MATHEMATICAL PROGRAMMING-II

PROJECT DOCUMENT:

SECOND ORDER CONE PROGRAMMING RELAXATION OF NON-CONVEX QUDRATIC OPTIMIZATION PROBLEMS

TEAM MEMBERS: -

MOHD ANWAR SAYEED -2010030109

SEEPANNA SAI CHATUR -2010030565

NITHISH KUMAR REDDY-2010030118

(DEEMED TO BE UNIVERSITY)

Department of Computer Science and Engineering

K L University Hyderabad,
Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India

ABSRTACT:

Nonconvex minimax separable quadratic optimization problems with multiple separable quadratic constraints and their second-order cone programming (SOCP) relaxations. Under suitable conditions, we establish exact SOCP relaxation for minimax nonconvex separable quadratic programs. We show that various important classes of specially structured minimax quadratic optimization problems admit exact SOCP relaxations under easily verifiable conditions. These classes include some minimax extended trust-region problems, minimax uniform quadratic optimization problems, max dispersion problems, and some robust quadratic optimization problems under bounded data uncertainty. The present work shows that nonconvex minimax separable quadratic problems with quadratic constraints, which contain a hidden closed and convex epigraphical set, exhibit exact SOCP relaxations.

INTRODUCTION

Nonconvex quadratic optimization problems involving multiple quadratic constraints are a class of important and computationally hard global optimization problems that arise in many practical applications. They have been extensively studied in the literature. Especially, in recent years, a great deal of attention has been focused on studying them using their semidefinite programming (SDP) relaxation problems and second-order cone programming (SOCP) relaxation problems [2, 3, 5, 10, 21, 23, 28, 27, 30]

$$(P)\ \inf_{x\in\mathbb{R}^n}\ \max_{1\le i\le p}\left\{\frac{1}{2}x^T\left(U\Sigma_iU^T\right)x + a_i^Tx + \alpha_i\right\}$$

$$\text{s.t.}\quad \frac{1}{2}x^T\left(U\Lambda_jU^T\right)x + b_j^Tx + \beta_j \le 0,\ j = 1,\ldots,q,$$

where U is an orthogonal matrix; Σ_i, $i = 1,\ldots,p$, and Λ_j, $j = 1,\ldots,q$, are diagonal matrices with diagonal elements given by $\sigma_1^i,\ldots,\sigma_n^i$ and $\mu_1^j,\ldots,\mu_n^j$, respectively, that is, $\Sigma_i = \mathrm{diag}(\sigma_1^i,\ldots,\sigma_n^i)$ and $\Lambda_j = \mathrm{diag}(\mu_1^j,\ldots,\mu_n^j)$; a_i, $i = 1,\ldots,p$, and b_j, $j = 1,\ldots,q$, are n-dimensional vectors; α_i, $i = 1,\ldots,p$, and β_j, $j = 1,\ldots,q$, are real numbers.

PROGRAMMING LANGUAGES/TECHNIQUES USED: -

Python with machine learning/SOCP relaxations and epigraphical sets

LITURATURE SURVEY

AUTHOR	YEAR	SURVEY
Masakazu Kojima	2015	This paper proposes a SOCP (second-order-cone programming) relaxation method, which strengthens the lift-and-project LP (linear programming) relaxation method by adding non-convex quadratic valid inequalities for the positive semidefinite cone involved in the SDP relaxation
RujunJiang(Fudan University) Duan Li(City University of Hong Kong)	2019	Second order cone constrained convex relaxations for nonconvex quadratically constrained quadratic programming
FranzRendl	2016	LP (linear programming) relaxation method by adding non-convex quadratic valid inequalities for the positive semidefinite cone involved in the SDP relaxation

