Measurement and Research Department Reports 91-1

A New Algorithm For Computing Elementary Symmetric Functions And Their First And Second Derivatives

N.D. Verhelst
N.H. Veldhuijzen

Cito

Measurement and Research Department Reports 91-1

A New Algorithm For Computing Elementary Symmetric Functions And Their First And Second Derivatives

N.D. Verhelst
N.H. Veldhuijzen

Cito
Arnhem, 1991

Abstract

Elementary symmetric functions are the coefficients of the powers of x in the expression $\prod_{i=1}^{k}(1+\varepsilon_i x)$. Functions where all εs are positive, are considered here. These functions, with their first and second derivatives, must be evaluated in, for example, estimating so-called item response models. The evaluation is a notoriously difficult thing to do. As the estimation procedures are iterative, a time-efficient algorithm is called for. In this report, a new algorithm is developed and compared with the well-known sum algorithm. It is shown that the new algorithm reduces computing time to a fair amount, and uses hardly more core memory than is needed to store the results wanted.

Keywords: conditional maximum likelihood estimation, divide-and-conquer procedure, numerical stability.

Introduction

In this report a new algorithm is described for computing elementary symmetric functions. Such functions are used extensively in item response models, such as the Rasch model (Fischer, 1974; Rasch, 1960; Verhelst, Glas & Van der Sluis, 1984). Specifically, these functions, together with their first and second derivatives, are needed in deriving conditional maximum likelihood estimators and their standard errors in these models. As the number of functions needed grows rapidly with the number of arguments, an algorithm is called for which cuts down computing time, without significantly increasing core requirements.

To set the stage, consider the one-parameter logistic model for dichotomous items, the so-called Rasch model. It is characterized by the item response function f_{vi}:

$$f_{vi} = Prob(A_{vi} = 1 \,|\, \theta_v, \varepsilon_i) = \frac{\theta_v \varepsilon_i}{1+\theta_v \varepsilon_i} \qquad \begin{cases} \theta_v,\ \varepsilon_i > 0 \\ i = 1, \cdots, k \\ v = 1, \cdots, N \end{cases}$$

where A_{vi} is a binary random variable, and θ_v and ε_i are subject and item parameters, respectively. This function gives the probability of a '1' response by subject v with ability parameter θ_v, to item i with easiness parameter ε_i. In estimating the item parameters, the subject parameters can be considered as nuisance parameters. Conditional Maximum Likelihood estimation (Andersen, 1973) maximizes the likelihood function, conditional on minimal sufficient statistics of the nuisance parameters. It can be shown (Fischer, 1974) that the likelihood equations for conditionally estimating the item parameters in the Rasch model can be written as

$$a_{.i} = \sum_{r=0}^{k} t_r \frac{\varepsilon_i \gamma_{r-1}^{(i)}(\underline{\varepsilon})}{\gamma_r(\underline{\varepsilon})} \qquad i = 1, \cdots, k$$

where $a_{.i}$ is the number of correct responses to item i, t_r is the number of subjects having precisely r correct responses, $\underline{\varepsilon}$ is the vector of item parameters, $\gamma_r(\underline{\varepsilon})$ is the elementary symmetric function of order r on the k arguments $\underline{\varepsilon}$, to be defined below, and $\gamma_{r-1}^{(i)}(\underline{\varepsilon})$ is the elementary symmetric

function of order r-1 on the k-1 arguments $\varepsilon_1 \ldots, \varepsilon_{i-1}, \varepsilon_{i+1}, \ldots, \varepsilon_k$. This last function is the first derivative of $\gamma_r(\underline{\varepsilon})$ with respect to ε_i.
To have not only estimates of parameters, but also of their standard errors, second derivatives of the elementary symmetric functions (which are elementary symmetric functions themselves) are needed. Computation of elementary symmetric functions poses a notoriously difficult problem. The difficulties have to do not only with numerical instability, but also with very demanding core use and computing time.

The algorithm to be presented focuses on the computation of second derivatives of elementary symmetric functions. It turns out that first derivatives, and the function values themselves, are got very easily as a byproduct of the computations. In this article, emphasis is given to computing time, but core requirements will be considered, too.

