Final Project

MATH-124A: Optimization
Professor Tyler Maunu

Billy Yu
Keer Xu
Pengqiu Li

December 7, 2022

Contents

In this project, we focus on the adaptive gradient method presented in the following paper:

- Rie Johnson and Tong Zhang. "Accelerating stochastic gradient descent using predictive variance reduction." Advances in neural information processing system 26 (2013).

1 Introduction

The optimization problem that we often encounter has the form

$$\min P(w),\ P(w) := \frac{1}{n}\sum_{i=1}^{n}\psi_i(w)$$

In MATH-124A class, we learned about two methods to find the optimal solution: full gradient descent (FGD) and stochastic gradient descent (SGD).

FGD can be described by the following update rule for $t = 1, 2, ...$

$$w^{(t)} = w^{(t-1)} - \frac{\eta_t}{n}\sum_{i=1}^{n}\nabla\psi_i(w^{(t-1)})$$

The full gradient descent takes the full gradient of the objective function at each iteration, providing fast convergence in terms of iteration number. However, taking full gradient of the objective function can be expensive, especially when n is large, slowing convergence in terms of time.

A modification that attempts to alleviate the computational cost is SGD. At each iteration $t = 1, 2, ...$, we randomly pick i_t from the set $1, ..., n$ and have the following update rule:

$$w^{(t)} = w^{(t-1)} - \eta_t\nabla\psi_{i_t}(w^{(t-1)})$$

At each iteration, we only calculate the one gradient of $\psi_i(w^{(t-1)})$, drastically reducing computational cost compared to FGD.

Nonetheless, as Johnson and Zhang point out in their paper, the randomness inherent in SGD results in variance, and a large variance would slow down the convergence. As a remedy to this drawback of SGD, the paper introduces a method named *stochastic variance reduced gradient* (SVRG).

In SGD, the learning rate η_t has to decay to zero to ensure convergence, reducing convergence rate; in SVRG method, we make a modification and do not need to do so. The procedure of SVRG is summarized in Figure 1. Different from SGD, we fix $\tilde{w}$ to be the same in m iterations and use it in the randomly chosen $\nabla\psi_{i_t}$ function in each of the m iterations. Only after the m iterations do we update $\tilde{w}$. In this way, the learning rate η_t does not have to decay, and thus convergence would be faster. Note that there are two ways to update $\tilde{w}$ as shown in Figure 1, and we use option II throughout our project.

Procedure SVRG

Parameters update frequency m and learning rate η
Initialize $\tilde{w}_0$
Iterate: for $s = 1, 2, \ldots$
  $\tilde{w} = \tilde{w}_{s-1}$
  $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \nabla\psi_i(\tilde{w})$
  $w_0 = \tilde{w}$
  Iterate: for $t = 1, 2, \ldots, m$
    Randomly pick $i_t \in \{1, \ldots, n\}$ and update weight
      $w_t = w_{t-1} - \eta(\nabla\psi_{i_t}(w_{t-1}) - \nabla\psi_{i_t}(\tilde{w}) + \tilde{\mu})$
  end
  option I: set $\tilde{w}_s = w_m$
  option II: set $\tilde{w}_s = w_t$ for randomly chosen $t \in \{0, \ldots, m-1\}$
end

Figure 1: The procedure of SVRG

In our project, we will conduct experiments on SVRG to evaluate its performance. Throughout the project, we fix our objective function to be that in the linear regression problem:

$$P(w) := \frac{1}{n}\sum_{i=1}^{n}(x_i^T w - y_i)$$

This function is convenient because we already know the analytical optimal solution

$$w^* = (X^T X)^{-1} X^T y$$

and can easily compare empirical results with the theoretical result.

Our project consists of four experiments on SVRG:

Conditioning Compare the rate of convergence across a range of conditioning of X.

Algorithm parameter Evaluate the effect of algorithm parameter choice on convergence.

Data parameter Assess convergence with respect to data parameters.

Comparison with other methods Compare the convergence speed of SVRG with that of FGD and SGD.

The results of these experiments are detailed in the next section.

