

## On Computing Schur Functions and Series Thereof

Cy Chan · Vesselin Drensky · Alan Edelman ·  
Raymond Kan · Plamen Koev

Received: date / Accepted: date

**Abstract** In this paper we present two new algorithms for computing all Schur functions  $s_{\kappa}(x_1, \dots, x_n)$  for partitions  $\kappa$  such that  $|\kappa| \leq N$ .

For nonnegative arguments,  $x_1, \dots, x_n$ , both algorithms are subtraction-free and thus each Schur function is computed to high relative accuracy in floating point arithmetic. The cost of each algorithm per Schur function is  $\mathcal{O}(n^2)$ .

**Keywords** Schur function · Hypergeometric function of a matrix argument · Computing · Accuracy

---

This research was partially supported by NSF Grants DMS-0314286, DMS-0411962, DMS-0608306, DMS-1016086, the Singapore-MIT Alliance (SMA), Grant MI-1503/2005 of the Bulgarian National Science Fund, and by RSCA Program and the Woodward Fund for Applied Mathematics at San Jose State University. The Woodward Fund is a gift from the estate of Mrs. Marie Woodward in memory of her son, Henry Teynham Woodward. He was an alumnus of the Mathematics Department at San Jose State University and worked with research groups at NASA Ames.

C. Chan

Computational Research Division, Lawrence Berkeley National Laboratory One Cyclotron Rd., Berkeley, CA 94720, U.S.A.

E-mail: cychan@lbl.gov

V. Drensky

Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria.

E-mail: drensky@math.bas.bg

A. Edelman

Department of Mathematics, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, U.S.A.

E-mail: edelman@math.mit.edu

R. Kan

Joseph L. Rotman School of Management, University of Toronto, 105 St. George Street, Toronto, Ontario, Canada M5S 3E6.

E-mail: kan@chass.utoronto.ca

P. Koev

Department of Mathematics, San Jose State University, One Washington Square, San Jose, CA 95192, U.S.A.

E-mail: koev@math.sjsu.edu

## 1 Introduction

We consider the problem of accurately and efficiently evaluating the Schur function  $s_{\kappa}(x_1, x_2, \dots, x_n)$  and series thereof for nonnegative arguments  $x_i \geq 0, i = 1, 2, \dots, n$ . The Schur functions are indexed by *partitions*  $\kappa$ . We say that  $\kappa = (\kappa_1, \kappa_2, \dots)$ ,  $\kappa_1 \geq \kappa_2 \geq \dots \geq 0$ , is a partition of an integer  $k$ , denoted  $\kappa \vdash k$ , if  $\kappa_1 + \kappa_2 + \dots = k$ . Only finitely many of the  $\kappa_i$  can be nonzero.

Our ultimate goal is to efficiently compute the the *hypergeometric function of an  $n \times n$  semidefinite matrix argument  $X$*  and parameter  $\alpha > 0$ :

$${}_pF_q^{(\alpha)}(a_1, \dots, a_p; b_1, \dots, b_q; X) = \sum_{k=0}^{\infty} \sum_{\kappa \vdash k} \frac{1}{k!} \cdot \frac{(a_1)_{\kappa}^{(\alpha)} \cdots (a_p)_{\kappa}^{(\alpha)}}{(b_1)_{\kappa}^{(\alpha)} \cdots (b_q)_{\kappa}^{(\alpha)}} \cdot C_{\kappa}^{(\alpha)}(X), \quad (1)$$

where

$$(c)_{\kappa}^{(\alpha)} \equiv \prod_{(i,j) \in \kappa} (c - (i-1)/\alpha + j - 1) \quad (2)$$

is the *generalized Pochhammer symbol*, and  $C_{\kappa}^{(\alpha)}(X)$  is the *Jack function*. The latter is a generalization of the Schur function and is normalized so that  $\sum_{\kappa \vdash k} C_{\kappa}^{(\alpha)}(X) = (\text{tr}(X))^k$  [24, 31, 39]. The argument  $X$  in (1) is a matrix for historical reasons only;  $C_{\kappa}^{(\alpha)}$  and  ${}_pF_q^{(\alpha)}$  are scalar-valued symmetric functions in the eigenvalues  $x_i \geq 0, i = 1, 2, \dots, n$ , of  $X$ .

The practical importance of computing the hypergeometric function of a matrix argument stems from far reaching applications in multitude of fields. For example, in multivariate statistics it provides a closed-form expression for the distributions of the eigenvalues of the Wishart, Jacobi, Laguerre, and MANOVA random matrix ensembles [9, 34, 10, 23]. These distributions, in turn, are needed in critical statistical tests in applications ranging from genomics [37] to wireless communications [14, 25, 38], finance [20], target classification [22], etc.

Despite its enormous practical importance, only limited progress has been made in the computation of this function since the 1960s. The problems with its computation come from two sources: (1) it converges extremely slowly, and (2) the straight-forward evaluation of a single Jack function is exponential [8]. The frustrations of many researchers with the lack of efficient algorithms are long standing and well documented [3, 17, 19, 21, 32, 34, 36].

The recent progress in computing the hypergeometric function of a matrix argument has focused on exploiting the combinatorial properties of the Jack function leading to new algorithms which are exponentially faster than the previous best ones (see Section 2 for an overview).

Interestingly, the computational potential of the combinatorial properties of the Jack function had been missed for quite some time. It is thus our hope to draw the attention of the combinatorics community to this problem and its far reaching practical applications.

Although the hypergeometric function of a matrix argument is defined for any  $\alpha > 0$  and there are corresponding theoretical interpretations [11], most applications focus on  $\alpha = 1$  and  $\alpha = 2$  only. This is largely because these values correspond to the distributions of the eigenvalues of complex and real random matrices, respectively.

In this paper we focus on  $\alpha = 1$ . In this case the Jack function  $C_{\kappa}^{(1)}(X)$  is a normalization of the Schur function  $s_{\kappa}(x_1, \dots, x_n)$  (see (3) in Section 2 for the exact relationship).

One way to compute the hypergeometric function of a matrix argument in practice is to truncate the series (1) for  $k \leq N$  for some sufficiently large  $N$ .

