Convergence rate for regularised linear least squares using distributed forward-backward method

Beilun Zhang, supervised by Matthew Tam

University of Melbourne

Introduction

Consider the problem where we want to find the most sparse vector x that satisfies $Ax = b$ (i.e. minimise $||x||_0$, the number of zero entries in x). This problem can be approximated as an optimisation problem as follow:
Let $\lambda > 0, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, find

$$\arg\min_{x \in \mathbb{R}^n} \lambda \|x\|_1 + \frac{1}{2}\|Ax - b\|_2^2 \tag{1}$$

This problem is called the ℓ_1-regularised linear least squares. There are already well-established algorithms to solve this problem (such as the forward-backward method), which works well in a centeralised setting, however, here we will consider an algorithm for this problem that is optimised for a distributed setting.

Model

In [1], Arago-Artacho, Malitsky, Tam and Torregrosa-Belen proposed an algorithms to solve monotone inclusion problems in a real Hilbert space $\mathscr{H}$ in the form of:

$$\text{find } x \in \mathscr{H} \text{ such that } 0 \in \Big(\sum_{i=1}^{n} \mathrm{A}_i + \sum_{i=1}^{m} \mathrm{B}_i\Big)(x) \tag{2}$$

where $\mathrm{A}_1, ..., \mathrm{A}_n : \mathscr{H} \rightrightarrows \mathscr{H}$ are maximally monotone operator and $\mathrm{B}_1, ..., \mathrm{B}_m : \mathscr{H} \to \mathscr{H}$ are $\frac{1}{L}$-cocoercive. By using zero operators if necessary, the algorithm can always assume that $m = n - 1$. The algorithm can be expressed as fixed point iteration $\mathbf{z}^{k+1} = \mathrm{T}(\mathbf{z}^k)$, $\mathrm{T} : \mathscr{H}^{n-1} \to \mathscr{H}^{n-1}$ is given by

$$\mathrm{T}(\mathbf{z}) := \mathbf{z} + \gamma \begin{pmatrix} x_2 - x_1 \\ x_3 - x_2 \\ \vdots \\ x_n - x_{n-1} \end{pmatrix} \tag{3}$$

where $\mathbf{x} = (x_1, ..., x_n) \in \mathscr{H}^n$ depends on $\mathbf{z} = (z_1, ..., z_{n-1}) \in \mathscr{H}^{n-1}$ and is given by

$$\begin{cases} x_1 = J_{\alpha \mathrm{A}_1}(z_1) \\ x_i = J_{\alpha \mathrm{A}_i}(z_i + x_{i-1} - z_{i-1} - \alpha \mathrm{B}_{i-1}(x_{i-1})) \quad \forall i \in [2, n-1] \\ x_n = J_{\alpha \mathrm{A}_n}(x_1 + x_{n-1} - z_{n-1} - \alpha \mathrm{B}_{n-1}(x_{n-1})) \end{cases}$$

Where J_f is the resolvent for operator f. After finding $\mathbf{z}^* \in \operatorname{Fix} \mathrm{T}$ through fixed point iteration, x as described in (2) is given by $x^* = J_{\alpha \mathrm{A}_1}(z_1^*)$.
In a distributed system with n agents, agent i is responsible for updating x_i and sending x_i to agent $i-1, i+1 \mod n$
We can apply this to our problems of ℓ_1-regularised linear least squares. The equation in (1) is equivalent to

$$\sum_{i=1}^{n} \lambda |x_i| + \sum_{i=1}^{m} \frac{1}{2}|A_i x - b_i|^2 = \sum_{i=1}^{n} g_i(x) + \sum_{i=1}^{m} f_i(x) \tag{4}$$

Since the problem is convex, finding $\arg\min_{x \in \mathbb{R}^n}$ for (4) is the same as finding $x \in \mathbb{R}^n$ such that $0 \in (\sum_{i=1}^{n} \partial g_i + \sum_{i=1}^{m} \nabla f_i)(x)$, where ∂g_i is the subdifferential of g_i, thus we have $\mathrm{A}_i = \partial g_i$ and $\mathrm{B}_i = \nabla f_i$, with

$$J_{\alpha \partial g_i} = \begin{bmatrix} x_1 \\ \vdots \\ S(x_i; \alpha\lambda) \\ \vdots \\ x_n \end{bmatrix} \text{ and } \nabla f_i(x) = A_i^T(A_i x - b_i)$$

