FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 3

“Algorithms for unconstrained nonlinear optimization. First- and second- order methods”

Performed by

Cheng-Yuan and Ma

Academic group

J4133c

Petr Chunaev *St.*

Petersburg 2022

Task# 3

Language : python

Goal : The use of first- and second-order methods (Gradient Descent, Non-linear Conjugate Gradient Descent, Newton's method and Levenberg-Marquardt algorithm) in the tasks of unconstrained nonlinear optimization

Problem : Generate random numbers $\alpha \in (0,1)$ and $\beta \in (0,1)$. Furthermore, generate the noisy data $\{x_+ , y_+ \}$, where $k = 0, ... ,100$, according to the following rule:

Y = AX + B + noise

X = k / 1000

where $\delta_+ \sim N(0,1)$ are values of a random variable with standard normal distribution. Approximate the data by the following linear and rational functions:

1. $F(x,a,b)=ax+b$ (linearapproximant), 2. $F(x, a, b) =$ (rational approximant),
by means of least squares through the numerical minimization (with precision $\varepsilon = 0.001$) of the following function:

$D(a,b) = \text{sum}(F(x_+,a,b) - y_+)^{\wedge 2}$

Theory:

Gradient descent :

Step 1 : Initialize x =3. Then, find the gradient of the function, dy/dx = 2*(x+5).
Step 2 : Move in the direction of the negative of the gradient (Why?). But wait, how much to move? For that, we require a learning rate. Let us assume the learning rate → 0.01
Step 3 : Let's perform 2 iterations of gradient descent
Step 4 : We can observe that the X value is slowly decreasing and should converge to -5 (the local minima). However, how many iterations should we perform?

Conjugate gradient descent :

1. Initialize x_0
2. Calculate $r_0 = Ax_0 - b$
3. Assign $p_0 = -r_0$
4. For k = 0, 1, 2, …:
   * calculate $\alpha_k = -r_k'p_k / p_k'Ap_k$
   * update $x_{k+1} = x_k + \alpha_k p_k$
   * calculate $r_{k+1} = Ax_{k+1} - b$

* calculate $\beta_{k+1} = r_{k+1}'Ap_k \,/\, p_k'Ap_k$
* update $p_{k+1} = -r_{k+1} + \beta_{k+1}p_k$

Newton method :

Hessian matrix = H(xk)
Gradient matrix = f(xk)

xk+1 = xk - H(xk)^-1 X f(xk)

