

Faster approach to evaluate nested sums of polynomials

Pedro Cleto
cl3t0.sh@gmail.com

January 2020

1 Abstract

In this paper we show 3 algorithms to evaluate nested sums of polynomials. The last one is the main work of this paper. Define f and S as

$$f(x) := a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$
$$S(a, b, k) := \sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} f(i_k).$$

The first algorithm consists in evaluate in a naive way, applying summations using recursion. Let's call it "Naive sum". The second one is what probably most of the people thought to use because it's not hard to think and hugely optimizes running time and depends almost equally on $b - a$, k and n . Let's call it "Memoized sum". Finally, the third algorithm uses binomial coefficients and matrices operations to depends mostly on n . It means that this algorithm is faster for bigger k and $b - a$. Let's call it BCBTS (Binomial Coefficients Base Transformation Sum).

2 Algorithm 1: Naive Sum

2.1 Code and complexity analysis

The algorithm is brute force so it doesn't require proof. Python is not the fastest programming language but we are using it only for comparison purposes and to avoid float precision problems.

```

1 from typing import List
2
3
4 def nested_sum_of_polynomial(a: int, b: int, k: int, f_coefs: List[float])
   -> float:
5     def func(x: int) -> int:
6         # horner's method for polynomial evaluation
7         acc = f_coefs[-1]
8         for coef in f_coefs[-2::-1]:
9             acc *= x
10            acc += coef
11        return acc
12
13    def recursive_sum(x: int, k: int) -> float:
14        res = (
15            func(x) if k == 0 else sum(recursive_sum(i, k - 1) for i in
range(a, x + 1))
16        )
17        return res
18
19    return recursive_sum(b, k)
20
21
22 result = nested_sum_of_polynomial(
23     a=1,
24     b=100,
25     k=4,
26     f_coefs=[5, -1, 1, -5, -3, 3], # Put your function coefficients here
27 )
28
29 print(result)

```

Note that it's a simple and short code but extremely inefficient due to function recursion. Doing some analysis we conclude that the time complexity of this algorithm is $O\left(n \binom{b-a+k-1}{k}\right)$ given a, b, k, n . It's not precise because Python don't do exactly what we order. Because of it we are going to do some practical comparison at the end.

3 Algorithm 2: Memoized Sum

3.1 Code and complexity analysis

Here we are going to rewrite the same algorithm as before but with memoization.

```

1 from typing import List
2
3
4 def nested_sum_of_polynomial(a: int, b: int, k: int, f_coefs: List[float])
   -> float:
5     mem = {}
6
7     def func(x: int) -> int:
8         # horner's method for polynomial evaluation
9         acc = f_coefs[-1]
10        for coef in f_coefs[-2::-1]:
11            acc *= x
12            acc += coef
13        return acc
14
15    def recursive_sum(x: int, k: int) -> float:
16        if mem.get((x, k)) is not None:
17            return mem[(x, k)]
18
19        res = (
20            func(x) if k == 0 else sum(recursive_sum(i, k - 1) for i in
range(a, x + 1))
21        )
22        mem[(x, k)] = res
23        return res
24
25    return recursive_sum(b, k)
26
27
28 result = nested_sum_of_polynomial(
29     a=1,
30     b=100,
31     k=4,
32     f_coefs=[5, -1, 1, -5, -3, 3], # Put your function coefficients here
33 )
34
35 print(result)

```

This simple technique drastically speed things up. Doing some analysis we conclude that the time complexity of this algorithm is $O((b - a)(n + k))$ given a, b, k, n . Again, it's not precise.

4 Algorithm 3: BCBTS (Binomial Coefficients Base Transformation Sum)

4.1 Presentation

BCBTS consists in rewriting S through binomial coefficients.

$$S(a, b, k) = \begin{bmatrix} \binom{b+k-a}{k} & \binom{b+1+k-a}{1+k} & \cdots & \binom{b+n+k-a}{n+k} \end{bmatrix} \Omega \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

where Ω is an $(n+1) \times (n+1)$ upper triangular matrix defined by the recurrence

- $\Omega_{1,1} := 1$;
- For all $i, j \in \mathbb{Z}_+$ where $j < i \leq n+1$, $\Omega_{i,j} := 0$; (upper triangular matrix)
- For all $i, j \in \mathbb{N}$ where $i, j \leq n$, $\Omega_{i+1,j+1} := \Omega_{i,j}i - \Omega_{i+1,j}(i+1-a)$.

