A Computational Study: Primal-Dual Hybrid Gradient (PDHG) Algorithm for Linear Programming

The code `my_pdhg.m` implemented a **first-order method** for solving the following standard LP pair:

$\min c^T x, \quad \text{s.t. } Ax = b,\ x \geq 0, \quad \max b^T y, \quad \text{s.t. } A^T y \leq c,$

where b and c are nonzero vectors.

1 Main Algorithm

1.1 Inilitalization

- Initialize $\mathbf{x}$ and $\mathbf{y}$ arbitarily as long as $\mathbf{x} \geq \mathbf{0}$.
- In my code, I initialized both to be a zero vector.

1.2 Update Rule

- $\mathbf{x}^{k+1} = proj_{\mathbb{R}^n_+}(\mathbf{x}^k - s(c - A^T\mathbf{y}^k))$;
- $\mathbf{y}^{k+1} = \mathbf{y}^k + s(b - A(2\mathbf{x}^{k+1} - \mathbf{x}^k))$.

2 Algorithm Enhancement

2.1 Scaling

- Set the scale parameter $\alpha = \frac{\|\mathbf{b}\|}{\|\mathbf{c}\|}$.
- Do the scaling:
  - At the beginning of the algorithm: $\mathbf{c} \leftarrow \alpha \cdot \mathbf{c}$;
  - At the end of the algorithm: $\mathbf{y} \leftarrow \mathbf{y}/\alpha$.

2.2 Preconditioning

- $D_1 A D_2 (D_2^{-1}\mathbf{x}) = D_1\mathbf{b}$, where $D1$ and $D2$ are diagonal matrices. Correspondingly, $D_2 A^T D_1 (D_1^{-1}\mathbf{y}) \leq D_2\mathbf{c}$.
- We scale the matrix A to make it well conditioned: $A \leftarrow D_1 A D_2$.
- Correspondingly, we need to scale $\mathbf{b}$ and c: $\mathbf{b} \leftarrow D_1\mathbf{b}$ and $\mathbf{c} \leftarrow D_1\mathbf{c}$.
- Thus, at the end of the algorithm, we need to scale $\mathbf{x}$ and $\mathbf{y}$ back by $\mathbf{x} = D_2\mathbf{x}$ and $\mathbf{y} = D_1\mathbf{y}$.
- Refer to the paper proposed by Pock and Chambolle, we set $D_1 = diag\{\frac{1}{\sum_{i=1}^m |A_{i,j}|^{2-\beta}}, i = 1, \ldots, m\}^{-\frac{1}{2}}$, and $D_2 = diag\{\frac{1}{\sum_{j=1}^n |A_{i,j}|^{\beta}}, j = 1, \ldots, n\}^{-\frac{1}{2}}$. In the code, I set $\beta = 1.2$.

2.3 Adaptive step

- Set the initial stepsize $s = \frac{\gamma}{\|A\|}$ after scaling A in the precoditioning part, where $\gamma \in (0, 1)$. I set $\gamma = 0.99$ in my code to be greedy.
- After every iteration, we adapt the stepsize $\tilde{s}^{k+1} = max\{s, t \cdot \frac{\|d\mathbf{x}^{k+1}\|}{\|Ad\mathbf{x}^{k+1}\|}\}$, where $d\mathbf{x}^{k+1} = \mathbf{x}^{k+1} - \mathbf{x}^k$ and t is the shrink parameter in $(0, 1)$.

2.4 Restart

- Set $N = min\{3 \cdot n, \frac{maxit}{2}\}$ and do the restart for every N iterations from the average $\mathbf{x}$ and $\mathbf{y}$ of the past 1000 iterations.

3 Implementation Details

The code is highly optimized by the following tricks:

- **Only calculate necessary errors among the three.** If the primal relative error is not less than tolerance, and the error information is not needed to printe, then other two errors are not calculated. If the primal relative error is less than the tolerance, then we calculate the relative gap error and see whether we need to calculate the relative dual error.
- **Only calculate Ax and A^Ty once per iteration**. These are the bottleneck for the algorithm efficiency in every iteration and nothing can be done to avoid these two multiplications between large-scale matrix and vector.

4 Observations and Code Performance

- My code achieves the efficiency of the instructor's code for all but one cases (i.e. for the case m=400, n=4000, mine is slower than his by 0.09s. for the others, mine is much faster). The code maybe further fastered by choosing better hyperparameters.
- My obversation for the code behavior under different enhancement for different problems are as follows:
  - The **scaling and adaptive step** enhancement are crucial for all the problems for good performance.
  - For the **random problem of small size with very low tolerance** $1e-14$, the **restarting** enhancement is crucial to reach high accuracy.
  - For the **random problem of midium size with midium tolerance 1e-6**, the algorithm costs a lot of time since the matrices are much larger.
  - For the **image impainting problem**, I turn off the **restarting** part since it normally leads to longer elapsed time. Restarting is not necessary for this problem because the tolerance $1e-2$ is so large that the algorithm can easily converges.
  - For the **Israel problem**, the **preconditioning** enhancement is very crucial since the matrix A are special (maybe ill-conditioned or high-unbalanced elements).