Computational Mathematics for Learning and Data Analysis

Interior Point Methods

Javad Khalili
Matricola : 546677

December 22, 2020

1 Introduction

Here we try to implement an algorithm for optimizing an objective function with the following structure:

$$(P) \quad min \quad f(x) = x^T Qx + q^T x, \tag{1a}$$

$$s.t. \sum_{i \in I^h} x_i = 1, \tag{1b}$$

$$x \geq 0 \tag{1c}$$

where,

- $Q \in R^{n \times n}$ is a symmetric positive semi-definite matrix
- $x \in R^n$ is the variable in which minimization is done
- $q \in R^n$ is a random $n \times 1$ vector
- The index sets $I^h, i = 1, ..., k$ form a partition of the set $1, ..., n$ for putting under consideration that vector x is set of $k \in N$ disjoint simplices

However we will change the representation of (1b) to see how we have to consider this constraint to use while we try to solve the problem.
To solve this problem we use the primal-dual barrier approach[1] which compute solutions for a sequence of barrier problems for a decreasing sequence of barrier parameter μ converging to *zero*. Since objective function is twice differentiable such that $\bigtriangledown f = Qx + q$ and $\bigtriangledown^2 f = Q$ where Q is Hessian matrix and since it's a positive semidefinite matrix which by definition for each vector $x \in R^{n \times 1}$ $x^T Qx \geq 0$, the objective function is convex and there is only one local minimum point which is global minimum too.
Since in each iteration of this method we have to factorized the KKT matrix(Section 2.5.2) which is part of Newton's Method solution, computational complexity we expect is $O(L\frac{(n+k)^3}{3})$ where L is the number of iterations that the algorithm execute in order to reach optimal solution which normally is less than 100 and $\frac{(n+k)^3}{3}$ is the cost of factorizing coefficient matrix to solve KKT system which is the most costly operation in the algorithm.

2 Solution To The Problem

2.1 Notations

- $n \in N$ is the number of elements in vector x which defines the dimension of matrix Q and vector z also.
- $k \in N$ is the number of disjoint simplices.
- $X \in R^{n \times n}$ is diagonal matrix of vector $x.(diag(x))$
- $z \in R^{n \times 1}$ such that $z > 0$ is Lagrangian multiplier for bound constraint.

- $Z \in R^{n\times n}$ is the diagonal matrix of vector z.($diag(z)$)
- $e \in R^{n\times 1}$ is a n-dimension vector of ones.
- μ is the barrier parameter
- $b \in R^{k\times 1}$ is a k-dimensions vector of ones.
- $\lambda \in R^{k\times 1}$ such that $\lambda > 0$ is the Lagrangian multiplier for equality constraint.
- $E \in R^{k\times n}$ is full row rank matrix such that there is only one nonzero element (which is *one*) in each column, which will be chosen randomly in our implementation. this matrix help us to construct k disjoint simplices that required to be consider in order to solve the problem.

  Example:
  We assume that $n = 5$ and $k = 2$, so we will have $x \in R^{5\times 1}$ and $E \in R^{k\times n}$ such that there is one nonzero element(i.e. 1) in each column of E matrix. suppose E matrix is as follow:

$$E = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

  since $k = 2$ there will be 2 dijoint simplices $I^{(1)}$ and $I^{(2)}$. Based on matrix E the member of each simplex is as following:
  $I^{(1)} = \{x_1, x_2, x_3\}$ and $I^{(2)} = \{x_4, x_5\}$
  According to equality constrain(1b) the sum of items in each simplex must be equal to 1. so, $\sum_{i\in I^1} x_i = 1$ for $\{x_1, x_2, x_3\}$ and also $\sum_{i\in I^2} x_i = 1$ for $\{x_4, x_5\}$.
  As you can see there will a vector of ones with k-dimensions which call it b that is result of multiplication of matrix E and vector x.
  As mentioned before another representation of our equality constraint (1b) will be shown through out the solution which is going to be as following:

$$Ex = b$$

2.2 Input Parameters

The input parameters will be provided in form of a structure to the main algorithm. All the elements that has been shown in previous subsection (Notations) will be considered as input parameters but μ that is going to be calculated inside each iteration in algorithm.

