Stochastic Algorithms for Convex Feasibility and Optimization with Many Constraints

Slavomír Hanzely

Supervisor: Peter Richtárik

May 22, 2018

Abstract

This project is about constraint optimization. It is focused on sketching algorithm finding point close to closed convex set by projecting on supersets and also on SAGA algorithm. Both algorithm have been described with motivation behind them. Their optimal parameters have been tested and compared with experimentally best parameters. It was shown, that theoretical parameters are not best in practice, however they are not much worse than experimentally best ones. They might be used in practice, because it is hard to compute experimentally better ones.

Contents

Chapter 1

Introduction

In this project, we are working with complicated convex set X defined as intersection of large number of simple convex sets. Starting with convex feasibility problem, our goal is to quickly find point of the convex set X. Due to the fact that X is complicated, the direct computation to the point X might be infeasible, or at least extremely costly. However, one might take advantage of the fact, that X is defined as the intersection of large number of simple sets.

The type of indirect algorithms, that could be used for this kind of problem, are iterative ones. Starting with initial guess, iterative algorithms are iteratively improving their guess, until they are sufficiently close. Often, they use to find point near optimum much faster than exact computation.

In our case, we are having a large number of simple constraints. Working with all of them in each step is slow. In particular we focus only on a subset of the constraints each iteration. In this work, convex feasibility problem is solved by sketching algorithm, which in each step project current point on randomly chosen individual constraint[1, 2].

Convex feasibility problem could be equivalently reformulated in many ways, so that the set of solutions remains the same, which brings different insights to the problem. One of the options is to reformulate it as stochastic optimization problem. Natural choice of algorithm for this newly constructed problem is well-known gradient descent. It turns out that stochastic gradient descent on the reformulated problem is equivalent to sketching algorithm on original problem[1, 2].

In both the original and reformulated problem, we were looking for arbitrary point sufficiently close to set X. However, if points in X are not equally good for us, we need to do something different. We may have access to the "quality" of each point in X, and wish to choose the best point in X accordingly. This is however equivalent to minimizing some function over set X. Let's suppose, that we want to find point in X, that also minimizes function, defined as sum of simple convex functions.

Our new problem turned out to be stochastic minimization problem over set X. We could once more reformulate as a stochastic optimization problem without constraints (its goal is to minimize sum of large number of functions) and try to use different algorithms on it[3]. Deterministic gradient descent is infeasible due to large number of functions.

One of the possible approaches is to use classical stochastic gradient descent. It is necessary

to remind, that stochastic gradient descent is moving according to subgradients, which are nonzero even in optimum. Thus stochastic gradient descent does not stop at optimum unless the size of steps is being reduced to zero. However, reducing size of steps leads to slower convergence rate[4].

Second option, how to make stochastic modification of gradient descent that will stop at optimum (without reducing size of the steps), is to move according to estimate of gradient of average of all functions and make this gradient to go to 0. If algorithm find point, that the function has average gradient 0 in it, than that point is minimum.

For this kind of approach, it is possible to use SAGA algorithm[5]. It has saved all (probably outdated) gradients in Jacobian and their average. In each step, it updates one gradient, their average and then does gradient descent step (using average of all gradients).

SAGA algorithm guarantees the convergence to the global optimum even without reducing stepsize, and enjoys fast, linear convergence rate. It has low iteration cost, because it computes gradient of only one function in each step. In terms of number of iteration until convergence, SAGA converges almost as fast as gradient descent (faster than stochastic gradient descent)[5].

The update of gradient estimator in SAGA can be viewed as Jacobian sketching. In particular, in each iteration of SAGA, one column of Jacobian is updated and rest of Jacobian stays the same. We could take updating of Jacobian as using sketching algorithm again and projecting old Jacobian to superset of new Jacobian (changing only one column). However, this can be seen as a special instance of the algorithm to find a point in set X defined as an intersection of affine subspaces. Thus we got by circle to algorithm, which we had been beginning with.

The goal of this work is to intuitively present motivation behind both sketching and SAGA algorithms, test them in practice and find out, whether there is possibility to improve by choosing different parameters than ones suggested by the theory. For both sketching and SAGA algorithms, there are also included experimental testing of their performance and also comparison, how fast does they converge with different parameters. Project was implemented in the language Julia, which is very fast and simple to use, especially for matrix operations.

Chapter 2

Stochastic Algorithms for Convex Feasibility

2.1 Convex feasibility

Definition 1. The *convex feasibility* problem is the problem

$$\text{Find} \quad x \in X, \tag{2.1}$$

where X is a closed convex subset of $\mathbb{R}^d$.

