CS 487 / · · ·
Introduction to Symbolic Computation

University of Waterloo
Éric Schost
eschost@uwaterloo.ca
Linear recurrences with polynomial coefficients
($n!$ and generalizations)
D-finite series

Def.

• A power series \( f(x) \) is **D-finite** if there exists a linear differential equation **with polynomial coefficients** such that

\[
q_d(x)f^{(d)} + q_{d-1}(x)f^{(d-1)} + \cdots + q_0(x)f = 0.
\]

• Equivalently, we can take **rational functions** as coefficients.

Examples.

• polynomials,
• rational functions,
• algebraic series (e.g., \( \sqrt{1 + x^2} \))
• \( \exp, \sin, \cos \),
• a lot more
Def.

A sequence $u_n$ is **P-recursive** if it satisfies a recurrence with polynomial coefficients

$$p_d(n)u_{n+d} + p_{d-1}(n)u_{n+d-1} + \cdots + p_0(n)u_n = 0$$

Examples.

- constant sequences,
- recurrences with constant coefficients,
- factorial, and generalizations.
Remark: matrix recurrence

\[ p_d(n)u_{n+d} + p_{d-1}(n)u_{n+d-1} + \cdots + p_0(n)u_n = 0 \]

means that

\[ u_{n+d} = -\frac{p_{d-1}(n)}{p_d(n)}u_{n+d-1} - \cdots - \frac{p_0(n)}{p_d(n)}u_n \]

so

\[
\begin{bmatrix}
  u_{n+d} \\
  u_{n+d-1} \\
  \vdots \\
  u_{n+1}
\end{bmatrix} =
\begin{bmatrix}
  -\frac{p_{d-1}(n)}{p_d(n)} & -\frac{p_{d-2}(n)}{p_d(n)} & \cdots & -\frac{p_0(n)}{p_d(n)} \\
  & & & \\
  & & & 1 \\
  & & & \\
  & & & \\
  & & & 1
\end{bmatrix}
\begin{bmatrix}
  u_{n+d-1} \\
  u_{n+d-2} \\
  \vdots \\
  u_n
\end{bmatrix}
\]
Theorem.

- The power series

\[ f = \sum_{i \geq 0} f_i x^i \]

is D-finite if and only if the sequence \((f_i)\) is P-recursive.

Examples.

- recurrence with constant coefficients \(\iff\) rational power series.
- \(f_i = 1/i! \iff\) exponential
Proof for the exponential

Suppose that $f$ is a solution of

$$f' = f.$$  

(We know that $f$ is the exponential.) With

$$f = \sum_{i \geq 0} f_i x^i,$$

we get

$$f' = \sum_{i \geq 0} (i + 1)f_{i+1} x^i.$$  

So

$$(i + 1)f_{i+1} = f_i.$$
Sketch of proof in general

In general, with

\[ f = \sum_{i \geq 0} f_i x^i, \]

we get

\[ f' = \sum_{i \geq 0} (i + 1)f_{i+1}x^i \quad \text{and} \quad f'' = \sum_{i \geq 0} (i + 1)(i + 2)f_{i+2}x^i, \quad \ldots \]

Multiplying by a monomial shifts the coefficients:

\[ x^\ell f' = \sum_{i \geq 0} (i + 1)f_{i+1}x^{i+\ell} = \sum_{i \geq \ell} (i - \ell + 1)f_{i-\ell+1}x^i, \quad \ldots \]

So extracting coefficients gives a recurrence on the \( f_i \).
Consider the factorial

\[ f_i = i!, \quad \text{so that} \quad f_{i+1} = (i + 1)f_i. \]

Let \( f = \sum_{i \geq 0} f_i x^i \).

