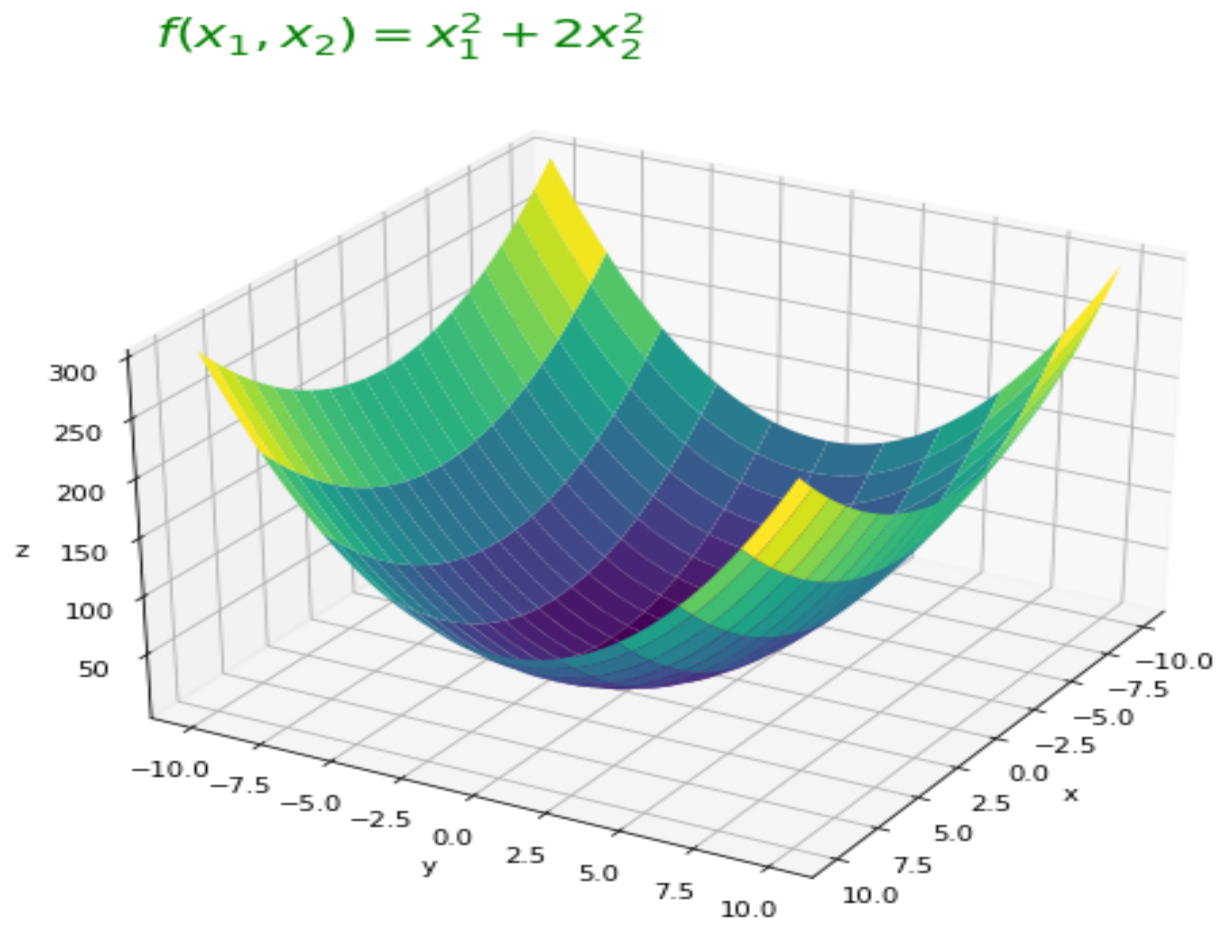
OPTAL

Report of Assignment on question-2 of midterm exam

Mahendra Nandi

16/06/2021



Question:2:

Minimize the function $f=x_1^2 + 2x_2^2$ by stochastic gradient method. To simulate a stochastic gradient, add normal (white) noise to the exact gradient at each step (this method is called perturbed GD).

*** [A]** First use piecewise constant step size $\eta(t) = \eta_i$, $t_i \le t \le t_{(i+1)}$, starting with $\eta = 0.1$, - i.e., decrease the step size whenever progress in optimization stalls. Next use two types of Dynamic Learning Rates -

*** [B] (i)** exponential decay $\eta(t) = \eta_0 e^{(-\lambda t)}$, using $\eta_0 = 0.1$, $\lambda = 0.1$

*** [B] (ii)** polynomial decay $\eta(t) = \eta_0(1 + \beta t)^{(-\infty)}$, using $\eta_0 = 0.1$, $\alpha = 0.5$. You have to find a good value for β. Compare the result of using three different step size rules in terms of convergence rate, accuracy, number of steps required etc. and show relevant plots.

*** [C]** Now suppose the function is changed to $f(x_1, x_2) = x_1^2 - 2 x_2^2$. What changes do you observe in the outcome of optimization?

Answer: To simulate a stochastic gradient, we add normal (white) noise to the exact gradient at each step (perturbed GD)

I will analyse the whole question in two major part among which

[A] the first part will contain 4 subparts having four different method to change step size and

[B] the second part is for comparison among these different methods of updating step size .

Abbreviations and variable names :

perturbed_GD_constant_lr = perturbed GD with constant learning rate

perturbed_GD_piecewise_decayed_lr = perturbed GD where learning rate is updated when function value tends to increase

perturbed_GD_exponentially_decayed_lr = perturbed GD where learning rate is updated exponentially in each iteration

perturbed_GD_polynomially_decayed_lr = perturbed GD where learning rate is updated by a polynomial equation in each iteration

eta= learning rate we have to update in different method

LR= Learning Rate (eta or η)

exp=exponentiall poly=polynomial pcwise=piecewise

max_iteration= maximum iteration allowed

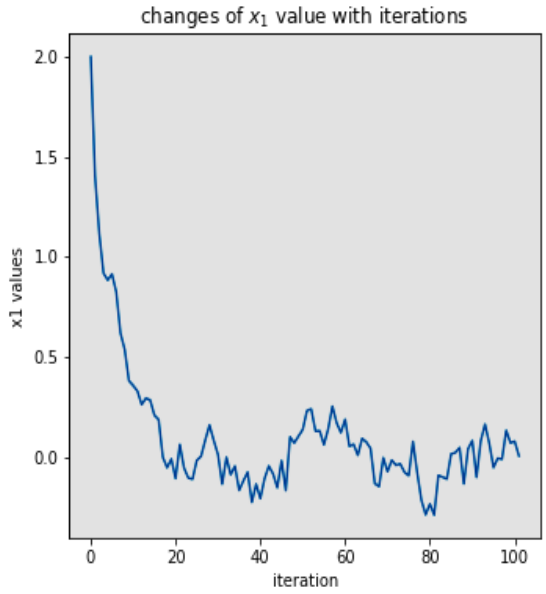
Std_Norm_Noise = True => Noise taken from standard normal distribution [N(0,1)]

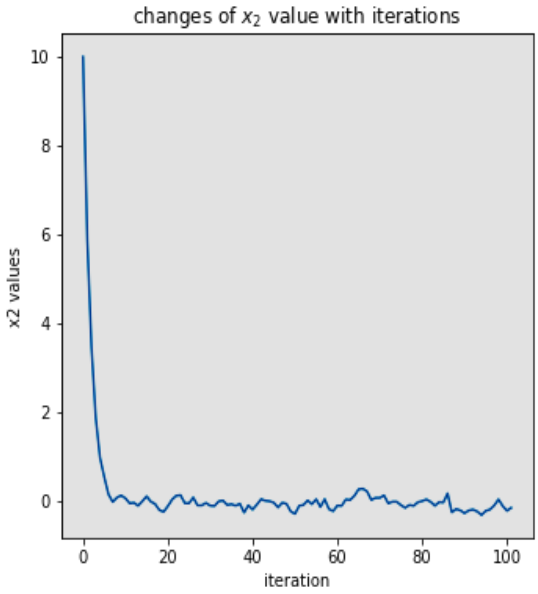
how_close_to_minima=0.001 => when the algorithm is 0.001 unit close to the minimum poin we are stopping it