2 Experiments

2.1 Conditioning

To investigate the behavior of the SVRG under different initial conditioning of the least square matrix X, I used the numpy package in python to change the standard deviation σ

of each entry of the matrix X from $\sigma = 1$ to $\sigma = 5$. Next, we wrote a function by ourselves to calculate the smoothness constant $L$and strong convexity constant γ defined as follow

$$\begin{cases} \psi_i(w) - \psi_i(w') - 0.5L||w - w'||_2 \leq \nabla\psi_i(w')^T(w - w') \\ P(w) - P(w') - 0.5\gamma||w - w'||_2 \geq \nabla P(w')^T(w - w') \end{cases} \tag{1}$$

The function uses SVD to decompose $\nabla^2\psi_i$ and $\nabla^2 P$ because for positive-semidefinite matrices, singular values are the same as eigenvalues. Then, we used the following values for the parameters:

1. X has dimension 100×10
2. iteration is 100
3. update frequency m is 200
4. total updates is $200 \times 100 = 20,000$
5. step size $\eta = 0.001$

After running the SVRG, we obtained the following graph.

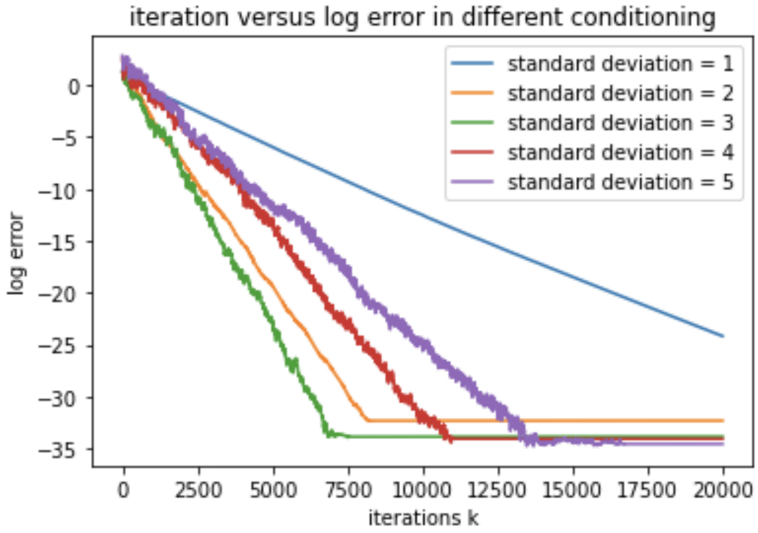

Figure 2: Log-error versus iteration number with various standard deviation of entries in X in SVRG

From the graph, we can see that for all σ I chose, the algorithm converges to the optimal point. When $\sigma = 1$, the speed of convergence is the slowest. When $\sigma = 3$, the speed of convergence is the fastest. There isn't a clear pattern that helps to predict the convergence speed if we choose other standard deviation values, but we can conclude that when σ is in the range $[1, 5]$, SVRG is a stable algorithm.

2.2 Algorithm parameter

As shown in Figure 1, there are two algorithm parameters that we can change in SVRG. One is the update frequency m; the other is the learning rate η. In this section, we experiment on how changing these algorithm parameters may affect convergence.

To ensure valid comparison, we control the data set X and y throughout this experiment. We generate the data set using the following procedure:

1. Generate a $X \in \mathbb{R}^{100\times 10}$, whose entries are $N(0,1)$.
2. Let $\beta = (1, ..., 1)^T \in \mathbb{R}^{10}$.
3. Generate $\epsilon \in \mathbb{R}^{20}$, whose entries are $N(0,1)$.
4. Let $y = X\beta + \epsilon$.

Update frequency m

First, we control the learning rate η to see how changing update frequency m may affect convergence.

In particular, we fix $\eta = 0.01$ and vary m in the set $\{50, 500, 1000, 2000, 5000, 10000\}$. For each m, we run SVRG with $s \times m = 100,000$ total iterations. Then, we plot the log-error $\log(\| \theta^{(k)} - \hat{\theta} \|)$ versus iteration number k.