Multiply by \( x^{i+1} \) and sum over all \( i \geq 0 \).

\[
\sum_{i \geq 0} f_{i+1} x^{i+1} = f - 1 \quad \text{and} \quad \sum_{i \geq 0} (i + 1)f_i x^{i+1} = x(xf' + f).
\]

So

\[ x^2f' + (x - 1)f = -1 \quad \text{or} \quad x^2f'' + (3x - 1)f' - f = 0. \]
Our questions

1. Computing one term in a P-recursive sequence
   • binary splitting
   • baby steps / giant steps

2. Computing several terms
   • unroll the recurrence
   • solve the differential equation using Newton iteration
Examples

1. Compute the first 100 coefficients $c_i$ of $bm(x + 1)^{10000}$
We know they are binomial coefficients, so we get

$$c_{i+1} = \frac{D - i}{i + 1} c_i$$

2. Compute the first 100 coefficients $d_i$ of $(x + 2)^{10000}(x + 1)^{1000}$
The generating series $S = \sum_{i \geq 0} d_i x^i$ satisfies

$$\frac{S'}{S} = 10000 \frac{1}{x + 2} + 1000 \frac{1}{x + 1}$$

which gives

$$(i - 1)d_{i-1} + 3id_i + 2(i + 1)d_{i+1} = 11000d_{i-1} + 12000d_i$$
Computing one term
Binary splitting

This is the method you want to use when the **coefficients size** matters
  - quasi-optimal algorithms exist, taking bit-size into account
  - not useful modulo $p$

**Example:** factorial.
  - We write $M_Z(n)$ for the cost of **multiplying integers of size** $n$.
  - The factorial $n!$ has about $n \log n$ digits.

**Prop.**
  - Using **binary splitting**, one can compute $n!$ in $O(M_Z(n \log n) \log n)$ bit operations.
The algorithm in a nutshell

It boils down to computing $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdots$ in a clever way.

Naive:

- $2 = 1 \cdot 2$
- $6 = 2 \cdot 3$
- $24 = 6 \cdot 4$
- $120 = 24 \cdot 5$
- $720 = 120 \cdot 6$
- $5040 = 720 \cdot 7$
- $40320 = 5040 \cdot 8$
- $362880 = 40320 \cdot 9$

**Consequence:** quadratic time!
The algorithm in a nutshell

It boils down to computing $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdots$ in a clever way.
Let $P(a, b) = a(a + 1) \cdots b$, so that we want $P(1, n)$.

**Binary splitting:**

$$P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.$$
Let $P(a, b) = a(a + 1) \cdots b$, so that we want $P(1, n)$.

Binary splitting:

$$P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.$$ 

Cost:

$$C(a, b) = C(a, m) + C(m, b) + M_\mathbb{Z}(\log P(m, b)) \leq 2C(m, b) + M_\mathbb{Z}(\log P(m, b)).$$
Let \( P(a, b) = a(a + 1) \cdots b \), so that we want \( P(1, n) \).

Binary splitting:

\[
P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.
\]

Cost:

\[
C(a, b) = C(a, m) + C(m, b) + M_\mathbb{Z}(\log P(m, b)) \\
\leq 2C(m, b) + M_\mathbb{Z}(\log P(m, b)).
\]

The splitting scheme

\[
\bullet \quad C(1, n) \leq 2C(n/2, n) + M_\mathbb{Z}(\log P(n/2, n))
\]
Splitting

Let $P(a, b) = a(a + 1) \cdots b$, so that we want $P(1, n)$.

Binary splitting:

$$P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.$$ 

Cost:

$$C(a, b) = C(a, m) + C(m, b) + M_{\mathbb{Z}}(\log P(m, b))$$

$$\leq 2C(m, b) + M_{\mathbb{Z}}(\log P(m, b)).$$

The splitting scheme

- $C(1, n) \leq 2C(n/2, n) + M_{\mathbb{Z}}(\log P(n/2, n))$
- $C(n/2, n) \leq 2C(3n/4, n) + M_{\mathbb{Z}}(\log P(3n/4, n))$
Let $P(a, b) = a(a + 1) \cdots b$, so that we want $P(1, n)$.

