# Stochastic Gradient Decent and Applications

Jing Guo

## 1 Introduction

This report mainly explores math descriptions and pseudocode of plain gradient descent, accelerated plain gradient descent, stochastic gradient descent and accelerated stochastic gradient descent. It also briefly develope derivations and convergences of asymptotic ODE and the application of calculating implied volatilities in real finance world. In section 2, we talked about the theorem of gradient descent algorithms (GD), accelerated gradient descent algorithms (NGD), stochastic gradient descent algorithms (SGD) and accelerated stochastic gradient descent algorithms and their pseudocodes. In section 4, we focused on solving some 1-dimension and 2-dimension functions by applying GD, NGD, SGD and accelerated SGD respectively. In section 5, we use SGD algorithms to solve implied volatilities of AAPL pull options. All my codes and references are listed in my github website (https://github.com/G750cloud/Capstone-Project-SGD). If you have any questions, please reach out to me any time.

## 2.1 Plain Gradient Descent

We claim multivariable function is $J(\emptyset)$. $J(\emptyset)$ is defined and differentiable in a neighborhood of a point a, then $J(\emptyset)$ decreases fastest if one goes from a in the direction of negative gradient of F(x) at a. This decreasing process can be described as $\emptyset_{n+1} = \emptyset_n - \alpha\nabla J(\emptyset)$ . Generally speaking, plain gradient descent is the method to search for the minimum point of $J(\emptyset)$, which goes from any point on $J(\emptyset)$ and the direction is $-\nabla J(\emptyset)$. In order to facilitate the explanation of accelerated GD, SGD and accelerated SGD, we introduce the linear equation fitting background to illustrate gradient descent and its pseudocode.

Notations:

N: observation scales

$X^i$: the input of the i-th observation.

$Y^i$: the lable of the i-th observation.

$H_{theta}(x^i)$: the fitting value of the i-th observation.

First, we define the equation:

$$h_\emptyset(x^i) = \sum_{k=0}^{d} \emptyset_k \, x_k^i$$

Goal: fit parameters theta.

Cost function:

$$J(\emptyset) = \frac{1}{2n}\sum_{i=1}^{n}(h_\emptyset(x^i) - y^i)^2$$

Also,

$$\frac{\partial J(\emptyset)}{\partial \emptyset_j} = \frac{1}{2n}\frac{\partial \sum_{i=1}^{n}(h_\emptyset(x^i) - y^i)^2}{\partial \emptyset_j}$$

$$= \frac{1}{2n}\frac{\partial}{\partial \emptyset_j}\sum_{i=1}^{n}(\sum_{k=0}^{d}\emptyset_k x_k^i - y^i)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k=0}^{d}\emptyset_k x_k^i - y^i\right) * \frac{\partial}{\partial \emptyset_j}(\sum_{k=0}^{d}\emptyset_k x_k^i - y^i)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k=0}^{d}\emptyset_k x_k^i - y^i\right) * x_j^i \quad (*)$$

We can transform notations $X_i$, $Y_i$ and theta to matrix forms:

$$\underline{X} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix},\ \underline{\emptyset} = \begin{pmatrix} \emptyset_1 \\ \emptyset_2 \end{pmatrix},\ \ \underline{Y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

So (*) can be rewritten as:

$$\frac{\partial J(\emptyset)}{\partial \emptyset} = \frac{1}{n}\underline{X}^T(\underline{Y} - \underline{X}\underline{\emptyset})$$

Plain gradient descent method:

$$\emptyset_{n+1} = \emptyset_n - \alpha \nabla J(\emptyset)$$

$$= \emptyset_n - \underline{\frac{\alpha}{n}} X^T(\underline{Y} - \underline{X}\underline{\emptyset})$$

Alpha is the hyperparameter of decreasing.

