ADMM-based Approach for the QCQP Problem

Jie Wang
School of Science and Engineering (pure mathematics major)
The Chinese University of Hongkong, Shenzhen
116010214@link.cuhk.edu.cn

I. Overview

Our final project aims to solve a large-scale non-convex QCQP problem with the Hessian matrix to be *sparse* and *negative definite*. In this report we consider a distributed method for solving this QCQP problems called *Alternating Direction Method of Multipliers* (ADMM). The results show that our implementation is able to correctly solve really large problems with a million dimensional QCQP problems.

II. Problem Formulation

The problem we wish to solve is

$$\begin{array}{ll}\min & f(x,y) := \frac{1}{2}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{y}) - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{y} \\ \text{s.t.} & h_1(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{2}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} - 1) = 0 \\ & h_2(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{2}(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 1) = 0 \\ & h_3(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = 0,\end{array}$$

where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ are the optimization variables. The terms $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}$ refer to the problem data. We further require that $\boldsymbol{A}$ to be sparse and negative definite.

Following the similar idea of ADMM [1], we consider solving the quadratic penalized version of the original problem:

$$\begin{array}{ll}\min & f(x,y) + \lambda_3(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}) + \frac{\rho}{2}\|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}\|_2^2 \\ \text{such that} & \frac{1}{2}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} - 1) = 0 \\ & \frac{1}{2}(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 1) = 0,\end{array} \tag{1}$$

where ρ is a penalty parameter and λ_3 refers to the *Lagrange multiplier* for the third constraint $h_3(\boldsymbol{x}, \boldsymbol{y})$. We can formulate an iteration for solving this problem:

$$\boldsymbol{x}^{k+1} = \arg\min_{\|\boldsymbol{x}\|=1} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x} + \lambda_3^k(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}^k) + \frac{\rho}{2}\|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}^k\|_2^2 \tag{2a}$$

$$\boldsymbol{y}^{k+1} = \arg\min_{\|\boldsymbol{y}\|=1} \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{y} + \lambda_3^k(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}^{k+1}) + \frac{\rho}{2}\|\boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}^{k+1}\|_2^2 \tag{2b}$$

$$\begin{aligned}\lambda_3^{k+1} &= \lambda_3^k + \rho((\boldsymbol{x}^{k+1})^{\mathrm{T}}\boldsymbol{y}^{k+1}) \\ \lambda_1^{k+1} &= -(\boldsymbol{x}^{k+1})^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + (\boldsymbol{x}^{k+1})^{\mathrm{T}}\boldsymbol{b} \\ \lambda_2^{k+1} &= -(\boldsymbol{y}^{k+1})^{\mathrm{T}}\boldsymbol{A}\boldsymbol{y} + (\boldsymbol{y}^{k+1})^{\mathrm{T}}\boldsymbol{c}\end{aligned} \tag{2c}$$

where $\boldsymbol{x}^k, \boldsymbol{y}^k$ are the kth iterate of the variables above, and the update rule for λ_i (associated for the constraint $h_i(\boldsymbol{x}, \boldsymbol{y}) = 0$) follows from the KKT condition. Here the stopping criteria is decided by the instructor:

$$\max\left\{\frac{\|\nabla_{\boldsymbol{x}} L(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\lambda})\|}{1+\|\boldsymbol{b}\|}, \frac{\|\nabla_{\boldsymbol{y}} L(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\lambda})\|}{1+\|\boldsymbol{c}\|}, \|h(\boldsymbol{x},\boldsymbol{y})\|\right\} \leq \text{tol.}$$

where $L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\lambda})$ refers to the *Augmented Lagrangian function* for the original problem.

The update rules (2a) and (2b) are equivalent to solve *trust-region subproblems* (TRS). As a result, the main computational challenge for this project is to solve the TRS.

III. Solving for the Trust-Region Subproblem

The subproblem for solving (2a) and (2b) is to

$$\min_{\|\boldsymbol{p}\|=1} \quad \boldsymbol{p}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{p} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{p} \tag{3}$$

where $\boldsymbol{p} \in \mathbb{R}^n$ is the optimization variable, $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ are symmetric. To guarantee both the convergence and reasonable running time of ADMM algorithm, we are required to find the exact global minimum to (3) in super-short time. Standard methods for solving the problem (3) exactly require either *factorization* or *inversion* of the matrix $\boldsymbol{Q}$. However, the cost of these operations scales poorly with the problem dimensionality. It is therefore of interest to explore the matrix-free methods guaranteed to solve (3) efficientlly.

