Series misdemeanors

David R. Stoutemyer*

Abstract

Puiseux series are power series in which the exponents can be fractional and/or negative rational numbers. Several computer algebra systems have one or more built-in or loadable functions for computing truncated Puiseux series – perhaps generalized to allow coefficients containing functions of the series variable that are dominated by any power of that variable, such as logarithms and nested logarithms of the series variable. Some computer-algebra systems also offer functions that can compute more-general truncated recursive *hierarchical* series. However, for all of these kinds of truncated series there are important implementation details that haven't been addressed before in the published literature and in current implementations.

For implementers this article contains ideas for designing more convenient, correct, and efficient implementations or improving existing ones. For users, this article is a warning about some of these limitations. More specifically, this article discusses issues such as

- avoiding unnecessary restrictions such as prohibiting negative or fractional requested orders,
- the pros and cons of displaying results with explicit infectious error terms of the form $o(\ldots)$, $O(\ldots)$, and/or $\Theta(\ldots)$,
- efficient data structures, and
- algorithms that efficiently give users exactly the order or number of nonzero terms they request.

Most of the ideas in this article have been implemented in the computer-algebra within the TI-Nspire calculator, Windows and Macintosh products.

1 Introduction

Here is a conversation recently overheard at a car-rental desk:

> *Customer:* "I followed your directions of three right turns to get on the highway, but that put me in a fenced corner from which I could only turn right, bringing me back to where I started!"
>
> *Agent*: "Make your first right turn *after* exiting the rental car lot."

The original directions were correct, but incomplete.

The same was true of published algorithms for truncated Puiseux series. After reading all that I could find about such algorithms, I implemented them for the computer algebra embedded in the TI Nspire™ hand held graphing calculator, which also runs on PC and Macintosh computers.

*dstout at hawaii dot edu

Testing revealed some incorrect results due to ignorance about some important issues. Results for other series implementations, reveal that their implementers have made similar oversights.

It required a significant effort to determine how to overcome these difficulties. This article is intended as a warning for users of implementations that exhibit the flaws – and as suggestions to implementers for repairing those flaws or avoiding them in new implementations. Most of the ideas here are implemented in TI-Nspire.

Additional issues for truncated and infinite series are described in a sequel to this article [11].

For real-world problems, exact closed-form symbolic solutions are less frequently obtainable than are various symbolic series solutions. Therefore in practice, symbolic series are among the most important features of computer algebra systems.

Almost all computer algebra systems have a function that produces at least truncated Taylor series. Iterated differentiation followed by substitution of the expansion point provides a very compact implementation. However, it can consume time and data memory that grows painfully with the requested order of the result. Knuth [4] presents algorithms that are significantly more efficient for addition, multiplication, raising to a numeric power, exponentials, logarithms, composition and reversion. He also suggests how to derive analogous algorithms for any function that satisfies a linear differential equation. Silver and Sullivan [9] additionally give such algorithms for sinusoids and hyperbolic functions. Brent and Kung [1] pioneered algorithms that are faster when many non-zero terms are needed.

If for a truncated Puiseux series expanded about $z = 0$ the degree of the lowest-degree non-zero term is α and g is the greatest common divisor of the increments between the exponents of non-zero terms, then the mapping $z^\beta \to t^{(\beta-\alpha)/g}$ can be used, with care, to adapt many of the Taylor series algorithms for Puiseux series.

As described by Zippel [12, 13], many Puiseux series implementations have generalized them to allow coefficients that contain appropriate logarithms and nested logarithms that depend on the series variable. Geddes and Gonnet [2], Gruntz [3], and Richardson *et. al.* [8] give algorithms for more general truncated *hierarchical* series that also correctly prioritize essential singularities and perhaps-nested logarithms in coefficients. Koepf [6] implemented *infinite* Puiseux series in which the result is expressed as a symbolic sum of terms using Σ notation: The general term in the summand typically depends on the summation index and a power of the series variable. Some implementations compute Dirichlet, Fourier, or Poisson series. Most of the issues described in this article are relevant to most kinds of truncated series, and some of the issues are also relevant to infinite series.

To obtain a series for $f(w)$ expanded about $w = w_0$ with w_0 finite and non-zero, we can substitute $w \to z + w_0$ into $f(w)$ giving $g(z)$, then determine the series expansion of $g(z)$ about $z = 0$, then back substitute $z \to w - w_0$ into that result.[1]

To obtain a series for $f(w)$ expanded from the complex circle of radius ∞, we can substitute $\zeta \to 1/z$ into $f(w)$ giving $h(z)$, then determine the series expansion of $h(z)$ about $z = 0$, then back substitute $z \to 1/w$ into that result.

[1]However, if the series is to be used only for real $w < w_0$ then using instead $w \to w_0 - z$ might give a result that more candidly avoids unnecessary appearances of i – particularly if $f(w)$ contains logarithms or fractional powers.

A proper subset of the complex circle at infinity can be expressed by an appropriate constraint on the series variable, such as

$$\mathrm{series}\left(f\left(w\right), w=-\infty, \ldots\right) \quad \rightarrow \quad H\left(\frac{1}{w}\right)$$

where

$$H\left(x\right) \quad = \quad \mathrm{series}\left(f\left(\frac{1}{x}\right), x=0, \ldots\right) \mid x<0.$$

Therefore without loss of generality, the expansion point is $z=0$ with, $z=x+iy=re^{i\theta}$ where $r\geq 0$, $-\pi<\theta\leq\pi$, and $x, y\in\mathbb{R}$ throughout the remainder of this article.[2]

Also, wherever braced case constructs occur, the tests are presumed to be done using short-circuit evaluation from top to bottom to avoid the clutter of making the tests mutually exclusive.

Section 2 describes how to give users exactly the order or the number of nonzero terms that they request. Section 3 discusses the alternatives, pros and cons for displaying and/or propagating an explicit $O(\ldots)$, $o(\ldots)$ or $\Theta(\ldots)$ error term. Section 4 describes an efficient dense data structure for representing generalized Puiseux series. Section 5 describes extra steps necessary for exponentials of series whose 0-degree coefficient contains logarithms. Section 6 discusses alternatives for handling essential singularities.

2 The disorder of order

"What we imagine is order is merely the prevailing form of chaos."
– Kerry Thornley

Truncated series functions usually have a parameter by which the user requests a certain numeric "order" for the result. Existing implementations treat this request in different ways, some of which are significantly more useful than others.

2.1 Render onto users what they request

"Good order is the foundation of all things."
– Edmund Burke

Definition. The *exact error order* of a truncated series result expanded about $z=0$ is τ if the error is $O\left(z^{\tau}\right)$ but the error isn't $o\left(z^{\tau}\right)$. The exact error order of an exact series result expanded about $z=0$ is $+\infty$.

Remark. Knuth [5] introduced the convenient notation $\Theta\left(z^{\tau}\right)$ to denote exact-order τ in z. In comparison to $O\left(z^{\tau}\right)$, $\Theta\left(z^{\tau}\right)$ avoids discarding valuable information when we also know that a result isn't $o\left(z^{\tau}\right)$.