Binary splitting:

$$P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.$$  

Cost:

$$C(a, b) = C(a, m) + C(m, b) + M_{\mathbb{Z}}(\log P(m, b))$$

$$\leq 2C(m, b) + M_{\mathbb{Z}}(\log P(m, b)).$$

The splitting scheme

- $C(1, n) \leq 2C(n/2, n) + M_{\mathbb{Z}}(\log P(n/2, n))$
- $C(n/2, n) \leq 2C(3n/4, n) + M_{\mathbb{Z}}(\log P(3n/4, n))$
- $C(3n/4, n) \leq 2C(7n/8, n) + M_{\mathbb{Z}}(\log P(7n/8, n))$
Let $P(a, b) = a(a + 1) \cdots b$, so that we want $P(1, n)$.

Binary splitting:

$$P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.$$ 

Cost:

$$C(a, b) = C(a, m) + C(m, b) + M_\mathbb{Z}(\log P(m, b))$$

$$\leq 2C(m, b) + M_\mathbb{Z}(\log P(m, b)).$$

The splitting scheme

- $C(1, n) \leq 2C(n/2, n) + M_\mathbb{Z}(\log P(n/2, n))$
- $2C(n/2, n) \leq 4C(3n/4, n) + 2M_\mathbb{Z}(\log P(3n/4, n))$
Splitting

Let \( P(a, b) = a(a + 1) \cdots b \), so that we want \( P(1, n) \).

Binary splitting:

\[
P(a, b) = P(a, m)P(m, b) \quad \text{with} \quad m = \lfloor (a + b)/2 \rfloor.
\]

Cost:

\[
C(a, b) = C(a, m) + C(m, b) + M_Z(\log P(m, b)) \\
\leq 2C(m, b) + M_Z(\log P(m, b)).
\]

The splitting scheme

- \( C(1, n) \leq 2C(n/2, n) + M_Z(\log P(n/2, n)) \)
- \( 2C(n/2, n) \leq 4C(3n/4, n) + 2M_Z(\log P(3n/4, n)) \)
- \( 4C(3n/4, n) \leq 8C(7n/8, n) + 4M_Z(\log P(7n/8, n)) \)
Solving the recurrence

These equalities give (for any $k \leq \log(n)$)

$$C(1, n) \leq 2^k C(n - \frac{n}{2^k}, n) + \sum_{j=1}^{k} 2^{j-1} M_{\mathbb{Z}}(\log P(n - \frac{n}{2^j}, n)).$$

Simplifications

- remember that

$$P(n - \frac{n}{2^j}, n) = (n - \frac{n}{2^j}) \cdots n \leq n^{n/2^j}$$

- so its log is $\leq \frac{n}{2^j} \log n$,

- so its contribution is $\leq M_{\mathbb{Z}}(n \log n)$

(using $M_{\mathbb{Z}}(t/2) \leq \frac{1}{2} M_{\mathbb{Z}}(t)$)
Solving the recurrence

Putting everything together gives

\[ C(1, n) \leq 2^k C(n - \frac{n}{2^k}, n) + kM_\mathbb{Z}(n \log n). \]

We stop the recursion for \( k = \log n \), which gives

\[ C(1, n) \in O(M_\mathbb{Z}(n \log n) \log n). \]
Second example: computing $e = \exp(1)$

The sequence

$$e_n = \sum_{k=0}^{n} \frac{1}{k!}$$

converges to $e$, and $0 \leq e - e_n \leq \frac{1}{n!}$.

