kStatistics: Unbiased Estimates of Joint Cumulant Products from the Multivariate Faà Di Bruno's Formula

by Elvira Di Nardo and Giuseppe Guarino

Abstract kStatistics is a package in R that serves as a unified framework for estimating univariate and multivariate cumulants as well as products of univariate and multivariate cumulants of a random sample, using unbiased estimators with minimum variance. The main computational machinery of kStatistics is an algorithm for computing multi-index partitions. The same algorithm underlies the general-purpose multivariate Faà di Bruno's formula, which therefore has been included in the last release of the package. This formula gives the coefficients of formal power series compositions as well as the partial derivatives of multivariable function compositions. One of the most significant applications of this formula is the possibility to generate many well-known polynomial families as special cases. So, in the package, there are special functions for generating very popular polynomial families, such as the Bell polynomials. However, further families can be obtained, for suitable choices of the formal power series involved in the composition or when suitable symbolic strategies are employed. In both cases, we give examples on how to modify the R codes of the package to accomplish this task. Future developments are addressed at the end of the paper

1 Introduction

Joint cumulants are usually employed for measuring interactions among two or more random variables simultaneously, extending the familiar notion of covariance to higher orders. More in details, suppose $\boldsymbol{Y}$ a random vector with moment generating function $M_{\boldsymbol{Y}}(\boldsymbol{z})$, for $\boldsymbol{z} = (z_1, \ldots, z_m)$ in a suitable neighborhood of $\boldsymbol{0}$. Thus $M_{\boldsymbol{Y}}(\boldsymbol{z})$ can be expressed as

$$M_{\boldsymbol{Y}}(\boldsymbol{z}) = \exp\left(K_{\boldsymbol{Y}}(\boldsymbol{z})\right) \tag{1}$$

where $K_{\boldsymbol{Y}}(\boldsymbol{z})$ is the cumulant generating function of $\boldsymbol{Y}$. If[1] $\boldsymbol{i} \in \mathbb{N}_0^m$ and

$$M_{\boldsymbol{Y}}(\boldsymbol{z}) = 1 + \sum_{|\boldsymbol{i}|>0} \frac{\mathbb{E}[\boldsymbol{Y}^{\boldsymbol{i}}]}{\boldsymbol{i}!}\boldsymbol{z}^{\boldsymbol{i}} \qquad K_{\boldsymbol{Y}}(\boldsymbol{z}) = \sum_{|\boldsymbol{i}|>0} \frac{k_{\boldsymbol{i}}(\boldsymbol{Y})}{\boldsymbol{i}!}\boldsymbol{z}^{\boldsymbol{i}} \tag{2}$$

then $\{k_{\boldsymbol{i}}(\boldsymbol{Y})\}$ are said the joint cumulants of $\{\mathbb{E}[\boldsymbol{Y}^{\boldsymbol{i}}]\}$. From a theoretical point of view, cumulants are a useful sequence due to the following properties (Elvira Di Nardo 2011):

- *Orthogonality:* Joint cumulants of independent random vectors are zero, that is $k_{\boldsymbol{i}}(\boldsymbol{Y}) = 0$ for $|\boldsymbol{i}| > 0$ if $\boldsymbol{Y} = (\boldsymbol{Y}_1, \boldsymbol{Y}_2)$ with $\boldsymbol{Y}_1$ independent of $\boldsymbol{Y}_2$.
- *Additivity:* Cumulants linearize on independent random vectors, that is $k_{\boldsymbol{i}}(\boldsymbol{Y}_1 + \boldsymbol{Y}_2) = k_{\boldsymbol{i}}(\boldsymbol{Y}_1) + k_{\boldsymbol{i}}(\boldsymbol{Y}_2)$ for $|\boldsymbol{i}| > 0$ with $\boldsymbol{Y}_1$ independent of $\boldsymbol{Y}_2$.
- *Multilinearity:* $k_{\boldsymbol{i}}(A\boldsymbol{Y}) = \sum_{j_1,\ldots,j_m} (A)_{i_1}^{j_1} \cdots (A)_{i_m}^{j_m} k_{\boldsymbol{j}}(\boldsymbol{Y})$ for $|\boldsymbol{i}| > 0$ with $A \in \mathbb{R}^m \times \mathbb{R}^m$.
- *Semi-invariance:* If $\boldsymbol{b} \in \mathbb{R}^m$ then $k_{\boldsymbol{i}}(\boldsymbol{Y} + \boldsymbol{b}) = k_{\boldsymbol{i}}(\boldsymbol{Y})$ for $|\boldsymbol{i}| \geq 2$.

Thanks to all these properties, joint cumulants have a wide range of applications: from statistical inference and time series (Jammalamadaka, Rao, and Terdik 2006) to asymptotic theory (Rao and Wong 1999), from spatial statistics modeling (Dimitrakopoulos, Mustapha, and Gloaguen 2010) to signal processing (Giannakis 1987), from non-linear systems identification (Oualla et al. 2021) to Wiener chaos (Peccati and Taqqu 2011), just to mention a few. Indeed it is also well known that cumulants of order greater than two are zero for random vectors which are Gaussian. Therefore, higher order cumulants are often used in testing for multivariate Gaussianity (Jammalamadaka, Rao, and Terdik 2006).

The $\boldsymbol{i}$-th multivariate k-statistic is a symmetric function of the multivariate random sample whose expectation is the joint cumulant of order $\boldsymbol{i}$ of the population characters. These estimators have minimum variance when compared to all other unbiased estimators and are built by free-distribution methods without using sample moments. Due to the properties of joint cumulants, multivariate k-statistics are employed to check multivariate gaussianity (Ferreira, Magueijo, and Silk 1997) or

[1] If $\boldsymbol{i} \in \mathbb{N}_0^m$ is a multi-index then we set $\boldsymbol{i}! = i_1! \cdots i_m!$ and $|\boldsymbol{i}| = i_1 + \cdots + i_m$.

to quantify high-order interactions among data (Geng, Liang, and Wang 2011), for applications in topology inference (Smith et al. 2022), in neuronal science (Staude, Rotter, and Grün 2010) and in mathematical finance (E. Di Nardo, Marena, and Semeraro 2020). Polykays are unbiased estimators of cumulant products (Robson 1957) and are particularly useful in estimating covariances between k-statistics (McCullagh 1987). In the **kStatistics** package (E. Di Nardo and Guarino 2021), the `nPolyk` function provides k-statistics and polykays as well as their multivariate generalizations. Further implementations are in `Phyton` (Smith 2020), in `Maple` (Guarino, Senato, and Di Nardo 2009) and in `Mathematica` (Rose and Smith 2002).

All these estimators are described with a wealth of details by Stuart and Ord (1994) and McCullagh (1987) and their construction relied on some well-known change of bases in the ring of symmetric polynomials. In Elvira Di Nardo (2011) a different approach is followed using suitable polynomial families and symbolic strategies. This procedure was the core of the first release (version 1.0) of the **kStatistics** package (E. Di Nardo and Guarino 2019), as the initial goal was to implement tools for the estimation of cumulants and cumulant products, both in the univariate and in the multivariate case. As the referred polynomial families can be traced back to the generalized (complete exponential) Bell polynomials, the latest version of the package (E. Di Nardo and Guarino 2021) has also included procedures to generate these polynomials together with a number of special cases.

Let us recall that the generalized (complete exponential) Bell polynomials are a family of polynomials involving multivariable Sheffer sequences (Brown 1979). Among its various applications, we recall the cumulant polynomial sequences and their connection with special families of stochastic processes (E. Di Nardo 2016a). Indeed, cumulant polynomials allow us to compute moments and cumulants of multivariate Lévy processes (E. Di Nardo and Oliva 2011), subordinated multivariate Lévy processes (E. Di Nardo, Marena, and Semeraro 2020) and multivariate compound Poisson processes (E. Di Nardo 2016b). Further examples can be found in Reiner (1976), Shrivastava (2002), Withers and Nadarajah (2010) or Privault (2021).

The generalized (complete exponential) Bell polynomials arise from the multivariate Faà di Bruno's formula, whose computation has been included in the latest version of the **kStatistics** package. In enumerative combinatorics, Faà di Bruno's formula is employed in dealing with formal power series. In particular the multivariate Faà di Bruno's formula gives the $\boldsymbol{i}$-th coefficient of the composition (E. Di Nardo, Guarino, and Senato 2011)

$$h(\boldsymbol{z}) = f\left(g_1(\boldsymbol{z}) - 1, \ldots, g_n(\boldsymbol{z}) - 1\right) \tag{3}$$

where f and g_j for $j = 1, \ldots, n$ are (exponential) formal power series

$$f(\boldsymbol{x}) = \sum_{|\boldsymbol{t}|\geq 0} f_{\boldsymbol{t}} \frac{\boldsymbol{x}^{\boldsymbol{t}}}{\boldsymbol{t}!} \quad \text{and} \quad g_j(\boldsymbol{z}) = \sum_{|\boldsymbol{s}|\geq 0} g_{j;\boldsymbol{s}} \frac{\boldsymbol{z}^{\boldsymbol{s}}}{\boldsymbol{s}!}, \tag{4}$$

with $\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{z} = (z_1, \ldots, z_m)$ and[2] $\boldsymbol{x}^{\boldsymbol{t}} = x_1^{t_1} \cdots x_n^{t_n}$, $\boldsymbol{z}^{\boldsymbol{s}} = z_1^{s_1} \cdots z_m^{s_m}$, $f_{\boldsymbol{t}} = f_{t_1,\ldots,t_n}$, $g_{j;\boldsymbol{s}} = g_{j;s_1,\ldots,s_m}$ for $j = 1, \ldots, n$, and $f_{\boldsymbol{0}} = g_{1;\boldsymbol{0}} = \cdots = g_{n;\boldsymbol{0}} = 1$. For instance, from (1) and (2) joint moments can be recovered from joint cumulants using the multivariate Faà di Bruno's formula for $n = 1$, $g(\boldsymbol{z}) = 1 + K_{\boldsymbol{Y}}(\boldsymbol{z})$ and $f(x) = \exp(x)$. As $1 + K_{\boldsymbol{Y}}(\boldsymbol{z}) = 1 + \log([M_{\boldsymbol{Y}}(\boldsymbol{z}) - 1] + 1)$ then joint cumulants can be recovered from joint moments using the multivariate Faà di Bruno's formula for $n = 1, g(\boldsymbol{z}) = M_{\boldsymbol{Y}}(\boldsymbol{z})$ and $f(x) = 1 + \log(1 + x)$. Let us remark that the exponential form (4) of the formal power series f and $\{g_j\}$ is not a constraint. To work with ordinary formal power series, the multi-index sequence $\{f_{\boldsymbol{t}}\}$ needs to be replaced by the sequence $\{\boldsymbol{t}! f_{\boldsymbol{t}}\}$ as well as the multi-index sequence $\{g_{j;\boldsymbol{s}}\}$ by the sequence $\{\boldsymbol{s}! g_{j;\boldsymbol{s}}\}$ for $j = 1, \ldots, n$. In this case, the multivariate Faà di Bruno's formula gives the coefficient $\boldsymbol{i}!\tilde{h}_{\boldsymbol{i}}$ with $\tilde{h}_{\boldsymbol{i}}$ the $\boldsymbol{i}$-th coefficient of the (ordinary) formal power series composition (3).

