+FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 3
“Optimization functions”

Performed by
Chernobrovkin T.V. (412642)
Academic group J4133c
Accepted by
Gladilin P.E.

St. Petersburg
2023

Goal

Construct and investigate the performance of optimization functions gradient descent with moments and Adam based on ordinary gradient descent.

Problem

1. Develop yourself using numpy library:

a) python function for implementation of gradient descent (GD) with momentum algorithm for the function of two variables f(x,y).

b) python function for implementation of ADAM optimization algorithm for the function of two variables f(x,y).

2. Come up with a function of two variables of an arbitrary form and implement the search for its minimum using those implemented in p.1 functions - a) and b).

3. Illustrate the process of finding an extremum in the form of a graph of the dependence of the value of the difference between two successive approximations of the solution (x_next – x_prev) on the iteration number N or in the form of 2D-plane graph.

4. Write a conclusions about the accuracy of the implemented algorithms.

Theory

Gradient descent, gradient descent method is a numerical method of finding the local minimum or maximum of a function by moving along a gradient, one of the main numerical methods of modern optimization.

It is actively used in computational mathematics not only for direct solution of optimization (minimization) problems, but also for problems that can be rewritten in the language of optimization (solution of nonlinear equations, search for equilibria, inverse problems, etc.). The gradient descent method can be used for optimization problems in infinite-dimensional spaces, for example, for numerical solution of optimal control problems.

A problem with gradient descent is that it can bounce around the search space on optimization problems that have large amounts of curvature or noisy gradients, and it can get stuck in flat spots in the search space that have no gradient.

Momentum is an extension to the gradient descent optimization algorithm that allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space.

Adam optimizer is the extended version of stochastic gradient descent which could be implemented in various deep learning applications such as computer vision and natural language processing in the future years. It is an optimization algorithm that can be an alternative for the stochastic gradient descent process. The name is derived from adaptive moment estimation. The optimizer is called Adam because uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network. Adam is proposed as the most efficient stochastic optimization which only requires first-order gradients where memory requirement is too little.

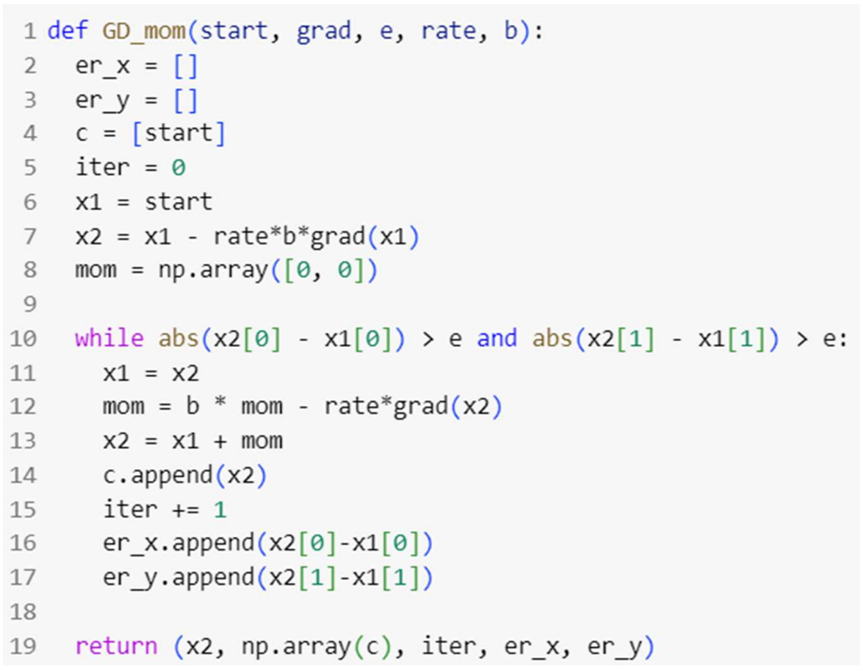
In Adam instead of adapting learning rates based on the average first moment as in RMSP, Adam makes use of the average of the second moments of the gradients. Adam. This algorithm calculates the exponential moving average of gradients and square gradients. And the parameters of β1 and β2 are used to control the decay rates of these moving averages.

