




Fast computation of binomial coefficients

Leonardo C. Araujo¹  · João P. H. Sansão¹ · Adriano S. Vale-Cardoso¹

Received: 7 June 2019 / Accepted: 20 February 2020 / Published online: 19 March 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract

One problem that arises in computation involving large numbers is precision. In certain situations, the result might be represented by the standard data type, but arithmetic precision might be compromised when dealing with large numbers in the course to the result. Binomial coefficients are an example that suffer from this torment. In the present paper, we review numerical methods to compute the binomial coefficients: Pascal's recursive method; prime factorization to cancel out terms; gamma function approximation; and a simplified iterative method that avoids loss in precision. Acknowledging that the binomial coefficients might be obtained by a successive convolution of a simple discrete rectangular function, we propose a different approach based on the discrete Fourier transform and another based on its fast implementation. We analyze and compare performance and precision of all of them. The proposed methods overcome the previous ones when computing all coefficients for a given level n , attaining better precision for large values of n .

Keywords Binomial coefficient · Discrete fourier transform · fft · Numerical analysis

1 Introduction

The number of combinations of n elements taken k at a time is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}, \quad (1)$$

✉ Leonardo C. Araujo
leolca@ufs.j.edu.br

João P. H. Sansão
joao@ufs.j.edu.br

Adriano S. Vale-Cardoso
adrianosvc@ufs.j.edu.br

¹ Universidade Federal de Sao Joao del-Rei, Ouro Branco, Minas Gerais, Brazil

Table 1 Results from computing the factorial of a number using single and double precision

n	$n!$	(float) $n!$	(double) $n!$
1	1	1	1
2	2	2	2
3	6	6	6
4	24	24	24
5	120	120	120
6	720	720	720
7	5040	5040	5040
8	40320	40320	40320
9	362880	362880	362880
10	3628800	3628800	3628800
11	39916800	39916800	39916800
12	479001600	479001600	479001600
13	6227020800	6227020800	6227020800
14	87178291200	87178289152	87178291200
15	1307674368000	1307674279936	1307674368000
16	20922789888000	20922788478976	20922789888000
17	355687428096000	355687414628352	355687428096000
18	6402373705728000	6402373530419200	6402373705728000
19	121645100408832000	121645096004222976	121645100408832000
20	2432902008176640000	2432902023163674624	2432902008176640000
21	51090942171709440000	51090940837169725440	51090942171709440000
22	112400072777607680000	1124000724806013026304	112400072777607680000
23	25852016738884976640000	25852017444594485559296	25852016738884978212864

1.1 Stirling's approximation for binomial coefficient

One method to approach the binomial coefficient without computing the involved factorials is by means of the Stirling's approximation [10]:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n. \quad (4)$$

Using this approach, the binomial coefficient may be approximated by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \approx \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n}{\sqrt{2\pi k} \left(\frac{k}{e}\right)^n \sqrt{2\pi(n-k)} \left(\frac{(n-k)}{e}\right)^n} \quad (5)$$

$$= \frac{\left(\frac{n}{k}\right)^k}{\sqrt{2\pi k} \left(1 - \frac{k}{n}\right)^{n-k+\frac{1}{2}}}. \quad (6)$$

The results given by this approximation are presented in Fig. 2 and Table 2. This approximation might not be used to compute the exact values of the binomial coefficients, even for small values of n and k .