The problem of finding suitable and easily manageable expressions of the multivariate Faà di Bruno's formula has received attention from several researchers over the years. This is because the multivariate Faà di Bruno's formula is a very general-purpose tool with many applications. We refer to the paper of Leipnik and Pearce (2007) for a detailed list of references on this subject and a detailed account of its applications. Further applications can be found in Savits (2006), Chacón and Duong (2015), Shabat and Efendiev (2017) and Nguwi, Penent, and Privault (2022). A classical way to generate the multivariate Faà di Bruno's formula involves the partial derivatives of a composition of multivariable functions. Suppose $f(\boldsymbol{x})$ and $g_1(\boldsymbol{z}), \ldots, g_n(\boldsymbol{z})$ in (3) be differentiable functions a certain number of times. The multivariate Faà di Bruno's formula gives the partial derivative of order $\boldsymbol{i}$ of $h(\boldsymbol{z})$ in $\boldsymbol{z}_0$

$$h_{\boldsymbol{i}} = \frac{\partial^{|\boldsymbol{i}|}}{\partial z_1^{i_1} \cdots \partial z_m^{i_m}} h(z_1, \ldots, z_m)\Big|_{z=z_0} \qquad \text{for } |\boldsymbol{i}| > 0, \tag{5}$$

[2]We use these notations independently if the powers or the subscripts are row vectors or column vectors.

assuming the partial derivatives of order $\boldsymbol{t}$ of $f(\boldsymbol{x})$ exist in $\boldsymbol{x}_0 = (g_1(\boldsymbol{z}_0), \ldots, g_n(\boldsymbol{z}_0))$

$$f_{\boldsymbol{t}} = \frac{\partial^{|\boldsymbol{t}|}}{\partial x_1^{t_1} \cdots \partial x_n^{t_n}} f(x_1, \ldots, x_n)\Big|_{\boldsymbol{x}=\boldsymbol{x}_0} \qquad \text{for } 0 < |\boldsymbol{t}| \leq |\boldsymbol{i}|,$$

and the partial derivatives of order $\boldsymbol{s}$ of $g_j(\boldsymbol{z})$ exist in $\boldsymbol{z}_0$ for $j = 1, \ldots, n$

$$g_{j,\boldsymbol{s}} = \frac{\partial^{|\boldsymbol{s}|}}{\partial z_1^{s_1} \cdots \partial z_m^{s_m}} g_j(z_1, \ldots, z_m)\Big|_{\boldsymbol{z}=\boldsymbol{z}_0} \qquad \text{for } 0 < |\boldsymbol{s}| \leq |\boldsymbol{i}|.$$

There are various ways to express $h_{\boldsymbol{i}}$ in (5), see for example Mishkov (2000), Hernández Encinas and Muñoz Masqué (2003) and Ma (2009). Symbolic manipulation using `Macsyma`, `Maple`, `Mathematica`, etc. can produce any required order of (5), by applying the chain rule recursively and using a function that provides partial derivatives. Also in `R`, there are some functions for computing partial derivatives (Clausen and Sokol 2020). Despite its conceptual simplicity, applications of the chain rule become impractical for its cumbersome computation even for small values of its order. As the number of additive terms becomes huge, the output is often untidy and further manipulations are required to simplify the result. By using combinatorial methods, Constantine and Savits (1996) have carried out the following expression of the multivariate Faà di Bruno's formula

$$h_{\boldsymbol{i}} = \boldsymbol{i}! \sum_{1 \leq |\boldsymbol{t}| \leq |\boldsymbol{i}|} f_{\boldsymbol{t}} \sum_{k=1}^{|\boldsymbol{i}|} \sum_{p_k(\boldsymbol{i},\boldsymbol{t})} \prod_{j=1}^{k} \frac{(\mathfrak{g}_{\boldsymbol{l}_j})^{\boldsymbol{q}_j}}{\boldsymbol{q}_j!(\boldsymbol{l}_j!)^{|\boldsymbol{q}_j|}} \tag{6}$$

where $(\mathfrak{g}_{\boldsymbol{s}})^{\boldsymbol{q}} = \prod_{j=1}^{n} (g_{j,\boldsymbol{s}})^{q_j}$ with $\boldsymbol{q} = (q_1, \ldots, q_n)$ and

$$p_k(\boldsymbol{i}, \boldsymbol{t}) = \left\{ (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k; \boldsymbol{l}_1, \ldots, \boldsymbol{l}_k) : |\boldsymbol{q}_j| > 0, \sum_{j=1}^{k} \boldsymbol{q}_j = \boldsymbol{t}, \sum_{j=1}^{k} |\boldsymbol{q}_j| \boldsymbol{l}_j = \boldsymbol{i} \right\}$$

with $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k \in \mathbb{N}_0^n$ and $\boldsymbol{l}_1, \ldots, \boldsymbol{l}_k \in \mathbb{N}_0^m$ such that[3] $\boldsymbol{0} \prec \boldsymbol{l}_1 \prec \ldots \prec \boldsymbol{l}_k$.

A completely different approach concerns the combinatorics of partial derivatives as Hardy (2006) pointed out for the univariate-multivariate composition using multisets and collapsing partitions. Motivated by his results and using the umbral calculus, which is a symbolic method particularly useful in dealing with formal power series (4), the combinatorics behind (6) has been simplified and a different expression has been given in E. Di Nardo, Guarino, and Senato (2011). The key tool is the notion of partition of a multi-index which parallels the multiset partitions given in Hardy (2006).

The contribution of this paper is multi-sided. We explain how to recover in `R` a multi-index partition, which is a combinatorial device. For statistical purposes, we show how to recover k-statistics and their multivariate generalizations using the referred polynomial approach and multi-index partitions. Then, we explain the main steps of the `MFB` function producing the multivariate Faà di Bruno's formula, without any reference to the umbral calculus or chain rules and whose applications go beyond statistical purposes. The main idea is to expand the multivariable polynomial

$$\sum \binom{\boldsymbol{i}}{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n} q_{1,\boldsymbol{s}_1}(y_1) \cdots q_{n,\boldsymbol{s}_n}(y_n)$$

where $q_{1,\boldsymbol{s}_1}(y_1) \ldots q_{n,\boldsymbol{s}_n}(y_n)$ are suitable polynomials and the sum is over all the compositions of $\boldsymbol{i}$ in n parts, that is all the n-tuples $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ of non-negative integer m-tuples such that $\boldsymbol{s}_1 + \cdots + \boldsymbol{s}_n = \boldsymbol{i}$. Readers interested in the umbral setting may refer to Elvira Di Nardo (2011) and references therein.

Consequently, the `MFB` function gives an efficient computation of the following compositions:

- univariate with univariate, that is $n = m = 1$;
- univariate with multivariate, that is $n = 1$ and $m > 1$;
- multivariate with univariate, that is $n > 1$ and $m = 1$;
- multivariate with multivariate, that is $n > 1$ and $m > 1$.

The **kStatistics** package includes additional functions, for some of the most widespread applications of the multivariate Faà di Bruno's formula. Indeed, not only this formula permits to generate joint cumulants and their inverse relations, but also further general families of polynomials. Therefore, we have set up special procedures for those families used very often in applications. These functions

[3] If $\boldsymbol{\mu}, \boldsymbol{\nu} \in \mathbb{N}_0^m$ we have $\boldsymbol{\mu} \prec \boldsymbol{\nu}$ if $|\boldsymbol{\mu}| < |\boldsymbol{\nu}|$ or $|\boldsymbol{\mu}| = |\boldsymbol{\nu}|$ and $\mu_1 < \nu_1$ or $|\boldsymbol{\mu}| = |\boldsymbol{\nu}|$ and $\mu_1 = \nu_1, \ldots, \mu_k = \nu_k, \mu_{k+1} < \nu_{k+1}$ for some $1 \leq k < m$.

should be considered an easy to manage interfaces of the `MFB` function, with the aim of simplifying its application. Moreover, since the `R` codes are free, the user might follow similar steps to generate polynomial families not included in the package but always coming from the multivariate Faà di Bruno's formula. The construction of new families of polynomials can be done mainly in two ways. The first way is to choose appropriately the coefficients $\{f_t\}$ and $\{g_{j;s}\}$ in (4). The second way is to use some suitable symbolic strategies, as discussed in Elvira Di Nardo (2011). For both cases, we provide examples.

The paper is organized as follows. The next section explains the main steps of the algorithm that produces multi-index partitions with particular emphasis on its combinatorics. Then we present the symbolic strategy to generate k-statistics and their generalizations using suitable polynomial sequences and multi-index partitions. The subsequent section deals with generalized (complete exponential) Bell polynomials and some special cases corresponding to well-known families of polynomials. We have also included the procedures to generate joint cumulants from joint moments and vice versa. In the last section we explain the main steps of the algorithm to produce the multivariate Faà di Bruno's formula. We give examples of how to build new polynomials not included in the package. Some concluding remarks end the paper.

2 Partitions of a multi-index

Most routines of the **kStatistics** package use the partitions of a multi-index $\boldsymbol{i}$. Therefore, before describing any of these routines, we recall the notion of multi-index partition and describe the algorithm for its construction as implemented in the `mkmSet` function of the package.

A partition of the multi-index $\boldsymbol{i} = (i_1, \ldots, i_m) \in \mathbb{N}_0^m$ is a matrix $\Lambda = (\boldsymbol{\lambda}_1^{r_1}, \boldsymbol{\lambda}_2^{r_2}, \ldots)$ of non-negative integers with m rows and no zero columns such that

- $r_1 \geq 1$ columns are equal to $\boldsymbol{\lambda}_1$, $r_2 \geq 1$ columns are equal to $\boldsymbol{\lambda}_2$ and so on;
- the columns $\boldsymbol{\lambda}_1 < \boldsymbol{\lambda}_2 < \ldots$ are in lexicographic order[4];
- the sum of the integers in the t-th row is equal to i_t, that is $\lambda_{t1} + \lambda_{t2} + \cdots = i_t$ for $t = 1, 2, \ldots, m$.

We write $\Lambda \vdash \boldsymbol{i}$ to denote that Λ is a partition of $\boldsymbol{i}$. Some further notations are:

- $\mathfrak{m}(\Lambda) = (r_1, r_2, \ldots)$, the vector of multiplicities of $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots$
- $l(\Lambda) = |\mathfrak{m}(\Lambda)| = r_1 + r_2 + \cdots$, the number of columns of Λ with $l(\Lambda) = 0$ if $\Lambda \vdash \mathbf{0}$
- $\Lambda! = (\boldsymbol{\lambda}_1!)^{r_1} (\boldsymbol{\lambda}_2!)^{r_2} \cdots$

Example 1: The partitions of $\boldsymbol{i} = (2, 1)$ are the matrices

$$\begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} = (\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2^2),$$

with

$$\boldsymbol{\lambda}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\lambda}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The algorithm to get all the partitions of a multi-index resorts to multiset subdivisions. Let's start by recalling the notion of multiset. A multiset M is a "set with multiplicities". Suppose $a \in M$. Then the multiplicity of a is the number of times a occurs in M as a member. For example, the integers 3 and 2 are the multiplicities of a and b respectively in $M = \{a, a, a, b, b\}$. A subdivision of the multiset M is a multiset of sub-multisets of M, such that their disjoint union returns M. Examples of subdivisions of $M = \{a, a, a, b, b\}$ are

$$S_1 = \{\{a\}, \{a, b\}, \{a, b\}\}, \qquad S_2 = \{\{a\}, \{a, a, b\}, \{b\}\}, \tag{7}$$

$$S_3 = \{\{a\}, \{a, a\}, \{b\}, \{b\}\}. \tag{8}$$

The subdivisions of the multiset $M = \{a, a, a, b, b\}$ are in one-to-one correspondence with the partitions $\Lambda \vdash (3, 2)$. For example, the subdivisions (7) correspond to the partitions $\Lambda_1 = (\boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3^2)$ and $\Lambda_2 = (\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_5)$ respectively, while (8) to $\Lambda_3 = (\boldsymbol{\lambda}_1^2, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_4)$ with

$$\boldsymbol{\lambda}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \to \{b\} \quad \boldsymbol{\lambda}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \to \{a\}$$

[4] As example $(a_1, b_1) < (a_2, b_2)$ if $a_1 < a_2$ or $a_1 = a_2$ and $b_1 < b_2$.

$$\boldsymbol{\lambda}_3 = \binom{1}{1} \to \{a,b\} \;\; \boldsymbol{\lambda}_4 = \binom{2}{0} \to \{a,a\} \;\; \boldsymbol{\lambda}_5 = \binom{2}{1} \to \{a,a,b\}.$$

Multiset subdivisions can be recovered by using collapsing set partitions (Hardy 2006). If the members $1,2,3$ of the set $\{1,2,3,4,5\}$ are made indistinguishable from each other and called a, and 4 and 5 are made indistinguishable from each other and called b, then the set $\{1,2,3,4,5\}$ has "collapsed" to the multiset $M = \{a,a,a,b,b\}$. Therefore the subdivisions of M can be recovered using the same substitution in the partitions of $\{1,2,3,4,5\}$. For example, S_1 in (7) can be recovered from $\{\{1,4\},\{2,5\},\{3\}\}$ or $\{\{3,5\},\{2,4\},\{1\}\}$ and so on. As this last example shows, a subdivision might correspond to several partitions. The number of partitions corresponding to the same subdivision can be computed using the `countP` function of the package. However, to find multi-index partitions using set partitions is not a particularly efficient algorithm since the computational cost is proportional to the n-th Bell number, if n is the sum of the multi-index components (Charalambides 2002).

The `mkmSet` function is based on a different strategy which takes into account the partitions of the multi-index components. When $m = 1$, the `mkmSet` function lists all the partitions λ of the integer i. Recall that a partition of an integer i is a sequence $\lambda = (\lambda_1, \lambda_2, \ldots)$ of weakly decreasing positive integers, named parts of λ, such that $\lambda_1 + \lambda_2 + \cdots = i$. A different notation is $\lambda = (1^{r_1}, 2^{r_2}, \ldots)$, where $r_1, r_2, \ldots$ are the number of parts of λ equal to $1, 2, \ldots$ respectively. The length of the partition is $l(\lambda) = r_1 + r_2 + \cdots$. We write $\lambda \vdash i$ to denote that λ is a partition of i. In the following, we describe the main steps of the `mkmSet` function by working on an example.

