EasyChair Preprint

№ 4

Computation of Some Integer Sequences in Maple

W.L. Fan, David J. Jeffrey and Erik Postma

November 20, 2017

Computation of Some Integer Sequences in Maple

W.L. Fan[1], D.J. Jeffrey[1], Erik Postma[2]

[1] Department of Applied Mathematics,
The University of Western Ontario, London, Ontario, Canada
[2] Maplesoft, Waterloo
wfan54@uwo.ca, djeffrey@uwo.ca

Abstract. We consider some integer sequences connected with combinatorial applications. Specifically, we consider Stirling partition and cycle numbers, associated Stirling partition and cycle numbers, and Eulerian numbers of the first and second kinds. We consider their evaluation in different contexts. One context is the calculation of a single value based on single input arguments. A more common context, however, is the calculation of a sequence of values. We compare strategies for both. Where possible, we compare with existing Maple implementations.

1 Introduction

For extended discussions of Stirling and Eulerian numbers, we refer to [1, 2]. These and similar numbers arise frequently in combinatorial applications, and have therefore been implemented in several computer algebra systems. To date, the standard libraries of most systems have included Stirling numbers, but not *associated* Stirling numbers [3], even though they have found several applications in recent years. For example, they have appeared in series expansions for the Lambert W function [4], and also appeared in one form of Stirling's series for the Gamma function [2]. (Stirling did not define the associated numbers.)

Another feature of many implementations is that the functions expect a single argument, and return a single value. In practice, however, an application will usually require a sequence of values, for example, to provide successive coefficients in a series. The requirement of returning multiple values has already been recognized in some Maple functions, for example, in the implementation of Bernoulli numbers: it accepts a `mode` parameter. To quote from Maple help:

> The mode parameter controls whether or not the `bernoulli` routine computes additional Bernoulli numbers in parallel with the requested one. For example, if your computer has 4 cores, then the command bernoulli(1000, singleton=false) will compute (and store) bernoulli(1002) bernoulli(1004), and bernoulli(1006). Since in practice nearly all computations which use Bernoulli numbers require many of them, and require them in sequence, this results in considerable efficiency gains.

This paper addresses both the computation of single values and of the integer sequences associated with the combinatorial functions under consideration. As a matter of terminology, we shall call a function that accepts a unique argument and returns the corresponding unique result a *singleton* function, and the corresponding operation a singleton computation. In contrast, a function accepting a range (explicit or implicit) of arguments and returning the corresponding list of values will be a sequence function, and the calculation a sequence calculation.

1.1 Definitions of numbers

We collect here the definitions of all numbers considered.

Definition 1. *The r-associated Stirling numbers of the first kind, more briefly Stirling r-cycle numbers, are defined by the generating function*

$$\left(\ln\frac{1}{1-z}-\sum_{j=1}^{r-1}\frac{z^j}{j}\right)^m = m!\sum_{n\geq 0}\begin{bmatrix}n\\m\end{bmatrix}_{\geq r}\frac{z^n}{n!}\ . \tag{1}$$

Remark 1. The number $\left[\begin{smallmatrix}n\\m\end{smallmatrix}\right]_{\geq r}$ gives the number of permutations of n distinct objects into m cycles, each cycle having a minimum cardinality r [2, p 256].

Definition 2. *The r-associated Stirling numbers of the second kind, called more briefly here Stirling r-partition numbers, are defined, using Karamata–Knuth notation, by the generating function*

$$\left(e^z-\sum_{j=0}^{r-1}\frac{z^j}{j!}\right)^m = m!\sum_{n\geq 0}\left\{\begin{matrix}n\\m\end{matrix}\right\}_{\geq r}\frac{z^n}{n!}\ . \tag{2}$$

Remark 2. The number $\left\{\begin{smallmatrix}n\\m\end{smallmatrix}\right\}_{\geq r}$ gives the number of partitions of a set of size n into m subsets, each subset having a minimum cardinality of r [2, 5, 6].

Definition 3. *The Eulerian numbers of the first kind $\left\langle\begin{smallmatrix}n\\k\end{smallmatrix}\right\rangle$ are defined as the number of permutations $\pi_1\pi_2\ldots\pi_n$ of $\{1,2,\ldots n\}$ that have k ascents, i.e. k places where $\pi_j<\pi_{j+1}$.*

Definition 4. *The Eulerian numbers of the second kind $\left\langle\!\left\langle\begin{smallmatrix}n\\k\end{smallmatrix}\right\rangle\!\right\rangle$ are defined as the number of permutations of the multiset $\{1,1,2,2,\ldots,n,n\}$ for which all numbers between the two occurrences of every m, with $1\leq m\leq n$, are greater than m, for each permutation having k ascents, i.e. k places where $\pi_j<\pi_{j+1}$.*

Remark 3. Note that m is not an argument. For example, given the multiset $\{112233\}$, permutations such as 122133 or 123321 are permitted, but 211233 is not. Amongst these permitted permutations, we count those with k ascents.