Materials and methods

In this task, all calculations were performed on the student's personal laptop in Google Colab. The work was performed in the Python programming language.

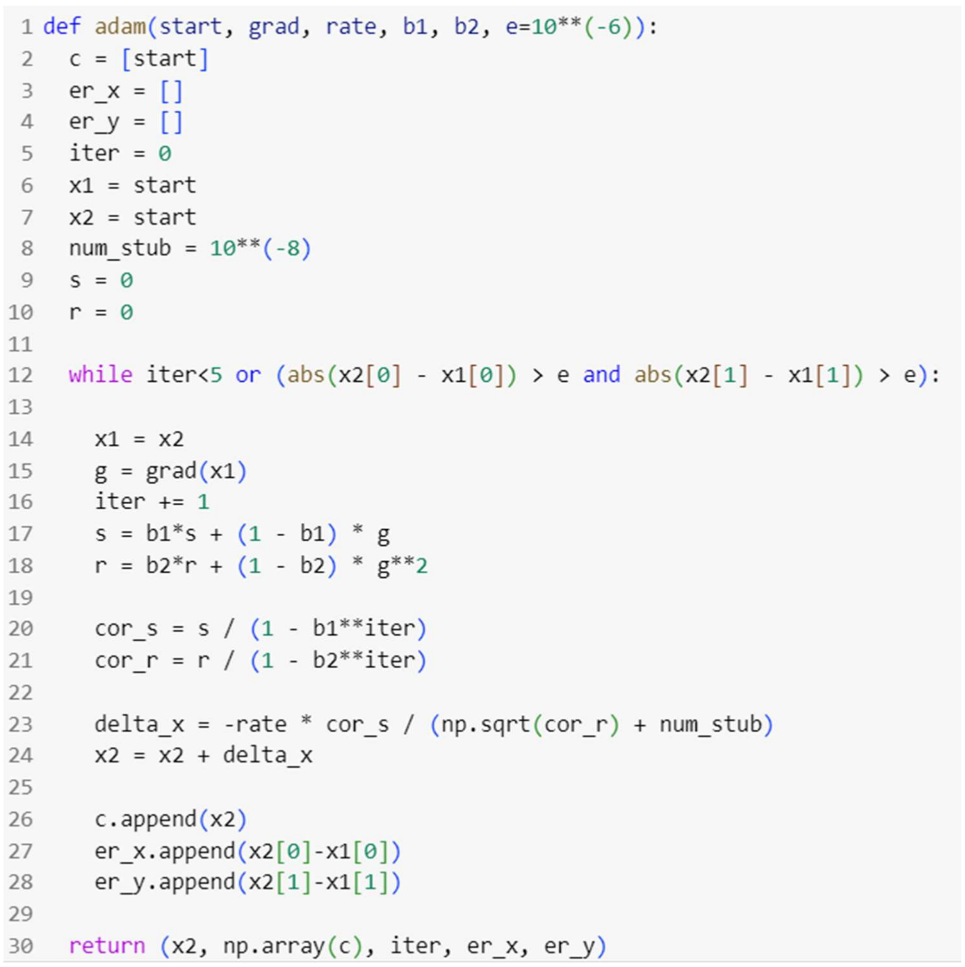
Results

1. Based on gradient descent, a gradient descent optimizer with moments and an Adam optimizer were written. The results are shown in Figure 1 and Figure 2, respectively.



```python
def GD_mom(start, grad, e, rate, b):
  er_x = []
  er_y = []
  c = [start]
  iter = 0
  x1 = start
  x2 = x1 - rate*b*grad(x1)
  mom = np.array([0, 0])

  while abs(x2[0] - x1[0]) > e and abs(x2[1] - x1[1]) > e:
    x1 = x2
    mom = b * mom - rate*grad(x2)
    x2 = x1 + mom
    c.append(x2)
    iter += 1
    er_x.append(x2[0]-x1[0])
    er_y.append(x2[1]-x1[1])

  return (x2, np.array(c), iter, er_x, er_y)
```

