


Nonlinear Differential Guessing in Practice

Bertrand Teguia Tabugua 

Reading, England, UK
email@bertrandteguia.com

Abstract. This paper introduces a Maple implementation designed to recover polynomial differential equations satisfied by an unknown power series, using a few of its first coefficients. Focusing on the nonlinear setting with graded lex monomial ordering, we discuss some algorithmic considerations implemented to enhance performance and computational efficiency. The tool is integrated into the [NLDE](#) package, a suite dedicated to operations with D-algebraic functions; however, for ease of access and reproducibility, a standalone version of the software and its source code are available at <https://github.com/T3guia/D-algebraic-Guessing>.

Keywords: D-algebraic functions · guessing

1 The D-algebraic ansatz

Let f be a formal power series with initial coefficients s_0, s_1, \dots, s_N . To extend the holonomic ansatz [6] for taking nonlinear polynomial (or algebraic) differential equations (ADEs) into account, we may proceed as follows. Let k be the total degree of the differential equation sought viewed as a multivariate polynomial in the derivatives of the dependent variable $y := y(x)$. We consider a monomial ordering on the set of differential monomials

$$Y_k := \left\{ \prod_{j=1}^{j_k} y^{(j)}, (j_1, \dots, j_k) \in (\mathbb{N} \cup \{-1\})^k \right\}, y^{(-1)} = 1,$$

of at most k -product of derivatives of y . Based on that ordering, we may define an algorithmic derivative operator, say δ_k , that maps natural integers to some specific element of Y_k . We focus on differential equations without constant terms. This choice simplifies computations using the corresponding recurrence relation without particular treatment of the constant term of f .

Definition 1. *Given an ordering \succ_k^* on Y_k . For any nonnegative integer j , we define the j th δ_k -derivative of y as the $(j+1)$ st minimal element of $Y_k \setminus \{1\}$ w.r.t. \succ_k^* . We use the notation $\delta_k(y, j)$.*

Let r be a nonnegative integer. It follows naturally from Definition 1 that the set of δ_k -derivatives of y of order r is finite. We denote by $m_{\delta_k}(r)$ and $M_{\delta_k}(r)$ the corresponding minimum and maximum δ_k -order, respectively. For $r > 0$, we have $m_{\delta_k}(r) = M_{\delta_k}(r-1) + 1$.

Let r_{δ_k} be the δ_k -order of the differential equation sought, and d be the maximum degree of its polynomial coefficients $C_j(x)$, $j = 0, \dots, r_{\delta_k}$. We call the D-algebraic ansatz the associated differential polynomial

$$p = \sum_{j=0}^{r_{\delta_k}} C_j(x) \delta_k(y, j).$$

Using the conversion of differential equations into corresponding recurrence equations, one establishes that the order of such a recurrence for p should be at most $r + d$. Thus, assuming that each $C_j(x)$ brings $d + 1$ unknowns, using the initial terms s_0, \dots, s_N , we can instantiate the recurrence equation for $n = 0, \dots, N - (r + d)$ to obtain $N - (r + d) + 1$ linear equations with at least $(m_{\delta_k}(r) + 1)(d + 1)$ unknowns. Therefore, the following inequality is a necessary condition for the guessing algorithm to find an equation satisfied by f .

$$(m_{\delta_k}(r) + 2)(d + 1) + r \leq N + 2. \quad (1)$$

Depending on the choice of ordering on Y_k , the relation (1) may be precise with the corresponding explicit formula for $m_{\delta_k}(r)$. This shows that the choice of monomial ordering affects the amount of data required for a guess to succeed. In [3], the chosen ordering is based on integer partitions viewed lexicographically. This is effective in many examples. For instance, with this ordering, one only needs 33 terms to guess a differential equation for $f = \sum_{n=0}^{\infty} (an + b)^n x^n / n!$. With our choice of ordering, our software requires at least 74 terms in its standard use to find the same guess. We plan to investigate many choices of orderings in future versions. We recommend [3] for various important aspects of guessing.

Definition 2 (Differential graded degree- k lex ordering). *Let \succ_k be the order on Y_k such that for any $t_1, t_2 \in Y_k$, we have $t_1 \succ_k t_2$ if and only if $\text{ord}(t_1) > \text{ord}(t_2)$; or $\text{ord}(t_1/\text{gcd}(t_1, t_2)) > \text{ord}(t_2/\text{gcd}(t_1, t_2))$.*

For the differential graded degree- k lex ordering, one proves that $M_{\delta_k}(r) := \binom{k+r+1}{k} - 2$ (see [5]).