Elementary Symmetric Functions

The elementary symmetric function of order s, defined on k real arguments $\varepsilon_1, \ldots, \varepsilon_k$, denoted by $\gamma_s(\varepsilon_1, \ldots, \varepsilon_k)$, is defined as:

$$\gamma_s(\varepsilon_1, \ldots, \varepsilon_k) = \sum_{\sum x_i = s} \prod_i \varepsilon_i^{x_i}, \qquad x_i \in \{0,1\},\ s=0,1,\ldots,k$$

In the applications considered here, all arguments are strictly positive.
A deceptively simple algorithm to evaluate this function is:
- determine all combinations of s out of k arguments;
- multiply all arguments of each combination;
- sum the products.

Of course, the deception lies in the first step; this is a very heavy combinatorial task. In case all arguments are positive, a numerically stable algorithm to evaluate the functions is the so-called sum algorithm (Gustafsson, 1980). This algorithm uses the recursion

$$\gamma_s(\varepsilon_1, \ldots, \varepsilon_i) = \varepsilon_i . \gamma_{s-1}(\varepsilon_1, \ldots, \varepsilon_{i-1}) + \gamma_s(\varepsilon_1, \ldots, \varepsilon_{i-1}) \qquad (1)$$

where $s = 1(1)k$ and $i = s(1)k$. We define $\gamma_s(\varepsilon_1, \ldots, \varepsilon_k) = 0$ for $s < 0$ and $s > k$. The sum algorithm boils down to computing an upper triangular matrix T, whose last column contains the function values of interest:

$$t_{ij} = \begin{cases} \sum_{c=1}^{j} \varepsilon_c = t_{i,j-1} + \varepsilon_j & (i=1) \\ t_{i,j-1} + \varepsilon_j t_{i-1,j-1} & (1 < i < j) \\ \prod_{c=1}^{j} \varepsilon_c = t_{j-1,j-1}\varepsilon_j & (i=j) \end{cases}$$

Each element of the matrix is computed by (1), which implies one addition and one multiplication. In this article, the term *unit* is used to denote a combination of one addition and one multiplication. Since the costs of the computations do not depend on the values of the arguments $\varepsilon_1, \ldots, \varepsilon_k$, but only on their number, k, we will denote the elementary symmetric function of order s, defined on k arguments, by $\gamma_s(k)$.
The upper triangular matrix mentioned above contains $\frac{1}{2}k(k+1)$ elements. The computation of each element costs one unit, so, computing all elementary symmetric functions for k arguments requires $\frac{1}{2}k(k+1)$ units.

The first derivative of $0\gamma_q(j)$ with respect to ε_i is equal to the elementary symmetric function of order $q-1$ for all arguments except ε_i; this function is denoted by $\gamma^{(i)}_{(q-1)}(j-1)$. Leaving out one argument at a time, computation of all first derivatives with the sum algorithm requires $\frac{1}{2}k^2(k-1)$ units. In the same way, it can be shown that computation of all second derivatives with the sum algorithm takes $\frac{1}{2}k(k-1)\{\frac{1}{2}(k-1)(k-2)\}$ units. So, the total costs M_0 of computing the elementary symmetric functions with their first and second derivatives by means of the sum algorithm are:

$$M_0 = \tfrac{1}{2}k(k+1) + k\{\tfrac{1}{2}(0k-1)k\} + \{\tfrac{1}{2}k(k-1)\}\{\tfrac{1}{2}(k-2)(k-1)\} = \tfrac{1}{4}k^4 - \tfrac{1}{2}k^3 + (5/4)k^2$$

This number for the total costs will be used as a benchmark in evaluating the costs of our algorithm.

A New Algorithm

The new algorithm is based on a fairly simple generalization of the sum algorithm; it inherits from it its numerical stability. Its main advantage obtains by computing intermediate results once and storing them for later use. The algorithm consists of three parts:

- the construction and combination of building blocks;
- the derivation of intermediate results from the building blocks;
- the derivation of the final result from the intermediate results.

The final result is the set of elementary symmetric functions $\{\gamma_s(k) : s = 1(1)k\}$, the k first derivatives $\{\gamma_s^{(i)}(k-1) ; s = 1(1)(k-1),\ i = 1(1)k\}$, and the $\frac{1}{2}k(k-1)$ second derivatives $\{\gamma_s^{(i,j)}(k-2) ;$ $s = 1(1)(k-2),\ i,j = 1(1)k,\ i < j\}$.