It is a basic result of convex geometry, that under the closeness and convexity assumptions, the Euclidean *projection operator* is well defined:

$$\Pi_X(x) = \arg\min_{y \in X} \|y - x\|. \tag{2.2}$$

It is well known that the projection operator enjoys the following inequality:

$$\langle x - \Pi_X(x), y - \Pi_X(x) \rangle \leq 0, \qquad x \in \mathbb{R}^d,\ y \in X. \tag{2.3}$$

Note that the above inequality implies

$$\langle \Pi_X(x) - x, y - x \rangle \geq 0, \qquad x \in \mathbb{R}^d,\ y \in X. \tag{2.4}$$

If the projection operator Π_X is available, it is possible to solve (2.1) by simply applying the projection operator to an arbitrary point. However, in situations when X is "difficult", this may not be possible. This is often the case when either d is large, or when X has a complicated structure. In many practical applications, it is sufficient to instead find a point x "close" to X:

$$\text{Find} \quad x \in \mathbb{R}^d \quad \text{such that} \quad \|x - \Pi_X(x)\| \leq \epsilon.$$

In many applications it is often the case that X is represented as the intersection of many "simple" constraints. In particular, let $X_1, X_2, \ldots, X_n$ be closed convex sets, and assume that

$$X = X_1 \cap X_2 \cap \cdots \cap X_n. \tag{2.5}$$

We assume that the sets X_i are simple in the sense that it is easy/cheap to project onto them. That is, it is easy to compute $\Pi_{X_i}(x)$ for any x.

2.2 Algorithm

The above assumptions and the structure of X as the intersection of many simple sets X_i point to the following simple iterative algorithm: given x^k, choose $i \in \{1, 2, \ldots, n\}$ and set

$$x^{k+1} = \Pi_{X_i}(x^k).$$

Note that there is no reason for x^{k+1} to belong to X. However, note also that by performing this step, we have moved closer to X. Indeed, using the definition of projection, we get

$$\|x^{k+1} - \Pi_X(x^{k+1})\| \leq \|x^{k+1} - \Pi_X(x^k)\| \leq \|x^k - \Pi_X(x^k)\|,$$

where the last inequality follows from (2.4). One may thus hope that by repeating the iterative process, the distance

$$\|x^k - \Pi_X(x^k)\|$$

keeps decreasing to zero.

Indeed, it was proved in [2] (see also [6] for the case when the sets X_i are hyperplanes) that under some technical assumptions on the sets $X_1, \ldots, X_n$, the above strategy works if we in each iteration choose i according to some fixed probability law. What we just described arises as a special case (for $\omega = 1$ and $\tau = 1$) of Algorithm 1 descried below.

Algorithm 1 Stochastic Projection Method [2]

1: Choose parameters ω, τ and X_i
2: Choose initial point x^0
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: Choose $S_k \subset \{1, 2, \ldots, n\}$ randomly, satisfying $|S_k| = \tau$
5: $x^{k+1} \leftarrow (1 - \omega)x^k + \frac{\omega}{\tau} \sum_{i \in S_k} \Pi_{X_i}(x^k)$

Parameter ω specifies the size of the step, parameter τ determines a number of the sets to project in every iteration.

As mentioned above, it can be shown that the algorithm converges relatively quickly even though X is a complicated set often given by a very large amount of data. To get to distance ϵ from optimum in expectation, $\mathcal{O}(\log(\frac{1}{\epsilon}))$ iterations suffice [1, 6]. However, choosing the parameters of the algorithm can be challenging.

2.3 Implementation

In simulation, convex set is represented by set of x such as $Ax = b$ (for randomly generated matrix A and vector b) intersected with unit ball.

Variant with only linear system is easy, and already known as random Kaczmarz method [6, 7]. We are interested in case, where there are also other convex sets. As convex sets to be projected on, we have chosen the affine subspaces (rows of matrix A) and the ball with center at origin.

The simulation took place in $\mathbb{R}^{10}$.

During the simulation, program computed (for each processed x^k) distance from the convergence spot (later I'll call it just distances). Also for processed x^k it computes, for which

ω would be the new point closest to the real projection spot. It is the best ω, that could be chosen for given (concrete) x^k (later, I'll call it just best omegas).

As the output of the simulation, these two informations (distances and best omegas) were plotted. Because the distances of the points x^k from the real projection spot are exponentially decreasing, plotted graph has axis y with logarithmic scale.

The most relevant data to compare speed of convergence is known convergence rate of only linear system (excluding ball). In k-th iteration, its expected distance from optimum is $||x - x^0||(1 - \lambda_{min})^k$ [6], where λ_{min} is smallest nonzero eigenvalue of $A^T A$ (this exponential function I'll call theoretical convergence). So by adding projections on ball it should still be a good approximation.

As baseline, theoretical convergence was plotted (it is also exponential function).

2.4 Simulation setup

There were multiple simulation setups, they differed in following parameters:

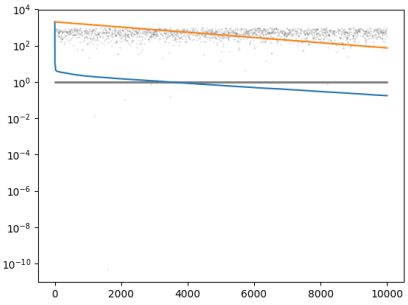
- τ, that is chosen from set $\{1, 3, 8\}$.
- Whether unit ball is between X_i or it is excluded.
- ω, whether it is constant[1] for whole simulation or it chosen each step for optimal value.