2 Implementation and Application

Before discussing our package’s application, we address a primary improvement in the implementation of Algorithm 1 from [5]. In [5, Remark 3], it was noted that recurrences derived from higher-degree ADEs often dominate computational costs due to the complexity of the involved convolutions.

The current version of the software mitigates this by implicitly exploiting optimized polynomial arithmetic, specifically high-performance multiplication algorithms. This improvement instantiates the D-algebraic ansatz directly using the truncation of f , allowing the underlying polynomial arithmetic to handle the expansion. We recall that standard algorithms for such operations include the Fast Fourier Transform (FFT) and Karatsuba’s algorithm [2, Chapter 4].

While this approach aligns with established “common sense” in computer algebra—having been utilized as a strategy already in [3]—it bears explicit mention here. Similarly, in discussing holonomic guessing for functions, [4, Section 1.5] favors polynomial substitution; though in the linear case, the efficiency difference compared to recurrence conversion is often negligible. In short, we are formalizing a consideration that is frequently left implicit in automated guessing.

Our package is available at <https://github.com/T3guia/D-algebraic-Guessing>. To use it, the Maple variable `libname` should contain the path to the directory where the `DalgGuessing.mla` file is located. Alternatively, this file should be in the same directory as the worksheet being used. After completing this setup, one loads the package using the command

```
> with(DalgGuessing);
[DalgFunGuess, modDalgFunGuess]
```

As the above display shows, there are two exported commands, one for computations in characteristic zero, and the other over some $\mathbb{Z}/m\mathbb{Z}$, with a positive integer m (usually prime) chosen by the user. These commands are made to enable users to deduce their own conclusions after obtaining equations. More details about these commands are provided on the GitHub page. The main arguments of the calling sequence for `DalgFunGuess` are used as follows.

```
> DalgFunGuess(L, degADE=k, degPoly=d, startfromord=r, allPolyDeg=false);
```

`L` is the list of initial terms; `degADE` is the (total) degree of the differential polynomial sought; `degPoly` is the maximum degree of the polynomial coefficients. Two other interesting arguments are `startfromord`, representing the starting order for the search, and `allPolyDeg`, a boolean variable, which, when assigned `true`, enables the code to fully exploit the data to search for a differential polynomial that has some polynomial coefficients of degree strictly less than `degPoly`.

We now consider a simple example resembling a “hard” one from [1]. Given two ADEs $p = 0, q = 0$, we seek an ADE $\rho = 0$ whose solutions are $h = f/g$, where $p(f) = q(h) = 0$. Two similar computations lasted for more than an hour without termination, with the algorithms from [1] (see Table 1).

$$p := y' - x y^2, \quad q := -y' + y + x + 1.$$

The Maple code below computes the initial coefficients of the series h up to `Order` for specified initial values `s_0_f` and `s_0_g` of f and g , respectively.

```
p:=diff(y(x), x) - x*y(x)^2:
q:=-diff(z(x), x) + z(x) + x + 1:
sys := {p=0,q=0}: #Note: it is better to treat both ADEs separately
InitialConds := {y(0) = s_0_f, z(0) = s_0_g}:
#Order is a global variable for truncating series solutions
Order :=35:

#Computing truncations of f and g of order Order
sol := dsolve(sys union InitialConds, {y(x), z(x)}, series):

# Extract the truncations
f_series := eval(y(x), sol):
g_series := eval(z(x), sol):

# Convert to polynomial
h_series := series(f_series/g_series, x, Order):
h_poly := convert(h_series, polynom):

# List of coefficients
L:=PolynomialTools:-CoefficientList(h_poly,x):
```

Using `s_0_f=2` and `s_0_g=1`, and `L` from the code above, we use the calling `DalgFunGuess(DalgFunGuess(L, degADE = 3, degPoly = 2, startfromord = 2))`, and get the following ADE within 20 seconds.

$$\left(-\frac{53}{5}C - 3Cx\right)y(x)^2 + \left(-\frac{11}{10}C - 10Cx - \frac{89}{10}Cx^2\right)y(x)^3 + \left(-\frac{28}{5}C - \frac{12}{5}Cx - 2Cx^2\right)\left(\frac{d}{dx}y(x)\right)^2 + \left(-Cx^2 + \frac{14}{5}C + \frac{6}{5}Cx\right)y(x)\frac{d^2}{dx^2}y(x) + \left(-\frac{39}{5}C - \frac{31}{5}Cx - Cx^2\right)\frac{d}{dx}y(x)y(x) = 0.$$