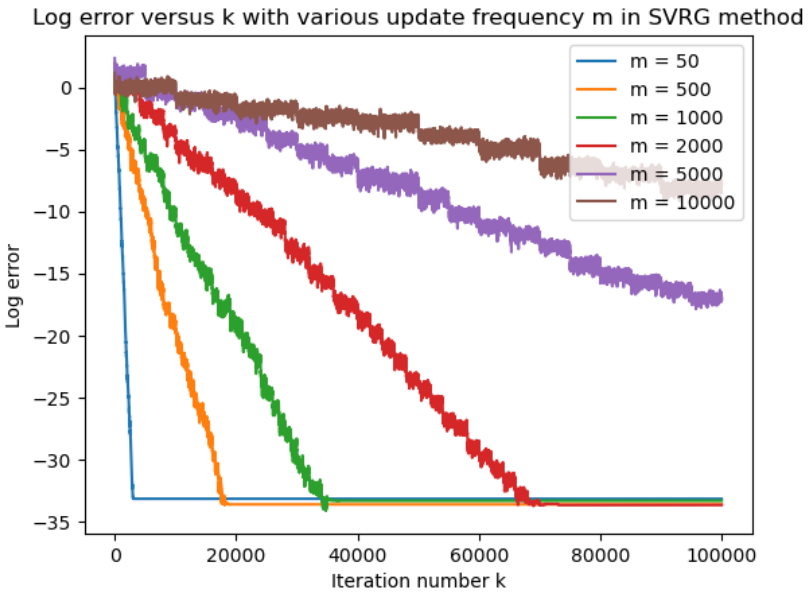

Figure 3: Log-error versus iteration number with various update frequency m in SVRG

As shown in Figure 3, as the update frequency m increases, SVRG converges slower. When $m = 50$ and $m = 500$, the method converges within the first 20,000 iterations. In comparison, for $m = 2000$, it does not converge even with 60,000 iterations. For $m = 5000$ and $m = 10000$, it does not seem to converge within the first 100,000 iterations, yet appears to be on track to

converge afterwards (the behavior is not captured in the figure because we only run SVRG with 100,000 total iterations).

Learning rate η

Next, we control the update frequency m to see how changing learning rate η may affect convergence.

In particular, we fix $m = 2000$ and vary η in the set $\{0.001, 0.005, 0.01, 0.02, 0.05\}$. For each η, we run SVRG with $s \times m = 100,000$ total iterations. Then, we plot the log-error $\log(\| \theta^{(k)} - \hat{\theta} \|)$ versus iteration number k.

Figure 4: Log-error versus iteration number with various learning rate η in SVRG

As shown in Figure 4, as the learning rate η increases, SVRG converges slower (except $\eta = 0.05$, where the method diverges). When $\eta = 0.001$ and $\eta = 0.005$, the method converges within the first 60,000 iterations. In comparison, for $\eta = 0.02$, it does not seem to converge within the first 100,000 iterations, yet appears to be on track to converge afterwards (the behavior is not captured in the figure because we only run SVRG with 100,000 total iterations). When $\eta = 0.05$, the method diverges.

2.3 Data parameter

For the third experiment, we have two data parameters to control: number of functions n and number of parameters p. In this experiment, we fix the learning rate to be at 0.005. The experiment results are outlined below.

Number of functions $n = 10$

First, we set the number of functions to $n = 10$, and plot four groups corresponding to four different numbers of parameters $p = 20, 40, 60, 80$ (see Figure 5).