Pseudocode:

```
Input: X(observations), Y(Lables), threshold, $\theta$
Output: $\theta$
1: $\alpha = 0.005$
2:   for i=1:n do
3:       $error = \frac{1}{2}\sqrt{(Y - X\theta)^2}$
4:       if $error <= threshold$ then
5:           break
6:       end if
7:       $\theta = \theta - \frac{\alpha}{i}X^T(Y - X\,\theta)$
8: end for
9: Return $\theta$
```

## 2.2 Accelerated Gradient Descent Method

The background and notations of accelerated gradient descent method are the same with plain gradient descent method. Accelerated gradient descent methods are faster than plain gradient descent method, whose next a point is related to two points in its neighborhood. Mathematic descriptions are shown below:

$$\emptyset_k = \pi_{k-1} - \alpha * \frac{\partial J_\emptyset}{\partial \emptyset},$$

$$\pi_{k-1} = \emptyset_{k-1} + \frac{k-2}{k+1}(\emptyset_{k-1} - \emptyset_{k-2}),$$

$$\frac{\partial J(\emptyset)}{\partial \emptyset} = \frac{1}{n} * \underline{X}^T(\underline{Y} - \underline{X}\underline{\emptyset})$$

Pseudocode:

**Input**: X(observations), Y(Lables), threshold, $\theta_1$, $\theta_2$

**Output**: $\theta_2$

```
1: $\alpha = 0.005$
2: for i=1:n do
3:   $\pi = \theta_2 + \frac{i-2}{i+1}(\theta_2 - \theta_1)$
4:   $error = \frac{1}{2}\sqrt{(Y - X\theta_2)^2}$
5:   if $error <= threshold$ then
6:     break
7:   end if
8:   $\theta_1 = \theta_2$
9:   $\theta_2 = \pi - \frac{\alpha}{i}X^T(Y - X\theta_1)$
10: end for
11: Return $\theta_2$
```

## 2.3 Stochastic Gradient Descent Method

We also claim that all backgrounds and notations are the same with GD method in this part. The SGD method goes faster to the minimum point comparing to the GD method. In SGD method, we choose initial point a and make it decrease in any direction, not the gradient direction. It may reach to the minimum point but mostly reaches to the neighborhood of the minimum point. We just loop the part of observations, not all observations.

Pseudocode:

**Input**: X(observations), Y(Lables), threshold, $\theta$

**Output**: $\theta$

```
1: $\alpha = 0.005$
```

```
2: for i=1:m do
3:   for j=1:n do
4:     $error = \frac{1}{2}\sqrt{(Y - X\theta)^2}$
5:     if $error <= threshold$ then
6:       break
7:     end if
8:     $\theta = \theta - \frac{\alpha}{i} X^T (Y - X\,\theta)$
9:   end for
10: end for
11: Return $\theta$
```

## 2.4 Accelerated Stochastic Gradient Descent

It's same with above. All backgrounds and notations are identical with GD, accelerated GD and accelerated SGD method. The accelerated SGD method reaches minimum point faster than the previous methods, which connects 2 points in the neighborhood.

Pseudocode:

```
Input: X(observations), Y(Lables), threshold, $\theta_1$, $\theta_2$
Output: $\theta_2$
1: $\alpha = 0.005$
2: for i=1:n do
3:   $\pi = \theta_2 + \frac{i-2}{i+1}(\theta_2 - \theta_1)$
4:   $error = \frac{1}{2}\sqrt{(Y - X\theta_2)^2}$
5:   if $error <= threshold$ then
6:     break
7:   end if
8:   $\theta_1 = \theta_2$
9:   $\theta_2 = \pi - \frac{\alpha}{i} X^T (Y - X\theta_1)$
10: end for
11: Return $\theta_2$
```

## 3.1 Derivation for the Asymptotic ODE

Initially, we focus on the ODE of GD algorithms. We got the curve function $x_k = x_{k-1} - \delta\nabla g(x_{k-1})$ out of the paper[1] we discussed in classes. X(t) is the curve of points $x_k$. and $x_k \approx X(k\delta)$. The step function is $x_\delta(t) = x_k$.

So, $x_k = x_{k-1} - \delta\nabla g(x_{k-1}) \xrightarrow{yields} X(k\delta) = X((k-1)\delta) - \delta\nabla g(X((k-1)\delta))$

We then transform it into $\frac{X(k\delta)-X((k-1)\delta)}{\delta}+\nabla g\left(X\left((k-1)\delta\right)\right)=0$. Since as $\delta\rightarrow 0,\ x_{\delta}(t)=X(t)$. So we can conclude that $\dot{X(t)}+\nabla g\left(X(t)\right)=0$. $\dot{X(t)}$ is the first-order derivative of X(t) and X(0) = $x_0$.

