Applied and Numerical Harmonic Analysis

Eugenio Hernández
Marco Maria Peloso
Fulvio Ricci
Fernando Soria
Anita Tabacco
Editors

The Mathematical Heritage of Guido Weiss

Birkhäuser

Chapter 15
On Low-Rank Convex-Convex Quadratic Fractional Programming

Ilya Krishtal and Brendan Miller

Abstract We present an efficient algorithm for solving fractional programming problems whose objective functions are the ratio of a low-rank quadratic to a positive definite quadratic with convex constraints. The proposed algorithm for these convex-convex problems is based on the Shen-Yu Quadratic Transform (Shen and Yu, IEEE Trans Signal Process 66(10):2616–2630, 2018) which finds stationary points of concave-convex sum-of-ratios problems. We further use elements of the algorithm proposed by Shen and Yu and the classic Dinkelbach approach to ensure convergence. We show that our algorithm performs better than previous algorithms for low-rank problems.

15.1 Introduction

Methods of fractional programming encompass a large range of techniques to solve problems of the form

$$\begin{aligned} \max \quad & F(x) = \sum_{m=1}^{M} \frac{N_m(x)}{D_m(x)} \\ \text{s.t.} \quad & x \in X \subset \mathbb{R}^n \end{aligned} \tag{15.1}$$

where $N_m(x), D_m(x) : \mathbb{R}^n \to \mathbb{R}$ are continuous functions, and X is a closed, convex set. A convention of the field, which we adopt throughout this paper, is that $N_m(x) \geq 0$ and $D_m(x) > 0$ for all $x \in X$. The problem is called *single-ratio* if $M = 1$ and a *sum-of-ratios* fractional programming problem if $M > 1$. Fractional programming problems arise in many different applications, such as finance and portfolio analysis,

I. Krishtal (✉) · B. Miller
Northern Illinois University, Dekalb, IL, USA
e-mail: ikrishtal@niu.edu; bmiller14@niu.edu

E. Hernández et al. (eds.), *The Mathematical Heritage of Guido Weiss*,
Applied and Numerical Harmonic Analysis,
https://doi.org/10.1007/978-3-031-76793-7_15

government contracting, and engineering (see [1, 2, 10, 11]). We were led to a single-ratio problem in the process of studying the problem of optimal sensor placement for dynamical sampling on graphs [8]. In particular, the relative error of reconstructing a signal on a graph from spatio-temporal samples in the presence of noise can be bounded above by a ratio of two quadratics depending on the eigenvalues of a certain frame operator.

Fractional programs have a long history. Below we outline a few milestones in the development of the theory.

In 1962, Charnes and Cooper showed in [3] that linear fractional programming problems, which are single-ratio fractional programming problems whose numerator, denominator, and constraints are linear, can be solved by simplex method.

In 1967, Dinkelbach [4] showed that any concave-convex single-ratio fractional programming problem can be solved efficiently by consecutively solving several concave maximization problems. The approach used by Dinkelbach was so useful that it has become standard and even seen many applications to fractional programs which are not concave-convex. In the latter case, however, the method involves solving a succession of non-concave maximization problems and becomes rather expensive.

In 1997, Lo and MacKinlay [11] proposed the problem of maximizing the ratio of two convex quadratic functions in the context of portfolio analysis. The following decade saw two papers published [7, 14] which presented algorithms for solving this problem exactly. Both follow the standard Dinkelbach approach, with different methods of handling the most expensive aspect of the problem: the need to solve multiple non-convex quadratic programming problems. To do so, the former paper by Gotoh and Konnoh implemented a branch-and-bound technique while the latter by Yamamoto and Konnoh iteratively approximated the quadratic by a piece-wise linear function, which is maximized by standard mixed integer linear programming techniques.

In recent years, there has also been much research on the sum-of-ratios problem. For example, [1, 2, 5] all produce branch-and-bound type algorithms for solving the concave-convex sum-of-ratios problem. Similarly, [10, 12] propose efficient algorithms for solving quadratically constrained quadratic sum-of-ratios type problems. Of particular interest to us is the Shen-Yu Quadratic Transform introduced in [13] which, like the Dinkelbach method, iteratively solves a concave programming problem to converge to a stationary point of the concave-convex sum-of-ratios problem.