[2]Not all implementations currently allow all non-real infinite-magnitude expansion points, such as for

$$\mathrm{series}\left(w^{-1}, w{=}i\infty, o\left(z^{3}\right)\right) \quad \rightarrow \quad w^{-1}+\theta\left(w^{\infty}\right),$$

despite the fact that such limit points can be mapped to a real infinity by a transformation such as $w\rightarrow -iz$. Try the equivalent of this example on your computer algebra systems!

Definition. The *degree of a truncated Puiseux series* with respect to z expanded about $z = 0$ is the largest exponent of z that occurs outside of any argument of any $O(\ldots)$, $o(\ldots)$, or $\Theta(\ldots)$ term that is included in the result.

It is unreasonable to request a *degree* because, for example, there is no way for

$$\text{series}\,(\cos z,\ z\!=\!0,\ \text{degree}\!=\!1)$$

to return a result of degree 1. It is also unreasonable to request an *exact error order* because, for example, series $(\cos z,\ z\!=\!0,\ \Theta(z^3))$ can't return a series having error $\Theta(z^3)$.

Definition. If a series-function order-argument τ denotes a request that the result be $\cdots + O(z^\tau)$, then the degree of an *as-requested big-O* result should be the largest degree that satisfies

$$\text{degree} < \tau \leq \text{exact error order}. \tag{1}$$

Remark. It seems likely that more often users prefer to specify the highest degree term they *want* to view rather than the lowest degree term they *don't want* to view. Thus most users would prefer that the series function parameter τ denotes a request for a result that is $\cdots + o(z^\tau)$. Therefore:

Definition. If a series-function order-argument τ denotes a request that the result be $\cdots + o(z^\tau)$, then the degree of an *as-requested little-o* result should be the largest degree that satisfies

$$\text{degree} \leq \tau < \text{exact error order}. \tag{2}$$

The Maxima, *Mathematica*® and TI-Nspire™ truncated Puiseux series functions use this little-o interpretation of a numeric parameter τ. For example, glossing over their input and output syntax differences, they all give

$$\text{series}\left(\frac{\sin z}{z^3},\ z\!=\!0,\ 5\right) \;\to\; \frac{1}{z^2} - \frac{1}{6} + \frac{z^2}{120} - \frac{z^4}{5040}. \tag{3}$$

To this Maxima appends "$+\cdots$" and *Mathematica* appends "$+\,O[z]^6$". Try this example on your computer algebra systems.

It is easiest to implement an interpretation in which the order parameter τ denotes that the inner-most sub-expressions are computed to $o(z^\tau)$ and the final result is computed to whatever order that yields. Unfortunately, for reasons described in Subsection 2.2, that can and often does lead to a result that is $o(z^\kappa)$ with a κ that is smaller or occasionally larger than τ. For example, it would omit the last term of result (3). For such an implementation it is essential to display an error term because otherwise:

1. If exact order $\leq$ requested little-o order, then the result doesn't reveal that it is less accurate than requested, which can be disastrous.
2. If degree $>$ requested little-o order, then the user must notice and perhaps somehow truncate the excess terms to use the result in further calculations as intended.

However, even if there is an error term indicating a result that doesn't have the requested accuracy, this design is inconvenient for users because:

1. Users often don't notice the deficient or excessive accuracy.

2. Users who notice excessive order, must perhaps somehow truncate the excess terms to use the result in further calculations as intended.
3. Users who notice deficient order are forced to iteratively estimate the order argument to use in series(...) to obtain sufficient accuracy – then perhaps somehow truncate a result that exceeds the desired order.
4. If the user is another function, then that function should test the returned order and correct it if necessary by iterative adjustment and/or truncation. This is a requirement that might not occur to many authors of such functions – particularly those who aren't professional computer-algebra implementers.

It is more considerate, reliable and efficient to build any necessary iterative adjustment and/or truncation into the series(...) function rather than to foist it on all function implementers and top-level users. It isn't prohibitively harder to implement an as-requested result.

2.2 How to deliver as-requested order

"Orders are orders."
– King of the Royal Mounted

Definition. If an infinite Puiseux series is 0, then its *dominant term* is 0. Otherwise the dominant term is the lowest-degree non-zero term.

Definition. If an infinite Puiseux series is 0, then its *dominant exponent* is ∞. Otherwise the dominant exponent is the exponent of the dominant term.

Remark. Some authors call the dominant exponent the *valuation* or *valence*, but other authors confusingly call it the *order*.

"Mathematics is the art of giving the same name to different things."
– Jules Henri Poincaré

A typical truncated-Puiseux-series implementation recursively computes series for the operands of each operator and the arguments of each function, combining those series according to a specific algorithm for each operator or function. Table 1, lists the dominant exponent of a result and the operand orders that are necessary and sufficient to determine a result to $o\left(z^k\right)$. In that table a result dominant exponent of $-\infty$ signifies an essential singularity.

As indicated there, cancellation of the dominant terms of series U and V can cause the dominant exponent of $U \pm V$ to exceed $\min(\alpha, \beta)$ when $\alpha = \beta$, such as for

$$\begin{aligned} \operatorname{series}\left(e^z - \cos z,\ z=0,\ o(z)\right) &\rightarrow ((1 + z + o(z)) - (1 + o(z)) \\ &\rightarrow z + o(z). \end{aligned}$$

If the coefficient domain has zero-divisors, such as for modular arithmetic or floating-point with underflow, then the dominant exponent of UV can exceed $\alpha + \beta$, and the dominant exponent of U^γ can exceed $\gamma\alpha$.

Unfortunately, most of the entries in column 3 require us to know the dominant exponents of the operands, perhaps also together with a dominant coefficient c and the exponent σ of the next non-zero term, if any. Therefore we need this information *before* computing the operand series to the correct order, but we don't have this information until *after* we have computed the first term or two of the operand series.

One way to overcome this difficulty is as follows: We can *estimate* the dominant exponent of the operands by using a function written according to rewrite rules such as the following, which are heuristically motivated by the second column of Table 1:

$$\begin{aligned}
\operatorname{estimateDE}(z,z) &\rightarrow 1,\\
\operatorname{estimateDE}(u+v,z) &\rightarrow \min\left(\operatorname{estimateDE}(u,z),\operatorname{estimateDE}(v,z)\right),\\
\operatorname{estimateDE}(uv,z) &\rightarrow \operatorname{estimateDE}(u,z)+\operatorname{estimateDE}(v,z),\\
\operatorname{estimateDE}\left(u^k,z\right) &\rightarrow k\operatorname{estimateDE}(u,z),\\
\operatorname{estimateDE}(e^u,z) &\rightarrow 0,\\
\operatorname{estimateDE}(\sin u,z) &\rightarrow \max\left(0,\operatorname{estimateDE}(u,z)\right),\\
\operatorname{estimateDE}(\ln u,z) &\rightarrow \begin{cases}\operatorname{estimateDE}(u-1,z), & \text{if } u(0)=1,\\ 0, & \text{otherwise},\end{cases}\\
\operatorname{estimateDE}(\arctan u,z) &\rightarrow \begin{cases}\operatorname{estimateDE}(u-i,z), & \text{if } u(0)=i,\\ \operatorname{estimateDE}(u+i,z), & \text{if } u(0)=-i,\\ \max\left(0,\operatorname{estimateDE}(u,z)\right), & \text{otherwise},\end{cases}\\
\operatorname{estimateDE}(u,z)\ \mid u \text{ is independent of } z &\rightarrow 0.
\end{aligned}$$

If using the estimate causes computation of more terms than necessary, then we should truncate the excess. If using the estimate doesn't produce the required order but reveals the dominant term (and where needed the next non-zero term), then we know precisely the necessary and sufficient order to request for recomputing the series operands.

