Title
On computing Schur functions and series thereof

Authors
Chan, Cy
Drensky, Vesselin
Edelman, Alan
et al.

Publication Date
2019-09-15

DOI
10.1007/s10801-018-0846-y

Peer reviewed

J Algebr Comb manuscript No.
(will be inserted by the editor)

On Computing Schur Functions and Series Thereof

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

Received: date / Accepted: date

Abstract We present two new algorithms for computing all Schur functions $s_\kappa(x_1,\ldots,x_n)$ for partitions κ such that $|\kappa| \leq N$.

Both algorithms have the property that for nonnegative arguments $x_1,\ldots,x_n$ the output is computed to high relative accuracy and the cost per Schur function is $\mathcal{O}(n^2)$.

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

Mathematics Subject Classification (2000) 05E05 · 65F50 · 65T50

This research was partially supported by NSF Grants DMS–0314286, DMS–0411962, DMS–0608306, the Singapore–MIT Alliance (SMA), and by Grant MI-1503/2005 of the Bulgarian National Science Fund.

C. Chan
Department of Computer Science, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, U.S.A.
E-mail: cychan@mit.edu

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 $s_\kappa(x_1, x_2, \ldots, x_n)$ and series thereof for nonnegative arguments $x_i \geq 0$, $i = 1, 2, \ldots, n$. The Schur functions are symmetric homogeneous polynomials

Our goal is to derive efficient algorithms for computing the *hypergeometric function of an $n \times n$ semidefinite matrix argument X* and parameter $\alpha > 0$:

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

where the summation is over all *partitions* $\kappa = (\kappa_1, \kappa_2, \ldots)$ of $k = 0, 1, 2, \ldots$;

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

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 $\sum_{\kappa \vdash k} C_\kappa^{(\alpha)}(X) = (\mathrm{tr}(X))^k$ [19, 25, 31]. 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 $x_i \geq 0$, $i = 1, 2, \ldots, n$, of X.

The practical importance of computing the hypergeometric function of a matrix argument stems from far reaching applications of multivariate statistics, where it provides a closed-form expression for the distributions of the eigenvalues of the classical random matrix ensembles—Wishart, Jacobi, and Laguerre [27]. These distributions, in turn, are needed in critical statistical tests in applications ranging from genomics [29] to wireless communications [10, 20, 30], finance [16], target classification [18], 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 straightforward evaluation of a single Jack function is exponential [6]. The frustrations of many researchers with the lack of efficient algorithms are long standing and well documented [2, 13, 15, 17, 26–28].

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

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

$$ \mathscr{S}_{N,n} \equiv \{ s_\kappa(x_1, \ldots, x_n) \,|\, \kappa = (\kappa_1, \ldots, \kappa_n), |\kappa| \equiv \kappa_1 + \cdots + \kappa_n \leq N \}. $$

The analysis in [6] suggests that 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 generalized Vandermonde determinants, the Jacobi–Trudi identity, its dual version, the Giambelli and Lascoux–Pragacz 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.

According to [6], 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 $\mathcal{O}((n|\kappa| + \kappa_1^3)(|\kappa|\kappa_1|^{1+\rho}))$.[1]

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 [6] in the sense that no extended precision arithmetic is required to achieve accuracy and the 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 ($\mathcal{O}(n^{|\kappa|})$ [6]) thus the similarly exponential cost of the previous algorithms based on it [6].
  We use dynamic programming and exploit various redundancies to reduce the cost to $\mathcal{O}(n^2)$ per Schur function as 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 most functions in $\mathcal{S}_{N,n}$ would need to be computed anyway. However, (unlike the second algorithm) the ideas may generalize beyond $\alpha = 1$; we elaborate on this in Section 5.
- 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 [22,23], this leads to an accurate algorithm for the evaluation of individual Schur functions extremely efficiently at the cost of only $O(n^2\kappa_1)$ each. The cost reduces further to $\mathcal{O}(n^2)$ each if all of $\mathcal{S}_{N,n}$ is computed.

[1] Here ρ is tiny and accounts for certain logarithmic functions.

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 5.

We made software implementations of both our new algorithms available online [21].

2 Preliminaries

Algorithms for computing the hypergeometric function of a matrix argument for specific values of p, q, and α can be found in [1,3,4,14,15].