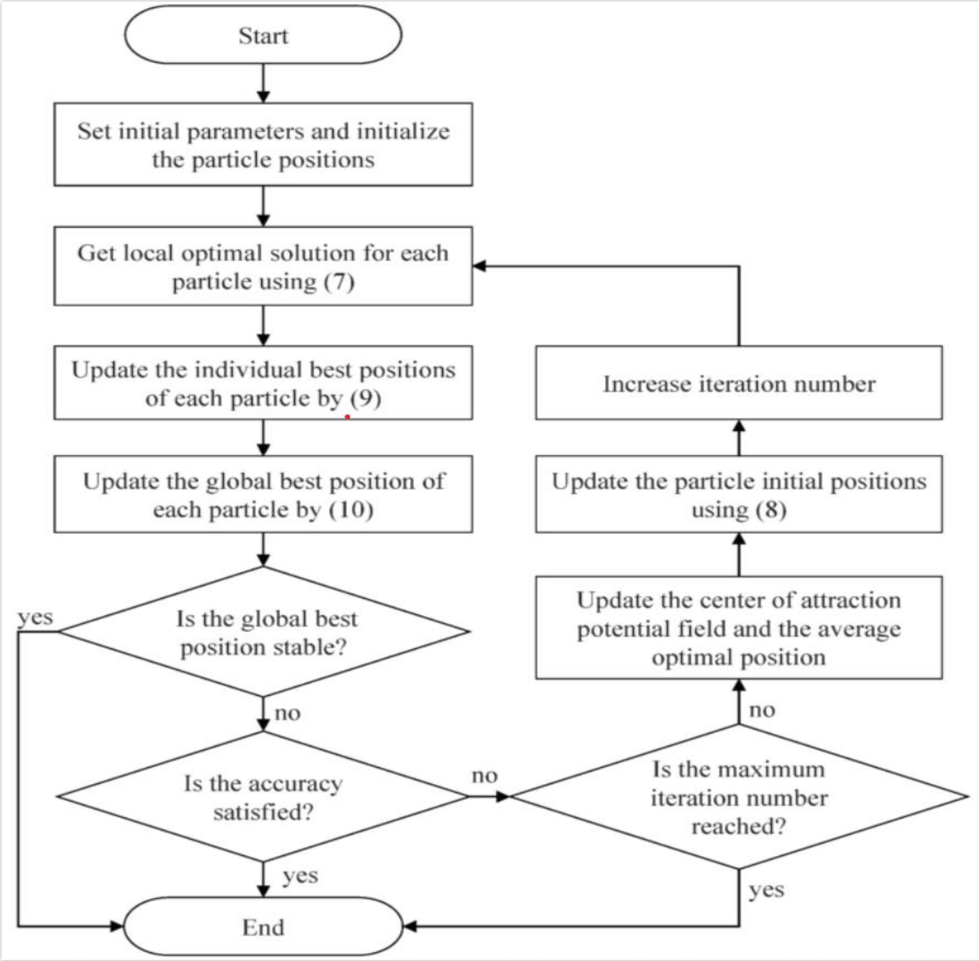
CONVEX OPTIMIZATION

- Convex optimization is a subfield of mathematical optimization that studies the problem of minimizing convex functions over convex sets. Many classes of convex optimization problems admit polynomial-time algorithms, whereas mathematical optimization is in general NP-hard.

NON-CONVEX OPTIMIZATION

- A non-convex optimization problem is any problem where the objective or any of the constraints are non-convex,. Such a problem may have multiple feasible regions and multiple locally optimal points within each region

FLOW CHART



Start
Set initial parameters and initialize the particle positions
Get local optimal solution for each particle using (7)
Update the individual best positions of each particle by (9)
Update the global best position of each particle by (10)
Is the global best position stable?
yes
no
Is the accuracy satisfied?
no
yes
Is the maximum iteration number reached?
no
yes
Update the center of attraction potential field and the average optimal position
Update the particle initial positions using (8)
Increase iteration number
End

CODE:

```
In [ ]: import pandas as pd
        import numpy as np

        import matplotlib.pyplot as plt
        %matplotlib inline

        import scipy
        from scipy.sparse import rand
        import scipy.sparse as sparse

        import cvxpy as cp

        from sklearn.model_selection import ParameterGrid

        import copy
        from time import time
        import random

        import warnings
        warnings.filterwarnings('ignore')
```