Where A_i is the ith row of matrix A and

$$S(y; k) = \begin{cases} y + k & y < -k \\ 0 & -k \le y \le k \\ y - k & y > k \end{cases}$$

Results

We explored 3 methods/conditions that affects the convergence rate

Values for α and γ

There are 3 free parameters in the model: λ (regularization parameter), α (step size) and γ (relaxation parameter). We will focus only on the parameter α and γ, as λ controls how good the approximation is and has little effect on the convergence rate. From [1], the algorithm would converge when $\alpha \in (0, \frac{2}{L})$ and $\gamma \in (0, 1 - \frac{\alpha L}{2})$. Fig. 1 shows the normalized number of iterations the algorithm takes when we set α and γ a certain percentage of their upper bound. The figure shows that the number of normalized iterations is minimum when α is approximately 50% of $\frac{2}{L}$ and γ is close to 100% of $1 - \frac{\alpha L}{2}$.

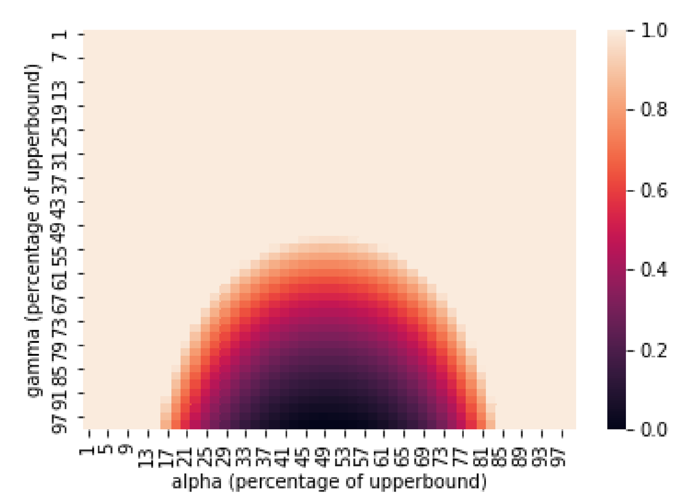

Fig. 1: Axis represent the % of theoretical upper bound α, γ is at

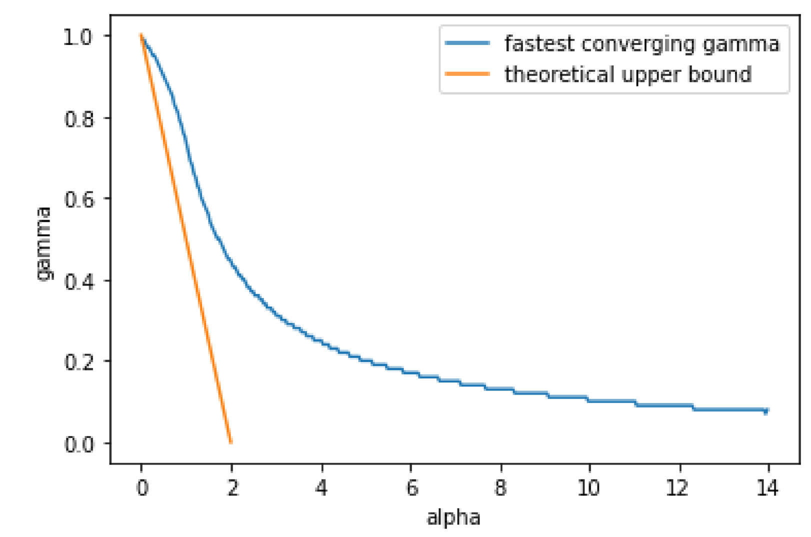

Fig. 2: Experimental upper bound for γ vs theoretical upper bound

However, it turns out that the upper bound for γ and α could be improved. Fig. 2 shows the γ values that gives the lowest number of iterations for each α, depicts that the algorithm still converges when the parameters are not within their theoretical upper bound. If we go beyond the theoretical upper bound, the optimal α value is still somewhere around 50% of the $\frac{2}{L}$ ($L = 1$ in this example), as shown in Fig. 4, with the optimal γ value being around 0.7

Preconditioning

Another way to improve the convergence rate is the scale the Lipschitz constant L so that each ∇f_i is at most L-Lipschitz (i.e. they are not $(L - \epsilon)$-Lipschitz). Note due to Baillon-Haddad theorem, ∇f_i is $\frac{1}{L}$-cocoercive iff it is L-Lipschitz. ∇f_i is L-Lipschitz if:

$$\|\nabla f_i(x) - \nabla f_i(y)\| = \|A_i^T A_i(x - y)\| \le L\|x - y\| \quad \forall x, y \in \mathbb{R}^n \tag{5}$$