Suppose we want to generate all the partitions of $(3,2)$. First consider the partitions of $(3,0)$ obtained from the partitions $(3), (1,2), (1^3)$ of the integer 3, and the partitions of $(0,2)$ obtained from the partitions $(2), (1^2)$ of the integer 2, that is

$$\Lambda_1 = \binom{3}{0}, \Lambda_2 = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix}, \Lambda_3 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \vdash \binom{3}{0} \tag{9}$$

$$\Lambda_4 = \binom{0}{2}, \Lambda_5 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \vdash \binom{0}{2}. \tag{10}$$

The following iterated adding-appending rule is thus implemented.

1. Consider the partition Λ_5 in (10).

1.1 Add the first column of Λ_5 to each column of Λ_1, Λ_2 and Λ_3 in (9) one by one with the following rules: the sum must be done only once (if the column has multiplicities greater than one) taking as reference the first column; the sum can be done only to columns whose second component is zero and without subsequent elements (in the same row) greater than or equal to the integer we are adding. Then we have

$$\Lambda_1^{(1,1)} = \binom{3}{1} \; \Lambda_2^{(1,1)} = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix} \; \Lambda_2^{(2,1)} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \; \Lambda_3^{(1,1)} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \tag{11}$$

1.2 Append the same column to each partition Λ_1, Λ_2 and Λ_3 in (10), that is

$$\Lambda_1^{(1,2)} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \; \Lambda_2^{(1,2)} = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \; \Lambda_3^{(1,2)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{12}$$

1.3 Repeat steps 1.1 and 1.2 for the second column of Λ_5 with respect to the partitions generated in (11) and (12) :

$$\begin{array}{llll}
\Lambda_1^{(1,1)} = \begin{pmatrix} 3 \\ 1 \end{pmatrix} & \text{add} \Rightarrow \text{rule out} & \text{append} \Rightarrow \begin{pmatrix} 3 & 0 \\ 1 & 1 \end{pmatrix} \\
\Lambda_2^{(1,1)} = \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix} & \text{add} \Rightarrow \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} & \text{append} \Rightarrow \begin{pmatrix} 1 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} \\
\Lambda_2^{(2,1)} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} & \text{add} \Rightarrow \text{rule out} & \text{append} \Rightarrow \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{pmatrix} \\
\Lambda_3^{(1,1)} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} & \text{add} \Rightarrow \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} & \text{append} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \\
\Lambda_1^{(1,2)} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} & \text{add} \Rightarrow \text{rule out} & \text{append} \Rightarrow \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \\
\Lambda_2^{(1,2)} = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \text{add} \Rightarrow \text{rule out} & \text{append} \Rightarrow \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \\
\Lambda_3^{(1,2)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \text{add} \Rightarrow \text{rule out} & \text{append} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}
\end{array}$$

2. Repeat step 1 for Λ_4 in (10) :

$$\Lambda_1 = \begin{pmatrix} 3 \\ 0 \end{pmatrix} \quad \text{add} \Rightarrow \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \text{append} \Rightarrow \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}$$
$$\Lambda_2 = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix} \quad \text{add} \Rightarrow \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 2 \end{pmatrix} \quad \text{append} \Rightarrow \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$
$$\Lambda_3 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{add} \Rightarrow \begin{pmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{pmatrix} \quad \text{append} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

More generally, the `mkmSet` function lists all the partitions $\Lambda \vdash \boldsymbol{i}$, with the columns reordered in increasing lexicographic order, together with the number of set partitions corresponding to the same multi-index partition, that is $\boldsymbol{i}!/\Lambda!\mathfrak{m}(\Lambda)!$. In the latest version of the **kStatistics** package, among the input parameters of the `mkmSet` function, an input flag parameter has been inserted aiming to print the multi-index partitions in a more compact form. See the following example.

Example 2: To get all the partitions of $(2,1)$ run

```
> mkmSet(c(2,1),TRUE)
[( 0 1 )( 1 0 )( 1 0 ),  1 ]
[( 0 1 )( 2 0 ),  1 ]
[( 1 0 )( 1 1 ),  2 ]
[( 2 1 ),  1 ]
```

Note that the integers $1,1,2,1$ correspond to the coefficients $2!1!/\Lambda!\mathfrak{m}(\Lambda)!$.

Example 3: To get all the partitions of the integer 3 run

```
> mkmSet(c(3),TRUE)
[( 1 )( 1 )( 1 ),  1 ]
[( 1 )( 2 ),  3 ]
[( 3 ),  1 ]
```

The `mkmSet` function is called by the `intPart` function, specifically designed with the purpose of listing only all the partitions of a given integer in increasing order. The input flag parameter allows us to print the partitions in a more compact form.

Example 4: To get all the partitions of the integer 4 run

```
> intPart(4,TRUE)
[ 1 1 1 1 ]
[ 1 1 2 ]
[ 2 2 ]
[ 1 3 ]
[ 4 ]
```

The `parts` function of the **partitions** package (Hankin 2006) lists all the partitions of a given integer, but in decreasing order. Instead the `get.partitions` function of the **nilde** package (Arnqvist et al. 2021) finds all the partitions of a given integer with a fixed length $l(\lambda)$ (Voinov and Pya Arnqvist 2017). If $l(\lambda)$ is equal to the given integer, the `get.partitions` function lists all the partitions in increasing order.

3 kStatistics

The i-th k-statistic κ_i is the (unique) symmetric estimator whose expectation is the i-th cumulant $k_i(Y)$ of a population character Y and whose variance is a minimum relative to all other unbiased estimators.

The `nKS` function generates the numerical value of the i-th k-statistic starting from a data sample. The computation relies on the following polynomials

$$\mathcal{P}_t(y) = \sum_{j=1}^{t} y^j S(t,j)(-1)^{j-1}(j-1)! \qquad \text{for} \quad t = 1,\ldots,i \tag{13}$$

where $\{S(t,j)\}$ are the Stirling numbers of second kind, generated trough the `nStirling2` function. In detail, suppose to have a sample $\{a_1,\ldots,a_N\}$ of N numerical data and denote with p_t the t-th power sum in the numerical data

$$p_t(a_1,\ldots,a_N) = \sum_{j=1}^{N} a_j^t, \qquad \text{for } t \geq 1. \tag{14}$$

To carry out the numerical value of the i-th k-statistic for $i \leq N$, the `nKS` function computes the explicit expression of the polynomial of degree i

$$Q_i(y) = \sum_{\lambda \vdash i} d_\lambda \mathcal{P}_\lambda(y) p_\lambda \tag{15}$$

where the sum is over all the partitions $\lambda = (1^{r_1}, 2^{r_2}, \ldots) \vdash i$, and

$$d_\lambda = \frac{i!}{(1!)^{r_1} r_1! (2!)^{r_2} r_2! \cdots} \quad \mathcal{P}_\lambda(y) = [\mathcal{P}_1(y)]^{r_1} [\mathcal{P}_2(y)]^{r_2} \cdots \quad p_\lambda = [p_1]^{r_1} [p_2]^{r_2} \cdots \tag{16}$$

with $\{\mathcal{P}_t(y)\}$ and $\{p_t\}$ given in (13) and (14) respectively. The final step is to replace the powers y^t in the explicit form of the polynomial (15) with $(-1)^{t-1}(t-1)!/(N)_t$ for $t = 1, \ldots, i$.

The main steps of the `nKS` function are summarized in the following.

***Function* `nKS`**

i) Compute the power sums p_t in (14) for $t = 1, \ldots, i$.

ii) Compute $S(t, j)(-1)^{j-1}(j-1)!$ in (13) for $j = 1, \ldots, t$ and $t = 1, \ldots, i$.

iii) Using the `mkmSet` function, compute all the partitions $\lambda \vdash i$.

iv) For a given partition λ, expand the product $\mathcal{P}_\lambda(y)$ in (15) and compute the coefficient $d_\lambda p_\lambda$ of each monomial in $Q_i(y)$ using (16).

v) For $t = 1, \ldots, i$ multiply $(-1)^{t-1}(t-1)!/(N)_t$ with the coefficients of the monomial of degree t carried out at the previous step and do the sum over all the resulting numerical values.

vi) Repeat steps *iv)* and *v)* for all the partitions λ carried out at step *iii)* and do the sum over all the resulting numerical values.

Example 5: Using (15) for $i = 1$, we have $Q_1(y) = \mathcal{P}_1(y)p_1 = y\sum_{j=1}^N a_j$ and plugging $1/N$ in place of y, the sample mean is recovered. Using (15) for $i = 2$, we have

$$Q_2(y) = \mathcal{P}_2(y)p_2 + \left(\mathcal{P}_1(y)p_1\right)^2 = y\sum_{j=1}^N a_j^2 + y^2\left(\left(\sum_{j=1}^N a_j\right)^2 - \sum_{j=1}^N a_j^2\right)$$

and plugging $1/N$ in place of y and $-1/N(N-1)$ in place of y^2, the sample variance is recovered. Compare the values of the sample mean, computed with the `nKS` function and the `mean` function, and the sample variance, computed with the `nKS` function and the `var` function, for the following dataset:

```
> data<-c(16.34, 10.76, 11.84, 13.55, 15.85, 18.20, 7.51, 10.22, 12.52, 14.68,
16.08, 19.43, 8.12, 11.20, 12.95, 14.77, 16.83, 19.80, 8.55, 11.58, 12.10, 15.02,
16.83, 16.98, 19.92, 9.47, 11.68, 13.41, 15.35, 19.11)
> nKS(1,data)
[1] 14.02167
> mean(data)
[1] 14.02167
> nKS(2,data)
[1] 12.65007
> var(data)
[1] 12.65007
```

Using the `nKS` function, for instance, the sample skewness and the sample kurtosis can be computed. Let us recall that the sample skewness is a measure of the central tendency of a univariate sample and can be computed as $\kappa_3/\kappa_2^{3/2}$ where κ_2 and κ_3 are the second and the third k-statistics respectively (Joanes and Gill 1998). The sample kurtosis is a measure of the tail-heaviness of a sample distribution. The sample excess kurtosis is defined as the sample kurtosis minus 3 and can be computed as κ_4/κ_2^2 where κ_2 and κ_4 are the second and the fourth k-statistics respectively (Joanes and Gill 1998).

```
> nKS(3,data)/sqrt(nKS(2,data))^(3/2)
[1] -0.03216229
> nKS(4,data)/nKS(2,data)^2 + 3
[1] 2.114708
```

A similar strategy is employed to compute multivariate k-statistics (the `nKM` function) of a sample data matrix whose columns each represent a population character. To simplify the notation, in the following we deal with the case of a bivariate data set $\{(a_{1,1}, a_{2,1}) \dots, (a_{1,N}, a_{2,N})\}$ of N paired numerical data. Denote with $p_{(s,t)}$ the bivariate power sum in the paired data

$$p_{(s,t)}[(a_{1,1}, a_{2,1}), \dots, (a_{1,N}, a_{2,N})] = \sum_{j=1}^{N} a_{1,j}^{s} a_{2,j}^{t} \quad \text{for } s, t \geq 1. \tag{17}$$

Suppose $\boldsymbol{i} = (i_1, i_2)$ with $i_1, i_2 \leq N$ and set $i = i_1 + i_2$. To carry out the numerical value of the $\boldsymbol{i}$-th multivariate k-statistic, the `nKM` function finds the explicit expression of the polynomial

$$\mathcal{Q}_{\boldsymbol{i}}(y) = \sum_{\Lambda \vdash \boldsymbol{i}} d_{\Lambda} P_{\Lambda}(y) p_{\Lambda} \tag{18}$$

where the sum is over all the partitions $\Lambda = (\boldsymbol{\lambda}_1^{r_1}, \boldsymbol{\lambda}_2^{r_2}, \dots) \vdash \boldsymbol{i}$, and

$$d_{\Lambda} = \frac{\boldsymbol{i}!}{\Lambda!\, \mathfrak{m}(\Lambda)!}\ P_{\Lambda}(y) = [\mathcal{P}_{|\lambda_1|}(y)]^{r_1} [\mathcal{P}_{|\lambda_2|}(y)]^{r_2} \cdots\ p_{\Lambda} = [p_{\lambda_1}]^{r_1} [p_{\lambda_2}]^{r_2} \cdots \tag{19}$$

with $\{\mathcal{P}_t(y)\}$ and $\{p_{(s,t)}\}$ given in (13) and (17) respectively. As for the univariate k-statistics, the final step consists in replacing the powers y^j in the explicit expression of the polynomial (18) with the numerical values $(-1)^{j-1}(j-1)!/(N)_j$ for $j = 1, \dots, i$.
The main steps of the `nKM` function are summarized in the following.

***Function* `nKM`**

i) Compute the bivariate power sums $p_{(s,t)}$ in (17) for $s = 1, \dots, i_1$ and $t = 1, \dots, i_2$.

ii) For $i = i_1 + i_2$, compute $S(t, j)(-1)^{j-1}(j-1)!$ in (13) for $j = 1, \dots, t$ and $t = 1, \dots, i$.

iii) Using the `mkmSet` function, compute all the partitions $\Lambda \vdash \boldsymbol{i}$.

iv) For a given partition Λ, expand the product $P_{\Lambda}(y)$ in (18) and compute the coefficient $d_{\Lambda} p_{\Lambda}$ of each monomial in $\mathcal{Q}_{\boldsymbol{i}}(y)$ using (19).

v) For $j = 1, \dots, i$, multiply $(-1)^{j-1}(j-1)!/(N)_j$ with the coefficient of the monomial of degree j carried out at the previous step and do the sum over all the resulting numerical values.

vi) Repeat steps *iv)* and *v)* for all the partitions Λ carried out at step *iii)* and do the sum over all the resulting numerical values.