In this section we survey the approach of Koev and Edelman [24] 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. Given a partition κ and a point (i, j) in the Young diagram of κ (i.e., $i \le \kappa'_j$ and $j \le \kappa_i$), the upper and lower *hook lengths* at $(i, j) \in \kappa$ are defined, respectively, as:

$$h^*_\kappa(i,j) \equiv \kappa'_j - i + \alpha(\kappa_i - j + 1);$$
$$h^\kappa_*(i,j) \equiv \kappa'_j - i + 1 + \alpha(\kappa_i - j).$$

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

$$H^*_\kappa \equiv \prod_{(i,j)\in\kappa} h^*_\kappa(i,j) \qquad \text{and} \qquad H^\kappa_* \equiv \prod_{(i,j)\in\kappa} h^\kappa_*(i,j).$$

We introduce the "Schur" normalization of the Jack function

$$S^{(\alpha)}_\kappa(X) = \frac{H^\kappa_*}{\alpha^{|\kappa|}|\kappa|!} C^{(\alpha)}_\kappa(X). \tag{3}$$

This normalization is such that $S^{(1)}_\kappa(X) = s_\kappa(x_1, \ldots, x_n)$ [31, Proposition 1.2].

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

$${}_pF^{(\alpha)}_q(a_1,\ldots,a_p;b_1,\ldots,b_q;X) = \sum_{k=0}^{\infty}\sum_{\kappa\vdash k} \frac{(a_1)^{(\alpha)}_\kappa \cdots (a_p)^{(\alpha)}_\kappa}{(b_1)^{(\alpha)}_\kappa \cdots (b_q)^{(\alpha)}_\kappa} \cdot \frac{\alpha^k}{H^\kappa_*} \cdot S^{(\alpha)}_\kappa(X). \tag{4}$$

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

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

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

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

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

In order to make the Q_κ update as simple as possible, we first express H_*^κ in a way that does not involve the conjugate partition, κ':

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

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:

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

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

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

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

The Jack function $S_\kappa^{(\alpha)}(X)$ can be dynamically updated using the formula of Stanley [31, Proposition 4.2] (see also [24, (3.8)] and (3)):

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

where the summation is over all partitions $\mu \le \kappa$ such that the skew shape κ/μ is a *horizontal strip* (i.e., $\kappa_1 \ge \mu_1 \ge \kappa_2 \ge \mu_2 \ge \cdots$ [32, p. 339]). The coefficients $\sigma_{\kappa\mu}$ are defined as

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

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 κ and μ.

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 μ be a partition such that $\kappa'_j = \mu'_j$ for $j = 1,2,\ldots,\mu_k - 1$, and κ/μ be a horizontal strip. Then we update $\sigma_{\kappa\mu_{(k)}}$ from $\sigma_{\kappa\mu}$ using:

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

which is obtained directly from (9).

We use (8) to compute $S^{(\alpha)}_{\kappa}(x_1,\ldots,x_i)$ for $i=h+1,\ldots,n$. For $i=h$, the result of Stanley [31, Propositions 5.1 and 5.5] allows for a very efficient update:

$$S^{(\alpha)}_{\kappa}(x_1,\ldots,x_h) = (x_1\cdots x_h)^{\kappa_h}\cdot S^{(\alpha)}_{\kappa-\kappa_h I}(x_1,\ldots,x_h)\cdot\prod_{j=1}^{\kappa_h}\prod_{i=1}^{h}\frac{h-i+1+\alpha(\kappa_i-j)}{h-i+\alpha(\kappa_i-j+1)}, \qquad (11)$$

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

The new results in this section comprise of the updates (7) and (10), which are more efficient than the analogous ones in [24, 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 $\mathscr{S}_{N,n}$. This algorithm is based on the classical definition of the Schur function [32, Section 7.10]:

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

where the summation is over the set A_{κ} of all semistandard κ-tableaux T filled with the numbers $1,2,\ldots,k$. Also, $X^T = x_1^{c_1}\cdots x_k^{c_k}$, and $(c_1,\ldots,c_k)$ is the content of T. Extending the notation, let:

$$s^m_{\kappa}(x_1,\ldots,x_k) = \sum_{T\in A_{\kappa,m}} X^T,$$

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

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

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

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

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 mth 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 κ-tableau $T\in A_{\kappa,m-1}$. If the entry in position (m,κ_m) is not equal to k, then none of the entries in the mth 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,κ_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 mth row results in a valid semistandard tableau in $A_{\kappa,m-1}$. The tableau retains its semistandardness because every element in the mth row (and in the

entire table as well) can be no larger than k, and every element in the κ_mth 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,κ_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

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

concluding our proof. □

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

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

```
for all κ ∈ S_{N,n} initialize s_κ = x_1^{κ_1} if κ ∈ S_{N,1} and s_κ = 0 otherwise
for k = 2 to n    (Loop 1)
  for m = k down to 1    (Loop 2)
    for all κ ∈ S_{N,k} such that κ_m > κ_{m+1}, in reverse lexicographic order
      s_κ = s_κ + s_{κ_(m)} · x_k
    endfor
  endfor
endfor
```

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

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 κ when this line is executed. Thus, at the time s_κ is updated, $s_{\kappa_{(m)}}$ contains $s_{\kappa_{(m)}}^{m-1}(x_1,\ldots,x_k)$, and s_κ is updated from $s_\kappa^m(x_1,\ldots,x_k)$ to $s_\kappa^{m-1}(x_1,\ldots,x_k)$. Our algorithm updates the Schur functions "in place" using a single memory location for each partition.

