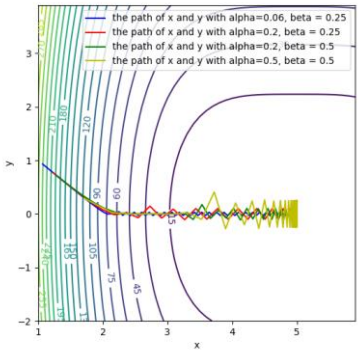

figure 4 The descending paths of the functions x and y vary with different x and y values

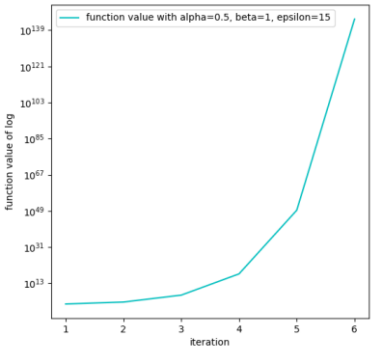

figure 5 The case where the function value diverges

We can see from figure 1 and figure 2 that as the alpha gets smaller and smaller, the function value approaches 0 and stops converging, but it takes more iterations to reach the minimum value as the beta value increases, the function decreases faster. This is because, according to the formula in a(ii), and function1.alpha as the numerator, the larger the value, the larger the step size and the larger the step, so the iteration is faster. The larger the step size, the larger the oscillation at a certain place, so a larger value is maintained. This is why the larger the alpha value, the larger the value of the equation is maintained. As for the beta value, if the beta value is large, then although the cumulative new value is very slow, the relative start and end sum value will be smaller, but it will help to maintain the sum value, and the increase/decrease will be very slow. The sum value is used as the denominator, the smaller the iteration, the faster the iteration, and the slower increase/decrease speed will lead to slower subsequent iterations. This results in the same alpha value in figure1 and figure2. The beta value is large, and the iteration is faster than the small beta value at the beginning, and then becomes slower. At the same time, it can be seen from figure 3 that the smaller the alpha value, the smaller the step size of x remains stable, while the y remains stable, which results in a smaller function value. The larger the alpha value in figure4, the larger the oscillation of the y value, which means that the function value is maintained at a large level. This is because the alpha value is relatively large, the step size is correspondingly large, and the algorithm maintains a relatively large oscillations at a certain time. figure 5 illustrates a divergent situation.

Function2:Max(x-5,0)+3*|y-0|

All experiments are performed at the starting point x=10 and y=1

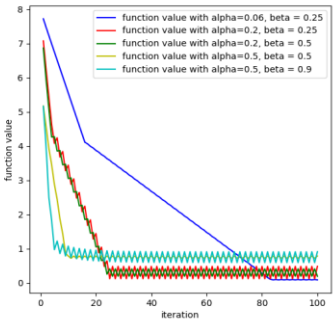

figure 6 Function value varies with different alpha and beta

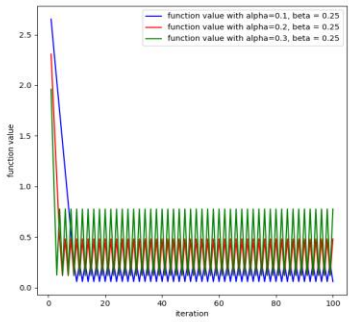

figure 7 Function value varies with different alpha and beta=0.25

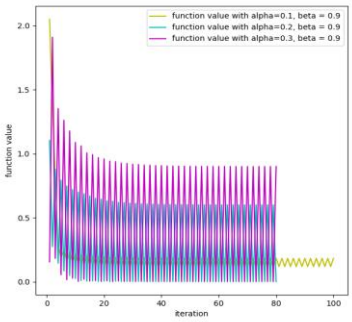

figure 8 Function value varies with different alpha and beta=0.9

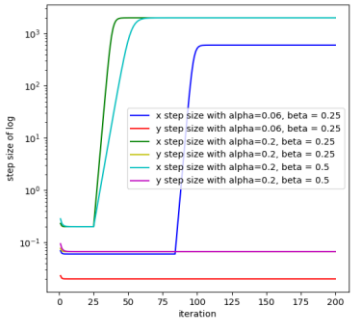

figure 9 The step size of x and y varies with different alpha and beta

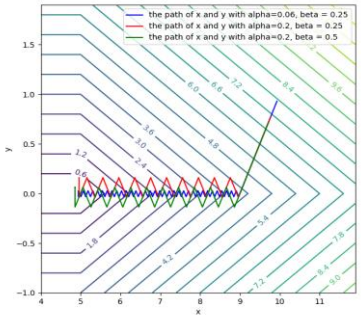

figure 10 Descending path of x and y

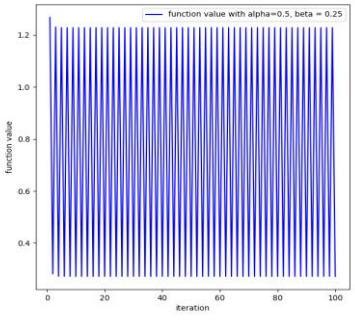

figure 11 The function has been fluctuating at alpha=0.5 and beta0.25 and cannot converge

We can see from figures 6, 7, and 8 that with the increase of alpha, the minimum value of the function will become larger and the falling speed will become faster, but it will increase the fluctuation of the function. Increasing the beta value helps reduce volatility in the case of small alpha values and increases it in the case of large alpha values. figure10 is a good illustration of the above situation. figure11 shows that when alpha=0.5, beta=0.25, the function fluctuates between 0.4-1.2 and fails to converge. figure9 shows the step size of x and y, y keeps almost constant value, x increases as alpha increases, which shows that the function value decreases as alpha increases. The reason of function2 is roughly the same as that of function1, but the partial derivatives in function2 are all fixed values, so the oscillation is relatively more.

(ii) α and β in Heavy Ball.