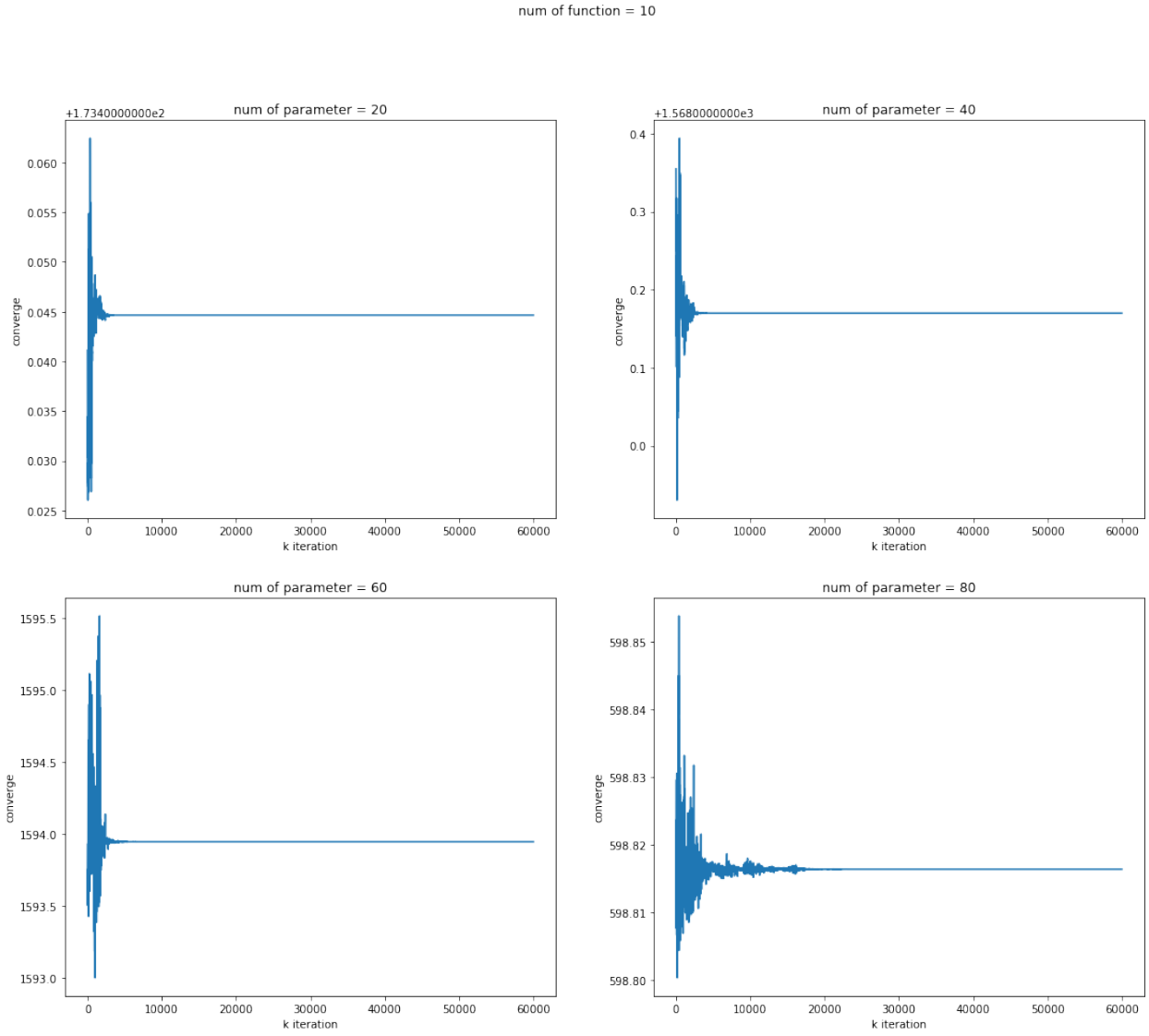

Figure 5: $n = 10$, $p = 20, 40, 60, 80$

As shown in Figure 5, when $n = 10$ and $p = 20, 40, 60, 80$, none of the graph converges, and with the increment of the number of parameters, the graph displays a greater variance. This observation makes sense since with a smaller number of parameters, the column space is smaller, which will lead to smaller variance and less divergence. And at the same time, with the increase of the number of parameters, the divergence become more observable.

Number of functions $n = 100$

Next, we set the number of functions to $n = 100$, and plot four groups corresponding to four different numbers of parameters $p = 20, 40, 60, 80$ (see Figure 6).

