Séminaire Lotharingien de Combinatoire **82B** (2019)
Article #101, 8 pp.

Proceedings of the 31st Conference on Formal Power Series and Algebraic Combinatorics (Ljubljana)

DD-finite functions in Sage

Antonio Jiménez-Pastor*

Doctoral Program Computational Mathematics, JKU, Linz

Abstract. We present here a Sage implementation for DD-finite functions, which are a natural extension of the class of holonomic or D-finite functions. The package, focused on a functional approach, provides natural commands for executing closure properties among DD-finite functions, a general framework for extracting their coefficient sequences and allows the user to compute the composition (as power series) of the functions treated in the package.

Keywords: holonomic, D-finite, generating functions, closure properties, formal power series

1 Introduction

A formal power series $f(x) = \sum_{n\geq 0} a_n x^n$ is called D-finite, if it satisfies a linear differential equation with polynomial coefficients [8, 13, 14]. Many generating functions of combinatorial sequences are of this type as well as the most commonly used special functions [1, 3, 11]. These objects can be represented in finite terms using a defining differential equation and sufficiently many initial values.

We recently extended this class to a wider set of formal power series that satisfy linear differential equations with D-finite coefficients, and call them DD-finite [4]. In the same way as D-finite functions satisfy several closure properties that have been implemented in several computer algebra systems [12, 10, 2, 9, 7], we present here a Sage implementation for the closure properties and other operations on DD-finite functions.

The finiteness of the representation for D-finite functions allows to finitely represent DD-finite functions, with a defining differential equation and initial conditions. A difference between our packages and other implementations is that our focus is on the whole function, not only its annihilating differential operator. This leads to two different steps while manipulating DD-finite functions: computing the final differential equation and computing the initial data required for the specific object. More details on the structure we used can be found in Section 2.

Kauer's et al. package `ore_algebra` [7] has implemented several operations for D-finite functions. On this base level, the package presented here uses these structures by

*antonio.jimenez-pastor@dk-compmath.jku.at.This author was funded by the Austrian Science Fund (FWF): W1214-N15, project DK15.

default.If the user desires to avoid that package, or `ore_algebra` is not installed, a different implementation [5] is provided. That is used by default with DD-finite functions.

At the time of writing, the package described in this document is still under construction and has not been added to the official Sage distribution. Readers who want to try it are invited to download the current version from

https://www.dk-compmath.jku.at/Members/antonio/sage-package-dd_functions

and are encouraged to send bug reports, feature requests or other comments.

Once downloaded and unpacked the file, the user should add the path to the obtained folder into the package folders in the Sage installation or just run Sage within the directory *diff_defined_functions*. Then the package is ready to be used using the following command:

```
sage: from ajpastor.dd_functions import *
```

All the features of the package (creating the appropriate objects, arithmetically manipulating them, getting some examples, etc.) can be used without further configuration.

```
sage: C = Catalan();   # Getting Catalan gen. function
sage: F = Fibonacci(); # Getting Fibonacci gen. function
sage: C*(1+x^2*C^2*F(x*C)) == F(x*C) # Check identity
        True
sage: T = Tan(x) # Getting the tangent function
sage: T.derivative() == T^2+1 # Derivative identity for the tangent
        True
```

The user can check that the package is ready to be used by running the included tests on the code. Running these tests will produce a text output that shows the timestamps of the tests that are been executed. The code to run those tests is the following:

```
sage: from ajpastor.tests import dd_functions
sage: dd_function.run()
```

More details on how to build the structures and the examples available can be found in Sections 2 and 3.

2 The basic structures

Although the main motivation behind our package is manipulating DD-finite functions, the theory involved can be described in a more general way. We recall the main definition and the basic closure properties. For details, see [4].

