Implementing 2 First-Order Conic Solvers in Python

Caleb Ki & Sanjana Gupta

December 12th, 2017

Background

Templates for first-order conic solvers, or TFOCS for short, is a software package that provides efficient solvers for various convex optimization problems [@beck:etal:2011]. With problems in signal processing, machine learning, and statistics in mind, the creators developed a general framework and approach for solving convex cone problems. The standard routine within the TFOCS library is a four-step process. The notation used below is taken from the paper accompanying the software package.

The first step is to express the convex optimization problem with an equivalent conic formulation. The optimization problem should follow the form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & \mathcal{A}(x) + b \in \mathcal{K} \end{aligned} \tag{1}$$

In the problem above, $x \in \mathbb{R}^n$ is the vector we are optimizing under the convex objective function f. It is important to note that f does not have to be a smooth function. $\mathcal{A}$ is a linear operator from $\mathbb{R}^n$ to $\mathbb{R}^m$, $b \in \mathbb{R}^m$ is simply a vector, and $\mathcal{K}$ is a convex cone in $\mathbb{R}^m$.

The two main hurdles to developing efficient first-order solutions for convex optimization problems of this form are that f is not restricted to the set of smooth functions and that finding feasible points under this conic constraint is often an expensive computation. These hurdles are circumvented by finding and solving the dual problem of the given convex optimization problem which leads us to the second step of the process, turning the problem into its dual form. The dual of the form given by equation 1 is as follows:

$$\begin{aligned} \underset{\lambda}{\text{maximize}} \quad & g(\lambda) \\ \text{subject to} \quad & \lambda \in \mathcal{K}^* \end{aligned} \tag{2}$$

g is simply the dual function given by $g(\lambda) = \inf_x f(x) - \sum_{i=1}^{m} \langle \mathcal{A}(x) + b, \lambda \rangle$, and $\mathcal{K}^* = \{\lambda \in \mathbb{R}^m : \langle \lambda, x \rangle \geq 0 \, \forall x \in \mathcal{K}\}$ is the dual cone.

The dual problem is not directly solved at this step. This is because the dual function is generally not differentiable for the class of problems we are considering. Further, directly using subgradient methods is not efficient since these methods converge very slowly. In order to convert this to a problem that can be efficiently optimized, we apply a smoothing technique which modifies the primal objective function and instead solves the following problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_\mu(x) \triangleq f(x) + \mu d(x) \\ \text{subject to} \quad & \mathcal{A}(x) + b \in \mathcal{K} \end{aligned} \tag{3}$$

Here μ is a positive scalar and $d(x)$ is a strongly convex function called the *proximity function* which satisfies $d(x) \geq d(x_0) + \frac{1}{2}\|x - x_0\|^2$ for some fixed $x_0 \in \mathbb{R}^n$. The dual of this problem is

$$\begin{aligned} \underset{\lambda}{\text{maximize}} \quad & g_\mu(\lambda) \\ \text{subject to} \quad & \lambda \in \mathcal{K}^* \end{aligned} \tag{4}$$

where g_μ is a smooth approximation of g. In many cases, the dual can be reduced to the following unconstrained problem:

$$\underset{z}{\text{maximize}} \quad -g_{sm}(z) - h(z) \tag{5}$$

This is known as the *composite form*. Here, $z \in \mathbb{R}^m$ is the optimization variable, g_{sm} is a smooth convex function and h is a non-smooth (possibly extended-value) convex function. Finally, we can efficiently solve both the smooth dual and the composite problems using optimal first order methods such as gradient descent.

For both the problems (4), (14), given a sequence of step sizes $\{t_k\}$, the optimization begins with a point $\lambda_0 \in \mathcal{K}^*$ and has the following updating rule respectively:

$$\lambda_{k+1} \leftarrow \arg\min_{\lambda \in \mathcal{K}^*} \|\lambda_k + t_k \nabla g_\mu(\lambda_k) - \lambda\|_2$$

$$z_{k+1} \leftarrow \arg\min_z \; g_{sm}(z_k) + \langle \nabla g_{sm}(z_k), z - z_k \rangle + \frac{1}{2t_k}\|z - z_k\|^2 + h(z).$$

The approximate primal solution can then be recovered from the optimal value of λ as follows:

$$x(\lambda) \triangleq \arg\min_x \; f(x) + \mu d(x) - \langle \mathcal{A}(x) + b, \lambda \rangle \tag{6}$$

Equation 6 shows that the fundamental computational primitive in this method is the efficient minimization of the sum of a linear term, a proximity function, and a non-smooth function.