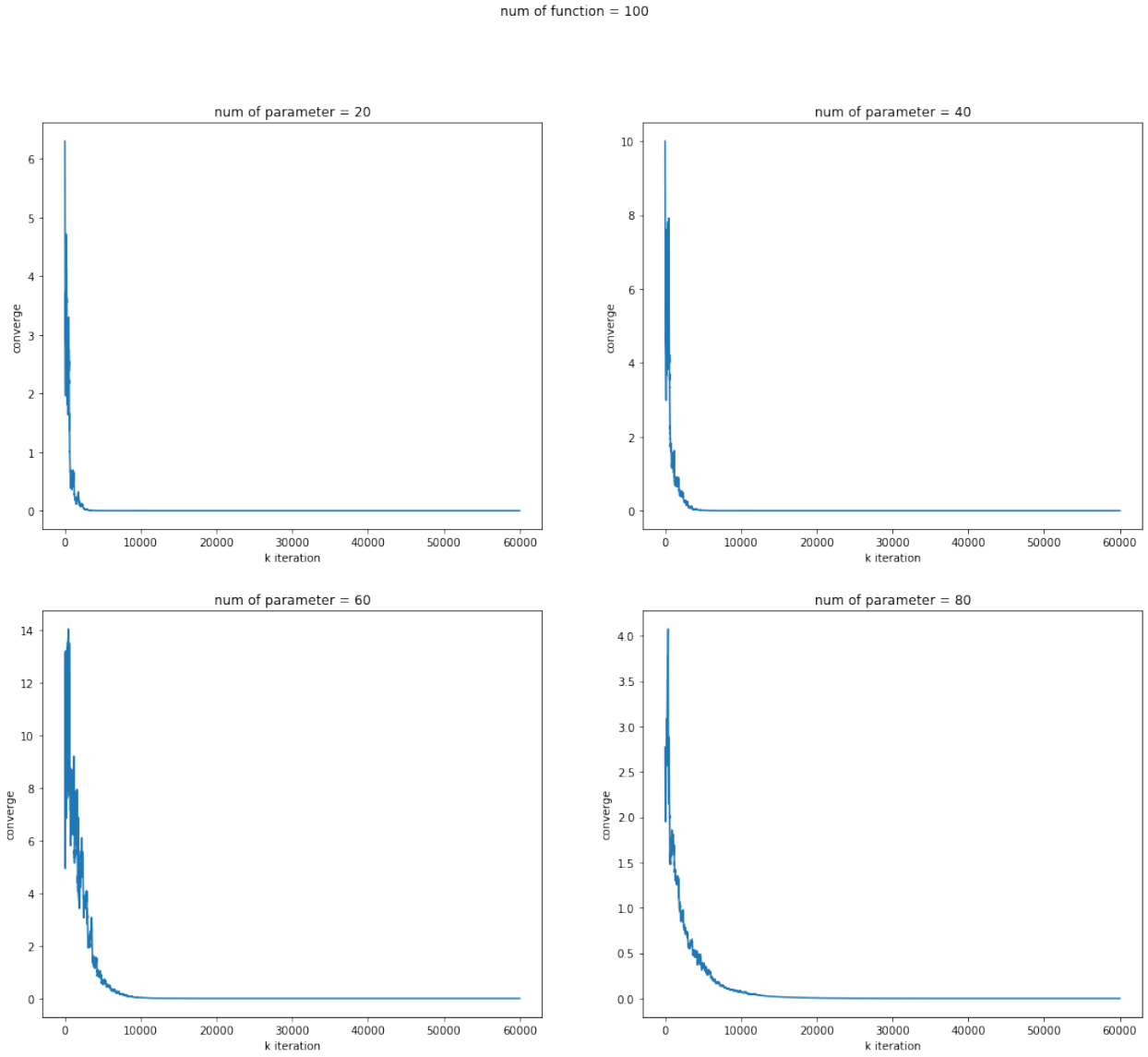

Figure 6: $n = 100$, $p = 20, 40, 60, 80$

As shown in Figure 6, when $n = 100$ and $p = 20, 40, 60, 80$, we observe that all four graphs converge, but the first graph (when number of parameters $p = 20$, which is the smallest in this group) converges at the fastest speed. This corresponds to our speculation drawn from the first set of graphs.

Number of functions $n = 1000$

Finally, we set the number of functions to $n = 1000$, and plot four groups corresponding to four different numbers of parameters $p = 20, 40, 60, 80$ (see Figure 7).

