Computing terms of linearly recurrent sequences

Marc Mezzarobba

November 2, 2023
Summary of last week’s results

**Theorem.** Let \( f = \sum_{n \geq 0} f_n x^n \in K[[x]] \).

- \( f \) is **D-finite** \( \iff \) \((f_n)_{n \in \mathbb{N}} \) is **P-recursive**
- \( f \) satisfies a linear ODE with coefficients in \( K[x] \) \( \iff \) \((f_n) \) satisfies a linear recurrence with coefficients in \( K[x] \)
- \( \dim \text{span}_{K(x)}(f, f', f'', \ldots) < \infty \)

**Theorem.**

- The **sum**, the **product**, the **Hadamard product** of D-finite series are D-finite
- **Algebraic series** are D-finite

...and the corresponding differential equations can be computed by linear algebra
Exercises from last week
Exercise 1

Prove the following identity of formal power series:

\[
\arcsin(x)^2 = \sum_{k=0}^{\infty} \frac{k!}{\frac{1}{2} \frac{3}{2} \cdots (k + \frac{1}{2})} \frac{x^{2k+2}}{2k+2}.
\]

For this:

1. Check that \( y(x) = \arcsin(x) \) is solution to \( (1 - x^2) y''(x) = x y'(x) \).
2. Deduce a linear differential equation satisfied by \( z(x) = y(x)^2 \).
3. Deduce a linear recurrence relation satisfied by the coefficients of the series.
Exercise 2

Sketch two algorithms for computing the first $n$ terms of the series

$$\exp\left( x (\sqrt{1+x} + \sqrt{2+x} + \cdots + \sqrt{k+x}) \right)$$

in $\tilde{O}(n)$ ops for fixed $k$: one based on Newton iteration, one based on D-finiteness. What are their respective complexities w.r.t. $n$? Which do you think is better in practice? (Some handwaving is okay.)
Exercise 2

Sketch two algorithms for computing the first $n$ terms of the series

$$\exp\left(x \left(\sqrt{1+x} + \sqrt{2+x} + \cdots + \sqrt{k+x}\right)\right)$$

in $\tilde{O}(n)$ ops for fixed $k$: one based on Newton iteration, one based on D-finiteness. What are their respective complexities w.r.t. $n$?

Which do you think is better in practice? (Some handwaving is okay.)

**Solution.** (Assuming $\sqrt{k} \in \mathbb{Q}$)

- Newton: Compute each $\sqrt{k+x} + O(x^n)$, sum, compute $\exp(\text{sum})$ $O(M(n))$
Exercise 2

Sketch two algorithms for computing the first $n$ terms of the series

$$\exp\left( x (\sqrt{1+x} + \sqrt{2+x} + \cdots + \sqrt{k+x}) \right)$$

in $\tilde{O}(n)$ ops for fixed $k$: one based on Newton iteration, one based on D-finiteness. What are their respective complexities w.r.t. $n$? Which do you think is better in practice? (Some handwaving is okay.)

Solution. (Assuming $\sqrt{k} \in \mathbb{K}$)

- Newton: Compute each $\sqrt{k+x} + O(x^n)$, sum, compute $\exp(\text{sum})$ $O(M(n))$
- D-finiteness: Compute a diff. eq. , then a rec., for the whole expression, use that recurrence $O(1)$ $O(n)$
Exercise 2

Sketch two algorithms for computing the first \( n \) terms of the series

\[
\exp\left( x (\sqrt{1+x} + \sqrt{2+x} + \cdots + \sqrt{k+x}) \right)
\]

in \( \tilde{O}(n) \) ops for fixed \( k \): one based on Newton iteration, one based on D-finiteness.

What are their respective complexities w.r.t. \( n \)?

Which do you think is better in practice? (Some handwaving is okay.)