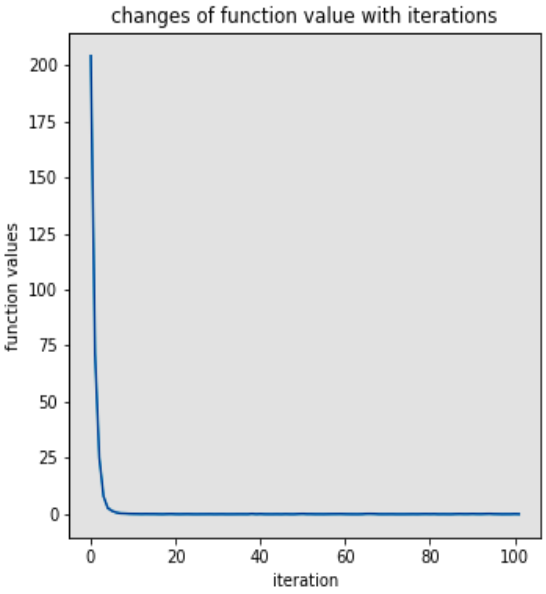
- **[A]** In question we are supposed to use three different methods (I named these as perturbed_GD_piecewise_decayed_lr , perturbed_GD_exponentially_decayed_lr , perturbed_GD_polynomially_decayed_lr) but in first part I will just initially keep the step size constant and then start updating it in other parts to show the results clearly .

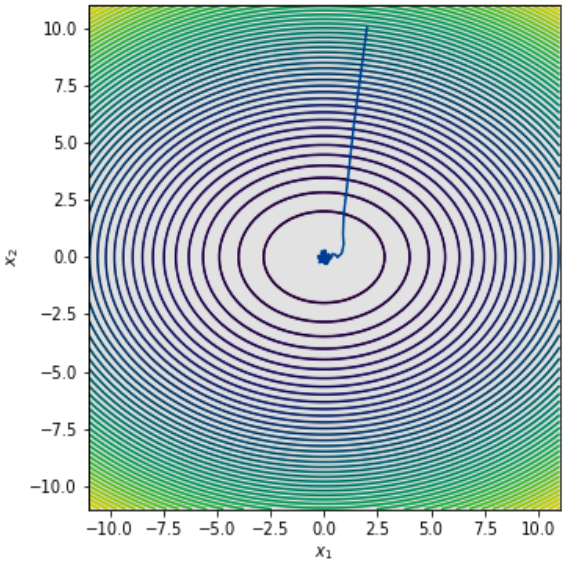
[A][a]. *Perturbed GD with constant learning rate :*

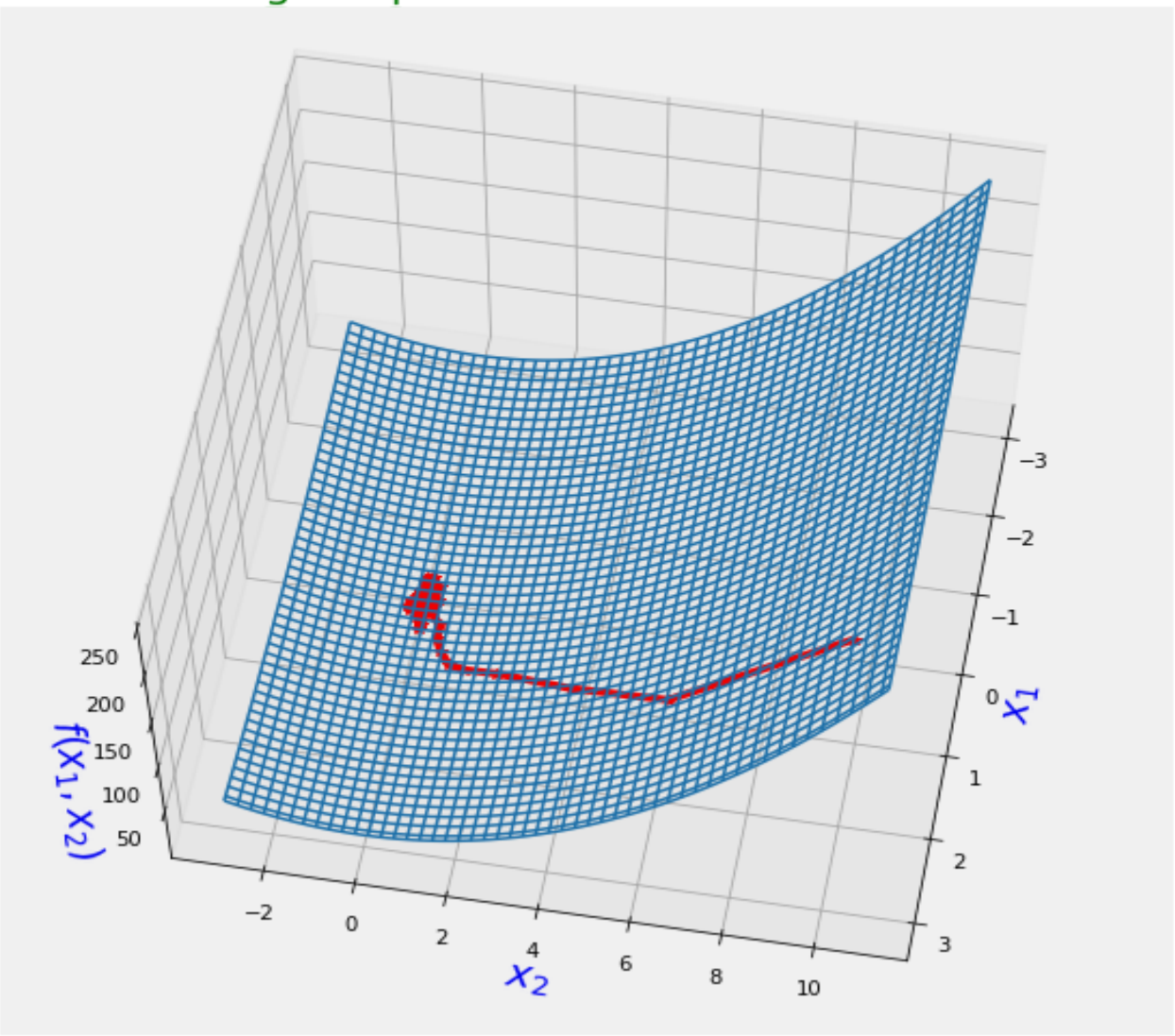
★ Here I have used the random noise from the *standard normal distribution* with the gradient and kept the learning rate constant to 0.1 and run 100 iterations starting from the point (2,10) . And the plots are as follows =>





