Notes on Number Theory and Discrete Mathematics
Print ISSN 1310–5132, Online ISSN 2367–8275
Vol. 26, 2020, No. 4, 113–121
DOI: 10.7546/nntdm.2020.26.4.113-121

On algorithms for computing the sums of powers of arithmetic progressions

Peter J. Shiue[1], Shen C. Huang[2] and Eric Jameson[3]

[1] Department of Mathematical Sciences, University of Nevada, Las Vegas,
Las Vegas, NV 89154-4020, USA
e-mail: shiue@unlv.nevada.edu

[2] Department of Mathematical Sciences, University of Nevada, Las Vegas,
Las Vegas, NV 89154-4020, USA
e-mail: huangs5@unlv.nevada.edu

[3] Department of Mathematical Sciences, University of Nevada, Las Vegas,
Las Vegas, NV 89154-4020, USA
e-mail: jamese1@unlv.nevada.edu

Received: 13 July 2020 **Revised:** 3 November 2020 **Accepted:** DD Month YYYY

Abstract: This paper is concerned with sums of powers of arithmetic progressions of the form $a^p + (a+d)^p + (a+2d)^p + \cdots + (a+(n-1)d)^p$, where $n \geq 1$, p is a non-negative integer, and a and d are complex numbers with $d \neq 0$. This paper gives an elementary proof to a theorem presented by Laissaoui and Rahmani [9] as well as an algorithm based on their formula. Additionally, this paper presents a simplification to Laissaoui and Rahmani's formula that is better suited to computation, and a second algorithm based on this simplification. Both formulas use Stirling numbers of the second kind, which are the number of ways to partition p labelled objects into k nonempty unlabelled subsets [4]. An analysis of both algorithms is presented to show the theoretical time complexities. Finally, this paper conducts experiments with varying values of p. The experimental results show the proposed algorithm remains around 10% faster as p increases.

Keywords: Analysis of algorithms, Arithmetic progressions, Dynamic programming, Stirling number of the second kind.
2010 Mathematics Subject Classification: 05A10, 11B25, 11B65, 11B73, 68N15, 68Q25.

1 Introduction

The sums of powers of arithmetic progressions is an important topic in computational combinatorics and number theory. This sum is of the form

$$S_{p,a,d}(n) = a^p + (a+d)^p + \cdots + (a+(n-1)d)^p = \sum_{j=0}^{n-1} (a+jd)^p, \tag{1}$$

where $n \geq 1$, p is a non-negative integer, and a and d are complex numbers with $d \neq 0$. Several formulae and algorithms have been developed to compute this sum. In 2017, Laissaoui and Rahmani's [9] presented a method to solve this problem using generating functions related to Howard's method [6]. This paper aims to give an elementary proof to Laissaoui and Rahmani's formula and to improve their method and algorithm.

Letting $a = d = 1$ in (1) gives one of the most famous problems as follows:

$$S_{p,1,1}(n) = \sum_{j=1}^{n} j^p = 1^p + 2^p + 3^p + \cdots + n^p. \tag{2}$$

For example, letting $p = 1$ gives the identity

$$S_{1,1,1}(n) = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}.$$

1.1 Origin

The classical theorem of Faulhaber states that the sums of odd powers

$$1^{2m-1} + 2^{2m-1} + \cdots + n^{2m-1}$$

can be expressed as a polynomial of the triangular numbers $T_n = \dfrac{n(n+1)}{2}$ (see Knuth [8]). Chen *et al.* [5] observed that the classical Faulhaber's theorem on sum of odd powers also holds for arbitrary arithmetic progressions, namely, the odd power sums of any arithmetic progression

$$a, a+b, \cdots, a+(n-1)b$$

can be expressed as a polynomial in $na + \dfrac{n(n+1)b}{2}$.

Finding the best formula and algorithm to compute (1) remains an interest within the mathematics and computer science community. Laissaoui and Rahmani [9] gave an explicit formula giving the polynomial in n that relies on Stirling numbers of the second kind and binomial coefficients, given in the next section. In addition, they gave an algorithm to compute such polynomials. However, no time-complexity analysis was presented.

1.2 Stirling numbers of the second kind

The Stirling numbers of the second kind, denoted by $S(p,k)$, count the number of ways to partition a set of p labelled objects into k nonempty unlabelled subsets [4]. The explicit formula

can be written as

$$S(p,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^j\binom{k}{j}(k-i)^p, \tag{3}$$

where $\binom{k}{j}$ denotes the binomial coefficients. The Stirling numbers of the second kind satisfy the following recurrence relation

$$S(p+1,k) = kS(p,k) + S(p,k-1), \tag{4}$$

with $S(0,0) = 1$ and $S(p,0) = S(0,k) = 0$.