Example 6: To estimate the joint cumulant $c_{2,1}$ of the following dataset, run

```
> data1<-list(c(5.31,11.16),c(3.26,3.26),c(2.35,2.35),c(8.32,14.34),
c(13.48,49.45),c(6.25,15.05),c(7.01,7.01),c(8.52,8.52),c(0.45,0.45),
c(12.08,12.08),c(19.39,10.42))
> nKM(c(2,1),data1)
[1] -23.7379
```

If the first column are observations of a population character X and the second column observations of a population character Y, then $c_{2,1}$ measures how far from connectedness (as opposite to independence) are X^2 and Y (E. Di Nardo, Marena, and Semeraro 2020). A similar meaning has the estimation of the joint cumulant $c_{2,2,2}$ of the following dataset:

```
> data2<-list(c(5.31,11.16,4.23),c(3.26,3.26,4.10),c(2.35,2.35,2.27),
c(4.31,10.16,6.45),c(3.1,2.3,3.2),c(3.20, 2.31, 7.3))
> nKM(c(2,2,2),data2)
[1] 678.1045
```

4 Polykays

Similarly to k-statistics, polykays are symmetric unbiased estimators of cumulant products. More in detail, when evaluated on a random sample, the $\boldsymbol{i}$-th polykay gives an estimation of the product $k_{i_1}(Y)\cdots k_{i_m}(Y)$, where $\boldsymbol{i}=(i_1,\ldots,i_m)\in\mathbb{N}_0^m$ and $\{k_{i_j}(Y)\}$ are cumulants of a population character Y.

To simplify the notation, in the following we show how to compute the $\boldsymbol{i}$-th polykay of N numerical data $\{a_1,\ldots,a_N\}$ using the `nPS` function for $\boldsymbol{i}=(i_1,i_2)$. Set $i=i_1+i_2$ and suppose $i\leq N$. The computation relies on the so-called logarithmic polynomials

$$\tilde{P}_t(y_1,\ldots,y_i)=\sum_{\lambda\vdash t} y_\lambda d_\lambda(-1)^{l(\lambda)-1}(l(\lambda)-1)! \tag{20}$$

for $t=1,\ldots,i$ where the sum is over all the partitions $\lambda=(1^{r_1},2^{r_2},\ldots)\vdash t$, d_λ is given in (16) and $y_\lambda=y_1^{r_1}y_2^{r_2}\cdots$. To compute the polykay of order (i_1,i_2), the `nPS` function finds the explicit expression of the polynomial

$$A_i(y_1,\ldots,y_i)=\sum_{\lambda\vdash i} d_\lambda\tilde{P}_\lambda(y_1,\ldots,y_i)p_\lambda \tag{21}$$

where the sum is over all the partitions $\lambda=(1^{r_1},2^{r_2},\ldots)\vdash i$, d_λ and p_λ are given in (16) and

$$\tilde{P}_\lambda(y_1,\ldots,y_i)=[\tilde{P}_1(y_1,\ldots,y_i)]^{r_1}[\tilde{P}_2(y_1,\ldots,y_i)]^{r_2}\cdots$$

with $\{\tilde{P}_t(y_1,\ldots,y_i)\}$ given in (20). Note that the monomials in $A_i(y_1,\ldots,y_i)$ are of type $y_\lambda=y_1^{r_1}y_2^{r_2}\cdots$ with $\lambda=(1^{r_1},2^{r_2},\ldots)\vdash i$. The final step is to plug suitable numerical values in place of y_λ depending on how the partition λ is constructed. Indeed, set

$$\tilde{q}(i_1,i_2)=\left\{\lambda'+\lambda''\vdash i \,\middle|\, \lambda'=(1^{s_1},2^{s_2},\ldots)\vdash i_1,\lambda''=(1^{t_1},2^{t_2},\ldots)\vdash i_2\right\} \tag{22}$$

where $\lambda'+\lambda''=(1^{r_1},2^{r_2},\ldots)$ with $r_j=s_j+t_j$ for $j=1,2,\ldots$. Then y_λ is replaced by 0 if $\lambda\notin\tilde{q}(i_1,i_2)$ otherwise by

$$\frac{(-1)^{l(\lambda')-1}(l(\lambda')-1)!(-1)^{l(\lambda'')-1}(l(\lambda'')-1)!}{(N)_{l(\lambda'')+l(\lambda'')}}\frac{d_{\lambda'}d_{\lambda''}}{d_{\lambda'+\lambda''}}. \tag{23}$$

Note that $d_{\lambda'}$ and $d_{\lambda''}$ in (23) are recovered from (16).

The main steps of the `nPS` function are summarized in the following.

***Function* nPS**

i) Set $i=i_1+i_2$ and compute the power sums p_t in (14) for $t=1,\ldots,i$.

ii) Generate the polynomials $\tilde{P}_t(y_1,\ldots,y_i)$ in (20) for $t=1,\ldots,i$.

iii) Using the `mkmSet` function, compute all the partitions $\lambda\vdash i$.

iv) For a given partition λ, expand the product $\tilde{P}_\lambda(y_1,\ldots,y_i)$ in (21); then plug (23) or 0 in each monomial y_λ, depending if λ is or not in the set $\tilde{q}(i_1,i_2)$ given in (22).

v) Multiply the numerical value of $\tilde{P}_\lambda$ carried out at step *iv)* with $d_\lambda p_\lambda$ given in (16).

vi) Repeat steps *iv)* and *v)* for all the partitions λ carried out at step *iii)* and do the sum over all the resulting numerical values.

Example 7: Suppose we need to estimate the square of the variance σ^2 of the population character Y from which the data of Example 5 have been sampled. We have

```
> nKS(2,data)^2
[1] 160.0243
> var(data)^2
[1] 160.0243
```

but k_2^2 is not an unbiased estimator of the square of σ^2. An unbiased estimator of such a square is the polykay of order $(2,2)$, that is

```
> nPS(c(2,2),data)
[1] 154.1177
```

Multivariate polykays are unbiased estimators of products of multivariate cumulants and the nPM function returns a numerical value for these estimators when evaluated on a random sample. As before, to show how the nPM function works, we consider a bivariate sample of N numerical data, that is $\{(a_{1,1}, a_{2,1}) \ldots, (a_{1,N}, a_{2,N})\}$. If we choose $\boldsymbol{i} = (i_1, i_2)$ and $\boldsymbol{j} = (j_1, j_2)$ with $i_1 + i_2 + j_1 + j_2 \leq N$ as input of the nPM function, the output is a numerical value which represents an estimated value of the product $k_i(X,Y)k_j(X,Y)$, where $k_i(X,Y)$ and $k_j(X,Y)$ are cumulants of the population characters (X,Y). The computation relies on suitable polynomials in the indeterminates $\{y_{(s,t)}\}$ for $s = 0, \ldots, w_1, t = 0, \ldots, w_2$, with $s + t > 0$ and $w_1 = i_1 + j_1, w_2 = i_2 + j_2$. These polynomials are a multivariable generalization of (20), that is

$$\tilde{P}_{\boldsymbol{k}}\left(\{y_{(s,t)}\}\right) = \sum_{\Lambda \vdash \boldsymbol{k}} y_\Lambda d_\Lambda (-1)^{l(\Lambda)-1}(l(\Lambda)-1)! \tag{24}$$

for $\boldsymbol{0} < \boldsymbol{k} \leq \boldsymbol{w} = (w_1, w_2)$, where the sum is over all the partitions $\Lambda = (\lambda_1^{r_1}, \lambda_2^{r_2}, \ldots) \vdash \boldsymbol{k}$ and $y_\Lambda = y_{\lambda_1}^{r_1} y_{\lambda_2}^{r_2} \cdots$. To compute the multivariate polykay of order $(\boldsymbol{i}, \boldsymbol{j})$, the nPM function finds the explicit expression of the polynomial

$$\mathcal{A}_{\boldsymbol{w}}\left(\{y_{(s,t)}\}\right) = \sum_{\Lambda \vdash \boldsymbol{w}} d_\Lambda \tilde{P}_\Lambda\left(\{y_{(s,t)}\}\right) p_\Lambda \tag{25}$$

where the sum is over all the partitions $\Lambda = (\boldsymbol{\lambda}_1^{r_1}, \boldsymbol{\lambda}_2^{r_2}, \ldots) \vdash \boldsymbol{w}$, d_Λ and p_Λ are given in (19), and

$$\tilde{P}_\Lambda\left(\{y_{(s,t)}\}\right) = [\tilde{P}_{\lambda_1}\left(\{y_{(s,t)}\}\right)]^{r_1}[\tilde{P}_{\lambda_2}\left(\{y_{(s,t)}\}\right)]^{r_2} \cdots$$

with $\{\tilde{P}_\lambda\left(\{y_{(s,t)}\}\right)\}$ given in (24). The monomials in $\mathcal{A}_{\boldsymbol{w}}\left(\{y_{(s,t)}\}\right)$ are of type y_Λ with $\Lambda \vdash \boldsymbol{w}$. The final step is to plug suitable numerical values in place of y_Λ depending on how the partition Λ is constructed. Indeed, set

$$\tilde{q}(\boldsymbol{w}) = \left\{\Lambda' + \Lambda'' \vdash \boldsymbol{w} \,|\, \Lambda' = (\boldsymbol{\lambda}_1'^{s_1}, \boldsymbol{\lambda}_2'^{s_2}, \ldots) \vdash \boldsymbol{i}, \Lambda'' = (\boldsymbol{\lambda}_1''^{t_1}, \boldsymbol{\lambda}_2''^{t_2}, \ldots) \vdash \boldsymbol{j}\right\}, \tag{26}$$

where $\Lambda' + \Lambda'' = (\tilde{\boldsymbol{\lambda}}_1^{r_1}, \tilde{\boldsymbol{\lambda}}_2^{r_2}, \ldots)$ is built with the columns of Λ' and Λ'' rearranged in increasing lexicographic order and such that $r_j = s_j$ if $\tilde{\boldsymbol{\lambda}}_j = \boldsymbol{\lambda}_j'$ or $r_j = t_j$ if $\tilde{\boldsymbol{\lambda}}_j = \boldsymbol{\lambda}_j''$ or $r_j = s_j + t_j$ if $\tilde{\boldsymbol{\lambda}}_j = \boldsymbol{\lambda}_j' = \boldsymbol{\lambda}_j''$. Therefore in the explicit expression of (25), y_Λ is replaced by 0 if $\Lambda \notin \tilde{q}(\boldsymbol{w})$ otherwise by

$$\frac{(-1)^{l(\Lambda')-1}(l(\Lambda')-1)!(-1)^{l(\Lambda'')-1}(l(\Lambda'')-1)!}{(N)_{l(\Lambda')+l(\Lambda'')}} \frac{d_{\Lambda'} d_{\Lambda''}}{d_{\Lambda'+\Lambda''}}. \tag{27}$$

Note that $d_{\Lambda'}$ and $d_{\Lambda''}$ in (27) are recovered from (19).

The main steps of the nPM function are summarized in the following.

***Function* nPM**

i) Set $w_1 = i_1 + j_1$ and $w_2 = i_2 + j_2$; compute the power sums $p_{(s,t)}$ in (17) for $s = 1, \ldots, w_1$ and $t = 1, \ldots, w_2$.

ii) Generate the polynomials $\tilde{P}_{\boldsymbol{k}}\left(\{y_{(s,t)}\}\right)$ in (24) for $0 < \boldsymbol{k} \leq \boldsymbol{w} = (w_1, w_2)$.

iii) Using the mkmSet function, compute all the partitions $\Lambda \vdash \boldsymbol{w}$.

iv) For a given partition Λ, expand the product $\tilde{P}_\Lambda\left(\{y_{(s,t)}\}\right)$ in (25) and plug (27) or 0 in each obtained monomial of type y_Λ depending if Λ is or not in $\tilde{q}(\boldsymbol{w})$ given in (26).

v) Multiply the numerical value of $\tilde{P}_\Lambda$ obtained at step *iv)* with $d_\Lambda p_\Lambda$ given in (19).

vi) Repeat steps *iv)* and *v)* for all the partitions Λ carried out at step *iii)* and do the sum over all the resulting numerical values.

Example 8: For the same dataset employed in Example 6, to estimate $k_{(2,1)}(X,Y)k_{(1,0)}(X,Y)$ run

```
> nPM(list(c(2,1),c(1,0)),data1)
[1] 48.43243
```

Remark 1: The master nPolyk function runs one of the nKS, nKM, nPS and nPM functions depending if we ask for simple k-statistics, multivariate k-statistics, simple polykays or multivariate polykays.