Implementation

As stated previously, TFOCS, when given the correct conic formulation of an optimization problem, will apply various first-order solvers to the smoothed dual problem. For this project, we port 2 of the 6 available first-order solvers in TFOCS software package from Matlab into Python. Specifically, the 2 solvers are the default method in TFOCS, the single-projection method developed by @ausl:tebo:2006, and the dual-projection method developed by @lan:etal:2011. Given a smooth function, the gradient of the smooth function, a nonsmooth function, and an initial point, the solver returns the optimal vector and the primal solution.

Motivation

For the purposes of this project, we assume that the user has expressed the optimization problem of concern in the correct specific conic form. Once the problem has been wrangled into the correct form, they can be solved through first-order methods. We assume that the optimization problem has been massaged into the following unconstrained form:

$$\text{minimize } \phi(z) \triangleq g(z) + h(z)$$

which is simply equation 14 except we have flipped the signs and turned the maximization problem into a minimization problem. To be notationally consistent we fix the update rule for z_{k+1} provided in the background section to

$$z_{k+1} \leftarrow \arg\min_{z} g(z_k) + \langle \nabla g(z_k), z - z_k \rangle + \frac{1}{2t_k} \|z - z_k\|^2 + h(z), \tag{7}$$

where t_k is the step size. @beck:etal:2011 assert that to ensure global convergence the following inequality must be satisfied:

$$\phi(z_{k+1}) \leq g(z_k) + \langle \nabla g(z_k), z_{k+1} - z_k \rangle + \frac{1}{2t_k} \|z_{k+1} - z_k\|^2 + h(z_{k+1}). \tag{8}$$

If we assume that the gradient of g satisfies,

$$\|\nabla g(x) - \nabla g(y)\|_* \leq L\|x - y\| \; \forall x, y \in \text{dom}\phi$$

(i.e., it satifies a generalized Lipschitz criterion), then convergence condition 8 holds for all $t_k \leq L^{-1}$. Of course L is difficult to find. Rather than finding the exact value for L, the solvers us an estimate L_k at each iteration which is explained in detail below. Returning to the main point, all the first-order solvers discussed in @beck:etal:2011 including the 2 we implement are based on update rule 7 and convergence criterion 8.

The Algorithm

The repeated calls to a generalized projection as described above manifests itself in the algorithm below. Again, we are assuming that the user has provided the smooth function g and its gradient, the nonsmooth function h and its prox function, a tolerance level for convergence, and a starting point x_0.

Lan, Lu, and Monteiro's modification of Nesterove's 2007 algorithm follows the same algorithm except we replace *line 8* with the following call:

$$z_{k+1} \leftarrow \arg\min_z \langle \nabla g(y_k), z \rangle + \frac{1}{2} L_k \|z - y_k\|^2 + h(z).$$

The key difference between these two similar variants is that, Lan, Lu, and Monteiro's method (LLM) requires 2 projections where Auslender and Teboulle's (AT) only requires 1. Of course, two projections per iteration is more computationally taxing, so we only prefer the two projection method in the case that it can reduce the number of iterations significantly.

