A Sage Package for the Symbolic-Numeric Factorization of Linear Differential Operators

Alexandre Goyer

INRIA Saclay – Île-de-France
Computer Science Laboratory of Ecole Polytechnique (LIX)

Software presentation, ISSAC'21

July 22, 2021 (online)

Object of interest:

$(E) \quad : \quad a_n(z) f^{(n)}(z) + \cdots + a_1(z) f'(z) + a_0(z) f(z) = 0$ whose a_i are rational fractions.

Point of view: linear differential operator $L = a_n \partial^n + \cdots + a_1 \partial + a_0 \in \mathbb{K}(z)\langle\partial\rangle$.

Action of $\mathbb{K}(z)\langle\partial\rangle$ on functions in such a way that f is a solution of (E) if and only if $L \cdot f = 0$.

Commutation rule: $\partial z = z\partial + 1$ (reflecting Leibniz's rule $(zf)' = zf' + f$).

My package `diffop_factorization` mainly provides the function `Dfactor`.

input: a linear differential operator L
output: a list of irreductible operators $L_1, \ldots, L_k$ such that L is equal to the composition $L_1 \cdots L_k$

▷ available at https://github.com/a-goyer/diffop_factorization

Example: (the package `ore_algebra` is available at https://github.com/mkauers/ore_algebra)

```
from ore_algebra import DifferentialOperators
Diffops, z, Dz = DifferentialOperators(QQ, 'z')

L = (4*z^3 + 2*z^2 - 4*z - 2)*Dz^3 + (16*z^2 + 2*z - 6)*Dz^2 + (7*z - 1)*Dz - 1

from diffop_factorization import Dfactor
Dfactor(L)
```

```
[(4*z - 4)*Dz + 4, (z^2 + 3/2*z + 1/2)*Dz^2 + (z + 1/2)*Dz - 1/4]
```

```
L1, L2 = _
L1 * L2 == L
```

```
True
```

Note: Here, the output ensures that $\left(z^2 + \frac{3}{2}z + \frac{1}{2}\right)\partial^2 + \left(z + \frac{1}{2}\right)\partial - \frac{1}{4}$ is irreducible.

History: algorithms of factorization

– **1894**: Beke
- "second exterior power method"
- first order right factor computation

– **1996**: Singer
- reducibility test ("eigenring" method)

– **1997**: van Hoeij
- "local to global" algorithm

– **2004**: Cluzeau, van Hoeij
- modular tricks

Complexity analysis:

– **1990**: Grigor'ev
– **2020**: Bostan, Rivoal, Salvy

Improvements of Beke's algorithm

– **1989**: Schwarz
– **1990**: Grigor'ev
– **1994**: Bronstein
– **1996**: Tsarev

Symbolic-numeric approaches

– **2007**: van der Hoeven

– **2013**: Johansson, Kauers, Mezzarobba
- first order right factor computation

van der Hoeven's algorithm

Key idea: $L_2 \mapsto \mathrm{Solutions}(L_2)$ is a bijection between the rigth factors of L and the subspaces of $\mathrm{Solutions}(L)$ invariant under the "monodromy" action (under a regularity assumption).

Function `right_Dfactor`

input: a linear differential operator L
output: a nontrivial right factor L_2 of L or `Irreducible`

1. *compute a basis of* $V := \mathrm{Solutions}(L)$
2. *compute a generating set* $\mathcal{M}$ *of the monodromy group by approximation*
3. *search for a nontrivial* $\mathcal{M}$*-invariant subspace* U *of* V*:*
    4. *if any:*
        5. *guess a candidate* L_2 *from* U
        6. *if* L_2 *does not divide exactly* L*: restart with higher precision*
        7. *else: return* L_2
    8. *else: return* `Irreducible`

Step-by-step presentation of my package with the previous example:

```
from diffop_factorization import right_Dfactor

L = (4*z^3 + 2*z^2 - 4*z - 2)*Dz^3 + (16*z^2 + 2*z - 6)*Dz^2 + (7*z - 1)*Dz - 1

L2 = right_Dfactor(L); L2
```

```
(z^2 + 3/2*z + 1/2)*Dz^2 + (z + 1/2)*Dz - 1/4
```

Computing a basis of solutions

Write $L = q(p_n\partial^n + \cdots + p_1\partial + p_0)$ with $q \in \mathbb{K}(z)$ and $p_i \in \mathbb{K}[z]$ coprime.

Definition: z_0 is a singular point of L if a z_0 is a root of p_n.

Proposition: z_0 is not singular $\Rightarrow$ there is a full set of power series solutions at z_0.