The construction and combination of building blocks

The algorithm starts with partitioning the k arguments $\varepsilon_1, \ldots, \varepsilon_k$ in p groups, each group containing at least two arguments. It is immaterial whether the number of arguments in each group is the same; however, in order to keep the computation of the costs simple, it will be assumed that each group contains exactly m arguments, that is, $k = mp$.
For each of the p groups, the m elementary symmetric functions and their $m(m-1)$ first derivatives and $\frac{1}{2}m(m-1)(m-2)$ second derivatives are computed with the sum algorithm. From the derivation of M_0 above it is seen that these computations require:

$$S = p[\tfrac{1}{4}m^4 - \tfrac{1}{2}m^3 + (5/4)m^2]$$

units. Each set of elementary symmetric functions (excluding first and second derivatives) pertaining to one group of m arguments, is called a building block. The question which values of p and m are optimal with respect to costs will be taken up later.

The sum algorithm given by (1) can easily be generalized to the combination of elementary symmetric functions, that is, building blocks. Consider $\gamma_i(k_1)$ and $\gamma_j(k_2)$, for $i = 0(1)k_1$ and $j = 0(1)k_2$, elementary symmetric functions with k_1 and k_2 mutually exclusive arguments, respectively. Then the following recursion can be used to combine building blocks (Fischer, 1981):

$$\gamma_g(k_1+k_2) = \sum_{i=a}^{b} \gamma_i(k_1) \cdot \gamma_{g-i\varepsilon}(k_2), \quad g = 0(1)(k_1+k_2), \tag{2}$$

where $a = \max(0, g-k_2)$ and $b = \min(g, k_1)$. Each product $\gamma_i(k_1) \cdot \gamma_{g-1}(k_2)$ is one term of $\gamma_g(k_1+k_2)$. As $i = 0(1)k_1$ and $g-i = k_2(-1)0$, $(1+k_1)(1+k_2)$ terms must be added; so the computation of all $\gamma_g(k_1+k_2)$, $g = 0(1)(k_1+k_2)$, requires $(1+k_1)(1+k_2)$ units.

The construction of intermediate results from the building blocks

In this section, the word "algorithm" means only the part of the whole algorithm in which intermediate results are constructed from the building blocks. It concerns the combination of elementary symmetric functions for all p groups of m arguments. Some notation is needed to describe this part of the algorithm.

Let the generic symbol G(1) denote a building block, that is, some set of elementary symmetric functions of one group of m arguments. Let the symbol $\oplus$ denote the combination of two sets of gamma functions as defined by (2), then (2) can be written symbolically as G(2) = G(1) $\oplus$ G(1); thus the argument of G is the number of building blocks G is based upon. It is clear that $\oplus$ is a commutative and associative operation. Hence G(n) = G(a) $\oplus$ G(b), where $a+b=n$ for all positive integers a and b.

> The reader is invited to have Figure 1 of the Appendix at hand when reading the following passages. That Figure depicts the algorithm for the case $p=7$.

The algorithm has p steps. In step i, $p-i$ sets G($p-2$) and one G($p-1$) are constructed. None of these sets contain the i-th group of arguments. Moreover, in each set G($p-2$), one other group of arguments is excluded, viz. one of the groups $(i+1)(1)p$.

For $i>2$, the iteration step i starts with a G($i-2$). This set is the combination of the building blocks based on groups 1 through $i-2$. For the cases $i=1$ and $i=2$, dummy sets G(-1) and G(0) are defined. While these sets formally contain $m+1$ elements, they are actually empty. To handle these dummy sets, the definition of $\oplus$ is extended as follows: G(-1) $\oplus$ G(0) = G(0) and G(0) $\oplus$ G(1) = G(1). (Using these dummy sets as if they contain $m+1$ elements, implies that the costs of our total algorithm will be somewhat overestimated; this overestimation is negligible, however.) So, the first iteration starts with G(-1); all other iterations start with a G($i-2$) which has been computed in the step before the current one.