Furthermore, we then talk more about Accelerated GD algorithms. We need to derive the first derivative of $x_k=y_{k-1}-\delta\nabla g(y_{k-1})\ and\ y_k=x_k+\frac{k-1}{k+2}(x_k-x_{k-1})$ in this section. Also, these notations are same with the GD algorithm part. We then do a little bit simple math on both of equations.

Devide by $\sqrt{\delta}$ : $\frac{x_{k+1}}{\sqrt{\delta}}=\frac{y_k}{\sqrt{\delta}}-\sqrt{\delta}\nabla g(y_k)$

Bring $y_k$ into the above equation: $\frac{x_{k+1}}{\sqrt{\delta}}=\frac{x_k}{\sqrt{\delta}}+\frac{k-1}{(k+2)\sqrt{\delta}}(x_k-x_{k-1})-\sqrt{\delta}\nabla g(x_k+\frac{k-1}{k+2}(x_k-x_{k-1}))$

So: $\frac{x_{k+1}-x_k}{\sqrt{\delta}}=\frac{k-1}{k+2}*\frac{x_k-x_{k-1}}{\sqrt{\delta}}-\sqrt{\delta}\nabla g(x_k+\frac{k-1}{k+2}(x_k-x_{k-1}))$ (1)

Claim that: $z_k=\frac{x_{k+1}-x_k}{\sqrt{\delta}}$

Take $z_k$ into the equation (1): $z_k=\frac{k-1}{k+2}*z_{k-1}-\sqrt{\delta}\nabla g(x_k+\frac{k-1}{k+2}(x_k-x_{k-1}))$

So:
$$z_{k+1}=\frac{k}{k+3}*z_k-\sqrt{\delta}\nabla g(x_{k+1}+\frac{k}{k+3}(x_{k+1}-x_k))$$

$$z_{k+1}=(1-\frac{3}{k+3})*z_k-\sqrt{\delta}\nabla g(x_{k+1}+\frac{k}{k+3}(x_{k+1}-x_k))$$

$$z_{k+1}=(1-\frac{3}{k+3})*z_k-\sqrt{\delta}\nabla g(x_k+\frac{2k+3}{k+3}(x_{k+1}-x_k))$$

Devide by $\sqrt{\delta}$: $\frac{z_{k+1}}{\sqrt{\delta}}=\frac{z_k}{\sqrt{\delta}}-\frac{3}{k+3}*\frac{z_k}{\sqrt{\delta}}-\nabla g(x_k+\frac{2k+3}{k+3}(x_{k+1}-x_k))$

$$\frac{z_{k+1}-z_k}{\sqrt{\delta}}=-\frac{3}{k+3}*\frac{z_k}{\sqrt{\delta}}-\nabla g(x_k+\frac{2k+3}{k+3}*\sqrt{\delta}*z_k)$$

As $\delta\rightarrow 0$, we conclude that:

$$\ddot{X}(t)+\frac{3}{t}\dot{X}(t)+\nabla g(X(t))=0$$

Also, $\ddot{X}(t)$ represents the second derivative and $\dot{X}(0)=0$ and $X(0)=x_0$.

## 3.2 Asymptotic Convergency Result

As we all know in reference, X(t) is the route of the point convergency and X(t) is applied to$x_k^n=x_{k-1}^n-\delta\nabla\mathcal{L}^n(x_{k-1}^n;U_n)$.

So according to the reference, we define $Vn(t)=\sqrt{n}*[Xn(t)-X(t)]$ and Xn(t), Xn(t), Vn(t) live on C([0,T]). And Vn(t) weakly converges to Gaussian process V(t), which applies to $\dot{V}(t)+[\mathcal{H}g\left(X(t)\right)]V(t)+Z\left(X(t)\right)=0,V(0)=0$. H is the the Hessian Operator.