Function1: 1*(x-5)^4+3*(y-0)^2

All experiments are performed at the starting point x=1 and y=1

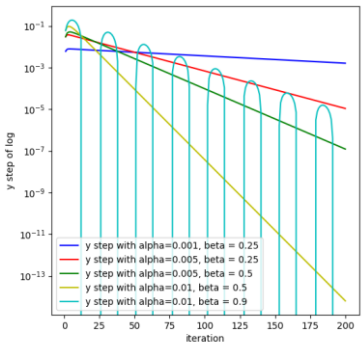

figure 12 y step varies with different alpha and beta

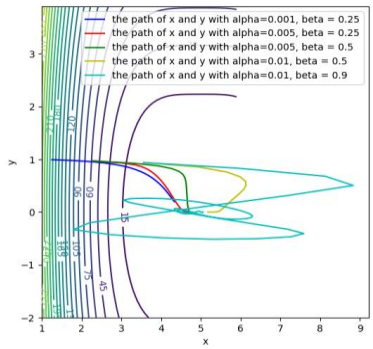

figure 13 Descending paths for different x and y

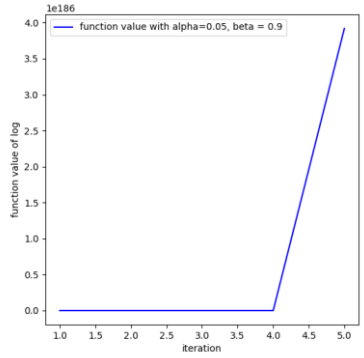

figure 14 a divergent situation

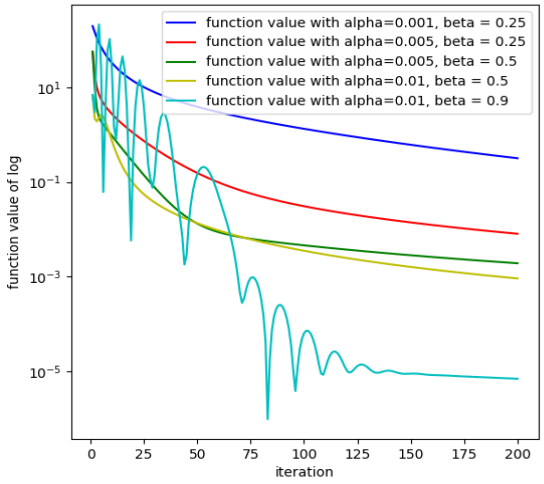

figure 15 Function value varies with different alpha and beta

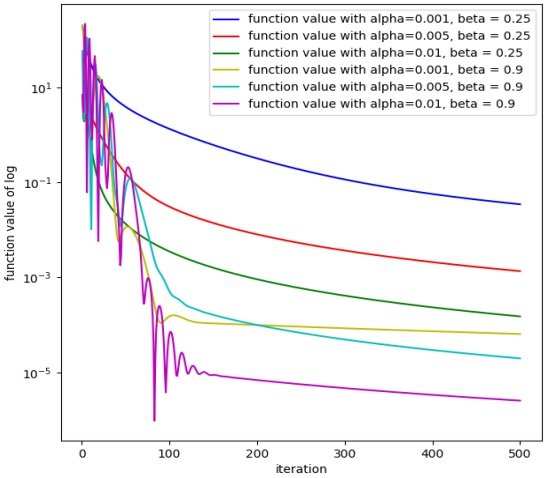

Figure 16 Function value with different alpha and beta = 0.25 and beta=0.9

We can see from figures 15 and 16 that both the increase in alpha and beta values lead to the magnitude of the function and the rate of decline of the function. According to formula $z_{t+1} = \beta z_t + \alpha \nabla f(x_t)$, it can be seen that increasing alpha and beta will increase step. When beta=0.9, the function will have a relatively large oscillation at the beginning. Figure 14 shows that when alpha=0.05 beta=0.9 the function starts to diverge and does not converge. Figures 12 and 13 show the size of y step and the size of y, explaining why the function oscillation.

Function2: Max(x-5,0)+3*|y-0|

All experiments are performed at the starting point x=10 and y=1

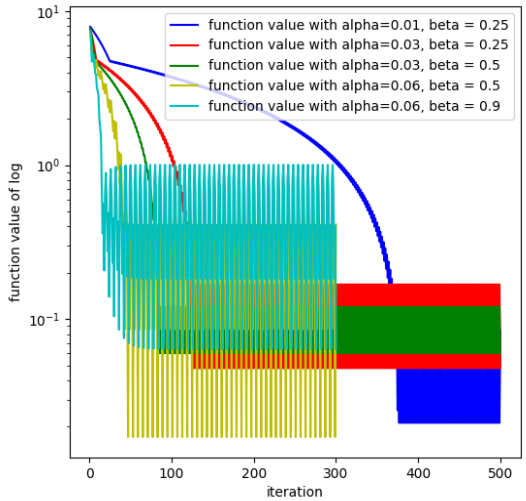

Figure 17 Function value varies with different alpha and beta

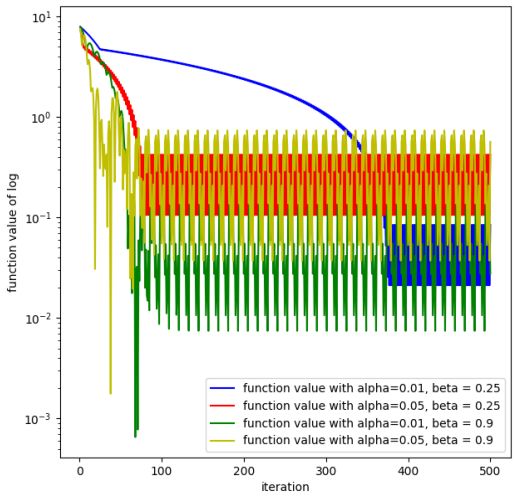

Figure 18 Function value varies with different alpha and beta

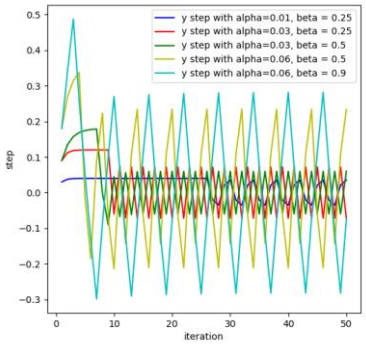

Figure 19 y step varies with different alpha and beta

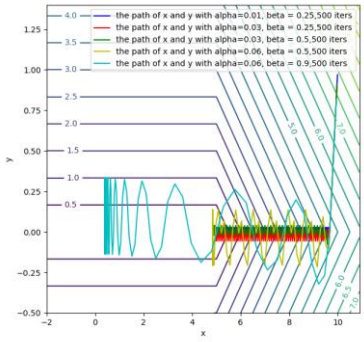

Figure 20 Descending paths of x and y under different alpha and beta

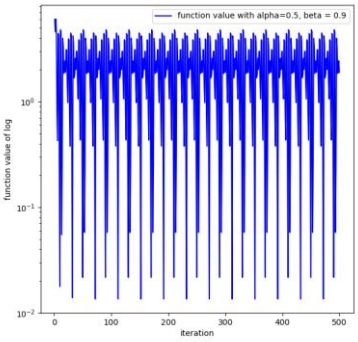

Figure 21

We can see from Figures 17 and 18 that the minimum value of the function value increases as alpha increases, and the rate at which the value of the function decreases increases, as does the effect of the beta value on the value of the function. These situations are similar to those in function1. We can also see from figures 19 and 20 that as the alpha and beta values increase, the y step and the y value tremble more severely. This also causes the function value to jitter, because the partial derivatives in function2 are all constant, and larger parameters lead to larger step sizes.

(iii) α, β1 and β2 in Adam.