Since  $s_{\kappa}(x_1, x_2, \dots, x_n) = 0$  if  $\kappa_{n+1} > 0$ , our goal is thus to compute, as quickly and accurately as possible, all Schur functions corresponding to partitions  $\kappa$  in not more than  $n$  parts and size not exceeding  $N$ .

Denote the set of those Schur functions by  $\mathcal{S}_{N,n}$ :

$$\mathcal{S}_{N,n} \equiv \{s_{\kappa}(x_1, \dots, x_n) \mid \kappa = (\kappa_1, \dots, \kappa_n), |\kappa| \equiv \kappa_1 + \dots + \kappa_n \leq N\}.$$

Computing even a single Schur function accurately and efficiently is far from trivial. We elaborate on this briefly.

There are several determinantal expressions for the Schur function (the classical definition as quotient of alternants [30, (0.1)], the Jacobi–Trudi identity [30, (0.2)], its dual version—the Nägelsbach–Kostka formula [30, (0.3)], [43], the Giambelli [31, p. 47] and Lascoux–Pragacz [31, (2), p. 87] determinants). Each one would seemingly provide a very efficient way to compute the Schur function. The problem with this approach is that the matrices involved quickly become ill conditioned as the sizes of the matrix argument ( $n$ ) and the partition ( $|\kappa|$ ) grow. This implies that conventional (Gaussian-elimination-based) algorithms will quickly lose accuracy to roundoff errors. The loss of accuracy is due to a phenomenon known as *subtractive cancellation*—loss of significant digits due to subtraction of intermediate (and thus approximate) quantities of similar magnitude.

The analysis in [8] shows that the loss of accuracy in evaluating the determinantal expressions for the Schur function can be arbitrarily large in all but the dual Jacobi–Trudi identity. In the latter the amount of subtractive cancellation can be bounded independent of the values of the input arguments  $x_i$ . By using extended precision one can compensate for that loss of accuracy leading to an algorithm that is guaranteed to be accurate and costs  $\mathcal{O}((n|\kappa| + \kappa_1^3)(|\kappa|\kappa_1|^{1+\rho}))$ .<sup>1</sup>

Subtraction is the only arithmetic operation that could lead to loss of accuracy; multiplication, division, and addition of same-sign quantities always preserves the relative accuracy.

In this paper we present two new algorithms for computing the Schur function. Both algorithms are subtraction-free, meaning that both are guaranteed to compute the value of the Schur function to high relative accuracy in floating point arithmetic. Both are also very efficient—the cost per Schur function, when computing all Schur functions in the set  $\mathcal{S}_{N,n}$ , is  $\mathcal{O}(n^2)$ .

This represents a major improvement over the previous best result in [8] in the sense that no extended precision arithmetic is required to achieve accuracy and the

<sup>1</sup> Here  $\rho$  is tiny and accounts for certain logarithmic functions.

cost of computing a single Schur function is reduced from  $\mathcal{O}((n|\kappa| + \kappa_1^3)(|\kappa|\kappa_1)^{1+\rho})$  to  $\mathcal{O}(n^2)$ .

While both our new algorithms have the same complexity and accuracy characteristics, each is significant in its own right for the following reasons:

- The first algorithm implements the classical definition of the Schur function as a sum of monomials over all semistandard Young tableaux. Since the coefficients in this expression are positive (integers), such an approach is subtraction-free, thus guaranteed to be accurate. The full expression of the Schur function as a sum of monomials contains exponentially many terms ( $\mathcal{O}(n^{|\kappa|})$  [8]) thus the similarly exponential cost of the previous algorithms based on it [8]. Since the practical evaluation of the hypergeometric function requires summation over partitions of size  $N$  much larger than the number of variables  $n$ , the computational savings in reducing the cost of an individual Schur function is considerable.

We use dynamic programming and exploit various redundancies to reduce the cost to  $\mathcal{O}(n^2)$  per Schur function so long as all Schur functions in the set  $\mathcal{S}_{N,n}$  are computed. This algorithm is not efficient for computing *individual* Schur functions (since it requires the computation of most functions in  $\mathcal{S}_{N,n}$ ). However, this is not an issue when computing the hypergeometric function of a matrix argument since all functions in  $\mathcal{S}_{N,n}$  need to be computed there anyway. These ideas generalize the approach of Neuman in [35, (5.1)] and may generalize further beyond  $\alpha = 1$ ; we elaborate on this in Section 6.

In contrast, the second algorithm does allow the efficient computation of individual Schur functions, but is based on identities with no known generalizations beyond  $\alpha = 1$ .

- The second algorithm represents an accurate evaluation of the expression of the Schur function as a quotient of (generalized, totally nonnegative) Vandermonde determinants. Since virtually all linear algebra with totally nonnegative matrices can be performed efficiently and in a subtraction-free fashion [27, 28], this leads to an accurate algorithm for the evaluation of individual Schur functions at the cost of only  $\mathcal{O}(n^2 \kappa_1)$  each. The cost reduces further to  $\mathcal{O}(n^2)$  each if all of  $\mathcal{S}_{N,n}$  is computed.

This paper is organized as follows. We present background information and survey existing algorithms for this problem in Section 2. Our new algorithms are presented in Sections 3 and 4. We draw conclusions and outline open problems in Section 6.

We made software implementations of both our new algorithms available online [26].

## 2 Preliminaries

Algorithms for computing the hypergeometric function of a matrix argument for specific values of  $p$ ,  $q$ , and  $\alpha$  can be found in [2, 4, 5, 18, 19].

In this section we survey the approach of Koev and Edelman [29] which works for any  $\alpha > 0$ . We also introduce a few improvements and set the stage for our new algorithms in the case  $\alpha = 1$ .