Nomenclature: In [1], the numbers $\left\langle\begin{smallmatrix}n\\k\end{smallmatrix}\right\rangle$ are called simply 'Eulerian numbers', while the numbers $\left\langle\!\left\langle\begin{smallmatrix}n\\k\end{smallmatrix}\right\rangle\!\right\rangle$ are called 'second-order Eulerian numbers'.

2 Stirling partition numbers

The Maple 2017 implementation is a singleton function, denoted `stirling2` in the `combinat` package. It uses the formula

$$\left\{ {n \atop m} \right\} = \frac{1}{m!} \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} k^n \ . \tag{3}$$

For the singleton computation, Table1 shows that the times[3] are much less using (3). In this table, we compared the Maple function `stirling2` with the method given below using the recurrence relation (7). Timings for a sequence calculation, however, given in Table 2, show the new method is more efficient.

n	m	recurrence	`stirling2`
100	50	0.002	0.002
200	100	0.010	0.003
500	250	0.079	0.007
4000	200	2.700	0.009
5000	250	6.940	0.013

Table 1. Timings (sec) for generating a singleton Stirling Partition number. The time using (7) is compared with the Maple `stirling2` function.

2.1 Sequence calculation

Given n, m, we wish to compute all Stirling partition numbers $\left\{ {i \atop j} \right\}$ such that $i \le n$ and $j \le m$. We use the recurrence relation

$$\left\{ {i \atop j} \right\} = j \left\{ {i-1 \atop j} \right\} + \left\{ {i-1 \atop j-1} \right\} , \tag{4}$$

subject to the boundary conditions

$$\left\{ {j \atop j} \right\} = 1 \ , \quad \text{and} \quad \left\{ {i \atop 1} \right\} = 1 \ . \tag{5}$$

Since $\left\{ {i \atop j} \right\} = 0$ for $j > i$ (see Fig. 1), we define a matrix P which will not store these zeros.

$$P_{ij} = \left\{ {i+j-1 \atop j} \right\} . \tag{6}$$

Then the recurrence relation becomes

$$P(i,j) = j\, P(i-1,j) + P(i,j-1) \ . \tag{7}$$

The boundary conditions then become, respectively, $P(1,j) = \left\{ {j \atop j} \right\} = 1$, and $P(i,1) = 1$.

[3] Product placement: times found using an Intel i7 in a Lenovo Ultrabook.

Timings Table 2 shows the timings for filling matrices of various sizes with integer sequences of Stirling partition numbers. The recurrence relation (7) is compared with creating each entry through a call to Maple's `stirling2`. Filling the square matrix $P(n,n)$ actually calculates all partition numbers $\left\{ {i \atop j} \right\}$ with $i \leq 2n$ and $j \leq n$. This is done for timing convenience, and the matrix can be reshaped for other applications.

n	m	recurrence	`stirling2`
100	100	0.031	7.87
200	200	0.093	69.2
300	300	0.265	259
400	400	0.437	667
500	500	0.843	1450

Table 2. Timings (sec) for generating sequences of Stirling Partition numbers. The time using (7) compared with Maple `stirling2` function.

3 Stirling cycle numbers

We consider the computation of $\left[{n \atop m} \right]$, implemented in Maple 2017 as `stirling1` in the `combinat` package. The computational method used by `stirling1` is based on Stirling's original definition of his numbers:

$$x^{\underline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k \ . \tag{8}$$

For given n, `stirling1` constructs the product on the left, which is then collected in powers of x, so that by equating the coefficients of x^k, all numbers $\left[{n \atop k} \right]$ for $1 \leq k \leq n$ are determined and stored. Thus, a future call to $\left[{n \atop m} \right]$ with $1 \leq m \leq n$ will be returned by table lookup, but a future call with a different n will initiate a new computation. It is interesting that although the interface appears to offer the user only a singleton computation, in fact a particular integer sequence has been computed silently.