Cauchy-type theorem: $f \in \mathrm{Solutions}(L, z_0) \longmapsto (f(z_0), f'(z_0), \ldots, f^{(n-1)}(z_0)) \in \mathbb{C}^n$ is bijective.

```
L # z0 = 0 is not a singular point
```

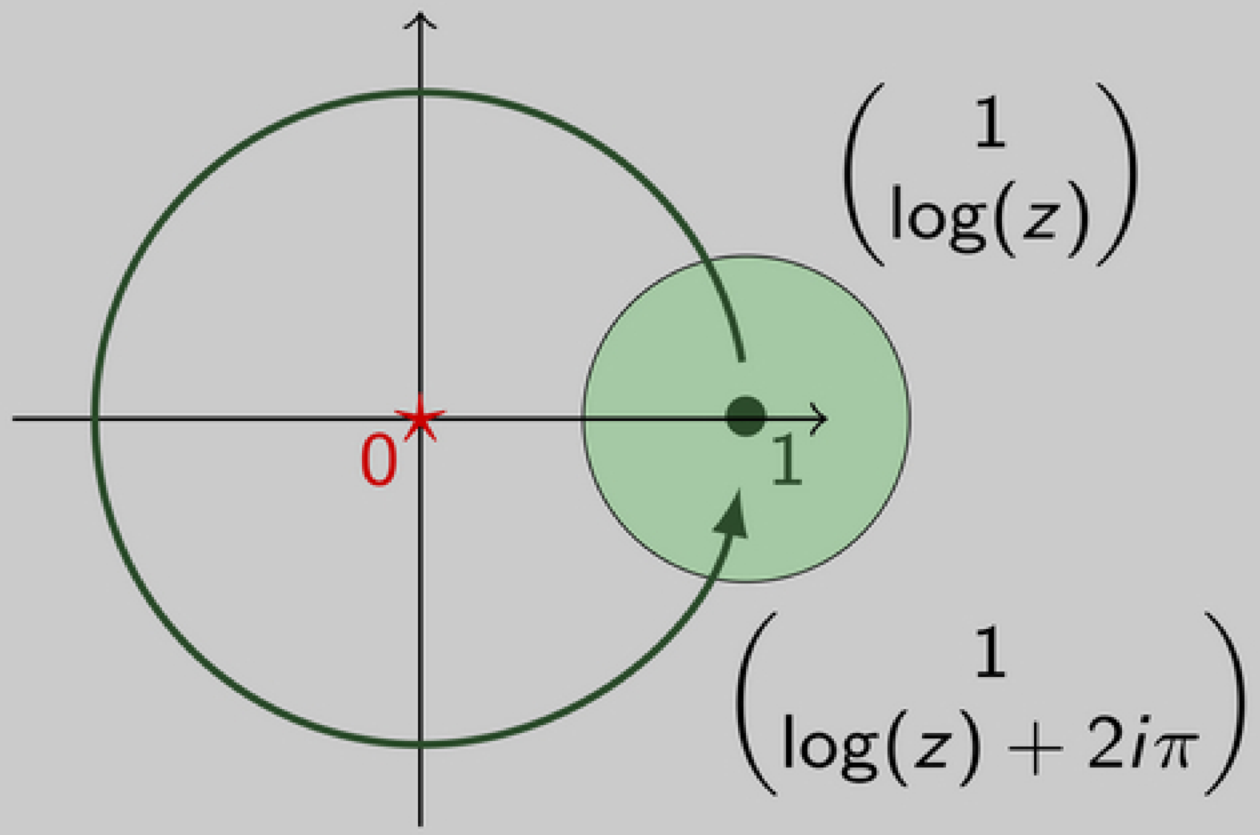
```
(4*z^3 + 2*z^2 - 4*z - 2)*Dz^3 + (16*z^2 + 2*z - 6)*Dz^2 + (7*z - 1)*Dz - 1
```

```
L.power_series_solutions(n=8) # differential equation <-> polynomial recurrence on coefficients
```

```
[z^2 - z^3 + 31/24*z^4 - 193/120*z^5 + 3161/1440*z^6 - 4427/1440*z^7 + 243773/53760*z^8 + O(z^9),
 z - 1/12*z^3 + 11/48*z^4 - 161/480*z^5 + 187/360*z^6 - 30973/40320*z^7 + 126773/107520*z^8 + O(z^9),
 1 - 1/12*z^3 + 5/48*z^4 - 77/480*z^5 + 319/1440*z^6 - 13273/40320*z^7 + 52691/107520*z^8 + O(z^9)]
```