We first recall a few definitions that are relevant. For a partition  $\kappa$ , its *conjugate* partition  $\kappa' = (\kappa'_1, \kappa'_2, \dots)$ , is defined as  $\kappa'_i = \#\{\kappa_j | \kappa_j \geq i, j = 1, 2, \dots\}$ . For positive integers  $i$  and  $j$  we say that  $(i, j) \in \kappa$  if  $i \leq \kappa'_j$  and  $j \leq \kappa_i$ . The *upper and lower hook lengths* at  $(i, j) \in \kappa$  are defined, respectively, as:

$$\begin{aligned} h_{\kappa}^*(i, j) &\equiv \kappa'_j - i + \alpha(\kappa_i - j + 1); \\ h_{\kappa}^{\kappa}(i, j) &\equiv \kappa'_j - i + 1 + \alpha(\kappa_i - j). \end{aligned}$$

The products of the upper and lower hook lengths are denoted, respectively, as:

$$H_{\kappa}^* \equiv \prod_{(i,j) \in \kappa} h_{\kappa}^*(i, j) \quad \text{and} \quad H_{\kappa}^{\kappa} \equiv \prod_{(i,j) \in \kappa} h_{\kappa}^{\kappa}(i, j).$$

We introduce the ‘‘Schur’’ normalization of the Jack function

$$S_{\kappa}^{(\alpha)}(X) = \frac{H_{\kappa}^{\kappa}}{\alpha^{|\kappa|} |\kappa|!} C_{\kappa}^{(\alpha)}(X). \quad (3)$$

This normalization is such that  $S_{\kappa}^{(1)}(X) = s_{\kappa}(x_1, \dots, x_n)$  [39, Proposition 1.2].

The hypergeometric function of a matrix argument in terms of  $S_{\kappa}^{(\alpha)}(X)$  is:

$${}_pF_q^{(\alpha)}(a_1, \dots, a_p; b_1, \dots, b_q; X) = \sum_{k=0}^{\infty} \sum_{\kappa \vdash k} \frac{(a_1)_{\kappa}^{(\alpha)} \cdots (a_p)_{\kappa}^{(\alpha)}}{(b_1)_{\kappa}^{(\alpha)} \cdots (b_q)_{\kappa}^{(\alpha)}} \cdot \frac{\alpha^k}{H_{\kappa}^{\kappa}} \cdot S_{\kappa}^{(\alpha)}(X). \quad (4)$$

Denote the coefficient in front of  $S_{\kappa}^{(\alpha)}(X)$  in (4) by:

$$Q_{\kappa} \equiv \frac{(a_1)_{\kappa}^{(\alpha)} \cdots (a_p)_{\kappa}^{(\alpha)}}{(b_1)_{\kappa}^{(\alpha)} \cdots (b_q)_{\kappa}^{(\alpha)}} \cdot \frac{\alpha^{|\kappa|}}{H_{\kappa}^{\kappa}}.$$

Let the partition  $\kappa = (\kappa_1, \kappa_2, \dots, \kappa_h)$  have  $h = \kappa'_1$  nonzero parts. When  $\kappa_i > \kappa_{i+1}$ , we define the partition:

$$\kappa_{(i)} \equiv (\kappa_1, \kappa_2, \dots, \kappa_{i-1}, \kappa_i - 1, \kappa_{i+1}, \dots, \kappa_h). \quad (5)$$

The main idea in the evaluation of (4) is to *update* the  $\kappa$  term in (4) from terms earlier in the series. In particular, we update  $Q_{\kappa}$  from  $Q_{\kappa_{(h)}}$  and  $S_{\kappa}^{(\alpha)}(x_1, \dots, x_n)$  from  $S_{\mu}^{(\alpha)}(x_1, \dots, x_{n-1})$ ,  $\mu \leq \kappa$ .

In order to make the  $Q_{\kappa}$  update as simple as possible, we first express  $H_{\kappa}^{\kappa}$  in a way that does not involve the conjugate partition,  $\kappa'$ :

$$\begin{aligned} H_{\kappa}^{\kappa} &= \prod_{r=1}^h \prod_{c=1}^{\kappa_r} (\kappa'_c - r + 1 + \alpha(\kappa_r - c)) \\ &= \prod_{r=1}^h \prod_{j=1}^r \prod_{c=\kappa_{j+1}+1}^{\kappa_j} (j - r + 1 + \alpha(\kappa_r - c)) \\ &= \alpha^{|\kappa|} \prod_{j=1}^h \prod_{r=1}^j \left( \frac{j - r + 1}{\alpha} + \kappa_r - \kappa_j \right)_{\kappa_j - \kappa_{j+1}}, \end{aligned}$$

where  $(c)_t = c(c+1)\cdots(c+t-1)$  is the *rising factorial*, the univariate version of the Pochhammer symbol defined in (2).

Defining  $\tilde{\kappa}_i \equiv \alpha\kappa_i - i$  we obtain:

$$\frac{H_*^{\kappa(h)}}{H_*^\kappa} = \frac{1}{\alpha\kappa_h - \alpha + 1} \prod_{j=1}^{h-1} \frac{\tilde{\kappa}_j - \tilde{\kappa}_h}{\tilde{\kappa}_j - \tilde{\kappa}_h + 1}. \quad (6)$$

Using (6),  $Q_\kappa$  can be updated from  $Q_{\kappa(h)}$  as

$$Q_\kappa = Q_{\kappa(h)} \cdot \frac{\prod_{j=1}^p (a_j + \tilde{\kappa}_h)}{\prod_{j=1}^q (b_j + \tilde{\kappa}_h)} \cdot \frac{\alpha}{\alpha\kappa_h - \alpha + 1} \cdot \prod_{j=1}^{h-1} \frac{\tilde{\kappa}_j - \tilde{\kappa}_h}{\tilde{\kappa}_j - \tilde{\kappa}_h + 1}, \quad (7)$$

where  $\tilde{\kappa}_h \equiv \kappa_h - 1 - \frac{h-1}{\alpha}$ .

The Jack function  $S_\kappa^{(\alpha)}(X)$  can be dynamically updated using the formula of Stanley [39, Proposition 4.2] (see also [29, (3.8)] and (3)):

$$S_\kappa^{(\alpha)}(x_1, \dots, x_n) = \sum_{\mu} S_\mu^{(\alpha)}(x_1, \dots, x_{n-1}) x_n^{|\kappa/\mu|} \sigma_{\kappa\mu}, \quad (8)$$

where the summation is over all partitions  $\mu \leq \kappa$  such that the skew shape  $\kappa/\mu$  is a *horizontal strip* (i.e.,  $\kappa_1 \geq \mu_1 \geq \kappa_2 \geq \mu_2 \geq \dots$  [40, p. 339]). The coefficients  $\sigma_{\kappa\mu}$  are defined as

$$\sigma_{\kappa\mu} = \prod_{(i,j) \in \kappa} \frac{h_\kappa^*(i,j)}{h_\kappa^*(i,j)} \prod_{(i,j) \in \mu} \frac{h_\mu^*(i,j)}{h_\mu^*(i,j)}, \quad (9)$$

where both products are over all  $(i,j) \in \kappa$  such that  $\kappa'_j = \mu'_j + 1$ . For  $\alpha = 1$ , clearly,  $\sigma_{\kappa\mu} = 1$  for all  $\kappa$  and  $\mu$ .