For example, when $a = 1$ we get

$$\Omega = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & -1 & 1 & -1 & 1 & \cdots \\ 0 & 0 & 2 & -6 & 14 & -30 & \cdots \\ 0 & 0 & 0 & 6 & -36 & 150 & \cdots \\ 0 & 0 & 0 & 0 & 24 & -240 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 120 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

4.2 Code and complexity analysis

```

1 import numpy as np
2 from typing import List
3
4
5 def nested_sum_of_polynomial(a: int, b: int, k: int, f_coefs: List[float])
  -> float:
6     n = len(f_coefs) - 1

```

```

7     coefs_vec = np.array([0, *f_coefs]).reshape(-1, 1)
8
9     # calculating pascal triangle elements
10    last_val = 1 # (b + k - a choose 0)
11    for i in range(k):
12        last_val *= (b + k - a - i) / (i + 1)
13
14    # calculating base vector values
15    base = [0, last_val]
16    # (b + k - a choose k)
17    for i in range(2, n + 2):
18        base.append(base[i - 1] * (b + k + i - 1 - a) / (k + i - 1))
19    base_vec = np.array(base).reshape(1, -1)
20
21    # initializing omega matrix
22    omega = np.zeros(shape=(n + 2, n + 2))
23    omega[1, 1] = 1
24
25    # calculating omega matrix
26    for j in range(2, n + 2):
27        for i in range(1, j + 1):
28            omega[i, j] = (i - 1) * omega[i - 1, j - 1] - (i - a) * omega[
29                i, j - 1]
30
31    result = np.linalg.multi_dot([base_vec, omega, coefs_vec])
32    return result[0, 0]
33
34 result = nested_sum_of_polynomial(
35     a=1,
36     b=100,
37     k=4,
38     f_coefs=[5, -1, 1, -5, -3, 3], # Put your function coefficients here
39 )
40
41 print(int(result))

```

This time it is possible to notice a more extensive code and more operations than algorithm 2 for small k and b . Doing some analysis we conclude that the time complexity of this algorithm is $O(k + n^2)$ given a, b, k, n . It means that it does not depend on b and a and depends linearly on k . This algorithm seems ideal for small n , big k and arbitrarily large $b - a$.

4.3 Proof

In order to represent summations in an alternative way we need to show some intermediate results.

Theorem 1 (Pascal's rule). *For all $i, j \in \mathbb{N}$ where $j < i$ it holds that*

$$\binom{i}{j} + \binom{i}{j+1} = \binom{i+1}{j+1}.$$

Proof. By developing and manipulating the binomial coefficients, we get

$$\begin{aligned} \binom{i}{j} + \binom{i}{j+1} &= \frac{i!}{j!(i-j)!} + \frac{i!}{(j+1)!(i-(j+1))!} \\ &= \frac{i!(j+1)}{(j+1)!(i-j)!} + \frac{i!(i-j)}{(j+1)!(i-j)!} \\ &= \frac{i!(i+1)}{(j+1)!(i-j)!} \\ &= \frac{(i+1)!}{(j+1)!((i+1)-(j+1))!}. \end{aligned}$$

Therefore,

$$\boxed{\binom{i}{j} + \binom{i}{j+1} = \binom{i+1}{j+1}}.$$

□

Theorem 2 (Hockey-stick identity). *For all $n, a, b \in \mathbb{N}$ where $b \geq a$ it holds that*

$$\sum_{i=a}^b \binom{i+n-a}{n} = \binom{b+n+1-a}{n+1}.$$

Proof. For $b = a$,

$$\begin{aligned}
\sum_{i=a}^a \binom{i+n-a}{n} &= \binom{n}{n} \\
&= 1 \\
&= \binom{n+1}{n+1} \\
&= \binom{b+n+1-a}{n+1}.
\end{aligned}$$

Therefore,

$$\sum_{i=a}^a \binom{i+n-a}{n} = \binom{b+n+1-a}{n+1}.$$

That is, it holds for $b = a$. Suppose it holds for some $b \in \mathbb{N}$ where $b \geq a$. Therefore,

$$\sum_{i=a}^b \binom{i+n-a}{n} = \binom{b+n+1-a}{n+1}.$$

Adding $\binom{(b+1)+n-a}{n}$ to both sides we get

$$\sum_{i=a}^{b+1} \binom{i+n-a}{n} = \binom{b+n+1-a}{n+1} + \binom{b+n+1-a}{n}.$$

By the theorem 1,

$$\boxed{\sum_{i=a}^{b+1} \binom{i+n-a}{n} = \binom{(b+1)+n-a}{n+1}}.$$

Therefore, it holds for $b+1$. By the principle of finite induction, it holds for all $b \in \mathbb{N}$ where $b \geq a$. □