Function 1:1*(x-5)^4+3*(y-0)^2

All experiments are performed at the starting point x=1 and y=1

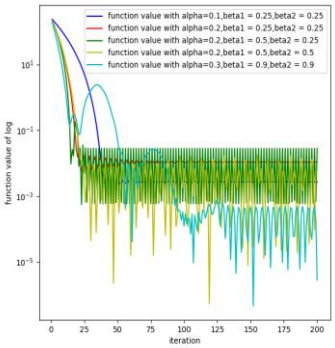

figure 22 Function value changes with different alpha, beta1 and beta2

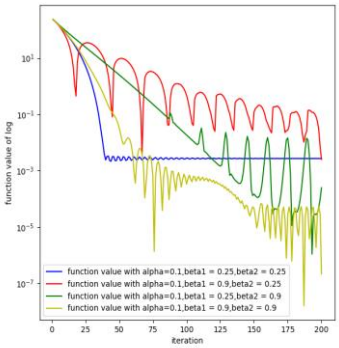

figure 23 The function value changes with alpha=0.1 and different beta1, beta2

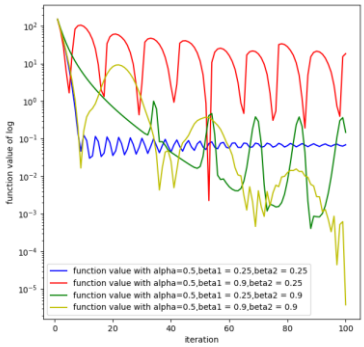

figure 24 The function value changes with alpha=0.5 and different beta1, beta2

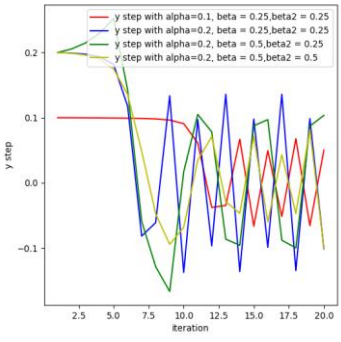

figure 25 Variation of y step under different parameters

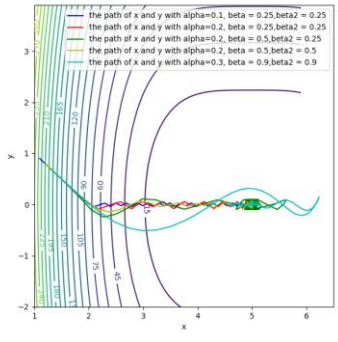

figure 26 Function descent path under different parameters

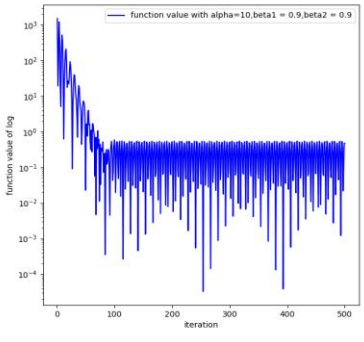

figure 27 function fails to converge

Through figure22,23,24 we can see that as alpha increases, the function value decreases faster and faster, this is because according to the formula in a(iv)$x_{t+1} = x_t - \alpha \frac{\widehat{m_1}}{\sqrt{\widehat{v_1}-\epsilon}}$。 The larger the value of Alpha, the larger the step. This can also be seen from figure25. We can also see that as beta1 increases, according to the formula：

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla f(x_t)$$

The value of m pays more attention to historical data, that is, it increases slowly, the maximum and minimum values are relatively small, and the increase/decrease is slow, and then according to the formula： $\widehat{m} = \frac{m_{t+1}}{(1-\beta_1^t)}$, $\widehat{m}$becomes a relatively large value as beta1 increases, which causes the red line in figure 23 and 24 to compare with the blue line. The red line starts at a high position and has a large step, but the decline is slow. As beta2 increases:

$$v_{t+1} = \beta_2 v_t + (1-\beta_2)[\frac{\partial f}{\partial x_1}(x_t)^2, \frac{\partial f}{\partial x_2}(x_t)^2, \ldots\ldots, \frac{\partial f}{\partial x_n}(x_t)^2]$$

vt is also more focused on historical data and increases/decreases slowly, according to $\hat{v} = \frac{v_{t+1}}{(1-\beta_2^t)}$, $\hat{v}$ also takes a relatively large value， according to $\frac{\widehat{m_1}}{\sqrt{\widehat{v_1}-\epsilon}}$, v as the denominator, the larger the value, the smaller the step, which results in the green lines in figures 23 and 24. Figure 25 illustrates the change in y step. While beta1 and beta2 increase at the same time, both focus on historical data. The accumulated data of m and v are slow, but the starting value is relatively high. according to $\frac{\widehat{m_1}}{\sqrt{\widehat{v_1}-\epsilon}}$. The numerator and denominator become larger at the same time, so the step will not be very large, so that you can iterate slowly and find the best value.

Function 2:Max(x-5,0)+3*|y-0|

All experiments are performed at the starting point x=10 and y=1

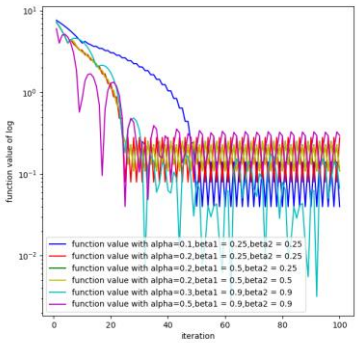

figure 28 Function value changes with different alpha, beta1 and beta2

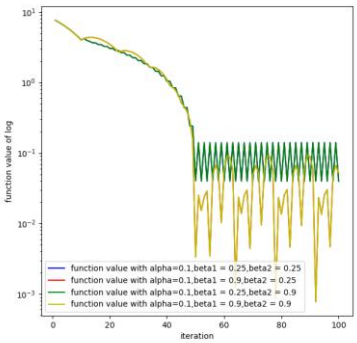

figure 29 The function value changes with alpha=0.1 and different beta1, beta2

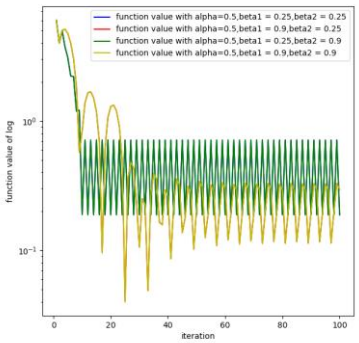

figure 30 The function value changes with alpha=0.5 and different beta1, beta2

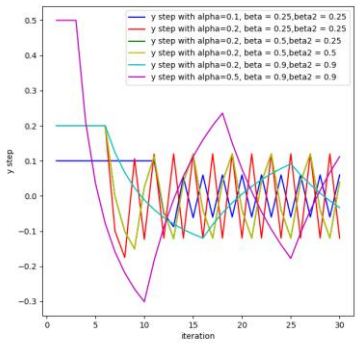

figure 31 Variation of y step under different parameters

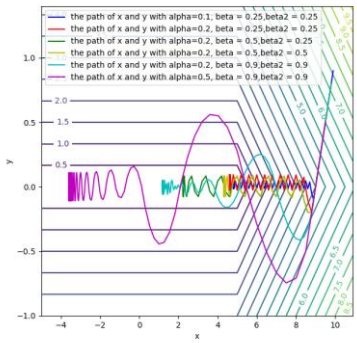

figure 32 Function descent path under different parameters

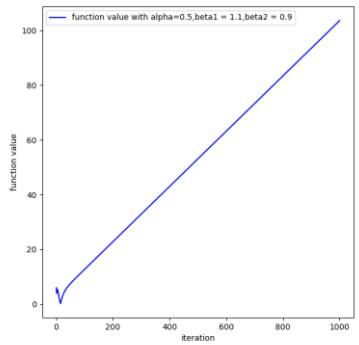

figure 33 function fails to converge

We can see from figure 28 that as the alpha value increases, the function value decreases faster with each iteration, which is the same as in function1. Figure 31 and figure 32 also illustrate this very well. The larger the alpha, the larger the step and value of the y vibration. But it can be seen from figures 29 and 30 that the function does not change as the value of beta2 changes, because the partial derivative of the equation for y is always a fixed value, according to the formula:$v_{t+1} = \beta_2 v_t + (1-\beta_2)\frac{\partial f}{\partial x_1}(x_t)^2$, Each iteration of vt maintains a fixed

value, so changing beta2 will not change the function, and as for beta1, the partial derivative of x has only two values, so changing the value of beta1 has less effect on the function value.