5 Bell polynomials and generalizations

The algorithms to produce k-statistics and polykays rely on handling suitable polynomial families which are special cases of generalizations of Bell polynomials, as introduced in this section. Moreover, there are further families of polynomials widely used in applications which are special cases of these polynomials. For the most popular ones, we have implemented special functions in the **kStatistics** package. The list is not exhaustive, see for instance Roman (1984). Furthermore additional families of polynomials might be recovered using the multivariate Faà di Bruno's formula. We will give some examples in the next section.

The $\boldsymbol{i}$-th generalized (complete exponential) Bell polynomial in the indeterminates $y_1, \ldots, y_n$ is

$$h_{\boldsymbol{i}}(y_1, \ldots, y_n) = \boldsymbol{i}! \sum_{\substack{\Lambda \vdash \boldsymbol{s}_1, \ldots, \tilde{\Lambda} \vdash \boldsymbol{s}_n \\ \boldsymbol{s}_1 + \cdots + \boldsymbol{s}_n = \boldsymbol{i}}} y_1^{l(\Lambda)} \cdots y_n^{l(\tilde{\Lambda})} \frac{g_{1,\Lambda} \cdots g_{n,\tilde{\Lambda}}}{\Lambda! \cdots \tilde{\Lambda}!\, \mathfrak{m}(\Lambda)! \cdots \mathfrak{m}(\tilde{\Lambda})!} \tag{28}$$

where the sum is over all the partitions $\Lambda \vdash \boldsymbol{s}_1, \ldots, \tilde{\Lambda} \vdash \boldsymbol{s}_n$ with $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ m-tuples of non-negative integers such that $\boldsymbol{s}_1 + \cdots + \boldsymbol{s}_n = \boldsymbol{i}$ and

$$\begin{aligned} g_{1,\Lambda} &= g_{1;\boldsymbol{\lambda}_1}^{r_1} g_{1;\boldsymbol{\lambda}_2}^{r_2} \cdots \qquad \text{for } \Lambda = (\boldsymbol{\lambda}_1^{r_1}, \boldsymbol{\lambda}_2^{r_2}, \ldots) \\ &\vdots \\ g_{n,\tilde{\Lambda}} &= g_{n;\tilde{\boldsymbol{\lambda}}_1}^{t_1} g_{n;\tilde{\boldsymbol{\lambda}}_2}^{t_2} \cdots \qquad \text{for } \tilde{\Lambda} = (\tilde{\boldsymbol{\lambda}}_1^{t_1}, \tilde{\boldsymbol{\lambda}}_2^{t_2}, \ldots) \end{aligned} \tag{29}$$

with $\{g_{1;\lambda}\}, \ldots, \{g_{n;\tilde{\lambda}}\}$ multi-indexed sequences. These polynomials are the output of the `GCBellPol` function.

Example 9: To get $h_{(1,1)}(y_1, y_2)$ run

```
> GCBellPol(c(1,1),2)
[1] (y1)(y2)g1[0,1]g2[1,0] + (y1)(y2)g1[1,0]g2[0,1] + (y1^2)g1[0,1]g1[1,0] +
(y1)g1[1,1] + (y2^2)g2[0,1]g2[1,0] + (y2)g2[1,1]
```

The `e_GCBellPol` function evaluates $h_{\boldsymbol{i}}(y_1, \ldots, y_n)$ when its indeterminates $y_1, \ldots, y_n$ and/or its coefficients are substituted with numerical values.

Example 10: To plug the values from 1 to 6 respectively into the coefficients `g1[ , ]` and `g2[ , ]` of the polynomial $h_{(1,1)}(y_1, y_2)$ given in Example 9 run

```
> e_GCBellPol(c(1,1), 2, "g1[0,1]=1, g1[1,0]=2, g1[1,1]=3, g2[0,1]=4, g2[1,0]=5,
g2[1,1]=6")
[1] 13(y1)(y2) + 2(y1^2) + 3(y1) + 20(y2^2) + 6(y2)
```

To evaluate $h_{(1,1)}(1, 5)$ run

```
> e_GCBellPol(c(1,1), 2, "y1=1,  y2=5, g1[0,1]=1, g1[1,0]=2, g1[1,1]=3, g2[0,1]=4,
g2[1,0]=5, g2[1,1]=6")
[1] 600
```

When the multi-indexed sequences $\{g_{1;\lambda}\}, \ldots, \{g_{n;\tilde{\lambda}}\}$ are all equal, the number of distinct addends in (28) might reduce and the corresponding generalized Bell polynomial is denoted by $\tilde{h}_{\boldsymbol{i}}(y_1, \ldots, y_n)$. To deal with this special case, we have inserted an input flag parameter in the `e_GCBellPol` function.

Example 11: To compare $\tilde{h}_{(1,1)}(y_1, y_2)$ with $h_{(1,1)}(y_1, y_2)$ given in Example 9 run

```
> GCBellPol(c(1,1),2,TRUE)
[1] 2(y1)(y2)g[0,1]g[1,0] + (y1^2)g[0,1]g[1,0] + (y1)g[1,1] + (y2^2)g[0,1]g[1,0] +
(y2)g[1,1]
```

Set $n = 1$ in (28). Then $h_{\boldsymbol{i}}(y_1, \ldots, y_n)$ reduces to the univariate polynomial

$$h_{\boldsymbol{i}}(y) = \sum_{\Lambda \vdash \boldsymbol{i}} y^{l(\Lambda)} d_\Lambda g_\Lambda \tag{30}$$

where the sum is over all the partitions $\Lambda = (\boldsymbol{\lambda}_1^{r_1}, \boldsymbol{\lambda}_2^{r_2}, \ldots) \vdash \boldsymbol{i}$, d_Λ is given in (19) and $g_\Lambda = g_{\boldsymbol{\lambda}_1}^{r_1} g_{\boldsymbol{\lambda}_2}^{r_2} \cdots$.

Example 12: To get $h_{(1,1)}(y)$ run

```
> GCBellPol(c(1,1),1)
[1] (y^2)g[0,1]g[1,0] + (y)g[1,1]
```

Remark 2: For all $\boldsymbol{i} \in \mathbb{N}_0^m$, we have $h_{\boldsymbol{i}}(y_1 + \cdots + y_n) = \tilde{h}_{\boldsymbol{i}}(y_1, \ldots, y_n)$, where $\tilde{h}_{\boldsymbol{i}}(y_1, \ldots, y_n)$ is the $\boldsymbol{i}$-th generalized Bell polynomial (28) corresponding to all equal multi-indexed sequences $\{g_{1,\lambda}\}, \ldots, \{g_{n,\bar{\lambda}}\}$ (Elvira Di Nardo 2011). Therefore the `e_GCBellPol` function, with the input flag `TRUE`, produces also an explicit expression of $h_{\boldsymbol{i}}(y_1 + \cdots + y_n)$.

The algorithm to generate joint moments in terms of joint cumulants and vice versa follows the same pattern designed to generate $\{h_{\boldsymbol{i}}(y)\}$. Indeed if $\{k_{\boldsymbol{i}}(\boldsymbol{Y})\}$ and $\{m_{\boldsymbol{i}}(\boldsymbol{Y})\}$ denote the sequences of joint cumulants and joint moments of a random vector $\boldsymbol{Y}$ respectively, then

$$m_{\boldsymbol{i}}(\boldsymbol{Y}) = \sum_{\Lambda \vdash \boldsymbol{i}} d_\Lambda k_\Lambda(\boldsymbol{Y}) \text{ and } k_{\boldsymbol{i}}(\boldsymbol{Y}) = \sum_{\Lambda \vdash \boldsymbol{i}} (-1)^{l(\Lambda)-1}(l(\Lambda)-1)! d_\Lambda m_\Lambda(\boldsymbol{Y}), \tag{31}$$

where the sum is over all the partitions $\Lambda = (\boldsymbol{\lambda}_1^{r_1}, \boldsymbol{\lambda}_2^{r_2}, \ldots) \vdash \boldsymbol{i}$, d_Λ is given in (19) and

$$m_\Lambda(\boldsymbol{Y}) = [m_{\boldsymbol{\lambda}_1}(\boldsymbol{Y})]^{r_1}[m_{\boldsymbol{\lambda}_2}(\boldsymbol{Y})]^{r_2} \cdots \quad k_\Lambda(\boldsymbol{Y}) = [k_{\boldsymbol{\lambda}_1}(\boldsymbol{Y})]^{r_1}[k_{\boldsymbol{\lambda}_2}(\boldsymbol{Y})]^{r_2} \cdots.$$

In particular

- the `mom2cum` function returns the right hand side of the first equation in (31), using the same algorithm producing $h_{\boldsymbol{i}}(y)$ in (30) with the sequence $\{k_{\boldsymbol{\lambda}}\}$ in place of $\{g_{\boldsymbol{\lambda}}\}$ and with 1 in place of y;
- the `cum2mom` function returns the right hand side of the latter equation in (31), using the same algorithm producing $h_{\boldsymbol{i}}(y)$ in (30) with the sequence $\{m_{\boldsymbol{\lambda}}\}$ in place of $\{g_{\boldsymbol{\lambda}}\}$ and with $(-1)^{j-1}(j-1)!$ in place of the powers y^j for $j = 1, \ldots, |\boldsymbol{i}|$.

When the multi-index $\boldsymbol{i}$ reduces to an integer i, formulae (31) are the classical expressions of univariate moments in terms of univariate cumulants and vice versa. The `mom2cum` and `cum2mom` functions do the same when the input is an integer.

Example 13: To get $m_{(3,1)}$ in terms of $k_{(i,j)}$ run

```
> mom2cum(c(3,1))
[1] k[0,1]k[1,0]^3 + 3k[0,1]k[1,0]k[2,0] + k[0,1]k[3,0] + 3k[1,0]^2k[1,1] +
3k[1,0]k[2,1] + 3k[1,1]k[2,0] + k[3,1]
```

To get $k_{(3,1)}$ in terms of $m_{(i,j)}$ run

```
> cum2mom(c(3,1))
[1]  - 6m[0,1]m[1,0]^3 + 6m[0,1]m[1,0]m[2,0] - m[0,1]m[3,0] +
6m[1,0]^2m[1,1] - 3m[1,0]m[2,1] - 3m[1,1]m[2,0] + m[3,1]
```

Remark 3: There are different functions in `R` performing similar computations for cumulants and moments: for instance see De Leeuw, J. (2012) for the multivariate case. A different strategy would rely on the recursive relations between cumulants and moments (Domino, Gawron, and Pawela 2018).

Similarly to (31), some of the polynomials employed in the previous sections are generated using the same pattern developed to find the explicit expression of $h_{\boldsymbol{i}}(y)$ in (30):

- The generation of an explicit expression of $\mathcal{Q}_{\boldsymbol{i}}(y)$ in (18) parallels the one implemented for $h_{\boldsymbol{i}}(y)$ with 1 in place of y and with the polynomial sequence $\{\mathcal{P}_{|\lambda|}(y)p_\lambda\}$ in place of the sequence $\{g_\lambda\}$;
- the same for the polynomials $\tilde{P}_k\left(\{y_{(s,t)}\}\right)$ in (24) with $(-1)^{j-1}(j-1)!$ for $j = 1, \ldots, |\boldsymbol{i}|$ in place of the powers y^j and with the polynomial sequence $\{y_\lambda\}$ in place of the sequence $\{g_\lambda\}$;
- the same for the polynomials $\mathcal{A}_w\left(\{y_{(s,t)}\}\right)$ in (25) with 1 in place of y and with the polynomial sequence $\left\{\tilde{P}_\lambda\left(\{y_{(s,t)}\}\right)p_\lambda\right\}$ in place of the sequence $\{g_\lambda\}$.

Note that when the multi-index $\boldsymbol{i}$ in (30) reduces to a positive integer i, then the polynomial $h_{\boldsymbol{i}}(y)$ becomes

$$h_i(y) = \sum_{\lambda \vdash i} d_\lambda y^{l(\lambda)} g_\lambda \tag{32}$$

where the sum is over all the partitions $\lambda = (1^{r_1}, 2^{r_2}, \ldots) \vdash i$, d_λ is given in (16) and $g_\lambda = g_1^{r_1} g_2^{r_2} \ldots$ with $\{g_j\}$ a suitable sequence.