In order to complete the algorithm in time $\mathscr{O}(n^2)$ per Schur function, we must be able to lookup the memory location of $s_{\kappa_{(m)}}$ in constant time for each $\kappa \in \mathscr{S}_{N,n}$ and $1 \le m \le n$. In our implementation, we keep all of the Schur variables $\{s_\kappa\}_{\kappa\in\mathscr{S}_{N,n}}$ in an array and use a lookup table of size $|\mathscr{S}_{N,n}|\cdot n$ to find the appropriate array index for each $s_{\kappa_{(m)}}$. Since this lookup table is invariant over all possible inputs $x_1,\ldots,x_n$, we simply precompute it (or load it from disk) and keep it in persistent memory for future calls to our algorithm.

It is also possible to compute the indexes of each $s_{\kappa_{(m)}}$ on the fly rather than having them stored in a lookup table. This modification reduces the memory requirement from $\mathscr{O}(|\mathscr{S}_{N,n}|\cdot n)$ to $\mathscr{O}(|\mathscr{S}_{N,n}|)$, but increases the time complexity by a factor of n to $\mathscr{O}(n^3)$ per Schur function. In practice, the constant hidden by the big-$\mathscr{O}$ notation (for time complexity) is greatly increased when computing indices on the fly; therefore, since $n \ll N$ we have found the lookup table approach to be much more efficient. Further discussion of implementation issues and MATLAB code for both methods are available online [21].

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:

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

where

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

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

Since $s_\kappa(x_1,x_2,\ldots,x_n)$ is a symmetric polynomial, we can assume that the x_i's are sorted in increasing order: $0 \le x_1 \le x_2 \le \cdots \le x_n$. This choice of ordering makes G and $V_{n,m}$ totally nonnegative [9, p. 76] thus the methods of [23, Section 6] can be used to evaluate (12) with guaranteed accuracy in $\mathcal{O}(n^2\kappa_1)$ time. The matrices G and $V_{n,m}$ are notoriously ill conditioned [11] meaning that conventional Gaussian-elimination-based algorithms will quickly lose all accuracy to roundoff [6].

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 $\mathcal{S}_{N,n}$.

It is convenient to see G as a submatrix of the rectangular Vandermonde matrix

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

$m = n-1+\kappa_1$, consisting of columns $1+\kappa_n, 2+\kappa_{n-1}, \ldots, n-1+\kappa_1$.

Consider the LDU decomposition $V_{n,n-1+\kappa_1} = LDU$, 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 $n \times (n-1+\kappa_1)$ 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 its columns $1+\kappa_n, 2+\kappa_{n-1}, \ldots, n-1+\kappa_1$.

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

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

The explicit form of U is known [8, Section 2.2], [33, eq. (2.3)], allowing us to write (13) also as:

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

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

In order to apply the algorithms of [23] to evaluate (13), we need the *bidiagonal decomposition* of U, which has a particularly easy form:

$$\mathcal{BD}(U) = \begin{Bmatrix} 1 & x_1 & x_1 & \ldots & x_1 & x_1 & x_1 & \ldots \\ 0 & 1 & x_2 & \ldots & x_2 & x_2 & x_2 & \ldots \\ 0 & 0 & 1 & \ldots & x_3 & x_3 & x_3 & \ldots \\ & & & \ddots & & & & \\ 0 & 0 & 0 & & 1 & x_n & x_n & \ldots \end{Bmatrix}. \tag{14}$$

For example, for $n = 3, m = 4$ [22, Section 3]:

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

Therefore computing the Schur function consists of using Algorithm 5.6 from [23] κ_1 times to remove the appropriate κ_1 columns in U and obtain the bidiagonal decomposition of $\bar{U}$. The determinant of $\bar{U}$, i.e., the Schur function, is easily computable by multiplying out the diagonal of the diagonal factor in the bidiagonal decomposition of $\bar{U}$.