In step i, $[(p-2)-(i+1-3)] = (p-i)$ combinations are needed to get the first G($p-2$). In the j-th combination, a G($i+j-3$) and a G(1) are joined. (The intermediate results G($i+j-3$), $1<i+j<p$, are kept, because they are used presently. Therefore, these results are denoted by G*($i+j-3$).) Hence, to come from G($i+1-3$) to the first G($p-2$) requires

$$C_{1i} = \sum_{j=1}^{p-i} (1+m)\,[1+(i+j-3)\,m]$$

units. As there are no combinations to perform when $i = p$ (the starting set is a $G(p\text{-}2)$), all combinations leading to the first $G(p\text{-}2)$ in every step, will cost:

units.

For $i > 1$, the $G(i\text{-}2)$ started with is the result of the first combination of step $i\text{-}1$ (see the arrows in Figure 1 of the Appendix). The first combination of step i concerns the arguments of the $(i\text{-}1)$-th group of arguments; the following combinations involve the groups $p, p\text{-}1, \ldots, i+2$ successively. So the result $G(p\text{-}2)$, reached in the i-th step of the algorithm, where $i < p$, contains the elementary symmetric functions of all groups except the groups i and $i+1$.

To summarize, the cost C_1 comprises all combinations up to and including the first box in every column in Figure 1, except the last column.

In step i, where $i < p\text{-}1$, there are $p\text{-}i\text{-}1$ intermediate results $G^*(i+j-2)$, $j = 1(1)(p\text{-}i\text{-}1)$. Each of these is combined with a $G[(p\text{-}2)-(i+j-2)] \equiv G(p\text{-}i\text{-}j)$ to get the other $G(p\text{-}2)$. These $p\text{-}i\text{-}1$ combinations require:

$$C_{2i} = \sum_{j=1}^{p-i-1} [1+(i+j-2)\,m]\,[1+(p-i-j)\,m]$$

units. So, the total costs of combinations of this kind are:

$$C_2 = \sum_{i=1}^{p-2} C_{2i}$$

units.

To combine the intermediate results $G^*(i+j-2)$ with a $G(p\text{-}i\text{-}j)$, for $j = 1(1)(p\text{-}i\text{-}1)$, these latter sets must be computed first. Now, if $j = p\text{-}i\text{-}1$, a building block $G(1)$ can be used. So the sets $G(p\text{-}i\text{-}j)$ are computed in the reverse order: $j = (p\text{-}i\text{-}1)(-1)1$, and every combination adds a $G(1)$ to the result of the foregoing one. (In Figure 1, these combinations appear under the first box in each column, and do not have a box around them.) In step i these combinations obviously require:

$$C_{3i} = \sum_{j=1}^{p-i-2} (1+m)\,(1+jm)$$

units. For all steps involving this kind of combinations ($i < p-2$), the costs are:

$$C_3 = \sum_{i=1}^{p-3} C_{3i}$$

units.

The sum $C_1 + C_2 + C_3$ represents the costs of computing all sets $G(p-2)$. (For $i = 3$ the allocation of the costs to the combinations is shown in Figure 2. The meaning of cost B is explained below.) Now it will be shown that each $G(p-2)$ is computed just once.

The $G(1)$ started with in step i ($i < p-1$) concerns the arguments of group $i+1$, and this set is augmented successively with a $G(1)$ based on the arguments of the next group up to the $(p-1)$th. For each j in the range $(i+2)\,(1)\,(p-1)$, the set $G(p-i-j)$ is combined with the kept result $G*(i+j+2)$, giving rise to a set $G(p-2)$ which is not based on the arguments in groups i and j. For each i, $p-i-1$ sets $G(p-2)$ are computed this way. Together with the first $G(p-2)$ based on the arguments of the groups i and $i+1$ (see above) the i-step of the algorithm yields all sets $G(p-2)$ with the arguments of group i excluded; moreover, in each of these $G(p-2)$ one other group j ($j = (i+1)\,(1)\,p$) is excluded. So, all $G(p-2)$ together contain exactly all function values with every pair of groups of arguments excluded once.

To compute the p sets $G(p-1)$, in each step i, $i = 1\,(1)\,(p-1)$, of the algorithm a $G(1)$ is combined with the $G(p-2)$ computed last. That $G(1)$ is based on the p-th group of arguments; the $G(p-2)$ used on entering the last step will be combined with the $G(1)$ based on the $(p-1)$-th group of arguments. In this way, all $G(p-1)$ are computed. These computations require the same number of unit costs in each step, the costs of combining a $G(p-2)$ and a $G(1)$. So, computation of all $G(p-1)$ requires

$$B = p(1+m)\,[1+(p-2)\,m]$$

units.