Theorem 3. For all $n, a, b \in \mathbb{N}$ where $b \geq a$ and $k \in \mathbb{Z}_+$ it holds that

$$\sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} \binom{i_k+n-a}{n} = \binom{b+n+k-a}{n+k}.$$

Proof. The case $k = 1$ is already proven by the theorem 2. Suppose that it holds for some $k \in \mathbb{Z}_+$. Therefore,

$$\sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} \binom{i_k + n - a}{n} = \binom{b + n + k - a}{n + k}.$$

By summing the expressions for b from 1 to m on both sides, we get

$$\sum_{b=1}^m \left(\sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} \binom{i_k + n - a}{n} \right) = \sum_{b=1}^m \binom{b + n + k - a}{n + k}.$$

Again using the theorem 2, we know that

$$\sum_{b=1}^m \sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} \binom{i_k + n - a}{n} = \binom{m + n + k + 1 - a}{n + k + 1}.$$

Changing the notation, we get

$$\boxed{\sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \sum_{i_3=a}^{i_2} \cdots \sum_{i_{k+1}=a}^{i_k} \binom{i_{k+1} + n - a}{n} = \binom{b + n + (k + 1) - a}{n + (k + 1)}}.$$

So it holds for $k+1$. Therefore, by the principle of finite induction, it holds for all $k \in \mathbb{Z}_+$. \square

Lemma 4. Let $n \in \mathbb{N}$. Then there exists a polynomial with rational coefficients $p_n \in \mathbb{Q}[x]$, such that p_n has degree n and for all $x \in \mathbb{N}$ where $x \geq a$, it holds that $p_n(x) = \binom{x+n-a}{n}$.

Proof. For $n = 0$, define $p_0(x) := 1$. Note that

$$\binom{x-a}{0} = \frac{(x-a)!}{0!(x-a)!} = 1.$$

Therefore, for all $x \in \mathbb{N}$ where $x \geq a$, $p_0(x) = \binom{x-a}{0}$, so it holds for $n = 0$. Suppose it holds for some $n \in \mathbb{N}$. Note that

$$\begin{aligned}
\binom{x+n-a}{n} &= \frac{(x+n-a)!}{n!(x+n-a-n)!} \\
\frac{x+n+1-a}{n+1} \binom{x+n-a}{n} &= \frac{(x+n+1-a)!}{(n+1)!(x+n-a-n)!} \\
\frac{x+n+1-a}{n+1} \binom{x+n-a}{n} &= \frac{(x+n+1-a)!}{(n+1)!((x+n+1-a)-(n+1))!} \\
\left(\frac{1}{n+1}x + \frac{n+1-a}{n+1}\right) \binom{x+n-a}{n} &= \binom{x+n+1-a}{n+1}.
\end{aligned}$$

Define

$$p_{n+1}(x) := \left(\frac{1}{n+1}x + \frac{n+1-a}{n+1}\right) p_n(x).$$

Therefore, for all $x \in \mathbb{N}$ where $x \geq a$, $p_{n+1}(x) = \binom{x+n+1-a}{n+1}$, so it holds for $n+1$. By the principle of finite induction, it holds for all $n \in \mathbb{N}$. □

Lemma 5. For all $i, a, x \in \mathbb{N}$ where $x \geq a$ it holds that

$$(i+1) \binom{x+i+1-a}{i+1} - (i+1-a) \binom{x+i-a}{i} = x \binom{x+i-a}{i}.$$

Proof. By developing and manipulating the binomial coefficients, we get

$$\begin{aligned}
(i+1) \binom{x+i+1-a}{i+1} - (i+1-a) \binom{x+i-a}{i} &= (i+1) \frac{(x+i+1-a)!}{(i+1)!(x-a)!} - (i+1-a) \frac{(x+i-a)!}{i!(x-a)!} \\
&= (x+i+1-a) \frac{(x+i-a)!}{i!(x-a)!} - (i+1-a) \frac{(x+i-a)!}{i!(x-a)!} \\
&= x \frac{(x+i-a)!}{i!(x-a)!}.
\end{aligned}$$

Therefore,

$$(i+1) \binom{x+i+1-a}{i+1} - (i+1-a) \binom{x+i-a}{i} = x \binom{x+i-a}{i}.$$

□

Theorem 6. For every $n \in \mathbb{N}$ let $p_0, p_1, p_2, \dots, p_n$ such that p_i has degree i for all $i \in \mathbb{N}$ where $i \leq n$. $P = \{p_0, p_1, p_2, \dots, p_n\}$ is linearly independent in $P_n(\mathbb{C})$.