## 4.1 Find the minimizer of $f(\theta) = \theta^2$ using the GD and N-GD

In this part, we implemented GD and N-GD method to find the minimizer of $f(\theta) = \theta^2$, whose mathematic theorems, algorithms and pseudocodes have been discussed in the previous part. We'll show results and differences out of GD algorithm and N-GD algorithm. We set the learning rate $\alpha$ be 0.0005 and stopping threshold be 0.0001, which means if the gradient of the observation point is less than 0.0001(it's almost equal to 0), we claim that this point is the minimizer. Also, we limit the maximum iteration number be 10000. For GD method, we set the initial point be 8 and then get the minimum point $\theta$ be 0.00036 after looping 99999 times. The result is pretty close to the theory solution 0. For N-GD algorithms, we set the initial point be 10 and 9.99 and get the minimum point -0.0011 after looping 1215 times. It's obvious that N-GD converges faster than Plain GD, but it's less accurate than GD.

## 4.2 Find the minimizer of $f(\theta) = {\theta_1}^2 + {\theta_2}^2$ using GD and N-GD

We worked the same process to find its minimum. We define (5,7) as the initial point of GD, learning rate $\alpha$ be 0.01, and the stopping threshold be 0.0001. Finally, we got the minimum point (0.0000289, 0.0000405) of $f(\theta) = {\theta_1}^2 + {\theta_2}^2$ after iterating 596 times and got the result (-0.000292, -0.000292) by N-GD algorithms after iterating 314 times, whose initial points are (10,10) and (9,9) with the same learning rate and thresholds. It seems like that both of results are nearly (0,0) but N-GD has iterated less times and converges faster than the Plain GD method. So basically, we believe that N-GD converged faster than GD methods in most cases, but it seems like the minimum point of N-GD moves further from the real theoretical minimum point comparing to the GD method.

## 4.3 Find the minimizer of $f(\theta) = (1-\theta_1)^2 + 100(\theta_2 - \theta_1^2)^2$ using GD and N-GD

In this part, we set (1,2) as initial point, 0.001 as learning rate and other things else all same for GD algorithms. After looping 9999 times, we find minimum point (1.011497,1.023171). Also, (1,2) and (1,2.5) are initial points for N-GD methods and the learning rate is also 0.001. The result shows that N-GD algorithms are faster than GD algorithms, which only iterates 217 times and gets the minimum point (0.999883,0.999767). As far as I'm concerned, (1.011497,1.023171) are nearly same with (0.999883,0.999767), which are both close to (1,1). So both results suggest us the theoretical minimum point is (1,1).

## 4.4 Find the minimizer of $f(\theta) = \frac{1}{2}(f_1(\theta) + f_2(\theta)), f_1(\theta) = \theta^2, f_2(\theta) = (\theta - 2)^2$ using SGD and N-SGD

Initially, we bring $f_1(\theta)$ and $f_2(\theta)$ into $f(\theta) = \frac{1}{2}(f_1(\theta) + f_2(\theta))$ and get $f(\theta) = \theta^2 - 2\theta + 2$. We then implement SGD algorithm to calculate minimum points of $f(\theta)$. We set the initial point be 8, learning rate be 0.0005, stopping thresholds be 0.0001 and maximum iteration numbers be 10000. We got the minimum point 1.000316 after iterating 1342 times. Also, we worked the same process on N-SGD algorithms whereas initial points be 9.999 and 10. After looping 425 times, the minimum point be 0.998988. It's obvious to notice that N-SGD is much faster than SGD algorithms. But comparing to Plain GD algorithms, SGD is less accurate and the minimum point out of SGD is in the neighborhood of real theoretical minimum points.

## 4.5 Find the minimizer of $f(\theta) = \frac{1}{2}(f_1(\theta) + f_2(\theta)), f_1(\theta) = -\theta^2, f_2(\theta) = 2(\theta - 1)^2$ using SGD and N-SGD

In this part, it's almost similar with part 4. The target function is $f(\theta) = \frac{1}{2}\theta^2 - 2\theta + 1$. The difference is that the initial point for SGD method is 7 and the initial points for N-SGD method are 10 and 9.999. The result for SGD is 2.0336476, which looped 2145 times and 2.00365587 for N-SGD algorithms which has iterated 325 times. That shows that N-SGD method got nearly 7 times as fast as GD methods with nearly the same result 2.00, which shows the great efficiency for N-SGD method.