If using the estimate doesn't reveal this information, then when there is only one function argument we can iteratively increase the estimate, starting with an initial increment $\delta > 0$. For each iteration we can double the increment added to the initial estimate. This way, in a modest multiple of the time required for the last iteration, the process terminates successfully or by resource exhaustion.

Resource exhaustion can be caused by an undetected essential singularity, insufficient simplification of an operand expression, or undetected constancy around the expansion point, such as for $|x+1|+|x-1|$ at $x=0$ over the real domain, which can be more candidly as

$$\begin{cases}-2x & x<-1,\\ 1 & -1\le x\le 1,\\ 2x & \text{otherwise}.\end{cases}$$

To increase the likelihood of the first increment δ being sufficient to expose the first non-zero term or two but not prohibitively more terms than needed, we can use a function that estimates the increment between the exponents of the first two non-zero terms. Let γ be independent of z, u and v be expressions with $\tilde{\alpha} = \operatorname{estimateDE}(u,z)$ and $\tilde{\beta} = \operatorname{estimateDE}(v,z)$. Then the following partially-ordered rewrite rules are examples for such a function:

$$\begin{aligned}
\mathrm{estimateInc}\left(z,z\right) &\rightarrow 0,\\
\mathrm{estimateInc}\left(u^{\gamma},z\right) &\rightarrow \mathrm{estimateInc}\left(u,z\right),\\
\mathrm{estimateInc}\left(uv,z\right) &\rightarrow \begin{cases}\mathrm{estimateInc}\left(u,z\right) & \text{if } \mathrm{estimateInc}(v,z)=0,\\ \mathrm{estimateInc}\left(v,z\right) & \text{if } \mathrm{estimateInc}(u,z)=0,\\ \min(\mathrm{estimateInc}(u,z)\,,\mathrm{estimateInc}(v,z))\,, & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(\ln u\right) &\rightarrow \begin{cases}\mathrm{estimateInc}\left(u-1,z\right) & \text{if } u\left(0\right)=1,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(e^{u},z\right) &\rightarrow \begin{cases}\mathrm{estimateInc}\left(u,z\right) & \text{if } \tilde{\alpha}=0,\\ \left|\tilde{\alpha}\right| & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(\sin u,z\right) &\rightarrow \begin{cases}2\left|\tilde{\alpha}\right| & \text{if } \mathrm{estimateInc}\left(u,z\right)=0,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases} \quad // \text{ Same for } \sinh u\\
\mathrm{estimateInc}\left(\cos u,z\right) &\rightarrow \begin{cases}\mathrm{estimateInc}\left(u,z\right) & \text{if } \tilde{\alpha}=0,\\ 2\left|\tilde{\alpha}\right| & \text{otherwise,}\end{cases} \quad // \text{ Same for } \cosh u\\
\mathrm{estimateInc}(\arctan u,z) &\rightarrow \begin{cases}-\tilde{\alpha} & \text{if } \tilde{\alpha}<0,\\ 2\tilde{\alpha} & \text{if } \tilde{\alpha}>0 \wedge \mathrm{estimateInc}\left(u,z\right)=0,\\ \mathrm{estimateInc}\left(u-u(0),z\right) & \text{if } u(0)=i \,\vee\, u(0)=-i,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(\operatorname{arctanh} u,z\right) &\rightarrow \begin{cases}-\alpha & \text{if } \alpha<0,\\ 2\tilde{\alpha} & \text{if } \alpha>0 \wedge \mathrm{estimateInc}\left(u,z\right)=0,\\ \mathrm{estimateInc}\left(u-u(0),z\right) & \text{if } u(0)=1 \,\vee\, u(0)=-1,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(\arcsin u,z\right) &\rightarrow \begin{cases}2\left|\tilde{\alpha}\right| & \text{if } \mathrm{estimateInc}\left(u,z\right)=0,\\ \mathrm{estimateInc}\left(u,z\right)/2 & \text{if } u\left(0\right)=1 \,\vee\, u\left(0\right)=-1,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(\operatorname{arcsinh} u,z\right) &\rightarrow \begin{cases}2\left|\tilde{\alpha}\right| & \text{if } \mathrm{estimateInc}\left(u,z\right)=0,\\ \mathrm{estimateInc}\left(u,z\right)/2 & \text{if } u\left(0\right)=i \,\vee\, u\left(0\right)=-i,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases}\\
\mathrm{estimateInc}\left(\operatorname{arccosh} u,z\right) &\rightarrow \begin{cases}\tilde{\alpha} & \text{if } \tilde{\alpha}>0,\\ -2\tilde{\alpha} & \text{if } \tilde{\alpha}<0 \,\wedge\, \mathrm{estimateInc}\left(u,z\right)=0,\\ \mathrm{estimateInc}\left(u,z\right)/2 & \text{if } u\left(0\right)=1 \,\vee\, u\left(0\right)=-1,\\ \mathrm{estimateInc}\left(u,z\right) & \text{otherwise,}\end{cases}
\end{aligned}$$

$$\mathrm{estimateInc}\left(u,z\right)\ \mid u \text{ is independent of } z \ \rightarrow\ 0.$$

For a sum, estimateInc is more easily stated procedurally: Let s be a sum and

$$\tilde{\sigma}=\mathrm{estimateDE}(s,z).$$

Let $\underline{S}$ be the set of all terms in s that have $\tilde{\sigma}$ as the estimate for their dominant exponent, and let $\bar{S}$ be the set of all the other terms. If $\bar{S}$ is empty, let $\tilde{\gamma} = 0$. Otherwise let $\tilde{\gamma}$ be the minimum estimated dominant exponent of the terms in $\bar{S}$. Let $\tilde{\delta} = 0$ if 0 is the estimated increment for all of the terms in $\underline{S}$. Otherwise let $\tilde{\delta}$ be the minimum non-zero estimated increment in $\underline{S}$. Then the estimated increment for the sum is

$$\min\left(\tilde{\delta}, \tilde{\gamma} - \tilde{\sigma}\right).$$

Because estimateDE(...) might return an incorrect estimate for a dominant exponent, it is possible for this estimateInc (...) to return 0 when the actual increment is positive. Therefore we can instead estimate an increment of 1 if a *top-level* invocation of estimateInc (...) returns 0.