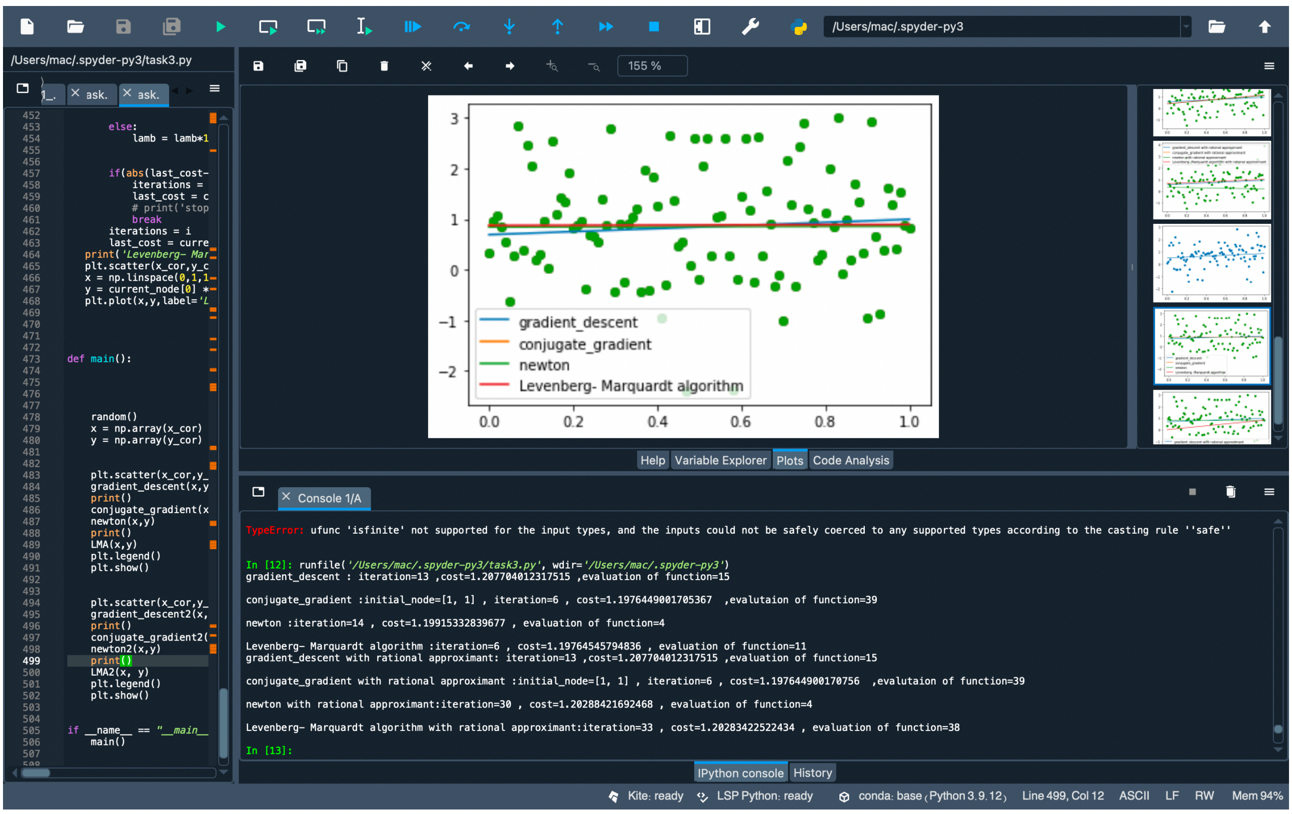
Levenberg-Marquardt algorithm :

This is one algorithm which combine Newton method and gradient descent .

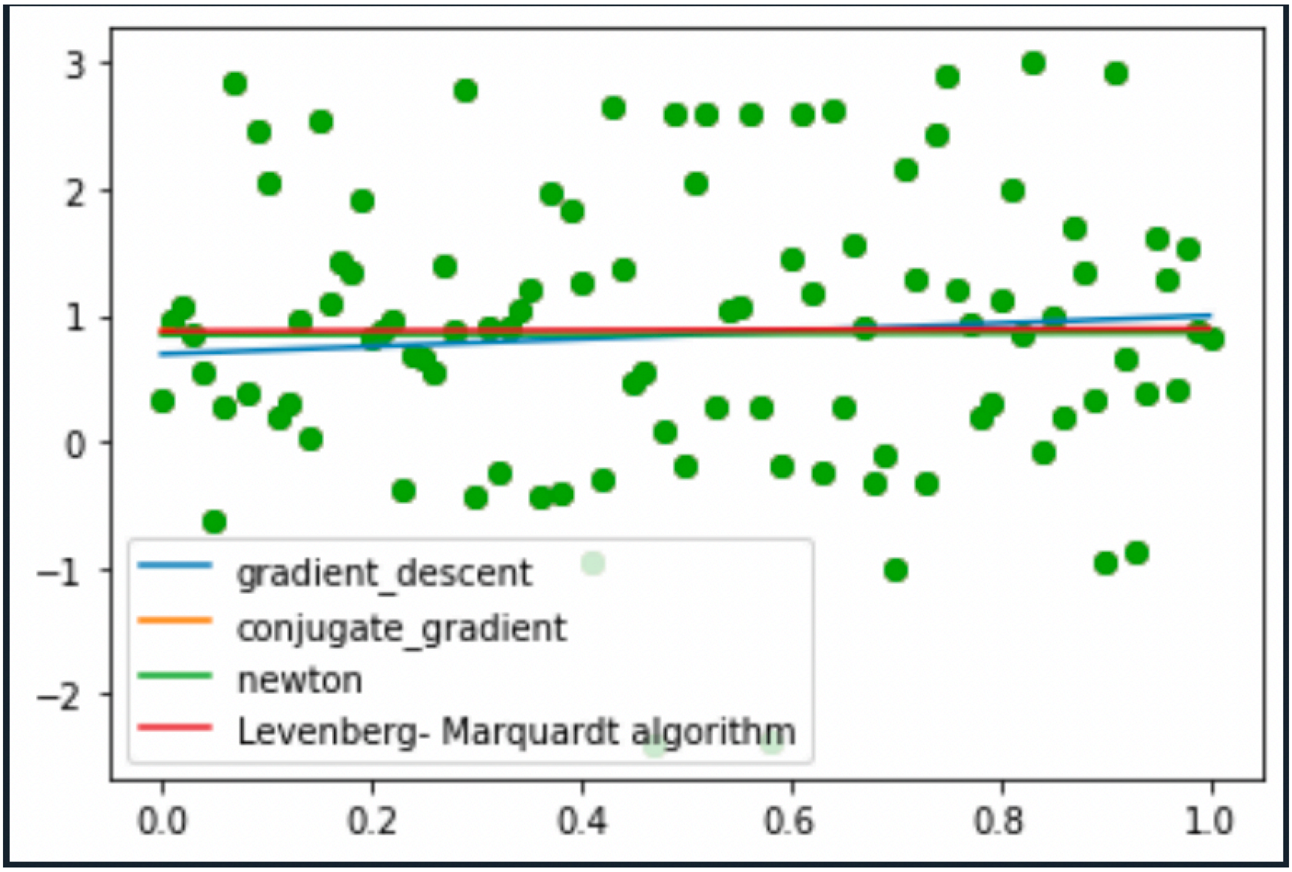
It applicate two method depends on situation , and alter lambda , a learning rate which can adjust proportion of two methods , depending on the steepness of slope