Once again, instead of computing the coefficients  $\sigma_{\kappa\mu}$  in (8) from scratch, it is much more efficient to start with  $\sigma_{\kappa\kappa} = 1$  and update the next coefficient in the sum (8) from the previous ones. To this end, let  $\mu$  be a partition such that  $\kappa'_j = \mu'_j$  for  $j = 1, 2, \dots, \mu_k - 1$ , and  $\kappa/\mu$  be a horizontal strip. Then we update  $\sigma_{\kappa\mu}$  from  $\sigma_{\kappa\mu(k)}$  using:

$$\frac{\sigma_{\kappa\mu(k)}}{\sigma_{\kappa\mu}} = \prod_{r=1}^k \frac{h_\kappa^*(r, \mu_k)}{h_\kappa^*(r, \mu_k)} \prod_{r=1}^{k-1} \frac{h_\mu^*(r, \mu_k)}{h_\mu^*(r, \mu_k)} = \prod_{r=1}^k \frac{1 + \tilde{\kappa}_r - \tilde{\mu}_k}{\alpha + \tilde{\kappa}_r - \tilde{\mu}_k} \prod_{r=1}^{k-1} \frac{\tilde{\mu}_r - \tilde{\mu}_k + \alpha - 1}{\tilde{\mu}_r - \tilde{\mu}_k}, \quad (10)$$

which is obtained directly from (9).

We use (8) to compute  $S_\kappa^{(\alpha)}(x_1, \dots, x_i)$  for  $i = h+1, \dots, n$ . For  $i = h$ , the result of Stanley [39, Propositions 5.1 and 5.5] allows for a very efficient update:

$$S_\kappa^{(\alpha)}(x_1, \dots, x_h) = (x_1 \cdots x_h)^{\kappa_h} \cdot S_{\kappa - \kappa_h I}^{(\alpha)}(x_1, \dots, x_h) \cdot \prod_{j=1}^{\kappa_h} \prod_{i=1}^h \frac{h-i+1 + \alpha(\kappa_i - j)}{h-i + \alpha(\kappa_i - j + 1)}, \quad (11)$$

where  $\kappa - \kappa_h I \equiv (\kappa_1 - \kappa_h, \kappa_2 - \kappa_h, \dots, \kappa_{h-1} - \kappa_h)$ .

The new results in this section comprise of the updates (7) and (10), which are more efficient than the analogous ones in [29, Lemmas 3.1 and 3.2]. These new updates do not require the conjugate partition to be computed and maintained by the

algorithm and cost  $2(p+q) + 4h$  and  $9k$ , down from  $2(p+q) + 11\kappa_h + 9h - 11$  and  $12k + 6\mu_k - 7$ , respectively.

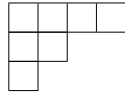
Additionally, the use of (11) reduces the cost of an evaluation of a truncation of (1) by a factor of about  $N/2$ .

### 3 The first algorithm

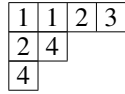
In this section we present the first of our two new algorithms for computing all Schur functions of the set  $\mathcal{S}_{N,n}$ .

We first recall the definition of a Young diagram and a semistandard tableau.

A Young diagram of a partition  $\kappa \vdash k$  is a finite collection of  $k$  boxes arranged in left-justified rows, with  $\kappa_i$  boxes in row  $i$ ,  $i = 1, 2, \dots$ . For example, the Young diagram below corresponds to the partition  $\kappa = (4, 2, 1) \vdash 7$ :



When the boxes of the Young diagram are filled with the numbers  $1, 2, \dots, n$  such that the numbers increase nonstrictly along rows and increase strictly along columns we obtain a *semistandard tableau*. We say that the *content* of the tableau is  $(c_1, c_2, \dots, c_n)$  if there are  $c_1$  ones,  $c_2$  twos, etc. For example, the following semistandard tableau of shape  $(4, 2, 1)$  has content  $(2, 2, 1, 2)$ :



To every semistandard tableau of content  $T = (c_1, c_2, \dots, c_n)$  we associate the term  $X^T = x_1^{c_1} x_2^{c_2} \dots x_n^{c_n}$ .

Our algorithm is based on the classical definition of the Schur function [40, Section 7.10]:

$$s_{\kappa}(x_1, \dots, x_k) = \sum_{T \in A_{\kappa}} X^T,$$

where the summation is over the set  $A_{\kappa}$  of all semistandard  $\kappa$ -tableaux  $T$  filled with the numbers  $1, 2, \dots, k$ .

Extending the notation, let

$$s_{\kappa}^m(x_1, \dots, x_k) = \sum_{T \in A_{\kappa, m}} X^T,$$

where the summation is over the set  $A_{\kappa, m}$ , which equals  $A_{\kappa}$  with the additional restriction that  $k$  does not appear in the first  $m$  rows.

Note that  $s_{\kappa}(x_1, \dots, x_k) = s_{\kappa}^0(x_1, \dots, x_k)$  and  $s_{\kappa}(x_1, \dots, x_{k-1}) = s_{\kappa}^k(x_1, \dots, x_k)$ .

**Lemma 1** *The following identity holds for all  $s_{\kappa}^{m-1}(x_1, \dots, x_k)$ :*

$$s_{\kappa}^{m-1}(x_1, \dots, x_k) = \begin{cases} s_{\kappa}^m(x_1, \dots, x_k), & \text{if } \kappa_m = \kappa_{m+1}; \\ s_{\kappa(m)}^{m-1}(x_1, \dots, x_k) \cdot x_k + s_{\kappa}^m(x_1, \dots, x_k), & \text{otherwise,} \end{cases}$$

where the partition  $\kappa(m)$  is defined as in (5).