Definition 1. *Let R be a non-trivial differential subring of $K[[x]]$ and $R[\partial]$ the ring of linear differential operators over R. We call $f \in K[[x]]$* differentially definable over R *if there is a non-zero operator $\mathcal{A} \in R[\partial]$ that annihilates f, i.e., $\mathcal{A} \cdot f = 0$. By $\mathrm{D}(R)$ we denote the set of all $f \in K[[x]]$ that are differentially definable over R.*

Theorem 2. *Let R be a non-trivial differential subring of $K[[x]]$ and $f(x), g(x) \in \mathrm{D}(R)$. Then:*

1. $f'(x) \in \mathrm{D}(R)$.
2. *Any antiderivative of* $f(x)$ *is in* $\mathrm{D}(R)$.
3. $f(x) + g(x) \in \mathrm{D}(R)$.
4. $f(x)g(x) \in \mathrm{D}(R)$.
5. *If* $r(0) \neq 0$, *then its multiplicative inverse* $1/r(x)$ *in* $K[[x]]$ *is in* $\mathrm{D}(R)$.

In particular, $\mathrm{D}(R)$ *is a differential subring of* $K[[x]]$.

2.1 DDRing

The structure `DDRing` describes the rings of differentially definable functions, $\mathrm{D}(R)$. Thus, they are built from any ring structure R in Sage. Following the *Parent-Element* model of Sage, `DDRing` is a parent class that will include as element any differentially definable function over R.

To create a `DDRing`, the user has to provide the base ring R for the coefficients of the differential operators. There are several optional inputs that allow more flexibility and a wider use of the structure. For more information, the user can type `DDRing?` in Sage to obtain all the information about it.

The use of a `DDRing` in Sage is restricted to creating elements within itself (see the description of the method `element` in the Section 2.2) and to get information about the ring R over the `DDRing` is based on.

Two `DDRings` are defined by default in the system when the package is loaded: the object `DFinite` that is the ring of D-finite functions, and `DDFinite` which represents the ring of DD-finite functions.

```
sage: DDFinite
        DD-Ring over (DD-Ring over (Univariate Polynomial Ring in x over
        Rational Field))
sage: Qi.<i> = NumberField(x^2+1)
sage: S = DDRing(Qi[x])
sage: S
        (DD-Ring over (Univariate Polynomial Ring in x over Number Field
        in i with defining polynomial x^2 + 1))
```

2.2 DDFunction

The class `DDFunction` represents differentially definable functions over any ring. It is composed of a linear differential operator and some initial conditions. Moreover, it is the *Element* class of a `DDRing`, so every `DDFunction` must be included in a particular `DDRing`.

To create a `DDFunction`, the user must use the method `element` of the `DDRing` where the function belongs. A function $f(x)$ satisfying a linear differential equation with coefficients r_i and initial values a_i is created from two lists $[r_0, \dots, r_d]$ and $[a_0, \dots, a_n]$, i.e.,

$$([r_0, \dots, r_d], [a_0, \dots, a_n]) \mapsto \begin{cases} r_d f^{(d)}(x) + \cdots + r_0 f(x) = 0, \\ f(0) = a_0, \dots, f^{(n)}(0) = a_n. \end{cases}$$

The method `element` also checks that the coefficients r_i are in the appropriate ring of coefficients R and that the initial conditions a_i are elements of the field where the function $f(x)$ is considered.

```
sage: f = R.element([-i,1],[1]) # f == exp(i*x)
sage: g = R.element([i,1],[1]) # g == exp(-i*x)
sage: (f+g)/2 == Cos(x)
        True
```

3 Selected methods and main features

3.1 The structure of `DDFunction`

The package provide several methods to extract information from a `DDFunction`

- `equation`: the differential operator that defines the function.
- `getInitialValue`: the value of the nth derivative of the function at $x = 0$.
- `getSequenceElement`: the value of the nth coefficient of the power series.
- `getOrder`: the order of the differential operator defining the function.
- `min_coefficient`: the first non-zero coefficient of the power series
- `zero_extraction`: the order of the power series and the `DDFunction` defined after factoring the maximal power of x possible.

```
sage: f = DFinite.element([1,0,1],[0,1]) # f == sin(x)
sage: f.equation
        Wrapped_OreOperator(D^2 + 1)
sage: [f.getInitialValue(n) for n in range(5)]
        [0, 1, 0, -1, 0]
sage: [f.getSequenceElement(n) for n in range(5)]
        [0, 1, 0, -1/6, 0]
sage: f.getOrder(), f.min_coefficient()
        (2, 1)
sage: n, g = f.zero_extraction
sage: n
        1
sage: print g
```

```
        (2:2:4)DD-Function in (DD-Ring over (Univariate Polynomial Ring
        in x over Rational Field)):
        ------------------------------------------------
                -- Equation (self = f):
                        f  * (x)
                        +   D(f) * (2)
                        + D^2(f) * (x)
                        = 0
                -- Initial values:
                [1, 0, -1/3]
        ------------------------------------------------
sage: x*g == f
        True
```