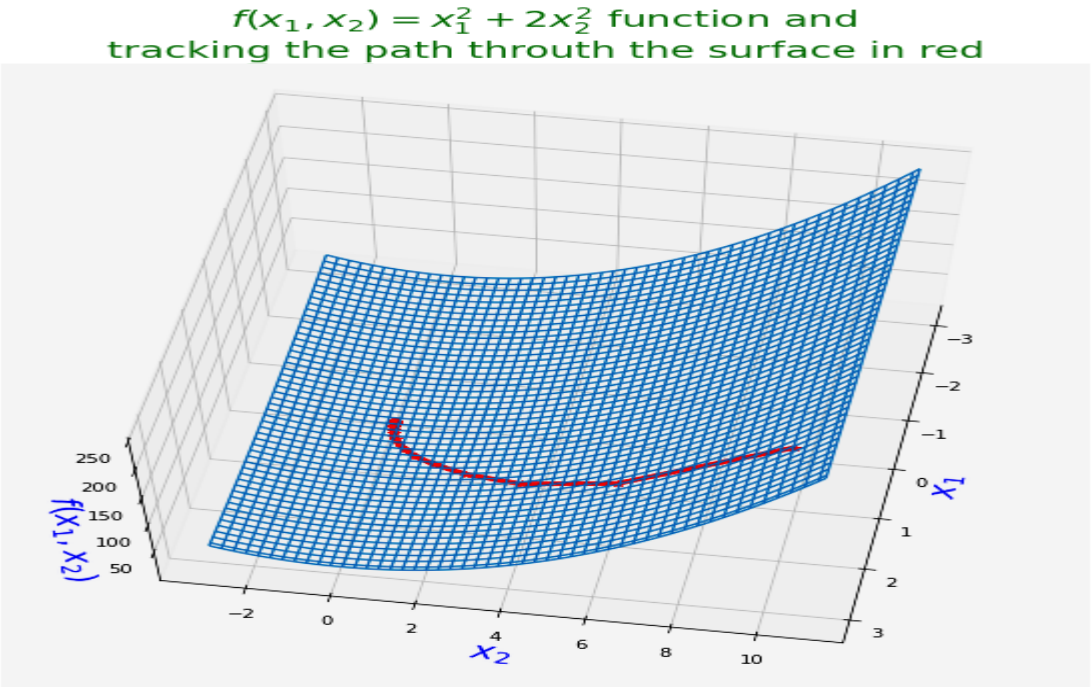






★ So, the graphs are self explanatory . We can see that when the noise is comparable to gradients when the gradient is too small [near the minimum point gradient is <<1] and then the algorithm can't stay firm to get the minimum rather it moves around the minimum. So we will see some change if we use noise value in small range ,i.e, white noise . From the graphs below , by using the `np.random.normal(mu=0, sigma=0.1, size=(1,2)` function

[which actually Draw random samples from a normal (Gaussian) distribution.] ,i.e, using noise of less positive value gives a better result for making gradients stochastic in nature.

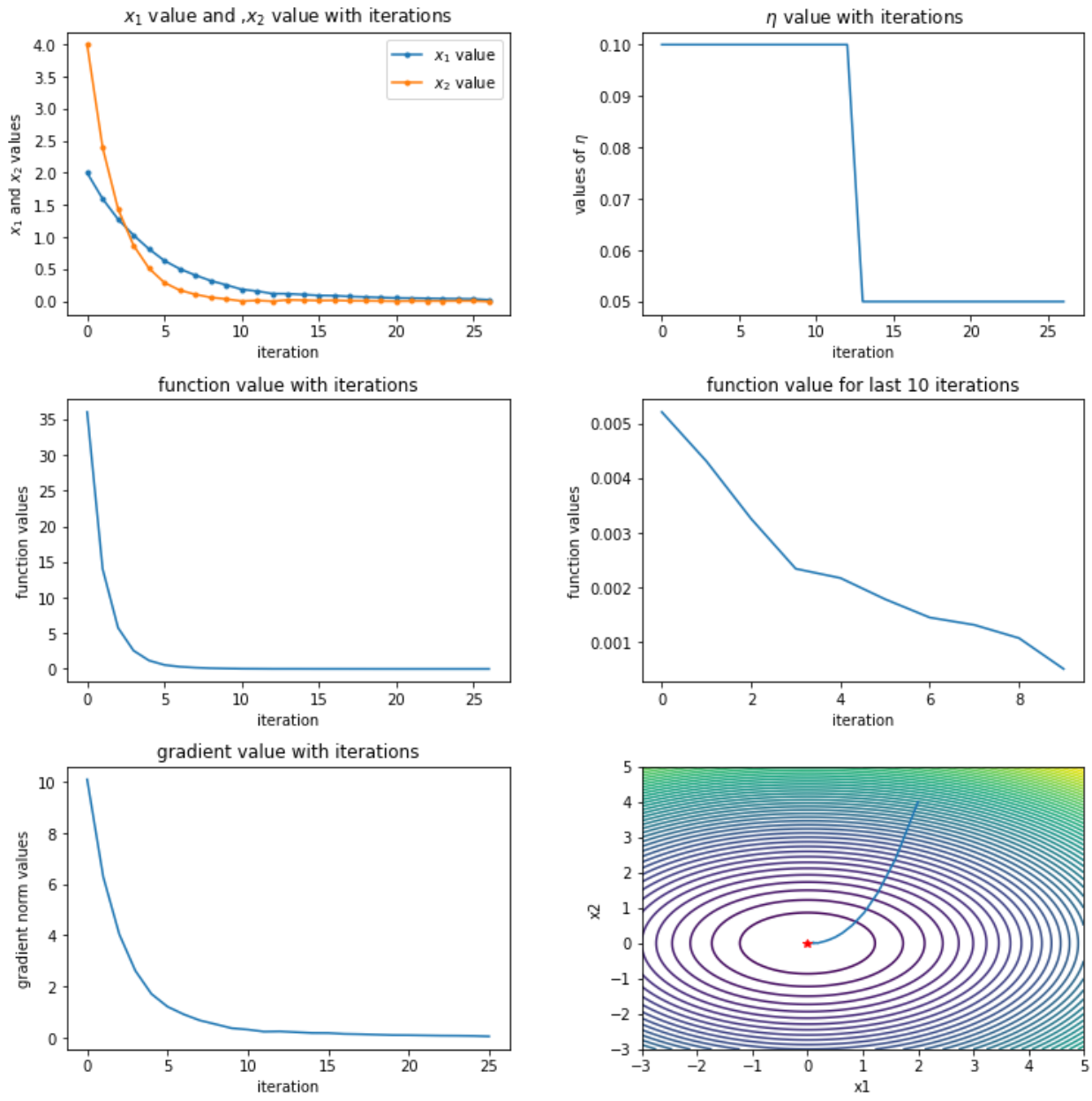


[A][b]. ***Perturbed GD with learning rate updated piecewise :***

Now we have to update LR , and from the first method mentioned in the question is to update LR piecewise. We can reduce LR in a certain interval(say after every 10 iterations)by a factor 0.5 . But here I used another condition: that when the function value continues to reduce we will not update the LR but when the function value starts to increase we will restrict this increase and so LR should be reduced by multiplying a factor(say 0.5).

- ★ First we use noise from Standard normal and run for 200 iterations with initial LR=0.1 and it still did not reach within the sphere of radius 0.001 unit and center at point having minimum value of the function [,i.e, at (0,0,0)].

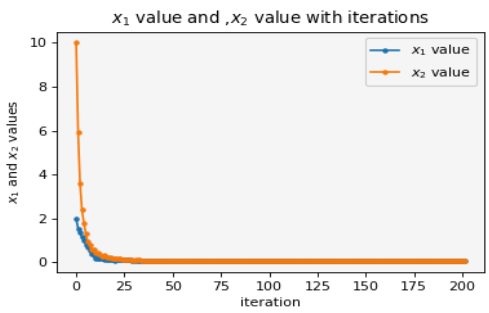
Now I use positive noise (value between 0 and 1) from normal distribution , and we will show that it took around 20 iterations only to reach the same sphere . Below are the plots describing the journey of the point (2,4).