For each combination of parameters above, multiple simulation were run (and graphs saved). Below, there are some of the graphs, for each combination of parameters one.

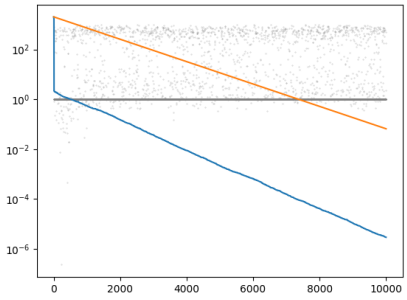
[1] As constant, it was chosen 1

Description of graphs:

- X axis shows iteration
- Y axis shows distance (logarithmic scale)
- Theoretical convergence is orange line.
- Distance from projection is blue line.
- Best omega, that could be chosen (for every concrete x^k) is drawn as grey dot (for x^k).

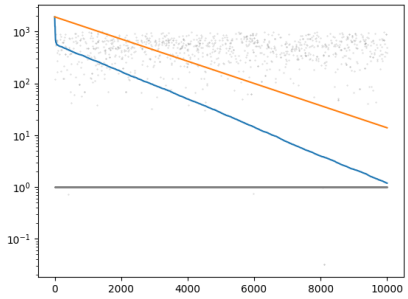


(a) fixed ω

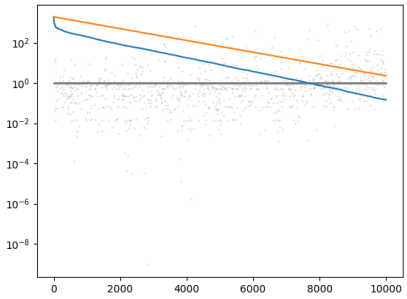


(b) best omega

Figure 2.1: $\tau = 1$, with ball

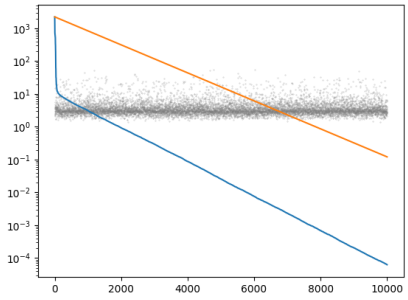


(a) fixed ω

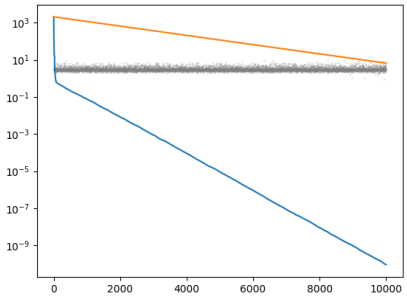


(b) best omega

Figure 2.2: $\tau = 1$, without ball

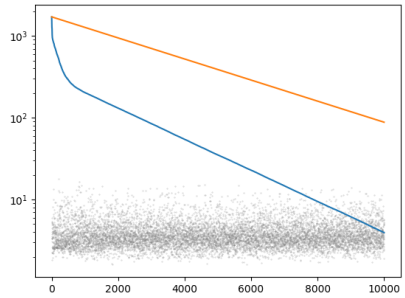


(a) fixed ω

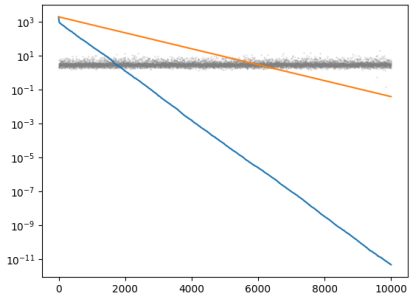


(b) best omega

Figure 2.3: $\tau = 3$, with ball

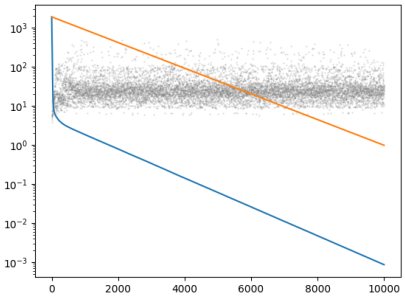


(a) fixed ω

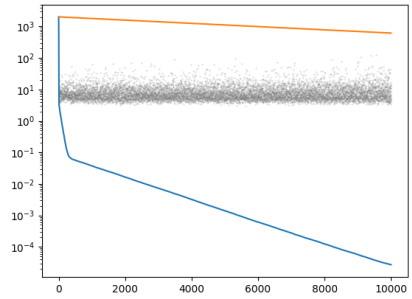


(b) best omega

Figure 2.4: $\tau = 3$, without ball

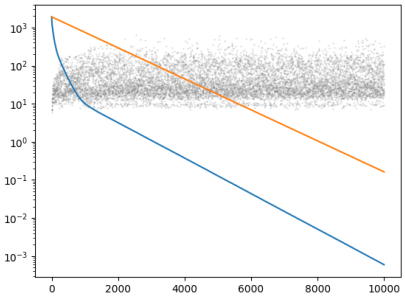


(a) fixed ω

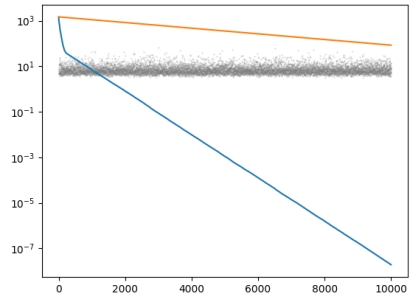


(b) best omega

Figure 2.5: $\tau = 8$, with ball



(a) fixed ω



(b) best omega

Figure 2.6: $\tau = 8$, without ball

2.5 Simulation result

Algorithm was successfully implemented and simulated, plenty of graphs were drawn here.

In every setup, distances of points x^k in graph made line (from some point), thus distances were always decreasing exponentially.

For $\tau = 1$, we were projecting only on 1 line (or ball) each iteration. The iteration cost in this setup was lowest, however the convergence rate per iteration was slowest. There was no visible difference between setting ω as constant 1 or choosing best omega each in every iteration. It was caused by fact, that for $\tau = 1$ were values of best omegas around 1 (that was also fixed value of ω).

With increasing τ (for fixed ω), simulation was converging a bit faster each iteration (lines of graphs were steeper), but computation of one iteration was slower (because it is necessary to compute τ projections each iteration).

Changing fixed ω to the best omega lead to huge convergence speedup. The convergence rate observed in practice was much faster then theoretical bound. However, in the simulation, best omegas were computed from the real projection point, thus it is not possible to use them for convergence speedup in practice.

Setups with projecting on a ball in beginning of the simulation jumped into smaller distance, but after that, the converge rate was same (even a bit slower) as setup without projection on ball (with same τ and ω). It is caused by the fact, that projecting on ball from outside of the ball lead to big distance reduction. Once point is inside in the ball, projecting on the ball makes no difference (projection of point is the same as projected point - in this case each ω is best omega).