In the last step of the algorithm, when i equals p, the $G(1)$ which is based on the p-th group of arguments will be combined with the computed $G(p-1)$. The result is the one and only $G(p)$, the set of all elementary symmetric functions. This last result requires

$$A = (1+m)\,[1+(p-1)m]$$

units.

The construction of the final result from the intermediate results

The final result is the set of all first and second derivatives of the elementary symmetric functions, as well as the functions themselves. The final result can be thought of as a symmetric $k \times k$-matrix, each cell of which contains an ordered set of elementary symmetric functions $\{\gamma^{(i,j)}(k-2)\}$, except the diagonal cells; these contain sets $\{\gamma^{(i)}(k-1)\}$. This matrix has the following block structure. The i-th diagonal element consists of the first derivative with respect to the i-th argument ε_i. As explained earlier, the first derivative of some $\gamma_q(k)$ with respect to ε_i is equal to $\gamma_{q-1}^{(i)}(k-1)$, the elementary symmetric function of order k-1 for all arguments except ε_i. Around the diagonal there are p block matrices. Each of these contain the second derivatives with respect to two arguments, ε_i and ε_j, from the same group of m arguments. The remaining block matrices, which do not touch the diagonal, contain the second derivatives with respect to two arguments, ε_i and ε_j, which are taken from two different groups of m arguments.

Computation of the diagonal elements

The i-th diagonal element is computed by combining the function value $\gamma^{(i)}(m-1)$ with a $G(p-1)$; this requires $[1+(m-1)][1+(p-1)m] = m(1+k-m)$ units. As there are k diagonal elements, computing the diagonal requires

$$D = km(1+k-m)$$

units.

Computation of the diagonal block matrices

Only the off-diagonal elements of the diagonal block matrices must be considered here. Take some function $\gamma_q^{(i,j)}(m-2)$. We combine the function value $\gamma_q^{(i,j)}(m-2)$, where i and j point to arguments in the same group of arguments, with a $G(p-1)$. This requires $[1+(m-2)][1+(p-1)m] = (m-1)(1+k-m)$ units. As there are $\frac{1}{2}m(m-1)$ off-diagonal elements in a diagonal block matrix, computation of such a matrix requires $\frac{1}{2}m(m-1)^2(1+k-m)$ units. So, the computation of all p block diagonal matrices requires

$$E = \tfrac{1}{2}pm(m-1)^2(1+k-m) = \tfrac{1}{2}k(m-1)^2(1+k-m)$$

units.

Computation of the off-diagonal block matrices

The computation consists of four steps:

- Pick a $\gamma_q^{(i)}(m-1)$ and combine it with a $G(p-2)$. This takes $[1+(m-1)][1+(p-2)m] = m(1+k-2m)$ units.
- Combine the result of the first step with a $\gamma_q^{(j)}(m-1)$. Then the (i,j)-th cell of the block has been determined, at a cost of $[1+(m-1)][1+(p-2)m+m-1] = m(k-m)$ units. To fill one complete row of the block matrix requires $m^2(k-m)$ units.
- To compute all rows of the block matrix, the foregoing two steps must be repeated. This takes $m[m(1+k-2m)+m^2(k-m)]$ units.
- As there are $\tfrac{1}{2}p(p-1)$ block matrices to fill, a total of

$$F = \tfrac{1}{2}p(p-1)m[m(1+k-2m)+m^2(k-m)]$$

units is required.

Results

For a selection of values of k and m, the costs of our algorithm are expressed as a percentage of the units required by the sum algorithm, that is, as a percentage of the value of M_0. The results are collected in Table 1.

TABLE 1

The costs of the new algorithm, expressed as a percentage of the costs M_0 of the sum algorithm, for several values of k and m. The lowest value in each column is underscored.

m\k	18	24	36	54	72	96	144
2	45.3	36.6	27.5	21.3	18.1	15.7	13.3
3	44.9	35.7	25.8	18.8	15.1	12.3	9.5
4	-	39.0	27.9	-	15.7	12.4	9.1
6	58.4	47.4	34.4	24.5	19.1	14.8	10.4
8	-	55.3	-	-	23.1	17.9	12.4
9	70.9	-	44.1	32.0	25.1	-	13.5
12	-	68.5	52.1	-	30.9	24.2	16.9
16	-	-	-	-	-	30.2	21.3
18	-	-	66.3	51.0	41.4	-	23.4
24	-	-	-	-	50.2	40.8	29.5