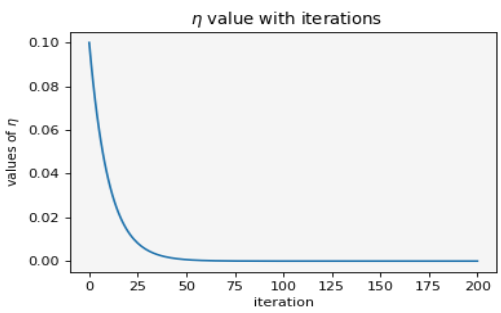


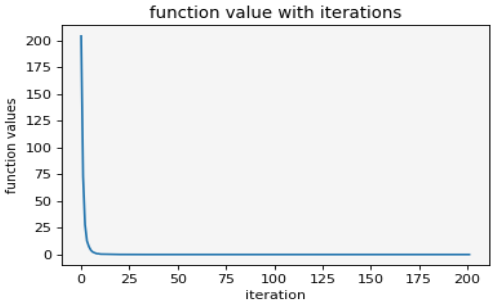
Here you can find ,most of times it didn't need to update LR while reaching near to minimum (in this figure it did updation for just once)

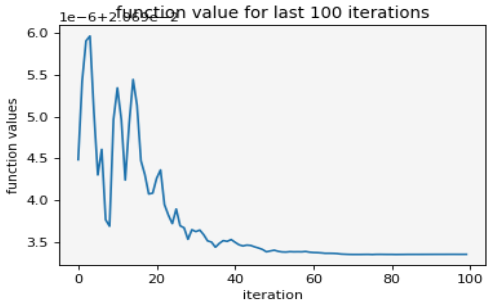
[A][c]. ***Perturbed GD with learning rate updated exponentially :***

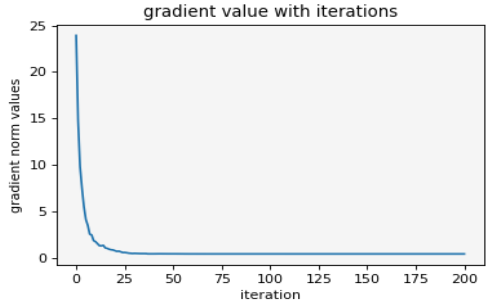
In this method of updation LR by an equation having expon- ential term. Using lamda =0.1, eta_0=0.1 ,I ran 200 epochs still it didn't reach the sphere of radius 0.001 from (2,4) , rather it stuck far from the minimum point because its LR became negligible after being exponentially decreased .