2.3 Primal-Dual Barrier Approach

The Lagrangian dual problem corresponding to our problem is another quadratic problem given by:

$$(D) \quad max \quad d(x,\lambda) = b^T\lambda - x^TQx \quad , \tag{2a}$$

$$s.t. \quad E^T\lambda - Qx + z = q, \quad z \geq 0 \tag{2b}$$

If problem (P) has an optimal solution x^* then there exist λ^* and z^* such that the point (x^*, λ^*, z^*) is an optimal solution of problem (D). Conversely, if problem (D) has an optimal solution then problem (P) has an optimal solution. Moreover, the optimal solution values of both problems are identical. It is well-known that for all x and λ that are feasible for (P) and (D) we have

$$d(x,\lambda) \leq opt \leq f(x)$$

where *opt* denotes the optimal objective value for (P).
Optimality holds if and only if the complementary slackness relation

$$x^Tz = 0$$

is satisfied.

We consider the logarithmic barrier function

$$min \quad f_\mu(x) = x^TQx + q^Tx - \mu\sum_{i=1}^{n} log(x_i), \tag{3a}$$

$$s.t. \quad Ex - b = 0 \tag{3b}$$

for decreasing sequence of positive Barrier parameter $\mu > 0$ converging to *zero.*

the first and second order derivation of f_μ are

$$\bigtriangledown f(x,\mu) = Qx + q - \mu X^{-1}e$$

$$\bigtriangledown^2 f(x,\mu) = Q + \mu X^{-2}$$

Consequently, f is strictly convex on the relative interior of the feasible set. thus it achieves a minimum value at a unique point.and that unique point is characterized by the Karush-Kuhn-Tucker stationary condition.

2.4 KKT Conditions

Finding an optimal solution of Primal and Dual is equivalent to solving KKT conditions which are as follows:

$$Qx + q + E^T\lambda - z = 0, \quad z \geq 0 \tag{4a}$$

$$Ex - b = 0, \quad x \geq 0 \tag{4b}$$

$$XZe = 0 \tag{4c}$$

where Z is diagonal matrix having $\text{diag}(Z) = z$ and e is n-dimensional ones vector.

Replacing the third equation , by the parameterized equation $XZe = \mu e$, the central path for QP is defined by the solution set $(x(\mu), \lambda(\mu), z(\mu)), \mu > 0$ of the following system.

$$Qx + q + E^T\lambda - z = 0 \tag{5a}$$
$$Ex - b = 0 \tag{5b}$$
$$XZe = \mu e \tag{5c}$$

2.5 Newton's Method

The present method computes an approximate solution to the barrier problem for a fix value of μ that decrease the barrier parameter and continue the solution of the next barrier problem from the approximate solution of the previous one. In order to solve the barrier problem for a given fixed value μ of the barrier parameter a Newton's method is applied to the primal-dual equations.
Here j denotes the iteration counter for the inner loop.Given an iterate (x_j,λ_j,z_j) with $x_j, z_j > 0$,search directions (Δx_j , $\Delta\lambda_j$, Δz_j) obtained from linearization of KKT conditions at (x_j, λ_j, z_j).
We obtain the linear system (6)

$$\begin{bmatrix} -Q & E^T & -I \\ E & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x_j \\ \Delta\lambda_j \\ \Delta z_j \end{bmatrix} = \begin{bmatrix} R_x \\ R_\lambda \\ R_z \end{bmatrix} \tag{6}$$

where

$$R_x = Qx + q - E^T\lambda - z$$
$$R_\lambda = b - Ex$$
$$R_z = \mu e - XZe$$

2.5.1 Solving System (6)

The system could be solved by predictor-corrector strategy which described in [2] known as higher order predictor-corrector such that we first compute an affine direction in predictor step by solving system in following manner.
Instead of solving the non-symmetric linear system directly , the proposed method computes the solution equivalently by first the smaller, symmetric linear system

$$\begin{bmatrix} -(Q + X^{-1}Z) & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{aff} \\ \Delta\lambda_{aff} \end{bmatrix} = \begin{bmatrix} R_x - X^{-1}R_z \\ b - Ex \end{bmatrix} \tag{7}$$