The required argument order in Table 1 for the inverse trigonometric and inverse hyperbolic functions depends on whether the dominant exponent α of the argument u is positive, negative, or else 0 with the corresponding coefficient being a branch point. Evaluating $u(0)$ can help us decide this: If $u(0)$ is a branch point, then we can use the above iteration scheme on $u - u(0)$ to obtain the required order $o\left(z^{k+\sigma}\right)$. If $u(0) \equiv 0$, then $\alpha > 0$. If $u(0)$ is otherwise finite and non-zero, then $\alpha = 0$. Either way, the required order for u is $o\left(z^{k}\right)$. Otherwise, either a negative dominant exponent or a coefficient having a logarithmic singularity caused $u(0)$ to have infinite magnitude or to be indeterminant. For such $u(0)$:

- If estimateDE $(u, z) \geq 0$, then we can request $o\left(z^{k}\right)$, truncating if the resulting dominant exponent is actually negative.
- Otherwise we can request an iterative determination of the appropriate order, with the proviso that the order be $o\left(z^{k}\right)$ if α is actually positive, or else 0 with c not a branch point.

For a product of two or more operands, let u contain a proper subset of the operands and v contain the complementary subset. Use recursion if u and/or v is thereby a product. Letting U and V denote the corresponding truncated series, we can estimate initial dominant exponents α and β, then alternatively increase them if necessary until either U or V reveals its true dominant exponent. Then we know enough to compute the other operand series to the appropriate order without iteration, after which we know enough to truncate or compute additional terms of its companion if necessary. We can terminate and return 0 for both U and V if an estimate for β yields $U = 0 + o\left(z^{k-\beta}\right)$, an estimate for α yields $U = 0 + o\left(z^{k-\alpha}\right)$, and $\alpha + \beta \geq k$. Algorithm 1 presents details.

The chances for needing to truncate or iterate are reduced if we choose for v a factor for which the estimate for β is most likely to be accurate, such as a linear combination of powers of z. To aid this choice we can have estimateDE (...) also return a status that is an element of

$$\{\mathit{lowerBound}, \mathit{exact}, \mathit{upperBound}, \mathit{uncertain}\},$$

with these constants being a guarantee about the estimate. Example rules for computing and propagating such a status are:

$$\begin{aligned}
\text{statusOfEDE}(z, z) &\rightarrow \mathit{equal},\\
\text{statusOfEDE}(u+v, z) &\rightarrow \mathit{lowerBound},\\
\text{statusOfEDE}\left(u^{k}, z\right) &\rightarrow \begin{cases}
\text{statusOfEDE}(u, z) & \text{if } k \geq 1,\\
\mathit{upperBound} & \text{if statusOfEDE}(u, z) = \mathit{lowerBound},\\
\mathit{lowerBound} & \text{if statusOfEDE}(u, z) = \mathit{upperBound},\\
\text{statusOfEDE}(u, z) & \text{otherwise.}
\end{cases}
\end{aligned}$$

For example, if $\mathrm{statusAndEDE}\,(u,z)$ returned $[lowerBound,6]$ and $\mathrm{statusAndEDE}\,(v)$ returned $[exact,4]$, then we know that $\mathrm{series}\,(uv, z=0, o\,(z^9))$ is $0+o\,(z^9)$ without computing series for u and v.

We can return and exploit a similar status for $\mathrm{estimateInc}\,(\ldots)$. For example, if

$$\begin{aligned}\mathrm{statusAndEDE}\,(u,z) &\rightarrow [lowerBound,6]\,,\\ \mathrm{statusAndEInc}\,(u,z) &\rightarrow [exact,0]\,,\end{aligned}$$

then there is no point to iteratively increasing the requested order for u beyond 6.

With or without this refinement, after the necessary and sufficient orders for sub-expressions have been determined, algorithms given in references such as Knuth [4] and Silver and Sullivan [9] can be generalized to compute the corresponding truncated Puiseux series for these sub-expressions.

Another way to compute operand series to the necessary and sufficient order, pioneered by Norman [7], is to use lazy evaluation, streams, or Lisp continuations. The idea is to generate at run time a network of co-routines that recursively request additional order or additional non-zero terms for sub-expressions on an incremental as-needed basis. The above estimates and iterative techniques are relevant there too, because if the request is for an increment to the order, it might not produce another non-zero term and if the request is for an additional non-zero term, iteration might be necessary to produce it. However, for such algorithms that don't recompute all of the terms each iteration, it is probably more efficient not to increase the increment each iteration.

2.3 Optional requested number of non-zero terms

This subsection is applicable not only to generalized Puiseux series, but also to more general hierarchical series.

Often rather than a requested order, users need a requested number of non-zero terms, regardless of what order is required to achieve that. Most often the needed number of terms is 1. For example, a particularly effective way to compute many limits is to compute the limit of the dominant term. This is particularly helpful for indeterminacies of the form $\infty-\infty$. As another example, if we equate a truncated series to a constant, then it is much more likely that we can solve this equation for z if there are only one or two terms in the truncated series. Thus it is important to implement a separate function such as

$$\mathrm{nTerms}\,(expression,\ variable\!=\!point,\ numberOfNonZeroTerms)\,.$$

Parameter *numberOfNonZeroTerms* could default to 1 and/or there could be a separate function such as

$$\mathrm{dominantTerm}\,(expression,\ variable\!=\!point)\,.$$

For hierarchical series, users often don't know *a priori* an appropriate set of basis functions for the series, and the dominant basis function can be an essential singularity or a logarithm rather than z. For such series it is much more appropriate for users to request the desired number of terms rather than an order in z. In this context, it is appropriate to count recursively-displayed terms of a hierarchical series as if they were fully expanded. For example, only one such distributed term is necessary for purposes such as computing a limit.

Not all implementations currently allow full generality for sub-polynomial coefficients or hierarchical basis functions. For example, try the equivalent of

$$\mathrm{series}\left(\arcsin\left(\ln(z)\right),\ z\!=\!0,\ o(z^3)\right) \tag{4}$$

on your computer algebra systems.

The sub-polynomial coefficient of z^0 for (4) can be developed as a truncated hierarchical infinite series $\ln z + (\ln z)^3/6 + \cdots$, which is preferable for some purposes such as computing a limit at $z = 0$. However, the main purpose of a generalized Puiseaux series is for an expansion in powers of z with sub-polynomial coefficients. No dictator has decreed that those sub-polynomial coefficients must be either independent of z or logarithms or nested logarithms of monomials in z. If we allow the sub-polynomial expression $\arcsin(\ln z)$ as a coefficient, then it is an allowable and exact result for input (4).

It is dangerously misleading to include a term unless all of the preceding terms are fully developed, which might and often does require *infinite* series for the *coefficients* of some preceding terms. For example, it is inappropriate to include the z^3 term of

$$z + \left(\sum_{k=0}^{\infty} \frac{(\ln z)^k}{2^{k+1}}\right) z^2 + z^3 + o\left(z^3\right) \tag{5}$$

if the series for the coefficient of z^2 is truncated, because $(\ln z)^k\, z^2/2^{k+1}$ dominates z^3 for all $k \geq 0$. For example, try

$$\text{series}\left(\arcsin\left(\ln(z)\right) + z,\ z\!=\!0,\ n\right), \quad \text{for } n = 2, 3.$$

on your computer algebra systems.