Recently, a generalized eigenvalue characterisation for TRS is derived in [2] via solving a *single* generalized eigenvalue problem (GEP). This algorithm is shown to be extremely efficient for solving the TRS. In this project we extend this result by making use of the special structure of our specific problem setting.

A. Theoretical results

In this sub-section, we show that how the optimal solution for a so-called *easy case* TRS can be obtained, and argue that it suffices only to deal with the easy case for this project setting.

Theorem III.1. *(From Theorem (3.3) in [2]) The optimal Lagrange multiplier (associated with the optimal solution to (3)) is the* ***largest real eigenvalue*** *λ^* of the $2n \times 2n$ matrix pencil $\boldsymbol{M}_0 + \lambda\boldsymbol{M}_1$, where*

$$\boldsymbol{M}_0 = \begin{pmatrix}-\boldsymbol{I}_n & \boldsymbol{Q} \\ \boldsymbol{Q} & -\boldsymbol{g}\boldsymbol{g}^{\mathrm{T}}\end{pmatrix}, \quad \boldsymbol{M}_1 = \begin{pmatrix}\boldsymbol{0}_n & \boldsymbol{I}_n \\ \boldsymbol{I}_n & \boldsymbol{0}_n\end{pmatrix} \tag{4a}$$

Moreover, suppose the corresponding eigenvector for λ^* is $\begin{pmatrix}y_1 \\ y_2\end{pmatrix}_{2n\times 1}$. If $\boldsymbol{g}^{\mathrm{T}}\boldsymbol{y}_2 \neq 0$ (easy case), then a global optimal solution $\boldsymbol{p}^*$ is obtained by:

$$\boldsymbol{p}^* = -sign(\boldsymbol{g}^{\mathrm{T}}\boldsymbol{y}_2)\frac{\boldsymbol{y}_1}{\|\boldsymbol{y}_1\|} \tag{4b}$$

In our project setting, the vector $\boldsymbol{g}$ is randomly generated. When $\boldsymbol{g}^{\mathrm{T}}\boldsymbol{y}_2 = 0$ holds, then from proposition (4.1) in

[2], we imply $\boldsymbol{y}_1 = \boldsymbol{0}$. Or equivalently $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}\boldsymbol{y}_2 = \lambda^*\boldsymbol{y}_2$. Therefore, the hard case will occur only when the randomly generated vector $\boldsymbol{g}$ belongs to the set $\mathcal{N}(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}-\lambda^*\boldsymbol{I})^{\perp}$, where $\dim((\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} - \lambda^*\boldsymbol{I})^{\perp}) = n - 1$. Since *the probability for which the vector drawn randomly from* $\mathbb{R}^n$ *lies in a set with dimension* $n-1$ *is zero*, it makes no difference to assume that all TRS we aim to solve are the *easy case*.

IV. Practical Implementation

The algorithm of the TRS solver described in subsection (III-A) is summarized as follows:

Algorithm 1: Solve the TRS (3)

Input : Problem Data $\boldsymbol{Q}, \boldsymbol{g}$

Output: global minimum to the TRS (3)

1 Compute the rightmost eigenvalue λ^* of $\boldsymbol{M}_0 + \lambda\boldsymbol{M}_1$, and an eigenvalue $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ such that $\boldsymbol{M}_0 \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = -\lambda\boldsymbol{M}_1 \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$;

2 Obtain the global minimum by computing $\boldsymbol{p}^* = -\text{sign}(\boldsymbol{g}^{\mathrm{T}}\boldsymbol{y}_2)\frac{\boldsymbol{y}_1}{\|\boldsymbol{y}_1\|}$;

3 return $\boldsymbol{p}^*$;

a) Remark 1: In general, the dominant computation cost in solving the TRS is for solving the GEP. We use the MATLAB command `eigs` instead of `eig` , considering the large-scale of $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$. Specifically, the code for solving the GEP should be

```
[v,~] = eigs(@(x)MM0timesx1(c,x,rho,A,p),...
              2*n,-MM1,1,'lm',OPTS);
```

where `MM0timesx1(x)` is a function handle that computes the matrix-vector multiplication $\boldsymbol{M}_0\boldsymbol{x}$. This method takes the advantage of the special structure of $\boldsymbol{M}_0$ and $\boldsymbol{Q}$ to *accelerate the algorithm*, and *saves memory* over storing the large-scale matrix $\boldsymbol{M}_0$.

b) Remark 2: Moreover, our solver makes use of the eigenvector $[y_1; y_2]$ from the previous iteration as the initial guess for the eigenvalue problem in the next iteration, in order to achieve fast convergence.