Example 14: To get $h_3(y)$ run

```
> GCBellPol(c(3),1)
[1] (y^3)g[1]^3 + 3(y^2)g[1]g[2] + (y)g[3]
```

With a combinatorial structure very similar to (32), the i-th general partition polynomial has the following expression in the indeterminates $y_1, \ldots, y_i$

$$G_i(a_1, \ldots, a_i; y_1, \ldots, y_i) = \sum_{\lambda \vdash i} d_\lambda a_{l(\lambda)} y_\lambda \tag{33}$$

where the sum is over all the partitions $\lambda = (1^{r_1}, 2^{r_2}, \ldots) \vdash i$, d_λ is given in (16), $\{a_j\}$ is a suitable numerical sequence and $y_\lambda = y_1^{r_1} y_2^{r_2} \ldots$. It's a straightforward exercise to prove that

$$G_i(a_1, \ldots, a_i; y_1, \ldots, y_i) = \sum_{j=1}^{i} a_j B_{i,j}(y_1, \ldots, y_{i-j+1}), \tag{34}$$

where $\{B_{i,j}\}$ are the (partial) exponential Bell polynomials

$$B_{i,j}(y_1, \ldots, y_{i-j+1}) = \sum_{\bar{p}(i,j)} d_\lambda y_\lambda \tag{35}$$

where $\bar{p}(i,j) = \{\lambda = (1^{r_1}, 2^{r_2}, \ldots) \vdash i | l(\lambda) = j\}$, d_λ is given in (16) and $y_\lambda = y_1^{r_1} y_2^{r_2} \cdots$. The polynomials in (33) are widely used in applications such as combinatorics, probability theory and statistics (Charalambides 2002). As particular cases, they include the exponential polynomials and their inverses, the logarithmic polynomials (20), the potential polynomials and many others (Roman 1984). The general partition polynomials are the output of the `gpPart` function.

Example 15: To get $G_4(a_1, a_2, a_3, a_4; y_1, y_2, y_3, y_4)$ run

```
> gpPart(4)
[1] a4(y1^4) + 6a3(y1^2)(y2) + 3a2(y2^2) + 4a2(y1)(y3) + a1(y4)
```

When $a_1 = \ldots = a_i = 1$, the i-th general partition polynomial in (34) reduces to the complete (exponential) Bell polynomial

$$G_i(1, \ldots, 1; y_1, \ldots, y_i) = \sum_{j=1}^{i} B_{i,j}(y_1, \ldots, y_{i-j+1}) \tag{36}$$

where $\{B_{i,j}\}$ are the (partial) exponential Bell polynomials (35). For instance, the polynomial $Q_i(y)$ in (15) is generated using the same pattern developed to generate (36) with $\mathcal{P}_j(y)p_j$ in place of y_j.

The `eBellPol` function returns the complete (exponential) Bell polynomials (36). The same function also produces the (partial) exponential Bell polynomial $B_{i,j}(y_1, \ldots, y_{i-j+1})$ using (33) with $a_k = \delta_{k,j}$ (the Kronecker delta) for $k = 1, \ldots, i$. Mihoubi (2008) gives a rather extensive survey of applications of these homogeneous polynomials.

Example 16: To get $B_{5,3}(y_1, y_2, y_3)$ run

```
> eBellPol(5,3)
[1] 15(y1)(y2^2) + 10(y1^2)(y3)
```

To get $G_4(1, 1, 1, 1; y_1, y_2, y_3, y_4)$ run

```
> eBellPol(4)
[1] (y1^4) + 6(y1^2)(y2) + 3(y2^2) + 4(y1)(y3) + (y4)
```

The `oBellPol` function returns the partial (ordinary) Bell polynomials

$$\hat{B}_{i,j}(y_1, \ldots, y_{i-j+1}) = \frac{j!}{i!} B_{i,j}(1!y_1, 2!y_2, \ldots, (i-j+1)!y_{i-j+1})$$

and the complete (ordinary) Bell polynomials

$$\hat{G}_i(y_1, \ldots, y_i) = G_i(1, \ldots, 1; 1!y_1, 2!y_2, \ldots, i!y_i).$$

Example 17: To get $\hat{B}_{5,3}(y_1, y_2, y_3)$ run

```
> oBellPol(5,3)
[1] 1/120( 360(y1)(y2^2) + 360(y1^2)(y3) )
```

To get $\hat{G}_3(y_1, y_2, y_3, y_4)$ run

```
> oBellPol(4)
[1] 1/24( 24(y1^4) + 72(y1^2)(y2) + 24(y2^2) + 48(y1)(y3) + 24(y4) )
```

The `e_eBellPol` function evaluates the exponential Bell polynomials when the indeterminates are substituted with numerical values. In Table 1 some special sequence of numbers are given obtained using this procedure.

Table 1: *Numerical sequences (second column) obtained evaluating the exponential Bell polynomials (last column) when suitable numerical values replace indeterminates.*

	Sequence	Bell polynomials
Lah numbers	$\frac{i!}{j!}\binom{i-1}{j-1}$	$B_{i,j}(1!, 2!, 3!, \ldots)$
Stirling numbers of first kind	$s(i,j)$	$B_{i,j}(0!, -1!, 2!, \ldots)$
unsigned Stirling numbers of first kind	$\lvert s(i,j)\rvert$	$B_{i,j}(0!, 1!, 2!, \ldots)$
Stirling numbers of second kind	$S(i,j)$	$B_{i,j}(1, 1, 1, \ldots)$
idempotent numbers	$\binom{i}{j} j^{i-j}$	$B_{i,j}(1, 2, 3, \ldots)$
Bell numbers	B_i	$\sum_{j=0}^{i} B_{i,j}(1, 1, 1, \ldots)$

By default, the `e_eBellPol` function returns the Stirling numbers of second kind, as the following example shows.

Example 18: To get $S(5,3)$ run

```
> e_eBellPol(5,3)
[1] 25
> e_eBellPol(5,3,c(1,1,1,1,1))
[1] 25
```

To get the 5-th Bell number B_5 run

```
> e_eBellPol(5)
[1] 52
```

To get $s(5,3)$ run

```
> e_eBellPol(5,3, c(1,-1,2,-6,24))
[1] 35
```

6 Composition of formal power series

In (3), suppose $f_{\boldsymbol{t}}$ the $\boldsymbol{t}$-th coefficient of $f(\boldsymbol{x})$ and $g_{1;\boldsymbol{s}_1}, \ldots, g_{n;\boldsymbol{s}_n}$ the $\boldsymbol{s}_1$-th,$\ldots$,$\boldsymbol{s}_n$-th coefficients of $g_1(\boldsymbol{z}), \ldots, g_n(\boldsymbol{z})$ respectively. Using multi-index partitions, the multivariate Faà di Bruno's formula (6) can be written as (E. Di Nardo, Guarino, and Senato 2011)

$$h_{\boldsymbol{i}} = \boldsymbol{i}! \sum_{\substack{\Lambda \vdash \boldsymbol{s}_1, \ldots, \tilde{\Lambda} \vdash \boldsymbol{s}_n \\ \boldsymbol{s}_1 + \cdots + \boldsymbol{s}_n = \boldsymbol{i}}} f_{(l(\Lambda), \ldots, l(\tilde{\Lambda}))} \frac{g_{1,\Lambda} \cdots g_{n,\tilde{\Lambda}}}{\Lambda! \cdots \tilde{\Lambda}!\, \mathfrak{m}(\Lambda)! \cdots \mathfrak{m}(\tilde{\Lambda})!} \tag{37}$$

where $g_{1,\Lambda}, \ldots, g_{n,\tilde{\Lambda}}$ are given in (29) and the sum is over all the partitions $\Lambda \vdash \boldsymbol{s}_1, \ldots, \tilde{\Lambda} \vdash \boldsymbol{s}_n$, with $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ m-tuples of non-negative integers such that $\boldsymbol{s}_1 + \cdots + \boldsymbol{s}_n = \boldsymbol{i}$.

The `MFB` function generates all the summands of (37). Its first step is to find the set $\tilde{p}(n, \boldsymbol{i})$ of all the compositions of $\boldsymbol{i}$ in n parts, that is all the n-tuples $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ of non-negative integer m-tuples such that $\boldsymbol{s}_1 + \cdots + \boldsymbol{s}_n = \boldsymbol{i}$. This task is performed by the `mkT` function.

Function **`mkT`**

i) Find all the partitions $\Lambda \vdash \boldsymbol{i}$, using the `mkmSet` function.

ii) Select the first partition Λ. If $l(\Lambda) = n$, then the columns of Λ are the m-tuples $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ such that $\boldsymbol{s}_1 + \ldots + \boldsymbol{s}_n = \boldsymbol{i}$. If $l(\Lambda) < n$, add $n - l(\Lambda)$ zero columns to Λ.

iii) Generate all the permutations of the columns of Λ as collected at step *ii)*.

iv) Repeat steps *ii)* and *iii)* for each partition Λ carried out at step *i)*.

In the `mkT` function an input flag variable permits to obtain the output in a more compact set up. See the following example.

Example 19: Suppose we are looking for the elements of the set $\tilde{p}(2,(2,1))$, that is the pairs $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ such that $\boldsymbol{s}_1 + \boldsymbol{s}_2 = (2,1)$. Then run

```
> mkT(c(2,1),2,TRUE)
[( 0 1 )( 2 0 )]
[( 2 0 )( 0 1 )]
[( 1 0 )( 1 1 )]
[( 1 1 )( 1 0 )]
[( 2 1 )( 0 0 )]
[( 0 0 )( 2 1 )]
```

Consider the partitions of $(2,1)$ as given in Example 2. Note that `[( 2 1 )( 0 0 )]` and `[( 0 0 )( 2 1 )]` are obtained adding a zero column to the partition `[( 2 1 ),  1 ]`, and then permuting the two columns. No zero columns are added to `[( 2 0 )( 0 1 )]` as the length of the partition is 2. The same is true for `[( 0 1 )( 2 0 )]` or `[( 1 1 )( 1 0 )]` which are only permuted.

The `MFB` function produces the multivariate Faà di Bruno's formula (37) making use of the following steps.

Function **`MFB`**

i) Find all the m-tuples $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ in $\tilde{p}(n, \boldsymbol{i})$ using the `mkT` function.

ii) Let $y_1, \ldots, y_n$ be indeterminates. For each $j = 1, \ldots, n$, compute all the partitions $\Lambda \vdash \boldsymbol{s}_j$ using the `mkmSet` function and find the explicit expression of the polynomial

$$q_{j,\boldsymbol{s}_j}(y_j) = \boldsymbol{s}_j! \sum_{\Lambda \vdash \boldsymbol{s}_j} y_j^{l(\Lambda)} \frac{g_{j,\Lambda}}{\Lambda! \mathfrak{m}(\Lambda)!}.$$

iii) Make the products $q_{1,\boldsymbol{s}_1}(y_1) \cdots q_{n,\boldsymbol{s}_n}(y_n)$ in the multivariable polynomial

$$h_{\boldsymbol{i}}(y_1, \ldots, y_n) = \sum_{(\boldsymbol{s}_1,\ldots,\boldsymbol{s}_n) \in \tilde{p}(n,\boldsymbol{i})} \binom{\boldsymbol{i}}{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n} q_{1,\boldsymbol{s}_1}(y_1) \cdots q_{n,\boldsymbol{s}_n}(y_n)$$

and compute its explicit expression.

iv) In the explicit expression of the polynomial $h_{\boldsymbol{i}}(y_1, \ldots, y_n)$ as carried out at the previous step *iii)*, replace the occurrences of the products $y_1^{l(\Lambda)} \cdots y_n^{l(\tilde{\Lambda})}$ with $f_{(l(\Lambda),\ldots,l(\tilde{\Lambda}))}$.

Step *iii)* is performed by the `joint` function. This function is not directly accessible in the package, as defined locally in the `MFB` function. The `joint` function realizes a recursive pair matching: each coefficient $g_{1,\Lambda}$ of $q_{1,\boldsymbol{s}_1}(y_1)$ is matched with each coefficient $g_{2,\tilde{\Lambda}}$ of $q_{2,\boldsymbol{s}_2}(y_2)$, then each paired coefficient $g_{1,\Lambda} g_{2,\tilde{\Lambda}}$ is matched with each coefficient g_{3,Λ^*} of $q_{3,\boldsymbol{s}_3}(y_3)$ and so on. Step *iv)* consists of multiplying each coefficient found at step *iii)* with $f_{\boldsymbol{t}}$, where $\boldsymbol{t}$ is the multi-index whose j-th component gives how many times $g_{j,\cdot}$ appears in this coefficient. See the following example.