This is another reason that a requested distributed term count is more appropriate than an order request for truncated hierarchical series. This is also a good reason for providing the option of not expanding coefficients as sub-series where the implementation can't express them as infinite series and they don't terminate at a finite number of terms. For example, there should be an option for even a hierarchical series function to return either (5) or its closed form

$$z + \left(\frac{1}{2 - \ln z}\right) z^2 + z^3 + o\left(z^3\right).$$

To achieve a requested number of non-zero terms, we can iteratively increase the requested order until we obtain at least that number of non-zero terms, then truncate any excess terms. The iteration could begin with a requested order somewhere in the interval $\text{estimateDE}\,(u, z) + [0, (n-1)\text{estimateInc}(u, z)]$, where n is the requested number of non-zero terms. If this attempt exposes no terms, then we can increment the request by $n \cdot \nabla$ where ∇ starts at $\text{estimateInc}(u, z)$ and doubles after each failed attempt.

However, the implementation should address the fact that an expansion might terminate as exact with fewer than the requested number of non-zero terms. The fact that the returned number of terms is less than requested is a subtle indication that the series is exact, but an explicit error-order term of the form $\Theta\left((\ldots)^{\infty}\right)$ makes this fact more noticeable. This is additional motivation for having each intermediate series result include an indication of exactness, if known, as described in subsection 3.2.

3 Issues about displaying truncated series results

In contrast to most other expressions, the terms of a truncated Puiseux series expanded about $w = w_0$ for finite w_0 are traditionally displayed in order of *increasing* powers of $w - w_0$, even if there are logarithms involving w in expressions multiplying some of those powers. If series are represented using the same data structures as general expressions but different ordering, then the different ordering might prevent key cancellations because efficient bottom-up simplification typically relies on the simplified operands of every operator having the same canonical ordering. For example,

$$\begin{aligned} -z+\operatorname{series}\left(e^{z},\, z=0,\, o\left(z^{2}\right)\right) &\rightarrow -z+\left(1+z+\frac{z^{2}}{2}\right) \\ &\rightarrow -z+1+z+\frac{z^{2}}{2}, \end{aligned}$$

rather than simplifying all the way to $1+z^2/2$.

The series $(\ldots)$ function is most often used alone as an input, perhaps with the result assigned to a variable, rather than embedded in an expression. If so and the assigned value is ordered normally or the system re-simplifies pasted and assigned values when used in subsequent expressions, then the differently-ordered series result would safely be re-ordered into the non-series order during simplification of that subsequent input.

One way to overcome this difficulty entirely is to use a special data structure for series results, then use a special method for displaying those results. However, the next subsection describes how onerous it is to integrate such special data structures into a system in a thorough seamless way.

3.1 The pros and cons of an explicit error order term.

Maxima 5.24.0 displays "$+\ldots$" at the end of a truncated series result, which means $o\left(z^{n}\right)$ where n is the order argument provided by the user. *Mathematica* 8.0.1.0 displays a big-O term. Maple 15.3.00 displays no big-O term if the result is known to be exact, but otherwise displays a big-O term.[3] Even when result orders are always as requested, a displayed ellipsis is useful and a displayed error term is even more valuable. They remind users that although the result is symbolic, it is perhaps or definitely approximate. Moreover, it provides an opportunity for the implementation to make such truncated series *infectious*, which helps prevent users from misusing inappropriate combinations of approximate results with exact results or with results having different orders or expansion points. For example, if a result of series $(\ldots)$ is

$$(w-2\pi)^{-1/2}+(w-2\pi)^{2}+o\left((w-2\pi)^{2}\right),$$

then adding $\sin(w)$ to this result would return

$$(w-2\pi)^{-1/2}+(w-2\pi)+(w-2\pi)^{2}+o\left((w-2\pi)^{2}\right).$$

With this infectiousness, $f(w)+o\left((w-w_0)^{m}\right)$ is an elegant and convenient alternative to the input series $\left(f(w),\, w=w_0,\, o\left((w-w_0)^{m}\right)\right)$.

Unfortunately, the effort required to do a thorough job of implementing this syntactic sugar is extensive. To thoroughly propagate the influence of an error order term correctly, every command, operator and function should have a method for properly treating it. This obligation also applies to every new command, operator or function that is subsequently added to a system, including user-contributed ones that aspire to be first-class citizens seamlessly integrated into the system.

[3]For example, try the equivalent of series$(x,\ x=0,\ 2)-x$ on your computer algebra systems.

Test your computer algebra systems on the examples in Table 1. Also, test if your systems can directly plot series results or apply operators and functions such as $\int$, $\sum$, lim, solve $(\ldots)$, and series $(\ldots)$ to series results without the nuisance of first explicitly converting the series result to an ordinary expression.

3.2 Computation, propagation and display of an order term

With a little-o interpretation of the series $(\ldots, o(z^\tau))$ function order parameter and strict adherence to delivering as-requested order, it is unnecessary to represent and propagate error-order during the internal calculations, even if we display, $o(z^\tau)$ with the result.

Mathematica 8.0.1.0 interprets n in `Series`$[f(z),\ \{z, z_0, n\}]$ as a request for a $o(z^n)$ result but displays an error term using O with a larger exponent.

Maple 15.3.00 interprets series$(f(z),\ z = z_0,\ n)$ as a request for a $O(z^n)$ result but, unless the result is known to be exact, displays an error term using O with an exponent that is $\leq n$.

However, correctly determining a correct and satisfying exponent to use in O requires more work than o: To return a result with a requested order $o(z^\tau)$ using a $O(z^\nu)$, we must determine a $\nu > \tau$ such that the degree of the first omitted non-zero term, if any, is at least ν. We can't just display $O(z^{\tau+1})$, because with fractional powers the exponent of the first omitted term can be arbitrarily close to τ.

One way to determine a correct τ is to actually compute the first omitted non-zero term, but not display it. However, that omitted term can have a degree arbitrarily greater than τ, costing substantial computation. Moreover, there might not be any non-zero terms having degree greater than τ. Therefore we don't know in advance what order if any will just expose a next non-zero term whose coefficient we discard. Also, if we find such a term, it would be more informative to display $\Theta(z^\nu)$ rather than $O(z^\nu)$.

Another way to determine a correct ν is to compute a series that is $o\left(z^{\tau+\Delta}\right)$ with a predetermined Δ such as 1, then truncate to $o(z^\tau)$ and display $O(z^\nu)$ or $\Theta(z^\nu)$ where ν is the dominant degree of the truncated terms, or display $O\left(z^{\tau+\Delta}\right)$ if no terms were truncated. However, with fractional exponents there can be arbitrarily many non-zero terms having exponents in the open-closed interval $[\tau^+, \tau+\Delta]$, which is costly. Moreover, users might judge the implementation unfavorably if the exponent in O is obviously less than it could be. For example with $\sin z$ the series $z - z^3/3 + O(z^4)$ is slightly disturbing compared to $z - z^3/3 + O(z^5)$, which can be more informatively displayed as $z - z^3/3 + \Theta(z^5)$.