In this paper, we propose an algorithm for solving the single-ratio convex-convex quadratic programming problem which can be effectively utilized when the numerator has low rank. We divide the feasible region into several subregions in which the Shen-Yu Quadratic Transform method can be applied successively until global convergence. Since the methods of Shen and Yu only guarantee convergence to a stationary point, we use the mixed integer linear programming techniques employed in [14] to check if a given stationary point is a global optimum. Although, as we will see empirically, this rather expensive procedure can often be omitted altogether (with a minimal chance of error). This yields a very efficient algorithm which, with high probability, converges to the global solution of a convex-convex quadratic programming problem by successively solving concave maximization problems. The only known algorithms for solving such problems involve successively maximizing a nonconvex quadratic programming problems.

The remainder of this paper is organized as follows. In Sect. 15.2, we state the problem we focus on and outline the classic Dinkelbach approach for ratio maximization. We then convert our problem into a sum-of-ratios problem and recall a recent Shen-Yu scheme for solving such problems. Section 15.3 is the centerpiece of the paper. In it, we present a natural way of subdividing the feasible region of our problem into a number of subregions thereby replacing a convex-convex sum-of-ratios problem with a finite number of much simpler concave-convex sum-of-ratios problems. This results in a region checking algorithm (Algorithm 15.3) which encompasses our approach to solving single-ratio quadratic convex-convex problems. Numerical experiments described in Sect. 15.4 illustrate the effectiveness of our approach in comparison with other algorithms. Finally, concluding remarks are presented in Sect. 15.5.

15.2 Basic Analysis

We study the problem of the form

$$\begin{aligned} \max \quad & F(x) = \frac{x^T Q x}{x^T P x} \\ \text{s.t.} \quad & Ax \le b \end{aligned} \tag{15.2}$$

where Q is an $n \times n$ positive semidefinite matrix, P is an $n \times n$ positive definite matrix, $A \in \mathbb{R}^{T \times n}$, $b \in \mathbb{R}^T$. We will denote the feasible region of this problem by $X \subset \mathbb{R}^n$. Such optimization problems are nonconcave in general, and thus require expensive algorithms to solve. We recall the standard Dinkelbach method which utilizes the function

$$\pi(\lambda) := \max_{x \in X} \; x^T (Q - \lambda P) x, \quad \lambda > 0.$$

It is convenient to introduce the following notation:

$$x(\lambda) = \arg\max_{x \in X} \; x^T (Q - \lambda P) x, \quad \lambda > 0.$$

Theorem 15.1 *The function $\pi(\lambda)$ is convex and strictly decreasing in λ. Furthermore, $\pi(\lambda) = 0$ if and only if $x(\lambda)$ maximizes (15.2) in X.*

The algorithms of [7, 14] are root-finding algorithms which use a scheme developed by Ibaraki [9] to search for the root of π. These algorithms become expensive because computing $\pi(\lambda)$ is a nonconvex quadratic programming problem, whose difficulty is larger for smaller values of λ. We will utilize this Theorem of Dinkelbach in our algorithm, but only to check if a local maximum of (15.2) is indeed a global maximum. We summarize the Ibaraki scheme here as Algorithm 15.1 for reference, but omit the explanation of convergence.

```
  input : Matrices $Q$ and $P$, linear inequality constraints $Ax \leq b$, and a tolerance $\varepsilon$.
1 Find $\lambda^u$ with $\pi(\lambda^u) < 0$ and $\lambda^l$ with $\pi(\lambda^l) > 0$;
2 repeat
3     Compute $\bar{\lambda}$ as follows:
          $$\bar{\lambda} = \begin{cases} -\frac{\pi(\lambda^u)}{\Delta\pi} + \lambda^u & \text{if } x(\lambda^u)^T Px(\lambda^u) + \Delta\pi \neq 0 \\ \frac{\pi(\lambda^u)}{x(\lambda^u)^T Px(\lambda^u)} + \lambda^u & \text{otherwise,} \end{cases}$$
      where $\Delta\pi = (\pi(\lambda^u) - \pi(\lambda^l))/(\lambda^u - \lambda^l)$;
4     Compute $\pi(\bar{\lambda})$;
5     Update $\lambda^l = \bar{\lambda}$ if $\pi(\bar{\lambda}) > 0$, or $\lambda^u = \bar{\lambda}$ if $\pi(\bar{\lambda}) < 0$;
6 until $|\pi(\bar{\lambda})| < \varepsilon$;
```

Algorithm 15.1: Interpolated binary search (Ibaraki scheme)

The algorithms presented in [7, 14] both utilize Algorithm 15.1, but employ different methods of computing $\pi(\bar{\lambda})$. We will use contemporary software to solve these non-convex quadratic programming problems to ensure the most accurate solutions.