(c) Now you'll look at the ReLu function M ax(0, x).

get the function equation:$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$,

get the derivative function equation:$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$

(i)

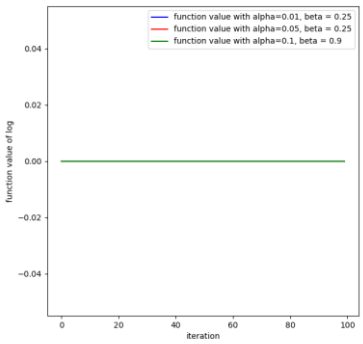

figure 34 RMSProp with start x = -1 and different parameters

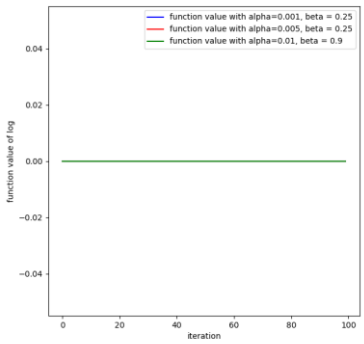

figure 35 heavyBall with start x = -1 and different parameters

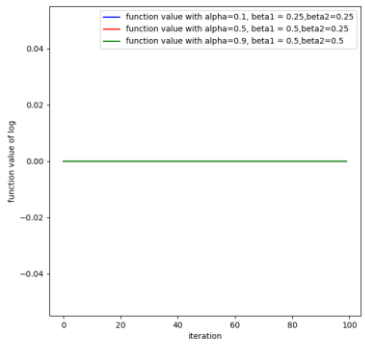

figure 36 adam with start x = -1 and different parameters

We can see in Figures 34, 35, and 36 that all algorithms are under different parameters, and the results are all 0. This is because according to the function and derivative function obtained in c, the participation obtained when x<0 The calculated function value and the derivative function value are both 0

(ii) Now look at initial condition x = +1 and repeat. How does the behaviour change, and why?

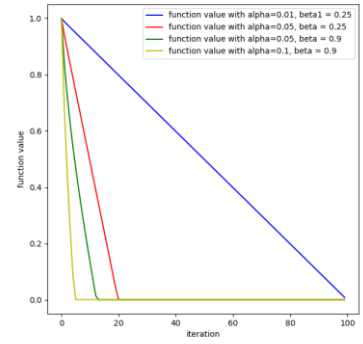

Figure 37 RMSProp with start x = 1 and different parameters

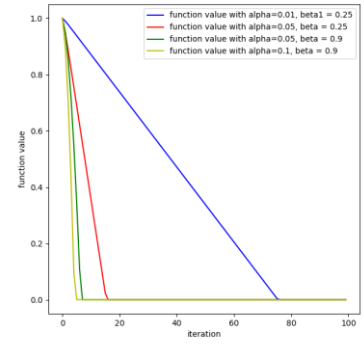

Figure 38 heavyBall with start x = 1 and different parameters

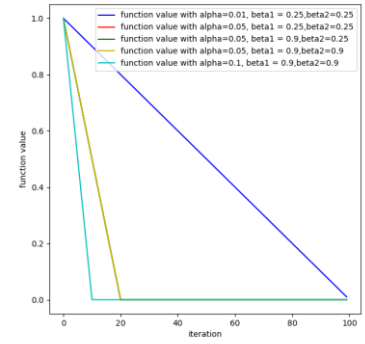

Figure 39 adam with start x = 1 and different parameters

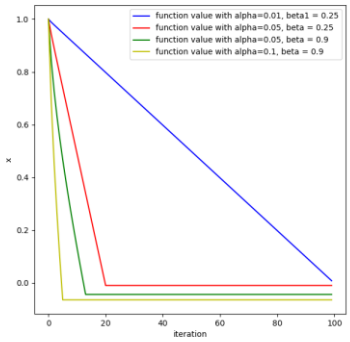

Figure 40 x changes under different parameters of the RMSProp algorithm

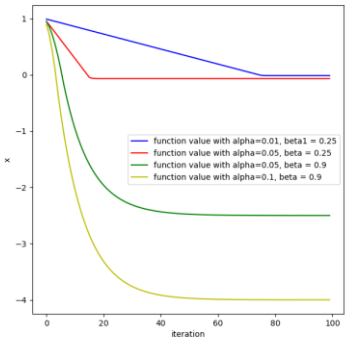

Figure 41 x changes under different parameters of the heavy ball algorithm

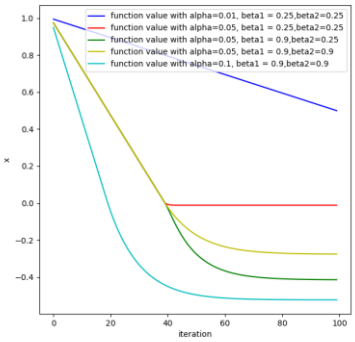

Figure 42 x changes under different parameters of the Adam algorithm

We can see from figure 37, 38, 39 that all three algorithms start to decline when the starting point is 1. From figure 37, we can see that RMSProp will accelerate the decline of the function value with the increase of alpha and beta. As in the case of b, alpha is used as the numerator, and the larger the value, the larger the step size. The larger the beta value, the smaller the sum value at the beginning, and the smaller the sum as the denominator, the larger the step size, resulting in a faster decline in the function value.
In heavy ball, beta and alpha directly affect the size of step as in b. Any increase will result in faster iterations. While in Adam, increasing beta1 and beta2 has no effect, this is because since the derivative is a constant, and equations: $\hat{v} = \frac{v_{t+1}}{(1-\beta_2^t)}$ and $\hat{m} = \frac{m_{t+1}}{(1-\beta_1^t)}$, cancel out the effects of beta2 and beta1 in equations

$v_{t+1} = \beta_2 v_t + (1-\beta_2)\frac{\partial f}{\partial x_1}(x_t)^2$ and $m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla f(x_t)$

so $\hat{v}$ and $\hat{m}$ both will not change, they are still constant, so the step size will not changed.

.