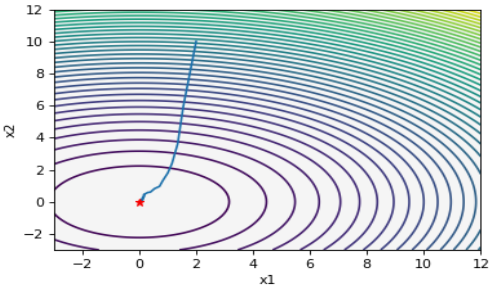


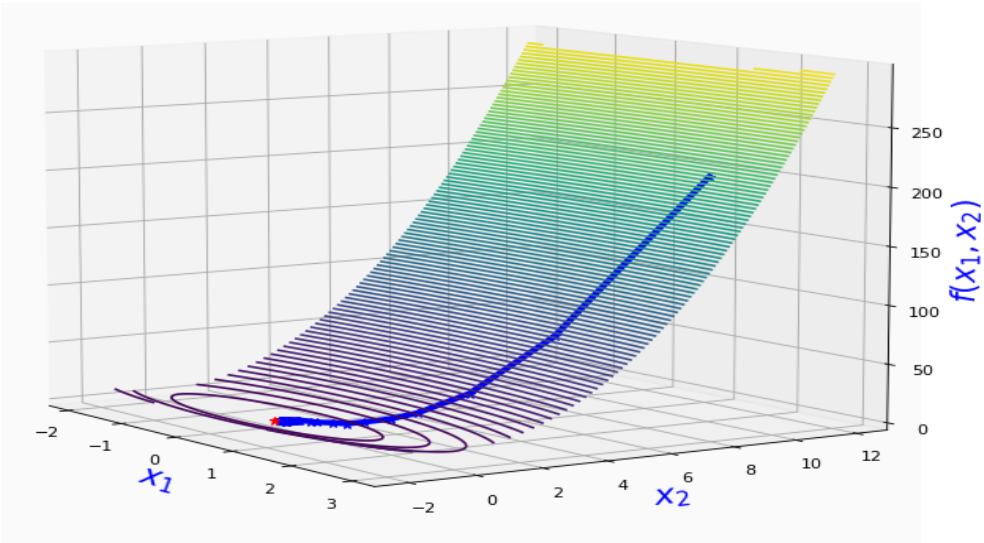




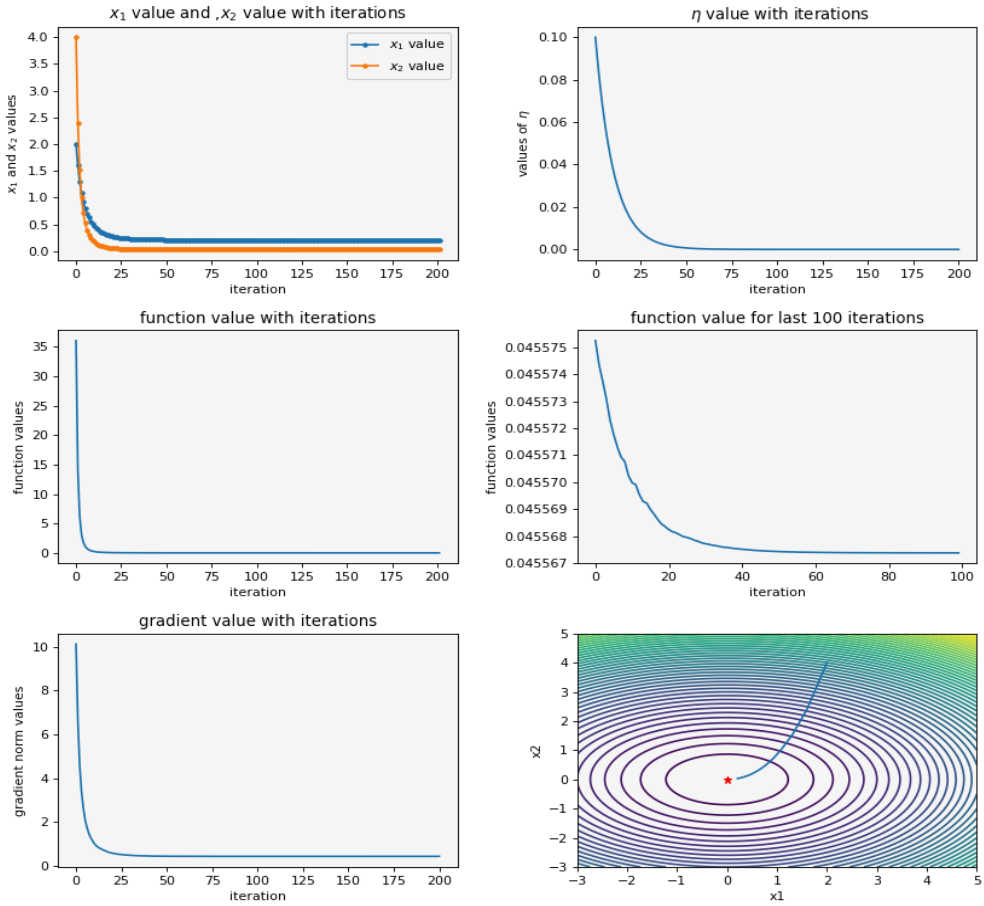


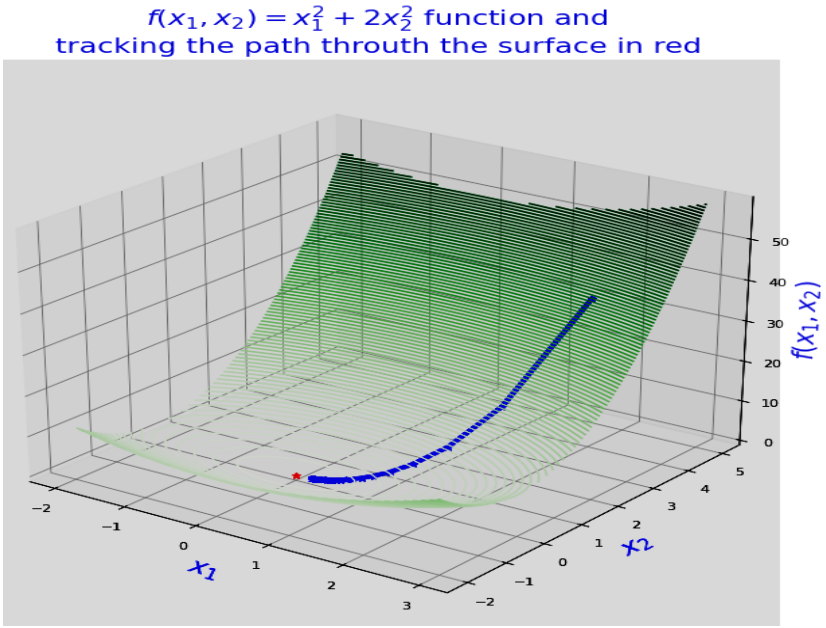






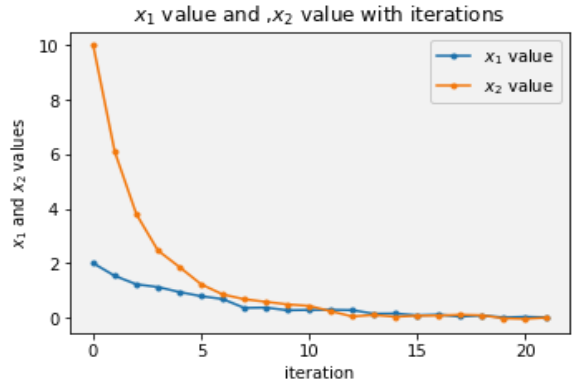
* Now using another type of noise and initial point is now (2,10), other param- eters are the same as before. The plots are self exploratory again.

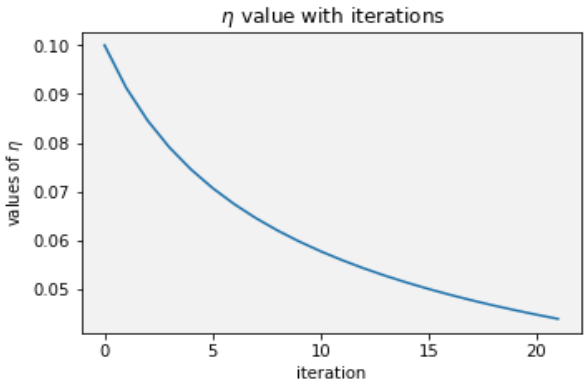


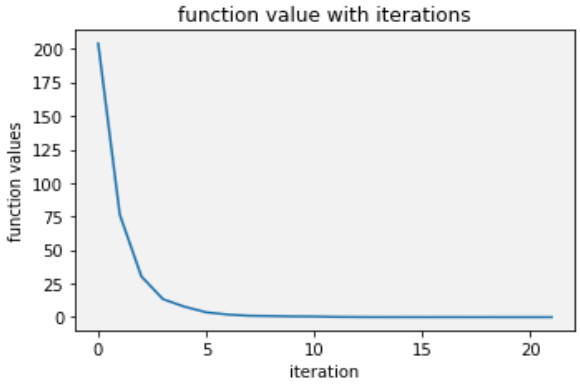


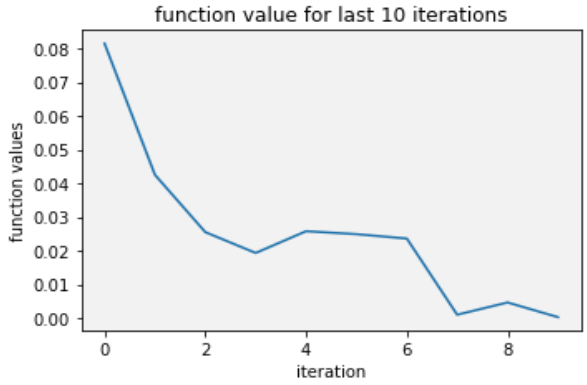
[A][d]. ***Perturbed GD with learning rate updated polynomially :***

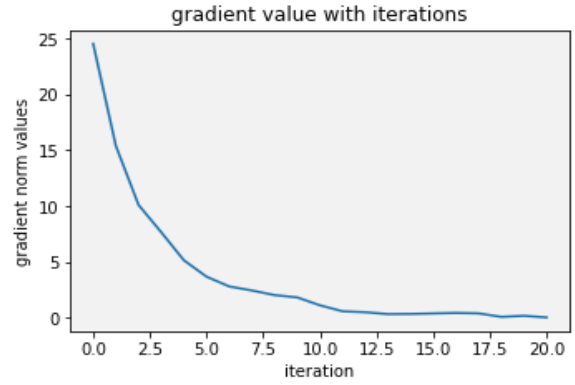
- With noise from standard normal distribution , taking beta=0.1,alpha=0.5 I ran it and it reached the sphere of radius 0.001 within 30 iterations. Though changing bita the required iteration to reach near minima is changing, but using bita in range [0.1,0.2] gives better results. The plots are shown below . This method works good to reach minima or near minima.