In Table 1, only those cells are filled where m divides k. It is clear that for a large range of values of k, a group size $m=3$ or 4 will do. For $k=96$, our algorithm is about eight times faster than the sum algorithm. It is illuminating to see the distribution of unit costs over the three parts of the algorithm. In Table 2 this is done, for several k and the associated fastest m. The costs for the building blocks consist of S, those for the intermediate results consist of $C_1+C_2+C_3+B+A$, and those for the final result consist of D+E+F.

TABLE 2

Splitting up of the costs (in percentages) for the three parts of the algorithm for several combinations of k and m.

k	18	24	36	54	72	96	144
m	3	3	3	3	3	3	4
building blocks	1.0	.5	.2	.1	<.1	<.1	<.1
intermediate	12.0	14.4	18.7	24.0	28.6	34.0	25.0
end result	87.0	85.1	81.1	75.9	71.3	66.0	75.0

Clearly, computing the final result is the most expensive part of the algorithm. Especially the parts E and F, which obtain when computing second derivatives, are rather expensive. If second derivatives are not needed, the present algorithm is not efficient. In this case, one can do better by using an efficient algorithm which generates the intermediate results. In Figure 3 of the Appendix, such an algorithm is depicted for $p=7$. Essentially, it

corresponds to one step of our main algorithm; therefore, we call it the reduced algorithm. In Table 3, the cost of the reduced algorithm is compared to that of the sum algorithm, similarly reduced to the symmetric functions and their first derivatives.

TABLE 3

The costs of the reduced algorithm, expressed as percentage of the costs of the reduced sum algorithm, for several values of k and m. The lowest value in each column is underscored.

m\k	18	24	36	54	72	96	144
2	52.7	44.2	35.4	29.3	26.2	23.8	21.5
3	52.4	43.0	33.0	26.0	22.4	19.6	16.8
4	-	45.8	34.2	-	21.7	18.5	15.2
6	67.5	54.5	39.9	29.2	23.6	19.3	14.8
8	-	64.1	-	-	27.0	21.6	15.9
9	84.0	-	50.5	36.5	28.9	-	16.7
12	-	81.6	61.0	-	34.9	27.4	19.6
16	-	-	-	-	-	33.7	23.8
18	-	-	79.3	59.1	46.9	-	26.0
24	-	-	-	-	58.2	46.1	32.6

The gain in computing speed requires a larger amount of memory for the computer implementation. We need memory to store all G(1)'s and all sets G(p-1) and G(p-2) which are needed in one single loop of the algorithm. However, we need no more scratch memory than when using the sum algorithm. Let n be the size of the memory cell needed to store one number. Then a G(1) takes $n(1+m)$ memory cells; a G(p-1) takes $n[1+(p-1)]$ cells, and a G(p-2) takes $n[1+(p-2)]$ cells. So, in total we need:

$$M(m) = np(1+m) + np[1+(p-1)] + \tfrac{1}{2}np(p-1)[1+(p-2)] + n(1+k)$$

memory cells. The last term is the scratch memory required. For $m = 3$, $M(3) = nk(k^2-2k+27)/18$, which rapidly exceeds all available core memory with growing k. An elegant solution for this problem is to store each G(p-2) on background memory as soon as it has been computed. Then, for $m = 3$, not more than $nk(k+2)/3$ memory cells are needed. The within group derivatives are stored successively in the same memory location, as they are needed only once in computing the final result.

Conclusion

In the preceding sections, an algorithm has been developed to compute the elementary symmetric functions together with their first and second derivatives. The basis of the algorithm consists in partitioning the arguments in p groups, to compute the symmetric functions within the groups, and to combine these within group functions in some sort of half-products, the sets G(•). If only the elementary functions are needed, it is easy to show that the fastest partition is given for $p=1$, i.e., the algorithm coincides with the sum algorithm. However, if first derivatives are required along with the functions, a reduction in computing time of about 80% with respect to the sum algorithm is realized if the number of arguments is of the order of one hundred. If second derivatives are required also, the percentage reduction is almost 90 for k=100. The computational costs are derived only for the case of p groups of equal size. In the case of unequal group sizes, the total cost is dependent on the specific order in which the combinations of the sets G(•) are taken, so that it is possible that the maximal reduction rates are higher than the ones reported in Tables 1 and 3. Some experimental trials however revealed that the gain will only be marginal. So the search for an optimal partition has been abandoned.