Suppose now that Q can be written as

$$Q = \sum_{m=1}^{M} q_m q_m^T,$$

which is a sum of rank-one matrices. Note that Q always admits such a decomposition as we can take $M = rank(Q)$ and $q_m = \sqrt{\lambda_m} v_m$ where v_m is the eigenvector of Q associated to the nonzero eigenvalue λ_m. Then we may rewrite the objective function in the following way:

$$\begin{aligned} F(x) &= \frac{\sum_{m=1}^{M} x^T (q_m q_m^T) x}{x^T P x} \\ &= \sum_{m=1}^{M} \frac{\langle q_m, x \rangle^2}{x^T P x}. \end{aligned} \tag{15.3}$$

This reformulation of the objective function converts a single-ratio fractional programming problem (15.2) into a convex-convex sum-of-ratios fractional programming problem which we will refer to as (15.3).

Next, we summarize a method for suboptimally solving sum-of-ratios fractional programming problems (15.1) that was introduced in [13].

Definition 15.1 (See [13]) Given the sum-of-ratios fractional programming problem *(15.1)*, the Shen-Yu Quadratic Transform of this problem is defined to be

$$g(x, y) = \sum_{m=1}^{M} \left(2y_m\sqrt{N_m(x)} - y_m^2 D_m(x)\right). \tag{15.4}$$

Theorem 15.2 ([13]) *The sum-of-ratios problem* (15.1) *and its quadratic transform* (15.4) *maximization problem are equivalent. More precisely, the maximum of* (15.4) *occurs at* (x^*, y^*) *where* x^* *maximizes* (15.1) *and* $y^* = (y_m^*)_{m=1}^M$ *satisfies*

$$y_m^* = \frac{\sqrt{N_m(x^*)}}{D_m(x^*)}.$$

The following Lemma is useful for deriving the Shen-Yu scheme.

Lemma 15.1 *Let* $g(x, y)$ *be as in* (15.4)*. If* $x_0 \in X$*, then* $y_0 := \arg\max_{y\in\mathbb{R}^M} g(x_0, y)$ *can be found analytically and is given by* $y_0 = (y_m^0)_{m=1}^M$*, where*

$$y_m^0 = \frac{\sqrt{N_m(x_0)}}{D_m(x_0)}.$$

Proof It suffices to maximize each term in the sum in (15.4) individually. Clearly, the vertex of the quadratic $f(y_m) = 2y_m\sqrt{N_m(x_0)} - y_m^2 D(x_0)$ occurs at the point

$$y_m^0 = \frac{\sqrt{N_m(x_0)}}{D_m(x_0)}$$

as desired. □

We may now state and prove a theorem from which, when taken together with Theorem 15.2, an algorithm for suboptimally solving (15.1) is naturally derived.

Theorem 15.3 (Shen, Yu [13]) *Consider the sum-of-ratios problem* (15.1) *and suppose that* $x_0 \in X$*. Let* $y_0 = (y_m^0)_{m=1}^M$ *with* $y_m^0 = \sqrt{N_m(x_0)}/D_m(x_0)$*. If*

$$x^* = \arg\max_{x\in X} g(x, y_0) = \arg\max_{x\in X} \sum_{m=1}^{M} \left(2y_m^0\sqrt{N_m(x)} - (y_m^0)^2 D_m(x)\right),$$

then $F(x^*) \geq F(x_0)$.

Proof Let x_0, y_0, and x^* be as above and set

$$y^*_m = \frac{\sqrt{N_m(x^*)}}{D_m(x^*)}$$

and $y^* = (y^*_m)_{m=1}^M$. Then we have the following string of inequalities

$$\begin{aligned} F(x_0) &= g(x_0, y_0) \\ &\leq g(x^*, y_0) \\ &\leq g(x^*, y^*) \\ &= F(x^*) \end{aligned}$$

where the third line follows from Lemma 15.1, and the last by a direct computation. The proof is complete. □