Algorithm 1 Auslender and Teboulle's Algorithm

```
Require: $z_0 \in \mathrm{dom}\theta$, $\alpha \in (0,1]$, $\beta \in (0,1)$, $\gamma$, tol $> 0$
 1: $\bar{z}_0 \leftarrow z_0$, $\theta_{-1} \leftarrow 1$, $L_{-1} \leftarrow 1$
 2: for $k = 0, 1, 2, \ldots$ do
 3:     $L_k = \alpha L_{k-1}$
 4:     loop
 5:         $\theta_k \leftarrow 2/(1 + (1 + 4L_k/\theta_{k-1}^2 L_{k-1})^{1/2})$
 6:         $y_k \leftarrow (1 - \theta_k) z_k + \theta_k \bar{z}_k$
 7:         $\bar{z}_{k+1} \leftarrow \arg\min_z \langle \nabla g(y_k), z \rangle + \frac{1}{2}\theta_k L_k \|z - \bar{z}_k\|^2 + h(z)$
 8:         $z_{k+1} \leftarrow (1 - \theta_k) z_k + \theta_k \bar{z}_{k+1}$
 9:         if $g(y_k) - g(z_{k+1}) \geq \gamma g(z_{k+1})$ then
10:             $\hat{L} \leftarrow 2(g(z_{k+1}) - g(y_k) - \langle \nabla g(y_k), z_{k+1} - y_k \rangle)/\|z_{k+1} - y_k\|_2^2$
11:         else
12:             $\hat{L} \leftarrow 2|\langle y_k - z_{k+1}, \nabla g(z_{k+1}) - \nabla g(y_k) \rangle|/\|z_{k+1} - y_k\|_2^2$
13:         end if
14:         if $L_k \geq \hat{L}$ then break
15:         end if
16:         $L_k \leftarrow \max\{L_k/\beta, \hat{L}\}$
17:     end loop
18:     if $\|z_k - z_{k-1}\|/\max\{1, \|z_k\|\} \leq$ tol then break
19:     end if
20: end for
```

Justification

In the following section, we go through several lines in our algorithm to justify and clarify what the solvers are actually doing. The two main components of the algorithm are how the solvers are updating the step size (backtracking) and how the solvers are updating the solution.

Updating Step Size

We begin with a discussion about step size (*lines 3,9-16*). Generally it's very difficult to calculate L, and the step size $\frac{1}{L}$ is often too conservative. While the performance can be improved by reducing L, reducing L too much can cause the algorithm to diverge. All these problems are simultaneously resolved by using *backtracking.* Backtracking is a technique used to find solutions to optimization problems by building partial candidates to the solution called *backtracks.* Each backtrack is dropped as soon as the algorithm realizes that it cannot be extended to a valid solution. Applying this technique to the Lipschitz constant in our problem, we estimate the global constant L by L_k which preserves convergence if the following inequality holds:

$$g(z_{k+1}) \leq g(y_k) + \langle \nabla g(y_k), z_{k+1} - y_k \rangle + \frac{1}{2} L_k \|z_{k+1} - y_k\|^2. \tag{9}$$

If $g(z_{k+1})$ is very close to $g(y_k)$, equation (9) suffers from severe cancellation errors. Generally, cancellation errors occur if $g(y_k) - g(z_{k+1}) < \gamma g(z_{k+1})$ where the threshold for γ is around 10^{-8} to 10^{-6}. If we believe that the algorithm will suffer from cancellation errors, we use the following inequality to ensure convergence:

$$|\langle y_k - z_{k+1}, \nabla g(z_{k+1}) - \nabla g(y_k)\rangle| \leq \frac{1}{2} L_k \|z_{k+1} - y_k\|^2. \tag{10}$$

Note that inequalities (9) and (10) automatically hold in the case when $L_k \geq L$, so the solver only needs to check if inequalities (9) or (10) hold (based on whether $g(y_k) - g(z_{k+1}) < \gamma g(z_{k+1})$). If it is the case the relevant inequality holds then we need not update L_k further. If however the inequality does not hold we need to increase L_k until it does (i.e., backtrack until we have satisfied the convergence criterion). As part of this backtracking process we introduce $\hat{L}$ which is the smallest value of L_k that satisfies the relevant inequality (9) or (10) at the kth iteration holds or not). L_k is then updated as $\max\{\frac{L_k}{\beta}, \hat{L}\}$. Here, $\hat{L}$ is obtained by changing the inequalities (9), (10) to equalities and solving for L_k. This process of checking and updating L_k is repeated until L_k satisfies inequalities (9) or (10).

In every iteration we try to reduce the value of the Lipschitz estimate L_k (*line 3*). This is done to improve the performance of the algorithm and is achieved by updating $L_k = \alpha L_{k-1}$ for some fixed $\alpha \in (0, 1]$. Reducing L_k at each iteration can of course lead to an increased number of backtracks which we try to minimize by picking an appropriate value of α. For these kinds of solvers, generally $\alpha = 0.9$ is used which is what we have set α to be.