3.1 Singleton computation

A singleton computation returns the value of a function for a single pair of input arguments. We implement the known recurrence relation

$$\begin{bmatrix} n \\ m \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ m \end{bmatrix} + \begin{bmatrix} n-1 \\ m-1 \end{bmatrix} \ , \tag{9}$$

subject to boundary conditions

$$\begin{bmatrix} m \\ m \end{bmatrix} = 1 \ , \ \text{for } m \geq 1 \ , \tag{10}$$

$$\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)! \ . \tag{11}$$

We define the vector

$$u_j^{(i)} = \begin{bmatrix} i+j-1 \\ j \end{bmatrix} .$$

In Fig. 1, we see that for fixed i, $u_j^{(i)}$ describes numbers along the ith diagonal line, counting from the left. The recurrence relation (9) can be written in terms of u as

$$u_j^{(i)} = (i+j-2)u_j^{(i-1)} + u_{j-1}^{(i)} \ ,$$

with $u_1^{(i)} = (i-1)!$. We note that once $u_j^{(i-1)}$ is used, it does not need to be stored further, so we can overwrite storage. Our iteration scheme is thus (Maple notation for the ith element of a vector is $u[i]$)

$$u[j] = (i+j-2)u[j] + u[j-1] \ .$$

Therefore, we initialize $u[1] = u_1^{(i)} = \begin{bmatrix} i \\ 1 \end{bmatrix} = (i-1)!$ and fill in diagonal lines successively.

Complexity The aim of this subsection is to gain insight into the best ways to test the implementations, by identifying the worse cases for the methods. A full bit complexity is beyond the scope of this paper, and will require more work on estimates for the sizes of Stirling numbers. As pointed out by Wilf [9], the available estimates are for $\begin{bmatrix} n \\ k \end{bmatrix}$ when k is fixed and $n \to \infty$, whereas the present algorithms require knowledge of the opposite case.

In order to calculate the number $\begin{bmatrix} n \\ m \end{bmatrix}$, a vector of length m must be recomputed (overwritten) $n-m$ times. Each iteration requires one multiplication and 3 additions. Therefore the complexity is $m(n-m)$. We can therefore expect that the worst case for the method will be $m = n/2$.

Since Maple's approach and the present one calculate different sets of numbers, a direct comparison is not very meaningful, and so we simply make a brief comparison between one-time calculations. Notice that in Table 3, the times taken by `stirling1` are approximately independent of m as expected.

3.2 A finite sum

For completeness, we mention that a singleton cycle number can be found from a finite sum, as was done for a singleton partition number. We have

$$\begin{bmatrix} n \\ m \end{bmatrix} = \sum_{j=0}^{n-m} (-1)^{n-k+j} \binom{n-1+j}{n-k+j} \binom{2n-k}{n-k-j} \left\{ {n-k+j \atop j} \right\} . \tag{12}$$

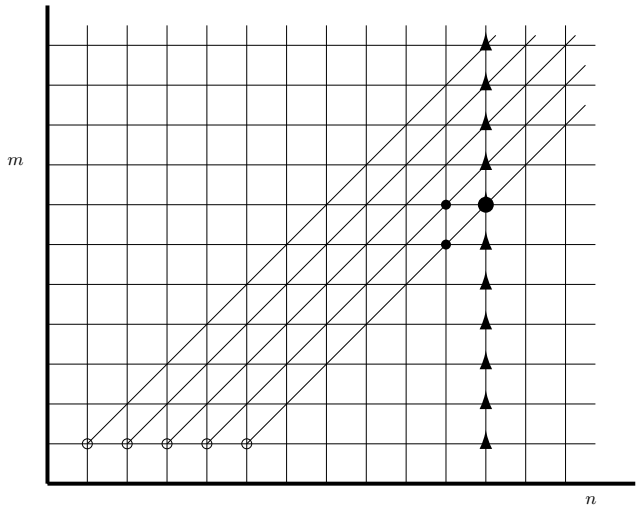

Fig. 1. Scheme for calculating singleton Stirling cycle $\left[{n \atop m}\right]$ or partition numbers $\left\{{n \atop m}\right\}$. The computation proceeds from left to right and bottom to top. At each stage only the numbers on one diagonal need to be stored in the vector $u_j^{(i)}$ which is progressively overwritten. The open circles show the base of each successive loop. The black filled circles show the recurrence relation used. The larger circle is calculated from the two smaller ones. The triangles line show the points computed by one call to `stirling1`.

n	m	`stirling1`	Present scheme
300	150	0.023	0.035
400	200	0.063	0.052
400	20	0.062	0.011
1000	500	0.612	0.491
2000	500	4.92	3.00

Table 3. Times for a single call to Maple's `stirling1` and the present singleton computation. Timings (sec) based on 10 trials, with memory being cleared before each call.