Solution. (Assuming \( \sqrt{k} \in \mathbb{K} \))

- Newton: Compute each \( \sqrt{k+x} + O(x^n) \), sum, compute \( \exp(\text{sum}) \) \( O(M(n)) \)
- D-finiteness: Compute a diff. eq. , then a rec., for the whole expression, use that recurrence \( O(1) \) \( O(n) \)

...but algebraic function of high degree ⇒ huge equations, huge ‘constant’

\( (r = 16, d \geq 300 \text{ for } k = 4) \)
1 C-finite sequences
1.1 Introduction
Setting

$I$ — an effective field (or integral domain)

**Definition.** A sequence $(u_n) \in I^\mathbb{N}$ is called **C-finite** when it satisfies a linear recurrence

$$\forall n \in \mathbb{N}, \quad u_{n+s} + c_{s-1} u_{n+1} + \cdots + c_0 u_n = 0 \quad \text{with } c_i \in I.$$  
(Note the leading coefficient $= 1$)

The polynomial $\chi(t) = t^s + c_{s-1} t^{s-1} + \cdots + c_0$ is called the **characteristic polynomial** of the recurrence.

$(u_n)_{n \in \mathbb{N}}$ is entirely determined by $(c_0, \ldots, c_{s-1})$ and $(u_0, \ldots, u_{s-1}), \quad s = \text{order of rec.}$
Setting

\( \mathbb{K} \) — an effective field (or integral domain)

**Definition.** A sequence \((u_n) \in \mathbb{K}^N\) is called **C-finite** when it satisfies a linear recurrence

\[
\forall n \in \mathbb{N}, \quad u_{n+s} + c_{s-1} u_{n+1} + \cdots + c_0 u_n = 0 \quad \text{with } c_i \in \mathbb{K}.
\]

(Note the leading coefficient = 1)

The polynomial \(\chi(t) = t^s + c_{s-1} t^{s-1} + \cdots + c_0\) is called the **characteristic polynomial** of the recurrence.

\((u_n)_{n \in \mathbb{N}}\) is entirely determined by \((c_0, \ldots, c_{s-1})\) and \((u_0, \ldots, u_{s-1})\), \(s = \text{order of rec.}\)

\[
 u_n = \sum_{\chi(\alpha) = 0} p_\alpha(n) \alpha^n \quad \text{where } p_\alpha \in \mathbb{K}[n]_{\text{mult}(\alpha, \chi)}
\]

In terms of operators: \(\chi(S) \cdot (u_n)_{n \in \mathbb{N}} = 0\)
Setting

\(\mathbb{K}\) — an effective field (or integral domain)

**Definition.** A sequence \((u_n) \in \mathbb{K}^N\) is called **C-finite** when it satisfies a linear recurrence

\[\forall n \in \mathbb{N}, \quad u_{n+s} + c_{s-1} u_{n+1} + \cdots + c_0 u_n = 0 \quad \text{with } c_i \in \mathbb{K}.\]

(Note the leading coefficient = 1)

The polynomial \(\chi(t) = t^s + c_{s-1} t^{s-1} + \cdots + c_0\) is called the **characteristic polynomial** of the recurrence.

\((u_n)_{n \in \mathbb{N}}\) is entirely determined by \((c_0, \ldots, c_{s-1})\) and \((u_0, \ldots, u_{s-1})\), \(s = \text{order of rec.}\)

\[u_n = \sum_{\chi(\alpha)=0} p_\alpha(n) \alpha^n \quad \text{where } p_\alpha \in \mathbb{K}[n]_{< \text{mult}(\alpha, \chi)}\]

In terms of operators: \(\chi(S) \cdot (u_n)_{n \in \mathbb{N}} = 0\)

**Notation.** \(x_{i:j} = (x_i, \ldots, x_{j-1})\)
By any other name...

**Linear Feedback Shift Registers (LFSR)**
- Circuits, cryptography...

\[ u_{n+16} = u_{n+5} + u_{n+3} + u_{n+2} + u_n \text{ (over } \mathbb{F}_2) \]

**Infinite Impulse Response (FIR) filters**
- Signal processing, control

\[ y_n = 0.5 y_{n-1} + x_n \text{ (over } \mathbb{R}) \]

inhomogeneous
First $N$ terms, $N$th term

\[ u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0 \]

