AN IMPROVED ALGORITHM FOR CALCULATING THE SUM-OF-FACTORIALS FUNCTION

Jon Perry
51 High Street, Belbroughton, West Midlands, DY9 9ST, England

Abstract  The sum of factorials function, also known as the left factorial function, is defined as \( !n = 0! + 1! + \cdots + (n - 1)! \). These have been used by Smarandache and Kurepa to define the Smarandache-Kurepa Function (see reference [1], [2]). This paper presents an effective method for calculating \( !n \), and implements the Smarandache-Kurepa function by using one new method.

1. Introduction
We define \( !n \) as \( 0! + 1! + \cdots + (n - 1)! \).
A simple PARI/GP program to calculate these values is below:
\[
soff(n) = \sum_{i=0}^{n-1} i!
\]
Then,
\[
\text{for}(i = 0, 10, \text{print1(","soff(i)))}
\]
gives the desired output;
0, 1, 2, 4, 10, 34, 154, 874, 5914, 46234, 409114,
which is A003422 at OEIS [3].

2. A new method
If we write out what the sum of factorials function is doing, we can write:

\[
1 + 1 + 1.2 + 1.2.3 + 1.2.3.4 + 1.2.3.4.5 +
\]
and so on.
If we now read down the columns, we see that this can be written as:
\[
1 + 1[1 + 2[1 + 3[1 + 4[1 + \cdots]].
\]
This is because we have an opening 1 from 0!. Then 1 is a factor of all the remaining factorials. However 1 is the only factor of 1 of the factorials, namely 1!, so we have

\[ 1 + 1[1 + \cdots]. \]

Having removed the 1!, 2 is now a factor is all remaining factorials, and is the final factor in 2!, hence

\[ 1 + 1[1 + 2[1 + \cdots. \]

and so on.

!n requires inputs from 0! to (n − 1)!, and hence we are required to stop the nested recursion by n − 1. e.g. for !5, we have

\[ 1 + 1[1 + 2[1 + 3[1 + 4[1]]]]. \]

We can validate this:

\[
\begin{align*}
1 + 1[1 + 2[1 + 3[1 + 4[1]]]] &= 1 + 1[1 + 2[1 + 3[5]]] \\
&= 1 + 1[1 + 2[16]] \\
&= 1 + 1[33] \\
&= 34.
\end{align*}
\]

3. Code for new method

We can see how the new method decreases execution time, the original method presented performs \(O(k^2)\) multiplications and \(O(k)\) additions. This method performs \(O(k)\) multiplications and \(O(k)\) additions.

PARI/GP code for the routine is below:

```pari
def qsoff(n) = local(r); r = n; forstep(i = n - 2, 1, 4 - 1, r *= i; r += n); r
```

4. Implementing the Smarandache-Kurepa function

We need only consider primes, and the \(sk\) variable needs only range from 1 to \(p - 1\) (if \(!1\) to \(!p\) are not divisible by \(p\), then \(!p + k\) will never be as all new terms have \(p\) as a factor).

For prime \((p = 2, 500)\), for \((sk = 1, p)\), if \(\text{qsoff}(sk)\)

This is obviously wasteful, we are calculating \(\text{qsoff}(sk)\) very repetitively. The code below stores the \(\text{qsoff}\) values in a vector.
An Improved Algorithm for Calculating the Sum-of-Factorials Function

\begin{verbatim}
v=vector(500, i, qsoff(i)); forprime (p = 2500, for (sk = 1, p, if (v[sk]))
The following output is produced:

2, -4, 6, 6, -5, 7, 7, -12, 22, 16, -, -, -, -55, -, -54, 42, -, -, 24, -, -, 25, -,
-6, -97, -, -133, -, -, -64, 94, 72, 58, -, -, -49, 69, 19, -, 78, -, 14, -, 208,
167, -, 138, 80, 59, -, -, -, -63, 142, 41, -, 110, 22, 286, 39, -, 84, -, -, 215, 80,
14, 305, -, 188, 151, 53, 187, -, 180, -, -, -, -44, 32, 83, 92, -, 300, 16, -.

5. Additional relations

The basic pattern created in this paper also allows for the rapid calculation
of other Smarandache-like functions based on the sum of factorials function.

For example, we could define \(SSF(n)\) as the sum of squares factorial, e.g.
\(SSF(10) = 0! + 1! + 4! + 9!\), and the corresponding general expansion is

\[1 + 1[1 + 2.3.4][1 + 5.6.7.8.9][1 + \cdots].\]

Or we can define the sum of factorials squared function as

\[0!2 + 1!2 + 2!2 + \cdots.\]

In this case, the expansion is

\[1 + 1[1 + 4][1 + 9][1 + \cdots].\]

References

/FUNCT1.TXT

Function.html

Anum=A003422
\end{verbatim}