1.2 Canceling equal terms and mutual factors

The binomial coefficient is symmetric

$$\binom{n}{k} = \binom{n}{n-k}, \quad (7)$$

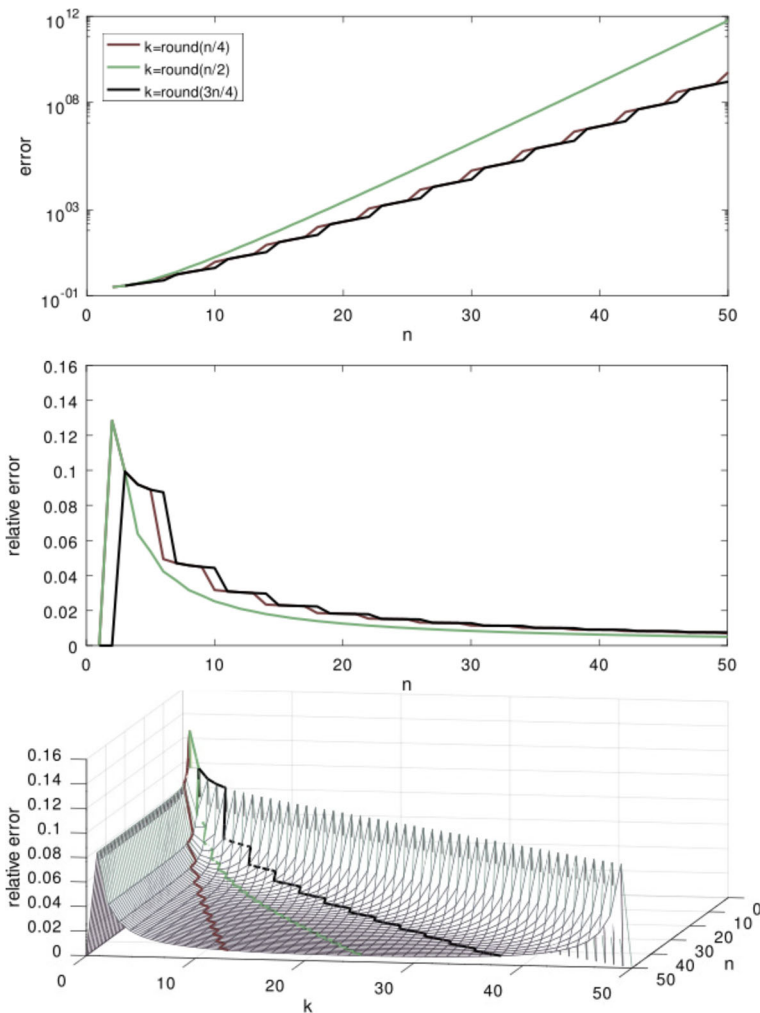


Fig. 2 Error in Stirling's approximation to the binomial coefficients, as a function of n and k . The first plot presents the absolute error as a function of n and k was chosen as $\lfloor \frac{n}{4} \rfloor$, $\lfloor \frac{n}{2} \rfloor$ and $\lfloor \frac{3n}{4} \rfloor$. The plots present the relative error of the approximation

Table 2 Comparison for some values of the Stirling's approximation to the binomial coefficients

n	k	$\binom{n}{k}$	Stirling's
2	1	2	2.2568
3	1	3	3.2981
3	2	3	3.2981
4	1	4	4.3677
4	2	6	6.3831
4	3	4	4.3677
5	1	5	5.4447
5	2	10	10.5377
5	3	10	10.5377
5	4	5	5.4447
6	1	6	6.5247
6	2	15	15.7416
6	3	20	20.8470
6	4	15	15.7416
6	5	6	6.5247
\vdots	\vdots	\vdots	\vdots
22	5	26334	26805.8286
22	6	74613	75763.0663
22	7	170544	172890.2765
22	8	319770	323816.5024
22	9	497420	503362.9991
22	10	646646	654116.0548
22	11	705432	713491.2077
22	12	646646	654116.0548
22	13	497420	503362.9991
22	14	319770	323816.5024
22	15	170544	172890.2765
22	16	74613	75763.0663

therefore, to simplify, we might always consider k as the smallest value between k and $n - k$. Equation (1) might be rewritten cancelling out the larger term in the denominator, which is $(n - k)!$, by the convention above. It leads to

$$\binom{n}{k} = \frac{n!}{k!(n - k)!} = \frac{\prod_{i=n-k+1}^n i}{k!}. \quad (8)$$

Since k is reasonably small compared to n , and since the result is an integer, we might decompose the denominator integers from 1 to k into primes and cancel them out with the factors in the numerator. This proposal is used in MATLAB's implementation of the function `nchoosek`. A simplification is given by [4]. For a 64-bit integer, it is enough to cancel out the primes 2, 3, and 5. This simplification will be unreasoned

when computing the binomial coefficients of large numbers, as $C(1000, 353)$ [2]. The example provided in [2] requires large precision arithmetic, even to represent the resulting number, which has 281 decimal digits and requires 932 bits.

1.3 Rolfe's good recursion

Equation (8) might be rewritten as

$$\binom{n}{k} = \frac{\prod_{i=n-k+1}^n i}{k!} = \frac{n}{k} \cdot \frac{\prod_{i=n-k+1}^{n-1} i}{(k-1)!} = \frac{n}{k} \binom{n-1}{k-1}, \quad \text{for } k > 0, \quad (9)$$

which might be used to provide a better recurrent relation to compute the binomial coefficient [9].