and the vector Δz_{aff} is then obtained from

$$\Delta z_{aff} = X^{-1}R_z - X^{-1}Z\Delta x_{aff}$$

and for corrector step we use Δx_{aff} and Δz_{aff} by placing them on RHS of nonlinear part of system (7) which is R_z that becomes $R_z = \mu e - XZe - \Delta X_{aff}\Delta Z_{aff}e$ where ΔX_{aff} and ΔZ_{aff} are diagonal matrix of Δx_{aff} and Δz_{aff} respectively. so the linear system for corrector becomes

$$\begin{bmatrix} -(Q + X^{-1}Z) & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{corr} \\ \Delta \lambda_{corr} \end{bmatrix} = \begin{bmatrix} R_x - X^{-1}R_z \\ b - Ex \end{bmatrix} \tag{8}$$

and the vector Δz_{corr} is then obtained from

$$\Delta z_{corr} = X^{-1}R_z - X^{-1}Z\Delta x_{corr}$$

Solving the corrector could be repeated for P number of times by substituting Δx_{aff} and Δz_{aff} with the latest computed values of Δx_{corr} and Δz_{corr} respectively in R_z in order to improve computation efficiency that leads to decrease in number of iterations reaching to optimal solution that cause decrease in number of time to compute factorization of coefficient matrix that is the most costly operation of the algorithm by computing longer step length. Choosing the appropriate value for P is discussed in section 3.2.

since Q is at least positive semi definite and we add positive diagonal matrix $X^{-1}Z$ to it, this matrix is non singular.Also since matrix Q is symmetric, coefficient matrix is also symmetric and indefinite(many zero eigenvalues). The most effective strategy to find solution in this case is to use a symmetric indefinite factorization [3], because of indefiniteness of matrix we cannot use the Cholesky factorization.

2.5.2 Symmetric Indefinite Factorization

Since the coefficient matrices of predictor and corrector are the same we need to factorize it only once.
Applying symmetric indefinite factorization on matrix A(Coefficient matrix) of system (7) will have the following form:

$$PAP^T = LBL^T \tag{9}$$

Where L is a lower triangular matrix , B is a 1×1 or 2×2 block diagonal matrix and P is a permutation matrix. To find the solution (we call it x here) and b would be R.H.S for system (7) and (8) by this approach we have to do series of computation which are going to be as following:
first we have to compute $P^T * b$ and result can be stored in some variable let's say d, then we have to compute $L^{-1}d$ to e , and next $B^{-1}e$ to f , next $L^{-T}f$ to g and finally $x = P * g$. It can be shown as following too:

$$x = \underbrace{\underbrace{\underbrace{P\, L^{-T}\, \underbrace{B^{-1}\, \underbrace{L^{-1}\, \underbrace{P^T b}_{d}}_{e}}_{f}}_{g}}}$$

For implementing this in Matlab we can use $ldl()$ factorization Matlab function such that:

$$[L, B, P] = ldl(A);$$

and to solve $Ax = b$ system

$$x = P * (L'\backslash(B\backslash(L\backslash(P' * b))));$$

The time complexity of factorization (9) is $n^3/3$ flops and once it's computed we can solve the system(8)($Ax = b$) with $O(n^2)$.

2.5.3 Alternative Approach

An alternative to the direct factorization techniques is to use an iterative method to solve the system (7&8). The conjugate gradient(CG) method is not recommended for solving the system(7&8), because it can be unstable on system that are not positive definite. Better options are Krylove methods (for example : GMRES) for general linear of symmetric indefinite systems.

2.6 Choosing the Barrier Parameter μ

Given an iterate (x, λ, z), consider an interior method that computes primal-dual search direction by(4). The most common approach for choosing the barrier parameter μ that appears on the right hand side of (4) is to make it proportional to the current complementary value, that is,

$$\mu_{j+1} = \sigma \frac{x_j^T z_j}{n}$$

where $\sigma > 0$ is a *centering parameter* which $\sigma = \frac{1}{n}$ where n denotes the number of variables.