The above Theorem guarantees that replacing x_0 by x^* and y_0 by y^* improves the value of the objective function with each iteration, and thus this method (with the scheme written explicitly in Algorithm 15.2) converges. It is clear that the algorithm converges to a local maximum, say (x^*, y^*), of the Quadratic Transform g. It follows easily that the value x^* is indeed a local maximum of the objective F as well.

```
  input : Functions $N_m$ and $D_m$, and a compact, convex set $X$.
1 Find an initial $x_0 \in X$;
2 repeat
3     set $y^0_m = \sqrt{N_m(x_0)}/D_m(x_0)$;
4     solve $x^* = \arg\max_{x \in X} \sum_{m=1}^M \left(2y^0_m\sqrt{N_m(x)} - (y^0_m)^2 D_m(x)\right)$;
5     update $x_0 = x^*$;
6 until convergence;
```

Algorithm 15.2: Shen-Yu iterative algorithm

Corollary 15.1 *Algorithm 15.2 converges to a stationary point of the sum-of-ratios fractional programming problem* (15.1).

Although Algorithm 15.2 is guaranteed to converge to a stationary point, its usefulness is limited to the difficulty of maximizing the Quadratic Transform over the variable x. In the case when each N_m is concave and each D_m is convex, this can be done by any method of concave programming. When the objective has the form (15.3), the Quadratic Transform becomes

$$g(x, y) = \sum_{m=1}^{M} 2y_m|\langle q_m, x\rangle| - y_m^2 x^T P x \tag{15.5}$$

which is not, in general, concave in x. However, in the event that the absolute values around each linear term in the sum can be dropped (i.e. each $\langle q_m, x\rangle$ is either nonnegative or nonpositive valued on X), then we may apply Algorithm 15.2 effectively.

15.3 Region Checking Algorithm

Henceforth we will assume the objective function F is as in (15.2) and, therefore, it can be rewritten in the sum-of-ratios form (15.3). The first observation to make is that combining Algorithm 15.2 with the Dinkelbach method yields an algorithm which converges to the global maximum of (15.3). Indeed, if x^* is the local optimum found by Algorithm 15.2, we may set $\lambda^* = F(x^*)$ and compute both $\pi(\lambda^*)$ and $x(\lambda^*)$. If $|\pi(\lambda^*)| > \varepsilon$ where ε is some tolerance, we again run Algorithm 15.2 with $x(\lambda^*)$ as the initial feasible point and repeat. This method is guaranteed to converge to the globally optimal solution since

$$x(\lambda^*)^T(Q - \lambda^* P)x(\lambda^*) > x^{*T}(Q - \lambda^* P)x^*$$

implies that

$$F(x(\lambda^*)) > F(x^*) \geq F(x_0)$$

where x_0 is the initial feasible point used in Algorithm 15.2.

Finding the value of $\pi(\lambda^*)$ and the vector $x(\lambda^*)$ can be achieved by the methods introduced in [7] and refined in [14], but maximizing the Quadratic Transform as in Algorithm 15.2 cannot. To circumvent this issue, we divide the feasible region into at most $2^{rank(Q)}$ subregions and perform this algorithm independently in each subregion. Suppose, as before, that

$$Q = \sum_{m=1}^{M} q_m q_m^T,$$

and for each $1 \leq m \leq M$ define

$$R_m^0 = \{x \in X \mid \langle q_m, x\rangle \leq 0\}, \quad R_m^1 = \{x \in X \mid \langle q_m, x\rangle \geq 0\}.$$

For each binary sequence $(n_m)_{m=1}^M = n \in \{0, 1\}^M$, we denote by R^n a (possibly empty) subregion of X given by

$$R^n = \bigcap_{m=1}^{M} R_m^{n_m}.$$

A straightforward observation then yields the following result.

Lemma 15.2 *The Quadratic Transform* (15.5) *of* (15.3) *is concave in the x variable over each nonempty subregion R^n for $n \in \{0, 1\}^M$.*

Proof Let $g(x, y)$ be as in (15.5) and fix a binary sequence $n \in \{0, 1\}^M$. Then, by definition of R^n, we see that for each $1 \le m \le M$ we have either $\langle q_m, x\rangle \ge 0$ or $\langle q_m, x\rangle \le 0$ for all $x \in R^n$. This implies that $|\langle q_m, x\rangle|$ is linear over R^n, and hence $g(x, y)$ is a sum of concave functions, which is itself concave. □