SDP

```
In [2]: def mySDP(M0, n, m, density):
            X = cp.Variable((n+1, n+1), symmetric=True)

            constraints = [cp.trace(get_matrix(n, density)@X.T) <= 0 for i in range(0, m)]
            constraints += [X>>0]
            constraints += [X[0][0] == 1]
            constraints += [G@cp.diag(X)[1:] <= h]

            problem = cp.Problem(cp.Minimize(cp.trace(M0@X.T)), constraints)

            start = time()
            problem.solve(solver = 'MOSEK')
            end = time()
            Time = end - start

            print('n =', n, '\t m =', m, '\t density =', density)
            print('Time:', Time)
            print(problem.value)
            return n, m, density, Time
```

```
In [ ]: N, M, Density = [100], [100], [0.5]
        Results = []

        for n in N:
            G = np.random.uniform(0, 1000, size=(n+2, n))
            h = G @ np.random.uniform(0, 1000, size=(n,))

            for density in Density:
                M0 = get_matrix(n, density)
                for m in M:
                    n, m, density, Time = mySDP(M0, n, m, density)
                    Results.append([n, m, density, Time])
```

SOCP

```
In [3]: def mySOCP(M0, n, m, density):
            Mp = []
            for i in range(0, m):
                Mp.append(get_matrix(n, density))

            X = cp.Variable((n+1, n+1), symmetric=True)
            constraints = [cp.trace(Mp[i]@X.T) <= 0 for i in range(0, m)]
            constraints += [X[0][0] == 1]
            constraints += [G@cp.diag(X)[1:] <= h]
            for j in range(1, n + 1):
                for k in range(0, j):
                    if Mp[0][k, j] < -0.000001 or Mp[0][k, j] > 0.000001:
                        constraints += [cp.norm(cp.hstack([X[k][k] - X[j][j], 2*X[k][j]])) <= X[k][k] + X[j][j]]

            problem = cp.Problem(cp.Minimize(cp.trace(M0@X.T)), constraints)

            start = time()
            problem.solve(solver = 'MOSEK')
            end = time()
            Time = end - start

            print('n =', n, '\t m =', m, '\t density =', density)
            print('Time:', Time)
            print(problem.value)
            return n, m, density, Time
```

```
In [9]: N, M, Density = [100], [100], [0.5]
        Results = []
        for n in N:
            G = np.random.uniform(0, 1000, size=(n+2, n))
            h = G @ np.random.uniform(0, 1000, size=(n,))
```

Main part

In [4]:

```python
def mySDP_id(M0, n, m, density):
    X = cp.Variable((n+1, n+1), PSD=True)

    constraints = [cp.trace(Mp[i]@X.T) <= 0 for i in range(0, m)]
    constraints += [X[0][0] == 1]
    constraints += [G@cp.diag(X)[1:] <= h]

    problem = cp.Problem(cp.Minimize(cp.trace(M0@X.T)), constraints)

    start = time()
    problem.solve(solver = 'MOSEK')
    end = time()
    Time = end - start

    print('n =', n, '\t m =', m, '\t density =', density)
    print('Time:', Time)
    print(problem.value)
    val = problem.value
    return n, m, density, Time, val
```

In [11]:

```python
def mySOCP_id(M0, n, m, density):
    X = cp.Variable((n+1, n+1), symmetric=True)

    constraints = [cp.trace(Mp[i]@X.T) <= 0 for i in range(0, m)]
    constraints += [X[0][0] == 1]
    constraints += [G@cp.diag(X)[1:] <= h]

    for j in range(1, n + 1):
        for k in range(0, j):
            if Mp[0][k, j] < -0.0000001 or Mp[0][k, j] > 0.0000001:
                constraints += [cp.norm(cp.vstack([X[k][k] - X[j][j], 2*X[k][j]])) <= X[k][k] + X[j][j]]
```

In [11]:

```python
def mySOCP_id(M0, n, m, density):
    X = cp.Variable((n+1, n+1), symmetric=True)

    constraints = [cp.trace(Mp[i]@X.T) <= 0 for i in range(0, m)]
    constraints += [X[0][0] == 1]
    constraints += [G@cp.diag(X)[1:] <= h]

    for j in range(1, n + 1):
        for k in range(0, j):
            if Mp[0][k, j] < -0.0000001 or Mp[0][k, j] > 0.0000001:
                constraints += [cp.norm(cp.vstack([X[k][k] - X[j][j], 2*X[k][j]])) <= X[k][k] + X[j][j]]

    problem = cp.Problem(cp.Minimize(cp.trace(M0@X.T)), constraints)

    start = time()
    problem.solve(solver = 'MOSEK')
    end = time()
    Time = end - start

    print('n =', n, '\t m =', m, '\t density =', density)
    print('Time:', Time)
    print(problem.value)
    val = problem.value
    return n, m, density, Time, val
```