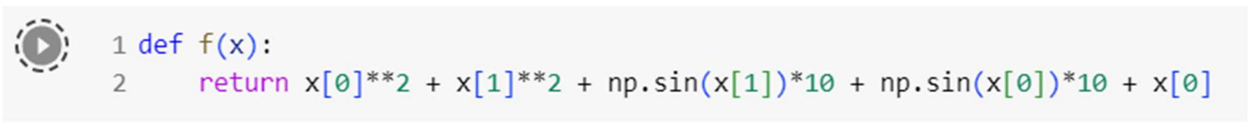
Figure 1 – Gradient desent with momentum



```python
def adam(start, grad, rate, b1, b2, e=10**(-6)):
  c = [start]
  er_x = []
  er_y = []
  iter = 0
  x1 = start
  x2 = start
  num_stub = 10**(-8)
  s = 0
  r = 0

  while iter<5 or (abs(x2[0] - x1[0]) > e and abs(x2[1] - x1[1]) > e):

    x1 = x2
    g = grad(x1)
    iter += 1
    s = b1*s + (1 - b1) * g
    r = b2*r + (1 - b2) * g**2

    cor_s = s / (1 - b1**iter)
    cor_r = r / (1 - b2**iter)

    delta_x = -rate * cor_s / (np.sqrt(cor_r) + num_stub)
    x2 = x2 + delta_x

    c.append(x2)
    er_x.append(x2[0]-x1[0])
    er_y.append(x2[1]-x1[1])

  return (x2, np.array(c), iter, er_x, er_y)
```

Figure 2 – Adam optimizer

2. A function of two variables of arbitrary form was invented (Figure 3). Its surface is presented in Figure 4.



```python
def f(x):
    return x[0]**2 + x[1]**2 + np.sin(x[1])*10 + np.sin(x[0])*10 + x[0]
```

Figure 3 – A function of two variables

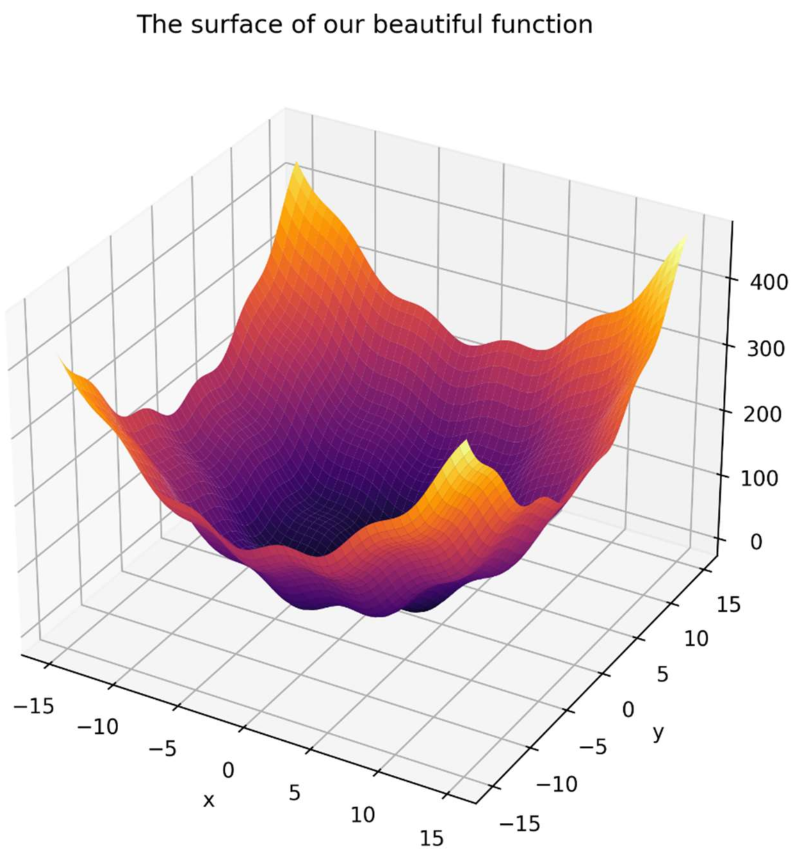
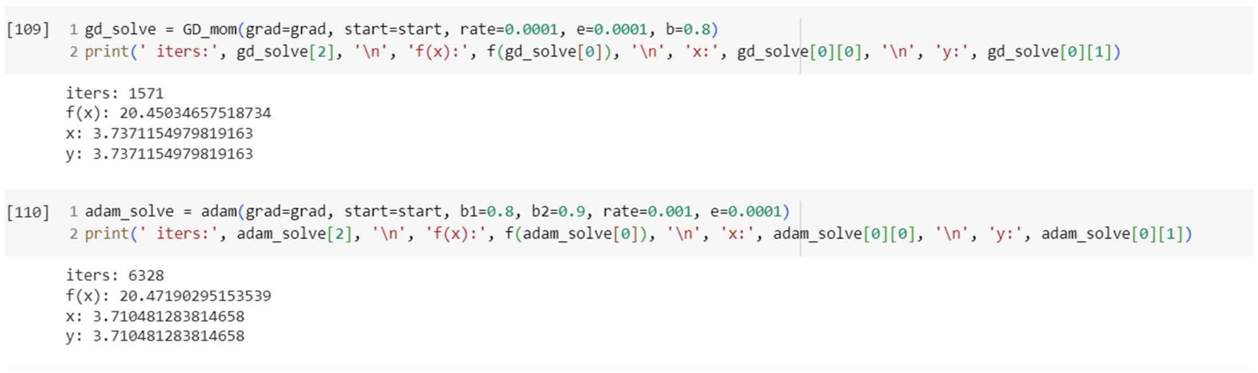