The total cost is $\mathscr{O}(n^2\kappa_1)$.

In this process of computing $s_\kappa(x_1,\ldots,x_n)$ a total of κ_1 (intermediate) Schur functions are computed, therefore all functions in the $\mathscr{S}_{N,n}$ can be computed at the cost of $\mathscr{O}(n^2)$ each.

5 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 α of major practical importance [27].

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 $\mathscr{S}_{N,n}$ ordered in reverse lexicographic order. Let $s^{(n-1)}$ be the same set, but on $n-1$ variables $x_1,\ldots,x_{n-1}$. Then

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

where M is an $|\mathscr{S}_{N,n}| \times |\mathscr{S}_{N,n-1}|$ matrix whose entries are indexed by partitions and $M_{\mu\nu} = x^{|\mu|-|\nu|}$ if μ/ν 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 is whether this approach can be generalized to other values of α. Unfortunately the matrix A in general has the form:

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

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 $k \times k$) [12, 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 [5, Section 9.1(2)] as well as Iain Johnstone and Richard Stanley for many fruitful discussions during the development of this material.

References

1. Butler, R.W., Wood, A.T.A.: Laplace approximations for hypergeometric functions with matrix argument. Ann. Statist. **30**(4), 1155–1177 (2002)
2. Byers, G., Takawira, F.: Spatially and temporally correlated MIMO channels: modeling and capacity analysis. IEEE Transactions on Vehicular Technology **53**, 634–643 (2004)
3. Chen, W.R.: Table for upper percentage points of the largest root of a determinantal equation with five roots
4. 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
5. 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)
6. Demmel, J., Koev, P.: Accurate and efficient evaluation of Schur and Jack functions. Math. Comp. **75**(253), 223–239 (2006)
7. Forrester, P.: Log-gasses and Random matrices. `http://www.ms.unimelb.edu.au/~matpjf/matpjf.html`
8. Gander, W.: Change of basis in polynomial interpolation. Numer. Linear Algebra Appl. **12**, 769–778 (2005)
9. Gantmacher, F., Krein, M.: Oscillation Matrices and Kernels and Small Vibrations of Mechanical Systems, revised edn. AMS Chelsea, Providence, RI (2002)
10. 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
11. Gautschi, W., Inglese, G.: Lower bounds for the condition number of Vandermonde matrices. Numer. Math. **52**(3), 241–250 (1988)
12. Golub, G., Van Loan, C.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore, MD (1996)
13. Grant, A.: Performance analysis of transmit beamforming. IEEE Transactions on Communications **53**, 738–744 (2005)
14. 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)
15. Gutiérrez, R., Rodriguez, 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)
16. Harding, M.: Explaining the single factor bias of arbitrage pricing models in finite samples. Economics Letters **9**, 85–88 (2008)

17. James, A.T.: Distributions of matrix variates and latent roots derived from normal samples. Ann. Math. Statist. **35**, 475–501 (1964)
18. 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)
19. Kaneko, J.: Selberg integrals and hypergeometric functions associated with Jack polynomials. SIAM J. Math. Anal. **24**(4), 1086–1110 (1993)
20. 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)
21. Koev, P.: `http://www.math.sjsu.edu/~koev`
22. Koev, P.: Accurate eigenvalues and SVDs of totally nonnegative matrices. SIAM J. Matrix Anal. Appl. **27**(1), 1–23 (2005)
23. Koev, P.: Accurate computations with totally nonnegative matrices. SIAM J. Matrix Anal. Appl. **29**, 731–751 (2007)
24. Koev, P., Edelman, A.: The efficient evaluation of the hypergeometric function of a matrix argument. Math. Comp. **75**(254), 833–846 (2006)
25. Macdonald, I.G.: Symmetric functions and Hall polynomials, Second edn. Oxford University Press, New York (1995)
26. 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)
27. Muirhead, R.J.: Aspects of multivariate statistical theory. John Wiley & Sons Inc., New York (1982)
28. 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/VETECF.1999.797042
29. Patterson, N., Price, A., Reich, D.: Population structure and eigenanalysis. PLoS Genetics **2**, 2074–2093 (2006)
30. 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
31. Stanley, R.P.: Some combinatorial properties of Jack symmetric functions. Adv. Math. **77**(1), 76–115 (1989)
32. Stanley, R.P.: Enumerative combinatorics. Vol. 2, *Cambridge Studies in Advanced Mathematics*, vol. 62. Cambridge University Press, Cambridge (1999)
33. Van de Vel, H.: Numerical treatment of a generalized Vandermonde system of equations. Linear Algebra Appl. **17**, 149–179 (1977)