Combining this with (3), we can express a cycle number as a double sum. This, however, is too slow to warrant further consideration.

3.3 Sequence calculation

The method used above for partition numbers can be readily adapted for cycle numbers. Given n, m, we compute all Stirling cycle numbers $\left[{i \atop j}\right]$ such that $i \le n$ and $j \le m$. We use the recurrence relation

$$\left[{i \atop j}\right] = (i-1)\left[{i-1 \atop j}\right] + \left[{i-1 \atop j-1}\right] , \tag{13}$$

subject to the boundary conditions

$$\left[{j \atop j}\right] = 1 , \quad \text{and} \quad \left[{i \atop 1}\right] = (i-1)! . \tag{14}$$

Since $\left[{i \atop j}\right] = 0$ for $j > i$ (see Fig. 1), we define a matrix C which will not store these zeros.

$$C_{ij} = \left[{i+j-1 \atop j}\right] . \tag{15}$$

Then the boundary conditions are $C(1, j) = \left[{j \atop j}\right] = 1$, and $C(i, 1) = (i-1)!$. The recurrence relation becomes

$$C(i, j) = (i + j - 2)\, C(i-1, j) + C(i, j-1) . \tag{16}$$

Timings Table 4 shows the timing for filling matrices of various sizes with integer sequences of Stirling cycle numbers. The recurrence relation (16) is compared with creating each entry through a call to Maple's `stirling1`. Filling the square matrix $C(n, n)$ actually calculated all cycle numbers $\left[{i \atop j}\right]$ with $i \le 2n$. This is done for timing purposes, and the matrix can be reshaped for other applications. The comparison is to compute the same numbers using the sequence calculation function `stirling1`. Larger values of (n, m) are not tabulated because a bug in Maple 2016 (and earlier) caused larger arguments to fail. This will be corrected in Maple 2017.

4 Associated Stirling numbers

There are no known analogues of (3) or (8) for the associated Stirling numbers for $r \ge 2$; hence we must use either the generating functions (2) and (1), or the following recurrence relations.

$$\left\{{n+1 \atop k}\right\}_{\ge r} = k\left\{{n \atop k}\right\}_{\ge r} + \binom{n}{r-1}\left\{{n-r+1 \atop k-1}\right\}_{\ge r} , \tag{17}$$

$$\left[{n+1 \atop k}\right]_{\ge r} = n\left[{n \atop k}\right]_{\ge r} + n^{\underline{r-1}}\left[{n-r+1 \atop k-1}\right]_{\ge r} . \tag{18}$$

n	m	recurrence	`stirling1`
40	40	0.000	0.842
60	60	0.000	4.446
80	80	0.015	14.414
100	100	0.015	35.037
120	120	0.015	193.004

Table 4. Timings (sec) for generating sequences of Stirling cycle numbers. The time using (16) compared with Maple's `stirling1` function.

Note that $n^{\underline{0}} = 1$. The boundary cases are

$$\left\{ {n \atop 1} \right\}_{\geq r} = 1 \ , \quad n \geq r \ , \tag{19}$$

$$\left[{n \atop 1} \right]_{\geq r} = (n-1)! \ , \quad n \geq r \ , \tag{20}$$

$$\left\{ {kr \atop k} \right\}_{\geq r} = \frac{(rk)!}{(r!)^k \, k!} \ , \quad k \geq 1 \ , \tag{21}$$

$$\left[{kr \atop k} \right]_{\geq r} = \frac{(rk)!}{r^k k!} \ , \quad k \geq 1 \ . \tag{22}$$

4.1 Singleton Stirling 2-partition and 2-cycle

The two computations have the same structure, and can be described in parallel. We choose to implement

$$\left\{ {n \atop m} \right\}_{\geq 2} = m \left\{ {n-1 \atop m} \right\}_{\geq 2} + (n-1) \left\{ {n-2 \atop m-1} \right\}_{\geq 2} \ , \tag{23}$$

$$\left[{n \atop m} \right]_{\geq 2} = (n-1) \left[{n-1 \atop m} \right]_{\geq 2} + (n-1) \left[{n-2 \atop m-1} \right]_{\geq 2} \ . \tag{24}$$

We also have boundary conditions

$$\left[{2n \atop n} \right]_{\geq 2} = \left\{ {2n \atop n} \right\}_{\geq 2} = \frac{(2n)!}{n! 2^n} = (2n-1)!!$$

$$\left[{2n+1 \atop n} \right]_{\geq 2} = 2 \, \frac{(2n+1)!}{3(n-1)! 2^n} = 2 \left\{ {2n+1 \atop n} \right\}_{\geq 2}$$