Monodromy and factorization

Example: $L = z\partial^2 + \partial$



$$\underbrace{\begin{pmatrix} 1 & 0 \\ 2i\pi & 1 \end{pmatrix}}\begin{pmatrix} 1 \\ \log(z) \end{pmatrix} = \begin{pmatrix} 1 \\ \log(z) + 2i\pi \end{pmatrix}$$

monodromy of L around the singularity 0

Proposition. Let $L \in \mathcal{F}\langle\partial\rangle$ be a Fuchsian operator. There is an one-to-one correspondance:

$$L = L_1 L_2$$

$$\updownarrow \quad U = \text{Solutions}(L_2)$$

subspace U invariant under the action of the monodromy matrices

Ingredients: differential Galois theory + density theorem of Schlesinger

- rigorous arbitrary-precision computation of monodromy matrices is now available in SageMath [Mezzarobba, 2016]

Computing a generating set of the monodromy group

Note: Interval arithmetic is used for tracking error bounds (C library `Arb`, see https://arblib.org/)

```
from ore_algebra.analytic import monodromy_matrices

prec = 200 # number of bits of precision
mono = monodromy_matrices(L, 0, eps=RR.one()>>prec)
len(mono), mono[0].parent()
```

```
(3,
 Full MatrixSpace of 3 by 3 dense matrices over Complex ball field with 204 bits of precision)
```

```
mono[1]
```

```
[  [0.9023752279903480744087375221937813832946680049763035596478 +/- 2.70e-59] + [0.8027531570009388969251856842904631105096949991669769234085 +/- 6.92e-59]*I   [0.1952495440193038511825249556124372334106639900473928807044 +/- 5.86e-59] + [2.3944936859981222061496286314190737789806100016660461531831 +/- 5.99e-59]*I    [-3.265073560317101300634683069156592226072813860023866305746 +/- 2.16e-58] + [4.788987371996244412299257262838147557961220003332092306366 +/- 3.11e-58]*I]
[  [0.8295760190096610927236970403644779845772403178330170085085 +/- 3.21e-59] + [0.5081995377471570317410443674642944122148556664842497535518 +/- 7.80e-59]*I    [3.717727192433631092226145800679973117364264809985099427222 +/- 4.98e-58] + [0.9836009245056859365179112650714111755702886670315004928965 +/- 7.81e-59]*I     [3.607668060689407682952425111169212888281458699910872820866 +/- 4.76e-58] + [5.967201849011371873035822530142822351140577334063000985793 +/- 1.91e-58]*I]
[[-0.4391942025072435277596641396337936464649531576724326143423 +/- 9.66e-60] + [-0.0534114796233437916392257626595314284800040834503806459238 +/- 4.61e-59]*I  [-1.3100512102119895833174416614368772503294664074807014934347 +/- 3.60e-59] + [0.1068229592466875832784515253190628569600081669007612918475 +/- 4.80e-59]*I  [-1.6201024204239791666348833228737545006589328149614029868694 +/- 8.15e-59] + [-1.786354081506624833443096949361874286079983666198477416305 +/- 1.06e-58]*I]
```

Computing an invariant subspace

Function `invariant_subspace`

input: a list [M1, ..., Mr] of matrices with interval coefficients
output: a nontrivial subspace invariant under the action of M1, ..., Mr or `none`

> **Note:** 1) Some computation can fail if the precision is insufficient.
> 2) The output `none` is rigorous (whatever the exact matrices represented by M1, ..., Mr, there is no nontrivial invariant subspace).

```
from diffop_factorization import invariant_subspace

U = invariant_subspace(mono); U
```

```
[([0.71411650967043355058327867843024036969550785196 +/- 8.24e-48] + [0.351163834146178437164698587264508
04444359387665 +/- 7.05e-48]*I, [1.0000000000000000000000000000000000000000000000 +/- 1.62e-47] + [+/- 1.
62e-47]*I, [-0.32147087258239161235418033039243990757612303701 +/- 8.81e-48] + [0.08779095853654460929117
46468161270111108984692 +/- 4.61e-47]*I),
 ([0.0079065991711078720140767941058736293597253662 +/- 4.67e-47] + [-0.146122825623590047814319631633549
1560360708095 +/- 3.74e-47]*I, [1.000000000000000000000000000000000000000000000 +/- 3.90e-47] + [+/- 3.90
e-47]*I, [-0.4980233502072230319964808014735315926600686584 +/- 6.51e-47] + [-0.0365307064058975119535799
079083872890090177024 +/- 4.24e-47]*I)]
```