Intertwined with updating L_k is the problem of updating θ_k (*line 5*). First we establish bounds on the prediction error at the $(k+1)$st iteration as follows:

$$\phi(z_{k+1}) - \phi(z^\star) \leq \frac{1}{2} L {\theta_k}^2 \|z_0 - z^\star\|^2 \leq 2 \frac{L}{k^2} \|z_0 - z^\star\|^2 \tag{11}$$

This shows that the bound on the error is directly proportional to $L_k {\theta_k}^2$. In a simple backtracking step, as we increase L_k (by at least a factor of $1/\beta$), the bound on the error increases too. This can be avoided by updating θ_k along with L_k in each iteration by using the following inequality which ensures that convergence is preserved:

$$\frac{L_{k+1} {\theta_{k+1}}^2}{1 - \theta_{k+1}} \geq L_k {\theta_k}^2 \tag{12}$$

Solving for θ_{k+1} gives the update as in *line 5* of algorithm 1.

Updating z_{k+1}

The algorithm states in *line 7* that to update $\bar{z}_k$ we must find the arg min of some function of the gradient of g and nonsmooth function h. This can be simply reduced to the proximity function for h with step size $\frac{1}{L}$ evaluated at $\bar{z}_k - \frac{\nabla g(y_k)}{L_k \theta_k}$ where y_k is a linear combination of z_k and $\bar{z}_k$. The proximity operator *prox* of a convex function h at x is defined as the unique solution to the following:

$$\text{prox}_{t,h}(x) = \arg\min_z \; h(y) + \frac{1}{2t} \|x - z\|^2 \tag{13}$$

Here t is the step size. The update in *line 7* is then equivalent to a proximity function as follows:

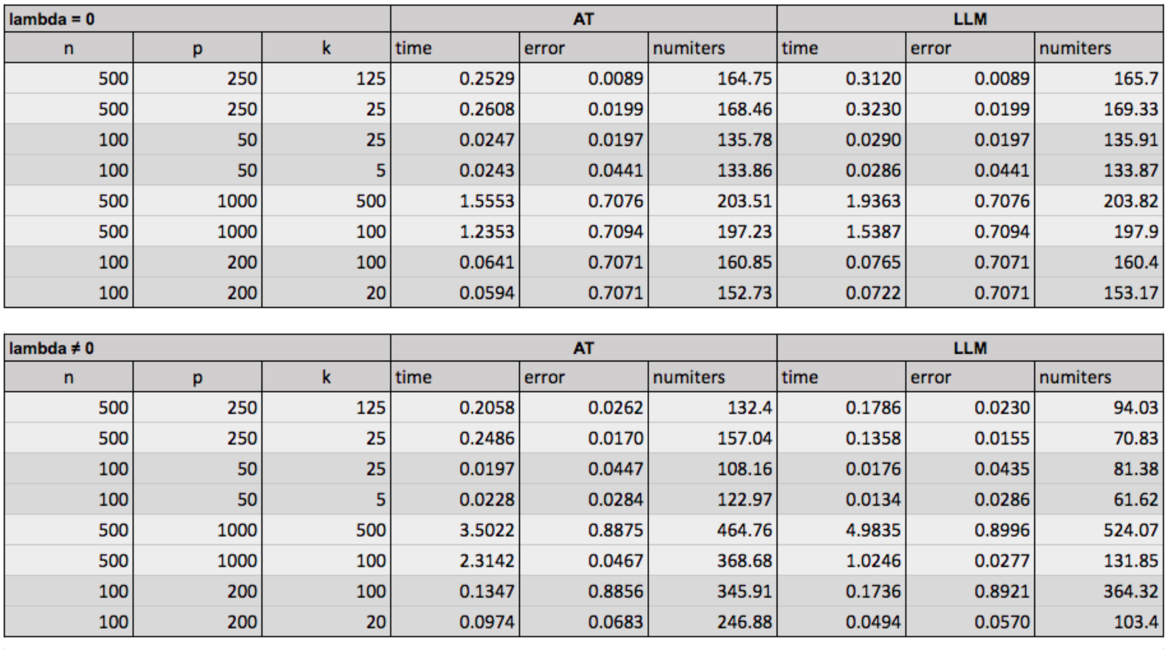
$$
\begin{aligned}
&\arg\min_z \, \langle \nabla g(y_k), z\rangle + \frac{L_k\theta_k}{2}\|z-\bar{z}_k\|^2 + h(z) \\
&= \arg\min_z \, h(z) + \frac{L_k\theta_k}{2}\left(\frac{2}{L_k\theta_k}\langle \nabla g(y_k), z\rangle + \|z\|^2 - 2\langle \bar{z}_k, z\rangle + \|\bar{z}_k\|^2\right) \\
&= \arg\min_z \, h(z) + \frac{L_k\theta_k}{2}\left(2\langle\frac{\nabla g(y_k)}{L_k\theta_k} - \bar{z}_k, z\rangle + \|z\|^2 + \|\bar{z}_k\|^2\right) \\
&= \arg\min_z \, h(z) + \frac{L_k\theta_k}{2}\left(2\langle\frac{\nabla g(y_k)}{L_k\theta_k} - y_k, z\rangle + \|z\|^2\right) + \frac{L_k\theta_k}{2}\|y_k\|^2 \\
&= \arg\min_z \, h(z) + \frac{L_k\theta_k}{2}\left(\|\frac{\nabla g(y_k)}{L_k\theta_k} - \bar{z}_k + z\|^2 - \|\frac{\nabla g(y_k)}{L_k\theta_k} - y_k\|^2\right) + \frac{L_k\theta_k}{2}\|\bar{z}_k\|^2 \\
&= \arg\min_z \, h(z) + \frac{L_k\theta_k}{2}\|\bar{z}_k - \frac{\nabla g(y_k)}{L_k\theta_k} - z\|^2 \\
&= \operatorname{prox}_{\frac{1}{L_k\theta_k},h}\left(\bar{z}_k - \frac{\nabla g(y_k)}{L_k\theta_k}\right)
\end{aligned}
$$