*Proof* In the first case ( $\kappa_m = \kappa_{m+1}$ ), no  $k$  is allowed in the  $m$ th row of  $T$  because of the strictly increasing property of each column of  $T$ . Therefore, the restriction that no  $k$  appear in the first  $m-1$  rows of  $T$  is equivalent to the restriction that no  $k$  appear in the first  $m$  rows of  $T$ , and  $A_{\kappa, m-1} = A_{\kappa, m}$ .

In the second case, there are two possibilities for the  $\kappa$ -tableau  $T \in A_{\kappa, m-1}$ . If the entry in position  $(m, \kappa_m)$  is not equal to  $k$ , then none of the entries in the  $m$ th row can equal  $k$  due to the nondecreasing nature of each row. Thus, the tableaux fitting this description are exactly the set  $A_{\kappa, m}$ .

If the entry in position  $(m, \kappa_m)$  is equal to  $k$ , then removal of that square of the tableau clearly results in an element of  $A_{\kappa(m), m-1}$ . Further, for every tableau in  $A_{\kappa(m), m-1}$ , the addition of a square containing  $k$  to the  $m$ th row results in a valid semi-standard tableau in  $A_{\kappa, m-1}$ . The tableau retains its semistandardness because every element in the  $m$ th row (and in the entire table as well) can be no larger than  $k$ , and every element in the  $\kappa_m$ th column above the new square can be no larger than  $k-1$  due to the restriction that every tableau in  $A_{\kappa(m), m-1}$  cannot have  $k$  in the first  $m-1$  columns.

We have thus constructed a bijection  $f$  mapping  $A_{\kappa(m), m-1}$  to the set (call it  $B$ ) of tableaux in  $A_{\kappa, m-1}$  where the entry in position  $(m, \kappa_m)$  equals  $k$ . Clearly, for each  $T \in A_{\kappa(m), m-1}$ ,  $X^{f(T)} = X^T \cdot x_k$ , so  $\sum_{T \in B} X^T = \sum_{T \in A_{\kappa(m), m-1}} X^T \cdot x_k$ .

Combining these two possibilities for  $T \in A_{\kappa, m-1}$ , we obtain

$$\begin{aligned} s_{\kappa}^{m-1}(x_1, \dots, x_k) &= \sum_{T \in A_{\kappa, m-1}} X^T \\ &= \sum_{T \in A_{\kappa, m}} X^T + \sum_{T \in A_{\kappa(m), m-1}} X^T \cdot x_k \\ &= s_{\kappa}^m(x_1, \dots, x_k) + s_{\kappa(m)}^{m-1}(x_1, \dots, x_k) \cdot x_k, \end{aligned}$$

concluding our proof.  $\square$

Our algorithm, based on Lemma 1, is very simple.

**Algorithm 1** *The following algorithm computes all Schur functions in  $\mathcal{S}_{N, n}$ .*

for all  $\kappa \in \mathcal{S}_{N, n}$  initialize  $s_{\kappa} = x_1^{\kappa_1}$  if  $\kappa \in \mathcal{S}_{N, 1}$  and  $s_{\kappa} = 0$  otherwise  
for  $k = 2$  to  $n$  (Loop 1)

  for  $m = k$  down to 1 (Loop 2)

    for all  $\kappa \in \mathcal{S}_{N, k}$  such that  $\kappa_m > \kappa_{m+1}$ , in reverse lexicographic  
    order

$$s_{\kappa} = s_{\kappa} + s_{\kappa(m)} \cdot x_k$$

  endfor

endfor

endfor



After the first line Algorithm 1, the variables  $s_\kappa$  contain  $s_\kappa(x_1)$ . During each iteration of Loop 1, the values stored in  $s_\kappa$  for  $\kappa \in \mathcal{S}_{N,k}$  are updated from  $s_\kappa(x_1, \dots, x_{k-1}) = s_\kappa^k(x_1, \dots, x_k)$  to  $s_\kappa(x_1, \dots, x_k) = s_\kappa^0(x_1, \dots, x_k)$ . During each iteration of Loop 2, the values in  $s_\kappa$  for  $\kappa \in \mathcal{S}_{N,k}$  are updated from  $s_\kappa^m(x_1, \dots, x_k)$  to  $s_\kappa^{m-1}(x_1, \dots, x_k)$ .

The last line of the algorithm implements Lemma 1. Since the partitions are processed in reverse lexicographic order,  $s_{\kappa(m)}$  will have already been updated for each  $\kappa$  when this line is executed. Thus, at the time  $s_\kappa$  is updated,  $s_{\kappa(m)}$  contains  $s_{\kappa(m)}^{m-1}(x_1, \dots, x_k)$ , and  $s_\kappa$  is updated from  $s_\kappa^m(x_1, \dots, x_k)$  to  $s_\kappa^{m-1}(x_1, \dots, x_k)$ . The algorithm updates the Schur functions “in place” using a single memory location for each partition.

One implementation issue is the need to retrieve the value of  $s_{\kappa(m)}$  in the innermost loop of the algorithm. Since  $\kappa$  is a partition in  $n$  parts, figuring the memory location of  $s_{\kappa(m)}$  on the fly would take  $O(n)$  time, an unfortunate situation since  $s_{\kappa(m)}$  only participates in two floating point operations. Our solution to this problem was to precompute a lookup table, which given the index of a partition  $\kappa$  and  $m$  contains the index of  $s_{\kappa(m)}$ .

#### 4 The second algorithm

Our second algorithm is based on the expression of the Schur function as a quotient of totally nonnegative generalized Vandermonde determinants:

$$s_\kappa(x_1, \dots, x_n) = \frac{\det G}{\det V_{n,n}}, \quad (12)$$

where

$$G \equiv (x_i^{j-1+\kappa_n-j+1})_{i,j=1}^n \quad \text{and} \quad V_{n,n} \equiv (x_i^{j-1})_{i,j=1}^n$$

are  $n \times n$  generalized and ordinary Vandermonde matrices, respectively.

Since  $s_\kappa(x_1, x_2, \dots, x_n)$  is a symmetric polynomial, we can assume that the  $x_i$ 's are sorted in increasing order:  $0 \leq x_1 \leq x_2 \leq \dots \leq x_n$ . This choice of ordering makes  $G$  and  $V_{n,n}$  totally nonnegative [13, p. 76] thus the methods of [28, Section 6] can be used to evaluate (12) with guaranteed accuracy in  $\mathcal{O}(n^2 \kappa_1)$  time. The matrices  $G$  and  $V_{n,n}$  are notoriously ill conditioned [15] meaning that conventional Gaussian-elimination-based algorithms will quickly lose all accuracy to roundoff [8].