We define the vector

$$u_j^{(i)} = \left[{i+2j-1 \atop j} \right]_{\geq 2} \ ,$$

and similarly for 2-partition numbers. In Fig. 2, we see that if we fix i, then $u_j^{(i)}$ describes numbers along the ith diagonal line. Now $u_1^{(i)} = i!$ and

$$u_j^{(i)} = (i + 2j - 2)u_j^{(i-1)} + (i + 2j - 2)u_{j-1}^{(i)} \ .$$

We note that once $u_j^{(i-1)}$ is used, it does not need to be stored further, so we can overwrite storage. Our iteration scheme is thus

$$u[j] = (i + 2j - 2)(u[j] + u[j - 1]) \ .$$

For initialization, we can use a special case of (24):

$$\begin{bmatrix} 2j+2 \\ j+1 \end{bmatrix}_{\geq 2} = u_{j+1}^{(1)} = (2j + 1) \begin{bmatrix} 2j \\ j \end{bmatrix}_{\geq 2} = (2j + 1)u_j^{(1)} \ .$$

Therefore, we initialize u to $i = 1$ using $u_j^{(1)} = u[j] = 1$ and fill in one line at a time by fixing i and looping over j. Each j loop starts setting $u_1^{(i)} = i! = iu_1^{(i-1)}$. We then loop over i.

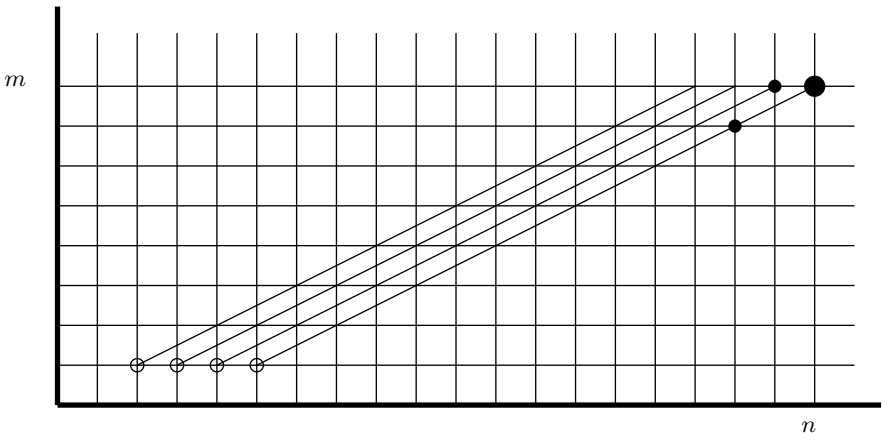

Fig. 2. Calculating 2-partition and 2-cycle numbers. As with the $r = 1$ case, only numbers on one sloping line need to be kept at any stage of the computation. The same convention for illustrating the recurrence relation is used.

4.2 Sequence calculation of 2-partition and 2-cycle numbers

Given n, m, we compute all Stirling 2-partition numbers $\left\{ {i \atop j} \right\}_{\geq 2}$ or 2-cycle numbers $\left[{i \atop j} \right]_{\geq 2}$ such that $i \leq n$ and $j \leq m$. We use the recurrence relations (23) or (24) as

appropriate. Since $\left\{ {i \atop j} \right\}_{\geq 2} = \left[{i \atop j} \right]_{\geq 2} = 0$ for $2j > i$ (see Fig. 2), we define a matrix C which will not store these zeros.

$$C_{ij} = \left[{i+2j-1 \atop j} \right]_{\geq 2} . \tag{25}$$

Then the recurrence relation for 2-partition becomes

$$C(i,j) = j\, C(i-1,j) + (i+2j-2) C(i-1,j-1) \ . \tag{26}$$

The recurrence relation for 2-cycle becomes

$$C(i,j) = (i+2j-2)\, C(i-1,j) + (i+2j-2) C(i-1,j-1) \ . \tag{27}$$

4.3 Singleton Stirling r-partition and r-cycle numbers

From the above discussion of 1-associated and 2-associated numbers, the generalization is clear. We have to implement

$$\left[{n+1 \atop m} \right]_{\geq r} = n \left[{n \atop m} \right]_{\geq r} + n(n-1)(n-2)\ldots(n-r+2) \left[{n-r+1 \atop m-1} \right]_{\geq r} . \tag{28}$$