1.4 Yannis Manolopoulos' simple iterative method

Still another approach to rewrite (8) in an iterative way is given by [7]. This approach alternates multiplication and division at each step, avoiding data type overflow. It is based on the relation

$$\binom{n}{k} = \frac{\prod_{i=n-k+1}^n i}{k!} = \prod_{i=n-k+1}^n \frac{i}{n-i+1}. \quad (10)$$

This approach is used in the implementation of the function `nchoosek` in GNU Octave.

1.5 Gamma function approach

The implementation of the function `bincoeff` in GNU Octave uses the approximation given by the gamma function. The factorial may be written as a gamma function: $z! = \Gamma(z+1)$. Therefore,

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)} = e^{\ln \Gamma(n+1) - \ln \Gamma(k+1) - \ln \Gamma(n-k+1)}. \quad (11)$$

1.6 Dynamic programming

Dynamic programming (DP) is a very general technique consisting in breaking a problem into simpler sub-problems, where several overlaps might exist. Problem solving could become more efficient by reusing solutions already computed. Computing the binomial coefficient by means of Pascal's recursion is the type of problem suitable to the DP approach. Figure 3 shows the recursion necessary to compute $\binom{4}{2}$. A whole sub-branch appears twice in solution, so we need to compute it only once, reusing the solution when the problem appears again on another branch. Algorithm 1

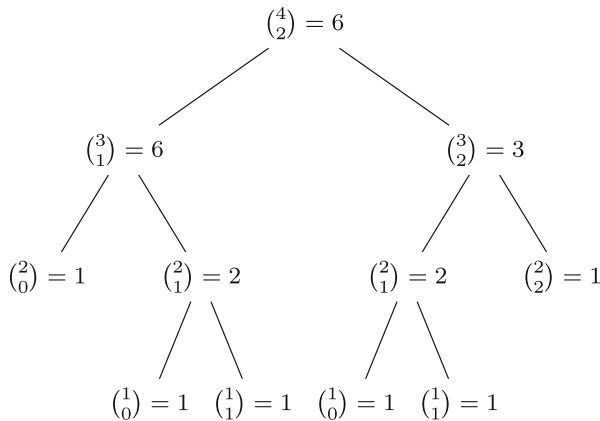


Fig. 3 Tree of values computed on the Pascal's recursion

presents the pseudo-code for that implementation. In this implementation, the coefficients are computed from level 0 up to level n (reverting the Pascal's recursion order), computing only the necessary values to achieve $\binom{n}{k}$ (column index j will be limited to $\min(i, k)$, where i is the line index).

Algorithm 1 DP approach on Pascal's recursion.

input : values of n and k

output: the computed binomial coefficient $\binom{n}{k}$

```

for  $i \leftarrow 0$  to  $n$  do
    fill each row at a time, computing only the values that will be used for
     $j \leftarrow 0$  to  $\min(i, k)$  do
        if  $j = 0 \vee j = n$  then
             $C[i, j] \leftarrow 1$ 
        else
            use the recursive relation  $C[i, j] \leftarrow C[i - 1, j - 1] + C[i - 1, j]$ 
        end
    end
end
return  $C[n, k]$ 
    
```

Another approach would be to use the natural Pascal's recursion order and store the computed values in a hash table. Before computing each new value, check the table if it was already computed, what costs $\mathcal{O}(1)$. The final algorithm will cost $\mathcal{O}(n \times k)$.

Applying DP to the Rolfe's good recursion (see Section 1.3), which is a better recursion compared to Pascal's, it leads us to Yannis Manolopoulos' simple iterative method (see Section 1.4).

In the following section, we present a method to compute all binomial coefficients in a row for a given n (see Section 2) and later (see Section 3) we come to a new formulation to compute individual coefficients $\binom{n}{k}$. The computational costs are given in Section 4 and finally they are compared to the previous methods (see Section 5).

2 Fast computation using FFT

Observing the Pascal's triangle in Fig. 1, we recognize that the elements in each line, for a given n , are of the form

$$C^n = \left[\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{k}, \dots, \binom{n}{n} \right] \quad (12)$$

and it might be interpreted as a $(n - 1)$ -time discrete convolution of $x[m] = \delta[m] + \delta[m - 1]$,

$$\underbrace{x[m] * x[m] * \dots * x[m]}_{(n-1) \text{ times}} \stackrel{\text{def}}{=} x^{*(n-1)}[m] = \text{IDFT}_{n+1} \left(X_{n+1}^n[l] \right), \quad (13)$$

where $X_{n+1}[l]$ is the $(n + 1)$ -point discrete Fourier transform (DFT) of $x[m]$.

