Hypergeometric Connections, Taylor Series, and Recurrence Formulations for Lemniscate Functions

Dr. Stefan Gruenwald & rAImanujan Google Chrome AI

November 4, 2026

1 Introduction and Hypergeometric Alignment

Generalized hypergeometric functions are solutions to linear differential equations. In contrast, $\mathrm{sl}(x)$ is a doubly periodic elliptic function that satisfies the **nonlinear first-order differential equation**:

$$(\mathrm{sl}'(x))^2 = 1 - \mathrm{sl}^4(x) \tag{1}$$

Because of its double periodicity and nonlinearity, the forward lemniscate sine function $\mathrm{sl}(x)$ cannot be expressed as a single ${}_pF_q$ series. However, the standard circular sine function possesses a clean forward transformation to a ${}_0F_1$ generalized hypergeometric series:

$$\sin(x) = x \cdot {}_0F_1\left(\frac{3}{2}; -\frac{x^2}{4}\right) \tag{2}$$

While you cannot write $\mathrm{sl}(x)$ as a basic hypergeometric function, its Taylor series expansion coefficients can still be computed sequentially, and it can be evaluated globally using Fourier series representations of Jacobi elliptic functions. However, an exact analytical connection exists between their respective **inverse functions**, as shown in Table 1.

Table 1: Comparison of Circular and Lemniscate Functions

Property	**Circular Version**	**Lemniscate Version**
Forward Function	$\sin(x) = x \cdot {}_0F_1\left(\frac{3}{2}; -\frac{x^2}{4}\right)$	$\mathrm{sl}(x)$ *(No single ${}_pF_q$ form)*
Inverse Integral	$\arcsin(x) = \int_0^x \frac{dt}{\sqrt{1-t^2}}$	$\mathrm{arcsl}(x) = \int_0^x \frac{dt}{\sqrt{1-t^4}}$
Inverse Hypergeometric	$\arcsin(x) = x \cdot {}_2F_1\left(\frac{1}{2}, \frac{1}{2}; \frac{3}{2}; x^2\right)$	$\mathrm{arcsl}(x) = x \cdot {}_2F_1\left(\frac{1}{2}, \frac{1}{4}; \frac{5}{4}; x^4\right)$

2 The Lemniscate Sine Function: $\mathbf{sl}(x)$

Differentiating the first-order relation $(\mathrm{sl}'(x))^2 = 1 - \mathrm{sl}^4(x)$ yields the secondary nonlinear governing equation:

$$2\mathrm{sl}'(x)\mathrm{sl}''(x) = -4\mathrm{sl}^3(x)\mathrm{sl}'(x) \implies \mathrm{sl}''(x) = -2\mathrm{sl}^3(x) \tag{3}$$

subject to initial parameters $\mathrm{sl}(0) = 0$ and $\mathrm{sl}'(0) = 1$. Due to its odd symmetry and a periodicity gap enforced by the $\mathrm{sl}^4(x)$ term, its power series takes the form:

$$\mathrm{sl}(x) = \sum_{n=0}^{\infty} c_n x^{4n+1} = c_0 x + c_1 x^5 + c_2 x^9 + c_3 x^{13} + \dots \tag{4}$$

where $c_0 = 1$. The resulting exact analytical recurrence relation for the coefficients is:

$$c_{k+1} = \frac{-1}{2(k+1)(4k+5)} \sum_{i+j+m=k} c_i c_j c_m \tag{5}$$

The series converges for $|x| < \frac{\varpi}{2} \approx 1.311$, where ϖ is the lemniscate constant.

2.1 Step-by-Step Derivation of $\mathbf{sl}(x)$ Coefficients

Using $c_0 = 1$, we compute the first few non-zero terms sequentially:

- **For c_1 ($k = 0$):** The only combination for $i + j + m = 0$ is $(0, 0, 0)$.

$$c_1 = \frac{-1}{2(1)(5)} \cdot (c_0 \cdot c_0 \cdot c_0) = \frac{-1}{10} \cdot (1)^3 = -\frac{1}{10} \tag{6}$$

- **For c_2 ($k = 1$):** The combinations for $i + j + m = 1$ are $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$.

$$\sum c_i c_j c_m = 3(c_1 c_0 c_0) = 3\left(-\frac{1}{10}\right) = -\frac{3}{10} \tag{7}$$

$$c_2 = \frac{-1}{2(2)(9)} \cdot \left(-\frac{3}{10}\right) = \frac{3}{360} = \frac{1}{120} \tag{8}$$

- **For c_3 ($k = 2$):** The combinations for $i + j + m = 2$ are $(2, 0, 0)$ and its 3 permutations, plus $(1, 1, 0)$ and its 3 permutations.

$$\sum c_i c_j c_m = 3(c_2 c_0^2) + 3(c_1^2 c_0) = 3\left(\frac{1}{120}\right) + 3\left(-\frac{1}{10}\right)^2 = \frac{1}{40} + \frac{3}{100} = \frac{11}{200} \tag{9}$$

$$c_3 = \frac{-1}{2(3)(13)} \cdot \left(\frac{11}{200}\right) = -\frac{11}{15,600} \tag{10}$$

Thus, the expansion yields:

$$\mathrm{sl}(x) = x - \frac{1}{10}x^5 + \frac{1}{120}x^9 - \frac{11}{15,600}x^{13} + \frac{211}{4,536,000}x^{17} - \dots \tag{11}$$

3 The Lemniscate Cosine Function: $\mathbf{cl}(x)$

The partner function $\mathrm{cl}(x)$ maps onto a related system. Differentiating its first-order identity $[\mathrm{cl}'(x)]^2 = 1 - \mathrm{cl}^4(x)$ yields the second-order nonlinear differential equation:

$$\mathrm{cl}''(x) = -2\mathrm{cl}^3(x) \tag{12}$$

under initial conditions $\mathrm{cl}(0) = 1$ and $\mathrm{cl}'(0) = 0$. Representing the expansion as an even series $\mathrm{cl}(x) = \sum_{k=0}^{\infty} d_k x^{2k}$ yields the exact recurrence formula:

$$d_{k+1} = \frac{-2}{(2k+1)(2k+2)} \sum_{i+j+m=k} d_i d_j d_m \tag{13}$$

where $d_0 = 1$.