We can now formulate our method for solving the convex-convex quadratic fractional programming problem as Algorithm 15.3.

```
input : Matrices $Q = \sum_{m=1}^{M} q_m q_m^T$, $P$, linear constraints $Ax \le b$, and a tolerance $\varepsilon$.
output: Global Solution of (15.2)
1  for $n \in \{0, 1\}^M$ do
2      if $R^n \neq \emptyset$ then
3          Find an initial $x_n \in R^n$;
4          repeat
5              Find stationary point $x^*$ of (15.2) via Algorithm 15.2 with initial point $x_n$;
6              Set $\lambda = F(x^*)$;
7              Compute $\pi(\lambda)$ and $x(\lambda)$;
8              Update $x_n = x(\lambda)$;
9          until $|\pi(\lambda)| < \varepsilon$;
10     end
11 end
12 Determine max $F(x_n)$;
```

Algorithm 15.3: Region-checking algorithm for convex-convex quadratic fractional programming

Remark 15.1 The utility of Algorithm 15.3 is, in full generality, limited by the rank of Q or, more precisely, by the number of nonempty subregions (which is controlled by the rank of Q). It is also worth noting that different decompositions of Q may yield different numbers of nonempty subregions of the feasible region. We leave the question of how to find better decompositions of Q beyond the scope of this paper. We do, however, mention explicitly the case when Q is a totally nonnegative matrix and the optimization problem in question is

$$\begin{aligned} \max \quad & F(x) = \frac{x^T Q x}{x^T P x} \\ \text{s.t.} \quad & Ax = b \\ & 0 \le x \le \alpha. \end{aligned}$$

In this case, we write $Q = LDL^T$ for D a diagonal matrix and L a lower triangular matrix. This gives a decomposition of Q as

$$Q = \sum_{m=1}^{M} d_m \ell_m \ell_m^T,$$

where ℓ_m and d_m are, respectively, the mth column of L and mth diagonal entry of D. Since Q is totally nonnegative, the entries of each vector ℓ_m are nonnegative [6], in which case $\langle \ell_m, x\rangle \geq 0$ for all feasible x. Thus, there is only one subregion R^n of the feasible region which is nonempty.

It is quite possible that Algorithm 15.3 is considerably slower than the algorithms proposed in [7] and [14] given the potentially large number of times $\pi(\lambda)$ is computed. However, as we will show empirically in the next section, often the first local maximum found by Algorithm 15.3 in a given region is, in fact, the global maximum of the region. Thus, we will also compare the efficiency and accuracy of Algorithm 15.3 without computing $\pi(\lambda)$ and assuming each x^* found by Algorithm 15.2 is a global maximum of the region. This modification of Algorithm 15.3 is Algorithm 15.4 below.

```
input  : Matrices $Q = \sum_{m=1}^{M} q_m q_m^T$, $P$, linear constraints $Ax \leq b$, and a tolerance $\varepsilon$.
output: Local Solution of (15.2)
1 for $n \in \{0,1\}^M$ do
2     if $R^n \neq \emptyset$ then
3         Find an initial $x_n \in R^n$;
4         Find stationary point $x^*$ of (15.2) via Algorithm 15.2 with initial point $x_n$;
5         Update $x_n = x^*$;
6     end
7 end
8 Determine max $F(x_n)$;
```

Algorithm 15.4: Fast region-checking algorithm

Remark 15.2 Suppose $Q = qq^T$ is a rank-one matrix. We write the objective function as

$$F(x) = \frac{\langle q, x\rangle^2}{x^T P x}.$$

In this case, there are only two subregions of the feasible region: R^0 and R^1. Also, we may equivalently maximize the square-root of the objective, which is given by

$$\sqrt{F(x)} = \begin{cases} -\langle q, x\rangle / \sqrt{x^T P x} & x \in R^0 \\ \langle q, x\rangle / \sqrt{x^T P x} & x \in R^1. \end{cases}$$

Thus, $\sqrt{F(x)}$ is a concave-convex fractional programming problem in each subregion of the feasible region, and hence can be solved by two applications of Algorithm 15.1 where computing $\pi(\lambda)$ is a concave programming problem. This method is superior to Algorithm 15.4 as it was shown in [13] that Algorithm 15.2 is slower than the standard Dinkelbach method for standard single-ratio concave-convex fractional programs.