We define the vector

$$u_j^{(i)} = \left[{i+rj-1 \atop j} \right]_{\geq r} .$$

The generalization of Fig. 2 to one containing lines of slope $1/r$ is not shown. For fixed i, $u_j^{(i)}$ describes numbers along one of the lines, with $u_1^{(i)} = (i+r-2)!$ and

$$u_j^{(i)} = (i+rj-2) u_j^{(i-1)} + (i+rj-2)(i+rj-3)\ldots(i+rj-r) u_{j-1}^{(i)} \ .$$

We note that once $u_j^{(i-1)}$ is used, it does not need to be stored further, so we can overwrite storage. Our iteration scheme is thus

$$u[j] = (i+rj-2) u[j] + (i+rj-2)\ldots(i+rj-r) u[j-1] \ .$$

For initialization, we can use a special case of (28):

$$\begin{aligned} \left[{rj+r \atop j+1} \right]_{\geq r} = u_{j+1}^{(1)} =& (rj+r-1)(rj+r-2)\ldots(rj+1) \left[{rj \atop j} \right]_{\geq r} \\ =& (rj+r-1)(rj+r-2)\ldots(rj+1) u_j^{(1)} \ . \end{aligned}$$

We initialize u using $u_j^{(1)} = u[j] = 1$ and fill in each line by fixing i and looping over j. We start each j loop by setting $u_1^{(i)} = (i+r-2)! = (i+r-2) u_1^{(i-1)}$. We then loop over i.

4.4 Sequence calculation of r-partition and r-cycle numbers

Given n, m, we compute all Stirling r-partition numbers $\left\{ {i \atop j} \right\}_{\geq r}$ or r-cycle numbers $\left[{i \atop j} \right]_{\geq r}$ such that $i \leq n$ and $j \leq m$. The recurrence relation applied here can refer to (17) and (18), which is subject to the boundary conditions

$$\left\{ {1 \atop 1} \right\}_{\geq r} = \left[{1 \atop 1} \right]_{\geq r} = 1 \ . \tag{29}$$

Since $\left\{ {i \atop j} \right\}_{\geq r} = \left[{i \atop j} \right]_{\geq r} = 0$ for $rj > i$, we define a matrix C which will not store these zeros.

$$C_{ij} = \left[{i + rj - 1 \atop j} \right]_{\geq r} . \tag{30}$$

Then the recurrence relation for r-partition becomes

$$C(i,j) = j\, C(i-1,j) + \binom{i + rj - 2}{r-1} C(i - r + 1, j - 1) \ . \tag{31}$$

The recurrence relation for r-cycle becomes

$$C(i,j) = (i+rj-2)\, C(i-1,j) + (i+rj-2)(i+rj-3) \ldots (i+rj-r)\, C(i-r+1, j-1) \tag{32}$$

4.5 Implementation in Maple

In our implementation of Stirling numbers, we provide procedures for users to compute either a singleton Stirling number or a sequence of Stirling numbers. The procedures are

1. StirlingRCycle: to calculate a singleton Stirling r-cycle number.
2. StirlingRCycleMatrix: to calculate a sequence of Stirling r-cycle numbers.
3. StirlingRPartition: to calculate a singleton Stirling r-partition number.
4. StirlingRPartitionMatrix: to calculate a sequence of Stirling r-partition numbers.

Neither Maple nor Mathematica has an implementation with which to compare our programs. Therefore we have programmed the recurrence relations, as well as the generating functions in Maple. In Table 5 below, we compared our new scheme for computing a singleton r-associated Stirling cycle number with using the generating function. The generating function for Stirling r-cycle numbers is:

```
StirRCycleGen := proc(n,k,r) local t, z, p;
      t:=series((ln(1/(1-z)) - add(z^p/p , p=1..r-1))^k, z=0,n+1);
      n!*coeff(t, z, n)/k!;
end proc;
```

m	Singleton scheme	Generating function
2	0.062	2.979
3	0.093	14.461
4	0.109	26.707
5	0.140	41.184
6	0.156	38.797
7	0.171	51.121
8	0.171	45.240
9	0.171	53.055
10	0.171	53.289

Table 5. Timings in seconds of computations of single Stirling r-cycle number. Column headings give the functions used. The numbers tested were $\left[\begin{smallmatrix}1700\\ m\end{smallmatrix}\right]_{\geq 100}$.