2.7 Computing Steplength

After the step $\Delta = (\Delta x, \Delta\lambda, \Delta z)$ has been determined, we compute primal and dual steplength and define the new iterate $(x_{j+1}, \lambda_{j+1}, z_{j+1})$ as

$$x_{j+1} = x_j + \alpha\Delta x \quad , \quad \lambda_{j+1} = \lambda_j + \alpha\Delta\lambda \quad , \quad z_{j+1} = z_j + \alpha\Delta z$$

where:

$$\alpha_{primal} = \beta\Big(max(1, \frac{-\Delta x_j}{x_j})\Big)^{-1} \tag{10a}$$

$$\alpha_{dual} = \beta\Big(max(1, \frac{-\Delta z_j}{z_j})\Big)^{-1} \tag{10b}$$

$$\alpha = min(\alpha_{primal}, \alpha_{dual}) \tag{10c}$$

for given $0 < \beta < 1$ (In this implementation $\beta = 0.995$).

2.8 Starting Point

As starting point for solver I used selecting starting point of LP approach in [1], in order to prevent potential failure of convergence. For this we can start by computing $\bar{x}, \bar{z}$ and $\bar{\lambda}$ such that:

$$\bar{x} = E^T(EE^T)^{-1}b,$$

$$\bar{\lambda} = (EE^T)^{-1}E\nabla f(x),$$

$$\bar{z} = \nabla f(x) - E^T \bar{\lambda}$$

Where $\bar{x}$ is vector of minimum norm that satisfies primal constraint ($Ex = b$) and , $\bar{z}$ and $\bar{\lambda}$ are vectors that satisfy dual constraint ($E^T\lambda + z = \nabla f(x)$) such that $\bar{z}$ has minimum norm.
$\bar{x}$ and $\bar{z}$ are computed such way that they satisfy linear constraints of primal and dual but yet not the sign constraints ($x > 0, z > 0$) because in general there will be non-positive elements in both vector which violate the constraints, for this reason we define:

$$\delta_x = max(-(1.5)min \quad \bar{x}, 0)$$

and

$$\delta_z = max(-(1.5)min \quad \bar{z}, 0)$$

and modify the vectors $\bar{x}$ and $\bar{z}$,

$$x^* = \bar{x} + \delta_x e,$$

$$z^* = \bar{z} + \delta_z e$$

where $e \in R^{n\times 1}$ vector of ones.
and to make sure the elements of x and z are not too close to zero we the following:

$$\bar{\delta_x} = \frac{{x^*}^T z^*}{2e^T z^*},$$

$$\bar{\delta_z} = \frac{{x^*}^T z^*}{2e^T x^*}$$

where $\bar{\delta_z}$ is the average size of elements of x^*, weighted by corresponding elements of z^*, and vice versa for $\bar{\delta_z}$, then we set the starting point as following:

$$x^0 = x^* + \bar{\delta_x} e,$$

$$\lambda^0 = \bar{\lambda},$$

$$z^0 = z^* + \bar{\delta_z} e$$

2.9 Stopping Criteria

The algorithm stops when it reaches to optimal solution of the problem where complementarity ($x^T z$) is less than optimality tolerance of $1e-8$ or it stops when the number of iterations exceeds the number assigned as $MaxIter$ by the user.
The optimal solution requires primal and dual infeasibility to be less than $1e-6$, and relative duality gap to become less than $1e-8$. These can be computed by

following:

$$primal - infeasibility = \frac{\|Ex - b\|}{\|b\| + 1}, \tag{11a}$$

$$dual - infeasibility = \frac{\|Qx + q - E^T\lambda - z\|}{\|q\| + 1}, \tag{11b}$$

$$relative - duality - gap = \frac{x^TQx + q^Tx - b^T\lambda}{|x^TQx + q^Tx| + 1}. \tag{11c}$$

2.10 Algorithm

The implementation algorithm is as follow:

Algorithm 1: Primal-Dual Interior Point Methods

```
Function PDIPM( Struct , MaxIter):
    Initial values and starting point define is structure that explained in
     section (2.8) along with σ = 1/n as centering value parameter
    while true do
        Check for optimality conditions by computing (11)
        if (11) satisfied then
            break;
        Check for stopping condition
        if i > MaxIter then
            break;
        Compute μ explained in section (2.6)
        Sovle predictor system (7) and Δz_aff to compute affine step
         direction
        for counter ≤ P do
            Solve corrector system(8) and Δz_corr for P number times
            Δx_aff = Δx_corr;
            Δz_aff = Δz_corr;
            counter++;
        Compute the step length using (10)
        Update the value of x, z and λ for next iteration
        x_{i+1} = x_i + αΔx_i
        λ_{i+1} = λ_i + αΔλ_i
        z_{i+1} = z_i + αΔz_i
    return [primal , x , status]
```

3 Numerical Experiment

In the following tables 1,2 and 3 the results of executing the algorithm for different instances with different sizes and structures are shown such that each instance has been created 5 times because of randomness of values and distribution of those values in sparse instances and then the average result has been recorded.