Simulation showed, that current estimation of ω is not the best estimation in practice. Thus it might be possible to prove better estimation.

2.6 Different application of convex feasibility problem

In our setup, we were interested in X to be set of solutions of linear system $Ax = b$ intersected with unit ball.

We could also set X to be set of solutions x such as $Ax = I$, hence by solving convex feasibility problem we will find inversion of matrix A.

We may also set X to be set of solutions x such as $x = B$, for fixed matrix B. This problem is trivial, but it can also be solved using sketching, which can be viewed as a step of the Jacobian sketching for SAGA as mentioned in the introduction. It converges to distance d in $\log(\frac{n}{d(n-1)})$ iterations, where n is dimension of x.

2.7 Alternative angles of views on convex feasibility problem

As mentioned in introduction, convex feasibility problem could be equivalently reformulated in many ways, so that the set of solutions reminds the same. Here are few different

angles of view.

Convex feasibility problem can be viewed as minimization problem over set X, find

$$\arg\min_{x \in X} 0.$$

It may also be reformulated as stochastic optimization problem [1]:

$$\text{minimize } f(x) = \mathbb{E}_{S \sim D}[f_S(x)],$$

where

$$f_S(x) = \frac{1}{2}||x - \Pi_{x^S}(x)||^2.$$

Stochastic gradient descent on latter problem does the same as sketching algorithm on original problem.

We have got algorithm, that can find some point in convex set. However, it can find arbitrary point near to set X. If points in X are not equally good for us, we need to do something different. We may assign values to points (using some function) and then minimize that function. Let's suppose, that we want to find point in X, that also minimizes sum of large number of simple functions over X, we would like to find

$$\arg\min_{x \in X} F(x),$$

where $F(x) = \sum_{i=1}^{l} f_i(x)$.

Chapter 3

Optimization with Many Constraints

3.1 Problem definition

Problem is defined as following: Given a large set of functions $f_1, f_2, \cdots, f_l$ (for $f_i : \mathbb{R}^n \to \mathbb{R}$) and complicated set of constrains $X \subset \mathbb{R}^n, X = \cap_{j=1}^m X_j$ (for simple convex constraints X_j), find a point $x \in \mathbb{R}^n$ that satisfies X and minimize average of functions at x [3]:

$$\min_{x \in \mathbb{R}^n \cap X} \frac{1}{l} \sum_{i=1}^{l} f_i(x).$$

Lets denote $F(x) = \frac{1}{l} \sum_{i=1}^{l} f_i(x)$, the problem is to minimize $F(x)$ also satisfying given constraints.