When computing series, we often know the exact order for the series of some or all sub-expressions. For example, if the requested order is 3 then z^2 is $\Theta(z^\infty)$, whereas z^5 is $\Theta(z^5)$. If we decide to store error-order information with the series for each sub-expression, then it preserves information to store with the error order whether it is of type o, O, or Θ, and to propagate it according to rules such as, for $\alpha < \beta$;

$$\begin{aligned}
\Theta(z^\alpha) + \Theta\left(z^\beta\right) &\rightarrow \Theta(z^\alpha), \\
\Theta(z^\alpha) + \Theta(z^\alpha) &\rightarrow O(z^\alpha), \\
\Theta(z^\alpha) + O(z^\alpha) &\rightarrow O(z^\alpha), \\
\Theta(z^\alpha) + o(z^\alpha) &\rightarrow \Theta(z^\alpha), \\
o(z^\alpha) + O(z^\alpha) &\rightarrow O(z^\alpha).
\end{aligned}$$

4 A frugal dense representation

A sparse series representation can more generally accommodate truncated Hahn series, which can also have *irrational* real exponents. For example,

$$z^{\sqrt{2}} + z^{\pi} + z^{4} + \ldots$$

This extra generality is desirable for hierarchical series. Adaptive-precision interval arithmetic can be used to keep the exponents properly ordered.

However, most published algorithms for series are written in a notation that encourages a dense representation as an array or list of coefficients with implied exponents. Adding two series is easy for sparse representation. Otherwise, adapting most of the published truncated power series algorithms to a sparse representation seems likely to make them more complicated. Moreover, with typical applications dense representation is efficient for most univariate polynomials, hence also for most series. As described in [10] recursive dense representation is also surprisingly efficient for most sparse multivariate polynomials, hence also for recursive hierarchical series or multi-variate series. Therefore this section describes a particularly efficient dense representation and some algorithmic necessities for maintaining it.

This dense representation is used for the TI-Nspire series(...) implementation, because it implements generalized Puiseux series rather than hierarchical series. The representation includes a sequence of coefficients beginning with the first non-zero coefficient and ending with the last non-zero coefficient.

The allowance of negative exponents requires explicit storage of the rational exponent of the dominant term.

The allowance of fractional exponents requires storage of the implicit positive rational exponent increment between successive stored coefficients. To minimize the number of stored 0 coefficients, this increment is the greatest common divisor of all the exponent increments between successive non-zero coefficients. (The gcd of two reduced fractions is the gcd of their numerators divided by the least common multiple of their denominators).

The truncated series 0 is represented canonically as a leading exponent of 0, an exponent increment of 0, and no coefficients.[4]

Rather than the gcd of the exponent increments, many implementation instead use the reciprocal of the *common denominator* of all the exponents. However, this can require arbitrarily more space and time. For example, it would store and process 21 coefficients rather than 3 for $1+2z^{10/3}+3z^{20/3}$, and it would store and process 31 coefficients rather than 4 for $z^{-10}+2+3z^{20}$.

Either way, for canonicality, programming safety and efficiency, it is important to trim leading, trailing and excessive intermediate zero coefficients from intermediate and final results wherever practical. However, *within* a function that adds two series, etc., it is convenient to temporarily use series that have leading 0 coefficients and/or an exponent increment that is larger than necessary. For example:

- When two series having different exponent increments are *multiplied*, we can use copies in which extra zeros are inserted between the given coefficients of one or both series so that their mutual exponent increment is the gcd of the two series exponent increments.

[4]Always using at least one coefficient would permit distinguishing a floating-point series $0.0 + o(z^{\tau})$ from a rational-coefficient series $0 + o(z^{\tau})$.

- Let γ be the gcd of the dominant exponents and exponent increments of two series. If the series have different dominant exponents and/or different exponent increments, then before the series are *added*, copies of one or both series can be padded with extra zeros before the dominant coefficient and/or between coefficients so that both series start with the same implicit exponent and have the same implicit exponent increment γ.

For both examples the resulting series are then adjusted if necessary so that its leading coefficient is canonically non-zero and its exponent increment is canonically as large as possible.

When computing any one series, the sub-expressions all have the same expansion variable and expansion point 0 after the transformations described in Section 1. Also, the desired order of the result is specified by the user and can be passed into the recursive calls for sub-expressions, adjusted according to Table 2. If an implementation delivers an as-requested o-order, then there is no need to store the order in the series data structure for intermediate series results. Therefore, only the dominant exponent, exponent increment and *frugalized* coefficient list or array are necessary for an *internal* data structure *during* computation of any one series.

For each function or operator, such as ln and "+", it is helpful to have a function that, given an expression and a requested order for expansion at $z = 0$:

1. estimates the dominant exponents of the operand series where needed,
2. computes the estimated necessary and sufficient order to request for the operand series from Table 1 and the estimated dominant exponents,
3. recursively computes those operand series to the estimated necessary and sufficient orders,
4. truncates an operand series if the requested order was excessive, or iteratively increases the requested orders if they are insufficient,
5. invokes a companion lower-level function to compute the result series from the resulting operand series. (For computing a function of a given *series*, this companion function would be invoked directly.)

If we wish to report to the user the type of the resulting order (θ, o, or O) and the corresponding exponent, then the internal data structure must also contain fields for those.

If we also wish to preserve with the final result the expansion variable and expansion point, then we must have an *external* data structure that includes those together with the internal data structure.

Not all Puiseux-series implementations currently allow fractional requested order. However, if fractional exponents are allowed in the result, then it is important to permit them as the requested order too. Otherwise, for example, a user will have to compute and view 1001 terms of $\exp\left(z^{1/1000}\right)$ merely to see the first two terms $1 + z^{1/1000}$. Try this example on your computer algebra systems.

Not all implementations currently allow negative requested order. However, if negative exponents are allowed in the result, then it is important to permit them as the requested order too. Otherwise, for example, a user will have to compute and view 1001 terms of e^z/z^{1000} merely to see the first two terms $z^{-1000} + z^{-999}$. Try this example on your computer algebra systems.

Some implementations currently restrict the magnitude of the requested order. For example, try computing the series of z^2 to order 10^9, 10^{10} and 10^{20} on your computer algebra systems. Most users are unlikely to enter such a large order, and it would probably exhaust memory or patience if the series didn't terminate with many fewer terms. However:

1. Some systems can compute a quickly terminating series quite fast no matter how large the requested order.
2. Another function that invokes a series function might generate such a request.
3. For robustness, that other function should be designed to catch the "excessive exponent" error and respond gracefully.
4. It is a great nuisance to have to implement such error catches.

These restrictions are probably caused by restricting some field in the data structure to an integer, a nonnegative value or one-word signed or unsigned integer. Although most likely motivated by a desire for efficiency, the savings are probably a negligible percentage of the time consumed by coefficient operations and other tasks.

5 Exponentials can interact with logarithmic coefficients

Definition. A function $f(z)$ is *sub-polynomial* with respect to z if

$$\lim_{r\to 0^+} \left|\left(re^{i\theta}\right)^{\gamma} f\left(re^{i\theta}\right)\right| = \begin{cases} 0 & \forall\gamma > 0, \\ \infty & \forall\gamma < 0. \end{cases}$$

Examples include

- an expression independent of the expansion variable z, or
- an expression that is piecewise constant with respect to z, or
- an expression of the form $\ln\left(c(z)\, z^{\alpha}\right)$, where $\alpha \in \mathbb{Q}$ and $c(z)$ is sub-polynomial, or
- any sub-exponential function of sub-polynomial expressions.

Definition. A sub-exponential function $g(z)$ is one for which $g(\ln z)$ is sub-polynomial.