15.4 Numerical Experiments

In this section we conduct several numerical experiments on the following optimization problem:

$$\begin{aligned} \max \quad & F(x) = \frac{x^T Qx}{x^T Px} \\ \text{s.t.} \quad & Ax \le b, \quad A \in \mathbb{R}^{T\times n}, \quad b \in \mathbb{R}^T \\ & \sum_{i=1}^{n} x_i = 1 \\ & 0 \le x_i \le 0.1, \quad i = 1, ..., n \end{aligned}$$

where A and b have random entries in the interval $[0, 1]$ so that the vector $(1/n, ..., 1/n)$ is feasible, and $Q = X^T X$ and $P = Y^T Y$ where X and Y are, respectively, an $M \times n$ and an $n \times n$ random matrix with entries in $[0, 1]$. Unless otherwise specified, we will always decompose Q as a sum of rank one matrices according to its eigendecomposition as noted in Sect. 15.2.

We first demonstrate the efficiency and accuracy of Algorithms 15.3 and 15.4 against Algorithm 15.1 for various combinations of (n, M, T). Next, we examine the average number of nonempty subregions in Algorithms 15.3 and 15.4. Finally, we demonstrate the accuracy of Algorithm 15.4 in a full-rank example (i.e. $r = n$, which has several local maxima) when there are a small number of subregions (see Remark 15.1).

All computation was done in MATLAB (on AMD A6-7400K Radeon R5 4.09 GHz processor), using Gurobi 9.5 interface to solve the nonconvex quadratic programming problems involved in Algorithms 15.1 and 15.3. The tolerance ε is always set as $\varepsilon = 10^{-3}$. All values in the forthcoming tables are averages of five tests.

15.4.1 Algorithm Comparison

We first give a demonstration of how differing combinations of (n, M, T) affect the computation time of Algorithm 15.3 in a single nonempty region of the feasible set.

Table 15.1 shows that the computation time for Algorithm 15.3 increases sharply with the number of inequality constraints due to the increasing complexity of solving the nonconvex quadratic subproblems. Likewise, the computation time increases with both the number of variables and the rank of Q, albeit not as sharply. Second, none of the problems solved in this experiment took more than two iterations of Algorithm 15.2 to converge. In fact, 209 of the 240 problems solved in this experiment converged in just one iteration of Algorithm 15.2. This suggests that the expensive procedure of computing $\pi(\lambda)$ can safely be dropped from the algorithm, if a low probability of missing the exact solution may be tolerated by the application.

In the next two experiments, we fix $T = 10$. We now compare Algorithm 15.3 with Algorithm 15.1. We show the results for Algorithm 15.3 both converging to the global solution and forcing only one iteration per region without computing $\pi(\lambda)$.

Table 15.2 shows the comparison of Algorithms 15.1, 15.3, and 15.4. There are several things to note about these results. First, the error incurred from performing only one iteration per region is negligible; it is always under one percent and quite often is under 0.1%. This implies that one need only perform Algorithm 15.2 once in each subregion of the feasible region, making irrelevant the need to compute $\pi(\lambda)$. Second, for a given value of M, the computation time needed to complete Algorithm 15.2 increases steadily as n increases, but does so at a much slower rate for Algorithm 15.4.

In fact, Table 15.3 shows that Algorithm 15.4 can be used efficiently for large scale problems when the number of subregions is less than 130. For comparison, the

Table 15.1 Efficiency of Algorithm 15.3 for one region

		CPU time (sec)				Iterations			
		$T = 1$	$T = 10$	$T = 30$	$T = 50$	$T = 1$	$T = 10$	$T = 30$	$T = 50$
$n = 25$	$M = 2$	0.4657	0.3689	0.7372	0.7063	1.2	1	1	1
	5	1.074	0.5517	1.183	0.7725	1	1.2	1	1
	7	1.037	2.339	1.473	1.337	1	1	1	1
	10	0.5639	1.253	9.06	1.849	1.2	1	1.4	1.2
50	2	2.022	2.512	2.403	5.308	1.2	1.2	1	1.2
	5	3.208	6.842	15.71	19.66	1	1	1.2	1
	7	60.85	5.055	6.284	31.49	1.4	1	1.2	1
	10	8.032	126.5	663.4	49.52	1.2	1	1.4	1
75	2	4.261	7.612	11.04	7.077	1	1	1.2	1
	5	5.299	32.95	58.67	79.35	1	1.2	1.4	1
	7	125.9	62.65	48.03	273.3	1.4	1.2	1.4	1
	10	23.8	20.38	93.34	904.7	1	1	1	1.4

Table 15.2 Comparison of algorithms in CPU time (sec) Error = $(F(x_{opt}) - F(x))/F(x_{opt})$