Guessing a candidate right factor

Hermite—Padé approximants: $\mathbb{C}^{\simeq}[[z]]_{<T} \to \mathbb{C}^{\simeq}(z)$

LLL algorithm: $\mathbb{C}^{\simeq} \to \overline{\mathbb{Q}}$

```
from diffop_factorization.complex_optimistic_field import ComplexOptimisticField
C = ComplexOptimisticField(prec, eps=RR.one()>>(prec//4))

basis = L.power_series_solutions(20); basis.reverse()
f = (vector(basis)*U[0].change_ring(C)).truncate() # candidate solution of a right factor of L

from diffop_factorization.guessing import hp_approximants, guess_exact_numbers
p, q, r = hp_approximants([f, f.derivative(), f.derivative().derivative()], 15); p, q, r
```

```
([-0.250000000000000000000000000000000000 +/- 1.32e-37] + [+/- 1.32e-37]*I,
 ([1.000000000000000000000000000000000000 +/- 2.82e-37] + [+/- 2.82e-37]*I)*z + [0.50000000000000000000000
0000000000000 +/- 1.34e-37] + [+/- 1.34e-37]*I,
 z^2 + ([1.500000000000000000000000000000000000 +/- 1.75e-38] + [+/- 1.75e-38]*I)*z + [0.500000000000000
0000000000000000000000 +/- 1.02e-38] + [+/- 1.02e-38]*I)
```

```
p, q, r
```

```
([-0.2500000000000000000000000000000000000 +/- 1.32e-37] + [+/- 1.32e-37]*I,
 ([1.00000000000000000000000000000000000 +/- 2.82e-37] + [+/- 2.82e-37]*I)*z + [0.50000000000000000000000
0000000000000 +/- 1.34e-37] + [+/- 1.34e-37]*I,
 z^2 + ([1.5000000000000000000000000000000000000 +/- 1.75e-38] + [+/- 1.75e-38]*I)*z + [0.500000000000000
0000000000000000000000 +/- 1.02e-38] + [+/- 1.02e-38]*I)
```

```
[p, q, r] = guess_exact_numbers([p, q, r])
p, q, r
```

```
(-1/4, z + 1/2, z^2 + 3/2*z + 1/2)
```

```
L2 = Diffops([p, q, r])
L2
```

```
(z^2 + 3/2*z + 1/2)*Dz^2 + (z + 1/2)*Dz - 1/4
```

Validating the result by Euclidean division

```
L % L2 # remainder of the right Euclidean division
```

```
0
```

```
L1 = L // L2 # quotient of the right Euclidean division
L1, L2, L1 * L2 == L
```

```
((4*z - 4)*Dz + 4, (z^2 + 3/2*z + 1/2)*Dz^2 + (z + 1/2)*Dz - 1/4, True)
```

Is L_2 irreducible?

```
invariant_subspace(monodromy_matrices(L2, 0)) is None
```

```
True
```

Comparison of running times

operator	order, deg. in z	DEtools (♒)	new package
fcc3 (♈)	3, 5	.182s	.148s (150*)
fcc4 (♈)	4, 10	.630s	1.29s (306*)
fcc5 (♈)	6, 17	61.9s	**7.98s** (475*)
fcc6 (♈)	8, 43	>10h	**159s** (1125*)
fcc4 × fcc3	7, 15	**1.88s**	27.6s (1659*)
fcc3 × fcc4	7, 15	**4.59s**	66.2s (3415*)
`lclm`(fcc3, fcc4)	7, 28	66.6s	85.0s (1636*)
fcc4^2	8, 20	122.s	108s (3387*)
`random4` × fcc3	7, 15	**2.04s**	144s (958*)
`random4` × `random3`	7, 15	**2.40s**	205s (1095*)
$(z^2\partial + 3)((z-3)\partial + 4z^5)$	2, 5	>10h	**1.74s** (150*)

the time of monodromy computation > 95% of the total time

(♒) command `DFactor` of the Maple package `DEtools` (author: van Hoeij)
(♈) irreducible operators at `http://koutschan.de/data/fcc1/` (probabilistic walks)
* = number of bits of precision needed

Future work:

- *implementing some tricks for simplest cases*
- *trying to adapt van der Hoeven's approach for the computation of an LCLM-decomposition*
- *clarifying the theoretical complexity (at least in function of the sufficient precision)*
- *experimenting symbolic-numeric approach on linked questions (e.g. algebraicity of solutions)*
- *extending our code to the non-Fuchsian case*

Thank you for listening!