Table 5 shows that the singleton scheme is much faster than the generating function for the computation of single r-associated Stirling cycle number. For the computation of a sequence of r-associated Stirling cycle numbers, we compared three methods: (1) a loop calling the singleton function; (2) a loop calling the generating function; (3) the sequence procedure. The results are collected in Tables 6 and 7, and show that the sequence procedure is fastest.

n	Singleton scheme	Generating function
10	0.011	2.402
20	0.024	8.746
30	0.037	18.952
40	0.063	36.477
50	0.771	72.817

Table 6. Timings in seconds of computations of a sequence of r-associated Stirling cycle numbers. Column headings give the functions used. The input argument is n, and the return is an $n \times n$ matrix.

Similar tests were performed for r-associated Stirling partition numbers. The generating function for Stirling r-partition numbers is:

```
StirRPartGen := proc(n,k,r) local t, z, p;
       t:=series((exp(z) - add(z^p/p! , p=0..r-1))^k, z=0, n+1);
       n!*coeff(t, z, n)/k!;
end proc;
```

The test data are collected in Tables 8, 9, 10. Since the pattern is similar to that for cycle numbers, the discussion and tables are abbreviated.

n	Singleton scheme	Sequence scheme
100	7.145	0.015
150	36.411	0.031
200	117.734	0.062
250	295.102	0.124
300	638.474	0.202

Table 7. Timings in seconds of computations of a sequence of r-associated Stirling cycle numbers. Column headings give the functions used. The input argument is n, and the return is an $n \times n$ matrix.

m	Singleton scheme	Generating function
2	0.031	7.129
4	0.062	18.969
6	0.093	29.250
8	0.140	32.276
10	0.156	43.664

Table 8. Timings in seconds of computations of single r-associated Stirling partition number. Column headings give the functions being used. The numbers tested were $\left\{ {1000 \atop m} \right\}_{\geq 18}$.

n	Singleton scheme	Generating function
10	0.020	1.864
20	0.035	8.345
30	0.070	22.047
40	0.144	44.074
50	0.201	79.233

Table 9. Timings in seconds of computations of a sequence of r-associated Stirling partition number. Column headings give the functions being used. The input argument is n, and the return is an $n \times n$ matrix.

n	Singleton scheme	Sequence scheme
100	7.145	0.015
200	117.734	0.062
250	295.102	0.124
300	638.474	0.202

Table 10. Timings in seconds of computations of a sequence of r-associated Stirling partition numbers. Column headings give the functions used. The input argument is n, and the return is an $n \times n$ matrix.

5 A multiple threads approach to sequence calculations

The Maple help for Bernoulli numbers, quoted in the introduction, states that additional values of Bernoulli numbers are calculated in parallel. This section explored ways in which parallel computation could be applied to Stirling numbers. For this, we use the `Threads` package in Maple. When we generate the numbers inside a matrix, instead of filling the matrix row by row and column by column, we fill each diagonal from left to right. Here is the main part in the sequential code to fill Stirling r-cycle numbers in the matrix by diagonal with given input arguments (n, r) where n is the size of matrix.

```
for N from 3 to n do
  for k from 2 to N-1 do
    pd := mul(N-k+r-1-l, l = 1 .. r-1);
    A(N-k+r, k) := pd*A(N-k, k-1)+(N-k+r-2)*A(N-k+r-1, k);
  end do;
end do;
```

According to the recurrence relation, we know that we can divide such diagonal into a left half and right half. So we define two subroutines accordingly.

```
fileft := proc (N, r) local k, Nsplit, pd, l; global A;
Nsplit := floor((1/2)*N+1/2);
for k from 2 to Nsplit do
  pd := mul(N-k+r-1-l, l = 1 .. r-1);
  A(N-k+r, k) := pd*A(N-k, k-1)+(N-k+r-2)*A(N-k+r-1, k);
end do; end proc;
```

and

```
filrght := proc (N, r) local k, Nsplit, pd, l; global A;
Nsplit := floor((1/2)*N+1/2);
for k from Nsplit+1 to N-1 do
  pd := mul(N-k+r-1-l, l = 1 .. r-1);
  A(N-k+r, k) := pd*A(N-k, k-1)+(N-k+r-2)*A(N-k+r-1, k);
end do; end proc;
```

And for each half of the diagonal, we can establish an independent thread to fulfill the task. We implemented this approach in Maple.

```
Threaded := proc (n, r) local N, k, Nsplit; global A;
A := Matrix(n, n, fill = 0);
A(1, 1) := 1;
for N from 3 to n do
  Threads:-Task:-Start(null, Task = [fileft, N, r],
                             Task = [filrght, N, r])
end do;  end proc;
```