3.1 Step-by-Step Derivation of cl(x) Fractions

Using $d_0 = 1$, we compute the explicit fractional terms sequentially:

- **For d_1 ($k = 0$):** The only integer triplet that sums to 0 is $(0, 0, 0)$.

$$d_1 = \frac{-2}{(1)(2)} \cdot (d_0 \cdot d_0 \cdot d_0) = -1 \cdot (1)^3 = -1 \tag{14}$$

- **For d_2 ($k = 1$):** The triplets that sum to 1 are $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$.

$$\sum d_i d_j d_m = 3(d_1 d_0^2) = 3(-1)(1)^2 = -3 \tag{15}$$

$$d_2 = \frac{-2}{(3)(4)} \cdot (-3) = \frac{6}{12} = \frac{1}{2} \tag{16}$$

- **For d_3 ($k = 2$):** The triplets that sum to 2 are $(2, 0, 0)$ and its 3 permutations, plus $(1, 1, 0)$ and its 3 permutations.

$$\sum d_i d_j d_m = 3(d_2 d_0^2) + 3(d_1^2 d_0) = 3\left(\frac{1}{2}\right)(1)^2 + 3(-1)^2(1) = \frac{3}{2} + 3 = \frac{9}{2} \tag{17}$$

$$d_3 = \frac{-2}{(5)(6)} \cdot \left(\frac{9}{2}\right) = -\frac{18}{60} = -\frac{3}{10} \tag{18}$$

- **For d_4 ($k = 3$):** The triplets that sum to 3 are $(3, 0, 0)$ and its permutations, $(2, 1, 0)$ and its permutations, and $(1, 1, 1)$.

$$\sum d_i d_j d_m = 3(d_3 d_0^2) + 6(d_2 d_1 d_0) + d_1^3 = 3\left(-\frac{3}{10}\right) + 6\left(\frac{1}{2}\right)(-1) + (-1)^3 = -\frac{49}{10} \tag{19}$$

$$d_4 = \frac{-2}{(7)(8)} \cdot \left(-\frac{49}{10}\right) = \frac{98}{560} = \frac{7}{40} \tag{20}$$

Continuing this fractional arithmetic iteratively yields the full standalone expansion:

$$\text{cl}(x) = 1 - x^2 + \frac{1}{2}x^4 - \frac{3}{10}x^6 + \frac{7}{40}x^8 - \frac{61}{600}x^{10} + \frac{71}{1200}x^{12} - \frac{179}{5200}x^{14} + \dots \tag{21}$$

4 Algorithmic Implementations in Python

Below are the two program components discussed to systematically model and verify these expansions.

4.1 Standard Floating-Point Iteration Script

This basic script demonstrates code efficiency using nested index boundaries to map the triple convolution.

```
def get_lemniscate_cos_coefficients(max_k):
    """
    Calculates the Taylor series coefficients of cl(x) up to x^(2*max_k).
    cl(x) = d_0 + d_1*x^2 + d_2*x^4 + d_3*x^6 + ...
    """
    d = [0.0] * (max_k + 1)
    d[0] = 1.0

    for k in range(0, max_k):
```

```
        triple_sum = 0.0
        for i in range(k + 1):
            for j in range(k + 1 - i):
                m = k - i - j
                triple_sum += d[i] * d[j] * d[m]

        numerator = -2.0 * triple_sum
        denominator = (2 * k + 1) * (2 * k + 2)
        d[k + 1] = numerator / denominator

    return d

# Generate up to x^14
num_terms = 8
coefficients = get_lemniscate_cos_coefficients(num_terms - 1)

print("Taylor Series Expansion for cl(x):")
for k, coef in enumerate(coefficients):
    print(f"d_{k} (coefficient of x^{2*k}) = {coef:.6f}")
```

Listing 1: Floating point expansion for cl(x)

4.2 Exact Symbolic Fraction Generation Script

By integrating SymPy's `Rational` classes, rounding errors are entirely avoided, revealing the raw algebraic fraction format.

```
from sympy import Rational

def get_lemniscate_cos_fractions(max_k):
    """
    Calculates the exact fractional Taylor series coefficients of cl(x)
    up to x^(2*max_k) using SymPy Rational numbers.
    """
    d = [Rational(0, 1)] * (max_k + 1)
    d[0] = Rational(1, 1)

    for k in range(0, max_k):
        triple_sum = Rational(0, 1)
        for i in range(k + 1):
            for j in range(k + 1 - i):
                m = k - i - j
triple_sum += d[i] * d[j] * d[m]
numerator = Rational(-2, 1) * triple_sum
denominator = Rational((2 * k + 1) * (2 * k + 2), 1)
d[k + 1] = numerator / denominator
return d

Compute exact fractions up to x^14 (k = 7)

num_terms = 8
exact_fractions = get_lemniscate_cos_fractions(num_terms - 1)
print("Exact Fractional Taylor Coefficients for cl(x):")
for k, frac in enumerate(exact_fractions):
print(f"d_{k} (coefficient of x^{2*k:02d}) = {frac}")
```

Listing 2: Symbolic Rational expansion for cl(x) using SymPy