## 4.6 Conclusion

From the result we have got in the previous 5 problems. It seems like that N-SGD is faster than SGD and SGD is faster than GD method with the same problem if we have to rank these three algorithms. However, there are still some drawbacks for these methods. For example, we should first check the target function whether it's convex function for the GD algorithms. Because we can only implement GD algorithms for convex function. And we should also appendix some comparing figures for these three algorithms, which makes it clearer to compare.

## 5 Calculating volatility Application by SGD Algorithm

In this section, we apply SGD algorithms to real financial world to calculate implied volatility. We aim to calibrate the BS volatility $\sigma^*$ from May 8th's call prices. Let $\{M_i: i = 1,2,\dots,n\}$ be market call option prices for AAPL (Apple.Inc) stocks with $(S, T, K_i)$. S represents initial stock prices, T means call option maturity and K points to option strike prices. Also, let $\{C_i(\sigma): i = 1,2,\dots,n\}$ be BS call prices for AAPL stocks. Our aim is to find the minimizer of the following MSE loss $f(\sigma) = \frac{1}{n}\sum_{i=1}^{n}(C_i(\sigma) - M_i)^2$. By BS equations, we collect 31 put options of AAPL, whose exercising day is May 8th and so their maturities are 1 month. We create a table to visualize these data.

| Pull Option Sticker | Initial Stock Price S($) | Option Price M($) | Strike Price K($) | Maturity T(year) | Free-risk Interest(%) |
|---|---|---|---|---|---|
| AAPL210514C00075000 | 132.32 | 55.45 | 75 | 1/12 | 0.0005 |
| AAPL210514C00080000 | 132.32 | 49.65 | 80 | 1/12 | 0.0005 |

| | | | | | |
|---|---|---|---|---|---|
| AAPL210514C00085000 | 130.14 | 44.6 | 85 | 1/12 | 0.0005 |
| AAPL210514C00090000 | 134.2 | 40.85 | 90 | 1/12 | 0.0005 |
| AAPL210514C00095000 | 131.72 | 35.75 | 95 | 1/12 | 0.0005 |
| AAPL210514C00100000 | 127.68 | 30.4 | 100 | 1/12 | 0.0005 |
| AAPL210514C00105000 | 130.14 | 25.2 | 105 | 1/12 | 0.0005 |
| AAPL210514C00107000 | 130.14 | 20.6 | 107 | 1/12 | 0.0005 |
| AAPL210514C00109000 | 132.77 | 23.6 | 109 | 1/12 | 0.0005 |
| AAPL210514C00110000 | 130.14 | 20.25 | 110 | 1/12 | 0.0005 |
| AAPL210514C00111000 | 130.14 | 19.8 | 111 | 1/12 | 0.0005 |
| AAPL210514C00112000 | 132.77 | 18.5 | 112 | 1/12 | 0.0005 |
| AAPL210514C00113000 | 130.14 | 16.15 | 113 | 1/12 | 0.0005 |
| AAPL210514C00114000 | 134.2 | 16.75 | 114 | 1/12 | 0.0005 |
| AAPL210514C00115000 | 125.69 | 15.35 | 115 | 1/12 | 0.0005 |
| AAPL210514C00116000 | 126 | 14.49 | 116 | 1/12 | 0.0005 |
| AAPL210514C00117000 | 126 | 13.45 | 117 | 1/12 | 0.0005 |
| AAPL210514C00118000 | 125.69 | 12.35 | 118 | 1/12 | 0.0005 |
| AAPL210514C00119000 | 125.69 | 11.8 | 119 | 1/12 | 0.0005 |
| AAPL210514C00120000 | 125.69 | 10.44 | 120 | 1/12 | 0.0005 |
| AAPL210514C00121000 | 125.69 | 10.15 | 121 | 1/12 | 0.0005 |
| AAPL210514C00122000 | 125.69 | 8.55 | 122 | 1/12 | 0.0005 |
| AAPL210514C00123000 | 125.69 | 7.4 | 123 | 1/12 | 0.0005 |
| AAPL210514C00124000 | 125.69 | 6.6 | 124 | 1/12 | 0.0005 |
| AAPL210514C00125000 | 125.69 | 5.55 | 125 | 1/12 | 0.0005 |
| AAPL210514C00126000 | 125.69 | 4.75 | 126 | 1/12 | 0.0005 |
| AAPL210514C00127000 | 125.69 | 3.8 | 127 | 1/12 | 0.0005 |
| AAPL210514C00128000 | 125.69 | 2.95 | 128 | 1/12 | 0.0005 |
| AAPL210514C00129000 | 125.69 | 2.32 | 129 | 1/12 | 0.0005 |
| AAPL210514C00130000 | 125.69 | 1.68 | 130 | 1/12 | 0.0005 |