The constant `_C` indicates that the linear system solved for the polynomial coefficients is one-dimensional. This is generally a good indicator of a correct guess. A simple candidate for ρ is obtained by taking `_C` as the lcm of the denominators, here 10. However, we want to find a differential polynomial ρ that is valid for all possible f and g . Since $p = 0$ and $q = 0$ are easy to solve, we can check if the above output is valid for all f, g by simply verifying whether the generic ratio $h_* = f_*/g_*$ for generic zeros f_* and g_* of p and q , respectively, satisfies it. We have $h_* = -2/(-x^2 + 2\theta)(x + 2 - e^x \beta)$, with arbitrary θ, β . We find that h_* would be a zero of ρ if and only if $\theta = 1/s_{0_f} = 1/2$. The same condition is obtained for several values of `s_0_f` and `s_0_g`, showing that there is a dependency between the initial term of f and the output ADE. Note that the Gröbner bases approach also aims to eliminate such dependencies.

Fortunately, a first-order guess can be obtained with the symbolic initial terms `s_0_f = \theta` and `s_0_g = \beta`. We use the calling `DalgFunGuess(L, degADE = 2, degPoly = 5, startfromord = 1, inputConstants = {beta, theta})`, where the argument `inputConstants` tells the algorithm not to replace the symbols in its set with Maple's generic constants. The computations take only a few seconds. After clearing the denominator, we get the ADE

$$(2\theta^2 x^2 + 4\theta^2 x - 4\theta) y(x) + (2\theta^2 x^2 - 4\theta) \frac{d}{dx} y(x) + (-\theta^2 x^5 - \theta^2 x^4 + 4\theta x^3 + 4\theta x^2 - 4x - 4) y(x)^2 = 0. \quad (2)$$

From this output, one deduce a candidate for ρ by computing the resultant $\text{res}_\theta(\rho_\theta, \rho'_\theta)$, where ρ_θ is the left-hand side in (2). We obtain a second-order differential polynomial of degree 6.

$$\begin{aligned} \rho = & (x^2 + x) y''^2 y^2 + (-4x^4 - 8x^3 - 4x^2) y'' y^4 + (-x^2 - 3x - 1) y^3 y'' + (-3x^2 - 5x - 1) y^2 y' y'' \\ & + (-4x^2 - 4x) y'' y y'^2 + (4x^6 + 12x^5 + 12x^4 + 4x^3) y^6 + (x^4 + 6x^3 + 5x^2 - 1) y^5 \\ & + (6x^4 + 16x^3 + 12x^2 + 2x) y^4 y' + (x + 1) y^4 + (8x^4 + 16x^3 + 8x^2) y^3 y'^2 + (x^2 + 5x + 3) y^3 y' \\ & + (4x^2 + 11x + 4) y^2 y'^2 + (6x^2 + 10x + 2) y y'^3 + (4x^2 + 4x) y'^4. \end{aligned}$$

One verifies that $\rho(h_*) = 0$, so the guess is correct. Note that it is also possible to obtain this result without knowing h_* explicitly. Automatizing such a process would provide an alternative for the arithmetic of D-algebraic functions. The current implementation of the algorithms from [1] also finds this ρ quite quickly.

Sparsity and the argument `allPolyDeg`. When set to `true`, the `allPolyDeg` argument fixes the order and assumes that some coefficients are of degree strictly less than `degPoly`. This reduces the number of unknowns in some polynomial coefficients, but expands the overall search space because the algorithm must consider all permutations of degree vectors where at least one entry is exactly `degPoly`. This method takes over when the standard approach reaches a δ_k -order where data is insufficient to continue. Using this argument, we can recover (2) with only 13 initial terms, while the standard approach would normally require at least 19 terms. A Maple worksheet and its PDF print are made available on the GitHub page for more examples.

References

1. Ait El Manssour, R., Sattelberger, A.L., Tegua Tabugua, B.: D-algebraic functions. *Journal of Symbolic Computation* p. 102377 (2024) [3](#), [4](#)
2. Geddes, K.O., Czapor, S.R., Labahn, G.: *Algorithms for Computer Algebra*. Springer (1992) [2](#)
3. Heibisch, W., Rubey, M.: Extended rate, more GFUN. *Journal of Symbolic Computation* **46**(8), 889–903 (2011) [2](#)
4. Kauers, M.: *D-finite Functions*. Springer (2023) [2](#)
5. Tegua Tabugua, B.: D-algebraic guessing. arXiv preprint arXiv:2510.26869 (2025) [2](#)
6. Zeilberger, D.: The holonomic ansatz I. *Foundations and applications to lattice path counting*. *Annals of Combinatorics* **11**(2), 227–239 (2007) [1](#)