Table 11 compares the threaded scheme with the sequential scheme in the computation of an $n \times n$ matrix of Stirling cycle numbers. The table reflects the limitation that there is an overhead cost to setting up new threads, and the benefit of the threaded approach is felt only when the amount of work achieved within a thread outweighs the overhead. In this implementation, new threads are created for each loop. We are exploring new methods of calculation which will allow the threads to work more efficiently, with less overhead.

n	Threaded scheme	Sequential scheme
500	0.856	0.329
1000	2.420	1.380
2000	9.240	9.610
2500	17.900	24.900
3000	29.880	39.870
4000	87.960	142.800

Table 11. Timings in seconds of comparison of threaded code with sequential code in generating sequences of Stirling Cycle numbers. The tests were made on an AMD 8-core processor.

6 Implementation of Eulerian numbers

The Eulerian numbers share many similarities with the Stirling numbers, and all the methods described above can be applied to their case. The numbers obey the following recurrence relations [1].

$$\left\langle {n \atop m} \right\rangle = (m+1)\left\langle {n-1 \atop m} \right\rangle + (n-m)\left\langle {n-1 \atop m-1} \right\rangle , \tag{33}$$

$$\left\langle\!\left\langle {n \atop m} \right\rangle\!\right\rangle = (m+1)\left\langle\!\left\langle {n-1 \atop m} \right\rangle\!\right\rangle + (2n-m-1)\left\langle\!\left\langle {n-1 \atop m-1} \right\rangle\!\right\rangle . \tag{34}$$

The present Maple functions `eulerian1` and `eulerian2` are recursively programmed implementations of these equations. As a consequence, they are very slow for large arguments. The new implementation of these numbers consists of 4 functions, which follow the patterns of the Stirling number implementations.

1. Eulerian1: calculates a singleton Eulerian number of the first kind.
   As with Stirling partition numbers, a finite sum is known which is distinctly the fastest method for a singleton computation [8]:

$$\left\langle {n \atop k} \right\rangle = \sum_{j=0}^{k+1} (-1)^j \binom{n+1}{j} (k-j+1)^n . \tag{35}$$

2. Eulerian1Matrix: calculates a sequence of Eulerian numbers of the first kind.
   This follows the sequence calculation of Stirling numbers, using (33).
3. Eulerian2: calculates singleton Eulerian numbers of the second kind.
   This follows the simpleton method used earlier for Stirling cycle numbers.
4. Eulerian2Matrix: calculates a sequence of Eulerian numbers of the second kind.
   This follows the sequence calculation of Stirling numbers.

6.1 Timings for Eulerian number calculations

In view of the similarities with Stirling numbers, we shall not labour the comparisons between methods, since they form the same procession of speeds seen before. Table 12 compares the new implementations, following the patterns set above.

n	Eulerian1	Eulerian1Matrix	Eulerian2	Eulerian2Matrix
60	2.776	0.015	3.135	0.000
80	9.656	0.015	10.795	0.015
100	23.743	0.015	27.924	0.015
120	52.884	0.015	60.684	0.015
140	101.634	0.046	115.159	0.046
160	180.430	0.062	204.049	0.062
180	300.036	0.062	342.952	0.062

Table 12. Timings in seconds of computations of Eulerian numbers. Column headings give the functions used.

References

1. Graham, R.L., Knuth, D.E., Patashnik, O., Concrete Mathematics, Addison-Wesley Publishing Co., Reading, Massachusetts, 1994.
2. Comtet, L., Advanced Combinatorics, D. Reidel Publishing Co., Dordrecht, Holland, 1974.
3. Howard, F. T., *Associated Stirling Numbers*, The Fibonacci Quarterly, Vol. 18(4), 303–315, 1980.
4. Corless, R.M., Jeffrey, D.J., Knuth, D.E., A Sequence of Series for the Lambert W Function, Proceedings of ISSAC 1997, ed. W.W. Kuechlin, ACM Press, 1997.
5. Karamata, J., Theoreme sur la sommabilite exponentielle et d'autres sommabilites rattachant, Mathematica, Cluj, Romania, vol. 9, 164–178, 1935.
6. Knuth, D.E., Two Notes on Notation, The American Mathematical Monthly, vol. 99, 403–422, 1992.
7. Stirling, J., Methodus Differentialis, London, 1730.
8. Lehmer, D. H., *Generalized Eulerian Numbers*, Journal of Combinatorial Theory, Series A 32, 195–215, 1982.
9. Wilf, H. S., *The Asymptotic Behavior of the Stirling Numbers of the First Kind*, Journal of Combinatorial Theory, Series A 64, 344–349, 1993.