**Consequence**

- To compute $m$ digits of $e$, compute $e_n$, with

$$n \approx \frac{m}{\log m}$$
The recursion

The sequence $f_n = 1/n!$ satisfies the recurrence

$$(n + 1)f_{n+1} = f_n.$$  

Because $e_{n+1} - e_n = f_{n+1}$, we get

$$(n + 1)(e_{n+1} - e_n) = (n + 1)f_{n+1} = f_n = e_n - e_{n-1},$$

which becomes

$$\begin{bmatrix} e_{n+1} \\ e_n \end{bmatrix} = \frac{1}{n+1} \begin{bmatrix} n + 2 & -1 \\ n + 1 & 0 \end{bmatrix} \begin{bmatrix} e_n \\ e_{n-1} \end{bmatrix} = \frac{1}{n+1} M(n) \begin{bmatrix} e_n \\ e_{n-1} \end{bmatrix}. $$

So to compute $e_{n-1}$ and $e_n$, we actually compute

$$\frac{1}{n!} M(n-1) \cdots M(1).$$

Same thing as the factorial!
Example

Take $n = 30$; then $M(n - 1) \cdots M(1)$ is

$$
\begin{bmatrix}
14129154237824555961821165045504732 & -5906315583646633144095602165504732 \\
14129154237824555961821165045504731 & -5906315583646633144095602165504731
\end{bmatrix}
$$

and $n! = 8222838654177922817725562880000000$.

Our approximation to $\exp(1)$ is the first entry of

$$
\frac{1}{n!} M(n-1) \cdots M(1) \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \frac{5587998223000619694886681981376183}{2055709663544480704431390720000000}.
$$

Gives 117 correct bits.

Remark: works for cos, sin, . . . , all D-finite functions.
Baby steps / giant steps

This is the method to use when coefficient size does not matter.

Prop.
- Consider $u_n$ defined by a recurrence of order $d$ with coefficients of degree $p$.
- Then the $n$th term can be computed in $O(M(\sqrt{n}) \log n)$, where the big-Oh depends on $d$ and $p$.

Example
- The sequence $u_{n+1} = (n + 1)u_n$, computed modulo an integer $N$.
- This leads to the best deterministic, proved algorithm for factoring integers.
Evaluation and interpolation

- Given a polynomial $P$ of degree $m - 1$, and $m$ evaluation points $a_0, \ldots, a_{m-1}$ one can compute $P(a_0), \ldots, P(a_{m-1})$ in $O(M(m) \log (m))$ operations.
- Conversely, given the values, one can recover $P$ in the same cost.

Main ideas

- **Divide-and-conquer**: replace the original problem by the evaluation of a polynomial $P_0$ at the first half of the points and a polynomial $P_1$ at the second half.
- **Cost**: $C(n) \leq 2C(n/2) + O(M(n))$. 
The example of the factorial

Consider the sequence \( u_{n+1} = (n+1)u_n, u_0 = 1 \).

To compute \( u_n \), let \( m = \sqrt{n} \) and introduce

\[
P = (x + 1) \cdots (x + m).
\]

Then \( u_n \) is given by

\[
u_n = P(0) P(m) P(2m) \cdots P((m - 1)m).
\]

Algorithm

- Compute \( P \) (divide-and-conquer) \( O(M(m) \log m) \)
- Evaluate it at \( 0, m, \ldots, (m - 1)m \) \( O(M(m) \log m) \)
- Multiply the values \( O(m) \)
Application to factoring integers

Suppose you want to factor $p \in \mathbb{N}$ into primes.

- It’s enough to find all prime factors $< \sqrt{p}$.
- Testing one number mod $p$ costs $O((\log p)^{O(1)})$.
- So naive cost $O(\sqrt{p}(\log p)^{O(1)})$

Better: let $n = \sqrt{p}$ and $m = \sqrt{n}$, and compute the slices

$$a_0 = 1 \cdots m \mod p, \ a_1 = (m+1) \cdots (2m) \mod p, \ldots a_{m-1} = (m^2-m+1) \cdots m^2 \mod p,$$

- cost almost linear in $\sqrt{p}$.
- if $\gcd(a_i, p) = 1$, no divisor in the slice $i$.
- as soon as you found $\gcd(a_i, p) \neq 1$, test all elements in $a_i$.
- repeat.