We are going to use these S,K,T,M and free-risk interest to calibrate AAPL pull option price C out of BS equations. And free-risk interest is out of US 1-year treasury bond rate 0.0005. For a BS theoretical price, the only missing parameter is the volatility $\sigma$. So we can introduce a function $g$:

$$g: \sigma \rightarrow BS(\sigma, others)$$

So the implied volatility $\hat{\sigma}$ satisfies $\hat{\sigma} = g^{-1}(M)$ and the error function is $\sigma \rightarrow f(\sigma) = |g(\sigma) - M|$.

And then we develop two basic classes VallinaOption classes and Gbm classes in the code. And we define the bsm function to calculate pull option prices and error functions to calibrate the volatility $\hat{\sigma}$. Finally we define stochastic gradient descent functions we have done in the previous phases to calculate error functions. The following table shows these implied volatilities of 30 AAPL pull options.

| Pull Option Sticker | Implied volatility $\sigma$ |
|---|---|
| AAPL210514C00075000 | 0.960 |
| AAPL210514C00080000 | 0.942 |
| AAPL210514C00085000 | 0.180 |

| | |
|---|---|
| AAPL210514C00090000 | 0.169 |
| AAPL210514C00095000 | 0.061 |
| AAPL210514C00100000 | 0.899 |
| AAPL210514C00105000 | 0.338 |
| AAPL210514C00107000 | 0.442 |
| AAPL210514C00109000 | 0.061 |
| AAPL210514C00110000 | 0.301 |
| AAPL210514C00111000 | 0.422 |
| AAPL210514C00112000 | 0.061 |
| AAPL210514C00113000 | 0.059 |
| AAPL210514C00114000 | 0.069 |
| AAPL210514C00115000 | 0.652 |
| AAPL210514C00116000 | 0.617 |
| AAPL210514C00117000 | 0.585 |
| AAPL210514C00118000 | 0.565 |
| AAPL210514C00119000 | 0.569 |
| AAPL210514C00120000 | 0.512 |
| AAPL210514C00121000 | 0.534 |
| AAPL210514C00122000 | 0.459 |
| AAPL210514C00123000 | 0.416 |
| AAPL210514C00124000 | 0.398 |
| AAPL210514C00125000 | 0.360 |
| AAPL210514C00126000 | 0.338 |
| AAPL210514C00127000 | 0.304 |
| AAPL210514C00128000 | 0.274 |
| AAPL210514C00129000 | 0.255 |
| AAPL210514C00130000 | 0.232 |
| Average | 0.401 |

Finally, we make an average on every pull options on May 8th and get the volatility 0.401. We compared this implied volatility to the volatility on Yahoo finance, which is around 1.23. That shows SGD method may lead to the heuristic point and it’s unstable in some cases. This minor mistake is also likely from the wrong data source.

## References:

**[1]** Yazhen Wang, Shang Wu, Asymptotic Analysis via Stochastic Differential Equations of Gradient Descent Algorithms in Statistical and Computational Paradigms, Journal of Machine learning research 2020.

**[2]** https://finance.yahoo.com/quote/AAPL210514C00090000?p=AAPL210514C00090000

**[3]** https://github.com/G750cloud/20s_ma573/blob/master/src/20imp_vol_v01.ipynb

**[4]** https://www.macrotrends.net/2492/1-year-treasury-rate-yield-chart