Figure 4 – Surface of our function

The resulting optimizers were applied to find the minimum of this function (Figure 5).



```
[109]  gd_solve = GD_mom(grad=grad, start=start, rate=0.0001, e=0.0001, b=0.8)
       print(' iters:', gd_solve[2], '\n', 'f(x):', f(gd_solve[0]), '\n', 'x:', gd_solve[0][0], '\n', 'y:', gd_solve[0][1])

       iters: 1571
       f(x): 20.45034657518734
       x: 3.7371154979819163
       y: 3.7371154979819163

[110]  adam_solve = adam(grad=grad, start=start, b1=0.8, b2=0.9, rate=0.001, e=0.0001)
       print(' iters:', adam_solve[2], '\n', 'f(x):', f(adam_solve[0]), '\n', 'x:', adam_solve[0][0], '\n', 'y:', adam_solve[0][1])

       iters: 6328
       f(x): 20.47190295153539
       x: 3.710481283814658
       y: 3.710481283814658
```

Figure 5 – Finding the minimum of a function

3. The process of finding the extremum was depicted as a graph of the dependence of the difference between two consecutive approximations of the solution (x2 - x1 and y2 - y1) on the iteration number. The results for GD with momentum and Adam are shown in Figure 6 and Figure 7 respectively.

From the plots obtained, it can be seen that the GD with momentum has a growing x2-x1 and y2-y1 difference over time, but after about 500 iterations the algorithm starts to converge.

The Adam optimizer does not exhibit such failures. The algorithm plateaus for quite a long time and after about 6000 iterations it starts to converge rapidly.