What we have acknowledged is just the $(n - 1)$ -time Vandermonde's convolution of the first line of Pascal's triangle. In fact, using Chu-Vandermonde's identity [6],

$$\binom{n+m}{k} = \sum_{r=0}^k \binom{n}{r} \binom{m}{k-r}. \quad (14)$$

We may recognize it as the convolution between the n th and m th lines in Pascal's triangle. It might also be computed using the DFT:

$$C^{n+m} = \text{IDFT}_{n+m+1} \left\{ \text{DFT}_{n+m+1} C^n \cdot \text{DFT}_{n+m+1} C^m \right\}, \quad (15)$$

and C_k^{n+m} is the k th element from C^{n+m} .

3 Fast computation using results from the discrete Fourier transform

The $(n + 1)$ -point DFT of $x[m]$ is expressed as

$$X_{n+1}[l] = 1 + e^{-j \frac{2\pi l}{n+1}}, \quad l = 0, \dots, n. \quad (16)$$

Therefore, the IDFT of $X_{n+1}[l]$ is given by

$$\text{IDFT} \{X_{n+1}[l]\} = \frac{1}{n+1} \sum_{l=0}^n X_{n+1}^n[l] e^{j \frac{2\pi l}{n+1} k} \quad (17)$$

$$= \frac{1}{n+1} \sum_{l=0}^n \left(1 + e^{-j \frac{2\pi l}{n+1}}\right)^n e^{j \frac{2\pi l}{n+1} k} \quad (18)$$

$$= \frac{1}{n+1} \sum_{l=0}^n e^{-j \frac{\pi l n}{n+1}} 2^n \cos^n \left(\frac{\pi l}{n+1}\right) e^{j \frac{2\pi l}{n+1} k} \quad (19)$$

$$= \frac{2^n}{n+1} \sum_{l=0}^n \cos^n \left(\frac{\pi l}{n+1}\right) e^{j \frac{\pi l}{n+1} (2k-n)} \quad (20)$$

$$= \frac{2^n}{n+1} \left(1 + 2 \sum_{l=1}^{\lfloor \frac{n}{2} \rfloor} \cos^n \left(\frac{\pi l}{n+1}\right) \cos \left(\frac{\pi l}{n+1} (2k-n)\right)\right), \quad (21)$$

where we have used in (19) that $1 + e^{-j\theta} = e^{-\frac{\theta}{2}} (e^{\frac{\theta}{2}} + e^{-\frac{\theta}{2}})$ and in (21) we used the symmetry illustrated in Fig. 4, which provides $\cos(\pi - \theta) e^{j(\pi - \theta)} = \cos \theta e^{-j\theta}$. We have also used in (21) the fact that if n is odd, the missing term will be zero since, at $l = (n+1)/2$, the cosine in (20) is null.

Empirical results show that it suffices to compute up to $\lfloor \frac{n}{2} \rfloor + 1$ terms in the summation in (21); therefore, it might be approximated by

$$\text{IDFT} \{X_{n+1}[l]\} \approx \frac{2^n}{n+1} \left(1 + 2 \sum_{l=1}^{\lfloor \frac{n}{2} \rfloor + 1} \cos^n \left(\frac{\pi l}{n+1}\right) \cos \left(\frac{\pi l}{n+1} (2k-n)\right)\right), \quad (22)$$

which leads to correct results even for large values of n .

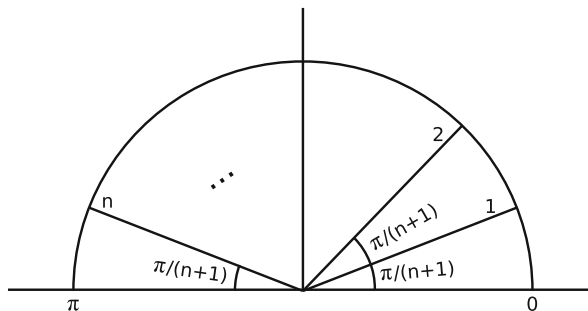


Fig. 4 Symmetry in (20)