As $\|A_i^T A_i(x - y)\| \le \sigma_{\max}(A_i^T A_i)\|x - y\|$, where $\sigma_{\max}(A_i^T A_i)$ is the maximum singular value of $A_i^T A_i$, ∇f_i is thus at most $\sigma_{\max}(A_i^T A_i)$-Lipschitz. We can multiply each row A_i with $\sqrt{\frac{L}{\sigma_{\max}(A_i^T A_i)}}$ to have each ∇f_i at most L-Lipschitz. This is equivalent to multiplying A with a diagonal matrix D, and our new problem has $A_{new} = DA$, and $b_{new} = Db$, note that this won't change the solution of our algorithm.

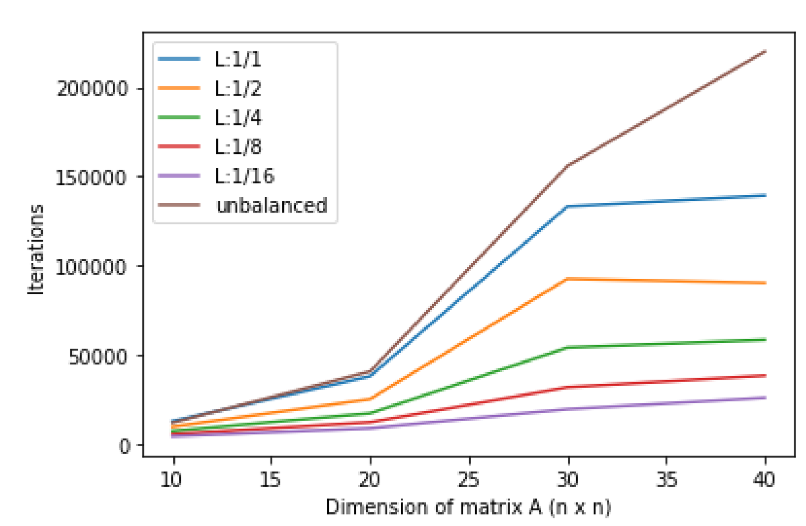

Fig. 3: Convergence rate for preconditioning with different L "unbalanced" is when we have no preconditioning

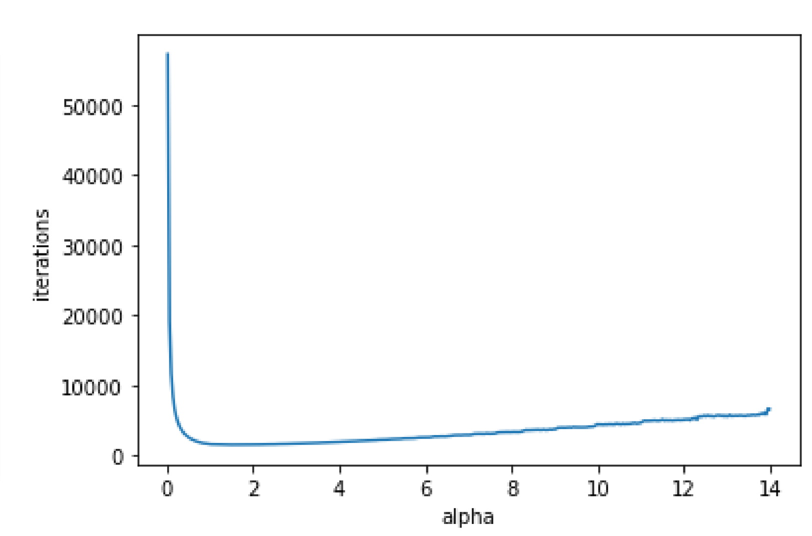

Fig. 4: iterations for the algorithm when the optimal γ value is used for that α

Preconditioning this way drastically improves the convergence rate. In general, the lower L is, the faster the algorithm converges, which is to be expected as the upper bound for α is larger so we can take a larger step for each iteration. However, having a small L comes at a cost - the λ parameter tends to be more sensitive when we have a small L so it would require more work to calibrate it in order for the approximation to be decent.