		CPU time (sec)			Alg 4 error
		Alg 1	Alg 3	Alg 4	Error (%)
$n = 10$	$M = 2$	0.01372	0.07733	0.09616	0
	3	0.006822	0.01969	0.1347	0
	4	0.005291	0.02188	0.2637	0
	5	0.01414	0.02462	0.5047	0
	7	0.01159	–	2.006	0
	10	0.01003	–	16.67	0
25	2	0.5673	0.9934	0.1193	0
	3	0.6599	1.266	0.1932	0.08
	4	1.068	3.975	0.3618	0
	5	1.105	26.85	0.7185	0.06
	7	0.7177	–	2.976	0.02
	10	0.8667	–	23.35	0.02
50	2	4.932	4.649	0.1331	0.04
	3	5.258	11.6	0.2104	0.05
	4	9.485	72.51	0.4216	0.02
	5	7.618	223.9	0.91	0.01
	7	5.573	–	3.18	0.03
	10	8.004	–	26.5	0
75	2	12.63	17.12	0.1229	0.01
	3	14.43	273.3	0.346	0
	4	30.47	211.2	0.531	0.2
	5	45.63	868.8	1.249	0.07
	7	34.33	–	4.407	0
	10	53.35	–	35.55	0.05

Table 15.3 CPU time (sec) for Algorithm 15.4

			Algorithm 15.4
$n = 250$	$M = 7$	$T = 10$	22.32
500	7	10	117.34
750	7	10	357.42
1000	7	10	743.24

authors in [14] state that the case when $n = 500$ is within reach via their algorithm only by employing an elaborate local search.

15.4.2 *Accuracy of Algorithm 4*

The numerical experiments above call into question the accuracy of Algorithm 15.4 in high-rank problems with a small number of nonempty regions. Table 15.1 shows

Table 15.4 Average number of subregions checked by Algorithm 15.4

		Number of subregions				
		$T = n/2$	$T = n$	$T = 3n/2$	$T = 2n$	$T = 5n/2$
$n = 30$	$M = 2$	2	2	2	1	1
	3	4	4	3.8	2	1
	5	16	16	12	3	1
	7	64	64	49.4	11	1
50	2	2	2	2	1.8	1
	3	4	4	4	2.8	1
	5	16	16	14.4	5	1
	7	64	64	64	19.4	1
100	2	2	2	2	1.4	1
	3	4	4	4	2.8	1
	5	16	16	16	6.6	1
	7	64	64	64	23.4	1
150	2	2	2	2	1.2	1
	3	4	4	4	2.8	1
	5	16	16	16	8	1
	7	64	64	64	13.6	1

that there is typically only one local maximum per subregion, and this may seem to be attributable to the large number of subregions relative to the number of local maxima. Thus, one might conclude that it may be disadvantageous to choose a decomposition of Q with fewer terms as this leads to fewer possible subregions. We aim to show that this is not the case.

The number of subregions is, of course, determined by the linear constraints. Table 15.4 shows that the number of subregions is usually $2^{rank(Q)-1}$, and this decreases only when the number of inequality constraints is much larger than the number of variables. This implies that the number of subregions checked by the algorithm increases exponentially with the rank of Q, and thus the accuracy of Algorithm 15.4 could be a product of the brute-force nature of checking each region.

For illustration we construct a full-rank example with a small number of subregions to check. For a given value of n, we generate n random, linearly independent vectors $(q_i)_{i=1}^n$ with values in the unit interval [0, 1]. We construct the matrix Q as

$$Q = \sum_{i=1}^{n} q_i q_i^T .$$

Note that since the q_i are constructed to be linearly independent, the matrix Q will be invertible. Using this decomposition of Q and the constraints as before, there will be only one non-empty subregion of the feasible region. This is because the entries

Table 15.5 CPU time (sec) and error in full rank example

		CPU time (sec)		Alg 4 error
		Alg 1	Alg 4	Error (%)
$n = 20$	$T = 1$	0.2189	0.01942	0
	10	0.6094	0.02111	0
	30	0.9902	0.02257	0.02936
	50	1.766	0.04667	0
35	1	1.423	0.02659	0.08328
	10	6.945	0.02921	0.01075
	30	7.726	0.03527	0.05105
	50	20.61	0.05936	0.02039
50	1	6.25	0.03352	0
	10	6.753	0.03664	0.02419
	30	25.31	0.04325	0.01203
	50	189	0.09199	0.01428

of each q_i are positive, and hence $\langle q_i, x\rangle \geq 0$ for all feasible x. All other matrices are constructed in the same manner as before.