1.3 Proposal

This paper proposes an elementary proof to Laissaoui and Rahmani's formula in the next section. In section 2.3, a new algorithm is proposed. Next, analysis on Laissaoui and Rahmani's algorithm and the new algorithm is presented. Lastly, experimental results on these algorithms are given.

2 Main results

In this section, an elementary proof to Laissaoui and Rahmani's formula is presented. This proof avoids the use of generating functions and relies solely on (5) and (6).

2.1 Preliminaries

Since the binomial coefficients are involved, it is important to note that Pascal's identity [1],

$$\binom{r}{k} = \binom{r-1}{k-1} + \binom{r-1}{k},$$

can be written as

$$\binom{r}{k} = \binom{r+1}{k+1} - \binom{r}{k+1}, \tag{5}$$

where r may not be a positive integer.

In [4], Charalambides gave the expansion of t^p in terms of the Stirling numbers of the second kind and the binomial coefficients, written as follows:

$$t^p = \sum_{j=1}^{p} j!S(p,j)\binom{t}{j}. \tag{6}$$

2.2 An elementary proof

Theorem 2.1. *Let a and d be complex numbers with $d \neq 0$, p a non-negative integer, and n a positive integer. Then the sum of the pth powers of the first n terms in the arithmetic progression with first term a and common difference d can be expressed as*

$$S_{p,a,d}(n) = d^p\sum_{k=0}^{p} k!S(p,k)\left[\binom{n+\frac{a}{d}}{k+1} - \binom{\frac{a}{d}}{k+1}\right], \tag{7}$$

where $S(p,k)$ denotes the Stirling number of the second kind.

Proof. Using (6), let $t = (k-1) + \frac{a}{d}$, then

$$\left((k-1) + \frac{a}{d}\right)^p = \sum_{j=1}^{p} j!S(p,j)\binom{(k-1)+\frac{a}{d}}{j}.$$

Multiplying d^p to both sides gives

$$(a + (k-1)d)^p = d^p \sum_{j=1}^{p} j!S(p,j)\binom{(k-1)+\frac{a}{d}}{j}.$$

Next, summing both sides from $k = 1$ to n gives

$$\begin{aligned}\sum_{k=1}^{n} \big(a + (k-1)d\big)^p \\ = d^p \sum_{j=1}^{p} j!S(p,j) \sum_{k=1}^{n} \binom{(k-1)+\frac{a}{d}}{j}.\end{aligned} \tag{8}$$

The binomial term on the right hand side of (8) needs to be investigated as k varies from 1 to n. Pascal's identity given in (5) is utilized.

For $k = 1$,

$$\binom{(k-1)+\frac{a}{d}}{j} = \binom{\frac{a}{d}}{j} = \binom{\frac{a}{d}+1}{j+1} - \binom{\frac{a}{d}}{j+1}.$$

For $k = 2$,

$$\binom{(k-1)+\frac{a}{d}}{j} = \binom{\frac{a}{d}+1}{j} = \binom{\frac{a}{d}+2}{j+1} - \binom{\frac{a}{d}+1}{j+1}.$$

Inductively, it can be shown that for $k = n$,

$$\binom{(n-1)+\frac{a}{d}}{j} = \binom{\frac{a}{d}+n}{j+1} - \binom{\frac{a}{d}+(n-1)}{j+1}.$$

Hence,

$$\sum_{k=1}^{n} \binom{(k-1)+\frac{a}{d}}{j} = \binom{\frac{a}{d}+n}{j+1} - \binom{\frac{a}{d}}{j+1}. \tag{9}$$

Combining the results of (8) and (9) gives

$$S_{p,a,d}(n) = d^p \sum_{j=1}^{p} j!S(p,j)\left[\binom{n+\frac{a}{d}}{j+1} - \binom{\frac{a}{d}}{j}\right],$$

which is the desired result. Note: the beginning index starts at $j = 1$ because $S(p,0) = 0$. □

2.3 A simplification

Laissaoui and Rahmani's formula may be simplified to reduce the actual run-time, which will be discussed in more details in section 3.

Corollary 2.1.1. *Under the same assumptions as in Theorem 2.1, the sums of p-th powers of the first n terms in the arithmetic progression with first term a and common difference d can be expressed as*

$$S_{p,a,d}(n) = \frac{d^p}{k+1} \sum_{k=1}^{p} S(p,k) \left[\left(n + \frac{a}{d} \right)_{k+1} - \left(\frac{a}{d} \right)_{k+1} \right]. \tag{10}$$

where S(p,k) denotes the Stirling number of the second kind, and $(x)_n$ denotes the falling factorial $(x)_n = x(x-1)\cdots(x-(n-1))$.