(iii)

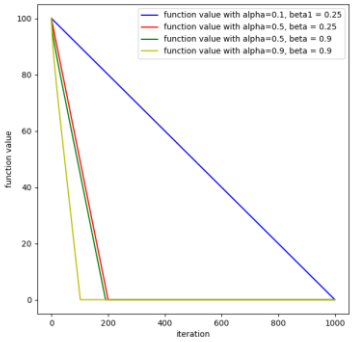

Figure 43 RMSProp with start x = 100 and different parameters

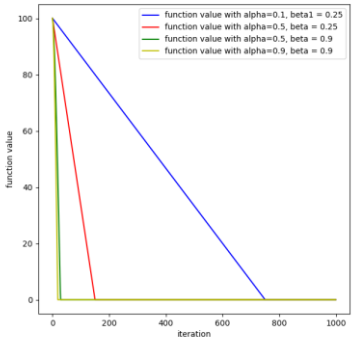

Figure 44 heavyBall with start x = 100 and different parameters

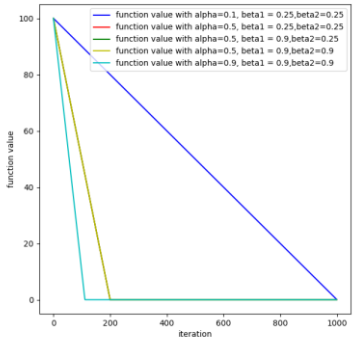

Figure 45 adam with start x = 100 and different parameters

We can see that when the starting point becomes 100, all three functions fall slower and require more iterations to converge to 0. This is because the function derivatives are all 1 when x>0.