Different splitting methods

In equation (4), we set $g_i(x) = \lambda|x_i|$, which uses one coordinate of x, and $f_i(x) = \frac{1}{2}|A_i x - b_i|^2$, which uses one row of A. However, we could also split $\lambda\|x\|_1$ and $\frac{1}{2}\|Ax - b\|_2^2$ into $g_i(x), f_i(x)$ differently, so that $g_i(x)$ uses k coordinates from x and $f_i(x)$ uses k rows from A (e.g. when $k = 2$, $g_1(x) = \lambda(|x_1| + |x_2|), f_1(x) = \frac{1}{2}(|A_1 x - b_1|^2 + |A_2 x - b_2|^2)$)

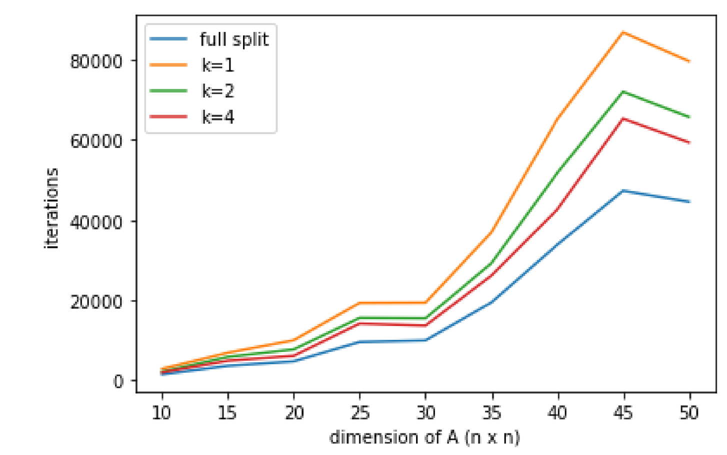

Fig. 5: Convergence rate of different splitting methods

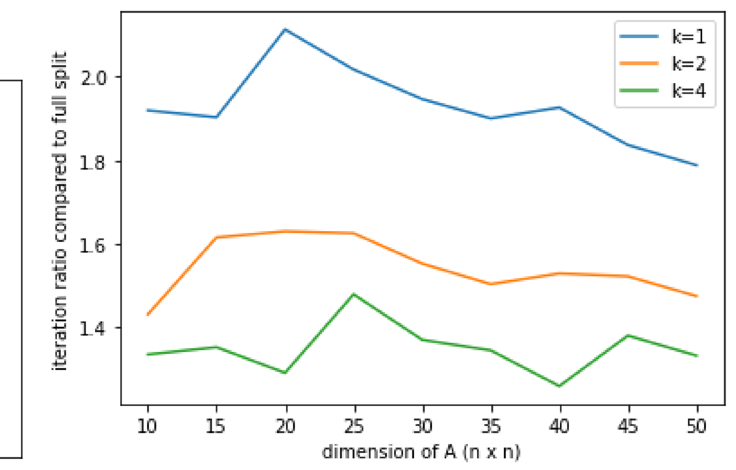

Fig. 6: Ratio of iteration between full split and other split methods

Some notable observations are that:

- When $g_1(x) = \lambda\|x\|_1, g_2(x) = 0, f_1(x) = \frac{1}{2}\|Ax - b\|_2^2$ (which we call the "full split"), the algorithm is the same as the Davis-Yin splitting
- The less number of g_i, f_i we split, the better the convergence rate
- Compared to full split (which can be seen as the a centralized algorithm), splitting with $k = 1$ (which can be seen as distributed algorithm) needs to be approximately 2 times faster to outperform it, which is rather manageable in a distributed setting as each agents do calculations in parallel

Remark

- We make the algorithm terminate when $\|\mathbf{z}^{k+1} - \mathbf{z}^k\| \le 1e-3$ or when $k = 10,000$, whichever comes first
- Trials were done by generating a random sparse x vector with dimension n, and then randomly generates a $n \times n$ matrix A. b is generated using $b = Ax$

Future Work

- Find the formula for the actual upper bound of γ and α
- Implement the distributed version of the algorithm

References

[1] Francisco J. Aragón-Artacho et al. *Distributed forward-backward methods for Ring Networks.* July 2022. URL: https://doi.org/10.48550/arXiv.2112.00274.

[2] Yu. Malitsky. *Projected reflected gradient methods for monotone variational inequalities.* Feb. 2015. URL: https://arxiv.org/abs/1502.04968.

[3] Ernest K. Ryu and Wotao Yin. *Large-scale convex optimization: Algorithms amp; analyses via monotone operators.* Cambridge University Press, 2023.