3.2 Closure properties

All the arithmetic operations are implemented using Python magic methods. This enables to use also Sage coercion system for a more *user friendly* use of the package. For more details on the algorithms, see [5]

- **Addition**: `add` or simply `+`
- **Multiplication**: `mult` or simply `*`
- **Difference**: `sub` or simply `-`
- **Exponentiation**: `pow` or simply `^`
- **Multiplicative Inverse:** `inverse` returns a `DDFunction` representing the multiplicative inverse of the function.

```
sage: sin = DFinite.element([1,0,1],[0,1])
sage: cos = DFinite.element([1,0,1],[1,0])
sage: sin^2 + cos^2 == 1
        True
sage: sec = cos.inverse
sage: print sec
        (1:1:5)DD-Function in (DD-Ring over (DD-Ring over (Univariate
        Polynomial Ring in x over Rational Field))):
        ------------------------------------------------
                -- Equation (self = f):
        (2:2:2) f  * (DD-Function in (DD-Ring over (Univariate Polynomial
        Ring in x over Rational Field)))
        (2:2:2) +   D(f) * (DD-Function in (DD-Ring over (Univariate
        Polynomial Ring in x over Rational Field)))
                        = 0
                -- Initial values:
                [1, 0]
        ------------------------------------------------
sage: cos*sec
        1
sage: sin*sec == Tan(x)
        True
```

The user can also perform differential operations over a `DDFunction`, like computing the derivative ($f'(x)$) or computing the indefinite integral ($\int f$).

- `derivative`: returns a `DDFunction` with the derivative.
- `integrate`: returns a `DDFunction` with an indefinite integral. The value at $x = 0$ can be specified. By default, it is 0.

```
sage: f = DDFinite.element([Cos(x)^3, Sin(x), Cos(x)], [0,1])
sage: df = f.derivative()
sage: f.integrate().derivative() == f
        True
sage: df.integrate(0) == f
        True
sage: all(f.getInitialValue(n+1) == df.getInitialValue(n) for n
....: in range(10))
        True
```

As recently proven [6], the composition of differentially definable functions is again differentially definable and the implementation described in that report is included into the package. The method `compose` or the magic method `__call__` do the job.

```
sage: f = DDFinite.element([Cos(x)^3, Sin(x), Cos(x)], [0,1])
sage: sin = DFinite.element([1,0,1],[0,1])
sage: g = sin(f)
sage: print g
        (2:2:95)DD-Function in (DD-Ring over (DD-Ring over (DD-Ring over
        (Univariate Polynomial Ring in x over Rational Field)))):
        ------------------------------------------------
                -- Equation (self = f):
        (4:4:37)        f  * (((sin(sin(x)))')^3)
        (2:2:37) -      D(f) * ((sin(sin(x)))'')
        (2:2:19) +      D^2(f) * ((sin(sin(x)))')
                        = 0
                -- Initial values:
                [0, 1, 0]
        ------------------------------------------------
sage: sin(sin) == f
        True
```

3.3 Algebraic and Differentially Algebraic properties

Functions that are algebraic over a differential field F are also differentially definable over F [8, 6]. Given the polynomial defining an algebraic function, the user can get a `DDFunction` associated with such function using the method `DAlgebraic`.

```
sage: Q.<x,y> = QQ[];
sage: p = x*y^2 - y + 1 # Minimal polynomial for Catalan gen. func.
sage: f = DAlgebraic(p, [1,1])
sage: print f
        (2:2:5)DD-Function in (DD-Ring over (Univariate Polynomial Ring
        in x over Rational Field)):
        ------------------------------------------------
```

```
                -- Equation (self = f):
                                f  * (2)
                        +       D(f) * (10*x - 2)
                        +       D^2(f) * (4*x^2 - x)
                        = 0
                -- Initial values:
                [1, 1, 4]
-------------------------------------------------
sage: [f.getSequenceElement(n) for n in range(10)] # Catalan numbers
        [1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862]
```