Appendix

Code

a and b:

```
1.  from typing import overload
```

```
2.
3.  import numpy as np
4.  import sympy
5.  import math
6.
7.  from matplotlib import pyplot as plt
8.  from sympy import Max
9.
10. x = sympy.symbols('x', real=True)
11. f = x ** 2
12. dfdx = sympy.diff(f, x)
13. f = sympy.lambdify(x, f)
14. dfdx = sympy.lambdify(x, dfdx)
15.
16. x1, y1 = sympy.symbols('x,y', real=True)
17. xa = sympy.Array([x1, y1])
18. f1 = 1 * (x1 - 5) ** 4 + 3 * (y1 - 0) ** 2
19. # f1 = (1 - x1) ** 2 + 100 * (y1 - x1 ** 2) ** 2
20. # f1 = abs(x1) + y1 ** 2
21. # f1 = 0.5 * (x1 ** 2 + 10 * y1 ** 2)
22. # f1 = Max(x1 - 5, 0) + 3 * abs(y1 - 0)
23. dfdx1 = sympy.diff(f1, xa)
24. # dfdx2 = sympy.diff(f2, xa)
25. print(dfdx1)
26.
27. f1 = sympy.lambdify(xa, f1)
28. print(f1(1, 1))
29. # f2 = sympy.lambdify(xa, f2)
30. dfdx1 = sympy.lambdify(xa, dfdx1)
31. # dfdx2 = sympy.lambdify(xa, dfdx2)
32.
33.
34. x2 = sympy.symbols('x', real=True)
35. f2 = 3 * abs(x2 - 0)
36. dfdx2 = sympy.diff(f2, x2)
37. dfdx2 = sympy.lambdify(x2, dfdx2)
38. print(dfdx2(2))
39.
40.
41. # print(dfdx2)
42.
43. # f = sympy.lambdify(x, f)
44. # dfdx = sympy.lambdify(x, dfdx)
45.
46.
47. def getF1(x):
48.     return f(x)
49.
50.
51. def getF2(x, y):
52.     return f1(x, y)
53.
54.
55. def getDf1(x):
56.     return dfdx(x)
57.
58.
59. def getDf2(x, y):
60.     return dfdx1(x, y)
61. # def getDf(x):
62. #     if x > 5:
63. #         return 1
64. #     else:
65. #         return 0
66. #
67. #
```

```
# def getDf2(x, y):
#     dx = getDf(x)
#     dy = dfdx2(y)
#     return dx, dy
#
#
# def getPF1(x, y):
#     x1 = x.copy()
#     for i in range(0, x.shape[0]):
#         for j in range(0, x.shape[1]):
#             if x[i][j] <= 5:
#                 x1[i][j] = 0
#             else:
#                 x1[i][j] = x[i][j] - 5
#
#
#     part2 = 3 * abs(y)
#
#     return x1 + part2


def polyak(startX, startY, maxIters, epsilon, ii, ss):
    x = startX
    i = 0
    if startY is not None:
        y = startY
        while i < maxIters:
            numerator = getF2(x, y) - 0
            dx, dy = getDf2(x, y)
            alpha = numerator / (dx ** 2 + dy ** 2 + epsilon)
            x = x - alpha * dx
            y = y - alpha * dy
            i = i + 1
            ii.append(i)
            ss.append(getF2(x, y))
    else:
        while i < maxIters:
            numerator = getF1(x) - 0
            alpha = numerator / (getDf1(x) ** 2 + epsilon)
            x = x - alpha * getDf1(x)
            i = i + 1
            ii.append(i)
            ss.append(alpha)
    return ii, ss


def RMSProp(startX, startY, maxIters, alpha0, Beta, epsilon, ii, ff, ssx, ssy, xx, yy):
    x = startX
    i = 0
    sum = 0
    if startY is None:
        while i < maxIters:
            sum = Beta * sum + (1 - Beta) * getDf1(x) ** 2
            alpha = alpha0 / (math.sqrt(sum) + epsilon)
            x = x - alpha * getDf1(x)
            i = i + 1
            ii.append(i)
            ff.append(alpha)
    else:
        y = startY
        sum2 = 0
        while i < maxIters:
            dx, dy = getDf2(x, y)
            sum = Beta * sum + (1 - Beta) * (dx ** 2)
            print(sum)
```

```
            sum2 = Beta * sum2 + (1 - Beta) * (dy ** 2)
            alpha = alpha0 / (math.sqrt(sum) + epsilon)
            alpha2 = alpha0 / (math.sqrt(sum2) + epsilon)
            x = x - alpha * dx
            y = y - alpha2 * dy
            i = i + 1
            ii.append(i)
            ff.append(getF2(x, y))
            ssx.append(alpha)
            ssy.append(alpha2)
            xx.append(x)
            yy.append(y)
    return ii, ff, ssx, ssy, xx, yy


def heavyBall(startX, startY, maxIters, alpha, Beta, ii, ff, ssx, ssy, xx, yy):
    x = startX
    i = 0
    z = 0
    if startY is None:
        while i < maxIters:
            z = Beta * z + alpha * getDf1(x)
            x = x - z
            i = i + 1
            ii.append(i)
            ff.append(getF1(x))
    else:
        y = startY
        z2 = 0
        while i < maxIters:
            dx, dy = getDf2(x, y)
            z = Beta * z + alpha * dx
            z2 = Beta * z2 + alpha * dy
            x = x - z
            y = y - z2
            i = i + 1
            ii.append(i)
            ff.append(getF2(x, y))
            ssx.append(z)
            ssy.append(z2)
            xx.append(x)
            yy.append(y)
    return ii, ff, ssx, ssy, xx, yy


def Adam(startX, startY, maxIters, alpha, Beta1, Beta2, epsilon, ii, ff, ssx, ssy, xx, yy):
    x = startX
    i = 0
    mt = 0
    vt = 0
    if startY is None:
        while i < maxIters:
            mt = Beta1 * mt + (1 - Beta1) * getDf1(x)
            vt = Beta2 * vt + (1 - Beta2) * (getDf1(x) ** 2)
            me = mt / (1 - Beta1 ** (i + 1))
            ve = vt / (1 - Beta2 ** (i + 1))
            x = x - alpha * (me / (math.sqrt(ve) + epsilon))
            i = i + 1
            ii.append(i)
            ff.append(getF1(x))

    else:
        y = startY
        mt2 = 0
```

```
        vt2 = 0
        while i < maxIters:
            dx, dy = getDf2(x, y)
            mt = Beta1 * mt + (1 - Beta1) * dx
            mt2 = Beta1 * mt2 + (1 - Beta1) * dy
            vt = Beta2 * vt + (1 - Beta2) * (dx ** 2)
            print(vt)
            vt2 = Beta2 * vt2 + (1 - Beta2) * (dy ** 2)
            me = mt / (1 - Beta1 ** (i + 1))
            me2 = mt2 / (1 - Beta1 ** (i + 1))
            ve = vt / (1 - Beta2 ** (i + 1))
            ve2 = vt2 / (1 - Beta2 ** (i + 1))
            x = x - alpha * (me / (math.sqrt(ve) + epsilon))
            y = y - alpha * (me2 / (math.sqrt(ve2) + epsilon))
            i = i + 1
            ii.append(i)
            ff.append(getF2(x, y))
            ssx.append(alpha * (me / (math.sqrt(ve) + epsilon)))
            ssy.append(alpha * (me2 / (math.sqrt(ve2) + epsilon)))
            xx.append(x)
            yy.append(y)
    return ii, ff, ssx, ssy, xx, yy

plt.figure(figsize=(7, 7))
# polyak(1, None, 50, 0.0001)
# polyak(-1.25,0.5, 1500, 0.0001)

# RMSProp(1,None,50,0.06,0.9,0.0001)
# ii, ff, ssx, ssy, xx, yy = RMSProp(1, 1, 100, 0.1, 0.9, 0.0001, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = RMSProp(1, 1, 500, 0.2, 0.25, 0.0001, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = RMSProp(1, 1, 500, 0.2, 0.5, 0.0001, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = RMSProp(1, 1, 500, 0.5, 0.5, 0.0001, [], [], [], [], [], [])
# ii4, ff4, ssx4, ssy4, xx4, yy4 = RMSProp(1, 1, 500, 0.5, 0.9, 0.0001, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = RMSProp(1, 1, 100, 0.1, 0.25, 0.0001, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = RMSProp(1, 1, 100, 0.2, 0.25, 0.0001, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = RMSProp(1, 1, 100, 0.3, 0.25, 0.0001, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = RMSProp(1, 1, 100, 0.1, 0.9, 0.0001, [], [], [], [], [], [])
# ii4, ff4, ssx4, ssy4, xx4, yy4 = RMSProp(1, 1, 80, 0.2, 0.9, 0.0001, [], [], [], [], [], [])
# ii5, ff5, ssx5, ssy5, xx5, yy5 = RMSProp(1, 1, 80, 0.3, 0.9, 0.0001, [], [], [], [], [], [])
# ii5, ff5, ssx5, ssy5, xx5, yy5 = RMSProp(1, 1, 200, 0.9, 0.9, 0.0001, [], [], [], [], [], [])
# RMSProp(0.02, 0.1, 1000, 0.01, 0.9, 0.00001)
# heavyBall(1, None, 50, 1, 0.25)
# ii, ff, ssx, ssy, xx, yy = heavyBall(1, 1, 200, 0.001, 0.25, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = heavyBall(1, 1, 200, 0.005, 0.25, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = heavyBall(1, 1, 200, 0.005, 0.5, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = heavyBall(1, 1, 200, 0.01, 0.5, [], [], [], [], [], [])
# ii4, ff4, ssx4, ssy4, xx4, yy4 = heavyBall(1, 1, 200, 0.01, 0.9, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = heavyBall(1, 1, 100, 1, 0.25, [], [], [], [], [], [])
```

```
# ii1, ff1, ssx1, ssy1, xx1, yy1 = heavyBall(1, 1, 500, 0.005, 0.25, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = heavyBall(1, 1, 500, 0.01, 0.5, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = heavyBall(1, 1, 500, 0.001, 0.9, [], [], [], [], [], [])
# ii4, ff4, ssx4, ssy4, xx4, yy4 = heavyBall(1, 1, 500, 0.005, 0.9, [], [], [], [], [], [])
# ii5, ff5, ssx5, ssy5, xx5, yy5 = heavyBall(1, 1, 500, 0.01, 0.9, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = heavyBall(1, 1, 5, 0.05, 0.9, [], [], [], [], [], [])


# plt.plot(ii, ff, color='b', label='function value with alpha=0.01, beta = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.2, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.2, beta = 0.5')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.5, beta = 0.5')
# plt.plot(ii4, ff4, color='c', label='function value with alpha=0.5, beta = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.1, beta = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.2, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.3, beta = 0.25')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.1, beta = 0.9')
# plt.plot(ii4, ff4, color='c', label='function value with alpha=0.2, beta = 0.9')
# plt.plot(ii5, ff5, color='m', label='function value with alpha=0.3, beta = 0.9')
# plt.plot(ii5, ff5, color='c', label='function value with alpha=5000, beta=0.9, epsilon=0.0001')
# plt.plot(ii, ssx, color='b', label='x step size with alpha=0.06, beta = 0.25')
# plt.plot(ii, ssy, color='r', label='y step size with alpha=0.06, beta = 0.25')
# plt.plot(ii1, ssx1, color='g', label='x step size with alpha=0.2, beta = 0.25')
# plt.plot(ii1, ssy1, color='y', label='y step size with alpha=0.2, beta = 0.25')
# plt.plot(ii2, ssx2, color='c', label='x step size with alpha=0.2, beta = 0.5')
# plt.plot(ii2, ssy2, color='m', label='y step size with alpha=0.2, beta = 0.5')
# x = np.arange(1, 6, 0.1)
# y = np.arange(-2, 2, 0.1)
# X, Y = np.meshgrid(x, y)
# contours = plt.contour(X, Y, getF2(X, Y), 20);
# plt.clabel(contours, inline=True, fontsize=10)
# plt.plot(xx,yy,color='b', label='the path of x and y with alpha=0.1, beta = 0.25')
# plt.plot(xx2,yy2,color='r', label='the path of x and y with alpha=0.9, beta = 0.25')
# plt.plot(xx3,yy3,color='g', label='the path of x and y with alpha=0.1, beta = 0.9')
# plt.plot(xx5,yy5,color='y', label='the path of x and y with alpha=0.9, beta = 0.9')