Hessian matrix = H(xk)
Gradient matrix = f(xk)
xk+1 = xk - (H(xk)+ **lambda*Identity matrix**)^-1 X f(xk)

Results



This is result for linear approximation




```
gradient_descent : iteration=13 ,cost=1.207704012317515 ,evaluation of function=15
conjugate_gradient :initial_node=[1, 1] , iteration=6 , cost=1.1976449001705367  ,evalutaion of function=39
newton :iteration=14 , cost=1.19915332839677 , evaluation of function=4
Levenberg- Marquardt algorithm :iteration=6 , cost=1.19764545794836 , evaluation of function=11
```

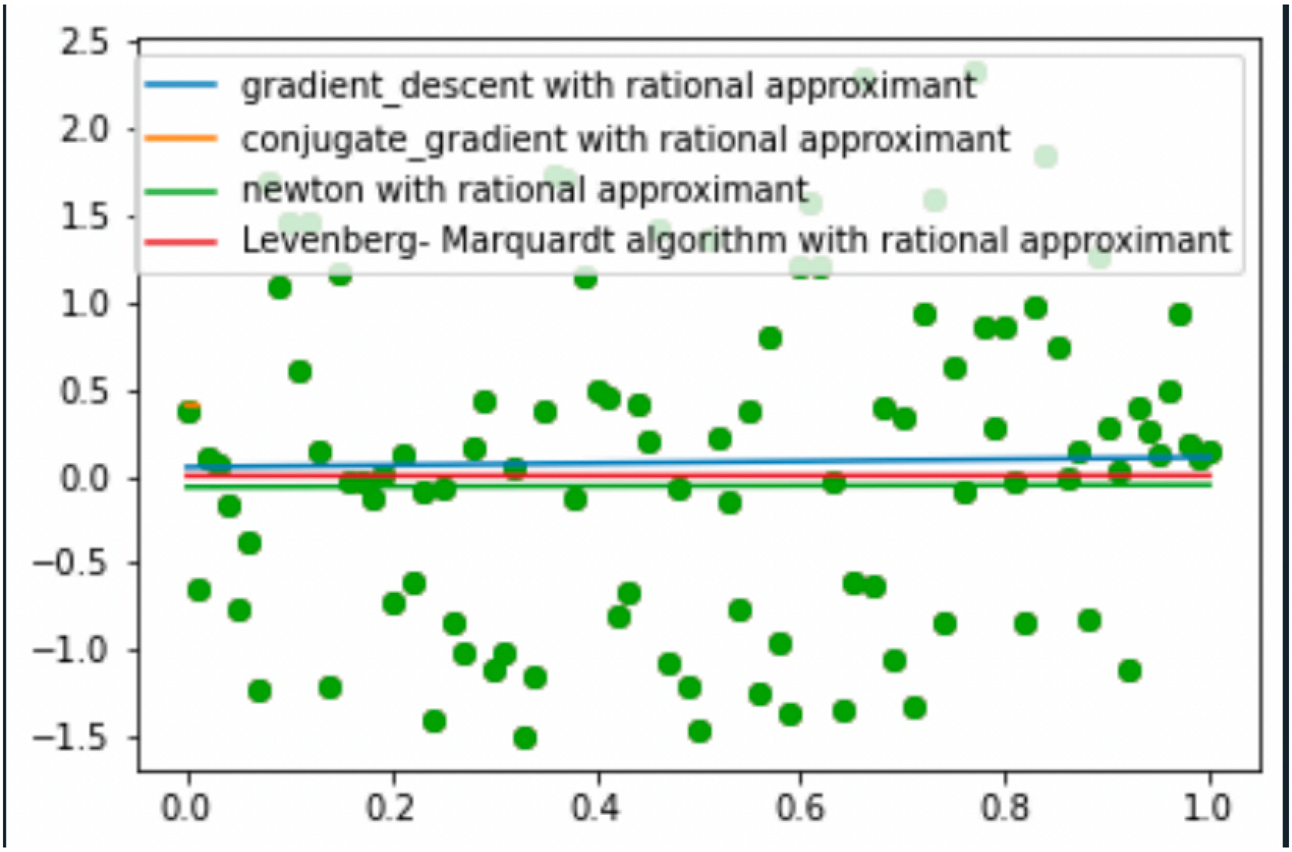
gradient_descent : iteration=13 ,cost=1.207704012317515 ,evaluation of function=15

conjugate_gradient :initial_node=[1, 1] , iteration=6 , cost=1.1976449001705367 ,evalutaion of function=39

newton :iteration=14 , cost=1.19915332839677 , evaluation of function=4

Levenberg- Marquardt algorithm :iteration=6 , cost=1.19764545794836 , evaluation of function=11

This is result for non-linear approximation




```
gradient_descent with rational approximant: iteration=13 ,cost=1.207704012317515 ,evaluation of function=15
conjugate_gradient with rational approximant :initial_node=[1, 1] , iteration=6 , cost=1.197644900170756  ,evalutaion of function=39
newton with rational approximant:iteration=30 , cost=1.20288421692468 , evaluation of function=4
Levenberg- Marquardt algorithm with rational approximant:iteration=33 , cost=1.20283422522434 , evaluation of function=38
```

gradient_descent with rational approximant : iteration=13 ,cost=1.207704012317515 ,evaluation of function=15

conjugate_gradient with rational approximant : initial_node=[1, 1] , iteration=6 , cost=1.197644900170756 ,evalutaion of function=39

newton with rational approximant : iteration=30 , cost=1.20288421692468 , evaluation of function=4

Levenberg- Marquardt algorithm with rational approximant: iteration=33 , cost=1.20283422522434 , evaluation of function=38

Conclusion

As we can see , using linear approximation , best line can almost converge to same line with all methods which we have used in analysis .

In non-linear approximation graph , all method converged to very very close line , but we can see that there is still little deviation , but as result , it is acceptable .

As I know , newton method has many shortcomings . Closer initial point to most optimized point , it is more likely to get to most optimized result . And also if its hessian matrix is not positive definite , its direction will not point in descent direction , and if it is singular matrix, it will not move at all . Also calculation of hessian matrix is very time-consuming when its size is large .

Comparing with methods in task 2 , we can see that method using derivative is highly efficient , usually can be done with few iteration .

https://github.com/MaChengYuan/task3/blob/de41cd3dc95d698533193a62abecd8fe6ebd3827/task3.py