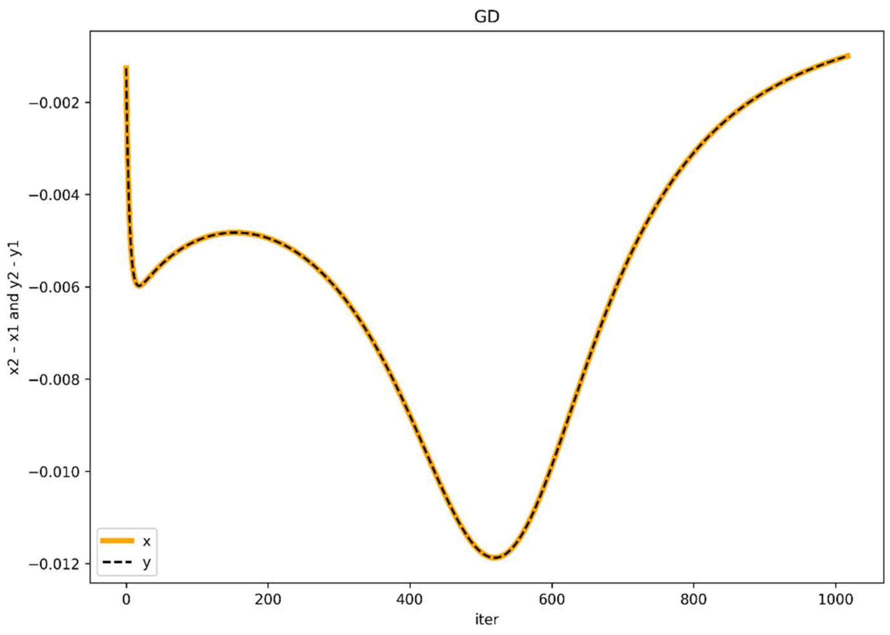


Figure 6 – The process of finding the minimum by gradient descent with momentum

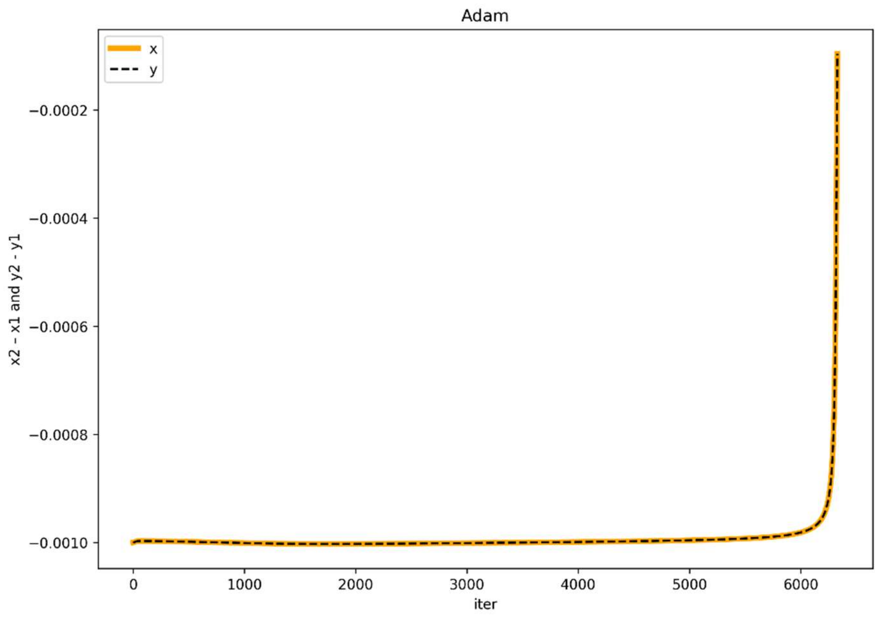


Figure 7 – The process of finding the minimum by Adam

3. Both algorithms worked at the same accuracy of 0.0001. The GD with momentum algorithm converges about 4 times faster than Adam. When working, the code of Adam algorithm I use has the following feature - it needs several operations to start working correctly. In turn, the GD with momentum algorithm works correctly with the convergence condition |x2 - x1| > error and |y2 - y1| > error.

Conclusion

In this assignment, we constructed gradient descent optimization functions with moments and Adam's function based on ordinary gradient descent and investigated their performance.

They were used to find the minimum of our arbitrary function. According to the results obtained, the Adam optimizer is 4 times slower.

Appendix

GitHub link:

https://github.com/LesostepnoyGnom/homework_ML/blob/main/task3/Task_3_Chernobrovkin_J4133c.ipynb