The contribution of this section is to show how to eliminate the removable singularity at  $x_i = x_j$ ,  $i \neq j$ , and to arrange the computations in such a way that the cost per Schur function is only  $\mathcal{O}(n^2)$  when evaluating all of  $\mathcal{S}_{N,n}$ .

To this end we will represent the Schur function as a determinant of certain complete symmetric polynomials (see also [1, 6, 33, 43] for other examples where the general idea of using other bases for the representation of symmetric functions has proven useful).

Since the generalized Vandermonde matrix  $G$  is a submatrix of the rectangular Vandermonde matrix

$$V_{n,m} \equiv (x_i^{j-1})_{i,j=1}^{n,m},$$

$m = n + \kappa_1$ , consisting of columns  $1 + \kappa_n, 2 + \kappa_{n-1}, \dots, n + \kappa_1$ , we consider the LDU decomposition  $V_{n, n+\kappa_1} = LDU$ , where  $L$  is a unit lower triangular  $n \times n$  matrix,  $D$  is a diagonal  $n \times n$  matrix, and  $U$  is a unit upper triangular  $n \times (n + \kappa_1)$  matrix.

The critical observation here is that the value of  $\det G$  is unaffected by  $L$ , namely

$$\det G = \det D \cdot \det \bar{U},$$

where  $\bar{U}$  is the  $(n \times n)$  submatrix of  $U$  consisting of columns  $1 + \kappa_n, 2 + \kappa_{n-1}, \dots, n + \kappa_1$ .

However,  $\det D = \det V_{n, n}$ , thus

$$s_{\kappa}(x_1, x_2, \dots, x_n) = \det \bar{U}. \quad (13)$$

The explicit form of  $U$  is known [12, Section 2.2], [41, eq. (2.3)], [42, Section 3] allowing us to write (13) also as:

$$s_{\kappa}(x_1, \dots, x_n) = \det (h_{i-j+\kappa_{n-j+1}}(x_1, \dots, x_i))_{i,j=1}^n,$$

where  $h_k, k = 1, 2, \dots$ , are the complete symmetric polynomials and, by default,  $h_k \equiv 0$  for  $k < 0$ .

The *bidagonal decomposition* of  $U$ , has a particularly easy form:

$$\mathcal{BD}(U) = \left\{ \begin{array}{cccccccc} 1 & x_1 & x_1 & \dots & x_1 & x_1 & x_1 & \dots \\ 0 & 1 & x_2 & \dots & x_2 & x_2 & x_2 & \dots \\ 0 & 0 & 1 & \dots & x_3 & x_3 & x_3 & \dots \\ & & & \ddots & & & & \\ 0 & 0 & 0 & & 1 & x_n & x_n & \dots \end{array} \right\}. \quad (14)$$

For example, for  $n = 3, m = 4$  [27, Section 3]:

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & x_1 & & \\ & 1 & x_2 & \\ & & 1 & x_3 \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & x_1 & & \\ & 1 & x_2 & \\ & & 1 & 0 \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & x_1 & & \\ & 1 & 0 & \\ & & 1 & 0 \\ & & & 1 \end{bmatrix}.$$

In order to compute the Schur function, it is therefore sufficient to compute the determinant of the matrix obtained by removing the appropriate  $\kappa_1$  columns of  $U$ . This can be readily done using existing techniques—Algorithm 5.6 from [28] starts with the bidiagonal decomposition of a totally nonnegative matrix and produces the bidiagonal decomposition of the totally nonnegative matrix obtained by removing any given column. This is done in a subtraction-free fashion and thus to high relative accuracy.

Therefore from  $\mathcal{BD}(U)$  we obtain  $\mathcal{BD}(\bar{U})$  in  $\mathcal{O}(n^2 \kappa_1)$  time. This is the cost per *individual* Schur function.

In order to compute the hypergeometric function of a matrix argument, we compute the Schur functions in reverse lexicographic order. For a given Schur function  $s_{\kappa}$ , let  $i$  be the smallest index such that  $\kappa_i > \kappa_{i+1}$ , i.e.,  $\kappa = (\kappa_i, \kappa_i, \dots, \kappa_i, \kappa_{i+1}, \dots)$ .

The generalized Vandermonde matrix  $G_\kappa$  corresponding to  $s_\kappa$  has exponents

$$(\kappa_n, 1 + \kappa_{n-1}, \dots, n - i - 1 + \kappa_{i+1}, n - i + \kappa_i, \dots, n - 1 + \kappa_1, \dots).$$

Consider now the partition  $\lambda = (\kappa_i - 1, \kappa_i - 1, \dots, \kappa_i - 1, \kappa_{i+1}, \dots)$ . It comes earlier in the reverse lexicographic order than  $\kappa$ , thus  $s_\lambda$  is already computed. The generalized Vandermonde matrix  $G_\lambda$  that corresponds to  $s_\lambda$  has exponents

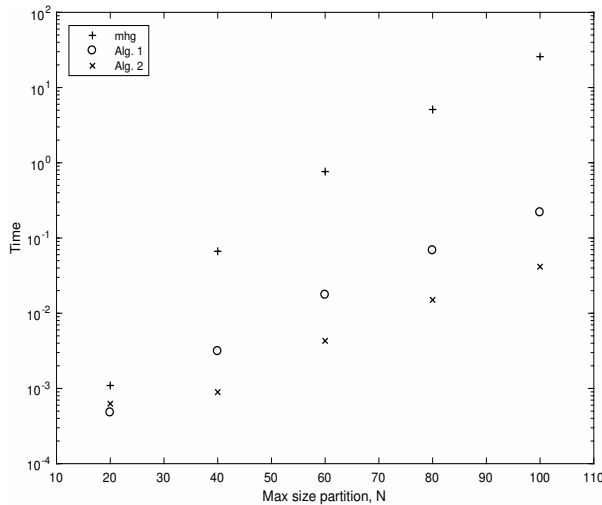
$$(\kappa_n, 1 + \kappa_{n-1}, \dots, n - i - 1 + \kappa_{i+1}, n - i - 1 + \kappa_i, \dots, n - 2 + \kappa_i, \dots).$$

Therefore  $G_\kappa$  can be obtained from  $G_\lambda$  by removing its  $(i + 1)$ st column. The same is true for the corresponding bidiagonal decompositions and can be achieved in  $O(n^2)$  time using Algorithm 5.6 from [28].