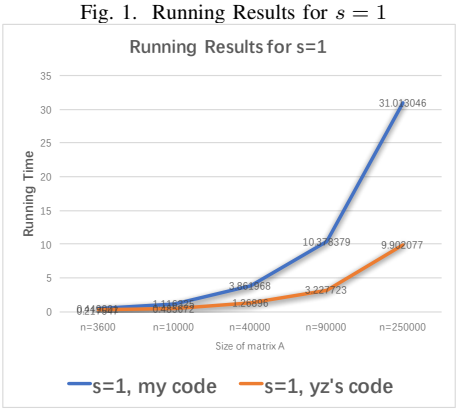
c) Remark 3: After some experiments, we find that this GEP can be solved quickly under the *Krylov subspace* with dimension $p = 5$. Thus we set the maximum size of Krylov subspace to be 5 for the `eigs` solver.

d) Remark 4: As suggested in [3], we enlarge the dual step size for updating the multiplier λ_3 from 1 to 1.95 to acclerate convergence. In this setting we can see that our algorithm achieves at least global linear convergence, which confirms the result shown in [3] as well.

e) Remark 5: We assume that the penalty parameter ρ should grow linearly with the norm of $\boldsymbol{b}$ divided by the size of $\boldsymbol{A}$. Over some experiments, we set the relation to be

$$\rho \cong 10.75 \times \frac{\|\boldsymbol{b}\|_2}{\|\boldsymbol{A}\|_F}$$

Fig. 1. Running Results for $s = 1$



f) Remark 6: We find that the update formula of dual variable (2c) is not the dominant cost, and therefore at each iteration we update this dual variable exactly after updating $\boldsymbol{x}$ (2a) or $\boldsymbol{y}$ (2b) to accelerate the covergence.

g) Remark 7: When the norm $\|\boldsymbol{b}\|$ and $\|\boldsymbol{c}\|$ is small, we choose the initial guess to be the solution to the *eigenvalue problem*

$$\min_{\|\boldsymbol{x}\|=1, \|\boldsymbol{y}\|=1} \quad \tfrac{1}{2}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{y})$$

Otherwise, we choose the initial guess for $\boldsymbol{y}$ to be $\boldsymbol{c}$ to minimize the affine term $-\boldsymbol{c}^{\mathrm{T}}\boldsymbol{y}$.

Here we present parital of our experiment results in Fig (1). For details you may refer to the appendix.

V. Conclusion

As we can see, the running time for our algorithm is roughly three times of that for instructor's. The main computation cost is the generalized eigenvalue solver. When computing a relatively large scale rightmost eigenvalues, the performance of MATLAB solver `eigs` is not so satisfying. My conjucture is that perhaps the idea for the *augmented Rayleigh-Ritz percedure* described in [4] can be extended into this problem, which looks promising. Due to the limited time, I don't have time to realize this idea. In the future I will think about it and discuss some ideas with the instructor.

References

[1] Stephen Boyd & Neal Parikh & Eric Chu & Borja Peleato & Jonathan Eckstein (2011) *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers* foundations and Trends in Machine Learning, 3, 1-122.

[2] Adachi, S., Iwata, S., Nakatsukasa, Y., & Takeda, A. (2017). *Solving the Trust-Region Subproblem By a Generalized Eigenvalue Problem* . SIAM Journal on Optimization, 27, 269-291.

[3] Guo, K & Han, D.R. & Wu, Ting-Ting. (2017). *Convergence of alternating direction method for minimizing sum of two nonconvex functions with linear constraints* . International Journal of Computer Mathematics. 94. 1653-1669.

[4] Wen, Zaiwen & Zhang, Yin. (2017). *Accelerating Convergence by Augmented Rayleigh–Ritz Projections For Large-Scale Eigenpair Computation*. SIAM Journal on Matrix Analysis and Applications. 38. 273-296.