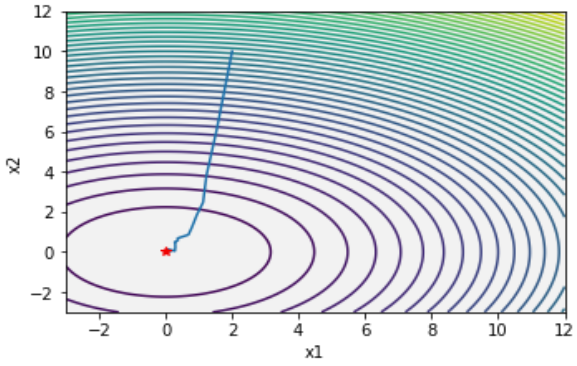


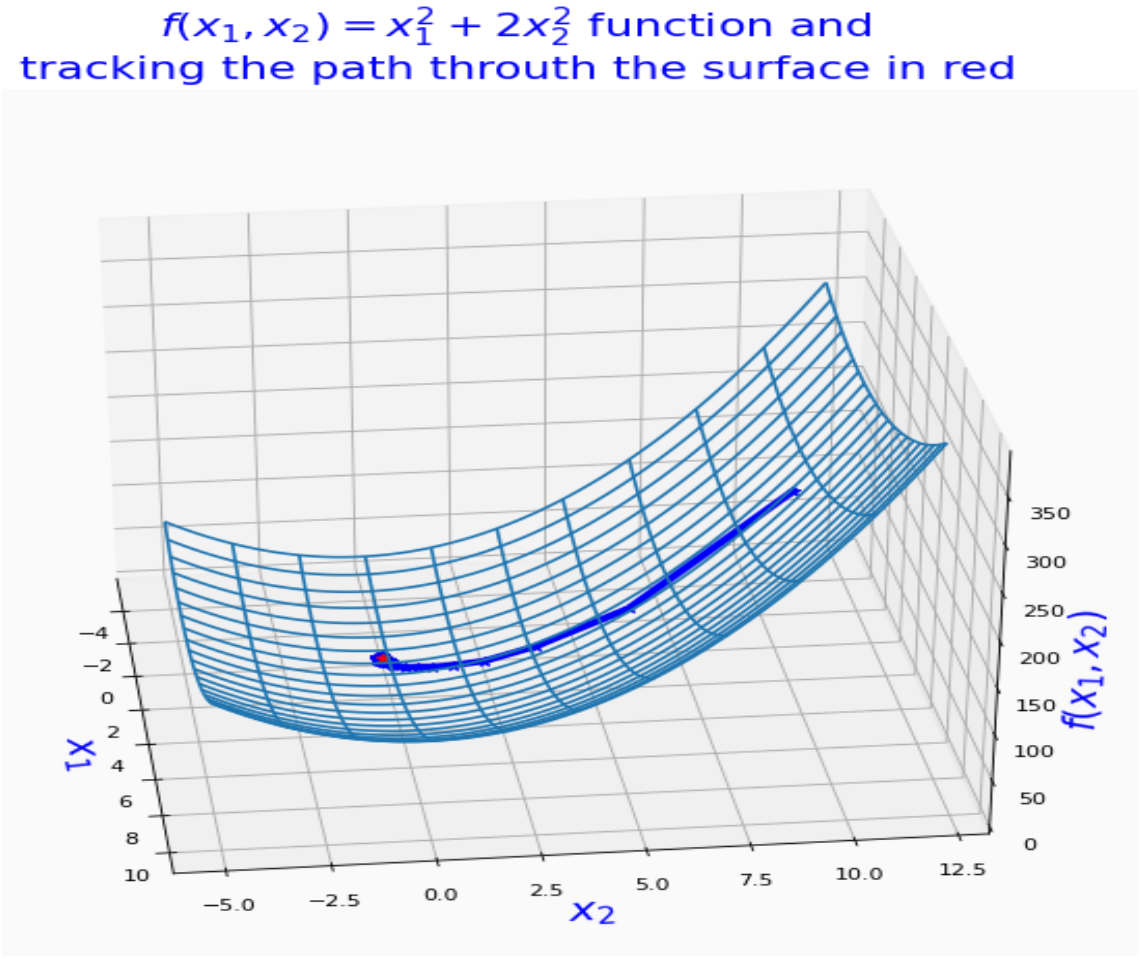




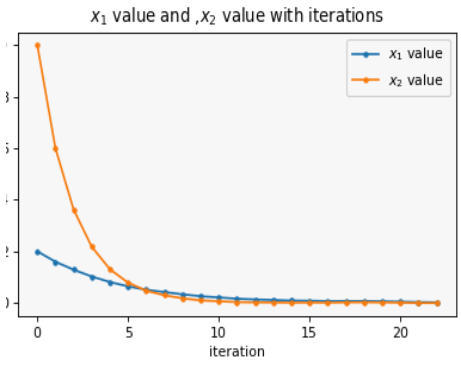


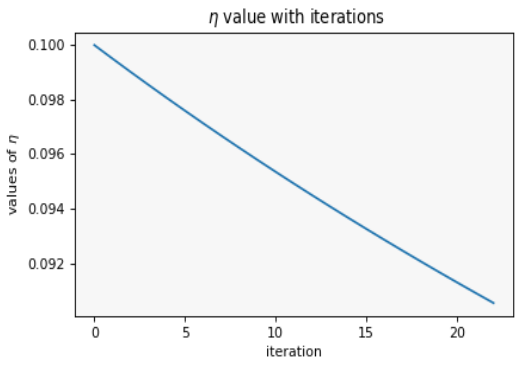


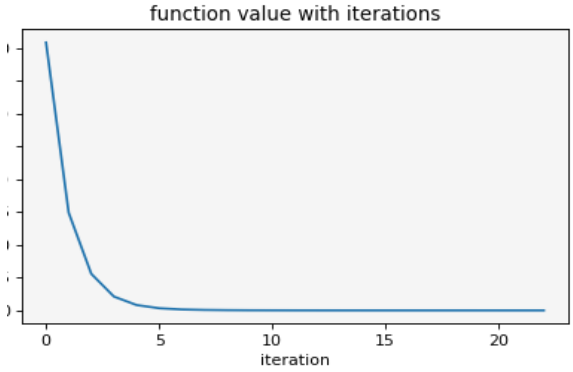


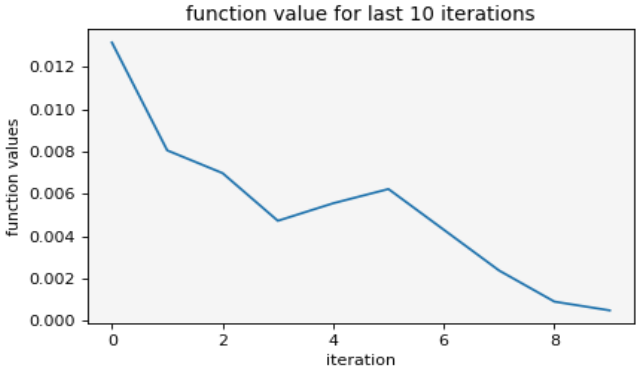


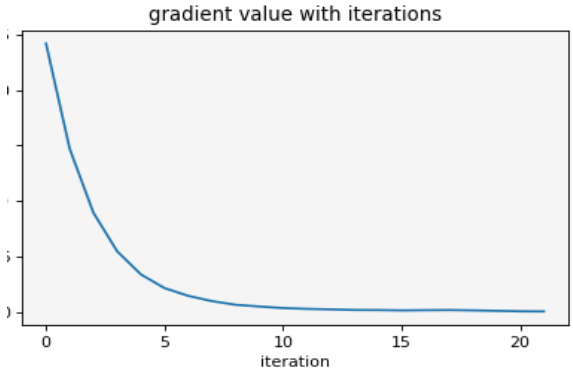
* Using another type of noise and keeping other parameters fixed I got the below results

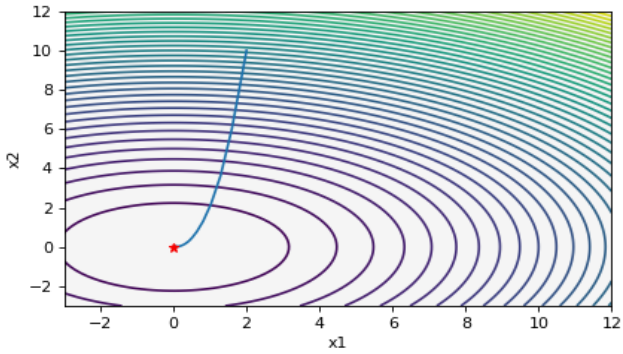





function value with iterations
iteration


function value for last 10 iterations
function values
iteration


gradient value with iterations
iteration


x2
x1

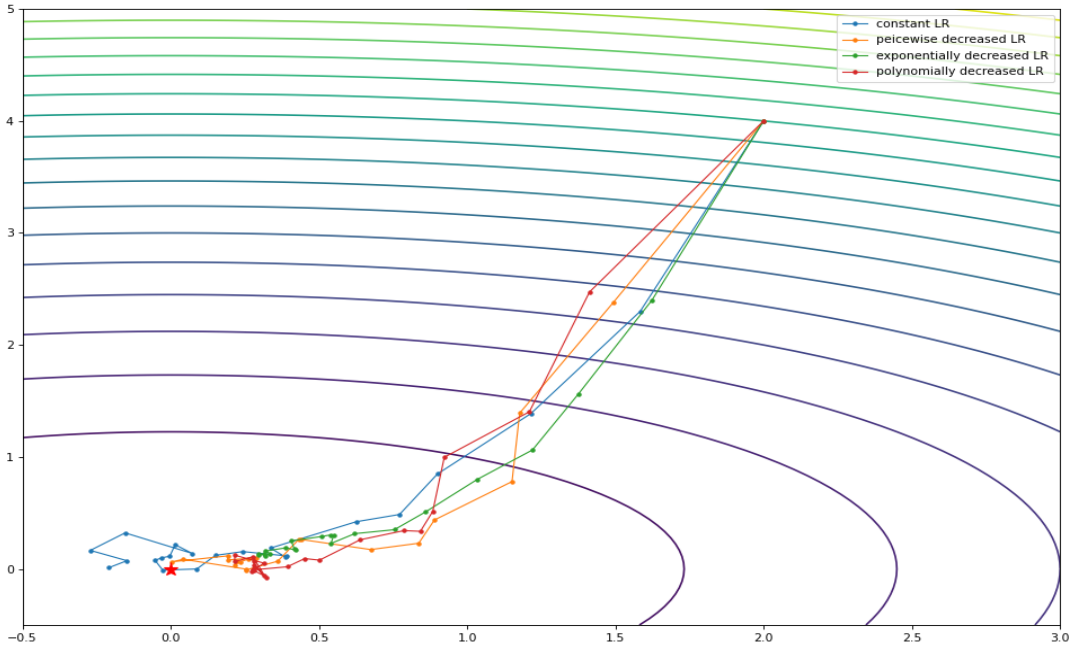
[B] Comparisons of those methods :