**Problems.** Given the coefficients $c_0, \ldots, c_{s-1}$ and initial values $u_0, \ldots, u_{s-1}$:

a) Compute $(u_0, \ldots, u_{N-1})$

b) Compute $u_N$

Complexity models: operations in $\mathbb{K}$ (“ops”)

- sometimes binary operations for $\mathbb{K} = \mathbb{Z}$
First $N$ terms, $N$th term

\[ u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0 \]

**Problems.** Given the coefficients $c_0, \ldots, c_{s-1}$ and initial values $u_0, \ldots, u_{s-1}$:

a) Compute $(u_0, \ldots, u_{N-1})$

b) Compute $u_N$

Complexity models: operations in $\mathbb{K}$ ("ops")
sometimes binary operations for $\mathbb{K} = \mathbb{Z}$

Direct algorithm:

\[
\begin{align*}
    u_s & := -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \\
    u_{s+1} & := -(c_{s-1} u_s + \cdots + c_0 u_1) \\
    & \vdots
\end{align*}
\]

$O(sN)$ ops
**First N terms, Nth term**

\[ u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0 \]

**Problems.** Given the coefficients \( c_0, \ldots, c_{s-1} \) and initial values \( u_0, \ldots, u_{s-1} \):

a) Compute \( (u_0, \ldots, u_{N-1}) \)

b) Compute \( u_N \)

Complexity models: operations in \( \mathbb{K} \) (“ops”)

sometimes binary operations for \( \mathbb{K} = \mathbb{Z} \)

Direct algorithm:

\[
\begin{align*}
\{ & u_s := -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \quad \text{O}(s N) \text{ ops} \\
& u_{s+1} := -(c_{s-1} u_s + \cdots + c_0 u_1) \\
& \vdots 
\}
\]

Over \( \mathbb{Z} \): explicit formula \( \Rightarrow |u_n| \leq a n^m |\lambda|^n \)

\( \lambda = \text{root of } \chi \text{ of maximum modulus} \)

Bit sizes:

\( \text{size}(u_N) \leq N \log_2 |\lambda| + o(N) \)

\( \text{size}(u_0:N) \leq \frac{1}{2} N^2 \log_2 |\lambda| + o(N^2) \) reached
The direct algorithm over $\mathbb{Z}$

Direct algorithm:
\[
\begin{align*}
    u_s &:= -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \\
    u_{s+1} &:= -(c_{s-1} u_s + \cdots + c_0 u_1) \\
    & \vdots
\end{align*}
\]

$|u_n| \leq 2^{Kn}$

Output size can reach $\Omega(KN^2)$ for $N$ terms
$\Omega(KN)$ for one term
The direct algorithm over $\mathbb{Z}$

Direct algorithm:

\[
\begin{align*}
    u_s &:= -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \\
    u_{s+1} &:= -(c_{s-1} u_s + \cdots + c_0 u_1) \\
    &\vdots
\end{align*}
\]

Bit operations:

\[
\sum_{n=s}^{N-1} C_s M_{\mathbb{Z}}(K n)
\]

Output size can reach $\Omega(K N^2)$ for $N$ terms  
$\Omega(K N)$ for one term
The direct algorithm over \( \mathbb{Z} \)

Direct algorithm:

\[
\begin{align*}
  u_s &:= -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \\
  u_{s+1} &:= -(c_{s-1} u_s + \cdots + c_0 u_1) \\
  \vdots
\end{align*}
\]

\(|u_n| \leq 2^{Kn}\)

Bit operations:

\[
\sum_{n=s}^{N-1} C s M_\mathbb{Z}(Kn) \leq C s M_\mathbb{Z}(K \frac{N(N-1)}{2})
\]

Output size can reach \( \Omega(KN^2) \) for \( N \) terms

\( \Omega(KN) \) for one term
The direct algorithm over $\mathbb{Z}$

Direct algorithm:

\[
\begin{align*}
  u_s &:= -(c_{s-1} u_{s-1} + \cdots + c_0 u_0) \\
  u_{s+1} &:= -(c_{s-1} u_s + \cdots + c_0 u_1) \\
  &\vdots
\end{align*}
\]

\[|u_n| \leq 2^{Kn}\]