4 Computational cost

Recursive algorithms have their elegance, they are simple to write and understand, but they require extra time cost for function calls and extra memory cost for stack bookkeeping. The recurrent approach given in (3) is a poor computational solution. To compute the coefficient $C(n, k)$, it will require $2C(n, k) - 1$ calls [9]. Each call will be $\mathcal{O}(1)$, so it takes $\mathcal{O}(2^n A(m))$ time to compute a given coefficient $C(n, k)$, where $A(m)$ is the cost of addition of two m -bit numbers (which is $\mathcal{O}(m)$). Since each step doubles the complexity,

$$T(n, k) = T(n - 1, k) + T(n - 1, k - 1) + \mathcal{O}(1), \quad (23)$$

with $T(n, n) = T(n, 0) = \mathcal{O}(1)$, we have the 2^n factor in the final complexity plus the additional function call cost.

The recurrent relation in (9) provides a better approach. It requires $k + 1$ calls but, by the problem's symmetry, one may guarantee that at most $\frac{n}{2}$ calls are required. The final complexity will be $\mathcal{O}(nM(m))$ plus recursion calls. Unfortunately, function calls require a significant cost in comparison with the cost of control structures, assignments, and operations. It will be evident in the results given in Section 5. It will also require larger memory amount to store backtracked values. The iterative relation given by [7], in (10), has at most $\frac{n}{2}$ multiplications and divisions. Its complexity will be only $\mathcal{O}(nM(m)M(m))$, since it avoids recursion calls.

The computational cost involved in computing (21) will be $\mathcal{O}(nM(m) \log m)$, since we have to compute two cosine functions at each step of the summation and one exponentiation. Trigonometric functions and exponentiations are $\mathcal{O}(M(m) \log m)$ complex, where $M(m)$ is the multiplication cost and m is the number of bits accuracy.

The discussion in this section is based on theoretic analysis, where the worst case scenario is considered. When taking such asymptotic analysis, we are ignoring machine-dependent constants and we want to focus on the growth on the running time, instead of the real machine execution time. We are not dealing here with some crucial implementation details, such as function calls, overhead in loops and control structures, memory usage, and other factors which impact algorithms run time. Nonetheless, in the following section, we present the running time results on a real computer.

5 Practical behavior

In this section, we present test results from executing the methods implemented in GNU Octave (all code is available at GitHub¹). We have implemented the following methods to compute the binomial coefficient:

- 1) `rbincoeff` uses the recurrent relation in (9);
- 2) `pfbcoeff` uses the prime factorization approach applied to (8) (MATLAB's method);

¹<https://github.com/leolca/bincoeff>

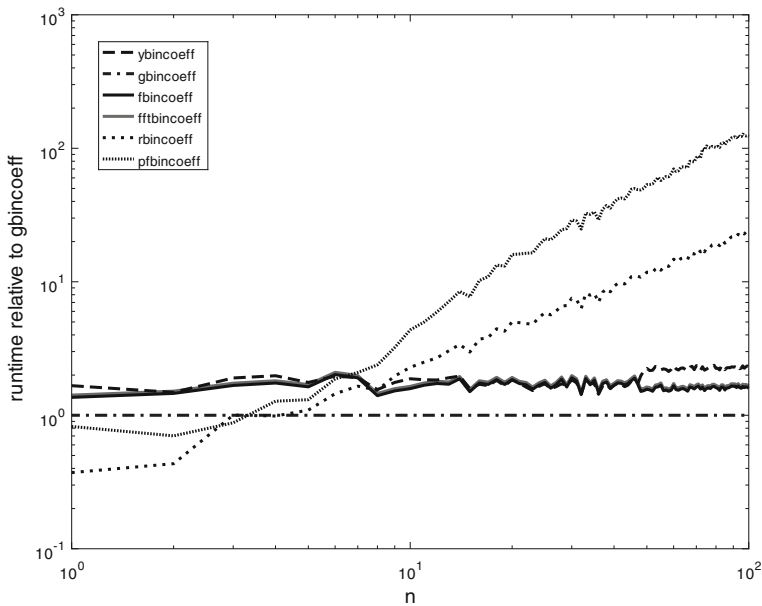


Fig. 5 Average execution time to compute individual binomial coefficients as a function of n . Results are normalized in relation to the `gbincoeff` method. The curve of `fbincoeff` was slightly dislocated, so that we might also see `fftbincoeff` curve

- 3) `gbincoeff` uses the gamma function, as described in (11) (method used in the function `bincoeff` in GNU Octave);
- 4) `ybincoeff` uses a simplified iterative method on (10) (method used in the function `nchoosek` in GNU Octave);
- 5) `fbincoeff` method proposed in this paper, using the DFT approach, as described in (22);
- 6) `fftbincoeff` method also proposed in this paper, using the FFT.