3.1 Generating Instances

For generating instances I used a Matlab structure that take two inputs from user (n and k) to create corresponding size instances. For creating a semi definite matrix (Q) for objective function we can take a rectangular $m \times n$ matrix and multiply it by it's transpose. I considered $m = n + \lceil \frac{n}{10} \rceil$ so that the rectangular matrix let's say G be a random $(n+m) \times n$ matrix, and $Q = G^T G$.

3.2 Choosing P

To find the appropriate value for P, I have ran the algorithm for 20 instances with different size, structure and sparsity and the average result is recorded in Table 1.

Instance			1-corr		2-corr		3-corr		4-corr		5-corr		6-corr	
No.	Name	nnz(Q)	iter	time(s)	iter	time(s)	iter	time(s)	iter	time(s)	iter	time(s)	iter	time(s)
1	Prob_1_1	dense	11.4	0.058	10	0.059	9.2	0.066	8.2	0.063	8	0.062	8	0.067
2	Prob_1_4	dense	10.2	0.043	9.2	0.059	9	0.063	8.4	0.059	8	0.062	8	0.063
3	Prob_2_1	dense	12.2	0.294	10.4	0.31	9	0.291	8.8	0.297	8	0.322	8	0.323
4	Prob_2_4	dense	11.8	0.381	10.4	0.405	9.4	0.414	8.4	0.386	8.2	0.424	8	0.432
5	Prob_3_1	dense	12.8	0.81	11.2	0.993	9.6	0.963	9.2	1.074	8.4	1.079	8.2	1.138
6	Prob_3_4	dense	11.8	1.454	10.2	1.391	9.2	1.474	8.8	1.529	8	1.444	8	1.63
7	Prob_4_1	dense	13.8	4.691	11.6	4.396	10	4.111	9.4	4.38	8.6	4.183	8.6	4.526
8	Prob_4_4	dense	12.6	7.007	10.6	7.191	9.4	6.818	9	6.968	8.4	6.897	8	6.958
9	Prob_5_1	dense	14.6	28.247	12.4	26.205	10.6	24.278	10	25.026	9	24.064	9.2	26.627
10	Prob_5_4	dense	13.8	51.284	11.8	46.298	10.6	43.776	9.6	41.52	9.2	41.886	8.6	40.949
11	Prob_1_1	496	-	-	10.2	0.07	-	-	8.8	0.069	8	0.073	8.2	0.073
12	Prob_1_4	524	10	0.058	8.8	0.072	8.2	0.075	8	0.08	7.8	0.085	7.4	0.078
13	Prob_2_1	2621	-	-	12.2	0.426	-	-	12	0.689	8	0.57	11.2	0.852
14	Prob_2_4	2583	11.2	0.64	9.6	0.668	8.4	0.72	8.2	0.836	7.6	0.869	7.2	0.717
15	Prob_3_1	5306	-	-	12.6	2.147	9.2	1.836	14	3.573	8	2.196	12.2	4.081
16	Prob_3_4	5333	11.2	2.628	9.8	3.077	8.8	3.361	8	3.561	7.4	3.807	7.2	4.079
17	Prob_4_1	10492	12.6	7.792	14.8	12.42	9.6	9.596	13.4	16.845	8	10.627	17.4	29.549
18	Prob_4_4	10504	12	15.96	10.2	17.267	9.2	18.773	8.4	19.852	8	21.615	7.4	22.238
19	Prob_5_1	21163	13.2	50.167	14.6	75.001	9.8	58.885	17.4	135.319	9	73.304	27.2	290.658
20	Prob_5_4	20975	12.6	107.04	10.6	113.682	9.6	124.937	8.6	128.692	8	136.221	8	154.531

Table 1: Results of Experiments for Determining P Value

As you can see above I have tested each instance with 1 to 6 corrections. "-" is written for those instances that algorithm failed to converged, mostly it

happened to the sparse instances where $k << n$ such as No. 11,13 and 15 with 1 and 3 corrections, because of this 1 and 3 could not be an appropriate value for P, 4 corrections mostly cause a good results by decreasing almost 30% in number of iterations but for instance No. 17 and 19 this is not the case and has inverse effect on number of iterations, 5 and 6 decreasing number of iterations in compare to 2 (in some cases 6 has negative effect) but for large instances they could take much more execution than 2. 2 could be the safest choice P value in general for all the instances because of first convergence for all the instances and second by decreasing rate of almost an average of 15% in number of iterations.