Instead solving that problem, we will solve following problem[3]:

$$\min_{x \in \mathbb{R}^n} \frac{1}{l} \sum_{i=1}^{l} f_i(x) + \lambda h(x),$$

where $h(x) = \frac{1}{2m} \sum_{j=1}^{m} ||x - \Pi_{X_j}(x)||^2$ and $\Pi_Y(x) = \min_{y \in Y} ||x - y||$. Lets denote $h_i(x) = ||x - \Pi_{X_i}(x)||^2$.

Our new problem is optimization problem without constraints. It could be shown, that solution of new problem is approximate solution of previous problem, accuracy of approximation depends on parameter λ (the bigger λ, more accurate solution). Thus minimization problem with constraints is reduced to minimization problem without constraints, if we assume convexity of functions f_i and h_i, we could find solution using gradient descent:

Algorithm 2 Gradient descent [8]

1: Choose parameter α
2: Choose initial point x^0
3: **for** $k = 0, 1, 2, \cdots$ **do**
4: $\quad x^{k+1} \leftarrow x^k - \alpha \nabla F(x^k)$

Gradient descent is much faster than exact computation, on strongly convex functions it converges to distance d with $\mathcal{O}(\log(\frac{1}{d}))$ iterations [8] . This convergence rate is referred as linear. However, it is not appropriate for this problem. In our case, both the number of functions f_i and number of constraints X_i are large, so it may not be able to process all data at once.

Hence it is necessary to use stochastic version. Let's denote τ minibatch size (number of functions processed in one step), let's define $\nabla F_k(x^k)$ as following:

$$\nabla F_k(x^k) = \frac{1}{\tau}\left(\sum_{i=1}^{\tau_1} \nabla f_{s_i}(x^k) + \lambda \sum_{i=1}^{\tau_2} \nabla h_i(x^k)\right)$$

where $\{r_1, r_2, \cdots, r_{\tau_1}, s_1, s_2, \cdots, s_{\tau_2}\}$ are uniformly randomly picked indexes of functions f_i and h_i (r_i are indexes of functions f_i, s_i are indexes of functions h_i), $\tau_1 + \tau_2 = \tau$, $\{r_1, r_2, \cdots, r_{\tau_1}\} \subseteq \{1, 2, \cdots, l\}$, $\{s_1, s_2, \cdots, s_{\tau_2}\} \subseteq \{1, 2, \cdots, m\}$. Therefore $\nabla F_k(x^k)$ is average gradient of randomly chosen τ functions at point x^k. Stochastic gradient descent is following:

Algorithm 3 Stochastic gradient descent [4]

1: Choose parameters α_0, α_1, ...
2: Choose initial point x^0
3: **for** $k = 0, 1, 2, \cdots$ **do**
4: $\quad x^{k+1} \leftarrow x^k - \alpha_k \nabla F_k(x^k)$

For stochastic gradient descent to stop at optimum, it is necessary for parameters α_k to satisfy: $\lim_{k\to\infty} \alpha_k = 0$. Convergence rate of stochastic gradient descent is slower than gradient descents rate, it converges to distance d with $\mathcal{O}(\frac{1}{d})$ iterations [4]. This convergence rate is referred as sub-linear.

We would like to use algorithm that does not compute all gradients at each step (like stochastic gradient descent), but stops at optimum without decreasing stepsize parameter α (like gradient descent).

Our algorithm is called SAGA, it will move according to average gradient of functions, but it will not compute average gradient exactly. It will store list of (probably outdated) gradients of functions and their average. In each step, it updates τ gradients (randomly chosen), it updates average of all gradients and does step of gradient descent[5].

Algorithm is in each step updating current point x^k, current estimate of gradients of all functions f_i and all functions h_i (in Jacobian J) and average of gradients

$$\nabla F_{avr} = \frac{1}{l+m}\left(\sum_{i=1}^{l} \nabla f_i + \lambda \sum_{i=1}^{m} \nabla h_i\right).$$

In each step current point x is used for better estimation of Jacobian and then current estimation of jacobian is used for step in gradient descent.

This algorithm requires all functions f_i and h_i to be μ-strongly convex and Lipschitz smooth with constant L [5]. μ-strong convexity of function g means, that for $\mu > 0$ for all x, y holds

$$\langle \nabla g(x) - \nabla g(y), (x - y) \rangle \geq \mu ||x - y||_2^2.$$

Lipschitz smoothness with constant L of function g means, that for all x, y holds

$$||\nabla g(x) - \nabla g(y)||_2 \leq L||x - y||_2.$$

Both constants μ and L bound, how much function values $\nabla f(x)$ could change by changing its argument.