The cost per Schur function when computing the hypergeometric function of a matrix argument is again  $O(n^2)$ .

## 5 Numerical experiments

We performed extensive numerical experiments to confirm the correctness and performance of our algorithms against the algorithm `mhg` from [29]. We present the results of one typical experiment on the logarithmic plot in Figure 1. We timed all three algorithms for different values of  $N$  on a matrix argument of size  $n = 5$ . For  $N \ll n$  the number of partitions of  $N$  in not more than  $n$  parts grows as  $\mathcal{O}(N^n)$ . Both our new algorithms performed, roughly, as  $N^5$ , which was expected. They are also way faster than the algorithm `mhg` from [29]. The practical implementation of Algorithm 1 of this paper is also clearly faster than that of Algorithm 2.



**Fig. 1** Logarithmic plot of the performance of Algorithms 1 and 2 vs the algorithm of [29] on a  $5 \times 5$  matrix argument.

## 6 Open problems

It is natural to ask if the ideas of this paper can extend beyond  $\alpha = 1$  and in particular to  $\alpha = 2$ , the other value of  $\alpha$  of major practical importance [34].

None of the determinantal expressions for the Schur function are believed to have analogues for  $\alpha \neq 1$ , thus we are skeptical of the potential of the ideas in Section 4 to generalize.

The results of Section 3, however, may extend beyond  $\alpha = 1$ .

Consider the (column) vector  $s^{(n)}$  consisting of all Schur functions in  $\mathcal{S}_{N,n}$  ordered in reverse lexicographic order. Let  $s^{(n-1)}$  be the same set, but on  $n-1$  variables  $x_1, \dots, x_{n-1}$ . Then

$$s^{(n)} = Ms^{(n-1)}$$

where  $M$  is an  $|\mathcal{S}_{N,n}| \times |\mathcal{S}_{N,n-1}|$  matrix whose entries are indexed by partitions and  $M_{\mu\nu} = x^{|\mu|-|\nu|}$  if  $\mu/\nu$  is a horizontal strip and 0 otherwise. The contribution of Section 4 was to recognize that  $M$  consists of blocks of the form

$$A = \begin{bmatrix} 1 & & & \\ x & 1 & & \\ x^2 & x & 1 & \\ x^3 & x^2 & x & 1 \end{bmatrix}.$$

Since  $A^{-1}$  is bidiagonal:

$$A^{-1} = \begin{bmatrix} 1 & & & \\ -x & 1 & & \\ & -x & 1 & \\ & & -x & 1 \end{bmatrix},$$

given a vector (call it  $z$ ), the matrix-vector product  $y = Az$  can be formed in linear (instead of quadratic) time by solving instead the *bidiagonal* linear system  $A^{-1}y = z$  for  $y$ .

This was our original approach in designing Algorithm 1. Ultimately we found the much more elegant proof which we presented instead.

The question remains whether this approach can be generalized to other values of  $\alpha$ . Unfortunately the matrix  $A$  in general has the form:

$$A^{(\alpha)} = \begin{bmatrix} 1 & & & & \\ x & 1 & & & \\ \frac{1}{2!}x^2(\alpha+1) & x & 1 & & \\ \frac{1}{3!}x^3(1+\alpha)(1+2\alpha) & \frac{1}{2!}x^2(\alpha+1) & x & 1 & \\ & & & & 1 \end{bmatrix},$$

where the general expression for the entries  $A^{(\alpha)}$  is  $a_{ij}^{(\alpha)} = \frac{x^{i-j}}{(i-j)!} \prod_{k=0}^{i-j-1} (k\alpha+1)$ ,  $i > j$ .

The matrix  $(A^{(\alpha)})^{-1}$  is not bidiagonal for  $\alpha \neq 1$  thus the approach of Section 3 cannot be carried over directly. One could consider exploiting the Toeplitz structure of  $A^{(\alpha)}$  to form a matrix-vector product with it in  $O(k \log k)$  instead of  $k^2$  time (assuming  $A^{(\alpha)}$  is  $k \times k$ ) [16, p. 193]. Current computing technology, however, limits  $N$  to about 200 and since  $k \leq N$ , this approach does not appear feasible in practice at this time.

**Acknowledgements** We thank James Demmel for posing the problem of computing the Schur function accurately and efficiently [7, Section 9.1(2)]; Iain Johnstone and Richard Stanley for many fruitful discussions during the development of this material, and the anonymous referee for pointing out the relevance of several references.