3.3 Tests

I also used a standard IP solver (*quadprog*) provided by Matlab to run the same instances on to make sure the solution is correct by comparing objective function values of both solvers and show performance difference between Algorithm 1 and the solver. The tests have been done on machine with Intel Core i5(dual core) 1.7 GHz processor and 8Gb of RAM.

Instance		**quadprog**		**PDIPM**						
Name	nnz(Q)	iter	t.time(s)	iter	t.time(s)	Fact.	Pred.	Corr.	Rel.dual	$\frac{\|x_q - x\|}{\|x\|}$
Prob_1_1	514	8.8	0.10	10.4	0.09	0.004	0.015	0.025	1.75e-9	6.72e-9
Prob_1_2	511	9.2	0.13	9.8	0.13	0.005	0.023	0.044	1.03e-10	2.80e-7
Prob_1_3	507	9	0.22	9,2	0.10	0.009	0.017	0.032	5.15e-10	2.47e-7
Prob_1_4	487	8.4	0.10	9	0.11	0.01	0.017	0.036	1.15e-10	2.47e-8
Prob_2_1	2560	9.8	0.19	12.2	0.51	0.115	0.127	0.279	2.04e-11	2.62e-8
Prob_2_2	2008	10	0.20	10.2	0.59	0.111	0.124	0.260	1.12e-11	2.73e-8
Prob_2_3	1840	10.2	0.24	9.8	0.78	0.164	0.165	0.337	1.12e-10	1.14e-6
Prob_2_4	1614	9.8	0.26	9.8	0.91	0.194	0.199	0.140	1.21e-11	2.11e-7
Prob_3_1	5305	10	0.48	14	3.05	0.568	0.675	1.473	2.94e-11	4.33e-7
Prob_3_2	5222	10.4	0.51	10.6	2.75	0.550	0.602	1.315	1.44e-11	4.74e-8
Prob_3_3	5302	10.6	0.72	10.8	3.98	0.876	0.875	1.908	2.02e-11	6.20e-7
Prob_3_4	5278	10.6	0.82	10.2	4.39	1.013	0.982	2.053	9.76e-11	1.65e-6
Prob_4_1	10605	10.8	1.73	14.2	14.83	3.35	3.204	6.902	3.80e-12	5.94e-7
Prob_4_2	10568	11	2.21	11.2	14.29	3.339	3.131	6.637	1.47e-11	7.26e-7
Prob_4_3	10574	10.8	3.01	10.4	18.88	4.642	4.221	8.714	1.04e-10	3.45e-7
Prob_4_4	10441	10.4	3.54	10.2	22.17	5.645	5.097	10.049	2.27e-12	1.25e-7
Prob_5_1	21141	11.2	10.34	14.2	9357	22.131	20.795	43.939	1.60e-11	4.23e-7
Prob_5_2	21170	10.6	12.47	11.4	92.18	22.31	20.735	43.438	1.78e-9	3.20e-7
Prob_5_3	21046	11	16.31	11	108.23	26.98	24.568	50.694	2.35e-11	1.36e-7
Prob_5_4	21192	10.8	23.72	10.8	151.62	38.791	35.589	70.108	4.92e-12	3.14e-7