Proof. For $n = 0$,

$$P = \{p_0\}.$$

Since the set has only one element, it is linearly independent. Suppose that P is linearly independent for some $n \in \mathbb{N}$. Let $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n, \alpha_{n+1} \in \mathbb{C}$ such that

$$\alpha_0 p_0 + \alpha_1 p_1 + \dots + \alpha_n p_n + \alpha_{n+1} p_{n+1} = 0$$

$$\alpha_0 p_0 + \alpha_1 p_1 + \dots + \alpha_n p_n = -\alpha_{n+1} p_{n+1}$$

The left-hand side has a maximum degree of n . Note that if $\alpha_{n+1} \neq 0$ the right-hand side of the equation will have degree $n+1$, which is absurd. Therefore $\alpha_{n+1} = 0$. Applying this new information to the equation, we get

$$\alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 + \dots + \alpha_n p_n = 0.$$

By hypothesis, this equation has a unique solution in $\alpha_0 = \alpha_1 = \dots = \alpha_n = 0$. Thus, it is concluded that P is also linearly independent for $n+1$. By the principle of finite induction, P is linearly independent for all $n \in \mathbb{N}$.

□

Corollary 6.1. For all $n \in \mathbb{N}$, $A = \{1, x, x^2, x^3, \dots, x^n\}$ is linearly independent in $P_n(\mathbb{C})$.

Corollary 6.2. Let $\binom{x+i-a}{i}$ be understood as the polynomial described in lemma 4. For all $n \in \mathbb{N}$, $B = \left\{ \binom{x-a}{0}, \binom{x+1-a}{1}, \binom{x+2-a}{2}, \dots, \binom{x+n-a}{n} \right\}$ is linearly independent in $P_n(\mathbb{C})$.

Proof. The proof is given by the theorem 6. □

Given that $n \in \mathbb{N}$ is fixed, we have,

$$A = \{1, x, x^2, x^3, \dots, x^n\}$$

$$B = \left\{ \binom{x-a}{0}, \binom{x+1-a}{1}, \binom{x+2-a}{2}, \dots, \binom{x+n-a}{n} \right\}$$

These, according to the corollaries 6.1 and 6, are bases of the same vector space $P_n(\mathbb{C})$, therefore there exists a matrix Ω that performs the base change from A to B .

Theorem 7. Ω is a matrix defined by

- (a) $\Omega_{1,1} := 1$;
- (b) For all $i, j \in \mathbb{Z}_+$ where $i > j$, $\Omega_{i,j} := 0$; (Ω is an upper triangular matrix)
- (c) For all $i, j \in \mathbb{N}$, $\Omega_{i+1,j+1} := \Omega_{i,j}i - \Omega_{i+1,j}(i+1-a)$.

Proof. To make Ω explicit, it is necessary to analyze how each element of the set A is written in the base B , representing each column of Ω . Therefore, it is known by the definition of Ω that for every $j \in \mathbb{Z}_+$, where $j \leq n+1$, it holds that

$$x^{j-1} = \sum_{i=0}^n \Omega_{i+1,j} \binom{x+i-a}{i}.$$

As $j-1 \leq n$, we can separate the summation into two parts. That is,

$$x^{j-1} = \sum_{i=0}^{j-1} \Omega_{i+1,j} \binom{x+i-a}{i} + \sum_{i=j}^n \Omega_{i+1,j} \binom{x+i-a}{i}.$$

Suppose that for some $i > j$, $\Omega_{i,j} \neq 0$. Note that, because of the assumption, some term of the second summation has degree bigger than $j-1$, which is absurd because the left-hand

side of the equation has degree $j - 1$. Therefore, for $i > j$, $\Omega_{i,j} = 0$, proving (b). Hence, x^{j-1} can be written as

$$x^{j-1} = \sum_{i=0}^{j-1} \Omega_{i+1,j} \binom{x+i-a}{i}. \quad (1)$$

Choosing $j = 1$ we have,

$$\begin{aligned} x^0 &= \sum_{i=0}^0 \Omega_{i+1,1} \binom{x+i-a}{i} \\ 1 &= \Omega_{1,1} \binom{x-a}{0} \\ 1 &= \Omega_{1,1} \end{aligned}$$

proving (a). Multiplying both sides of equation 1 by x we get

$$x^j = \sum_{i=0}^{j-1} \Omega_{i+1,j} x \binom{x+i-a}{i}.$$

Applying lemma 5 we get

$$\begin{aligned} x^j &= \sum_{i=0}^{j-1} \Omega_{i+1,j} \left((i+1) \binom{x+i+1-a}{i+1} - (i+1-a) \binom{x+i-a}{i} \right) \\ &= \sum_{i=0}^{j-1} \Omega_{i+1,j} (i+1) \binom{x+i+1-a}{i+1} - \sum_{i=0}^{j-1} \Omega_{i+1,j} (i+1-a) \binom{x+i-a}{i} \\ &= \sum_{i=1}^j \Omega_{i,j} i \binom{x+i-a}{i} - \sum_{i=0}^{j-1} \Omega_{i+1,j} (i+1-a) \binom{x+i-a}{i} \end{aligned}$$