## References

1. Biedenharn, L.C., Louck, J.D.: A new class of symmetric polynomials defined in terms of tableaux. *Adv. in Appl. Math.* **10**(4), 396–438 (1989)
2. Butler, R.W., Wood, A.T.A.: Laplace approximations for hypergeometric functions with matrix argument. *Ann. Statist.* **30**(4), 1155–1177 (2002)
3. Byers, G., Takawira, F.: Spatially and temporally correlated MIMO channels: modeling and capacity analysis. *IEEE Transactions on Vehicular Technology* **53**, 634–643 (2004)
4. Chen, W.R.: Table for upper percentage points of the largest root of a determinantal equation with five roots
5. Chen, W.R.: Some new tables of the largest root of a matrix in multivariate analysis: A computer approach from 2 to 6 (2002). Presented at the 2002 American Statistical Association
6. Chen, W.Y.C., Louck, J.D.: The factorial Schur function. *J. Math. Phys.* **34**(9), 4144–4160 (1993)
7. Demmel, J., Gu, M., Eisenstat, S., Slapničar, I., Veselić, K., Drmač, Z.: Computing the singular value decomposition with high relative accuracy. *Linear Algebra Appl.* **299**(1–3), 21–80 (1999)
8. Demmel, J., Koev, P.: Accurate and efficient evaluation of Schur and Jack functions. *Math. Comp.* **75**(253), 223–239 (2006)
9. Dumitriu, I., Edelman, A.: Matrix models for beta-ensembles. *J. Math. Phys.* **43**, 5830–5847 (2002)
10. Edelman, A., Koev, P.: Eigenvalue distributions of Beta–Wishart matrices. *Random Matrices: Theory and Applications* **03**, 1450,009 (2014)
11. Forrester, P.J.: Log-gases and random matrices, *London Mathematical Society Monographs Series*, vol. 34. Princeton University Press, Princeton, NJ (2010). DOI 10.1515/9781400835416. URL <http://dx.doi.org/10.1515/9781400835416>
12. Gander, W.: Change of basis in polynomial interpolation. *Numer. Linear Algebra Appl.* **12**, 769–778 (2005)
13. Gantmacher, F., Krein, M.: *Oscillation Matrices and Kernels and Small Vibrations of Mechanical Systems*, revised edn. AMS Chelsea, Providence, RI (2002)
14. Gao, H., Smith, P., Clark, M.: Theoretical reliability of MMSE linear diversity combining in Rayleigh-fading additive interference channels. *IEEE Transactions on Communications* **46**, 666–672 (1998). DOI 10.1109/26.668742
15. Gautschi, W., Inglese, G.: Lower bounds for the condition number of Vandermonde matrices. *Numer. Math.* **52**(3), 241–250 (1988)
16. Golub, G., Van Loan, C.: *Matrix Computations*, 3rd edn. Johns Hopkins University Press, Baltimore, MD (1996)
17. Grant, A.: Performance analysis of transmit beamforming. *IEEE Transactions on Communications* **53**, 738–744 (2005)
18. Gross, K.I., Richards, D.S.P.: Total positivity, spherical series, and hypergeometric functions of matrix argument. *J. Approx. Theory* **59**(2), 224–246 (1989)
19. Gutiérrez, R., Rodríguez, J., Sáez, A.J.: Approximation of hypergeometric functions with matricial argument through their development in series of zonal polynomials. *Electron. Trans. Numer. Anal.* **11**, 121–130 (2000)
20. Harding, M.: Explaining the single factor bias of arbitrage pricing models in finite samples. *Economics Letters* **9**, 85–88 (2008)
21. James, A.T.: Distributions of matrix variates and latent roots derived from normal samples. *Ann. Math. Statist.* **35**, 475–501 (1964)
22. Jeffris, M.: Robust 3D ATR techniques based on geometric invariants. In: F.A. Sadjadi (ed.) *Automatic Target Recognition XV, Proceedings of SPIE*, vol. 5807, pp. 190–200. SPIE, Bellingham, WA (2005)
23. Kan, R., Koev, P.: Densities of the extreme eigenvalues of Beta–MANOVA matrices. Preprint (2017)
24. Kaneko, J.: Selberg integrals and hypergeometric functions associated with Jack polynomials. *SIAM J. Math. Anal.* **24**(4), 1086–1110 (1993)
25. Kang, M., Alouini, M.S.: Largest eigenvalue of complex Wishart matrices and performance analysis of MIMO MRC systems. *IEEE Journal on Selected Areas in Communications* **21**(3), 418–431 (2003)

- 
26. Koev, P.: <http://www.math.sjsu.edu/~koev>
  27. Koev, P.: Accurate eigenvalues and SVDs of totally nonnegative matrices. *SIAM J. Matrix Anal. Appl.* **27**(1), 1–23 (2005)
  28. Koev, P.: Accurate computations with totally nonnegative matrices. *SIAM J. Matrix Anal. Appl.* **29**, 731–751 (2007)
  29. Koev, P., Edelman, A.: The efficient evaluation of the hypergeometric function of a matrix argument. *Math. Comp.* **75**(254), 833–846 (2006)
  30. Macdonald, I.G.: Schur functions: theme and variations. In: Séminaire Lotharingien de Combinatoire (Saint-Nabor, 1992), *Publ. Inst. Rech. Math. Av.*, vol. 498, pp. 5–39. Univ. Louis Pasteur, Strasbourg (1992)
  31. Macdonald, I.G.: Symmetric functions and Hall polynomials, Second edn. Oxford University Press, New York (1995)
  32. McKay, M., Collings, I.: Capacity bounds for correlated rician MIMO channels. In: 2005 IEEE International Conference on Communications. ICC 2005., vol. 2, pp. 772–776 (2005)
  33. Méndez, M.A.: The umbral calculus of symmetric functions. *Adv. Math.* **124**(2), 207–271 (1996)
  34. Muirhead, R.J.: Aspects of multivariate statistical theory. John Wiley & Sons Inc., New York (1982)
  35. Neuman, E.: On complete symmetric functions. *SIAM J. Math. Anal.* **19**(3), 736–750 (1988)
  36. Ozyildirim, A., Tanik, Y.: SIR statistics in antenna arrays in the presence of correlated Rayleigh fading. In: IEEE VTS 50th Vehicular Technology Conference, 1999. VTC 1999 - Fall, vol. 1, pp. 67–71 (1999). DOI 10.1109/VETECE.1999.797042
  37. Patterson, N., Price, A., Reich, D.: Population structure and eigenanalysis. *PLoS Genetics* **2**, 2074–2093 (2006)
  38. Smidl, V., Quinn, A.: Fast variational PCA for functional analysis of dynamic image sequences. In: Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis, 2003. ISPA 2003., vol. 1, pp. 555–560 (2003). DOI 10.1109/ISPA.2003.1296958
  39. Stanley, R.P.: Some combinatorial properties of Jack symmetric functions. *Adv. Math.* **77**(1), 76–115 (1989)
  40. Stanley, R.P.: Enumerative combinatorics. Vol. 2, *Cambridge Studies in Advanced Mathematics*, vol. 62. Cambridge University Press, Cambridge (1999)
  41. Van de Vel, H.: Numerical treatment of a generalized Vandermonde system of equations. *Linear Algebra Appl.* **17**, 149–179 (1977)
  42. Verde-Star, L.: Biorthogonal polynomial bases and Vandermonde-like matrices. *Stud. Appl. Math.* **95**(3), 269–295 (1995)
  43. Verde-Star, L.: Representation of symmetric functions as Gram determinants. *Adv. Math.* **140**(1), 128–143 (1998)