Table 2: Average Results on Sparse Instances

Instance		quadprog		PDIPM						
Name	nnz(Q)	iter	t.time(s)	iter	t.time(s)	Fact.	Pred.	Corr.	Rel.dual	$\frac{\|x_q-x\|}{\|x\|}$
Prob_1_1	Dense	9.8	0.06	10	0.1	0.004	0.015	0.029	3.53e-11	3.14e-8
Prob_1_2	Dense	9.2	0.1	9.8	0.07	0.004	0.011	0.015	2.05e-11	8.16e-9
Prob_1_3	Dense	9.2	0.08	10	0.09	0.010	0.013	0.021	1.30e-12	5.46e-10
Prob_1_4	Dense	9	0.11	9.4	0.08	0.007	0.009	0.018	1.06e-12	2.13e-9
Prob_2_1	Dense	12.4	0.19	10.4	0.35	0.106	0.036	0.107	2.19e-12	9.04e-9
Prob_2_2	Dense	13.2	0.23	10.4	0.38	0.118	0.041	0.117	1.63e-13	3.47e-8
Prob_2_3	Dense	13.8	0.3	10.6	0.43	0.155	0.043	0.123	3.93e-11	7.44e-10
Prob_2_4	Dense	13.4	0.32	10.6	0.49	0.184	0.05	0.139	2.77e-14	1.42e-8
Prob_3_1	Dense	14.6	0.6	10.8	1.14	0.437	0.109	0.336	6.53e-13	8.7e-9
Prob_3_2	Dense	16.2	0.8	11.4	1.35	0.56	0.133	0.371	4.3e-14	1.86e-9
Prob_3_3	Dense	14.6	1.01	10.4	1.6	0.793	0.132	0.382	9.16e-15	3.68e-11
Prob_3_4	Dense	15	1.1	10.8	1.88	0.419	0.154	0.408	1.6e-14	2.54e-10
Prob_4_1	Dense	16.8	2.68	11.2	5.27	2.556	0.393	1.215	1.33e-14	4.36e-9
Prob_4_2	Dense	18	3.42	11.4	6.32	3.273	0.451	1.355	1.84e-14	1.53e-9
Prob_4_3	Dense	18.6	5.09	11.4	8.54	4.867	0.564	1.509	4.66e-15	2.61e-10
Prob_4_4	Dense	17.4	5.9	11.2	9.34	5.876	0.579	1.381	2.38e-15	2.7e-10
Prob_5_1	Dense	20	10.45	12.4	32.1	18.45	1.83	6.09	1.34e-14	2.23e-8
Prob_5_2	Dense	21	24.49	11.8	36.48	22.67	1.935	5.963	1.35e-14	2.32e-9
Prob_5_3	Dense	21.6	31.85	12.2	45.03	29.713	2.233	6.476	3.35e-15	6.34e-10
Prob_5_4	Dense	20.4	44.93	12	61.03	43.629	2.651	6.291	2.41e-15	4.12e-10

Table 3: Average Results on Dense Instances

To track the execution time (**t.time(s)**) of solvers I used Matlab profiler,and also for seeing how PDIPM's execution is spent for, different parts of it's implementation is profiled which are predictor backsolves(**Pred.**), corrector backsolves(**Corr.**) and the factorization of coefficient matrix(**Fact.**) such that the execution time of corrector depends on value of P, since I considered P=2 it take more or less twice time in compare to predictor which executes only once in each main iteration.
For the factorization part which as mentioned before is most costly part of the algorithm it almost take from 20%(for sparse matrices) up to 70%(for dense instances) of total execution time. The reason that factorization take only 20% of total execution time by sparse instances is that PDIPM spent more time to do the backsolves in compare to dense matrices.
$\frac{\|x_q-x\|}{\|x\|}$ shows the accuracy of PDIPM's solution(x) with respect to quadprog's solution(x_q) which is from *e-7 to *e-11 meaning 7 to 11 correct digits. Relative duality gap(**Rel.dual**) shows how different between primal and dual value since it's one of the optimality conditions mentioned in section 2.9 it should be less than a certain value.
In term of number of iterations, both solvers in case of sparse instances are almost identical but in case of dense instances PDIPM do less than quadprog, but in term execution time quadprog in both cases has the upper hand.

References

[1] Jorge Nocedal, Stephen J. Wright "*Numerical Optimization*," Second Edition

[2] David F. Shanno, Robert J. Vanderbei "*Interior − Point Methods for Nonconvex Nonlinear Programming : Ordering and Higher − Order Methods*," Statistics and Operation Research Princeton University.

[3] Glub G. H. and Loan C. F. V. " *Matrix Computations*," The Johns Hopkins University Press, Baltimore,Maryland, United States,2012.