Proof. Use the identity

$$\binom{x}{k} = \frac{(x)_k}{k!}$$

to simplify the binomial coefficients in (7). □

3 Analysis of algorithms

In this section, the time-complexities of the original algorithm by Laissaoui and Rahmani and the new improved algorithm, based on Corollary 2.1.1, are compared, both with a naïve brute force approach, as well as a dynamic programming approach.

3.1 Laissaoui and Rahmani's method

The first algorithm to compute $S_{p,a,d}(n)$ is based on equation (7). The time-complexity of this algorithm as given as follows. First of all, $k!$ takes $k-1$ operations to compute by definition. The binomial coefficients $\binom{n}{k}$ can therefore be computed with $n-1+k-1+(n-k)-1+2 = 2n-1$ operations. Thus, from (7), the portion of the algorithm which does not compute $S(p,k)$ takes

$$O(k + 2(n + \tfrac{a}{d}) + 2(\tfrac{a}{d})) = O(k + n + \tfrac{a}{d})$$

operations to compute.

The algorithm to compute $S(p,k)$ is based on the recurrence relation (4). Let $T(p,k)$ be the number of operations needed to compute $S(p,k)$. From the recurrence relation (4), we see that $T(p,k)$ can also be written as a recurrence relation

$$T(p,k) = T(p-1,k) + T(p,k-1) + 2$$

where the extra two operations are from multiplying the first term by k and adding the two terms together. Solving this recurrence relation as in [2] gives that $T(p,k) = O(2^p)$.

The calculations shown above are for one summand in (7). Summing from $k=0$ to p gives an upper bound for the time complexity of the first algorithm as

$$O(p2^p + p^2 + np + \tfrac{ap}{d}) = O(p2^p + np + \tfrac{ap}{d}) \tag{11}$$

in terms of the input variables p, a, d, n.

3.2 An improvement

By using formula (10), the complexity of the second algorithm to compute $S_{p,a,d}(n)$ is slightly improved. The falling factorial $(x)_k$ takes $k-1$ operations to compute. Thus from (10), this adds $2k$ operations. Since we no longer compute $k!$ and instead compute $\frac{1}{k+1}$, the portion of the algorithm that does not compute $S(p,k)$ takes $O(k)$ operations. Summing from $k=0$ to p gives an upper bound for the time complexity of

$$O(p2^p + p^2 + 1) = O(p2^p). \tag{12}$$

Note that the complexity no longer depends directly on the values of a, d, n. Comparing (11) and (12) gives evidence to suggest that the improved algorithm will run faster than the original algorithm on average.

3.3 Dynamic programming

In Bein's chapter of [2], he explains that the usage of dynamic programming can reduce the computational time-complexities of certain recurrence relations using a matrix-like table. The recurrence relation (4) allows the construction of a bottom-up table to avoid repeating computations. Altogether, computing $S(p,k)$ can be done in $O(pk)$ operations. Based on this information, the upper bound for the time complexity of the original algorithm is

$$O(p^3 + p^2 + np + \tfrac{ap}{d}) = O(p^3 + np + \tfrac{ap}{d}) \tag{13}$$

and the upper bound for the time complexity of the improved algorithm is

$$O(p^3 + p^2 + 1) = O(p^3). \tag{14}$$

Similar to the recurrence relation approach, comparing (13) and (14) gives evidence to suggest that the improved algorithm will still run faster than the original algorithm on average.

4 Experimental results

In this section, experimental results for evaluating the sums of powers of arithmetic progressions are presented. First, the specifications of the testing environment are given. Next, the details of the experiments are described. Lastly, the results of the original method versus the simplified method will be discussed.

4.1 Specifications

All of the experiments are implemented using Python 3.7, with PyCharm IDE. The libraries include Sympy and time. Sympy is a library that does symbolic mathematics. Data visualizations are done using Matlab. The experiments are run in a 64-bit Windows 10.0.18362 operating system. The computer used to run the tests consists of an Intel i7-8700K running at 3.7GHz with 6 physical cores and 16 GB of RAM.

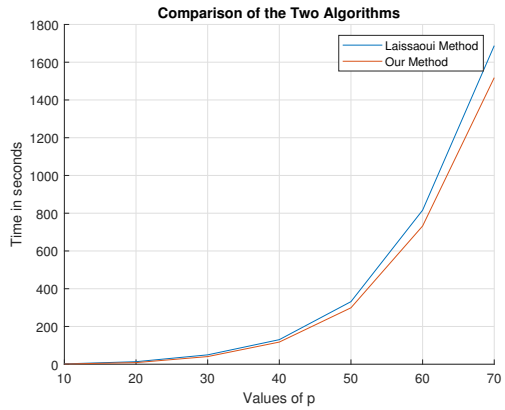
4.2 Description of experiments

There are a total of seven experiments. In each experiment, the parameter p, which is the same p as in (7), varies. The first experiment sets $p = 10$. In subsequent experiments, p increments by 10 each time, with the last experiment testing on $p = 70$. The run-times are calculated using arithmetic mean over 100 trials.