```

```
# plt.plot(ii, ff, color='b', label='function value with alpha=0.001, beta = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.005, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.005, beta = 0.5')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.01, beta = 0.5')
# plt.plot(ii4, ff4, color='c', label='function value with alpha=0.01, beta = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.001, beta = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.005, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.008, beta = 0.25')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.001, beta = 0.9')
# plt.plot(ii4, ff4, color='c', label='function value with alpha=0.005, beta = 0.9')
# plt.plot(ii5, ff5, color='m', label='function value with alpha=0.01, beta = 0.9')
# plt.plot(ii, ssy, color='b', label='y step with alpha=0.001, beta = 0.25')

# plt.plot(ii1, ssy1, color='r', label='y step with alpha=0.005, beta = 0.25')
# plt.plot(ii2, ssy2, color='g', label='y step with alpha=0.005, beta = 0.5')
# plt.plot(ii, ssy3, color='y', label='y step with alpha=0.01, beta = 0.5')

# plt.plot(ii, ssy4, color='c', label='y step with alpha=0.01, beta = 0.9')

#
# plt.plot(xx,yy,color='b', label='the path of x and y with alpha=0.001, beta = 0.25')
# plt.plot(xx1,yy1,color='r', label='the path of x and y with alpha=0.005, beta = 0.25')
# plt.plot(xx2,yy2,color='g', label='the path of x and y with alpha=0.005, beta = 0.5')
# plt.plot(xx3,yy3,color='y', label='the path of x and y with alpha=0.01, beta = 0.5')
# plt.plot(xx4,yy4,color='c', label='the path of x and y with alpha=0.01, beta = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.05, beta = 0.9')
#
# Adam(1.0,None, 50, 0.1, 0.9, 0.999, 0)
ii, ff, ssx, ssy, xx, yy = Adam(10, 1, 100, 0.1, 0.25, 0.9, 0.0001, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = Adam(1, 1, 100, 0.5, 0.25, 0.25, 0.0001, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = Adam(1, 1, 100, 0.2, 0.5, 0.25, 0.0001, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = Adam(1, 1, 100, 0.2, 0.5, 0.5, 0.0001, [], [], [], [], [], [])
# ii4, ff4, ssx4, ssy4, xx4, yy4 = Adam(1, 1, 100, 0.9, 0.9, 0.9, 0.0001, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = Adam(1, 1, 200, 0.1, 0.25, 0.25, 0.0001, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = Adam(1, 1, 200, 0.1, 0.9, 0.25, 0.0001, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = Adam(1, 1, 200, 0.1, 0.25, 0.9, 0.0001, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = Adam(1, 1, 100, 0.5, 0.25, 0.25, 0.0001, [], [], [], [], [], [])
```

```
# ii4, ff4, ssx4, ssy4, xx4, yy4 = Adam(1, 1, 100, 0.5, 0.9, 0.25, 0.0001, [], [], [], [], [], [])
# ii5, ff5, ssx5, ssy5, xx5, yy5 = Adam(1, 1, 100, 0.5, 0.25, 0.9, 0.0001, [], [], [], [], [], [])
# ii6, ff6, ssx6, ssy6, xx6, yy6 = Adam(1, 1, 200, 0.1, 0.9, 0.9, 0.0001, [], [], [], [], [], [])
# ii7, ff7, ssx7, ssy7, xx7, yy7 = Adam(1, 1, 100, 0.5, 0.9, 0.9, 0.0001, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = Adam(1, 1, 2000, 0.3, 1.1, 0.95, 0.0001, [], [], [], [], [], [])

# plt.plot(ii, ff, color='b', label='function value with alpha=0.1,beta1 = 0.25,beta2 = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.2,beta1 = 0.25,beta2 = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.2,beta1 = 0.5,beta2 = 0.25')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.2,beta1 = 0.5,beta2 = 0.5')
# plt.plot(ii4, ff4, color='c', label='function value with alpha=0.3,beta1 = 0.9,beta2 = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.1,beta1 = 0.25,beta2 = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.1,beta1 = 0.9,beta2 = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.1,beta1 = 0.25,beta2 = 0.9')
# plt.plot(ii6, ff6, color='y', label='function value with alpha=0.1,beta1 = 0.9,beta2 = 0.9')
# plt.plot(ii3, ff3, color='b', label='function value with alpha=0.5,beta1 = 0.25,beta2 = 0.25')
# plt.plot(ii4, ff4, color='r', label='function value with alpha=0.5,beta1 = 0.9,beta2 = 0.25')
# plt.plot(ii5, ff5, color='g', label='function value with alpha=0.5,beta1 = 0.25,beta2 = 0.9')
# plt.plot(ii7, ff7, color='y', label='function value with alpha=0.5,beta1 = 0.9,beta2 = 0.9')
# plt.plot(ii4, ff4, color='c', label='function value with alpha=0.5,beta1 = 0.9,beta2 = 0.9')
# plt.plot(ii5, ff5, color='m', label='function value with alpha=0.9,beta1 = 0.9,beta2 = 0.9')
# plt.plot(ii, ssx, color='b', label='x step with alpha=0.1, beta = 0.25,beta2 = 0.25')
plt.plot(ii, ssy, color='r', label='y step with alpha=0.1, beta = 0.25,beta2 = 0.25')
# plt.plot(ii1, ssy1, color='b', label='y step with alpha=0.2, beta = 0.25,beta2 = 0.25')
# plt.plot(ii2, ssy2, color='g', label='y step with alpha=0.2, beta = 0.5,beta2 = 0.25')
# plt.plot(ii3, ssy3, color='y', label='y step with alpha=0.2, beta = 0.5,beta2 = 0.5')
# plt.plot(xx, yy, color='b', label='the path of x and y with alpha=0.1, beta = 0.25,beta2 = 0.25')
# plt.plot(xx1, yy1, color='r', label='the path of x and y with alpha=0.2, beta = 0.25,beta2 = 0.25')
# plt.plot(xx2, yy2, color='g', label='the path of x and y with alpha=0.2, beta = 0.5,beta2 = 0.25')
# plt.plot(xx3, yy3, color='y', label='the path of x and y with alpha=0.2, beta = 0.5,beta2 = 0.5')
# plt.plot(xx4, yy4, color='c', label='the path of x and y with alpha=0.3, beta = 0.9,beta2 = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.3, beta1 = 1.1,beta2 = 0.95')

plt.xlabel('iteration')
plt.ylabel('function value of log')
```

```
# plt.yscale("log")
# plt.xscale("log")
# plt.ylim((-0.5, 0.1))
plt.legend()
plt.show()
```