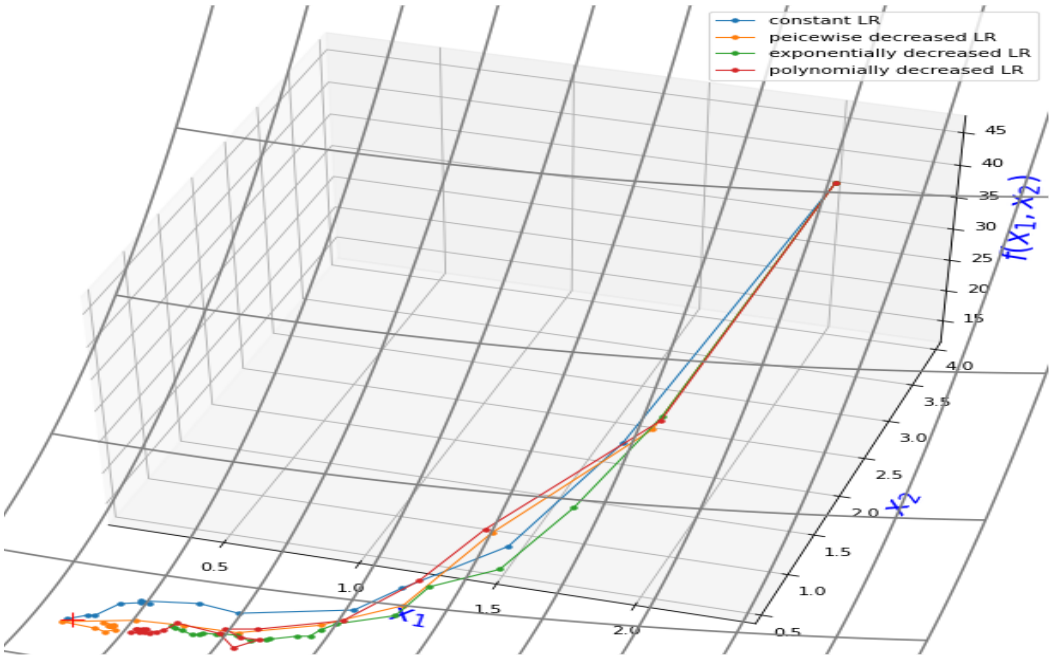
Here I will show a table comparing 4 methods regarding steps required to reach the contour having distance *r [when the noise taken from N(0,1)]* and *R [when the noise is taken from +N(0,0.1)]* . I have used three different norm distances as radius (None of these methods converge at minima). It is shown that every time in all cases polynomials one is doing better though in other cases the randomness is too high .

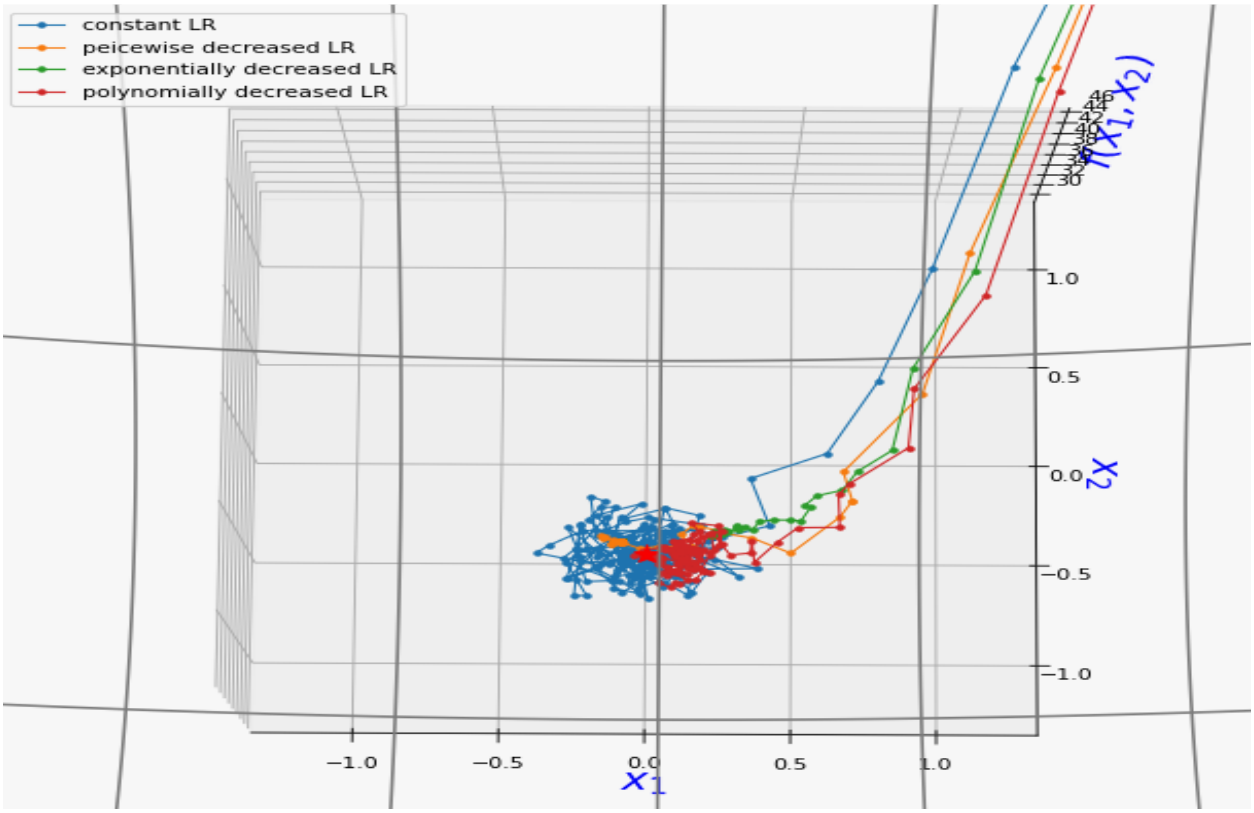
Here # means iteration needed is greater than 1500 (sometime greater than 10,000) and after the digits signifying that the number has a high variance (sometimes it is seen that at the same condition required iteration running in the range [20,10000] as it is stochastic in nature , more the noise value more the oscillating nature the trajectory has.

method	r=0.01	r=0.001	r=0.0001		R=0.01	R=0.001	R=0.0001	BEST PERFORMANCE IN
M-1	20*	80*	500*		15	20	30	R=0.01
M-2	#*	#*	#*		15	19	#*	R=0.01
M-3	#	#	#		#	#	#	r>0.1 , R>0.2
M-4	25	50	150		20	30	45	In all cases it has consistency
WINNER	M4	M4	M4		M1	M2	M1	