Algorithm 4 SAGA [5]

Choose parameters α, τ, r, s.
Choose initial point x^0.
Choose initial Jacobian J_f of functions f_1, f_2, $\cdots$, f_l.
Choose initial Jacobian J_h of functions h_1, h_2, $\cdots$, h_m.
Compute initial average of gradients $\nabla_{avr} = \frac{1}{l+m}\left(\sum_{i=1}^{l} \nabla f_i + \lambda \sum_{i=1}^{m} \nabla h_i\right)$.
for $k = 0, 1, 2, \cdots$ **do**
    Choose together τ functions uniformly randomly from all functions f_i and h_i.
    (choose indexes of functions f_i: $\{r_1, r_2, \cdots, r_{\tau_1}\} \leftarrow$ random subset of $\{1, 2, \cdots, l\}$,
    choose indexes of functions h_i: $\{s_1, s_2, \cdots, s_{\tau_2}\} \leftarrow$ random subset of $\{1, 2, \cdots, m\}$,
    so that $\tau_1 + \tau_2 = \tau$.)
    for $i = 1, \cdots, \tau_1$ **do** update gradient of function f_{r_i} in $J_{f_{r_i}}$
        compute new gradient $J_{new} \leftarrow \nabla f_{r_i}$
        update average gradient of all functions $\nabla_{avr} = \nabla_{avr} + \frac{1}{l+m}(J_{new} - J_{f_{r_i}})$
        update gradient in Jacobian $J_{f_{r_i}} \leftarrow J_{new}$
    for $i = 1, \cdots, \tau_2$ **do** update gradient of function h_{s_i} in $J_{h_{s_i}}$
        compute new gradient $J_{new} \leftarrow \nabla f_{s_i}$
        update average gradient of all functions $\nabla_{avr} = \nabla_{avr} + \frac{1}{l+m}(J_{new} - J_{h_{s_i}})$
        update gradient in Jacobian $J_{h_{s_i}} \leftarrow J_{new}$
    do gradient descent step $x^{k+1} \leftarrow x^k - \alpha \nabla F_{avr}(x^k)$

Remark: SAGA algorithm with $\tau = l + m$ is gradient descent algorithm.

Convergence rate of SAGA is referred as linear, it converges to distance d with $\mathcal{O}(\log(\frac{1}{d}))$ iterations [5], so it should converge as fast as gradient descent.

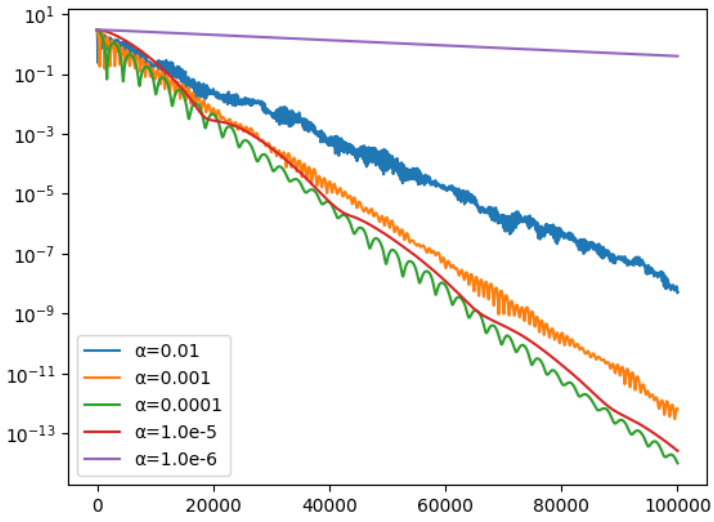


Figure 3.1: convergence of SAGA algorithm for different stepsize α

3.2 Parameters in theory

First of all, let's take a look on parameter λ in $F(x) + \lambda h(x)$ (function, we want to minimize). As mentioned above, bigger λ put more emphasis on constraints. However, bigger λ increase the conditional number of that function, so overall convergence speed is slower.

The most import parameter in our simulation is minibatch parameter τ. The bigger minibatch parameter τ, the more functions are updated each step. Consequently, the average gradient of functions is more precise, so x^i will go in less iterations to optimum. Also for bigger τ the optimal stepsize parameter α may be bigger. Hence the algorithm should converge in less iterations. But the bigger minibatch parameter τ leads to more gradients computation in each step, so the iterations are slower to compute. It is not clear, what combination of parameters τ and α leads to fastest convergence. As measure of speed of convergence, number of gradients computed before convergence is used.

In theory, algorithm should be fastest when $\tau = 1$, and with increasing τ, it should get monotonically worse, the worst possible result is for $\tau = k + m$ (gradient descent case). Let's denote L Lipschitz smooth constant for every f_i and μ strongly convex constant for every f_i. Fastest convergence for strongly convex functions is achieved for $\alpha = \frac{1}{2(\mu n + L)}$. For non-strongly convex functions, the best convergence rate is achieved for $\alpha = \frac{1}{3L}$. These values are best for general functions, so it may be possible that they are not optimal in practice. In simulation we would like to find out, how good is best stepsize parameter in practice and how does it change with increase of minibatch size.

3.3 Implementation

The simulation took place in $\mathbb{R}^{10}$, number of functions was 1500, number of constraints was 500.

Functions f_i were chosen as quadratic functions, $f_i(x) = (A_i x - b_i)^T (A_i x - b_i)$, for random matrix A_i (of size 10×10) and random vector b_i (of size 10). Constraints were chosen to be linear, they were given by rows of matrix $A_c x = b_c$, for random matrix A_c (of size 500×10), and random vector b_c (of size 10).