Since $\Omega_{0,j} 0 \binom{x+0-a}{0} = 0$ and $\Omega_{j+1,j} (j+1-a) \binom{x+j-a}{j} = 0$, it is possible to change the lower limit of the first sum to 0 and the upper limit of the second sum to j because both added terms are null. That is,

$$\begin{aligned}
x^j &= \sum_{i=0}^j \Omega_{i,j} i \binom{x+i-a}{i} - \sum_{i=0}^j \Omega_{i+1,j} (i+1-a) \binom{x+i-a}{i} \\
&= \sum_{i=0}^j (\Omega_{i,j} i - \Omega_{i+1,j} (i+1-a)) \binom{x+i-a}{i}.
\end{aligned}$$

By the equation 1 but for $j-1 \rightsquigarrow j$ it is known that

$$x^j = \sum_{i=0}^j \Omega_{i+1,j+1} \binom{x+i-a}{i}.$$

By the corollary 6.2 it holds that

$$\Omega_{i+1,j+1} = \Omega_{i,j} i - \Omega_{i+1,j} (i+1-a).$$

Proving (c). □

Knowing this law of formation, it is possible to calculate all the terms of Ω . To use our new information in the original problem, we can start by applying Ω to the coefficients of f to make the change from A to B , resulting in

$$\begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix} = \Omega \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}.$$

By multiplying our new coefficients by the binomial coefficients we get f in the form

$$f(x) = \begin{bmatrix} \binom{x-a}{0} & \binom{x+1-a}{1} & \binom{x+2-a}{2} & \binom{x+3-a}{3} & \cdots & \binom{x+n-a}{n} \end{bmatrix} \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}.$$

To calculate $S(a, b, k)$ we apply the definition and we get

$$S(a, b, k) = \sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} f(i_k)$$

$$S(a, b, k) = \sum_{i_1=a}^b \sum_{i_2=a}^{i_1} \cdots \sum_{i_k=a}^{i_{k-1}} \begin{bmatrix} \binom{i_k-a}{0} & \binom{i_k+1-a}{1} & \binom{i_k+2-a}{2} & \binom{i_k+3-a}{3} & \cdots & \binom{i_k+n-a}{n} \end{bmatrix} \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}.$$

Note that when we these products, we get a linear combination of binomial coefficients and will be possible to apply the theorem 3, getting

$$S(a, b, k) = \begin{bmatrix} \binom{b+k-a}{k} & \binom{b+1+k-a}{1+k} & \binom{b+2+k-a}{2+k} & \binom{b+3+k-a}{3+k} & \cdots & \binom{b+n+k-a}{n+k} \end{bmatrix} \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}.$$

Substituting the vector of coefficients ω we get

$$S(a, b, k) = \begin{bmatrix} \binom{b+k-a}{k} & \binom{b+1+k-a}{1+k} & \binom{b+2+k-a}{2+k} & \binom{b+3+k-a}{3+k} & \cdots & \binom{b+n+k-a}{n+k} \end{bmatrix} \Omega \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}.$$

■

5 Análise de eficiência

Agora iremos fazer testes utilizando os algoritmos 1, 2 e 3 apresentados neste artigo. Essa diferença é tão grande que para $(a, b, k, n) = (1, 100, 6, 5)$ obtemos as seguintes medições de tempo.

Algoritmo	Tempo de execução
1	16,472 segundos
2	0,007 segundos
3	0,007 segundos

Aumentando apenas uma unidade em k ($k = 7$) obtém-se as seguintes medições.

Algoritmo	Tempo de execução
1	240,762 segundos
2	0,007 segundos
3	0,007 segundos

(Nessa parte ainda não terminei todas as comparações que quero fazer)

6 Conclusão

Depois da demonstração matemática e das comparações de eficiência foi possível conhecer um algoritmo extremamente mais eficiente para o cálculo de somatórios recursivos, ganhando uma grande quantidade de tempo.