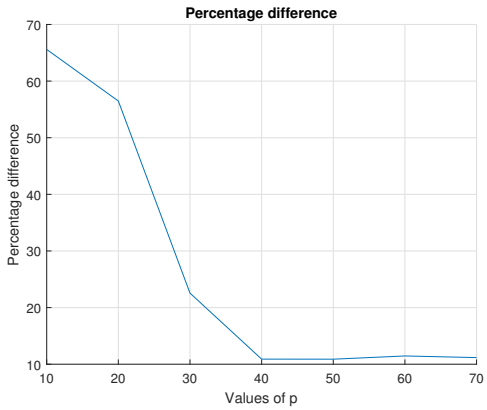
In each experiment, the "stirling" function from the Sympy library is used to compute the Stirling numbers of the second kind. In addition, combinatorial functions such as "factorial" and "combsimp" are used. The functions for falling factorial and binomial are coded without dynamic programming.

4.3 Results and discussion

The results of the experiments are shown in Figures 1(a) and 1(b). Figure 1(a) shows that the simplified version based on Corollary 2.1.1 is superior to the original method from Theorem 1. Figure 1(b) shows the percentage difference between the two methods. For lower values of p, the percentage difference is much higher. Due to the p values being smaller, the overhead associated with the proposed algorithm may have contributed to the drastic difference. As the value of p increases, the simplified algorithm performs about 10% to 11% consistently better than the original.



(a) A comparison of the runtimes of the two algorithms shows that our method runs faster overall for all values of p.



(b) The percentage difference in runtimes of the two algorithms is consistent with our analysis.

Figure 1: Average run time of Laissaoui and Rahmani's method vs Our method based on Corollary 2.1.1.

5 Conclusion

In this paper, both an elementary proof and a simplification to Laissaoui and Rahmani's method for computing $S_{p,a,d}(n)$ are proposed. The elementary proof given in section 2 is an alternative proof to the one found in the original paper [9]. Corollary 2.1.1 is a simplification to the original formula given in (7). It uses falling factorials instead of binomial coefficients.

In Section 3, extensive analyses of both algorithms are presented. These analyses suggest that the simplified algorithm will on average run faster than the original algorithm. The experimental results also support the analyses.

This paper focused on using Stirling numbers of the second kind. A future project may involve algorithms for solving the sums of powers of arithmetic progressions using other special combinatorial numbers such as Bernoulli polynomials/numbers [3, 6, 11] and Eulerian numbers [7, 10].

Acknowledgements

The authors would like to thank Professor Wolfgang Bein, Professor Chein-I Chang, and Professor Anthony Shannon and three anonymous referees for their comments and suggestions which led to considerable improvements to this paper. We also like to thank the referee for providing reference number 3.

References

[1] Aigner, M. (2010). *Discrete Mathematics*. American Mathematical Society.

[2] Bein, W. (2013). Advanced techniques for dynamic programming. In: Pardalos, P., M., Du, D.-Z., & Graham, R. L. (Editors), *Handbook of Combinatorial Optimization*, Chapter 2, Springer, pp. 41–92.

[3] Bounebirat, F., Laissaoui, D., & Rahmani, M. (2018). Several explicit formulae of sums and hyper-sums of powers of integers. *Online Journal of Analytic Combinatorics*, 13.

[4] Charalambides, C. A. (2002). *Enumerative Combinatorics*. Chapman & Hall / CRC Press, pp. 278–279.

[5] Chen, W. Y. C., Fu, A. M., & Zhang, I. F. (2009). Faulhaber's theorem on power sums. *Discrete Mathematics*, 309 (10), 2974–2981.

[6] Howard, F.T. (1996). Sums of powers of integers via generating functions. *Fibonacci Quarterly*, 34 (3), 244–256.

[7] Hsu, L. C., & Shiue, P. J.-S. (1999). On certain summation problems and generalizations of Eulerian polynomials and numbers. *Discrete Mathematics*, 204 (1–3), 237–247.

[8] Knuth, D. E. (1993). Johann Faulhaber and sums of powers. *Mathematics of Computation*, 61 (203), 277—294.

[9] Laissaoui, D., & Rahmani, R. (2017). An explicit formula for sums of powers of integers in terms of Stirling numbers. *Journal of Integer Sequences*, 20 (4), Aricle 17.4.8.

[10] Pita-Ruiz, C. (2018). On a generalization of Eulerian numbers. *Notes on Number Theory and Discrete Mathematics*, 24 (1), 16–42.

[11] Vassilev, P., & Vassilev-Missana, M. (2005). On the sum of equal powers of the first n terms of an arbitrary arithmetic progression. *Notes on Number Theory and Discrete Mathematics*, 11 (3), 15–21.