Example 20: Suppose $n = m = 2$ and $\boldsymbol{i} = (1,1)$. To get $h_{(1,1)}$ in (37) run

```
> MFB(c(1,1),2)
[1] f[1,1]g1[0,1]g2[1,0] + f[1,1]g1[1,0]g2[0,1] + f[2,0]g1[0,1]g1[1,0] +
f[1,0]g1[1,1] + f[0,2]g2[0,1]g2[1,0] + f[0,1]g2[1,1]
```

Taking into account (4), in the previous output `f[i,j]` corresponds to $f_{(i,j)}$ as well as `g1[i,j]` and `g2[i,j]` correspond to $g_{1;(i,j)}$ and $g_{2;(i,j)}$ respectively for $i, j = 0, 1, 2$. Note that `g1[1,1]` is multiplied with `f[1,0]` as there is one occurrence of `g1` and no occurrence of `g2`. In the same way, `g1[1,0]g1[0,1]` is multiplied with `f[2,0]` as there are two occurrences of `g1` and no occurrence of `g2` and `g1[1,0]g2[0,1]` is multiplied with `f[1,1]` as there is one occurrence of `g1` and one occurrence of `g2` and so on. Compare the previous output with the one obtained in `Maple` running

diff(f(g1(x1,x2),g2(x1,x2),x1,x2):

$$
\begin{aligned}
&D_{2,2}(f)(g1(x1,x2),g2(x1,x2))\left(\tfrac{\partial}{\partial x1}g2(x1,x2)\right)\left(\tfrac{\partial}{\partial x2}g2(x1,x2)\right)\\
&+D_{1,2}(f)(g1(x1,x2),g2(x1,x2))\left(\tfrac{\partial}{\partial x2}g1(x1,x2)\right)\left(\tfrac{\partial}{\partial x1}g2(x1,x2)\right)\\
&+D_{1,2}(f)(g1(x1,x2),g2(x1,x2))\left(\tfrac{\partial}{\partial x1}g1(x1,x2))\right)\left(\tfrac{\partial}{\partial x2}g2(x1,x2)\right)\\
&+D_{1,1}(f)(g1(x1,x2),g2(x1,x2))\left(\tfrac{\partial}{\partial x1}g1(x1,x2))\right)\left(\tfrac{\partial}{\partial x2}g1(x1,x2))\right)\\
&+D_2(f)(g1(x1,x2),g2(x1,x2))\left(\tfrac{\partial^2}{\partial x2\partial x1}g2(x1,x2)\right)\\
&+D_1(f)(g1(x1,x2),g2(x1,x2))\left(\tfrac{\partial^2}{\partial x2\partial x1}g1(x1,x2)\right)
\end{aligned}
$$

where $D_1(f)$ denotes $\partial f(x_1,x_2)/\partial x_1$, $D_2(f)$ denotes $\partial f(x_1,x_2)/\partial x_2$ and similarly

$$D_{1,1}(f) \leftarrow \frac{\partial^2 f(x_1,x_2)}{\partial x_1^2},\ D_{2,2}(f) \leftarrow \frac{\partial^2 f(x_1,x_2)}{\partial x_2^2},\ D_{1,2}(f) \leftarrow \frac{\partial^2 f(x_1,x_2)}{\partial x_1 \partial x_2}.$$

The `eMFB` function evaluates the multivariate Faà di Bruno's formula (37) when the coefficients of the formal power series f and $g_1,\ldots,g_n$ in (4) are substituted with numerical values.

Example 21: To evaluate the output of Example 20 for some numerical values of the coefficients, run

```
> cfVal<-"f[0,1]=2, f[0,2]=5, f[1,0]=13, f[1,1]=-4, f[2,0]=0"
> cgVal<-"g1[0,1]=-2.1, g1[1,0]=2,g1[1,1]=3.1,g2[0,1]=5,g2[1,0]=0,g2[1,1]=6.1"
> cVal<-paste0(cfVal,",",cgVal)
> e_MFB(c(1,1),2,cVal)
[1] 12.5
```

The polynomial families discussed in the previous sections are generated using the `MFB` function. Indeed, the generalized (complete exponential) Bell polynomials in (28) are coefficients of the following formal power series

$$H(y_1,\ldots,y_n;z) = 1 + \sum_{|\boldsymbol{i}|>0} h_{\boldsymbol{i}}(y_1,\ldots,y_n)\frac{\boldsymbol{z}^{\boldsymbol{i}}}{\boldsymbol{i}!} = \exp\left[\sum_{i=1}^{n} y_i(g_i(z)-1)\right], \tag{38}$$

which turns to be a composition (3), with $f(x_1,\ldots,x_n) = \exp(x_1y_1+\cdots+x_ny_n)$ and $f_{\boldsymbol{t}} = y_1^{t_1}\cdots y_n^{t_n}$ for $\boldsymbol{t}\in\mathbb{N}_0^n$. In this case, $y_1,\ldots,y_n$ play the role of indeterminates. The $\boldsymbol{i}$-th coefficient $h_{\boldsymbol{i}}(y_1,\ldots,y_n)$ - output of the `GCBellPol` function - is obtained from the multivariate Faà di Bruno's formula (37) dealing with $y_1,\ldots,y_n$ as they were constants. When $\{g_1(z),\ldots,g_n(z)\}$ are the same formal power series $g(z)$, the formal power series $H(y_1,\ldots,y_n;z)$ in (38) reduces to

$$H(y_1,\ldots,y_n;z) = \exp\left[(y_1+\cdots+y_n)(g(z)-1)\right] \tag{39}$$

with coefficients $\tilde{h}_{\boldsymbol{i}}(y_1,\ldots,y_n)$ as given in the previous section.

If $n=1$ then $H(y_1,\ldots,y_n;z)$ reduces to the composition $\exp\left[y(g(z)-1)\right]$ whose coefficients are the polynomials given in (30). More in general the coefficients of $f(g(z)-1)$ are

$$h_{\boldsymbol{i}} = \sum_{\Lambda\vdash\boldsymbol{i}} d_\Lambda f_{l(\Lambda)} g_\Lambda \tag{40}$$

where the sum is over all the partitions $\Lambda = (\lambda_1^{r_1},\lambda_2^{r_2},\ldots)\vdash\boldsymbol{i}$, d_Λ is given in (19) and $g_\Lambda = g_{\lambda_1}^{r_1}g_{\lambda_2}^{r_2}\cdots$. If also $m=1$, then $h_{\boldsymbol{i}}$ in (40) reduces to

$$h_i = \sum_{\lambda\vdash i} d_\lambda f_{l(\lambda)} g_\lambda \tag{41}$$

where the sum is over all the partitions $\lambda = (1^{r_1},2^{r_2},\ldots)\vdash i$, d_λ is given in (16) and $g_\lambda = g_1^{r_1}g_2^{r_2}\cdots$. Formula (41) corresponds to the univariate Faà di Bruno's formula and gives the i-th coefficient of $f(g(z)-1)$ with

$$f(x) = 1 + \sum_{j\geq 1} f_j\frac{x^j}{j!} \qquad \text{and} \qquad g(z) = 1 + \sum_{s\geq 1} g_s\frac{z^s}{s!}.$$

Example 22: To get h_5 in (41) run

```
> MFB(c(5), 1)
[1] f[5]g[1]^5 + 10f[4]g[1]^3g[2] + 15f[3]g[1]g[2]^2 + 10f[3]g[1]^2g[3] +
10f[2]g[2]g[3] + 5f[2]g[1]g[4] + f[1]g[5]
```

For instance, the i-th general partition polynomial in (33) is generated using the MFB function: in such a case the univariate Faà di Bruno's formula (41) is generated with $\{y_s\}$ in place of $\{g_s\}$ and $\{a_j\}$ in place of $\{f_j\}$.

Examples of how to generate polynomials not included in the kStatistics package

In the following we give some suggestions on how to use the R codes of the kStatistics package to generate additional polynomial families.

The pPart function is an example of how to use the univariate Faà di Bruno's formula and a symbolic strategy different from those presented so far. Indeed the pPart function generates the so-called partition polynomial $F_i(y)$ of degree i, whose coefficients are the number of partitions of i with j parts for $j = 1, \ldots, i$ (Boyer and Goh 2008). The partition polynomial $F_i(y)$ is obtained from the univariate Faà di Bruno's formula (41) setting

$$f_j = 1/i! \qquad \text{and} \qquad g_s^{r_s} = (s!)^{r_s} r_s! y^{r_s} \tag{42}$$

for $s = 1, \ldots, i-j+1, j = 1, \ldots, i$ and $r_s = 1, \ldots, i$. Note the symbolic substitution of $g_s^{r_s}$ with the powers y^{r_s}.

Example 23: To get $F_5(y)$ run

```
> pPart(5)
[1] y^5 + y^4 + 2y^3 + 2y^2 + y
```

Note that $F_5(y)$ is obtained from the output of Example 22 using (42).

Example 24: The following code shows how to evaluate $F_{11}(y)$ in $y = 7$.

```
> s<-pPart(11)         # generate the partition polynomial of degree 11
> s<-paste0("1",s)     # add the coefficient to the first term
> s<-gsub(" y","1y",s) # replace the variable y without coefficient
> s<-gsub("y", "*7",s) # assign y = 7
> eval(parse(text=s))  # evaluation of the expression
[1] 3.476775e+182
```

We give a further example on how to generate a polynomial family not introduced so far but still coming from (41) for suitable choices of $\{f_j\}$ and $\{g_s\}$. Consider the elementary symmetric polynomials in the indeterminates $y_1, \ldots, y_n$

$$e_i(y_1, \ldots, y_n) = \begin{cases} \displaystyle\sum_{1 \le j_1 < \cdots < j_i \le n} y_{j_1} \cdots y_{j_i}, & 1 \le i \le n, \\ 0, & i > n. \end{cases} \tag{43}$$

A well-known result (Charalambides 2002) states that these polynomials can be expressed in terms of the power sum symmetric polynomials (14) in the same indeterminates $y_1, \ldots, y_n$, using the general partition polynomials (34), that is

$$e_i = \frac{(-1)^i}{i!} G_i(1, \ldots, 1; -p_1, -1!p_2, -2!p_3, \ldots, -(i-1)!p_i) \tag{44}$$

for $i = 1, \ldots, n$. The following e2p function expresses the i-th elementary symmetric polynomial e_i in terms of the power sum symmetric polynomials $p_1, \ldots, p_i$, using (44) and the MFB function.

```
> e2p <-  function(n=0){
+     v<-MFB(n,1);                    # Call the MFB Function
+     v<-MFB2Set( v );                # Expression to vector
+     for (j in 1:length(v)) {
+         # ----- read -----------[ fix block ]----------------------#
+         c <- as.character(v[[j]][2]);    # coefficient
+         x <- v[[j]][3];                  # variable
+         i <- v[[j]][4];                  # subscript
+         k <- strtoi(v[[j]][5]);          # power
+         # ----- change -------------------------------------------#
+         if (x=="f") {
+             c<-paste0(c,"*( (-1)^",n,")");
```

```
+             x<-"";
+             i<-"";
+         }
+         else if (x=="g") {
+             c<-paste0(c,"*((-factorial(",strtoi(i)-1,"))^",k,")");
+             x<-paste0("(p",i,ifelse(k>1,paste0("^",k),""),")");
+             i<-"";k<-1;
+         }
+         # ----- write ---------[ fix block ]----------------------#
+         v[[j]][2] <- c;
+         v[[j]][3] <- x;
+         v[[j]][4] <- i;
+         v[[j]][5] <- k;
+         # -------------------------------------------------------#
+     }
+     noquote(paste0("1/",factorial(n),"( ",Set2expr(v), " )"));
+ }
```

This function starts by initializing the vector `v` with (41) by means of the `MFB` function. There is a first code snippet `[fix block]` for extracting a set with the coefficients, variables, indexes and powers of `v` by means of the `MFB2Set` function. This first code snippet should not be changed whatever polynomial family we are generating. The second code snippet `change` includes instructions that can be changed according to the expressions of the coefficients $\{f_j\}$ and $\{g_s\}$ in (41). To get (44), we set $f_j = 1$ and $g_s = -(s-1)!p_s$. Once these coefficients have been changed, the last code snippet `[fix block]` updates the vector `v`. The `Set2expr` function assembles the final expression.

Example 25: To get e_4 in (44) run

```
> e2p(4)
[1] 1/24( (p1^4) - 6(p1^2)(p2) + 3(p2^2) + 8(p1)(p3) - 6(p4) )
```

7 Concluding remarks

We have developed the **kStatistics** package with the aim to generate univariate and multivariate k-statistics/polykays, togheter with the multivariate Faà di Bruno's formula and various user-friendly functions related to this formula. The paper briefly introduces the combinatorial tools involved in the package and presents, in detail, the core function of the package which generates multi-index partitions. We emphasize that the algorithms presented here have been designed with the aid of the umbral calculus, even if we did not mentioned this method in the paper.

One of the main applications we have dealt with is the generation and evaluation of various families of polynomials: from generalized complete Bell polynomials to general partition polynomials, from partial Bell polynomials to complete Bell polynomials. Numerical sequences obtained from the Bell polynomials can also be generated.

All these utilities intend to make the **kStatistics** package a useful tool not only for statisticians but also for users who need to work with families of polynomials usually available in symbolic software or tables. Indeed, we have provided examples on how to generate polynomial families not included in the package but which can still be recovered using the Faà di Bruno's formula and suitable strategies, both numerical and symbolic. Following this approach, also the estimations of joint cumulants or products of joint cumulants is one further example of symbolic strategy coming from the multivariate Faà di Bruno's formula.

Future works consist in expanding the **kStatistics** package by including extensions of the multivariate Faà di Bruno's formula, as addressed in Bernardini, Natalini, and Ricci (2005) and references therein, aiming to manage nested compositions, as the `BellY` function in the Wolfram Language and System does. Moreover, further procedures can be inserted relied on symbolic strategies not apparently related to the multivariate Faà di Bruno's formula but referable to this formula, as for example the central Bell polynomials (Kim, Kim, and Jang 2019).