In Fig. 5, we present the average time of computation of $\binom{n}{k}$ as n increases. We observe that `gbincoeff` has the best performance, followed by the methods proposed here and `ybincoeff`. Prime factorization and the recursive method present the worst performance (they work well only for very small values of n). In the result presented, we computed the binomial coefficients ranging from $n = 2$ to $n = 100$, for each value of k from zero up to n , 5000 times each. We used a 1000-iteration loop with 5 copy-and-paste repetition of the code. Each group of 5 was averaged² and, in the end, we get the median from the 1000 averages generated [8].

²We have also tested the minimum, which gave similar result.

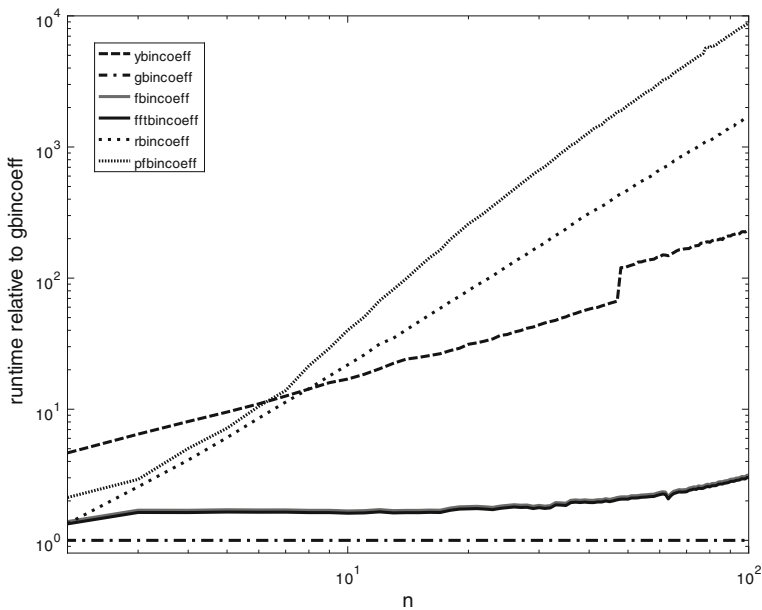


Fig. 6 Average execution time to compute the range (from $k = 0$ to n) of all coefficients for each value of n . Results are normalized in relation to the `gbincoeff` method. The curve of `fbincoeff` was slightly dislocated, so that we might also see `fftbincoeff` curve

To compute $\binom{n}{k}$, in general, we do not need to compute $\binom{n}{k-1}$, nor $\binom{n}{k+1}$, nor many other coefficients in the same level or in levels below. In the FFT-based method, we have no choice, but to compute all coefficients $\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{k}, \dots, \binom{n}{n}$, in order to compute the desired $\binom{n}{k}$. On the other hand, it does not require the computation of any coefficient on other levels, but only the coefficients at level n . It suggests the FFT method could give a better performance, compared with the other methods, when it is desired to compute all coefficients from $\binom{n}{0}$ up to $\binom{n}{n}$.

In order to inquire into this question, the second experimentation presents the average time to compute the list of all binomial coefficients for a given n . For each n ranging from 2 to 100, we computed all coefficients $\binom{n}{k}$ for a given n , 5000 times, and computed the median of averages, using the same procedure as described in the previous test. The results in Fig. 6 show that in this scenario, the method proposed here outperform the iterative method `ybincoeff`, but the approach using the gamma functions still lead to the best performance.

In the last experimentation, we have evaluated the average error in computing the binomial coefficients using the standard arithmetic precision in GNU Octave³.

³Octave documentation states that “by default numeric constants are represented within Octave by IEEE 754 double precision (binary64) floating-point format” [1] which uses 1 bit for sign, 11 to represent the exponent and 52 to the significant (53-bit precision) [12].