This clearly holds because $\frac{L_k\theta_k}{2}\|\bar{z}_k\|^2 - \frac{L_k\theta_k}{2}\|\frac{\nabla g(y_k)}{L_k\theta_k} - y_k\|^2$ is independent of z (i.e., we are able to treat it has a constant), and thus it does not affect the optimization. Therefore, we can drop this term to get the proximity function in the last line. For Auslender and Teboulle's method, we take z_{k+1} to be a linear combination of y_k and $barz_{k+1}$ which was updated with a call to the prox function.

For the Lan, Lu, and Monteiro method, the update for z_k in *line 8* is not a linear combination of y_k and $\bar{z}_{k+1}$, but another projection which can be computed similarly using the proximity function as follows:

$$
\begin{aligned}
&\arg\min_z \, \langle \nabla g(y_k), z\rangle + \frac{L_k}{2}\|z-y_k\|^2 + h(z) \\
&= \arg\min_z \, h(z) + \frac{L_k}{2}\left(\frac{2}{L_k}\langle \nabla g(y_k), z\rangle + \|z\|^2 - 2\langle y_k, z\rangle + \|y_k\|^2\right) \\
&= \arg\min_z \, h(z) + \frac{L_k}{2}\left(2\langle\frac{\nabla g(y_k)}{L_k} - y_k, z\rangle + \|z\|^2 + \|y_k\|^2\right) \\
&= \arg\min_z \, h(z) + \frac{L_k}{2}\left(2\langle\frac{\nabla g(y_k)}{L_k} - y_k, z\rangle + \|z\|^2\right) + \frac{L_k}{2}\|y_k\|^2 \\
&= \arg\min_z \, h(z) + \frac{L_k}{2}\left(\|\frac{g(y_k)}{L_k} - y_k + z\|^2 - \|\frac{\nabla g(y_k)}{L_k} - y_k\|^2\right) + \frac{L_k}{2}\|y_k\|^2 \\
&= \arg\min_z \, h(z) + \frac{L_k}{2}\|y_k - \frac{\nabla g(y_k)}{L_k} - z\|^2 \\
&= \operatorname{prox}_{\frac{1}{L_k},h}\left(y_k - \frac{\nabla g(y_k)}{L_k}\right)
\end{aligned}
$$

Since the term $\frac{L_k}{2}\|y_k\|^2 - \frac{L_k}{2}\|\frac{\nabla g(y_k)}{L_k} - y_k\|^2$ is independent of z, it does not affect the minimization and can be dropped to obtained the proximity function as in the last line.