In our setup, L is the biggest eigenvalue of functions f_i and $\mu = 0$.

3.4 Simulation setup

If stepsize parameter was fixed for different minibatch sizes, the plots looked like following:

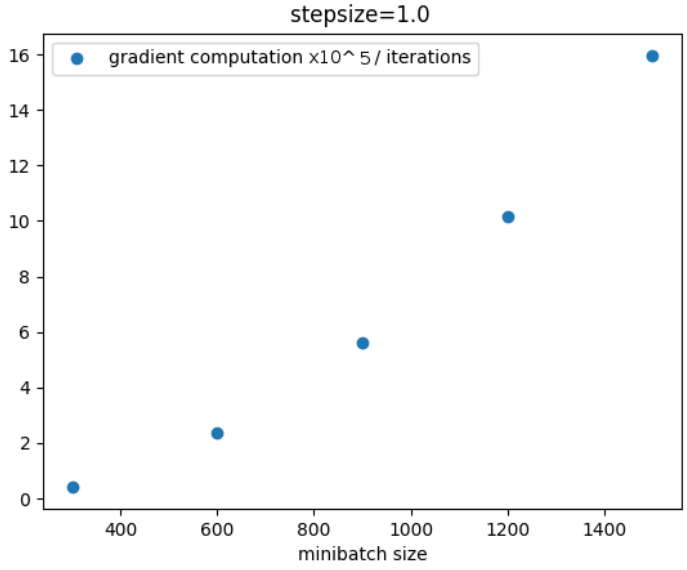


Figure 3.2: fixed stepsize for all minibatch sizes

It turned out, that optimal stepsize parameters for different minibatch sizes were completely different. Increase of gradient computation in Figure 3.2 was caused by wrong stepsize parameter choice. Hence it is necessary to compute experimentally optimal stepsize parameter for each minibatch size.

In each simulation, functions, initial point x^0 and parameters τ, λ were fixed. Then different minibatch sizes τ were chosen. For each of them experimentally best parameter α_{τ_i} was found and number of gradient computed until convergence (for α_{τ_i}) was plotted.

To find α_{τ_i}, many different values were tried, the best of them was picked. Initial guesses had form 10^t (to capture α with low relative error to optimal value), after observation, more

values were tried close to the best guess so far.

3.5 Simulation result

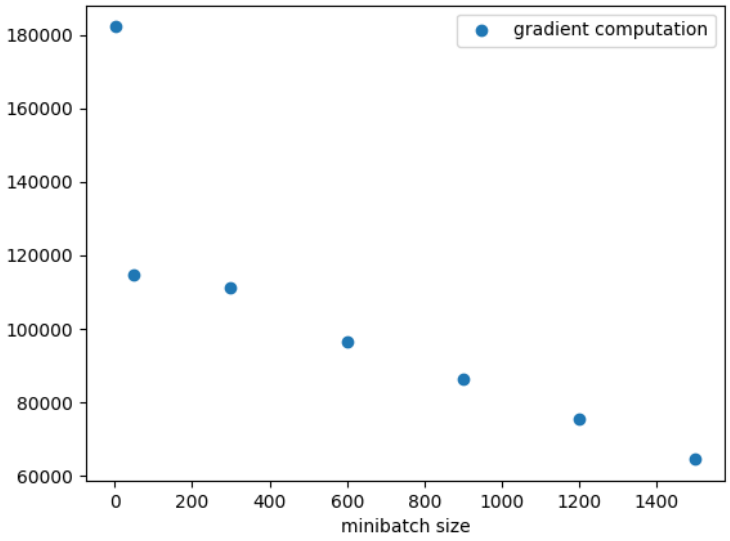


Figure 3.3: experimentally optimal stepsize

For minibatch size $\tau = 1$ was experimentally optimal stepsize parameter α much smaller than theoretical optimal parameter. Choosing α to be theoretical best one lead to much slower convergence speed.

With increasing τ number of gradient computation necessary to converge decreased, difference was $10\% - 66\%$ decrease of gradient computation.

Ratio between gradient computation with theoretical and experimental stepsizes increased with increase in λ. For $\lambda < 1$, ratio was less than 20%, for $\lambda = 100$, it was $\sim 60\%$, for $\lambda = 1000$, it was even more than 95%. It may be caused by simulation setup and uniform function sampling.

In this setup, theoretical value of α for τ lead to slower convergence rate. However, computing experimentally better parameters required much more computation than converging with theoretical parameters itself.

Chapter 4

Conclusion

4.1 Jacobian sketching

Let's look once again on sketching algorithm. We had convex set X as set of solutions x in $Ax = b$. In each step, we were projecting point x^k on superset of X, that had form of rows of matrix A.

Later, in SAGA algorithm, we have been updating columns of Jacobian in each iteration. This update could be viewed as projecting old Jacobian to column of new Jacobian. It change only one column at time. This is exactly convex feasibility problem, that what we were doing before.