In [12]:

```python
N, M, Density = [50, 100], [50], [0.1]
fResults = []
for n in N:
    G = np.random.uniform(0, 100, size=(n+3, n))
    h = G @ np.random.uniform(0, 100, size=(n,))

    for density in Density:
        M0 = get_matrix(n, density)
        for m in M:
            Results = []
            Mp = []
```

```
n = 50    m = 50          density = 0.1
Time: 1.807164192199707
-28567.588126761653
n = 50    m = 50          density = 0.1
Time: 0.3102538585662842
-28567.585759113666
n = 100          m = 50          density = 0.1
Time: 50.20961308479309
-193564.1780348802
n = 100          m = 50          density = 0.1
Time: 6.866625070571899
-193564.17182150309
```

	n	m	density	Time_SDP	value_SDP	Time_SOCP	value_SOCP
0	50	50	0.1	1.807164	-28567.588127	0.310254	-28567.585759
1	100	50	0.1	50.209613	-193564.178035	6.866625	-193564.171822

```
In [24]: fResults = []
density = 0

for n, m in zip([50, 50, 50, 100, 100, 100, 200, 200, 200], [50, 100, 200, 50, 100, 200, 50, 100, 200]):
    G = np.random.uniform(0, 100, size=(n+3, n))
    h = G @ np.random.uniform(0, 100, size=(n,))
    M0 = get_matrix(n, density)
    Results = []
    Mp = []

    for i in range(m):
        Mp.append(get_matrix42(n, density))

    n, m, density, Time, val = mySDP_id(M0, n, m, density)
    Results = [n, m, density, Time]

    n, m, density, Time, val = mySOCP_id(M0, n, m, density)
    Results.append(Time)
    fResults.append(Results)

to_p = pd.DataFrame(fResults, columns = ['n', 'm', 'density', 'Time_SDP', 'Time_SOCP'])
display(to_p)
to_p.to_csv('memorize_42.csv', index=False)
```

```
n = 50    m = 50          density = 0
Time: 2.0375208854675293
-1178.2387027880575
n = 50    m = 50          density = 0
Time: 0.2922179698944092
-1178.2384146622246
n = 50    m = 100         density = 0
Time: 1.9457674026489258
-1086.7229284910786
n = 50    m = 100         density = 0
Time: 0.3830084800720215
-1086.7229134090267
```

CONCLUSION:

we will established exact SOCP relaxations for nonconvex minimax separable quadratic optimization problems with multiple separable quadratic constraints under an epigraphical condition. We exploited hidden convexity in the form of a convex epigraphical set to achieve our results. We have also provided various classes of minimax problems for which our results hold under easily verifiable conditions

REFERENCES:

- Anstreicher, K.: Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. J. Glob. Optim. **43**(2–3), 471–484 (2009)
- Michale.: On convex relaxations for quadratically constrained quadratic programming. Math. Program. 136(2), 233–251 (2012)Google Scholar
- Masakazu:https://www.proquest.com/docview/215642889?pq-origsite=gscholar&fromopenview=true
- Beck, A., Eldar, Y.C.: Strong duality in nonconvex quadratic optimization with two quadratic constraints. SIAM J. Optim. 17(3), 844–860 (2006)Google Scholar
- Boyd, S., Vandenberghe, L.: Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization. In: Paulraj, A., Roychowdhury, V., Schaper, C.D. (eds.) Communications, Computation, Control, and Signal Processing, pp. 279–287. Springer, Berlin (1997)
- Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM (JACM) **42**(6), 1115–1145 (1995)
- Pardalos, P.M., Vavasis, S.A.: Quadratic programming with one negative eigenvalue is NP-hard. J. Glob. Optim. 1(1), 15–22 (1991)Google Scholar