lambda = 0			AT			LLM		
n	p	k	time	error	numiters	time	error	numiters
500	250	125	0.2529	0.0089	164.75	0.3120	0.0089	165.7
500	250	25	0.2608	0.0199	168.46	0.3230	0.0199	169.33
100	50	25	0.0247	0.0197	135.78	0.0290	0.0197	135.91
100	50	5	0.0243	0.0441	133.86	0.0286	0.0441	133.87
500	1000	500	1.5553	0.7076	203.51	1.9363	0.7076	203.82
500	1000	100	1.2353	0.7094	197.23	1.5387	0.7094	197.9
100	200	100	0.0641	0.7071	160.85	0.0765	0.7071	160.4
100	200	20	0.0594	0.7071	152.73	0.0722	0.7071	153.17

lambda ≠ 0			AT			LLM		
n	p	k	time	error	numiters	time	error	numiters
500	250	125	0.2058	0.0262	132.4	0.1786	0.0230	94.03
500	250	25	0.2486	0.0170	157.04	0.1358	0.0155	70.83
100	50	25	0.0197	0.0447	108.16	0.0176	0.0435	81.38
100	50	5	0.0228	0.0284	122.97	0.0134	0.0286	61.62
500	1000	500	3.5022	0.8875	464.76	4.9835	0.8996	524.07
500	1000	100	2.3142	0.0467	368.68	1.0246	0.0277	131.85
100	200	100	0.1347	0.8856	345.91	0.1736	0.8921	364.32
100	200	20	0.0974	0.0683	246.88	0.0494	0.0570	103.4

Table 1: Simulation resuls for fitting LASSO to the datasets with $\lambda = 0$ and $\lambda = \|X^T\epsilon\|_\infty$.

The final piece to the algorithm is the stopping criterion. *Line 17* of the algorithm states that if the distance between sequential z_k's becomes sufficiently small (i.e., less than the given tolerance) then the algorithm has converged and we have found our optimal z. Usually, tolerance is taken to be 10^{-8}.

Simulation Study

In this section we compare the performance of our two different solvers through an example. In particular, we solve the Lasso problem on different simulated datasets and evaluate the two different solvers by their error rates, number of iterations, and run times. The outcome variable y was generated by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}_n(0, \mathbf{I})$$

where the independent variables $\mathbf{X}_i$ are iid Gaussian with mean 0 and variance 1. Further, $\boldsymbol{\beta} = [0_{p-k}, 10_k]^T$, that is, the last k $\mathbf{X}_i$'s have a nonzero effect on y whereas the the first $p-k$ $\mathbf{X}_i$'s have no relationship with y. 100 datasets were generated for all combinations of the following parameter settings, $n \in \{100, 500\}$, $p \in \{\frac{n}{2}, 2n\}$ and $k \in \{0.5p, 0.1p\}$ (giving us 8 different settings total). We fit two Lasso models for each simulation using both AT and LLM. The general form of a Lasso problem is as follows

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1 \tag{14}$$

We fit models corresponding to $\lambda = 0, \|X^T\epsilon\|_\infty$. Table 1 summarizes the performance of the 2 techniques; the calculations are averaged over the 100 simulations per setting.

The top half of each table corresponds to overdetermined cases whereas the bottom half corresponds to undetdetermined ones. As we can see, neither one of the methods performs better than the other always. In the case where $\lambda = 0$ (i.e. ordinary least squares), we observe that AT performs better than LLM irrespective of the ratio of the observations: variables or the sparcity of the data. While the errors for both the methods are almost the same, the average number of iterations are silimar which resuls in AT converging much faster as it only computes a single projection in every iteration. The second case is more interesting. In the overdetermined case, LLM performs better than AT in terms of both run time and error with the performance being significantly better for sparse data (k:p ratio 0.1). In the underdetermined case, however, we observe that AT outperforms LLM when the k:p ratio is 0.5. It has lesser number of iterations (making it almost 1.4 times as fast as LLM) and has lower error rates. This trend is particularly visible in the underdetrmined case for $\lambda = 0$ as well. Summarizing, we observe that \

- AT outperforms LLM in underdetermined cases when the **k:p ratio is 0.5**.
- Despite being more computationally heavy, LLM performs almost twice as fast for all the other datasets when λ is not zero.

References