As to the accuracy of the results, it should be noticed that the only operations performed by the algorithm are additions and multiplications of positive numbers. Moreover, the number of operations required to arrive at a specific function value is the same as in the sum-algorithm. Hence, the error analysis presented in Verhelst, Glas and van der Sluis (1984) on the sum algorithm is valid for our algorithm too. Specifically, this means that the number of decimal digits lost through rounding errors is at most $\log_{10}(2k)$, irrespective of the partition used.

The price to be paid for this gain in time is a considerable increase in memory as compared to the sum algorithm: if a number occupies n memory cells, the sum algorithm needs no more than $2n(1+k)$ cells to compute the symmetric functions of k arguments, while the present algorithm needs M(k) cells, a third degree polynomial of k. This number M(k) can not be reduced by storing each element of the final result successively in the same storage location; the M(k) cells are needed as working storage anyway.

References

Andersen, E.B. (1973). *Conditional Inference and Models for Measuring.* København, Mentalhygienisk Forskningsinstitut.

Fischer, G.H. (1974). *Einführing in die Theorie Psychologischer Tests.* Bern, Huber Verlag.

Fischer, G.H. (1981). On the Existence and Uniqueness of Maximum-Likelihood Estimates in the Rasch Model. *Psychometrika*, 46, 59-77.

Gustafsson, J.E. (1980). Testing and obtaining fit of data to the Rasch model *British Journal of Mathematical and Statistical Psychology*, 33, 205-233.

Rasch, G. (1960). *Probabilistic Models for Some Intelligence and Attainment Tests.* København, Danmarks Pædagogiske Institut.

Verhelst, N.D, C.A.W Glas & A. van der Sluis (1984). Estimation problems in the Rasch model: the basic symmetric functions. *Computational Statistics Quarterly*, 1, 245-262.

Appendix

Figure 1 depicts the algorithm for $p=7$. Each number denotes a building block, that is, a G(1). The symbol '+' denotes a combination or concatenation. It is used here instead of the symbol $\oplus$ for typographical reasons. The symbol '*' denotes a G*(•). The G(p-2) which must be stored for a while are put in a box. An iteration is finished at the '~'-line. Each box directly above the wavy line is a G(p-1), except when $i=p=7$, where the box directly above the wavy line is the G(p) itself. For each i, the result of the first combination is used to enter iteration $i+1$. This is shown by the arrows. The numbers '-1' and '0' for $i=1$ denote the dummy sets. For typographical reasons, the '-1' and '0' are not shown in the combinations that involve real sets.

Figure 2 shows the third iteration step from Figure 1. It is annotated with indications of the costs required for each part of the iteration.

Figure 3 depicts the reduced algorithm. Its legends are the same as for Figure 1.