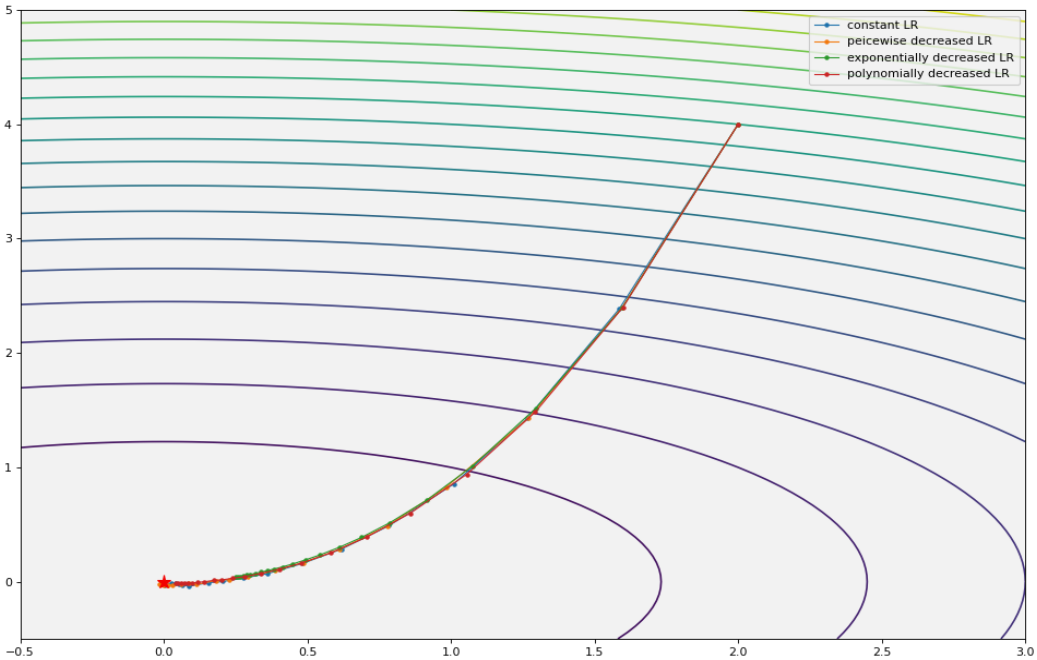
★ Below are 3 plots , the 1st one showing the 2D contour plot of the trajectory and 2nd one is also the same for the 3D with 20 iterations. Whereas I have shown what is happening if I allow more iteration . Then it is actually moving around the minimum point marked as red star(*) . **For these 3 plots noise is from N(0,1)**


constant LR
peicewise decreased LR
exponentially decreased LR
polynomially decreased LR


constant LR
peicewise decreased LR
exponentially decreased LR
polynomially decreased LR
x_1
x_2
$f(x_1, x_2)$

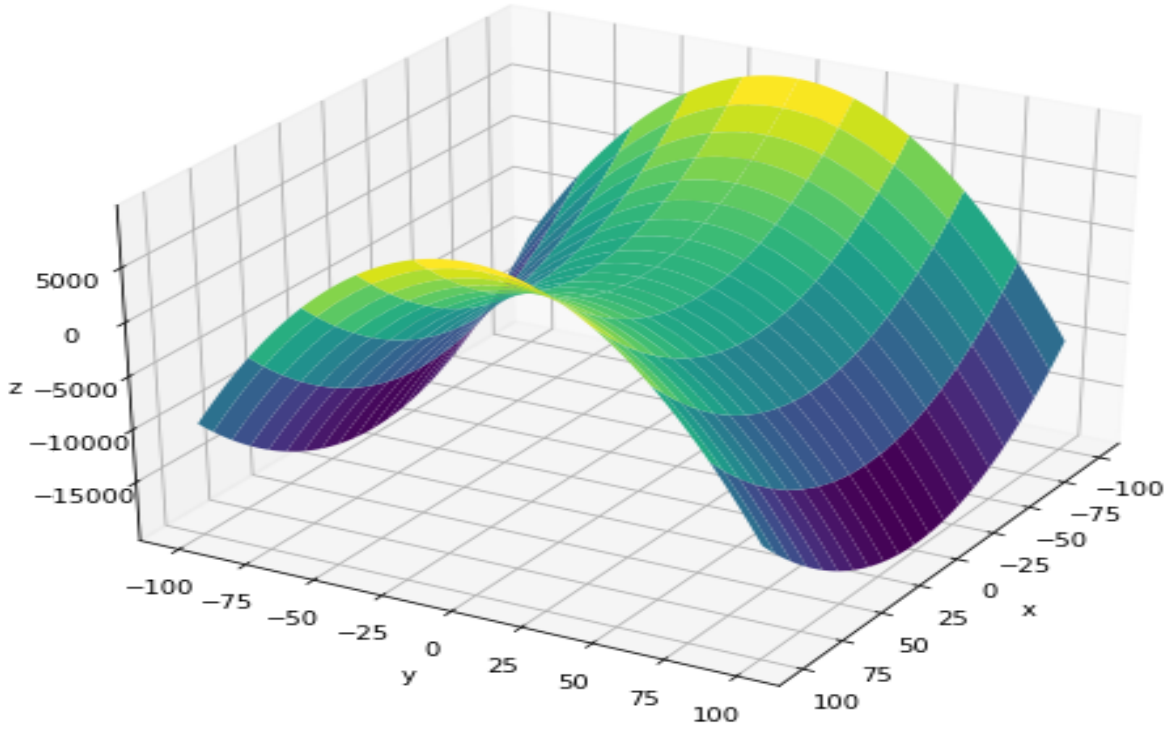


★ Here again I show if the noise is small all the methods are almost following a similar path though there is a difference in iteration taken to reach. I just showed here for 20 iterations .

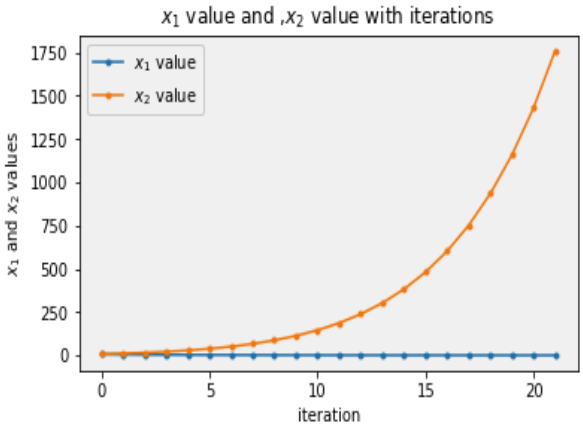


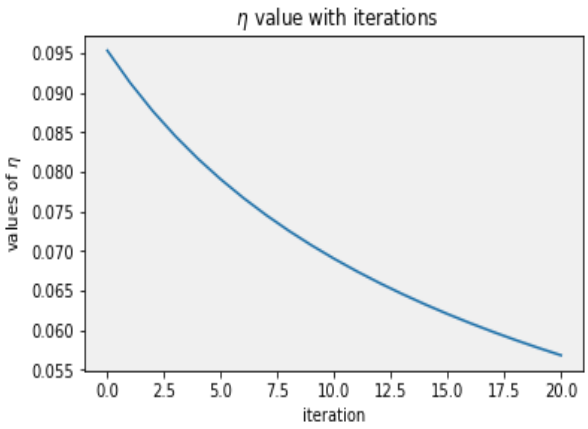
[C] If the function changed into :

$$f(x_1, x_2) = x_1^2 - 2x_2^2$$



As the function $x_1 - 2x_2^2$ is non convex and has sadel point , even if we choose a good initial point it will be deflected near the minimum . For each case the algorithm is almost behaving the same , so I am showing one of them here , the remaining explanation is the





same as above). Here it is tough to compare among these functions though. Initial point=(6,8).