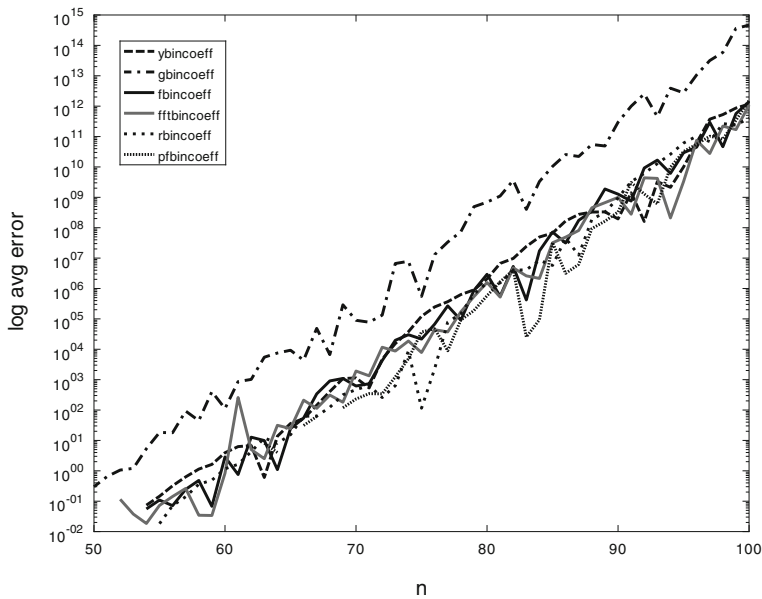


Fig. 7 Average error in computing the binomial coefficients for large values of n using standard precision arithmetic

The test results are presented in Fig. 7. In order to compute the errors, we used the *bigfloat*⁴ package in Python to implement the iterative method in (10) and the method proposed here in (22). All coefficients up to $n = 100$ were computed using 256-bit precision, providing reliable results, confirmed by comparing with the results in [2, 3]. The average error in the binomial coefficient calculation was computed for each method. We observe that the fastest method, based on gamma functions, provided larger error than all the others. The method proposed here showed similar errors when compared with the remaining methods (except *gbincoeff*).

6 Conclusion

All methods analyzed in this paper are able to compute the binomial coefficients, using standard precision, up to $n = 47$, with zero error. The methods proposed here might not be the fastest to compute individual coefficients, but it is the best approach to compute all the range of coefficients for a given level n , since it achieves good performance and precision.

⁴The *bigfloat* package in Python is a wrapper for the GNU MPFR library for arbitrary-precision floating-point reliable arithmetic.

References

1. Eaton, J.W.: Gnu octave (version 5.1.0). <https://octave.org/doc/v5.1.0/Numeric-Data-Types.html> (2018)
2. Goetgheluck, P.: Computing binomial coefficients. *Am. Math. Mon.* **94**(4), 360 (1987)
3. Wolfram Research, Inc. Wolfram—alpha. <https://www.wolframalpha.com/>
4. Keprt, A.: Simple and fast computation of binomial coefficients. In: Wofex 2006, pp. 346–351, VŠB Technical University, Ostrava, Czech Republic, ISBN 80-248-1152-9 (2006)
5. Knuth, D.E.: The Art of Computer Programming: Volume 1: Fundamental Algorithms. Addison Wesley. ISBN 0-201-89683-4 (1997)
6. Koepf, W.: Hypergeometric Summation: An Algorithmic Approach to Summation and Special Function Identities (Advanced Lectures in Mathematics). Vieweg Teubner Verlag. ISBN 3528069503 (1998)
7. Manolopoulos, Y.: Binomial coefficient computation: recursion or iteration? *ACM SIGCSE Bullet.* **34**(4), 65 (2002). <https://doi.org/10.1145/820127.820168>
8. McKeeman, B.: Matlab performance measurement. Technical report, Mathworks. <https://www.mathworks.com/matlabcentral/fileexchange/18510-matlab-performance-measurement>. [Online; accessed 03-Feb-2020] (2008)
9. Rolfe, T.: Binomial coefficient recursion. *ACM SIGCSE Bullet.* **33**(2), 35 (2001). <https://doi.org/10.1145/571922.571950>
10. Wells, D.: The Penguin Dictionary of Curious and Interesting Numbers. Penguin Books. ISBN 0140080295 (1987)
11. Widrow, B., Kollár, I.: Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications. Cambridge University Press. ISBN 9780521886710 (2008)
12. Wikipedia contributors. Double-precision floating-point format - Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Double-precision_floating-point_format. [Online; accessed 17-May-2019] (2019)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.