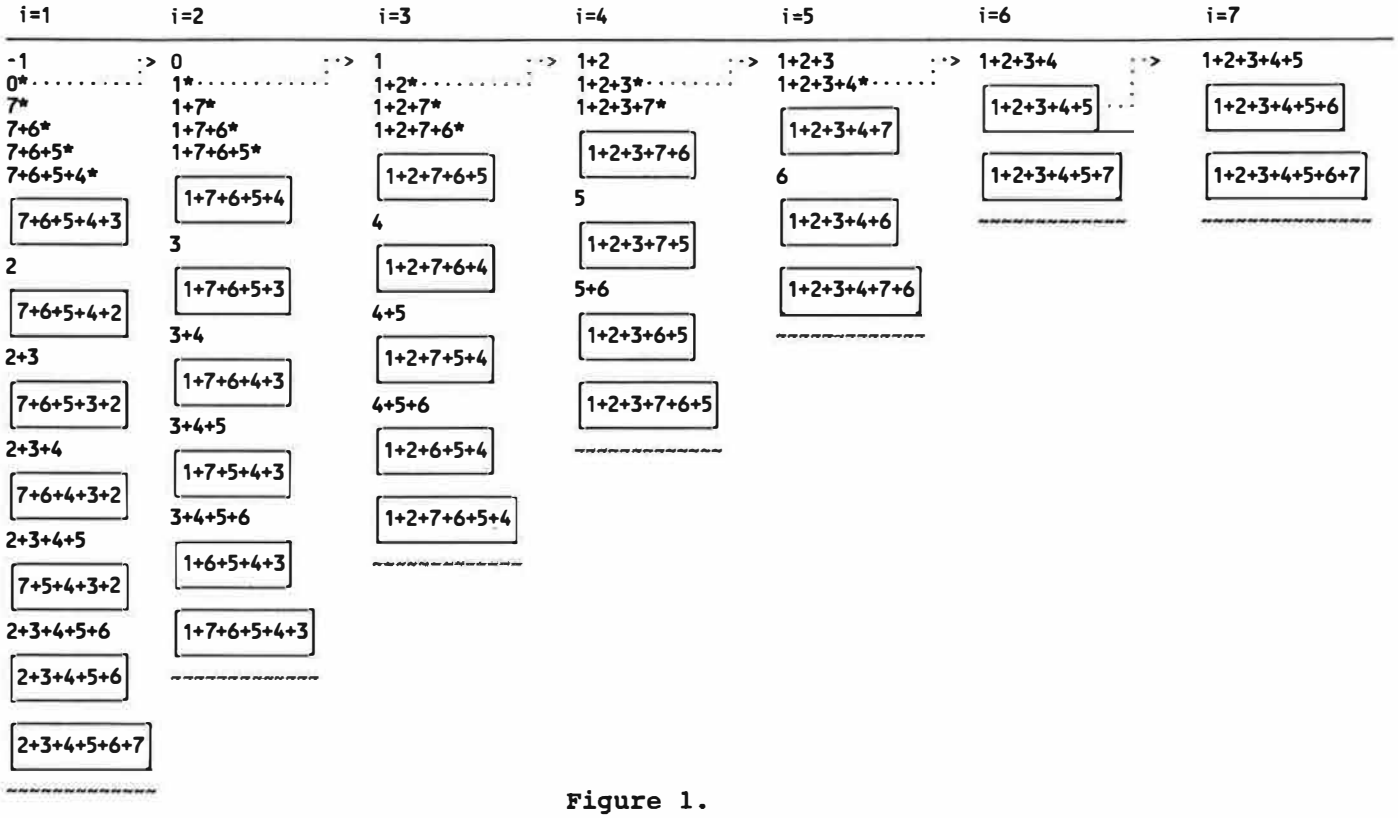

Figure 1.
The algorithm for $p = 7$

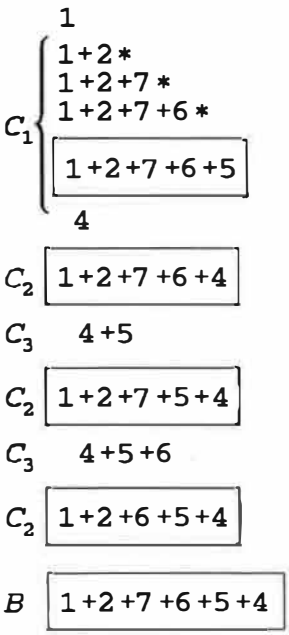

Figure 2.
The third iteration step with cost functions attached

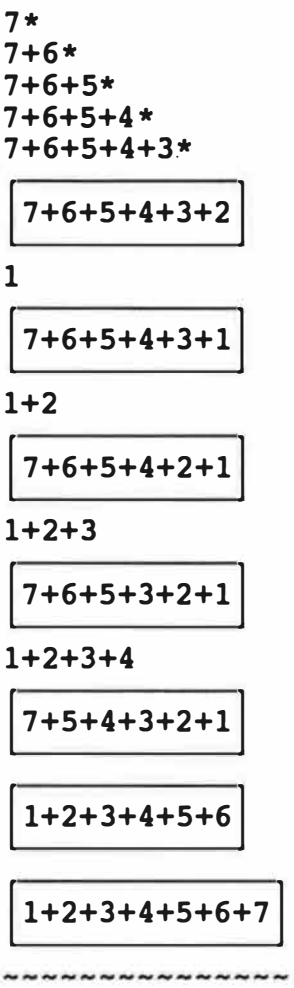

Figure 3.
The reduced algorithm for $p = 7$