The results in this paper were obtained using the **kStatistics** 2.1.1 package. The package is currently available with a general public license (GPL) from the Comprehensive `R` Archive Network.

8 Acknowledgements

The authors would like to thank the reviewers for their constructive feedback.

References

Arnqvist, Natalya Pya, Vassilly Voinov, Rashid Makarov, and Yevgeniy Voinov. 2021. *Nilde: Nonnegative Integer Solutions of Linear Diophantine Equations with Applications*. https://CRAN.R-project.org/package=nilde.

Bernardini, A., P. Natalini, and P. E. Ricci. 2005. "Multidimensional Bell Polynomials of Higher Order." *Comput. Math. Appl.* 50 (10-12): 1697–1708. https://doi.org/10.1016/j.camwa.2005.05.008.

Boyer, Robert P., and William M. Y. Goh. 2008. "Partition Polynomials: Asymptotics and Zeros." In *Tapas in Experimental Mathematics*, 457:99–111. Contemp. Math. Amer. Math. Soc., Providence, RI. https://doi.org/10.1090/conm/457/08904.

Brown, James Ward. 1979. "On Multivariable Sheffer Sequences." *J. Math. Anal. Appl.* 69 (2): 398–410. https://doi.org/10.1016/0022-247X(79)90151-3.

Chacón, José E., and Tarn Duong. 2015. "Efficient Recursive Algorithms for Functionals Based on Higher Order Derivatives of the Multivariate Gaussian Density." *Statistics and Computing* 25 (5): 959–74. https://doi.org/10.1007/s11222-014-9465-1.

Charalambides, Charalambos A. 2002. *Enumerative Combinatorics*. CRC Press Series on Discrete Mathematics and Its Applications. Chapman & Hall/CRC, Boca Raton, FL.

Clausen, Andrew, and Serguei Sokol. 2020. *Deriv: R-Based Symbolic Differentiation*. https://CRAN.R-project.org/package=Deriv.

Constantine, G. M., and T. H. Savits. 1996. "A Multivariate Faà Di Bruno Formula with Applications." *Transactions of the American Mathematical Society* 348 (2): 503–20. https://doi.org/10.1090/S0002-9947-96-01501-2.

De Leeuw, J. 2012. "Multivariate Cumulates in R." https://escholarship.org/uc/item/1fw1h53c.

Di Nardo, E. 2016a. "On Multivariable Cumulant Polynomial Sequences with Applications." *Journal of Algebraic Statistics* 7 (1): 72–89. https://doi.org/10.18409/jas.v7i1.49.

———. 2016b. "On Photon Statistics Parametrized by a Non-Central Wishart Random Matrix." *Journal of Statistical Planning and Inference* 169: 1–12. https://doi.org/10.1016/j.jspi.2015.07.002.

Di Nardo, E., and G. Guarino. 2019. *Unbiased Estimators for Cumulant Products*. https://cran.r-project.org/web/packages/kStatistics/index.html.

———. 2021. *kStatistics: Unbiased Estimators for Cumulant Products and Faa Di Bruno's Formula*. https://CRAN.R-project.org/package=kStatistics.

Di Nardo, E., G. Guarino, and D. Senato. 2011. "A New Algorithm for Computing the Multivariate Faà Di Bruno's Formula" 217 (13): 6286–95. https://doi.org/10.1016/j.amc.2011.01.001.

Di Nardo, Elvira. 2011. "Symbolic Calculus in Mathematical Statistics: A Review." *Séminaire Lotharingien de Combinatoire* 67: Art. B67a, 72. https://www.mat.univie.ac.at/~slc/wpapers/s67dinardo.pdf.

Di Nardo, E., M. Marena, and P. Semeraro. 2020. "On Non-Linear Dependence of Multivariate Subordinated Lévy Processes." *Statistics & Probability Letters* 166: 108870–77. https://doi.org/10.1016/j.spl.2020.108870.

Di Nardo, E., and I. Oliva. 2011. "On a Symbolic Version of Multivariate Lévy Processes." *American Institute of Physics Conference Proceedings* 1389 (1): 345–48. https://doi.org/10.1063/1.3636735.

Dimitrakopoulos, Roussos, Hussein Mustapha, and Erwan Gloaguen. 2010. "High-Order Statistics of Spatial Random Fields: Exploring Spatial Cumulants for Modeling Complex Non-Gaussian and Non-Linear Phenomena." *Mathematical Geosciences* 42 (1): 65–99. https://doi.org/10.1007/s11004-009-9258-9.

Domino, Krzysztof, Piotr Gawron, and Łukasz Pawela. 2018. "Efficient Computation of Higher-Order Cumulant Tensors." *SIAM J. Sci. Comput.* 40 (3): A1590–610. https://doi.org/10.1137/17M1149365.

Ferreira, Pedro G, Joao Magueijo, and Joseph Silk. 1997. "Cumulants as Non-Gaussian Qualifiers." *Physical Review D* 56 (8): 4592. https://doi.org/10.1103/PhysRevD.56.4592.

Geng, Min, Huaqing Liang, and Junwu Wang. 2011. "Research on Methods of Higher-Order Statistics for Phase Difference Detection and Frequency Estimation." In *2011 4th International Congress on Image and Signal Processing*, 4:2189–93. https://doi.org/10.1109/CISP.2011.6100593.

Giannakis, G. B. 1987. "Cumulants: A Powerful Tool in Signal Processing." *Proceedings of the IEEE* 75 (9): 1333–34. https://doi.org/10.1109/PROC.1987.13884.

Guarino, G., D. Senato, and E. Di Nardo. 2009. "Fast Maple Algorithms for *k*-Statistics, Polykays and Their Multivariate Generalization." https://www.maplesoft.com/applications/view.aspx?SID=33041.

Hankin, R. K. S. 2006. "Additive Integer Partitions in r." *Journal of Statistical Software, Code Snippets* 16.

Hardy, Michael. 2006. "Combinatorics of Partial Derivatives." *Electronic Journal of Combinatorics* 13 (1): Research Paper 1, 13. http://www.combinatorics.org/Volume_13/Abstracts/v13i1r1.html.

Hernández Encinas, L., and J. Muñoz Masqué. 2003. "A Short Proof of the Generalized Faà Di Bruno's Formula." *Applied Mathematics Letters. An International Journal of Rapid Publication* 16 (6): 975–79. https://doi.org/10.1016/S0893-9659(03)90026-7.

Jammalamadaka, S. Rao, T. Subba Rao, and György Terdik. 2006. "Higher Order Cumulants of Random Vectors and Applications to Statistical Inference and Time Series." *Sankhyā. The Indian Journal of Statistics* 68 (2): 326–56. https://www.jstor.org/stable/25053499.

Joanes, Derrick N, and Christine A Gill. 1998. "Comparing Measures of Sample Skewness and Kurtosis." *Journal of the Royal Statistical Society: Series D (The Statistician)* 47 (1): 183–89.

Kim, Taekyun, Dae San Kim, and Gwan-Woo Jang. 2019. "On Central Complete and Incomplete Bell Polynomials i." *Symmetry* 11 (2). https://www.mdpi.com/2073-8994/11/2/288.

Leipnik, Roy B., and Charles E. M. Pearce. 2007. "The Multivariate Faà Di Bruno Formula and Multivariate Taylor Expansions with Explicit Integral Remainder Term." *The ANZIAM Journal. The Australian & New Zealand Industrial and Applied Mathematics Journal* 48 (3): 327–41. https://doi.org/10.1017/S1446181100003527.

Ma, Tsoy-Wo. 2009. "Higher Chain Formula Proved by Combinatorics." *Electronic Journal of Combinatorics* 16 (1): N21, 7. https://doi.org/10.37236/259.

McCullagh, Peter. 1987. *Tensor Methods in Statistics*. Monographs on Statistics and Applied Probability. Chapman & Hall, London.

Mihoubi, Miloud. 2008. "Polynômes Multivariés de Bell Et Polynômes de Type Binomial." PhD thesis, L'Université des Sciences et de la Technologie Houari Boumedienne.

Mishkov, Rumen L. 2000. "Generalization of the Formula of Faà Di Bruno for a Composite Function with a Vector Argument." *International Journal of Mathematics and Mathematical Sciences* 24 (7): 481–91. https://doi.org/10.1155/S0161171200002970.

Nguwi, Jiang Yu, Guillaume Penent, and Nicolas Privault. 2022. "A Deep Branching Solver for Fully Nonlinear Partial Differential Equations." arXiv. https://arxiv.org/abs/2203.03234.

Oualla, Hicham, Rachid Fateh, Anouar Darif, Said Safi, Mathieu Pouliquen, and Miloud Frikel. 2021. "Channel Identification Based on Cumulants, Binary Measurements, and Kernels." *Systems* 9 (2). https://doi.org/10.3390/systems9020046.

Peccati, Giovanni, and Murad S. Taqqu. 2011. "Combinatorial Expressions of Cumulants and Moments." In *Wiener Chaos: Moments, Cumulants and Diagrams.*, edited by Milano Springer, 1:201–13. Bocconi & Springer Series.

Privault, Nicolas. 2021. "Recursive Computation of the Hawkes Cumulants." *Statistics and Probability Letters*, 109161. https://doi.org/10.1016/j.spl.2021.109161.

Rao, T. Subba, and W. K. Wong. 1999. "Some Contributions to Multivariate Nonlinear Time Series and to Bilinear Models." In *Asymptotics, Nonparametrics, and Time Series*, 158:259–94. Statist. Textbooks Monogr. Dekker, New York.

Reiner, David L. 1976. "Multivariate Sequences of Binomial Type." *Studies in Applied Mathematics* 57 (2): 119–33. https://doi.org/10.1002/sapm1977572119.

Robson, D. S. 1957. "Applications of Multivariate Polykays to the Theory of Unbiased Ratio-Type Estimation." *Journal of the American Statistical Association* 52 (280): 511–22. https://www.tandfonline.com/doi/abs/10.1080/01621459.1957.10501407.

Roman, Steven. 1984. *The Umbral Calculus*. Vol. 111. Pure and Applied Mathematics. Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York.

Rose, C., and M. D. Smith. 2002. *Mathematical Statistics with Mathematica®*. Springer Texts in Statistics. Springer-Verlag, New York. https://doi.org/10.1007/978-1-4612-2072-5.

Savits, Thomas H. 2006. "Some Statistical Applications of Faà Di Bruno." *Journal of Multivariate Analysis* 97 (10): 2131–40. https://doi.org/10.1016/j.jmva.2006.03.001.

Shabat, A. B., and M. Kh. Efendiev. 2017. "On Applications of Faà-Di-Bruno Formula." *Ufa Mathematical Journal* 9 (3): 131–36. https://doi.org/10.1007/s12572-017-0181-x.

Shrivastava, HSP. 2002. "Multiindex Multivariable Hermite Polynomials." *Math. Comput. Appl.* 7 (2): 139–49. https://doi.org/10.3390/mca7020139.

Smith, Kevin D. 2020. "A Tutorial on Multivariate *k*-Statistics and Their Computation." arXiv. https://arxiv.org/abs/2005.08373.

Smith, Kevin D., Saber Jafarpour, Ananthram Swami, and Francesco Bullo. 2022. "Topology Inference with Multivariate Cumulants: The Möbius Inference Algorithm." *IEEE/ACM Transactions on Networking*, 1–15. https://doi.org/10.1109/TNET.2022.3164336.

Staude, Benjamin, Stefan Rotter, and Sonja Grün. 2010. "CuBIC: Cumulant Based Inference of Higher-Order Correlations in Massively Parallel Spike Trains." *Journal of Computational Neuroscience* 29 (1-2): 327–50. https://doi.org/10.1007/s10827-009-0195-x.

Stuart, Alan, and J. Keith Ord. 1994. *Kendall's Advanced Theory of Statistics. Vol. 1*. Sixth. Edward Arnold, London; copublished in the Americas by Halsted Press [John Wiley & Sons, Inc.], New York.

Voinov, Vassilly, and Natalya Pya Arnqvist. 2017. "R-Software for Additive Partitioning of Positive Integers." *Mathematical Journal* 17 (1): 69–76. http://www.math.kz/media/journal/journal2018-05-1574083.pdf.

Withers, Christopher S., and Saralees Nadarajah. 2010. "Multivariate Bell Polynomials." *International Journal of Computer Mathematics* 87 (11): 2607–11. https://doi.org/10.1080/00207160802702418.

Elvira Di Nardo
University of Turin
Department of Mathematics "G.Peano"
Via Carlo Alberto 10, 10123 Turin (Italy)
https://www.elviradinardo.it
ORCiD: 0000-0003-3447-9155
elvira.dinardo@unito.it

Giuseppe Guarino
Università Cattolica del Sacro Cuore
Largo Agostino Gemelli 8, 00168, Rome (Italy)
giuseppe.guarino@rete.basilicata.it