We could take updating of Jacobian as convex feasibility sketching achieved by projecting old Jacobian to supersets (columns) of new Jacobian, we would like to find $x \in X$, where $X = Jacobian_{real}$ [9]. It is similar to sketching problem $x = B$ 2.6. However, in SAGA case, the real Jacobian is changing in each iteration, thus we were doing sketching on changing set in each iteration. The closer we are to optimum, the less real Jacobian is changing. With sufficiently small stepsize parameter, average of Jacobian converges to 0, hence SAGA algorithm converges to local minimum.

4.2 Summary

This work was about sketching algorithm and SAGA algorithm, they were motivated from constraint optimization perspective. However, they could be viewed from other points of view (e.g. stochastic optimization problem), different angle may help us better comprehend those algorithm.

In simulations of both algorithms, it was shown, that theoretical best parameters are not best in practice. There is clear gap between theory and practice. But parameter tuning is expensive, so for practical purposes in unknown setup would it be better to use theoretical parameters.

4.3 Limitation

When optimization problem has big condition number (problem is ill-conditioned), than all gradient-based iterative methods converge slowly.

Condition number of $\min_{x\in\mathbb{R}^n} \frac{1}{l}\sum_{i=1}^{l} f_i(x)+\lambda h(x)$ depends on λ, it increases linearly with increase in λ. When λ is large, condition number of the problem is large, hence computation became lengthy and not usable.

4.4 Extensions

This work was meant to be for school course, due to the limited amount of time, there are multiple possibilities, how could it be extended:

- In convex feasibility setup, by analysis of how fast simulation converges on best omegas setups, relation between theoretical convergence, parameters and best omegas may be found. To find out, whether there is correlation, I suggest using machine learning algorithms. Its goal would be to predict steepness of line of distances, as parameters it would get theoretical convergence curve, τ, best omegas. If algorithm will predict line accurately, best omegas estimation could be retrieved from it.

- SAGA algorithm could be compared to Random Kaczmarz method on linear systems. It would be special case of setup in this work, without functions.

- Gradient-based iterative optimization methods could be slightly adjusted to converge to distance d in rate $\mathcal{O}(\frac{1}{\sqrt{d}})$ iterations. It is called Nesterov's accelerated gradient method [10], its idea may be applied to SAGA algorithm, leading to faster theoretical convergence rate. Experimental testing might be done to test its performance.

- In article [3], they considered optimization problem

$$\min_{x\in\mathbb{R}^n} \frac{1}{l}\sum_{i=1}^{l} f_i(x) + \lambda h(x),$$

  where $h(x) = \frac{1}{2m}\sum_{j=1}^{m} ||x-\Pi_{X_j}(x)||^2$ (as we do in 3.1). They suggested to dynamically change parameter λ in the simulation, starting with smaller value, increasing it over time. Experimentally they found out, that it should lead to faster convergence. It may be tested, how should the parameter λ be adjusted for optimal converge.

Bibliography

[1] Robert Mansel Gower. Sketch and project: Randomized iterative methods for linear systems and inverting matrices, 2016. https://epubs.siam.org/doi/abs/10.1137/15M1025487.

[2] Andrei Patrascu Ion Necoara, Peter Richtárik. Randomized projection methods for convex feasibility. 2017. http://141.85.225.150/papers/feasibility.pdf.

[3] Peter Richtárik Konstantin Mishchenko. A stochastic penalty model for convex and non-convex optimization with big constraints. 2018. http://epostersonline.com/obd2018/node/65.

[4] Mark Schmidt Simon Lacoste-Julien and Francis Bach. A simpler approach to obtaining an $\mathcal{O}(1/t)$ convergence rate for the projected stochastic subgradient method. 2012. www.arxiv.org/pdf/1212.2002v2.pdf.

[5] Francis Bach Aaron Defazio. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. 2014. www.di.ens.fr/~fbach/Defazio_NIPS2014.pdf.

[6] Roman Vershynin Thomas Strohmer. randomized kaczmarz algorithm with exponential convergence. 2007. www.arxiv.org/pdf/math/0702226.pdf.

[7] Peter Richtárik Robert Mansel Gower. Randomized iterative methods for linear systems. 2015. http://epubs.siam.org/doi/abs/10.1137/15M1025487.

[8] RadhaKrishna Ganti. Convergence rate of gradient descent algorithm, 2015. www.rkganti.wordpress.com/2015/08/21/convergence-rate-of-gradient-descent-algorithm/.

[9] Francis Bach Robert M. Gower, Peter Richtárik. Stochastic quasi-gradient methods: Variance reduction via jacobian sketching. 2018. https://arxiv.org/abs/1805.02632.

[10] David Barber Aleksandar Botev, Guy Lever. Nesterov's accelerated gradient and momentum as approximations to regularised update descent. 2016. www.arxiv.org/pdf/1607.01981.pdf.