Examples of sub-exponential functions include rational functions, fractional powers, ln, inverse trigonometric and inverse hyperbolic functions.

Non-constant sub-polynomial coefficients can arise for a series of an expression that contains logarithms, inverse trigonometric or inverse hyperbolic functions of the series variable.

Most Puiseux-series algorithms require no change for coefficients that are generalized from constants to sub-polynomial expressions, making this powerful generalization of Puiseux series cost very little additional program space. For example, some algorithms for computing functions of *constant-coefficient* Taylor series are derived via a differential equation. However, once the algorithms are obtained for constant coefficients, there is no need to incur the difficulties of including any dependent coefficients in the differentiations and integrations used in the derivations.

However, the algorithm for computing the exponential of a series does require a change: For series having a non-negative dominant exponent, the algorithm begins by computing the exponential of the degree-0 term to use in the result coefficients. If the degree-0 term is a multiple of a logarithm of a monomial containing a power of z, then the exponential of the leading coefficient generates a power of z. To avoid incorrect truncation levels, this power of z should be combined with the (perhaps implicit) power portions of the data structure so that the true degree of each term is manifest in a canonical way.

Other places where coefficients interact with powers are computing derivatives or integrals of a series with respect to z. For example, $\frac{d}{dz}\ln z \to z^{-1}$, and $\int \ln z\, dz \to (\ln z - 1)\, z$.

6 Essential singularities

Exponentials and sinusoids of negative powers of z are essential singularities at $z = 0$. For a generalized Puiseux series, let $\underline{U}$ be the sum of the terms having negative exponents and $\overline{U}$ be the sum of all the other terms. We could use the transformation $e^{\underline{U}+\overline{U}} \to e^{\underline{U}} \cdot e^{\overline{U}}$, then compute the series for $e^{\overline{U}}$, then distribute the essential singularity $e^{\underline{U}}$ over the resulting terms as factors in the coefficients. We could similarly use angle sum transformations for sinusoids of series having negative exponents. However, essential singularities dominate any power of z at $z = 0$. If we distribute the essential singularity, then subsequent series operations can truncate terms that dominate retained terms that don't contain the essential singularity, giving an incorrect result. For example,

$$\begin{aligned} \text{series}\left(e^{z^{-1}}\left(1+z^2\right)+\sin z,\, z=0,\, o(z)\right) \quad &\longrightarrow \quad \left(e^{z^{-1}}+e^{z^{-1}}z^2\right)+(z+o(z)) \\ &\xrightarrow{\textit{incorrect}} \quad e^{z^{-1}}+z+o(z). \end{aligned}$$

Therefore it is more appropriate to produce a recursively represented series in $e^{\underline{U}}$ having coefficients that are generalized Puiseux series in z – a hierarchical series.

We could have one extra field in our data structure for a multiplicative essential singularity. However, that complicates the algorithms for very little gain, because subsequent operations can easily require a more general representation. For example, one field for a single multiplicative essential singularity can't represent

$$e^{z^{-2}} + z.$$

The extra effort of implementing such a limited ability to handle essential singularities is better spent implementing more general hierarchical series.

Collecting exponentials in an expression permits computation of generalized Puiseux series for some expressions containing essential singularities that are canceled by the collection. For example,

$$\begin{aligned} \frac{e^{\csc z}}{e^{\cot z}} \quad &\to \quad e^{\csc z - \cot z} \to e^{z/2+z^3/24+\cdots} \to 1+\frac{z}{2}+\frac{z^2}{8}+\cdots, \\ \left(e^{1/x}\right)^{\sin x} |x \in \mathbb{R} \quad &\to \quad e^{(\sin x)/x} \to e^{1-x^2/6+\cdots} \to e-\frac{ex^2}{6}+\cdots. \end{aligned}$$

Try these examples on your computer algebra systems.

Summary

The generalization from Taylor series to generalized Puiseux series introduces a surprising number of difficulties that haven't been fully addressed in previous literature and implementations. Such issues discussed in this article include:

- avoiding unnecessary restrictions such as prohibiting negative or fractional orders,
- the pros and cons of displaying results with explicit infectious error terms of the form $o(\ldots)$, $O(\ldots)$, and/or $\Theta(\ldots)$,
- efficient data structures, and
- algorithms that efficiently give users exactly the order or number of nonzero terms they request.

Acknowledgments

I thank David Diminnie, Arthur Norman and a referee for their assistance.

References

[1] Brent, P. P., and Kung, H. T, Fast algorithms for manipulating formal power series, *Journal of the ACM*, 178, 1978, pp. 581–595.

[2] Geddes, K.O. and Gonnet, G.H., A new algorithm for computing symbolic limits using hierarchical series, *Proceedings of ISSAC 1998, Lecture Notes in Computer Science* 358, pp. 490-495.

[3] Gruntz, D., A new algorithm for computing asymptotic series, *Proceedings of ISSAC 1993*, pp. 239-244.

[4] Knuth, D.E., *The Art of Computer Programming, Volume 2*, Addison-Wesley, 3rd edition, 1998, Section 4.7.

[5] Knuth, D.E., Big omicron and big omega and big theta, *ACM SIGACT News* 8 (2), 1976, pp. 18-24.

[6] Koepf, W., Power series in computer algebra. *J. Symb. Comput.* 13, 1992, pp. 581-603.

[7] Norman, A. C., Computing with formal power series, *ACM Transactions on mathematical software* 1(4), 1975, ACM Press: New York, pp. 346-356.

[8] Richardson, D., Salvy, B., Shackell, J., van der Hoeven, J., Asymptotic expansions of exp-log functions, *Proceedings of the 1996 ACM Symposium on Symbolic and Algebraic Manipulation*, pp. 309-313.

[9] Silver, A. and Sullivan, E., The numerical solution of ordinary differential equations by the Taylor series method, NASA-TM-X-70438, 1973.

[10] Stoutemyer, D.R., Which polynomial representation is best?, *Proceedings of the 1984 Macsyma User's Conference*, Schenectady N.Y., pp. 221-243.

[11] Stoutemyer, D.R., Series Crimes, submitted for publication.

[12] Zippel, R.E., Univariate power series expansions in algebraic manipulation, *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Manipulation*, pp. 198-208.

[13] Zippel, R.E., *Effective Polynomial Computation*, Kluwer Academic Publishers, 1993, Chapter 10, pp. 157-171.