c:

```
import sympy
import math

from matplotlib import pyplot as plt
from sympy import Max

x = sympy.symbols('x', real=True)
f = Max(x,0)
dfdx = sympy.diff(f, x)
print(dfdx)
f = sympy.lambdify(x, f)
dfdx = sympy.lambdify(x, dfdx)


def getF1(x):
    return f(x)


def getDf1(x):
    return dfdx(x)


def getDf(x):
    if x > 5:
        return 1
    else:
        return 0


def RMSProp(startX, startY, maxIters, alpha0, Beta, epsilon, ii, ff, ssx, ssy, xx, yy):
    x = startX
    i = 0
    sum = 0
    while i < maxIters:
        ii.append(i)
        ff.append(getF1(x))
        xx.append(x)
        sum = Beta * sum + (1 - Beta) * getDf1(x) ** 2
        alpha = alpha0 / (math.sqrt(sum) + epsilon)

        x = x - alpha * getDf1(x)
        i = i + 1
        ssx.append(alpha)



    return ii, ff, ssx, ssy, xx, yy


def heavyBall(startX, startY, maxIters, alpha, Beta, ii, ff, ssx, ssy, xx, yy):
    x = startX
    i = 0
    z = 0
    while i < maxIters:
        ii.append(i)
        ff.append(getF1(x))
        z = Beta * z + alpha * getDf1(x)
```

```
        x = x - z
        i = i + 1
        ssx.append(z)
        xx.append(x)


    return ii, ff, ssx, ssy, xx, yy


def Adam(startX, startY, maxIters, alpha, Beta1, Beta2, epsilon, ii, ff, ssx, ssy, xx, yy):
    x = startX
    i = 0
    mt = 0
    vt = 0
    while i < maxIters:
        ii.append(i)
        ff.append(getF1(x))
        mt = Beta1 * mt + (1 - Beta1) * getDf1(x)
        print("mt:" + str(mt))
        vt = Beta2 * vt + (1 - Beta2) * (getDf1(x) ** 2)
        print("vt:" + str(vt))
        me = mt / (1 - Beta1 ** (i + 1))
        print("me:"+str(me))
        ve = vt / (1 - Beta2 ** (i + 1))
        print("ve:" + str(ve))
        x = x - alpha * (me / (math.sqrt(ve) + epsilon))
        i = i + 1
        ssx.append(alpha * (me / (math.sqrt(ve) + epsilon)))
        xx.append(x)

    return ii, ff, ssx, ssy, xx, yy

plt.figure(figsize=(7, 7))

# ii, ff, ssx, ssy, xx, yy = RMSProp(1, None, 100, 0.01, 0.9, 0.0001, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = RMSProp(1, None, 100, 0.05, 0.25, 0.0001, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = RMSProp(1, None, 100, 0.05, 0.9, 0.0001, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = RMSProp(1, None, 100, 0.1, 0.9, 0.0001, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = heavyBall(100, None, 1000, 0.1, 0.25, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = heavyBall(100, None, 1000, 0.5, 0.25, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = heavyBall(100, None, 1000, 0.5, 0.9, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = heavyBall(100, None, 1000, 0.9, 0.9, [], [], [], [], [], [])
# ii, ff, ssx, ssy, xx, yy = heavyBall(100, None, 1000, 0.1, 0.25, [], [], [], [], [], [])
# ii1, ff1, ssx1, ssy1, xx1, yy1 = heavyBall(100, None, 1000, 0.5, 0.25, [], [], [], [], [], [])
# ii2, ff2, ssx2, ssy2, xx2, yy2 = heavyBall(100, None, 1000, 0.5, 0.9, [], [], [], [], [], [])
# ii3, ff3, ssx3, ssy3, xx3, yy3 = heavyBall(100, None, 1000, 0.9, 0.9, [], [], [], [], [], [])
ii, ff, ssx, ssy, xx, yy = Adam(100, None,1000 , 0.1, 0.25, 0.25, 0.0001, [], [], [], [], [], [])
ii1, ff1, ssx1, ssy1, xx1, yy1 = Adam(100, None, 1000, 0.5, 0.25, 0.25, 0.0001, [], [], [], [], [], [])
ii2, ff2, ssx2, ssy2, xx2, yy2 = Adam(100, None, 1000, 0.5, 0.9, 0.25, 0.0001, [], [], [], [], [], [])
```

```
ii3, ff3, ssx3, ssy3, xx3, yy3 = Adam(100, None, 1000, 0.5, 0.9, 0.9, 0.0001, [], [], [], [], [], [])
ii4, ff4, ssx4, ssy4, xx4, yy4 = Adam(100, None, 1000, 0.9, 0.9, 0.9, 0.0001, [], [], [], [], [], [])

# plt.plot(ii, ff, color='b', label='function value with alpha=0.1, beta1 = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.5, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.5, beta = 0.9')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.9, beta = 0.9')
# plt.plot(ii, xx, color='b', label='function value with alpha=0.01, beta1 = 0.25')
# plt.plot(ii1, xx1, color='r', label='function value with alpha=0.05, beta = 0.25')
# plt.plot(ii2, xx2, color='g', label='function value with alpha=0.05, beta = 0.9')
# plt.plot(ii3, xx3, color='y', label='function value with alpha=0.1, beta = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.01, beta1 = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.05, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.05, beta = 0.9')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.1, beta = 0.9')
# plt.plot(ii, xx, color='b', label='function value with alpha=0.01, beta1 = 0.25')
# plt.plot(ii1, xx1, color='r', label='function value with alpha=0.05, beta = 0.25')
# plt.plot(ii2, xx2, color='g', label='function value with alpha=0.05, beta = 0.9')
# plt.plot(ii3, xx3, color='y', label='function value with alpha=0.1, beta = 0.9')
# plt.plot(ii, ff, color='b', label='function value with alpha=0.1, beta1 = 0.25')
# plt.plot(ii1, ff1, color='r', label='function value with alpha=0.5, beta = 0.25')
# plt.plot(ii2, ff2, color='g', label='function value with alpha=0.5, beta = 0.9')
# plt.plot(ii3, ff3, color='y', label='function value with alpha=0.9, beta = 0.9')
# plt.plot(ii, ssx, color='b', label='x step size with alpha=0.06, beta = 0.25')
# plt.plot(ii, ssy, color='r', label='y step size with alpha=0.06, beta = 0.25')
plt.plot(ii, ff, color='b', label='function value with alpha=0.1, beta1 = 0.25,beta2=0.25')
plt.plot(ii1, ff1, color='r', label='function value with alpha=0.5, beta1 = 0.25,beta2=0.25')
plt.plot(ii2, ff2, color='g', label='function value with alpha=0.5, beta1 = 0.9,beta2=0.25')
plt.plot(ii3, ff3, color='y', label='function value with alpha=0.5, beta1 = 0.9,beta2=0.9')
plt.plot(ii4, ff4, color='c', label='function value with alpha=0.9, beta1 = 0.9,beta2=0.9')
# plt.plot(ii, xx, color='b', label='function value with alpha=0.01, beta1 = 0.25,beta2=0.25')
# plt.plot(ii1, xx1, color='r', label='function value with alpha=0.05, beta1 = 0.25,beta2=0.25')
# plt.plot(ii2, xx2, color='g', label='function value with alpha=0.05, beta1 = 0.9,beta2=0.25')
```

```
# plt.plot(ii3, xx3, color='y', label='function value with alpha=0.05, beta1 = 0.9,beta2=0.9')
# plt.plot(ii4, xx4, color='c', label='function value with alpha=0.1, beta1 = 0.9,beta2=0.9')


plt.xlabel('iteration')
plt.ylabel('function value')
# plt.yscale("log")
# plt.xscale("log")
# plt.ylim((-0.5, 0.1))
plt.legend()
plt.show()
```