We also proved in [6] that DD-finite functions are differentially algebraic. Using the method `diff_to_diffalg`, the user can compute from a `DDFunction` an associated differentially algebraic equation with polynomial coefficients.

```
sage: cos = DFinite.element([1,0,1],[1,0])
sage: diff_to_diffalg(cos)
        y_2 + y_0
sage: f = DDFinite.element([-Exp(x), 1], [1]) # f == exp(exp(x)-1)
sage: diff_to_diffalg(f)
        y_1^2 - y_2*y_0 + y_1*y_0
```

3.4 Built-in functions of the package

Aiming for a more user friendly interface, we provide a set of examples of D-finite and DD-finite functions that can be easily called and built from Sage once the package is loaded. Some of these functions are elementary as the exponential function or trigonometric functions; some are special functions as the hypergeometric ${}_pF_q$ functions, solutions to the Riccati equation, the Mathieu functions or some generating functions for combinatorial sequences.

All implemented functions can be checked in Sage by typing `ddExamples?`. This command displays the documentation for the `ddExamples` file where all these built-in functions are explained.

4 Conclusions

The Sage package `ajpastor.dd_functions` provides a functional-focused implementation of differentially definable functions for arbitrary differential rings. In particular, it can be used for working with regular D-finite functions and DD-finite functions.

This package can be used to prove symbolically identities between DD-finite functions and to obtain combinatorial sequences that are out of the D-finite scope.

Closure properties, such as addition, multiplication and exponentiation can be performed by the package, as well as other operations such as composition or division. It also provides several methods to convert algebraic functions to `DDFunction` manipulating a defining polynomial to obtain a differential equation, and to compute a non-linear equation with polynomial coefficients for any differentially definable function.

References

[1] G. E. Andrews, R. Askey, and R. Roy. *Special Functions*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1999.

[2] F. Chyzak. "Gröbner bases, symbolic summation and symbolic integration". *Gröbner Bases and Applications (Linz, 1998)*. London Math. Soc. Lecture Note Ser. 251. Cambridge Univ. Press, Cambridge, 1998, pp. 32–60.

[3] "*NIST Digital Library of Mathematical Functions*". http://dlmf.nist.gov/, Release 1.0.16 of 2017-09-18. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds.

[4] A. Jiménez-Pastor and V. Pillwein. "A computable extension for holonomic functions: DD-finite functions". *J. Symbolic Comput.* **94** (2018), pp. 90–104. Link.

[5] A. Jiménez-Pastor and V. Pillwein. "Algorithmic Arithmetics with DD-Finite Functions". *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation*. Ed. by A. Carlos. ACM, New York, NY, USA, 2018, pp. 231–237. Link.

[6] A. Jiménez-Pastor, V. Pillwein, and M. F. Singer. "Some structural results on Dn-finite functions". Doctoral Program Computational Mathematics. 2019. Link.

[7] M. Kauers, M. Jaroschek, and F. Johansson. "Ore Polynomials in Sage". *Computer Algebra and Polynomials*. Ed. by J. Gutierrez, J. Schicho, and M. Weimann. Lecture Notes in Computer Science. 2014, pp. 105–125. Link.

[8] M. Kauers and P. Paule. *The Concrete Tetrahedron: Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates*. 1st ed. Springer, 2011.

[9] C. Koutschan. "Advanced Applications of the Holonomic Systems Approach". PhD thesis. RISC-Linz, Johannes Kepler University, 2009. Link.

[10] C. Mallinger. "Algorithmic Manipulations and Transformations of Univariate Holonomic Functions and Sequences". MA thesis. RISC, J. Kepler University, 1996.

[11] E. D. Rainville. *Special Functions*. 1st ed. Bronx, N.Y.: Chelsea Publishing Co., 1971.

[12] B. Salvy and P. Zimmermann. "Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable". *ACM Transactions on Mathematical Software* **20**.2 (1994), pp. 163–177. Link.

[13] R. P. Stanley. "Differentiably Finite Power Series". *European J. Combin.* **1**.2 (1980), pp. 175–188. Link.

[14] R. P. Stanley. *Enumerative Combinatorics, Vol. 2*. Cambridge University Press, Cambridge, 1999.