It is shown in Table 15.5 that the accuracy of Algorithm 15.4 remains quite high using this decomposition of Q. So, the accuracy of Algorithm 15.4 should not be attributed primarily to the number of subregions. Therefore, since choosing a decomposition of Q which results in few nonempty subregions yields a faster algorithm, it is advantageous to choose one which yields the fewest number of nonempty subregions of the feasible region.

15.5 Conclusion

We have presented an efficient and accurate algorithm for globally maximizing low-rank convex-convex quadratic fractional programming problems. We have also demonstrated that this algorithm can be utilized in high-rank problems if the numerator admits a decomposition which divides the feasible region into a small number of subregions. Although Algorithm 15.4 is only guaranteed to converge to a local maximum of (15.2), we have shown heuristically that it almost always converges to the global solution.

To guarantee global convergence of Algorithm 15.4, one needs only to have a method to determine if there is a feasible $x \in X \cap R$ such that $F(x) > F(x^*)$ where x^* is the local maximum found in the subregion R by Algorithm 15.2. One way this may be done is by using a solver to maximize the nonconvex quadratic

$$G(x) = x^T(Q - F(x^*)P)x$$

with the added quadratic constraint that $G(x) > 0$, and artificially terminating the solver once a feasible x is found. If no such x can be found, then x^* is the global solution in the subregion.

Finally, we remark that Algorithm 15.4 is also applicable to the quadratic sum-of-ratios problems, i.e. in the case when the matrix P in the denominator of (15.3) is allowed to vary with m (in fact, the denominators need not be quadratic, they just need to be convex).

Acknowledgments Both authors of the paper were supported in part by the NSF grant DMS-2208031. The authors are grateful to the anonymous referees for their suggestions that helped improve the exposition. The paper is dedicated to the everlasting memory of Guido L. Weiss whose research, teaching, and friendship has inspired generations.

References

1. H.P. Benson, Global optimization algorithm for the nonlinear sum of ratios problem. J. Optim. Theory Appl. **112**(1), 1–29 (2002)
2. H.P. Benson, Using concave envelopes to globally solve the nonlinear sum of ratios problem. J. Global Optim. **22**(1/4), 343–364 (2002)
3. A. Charnes, W.W. Cooper, Programming with linear fractional functionals. Naval Res. Logist. Q. **10**(1), 273–274 (1962)
4. W. Dinkelbach, On nonlinear fractional programming. Manag. Sci. **13**(7), 492–498 (1967)
5. L. Gao, S.K. Mishra, J. Shi, An extension of branch-and-bound algorithm for solving sum-of-nonlinear-ratios problem. Optim. Lett. **6**(2), 221–230 (2010)
6. K.R. Goodearl, T.H. Lenagan, Lu decomposition of totally nonnegative matrices. Linear Algebra Appl. **436**(7), 2554–2566 (2012)
7. J.-Y. Gotoh, H. Konno, Maximization of the ratio of two convex quadratic functions over a polytope. Comput. Optim. Appl. **20**(1), 43–60 (2001)
8. L. Huang, D. Needell, S. Tang, Robust recovery of bandlimited graph signals via randomized dynamical sampling. arXiv:2109.14079 (2021)
9. T. Ibaraki, Parametric approaches to fractional programs. Math. Program. **26**(3), 345–362 (1983)
10. H. Jiao, S. Liu, An efficient algorithm for quadratic sum-of-ratios fractional programs problem. Numer. Funct. Anal. Optim. **38**(11), 1426–1445 (2017)
11. A. Lo, A.C. MacKinlay. Maximizing predictability in the stock and bond markets. Macroecon. Dynamics **1**, 102 (1997)
12. S.-J. Qu, K.-C. Zhang, J.-K. Zhao, An efficient algorithm for globally minimizing sum of quadratic ratios problem with nonconvex quadratic constraints. Appl. Math. Comput. **189**(2), 1624–1636 (2007)
13. K. Shen, W. Yu, Fractional programming for communication systems–part I: Power control and beamforming. IEEE Trans. Signal Process. **66**(10), 2616–2630 (2018)
14. R. Yamamoto, H. Konno, An efficient algorithm for solving convex–convex quadratic fractional programs. J. Optim. Theory Appl. **133**(2), 241–255 (2007)