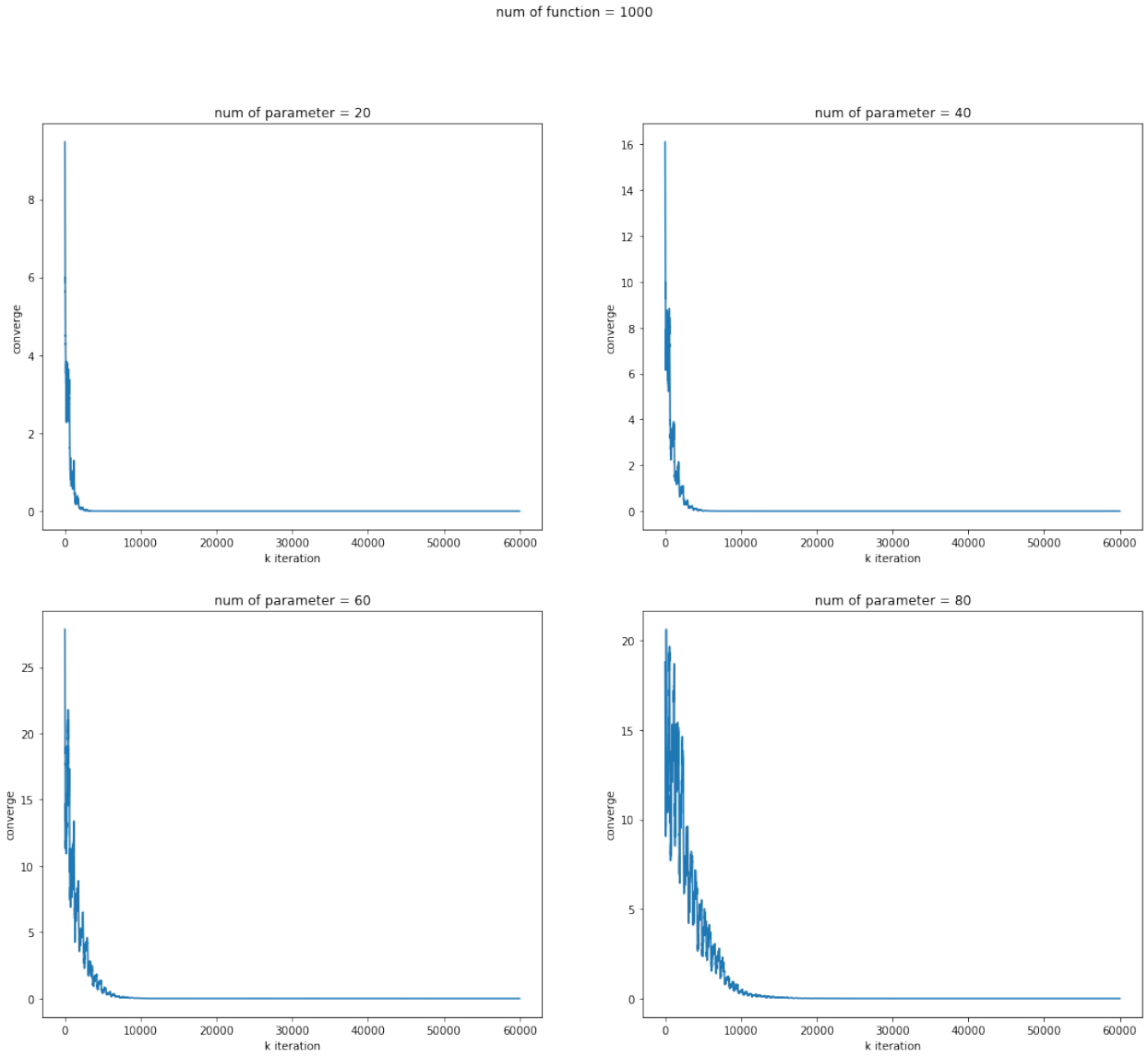

Figure 7: $n = 1000$, $p = 20, 40, 60, 80$

As shown in Figure 7, when $n = 1000$ and $p = 20, 40, 60, 80$, we observe that the four graphs still reveal the same pattern: with the increase of the number of parameters, the graph converges slower (and may even diverge as shown in the first set of graphs).

To conclude, when we fix the number of parameters, the larger the number of functions is, the faster the graph converges.

2.4 Comparison with other methods

We compared SVRG to GD and SGD in two ways:

1. Compare log error versus iterations.

2. Compare log error versus running time.

We chose the following parameters.

1. $\eta^{(k)} = 0.01$ for GD and SVRG, $\eta^{(k)} = \frac{1}{k}$ for SGD to make sure the convergence.