Table 1: Requested operand orders for a result having order $o\left(z^k\right)$, with $U = cz^\alpha + bz^\sigma + \cdots + o\left(z^m\right)$ and $V = az^\beta + hz^\gamma + \cdots + o\left(z^n\right)$ where $\alpha, \beta, \sigma, \gamma, m, n, k \in \mathbb{Q}$

operation	result dominant exponent	request m and n
$U \pm V$	$\geq \min(\alpha, \beta)$	$m = n = k$
UV	$\geq \alpha + \beta$	$m = k - \beta$ $n = k - \alpha$
$\dfrac{U}{V}$	$\geq \alpha - \beta$	$m = k + \beta$ $n = k - \alpha + 2\beta$
U^γ	$\geq \gamma\alpha$	$m = k + (1 - \gamma)\alpha$
e^U $\cos U$ $\cosh U$	$\begin{cases} 0 & \text{if } \alpha \geq 0 \\ -\infty & \text{otherwise} \end{cases}$	$m = \begin{cases} k & \text{if } \alpha \geq 0 \\ \text{essential singularity} & \text{otherwise} \end{cases}$
$\sin U$ $\tan U$ $\sinh U$ $\tanh U$	$\begin{cases} \alpha & \text{if } \alpha \geq 0 \\ -\infty & \text{otherwise} \end{cases}$	$m = \begin{cases} k & \text{if } \alpha \geq 0 \\ \text{essential singularity} & \text{otherwise} \end{cases}$
$\ln U$	$\begin{cases} \sigma & \text{if } cz^\alpha = 1 \\ 0 & \text{otherwise} \end{cases}$	$m = k + \alpha$
$\operatorname{arctanh} U$	$\begin{cases} \alpha & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise} \end{cases}$	$m = \begin{cases} k + \sigma & \text{if } cz^\alpha = 1 \vee cz^\alpha = -1 \\ k + 2\alpha & \text{if } \alpha < 0 \\ k & \text{otherwise} \end{cases}$
$\arctan U$	$\begin{cases} \alpha & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise} \end{cases}$	$m = \begin{cases} k + \sigma & \text{if } cz^\alpha = i \vee cz^\alpha = -i \\ k + 2\alpha & \text{if } \alpha < 0 \\ k & \text{otherwise} \end{cases}$
$\operatorname{arcsinh} U$	$\begin{cases} \alpha & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise} \end{cases}$	$m = \begin{cases} k + \sigma/2 & \text{if } cz^\alpha = i \vee cz^\alpha = -i \\ k + \alpha, & \alpha < 0 \\ k & \text{otherwise} \end{cases}$
$\arcsin U$	$\begin{cases} \alpha & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise} \end{cases}$	$m = \begin{cases} k + \sigma/2 & \text{if } cz^\alpha = 1 \vee cz^\alpha = -1 \\ k + \alpha, & \alpha < 0 \\ k & \text{otherwise} \end{cases}$
$\arccos U$ $\operatorname{arccosh} U$	$\begin{cases} \sigma/2 & \text{if } cz^\alpha = 1 \\ 0 & \text{otherwise} \end{cases}$	$m = \begin{cases} k + \sigma/2 & \text{if } cz^\alpha = 1 \vee cz^\alpha = -1 \\ k + \alpha & \text{if } \alpha < 0 \\ k & \text{otherwise} \end{cases}$

Algorithm 1 Compute series(u) and series(v) for computing series(uv) to $o\left(z^k\right)$

Input: Symbolic expressions u and v, variable z, and rational number k.
Output: The ordered pair of truncated series $[U, V]$, each computed to the necessary and sufficient order so that UV is $o\left(z^k\right)$.

$\delta_u \leftarrow -1;\quad \delta_v \leftarrow -1;\qquad$ // -1 means these increments haven't yet been computed
$\alpha \leftarrow \text{estimateDE}\,(u, z)\,;\quad \beta \leftarrow \text{estimateDE}\,(v, z);$
$m \leftarrow m_0 \leftarrow k - \beta;\quad n \leftarrow n_0 \leftarrow k - \alpha;$
loop
  $U \leftarrow \text{series}\,(u,\ z{=}0,\ o\,(z^m))\,;$
  if $U \neq 0 + o\,(z^m)$, then
    $\alpha \leftarrow \text{dominantExponent}\,(U)\,;$
    $n \leftarrow k - \alpha;$
    $v \leftarrow \text{series}\,(v,\ z{=}0,\ o\,(z^n))\,;$
    if $V = 0 + o\,(z^n)$, then return $[U, V]\,;$
    $\beta \leftarrow \text{dominantExponent}\,(V)\,;$
    if $m = k - \beta$, then return $[U, V]\,;$
    if $m > k - \beta$, then return $[\text{truncate}\,(U, k - \beta)\,, V]\,;$
    return $\left[\text{series}\left(u,\ z{=}0,\ o\left(z^{m-\beta}\right)\right), V\right];$
  $V \leftarrow \text{series}\,(v,\ z{=}0,\ o\,(z^n))\,;$
  if $V \neq 0 + o\,(z^n)$, then
    $\beta \leftarrow \text{dominantExponent}\,(V)\,;$
    $m \leftarrow k - \beta;$
    $U \leftarrow \text{series}\,(u,\ z{=}0,\ o\,(z^m))\,;$
    if $U = 0 + o\,(z^n)$, then return $[U, V]\,;$
    $\alpha \leftarrow \text{dominantExponent}\,(U)\,;$
    if $n = k - \alpha$, then return $[U, V]\,;$
    if $n > k - \alpha$, then return $[U, \text{truncate}\,(V, k - \alpha)]\,;$
    return $[U, \text{series}\,(v,\ z{=}0,\ o\,(z^{n-\alpha}))]\,;$
  if $m + n \geq k$, then return $[U, V]\,;$

$$\delta_u \leftarrow \begin{cases} 1 & \delta_u < 0 \wedge \text{estimateInc}\,(u, z) = 0, \\ \text{estimateInc}\,(u, z) & \delta_u < 0, \\ \delta_u + \delta_u & \text{otherwise;} \end{cases}$$

$$\delta_v \leftarrow \begin{cases} 1 & \delta_v < 0 \wedge \text{estimateInc}\,(v, z) = 0, \\ \text{estimateInc}\,(v, z) & \delta_v < 0, \\ \delta_v + \delta_v & \text{otherwise;} \end{cases}$$

  $m \leftarrow \min\,(m_0 + \delta_u, k - n)\,;$
  $n \leftarrow \min\,(n_0 + \delta_v, k - m)\,;$
endloop;

Table 2: Test examples for treating explicit error order terms

Input equivalent to	Increasingly informative results
$z - \text{series}\,(z,\ z{=}0,\ o\,(z^2))$	$o\,(z),\ O\,(z^3),\ 0$
$\ln\,(\text{series}\,(e^z,\ z{=}0,\ o\,(z^2)))$	$z{+}o\,(z^2),\ z{+}O\,(z^3),\ z{+}\Theta\,(z^3)$
$1 + z - \text{series}\,(e^z,\ z{=}0,\ o\,(z))$	$o\,(z),\ O\,(z^2),\Theta\,(z^3)$
$\text{series}\,(e^z,\ z{=}0,\ o\,(z^5)) - \text{series}\,(e^z\ z{=}0,\ o\,(z^2))$	$o\,(z^2),\ O\,(z^3),\ \Theta\,(z^3)$
$\text{series}\,(e^z{+}z^3,\ z{=}0,\ o\,(z^2)) - \text{series}\,(e^z,\ z{=}0,\ o\,(z^2))$	$o\,(z^2),\ O\,(z^3),\ \Theta\,(z^3)$
$\dfrac{\text{series}\,(e^z{+}z^3,\ z{=}0,\ o\,(z^2))}{\text{series}\,(e^z,\ z{=}0,\ o\,(z^2))}$	$1{+}o\,(z^2),\ 1{+}O\,(z^3),\ 1{+}\Theta\,(z^3)$