2. total update for GD is $10,000$ because it can give good enough convergence, and total update for SGD and SVRG is $100,000$.

After running the code, we got following results.

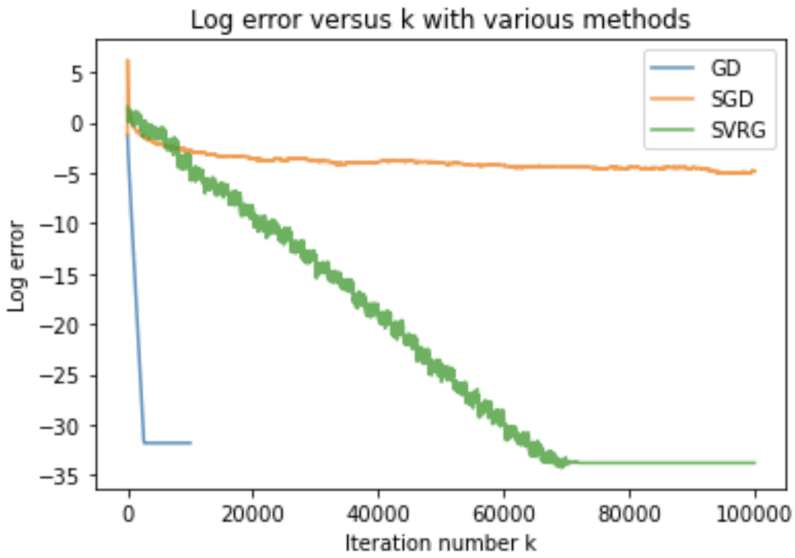

Figure 8: Log-error versus iteration number for SVRG, SGD, and GD

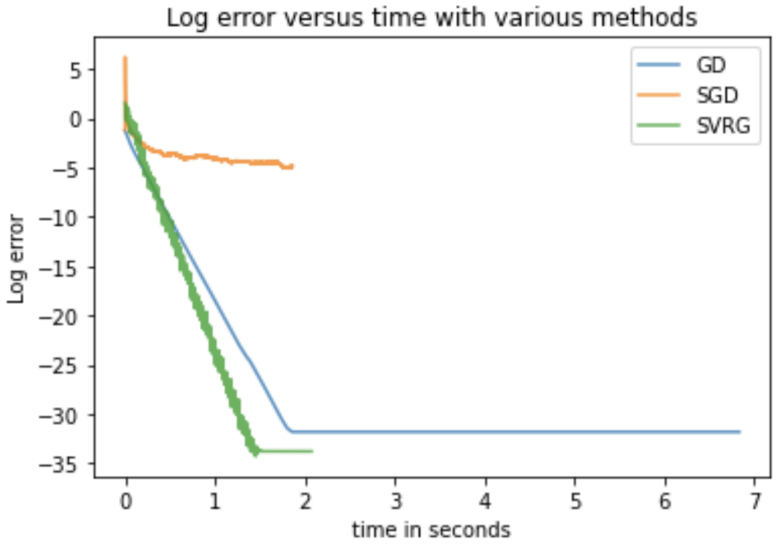

Figure 9: Log-error versus times for SVRG, SGD, and GD

We can see that the convergence speed of SVRG is in between GD, and SGD in terms of number of iterations. However, from the graph of convergence versus running time, we can see that SVRG did a vert good job by beating both SGD and GD in long run although GD and SGD are faster initially. This motivates us to find the best strategy in dealing with gradient descent given what we have found: We use SGD or GD at the beginning of the descent and then turn to use SVRG to find an more precision solution.

3 Appendix: Group Work Distribution

Experiment plan write-up. Keer Xu.

Main code writing. Billy Yu, Keer Xu, Pengqiu Li.

Introduction write-up. Billy Yu, Pengqiu Li.

Experiment 1: Conditioning. Pengqiu Li.

Experiment 2: Algorithm parameter. Billy Yu.

Experiment 3: Data parameter. Keer Xu

Experiment 4: Comparison with other methods Billy Yu, Pengqiu Li.

The group work was generally a pleasant experience for all three group members. We have worked together and helped one another throughout the project.