Bit operations:

\[
\sum_{n=s}^{N-1} C s M_{\mathbb{Z}}(Kn) \leq C s M_{\mathbb{Z}} \left( K \frac{N(N-1)}{2} \right)
\]

\[= O(s M_{\mathbb{Z}}(KN^2))\]

Output size can reach $\Omega(KN^2)$ for $N$ terms
$\Omega(KN)$ for one term
The direct algorithm over \( \mathbb{Z} \)

Direct algorithm: \[
\begin{align*}
\mathbf{u}_s & := -(c_{s-1} \mathbf{u}_{s-1} + \cdots + c_0 \mathbf{u}_0) \\
\mathbf{u}_{s+1} & := -(c_{s-1} \mathbf{u}_s + \cdots + c_0 \mathbf{u}_1) \\
& \vdots
\end{align*}
\]

\[|\mathbf{u}_n| \leq 2^{Kn}\]

Bit operations: \[
\sum_{n=s}^{N-1} C_s M_\mathbb{Z}(K n) \leq C_s M_\mathbb{Z}\left(K \frac{N(N-1)}{2}\right)
\]

\[= O(s M_\mathbb{Z}(K N^2))
\]

\[= O(M_\mathbb{Z}(N^2))\quad \text{for a fixed rec.}
\]

Output size can reach \( \Omega(K N^2) \) for \( N \) terms

\( \Omega(K N) \) for one term
Nth term by binary powering in $\mathbb{K}^{s \times s}$

Matrix form of the recurrence:

$$
\begin{pmatrix}
  u_{n+1} \\
  u_{n+2} \\
  \vdots \\
  u_{n+s}
\end{pmatrix}
= 
\begin{pmatrix}
  1 & & \\
  & \ddots & \\
  & & 1 \\
  -c_0 & -c_1 & \cdots & -c_{s-1}
\end{pmatrix}
\begin{pmatrix}
  u_n \\
  u_{n+1} \\
  \vdots \\
  u_{n+s-1}
\end{pmatrix}
$$

\[ A \]

\[ A \]

**Algorithm.** *Input: $u_{0:s}, c_{0:s}, N$  \quad Output: $u_{N:N+s}$*

1. Compute $B = A^N$ by binary powering
2. Return $B \cdot (u_0, \ldots, u_{s-1})^T$
Nth term by binary powering in $\mathbb{K}^{s \times s}$

Matrix form of the recurrence:

$$
\begin{pmatrix}
    u_{n+1} \\
    u_{n+2} \\
    \vdots \\
    u_{n+s}
\end{pmatrix} = 
\underbrace{
\begin{pmatrix}
    1 & & & \\
    & \ddots & & \\
    & & 1 & \\
    -c_0 & -c_1 & \cdots & -c_{s-1}
\end{pmatrix}
}_{A}
\begin{pmatrix}
    u_n \\
    u_{n+1} \\
    \vdots \\
    u_{n+s-1}
\end{pmatrix}
$$

**Algorithm.** Input: $u_{0:s}, c_{0:s}, N$  \hspace{1cm} Output: $u_{N:N+s}$

1. Compute $B = A^N$ by binary powering  \hspace{1cm} $O(s^\theta \log N)$
2. Return $B \cdot (u_0, \ldots, u_{s-1})^T$
Nth term by binary powering in $\mathbb{K}^{s \times s}$

Matrix form of the recurrence:

$$
\begin{pmatrix}
    u_{n+1} \\
    u_{n+2} \\
    \vdots \\
    u_{n+s}
\end{pmatrix}
= 
\begin{pmatrix}
    1 & & \\
    & \ddots & \\
    & & 1 \\
    -c_0 & -c_1 & \cdots & -c_{s-1}
\end{pmatrix}
\begin{pmatrix}
    u_n \\
    u_{n+1} \\
    \vdots \\
    u_{n+s-1}
\end{pmatrix}
$$

Algorithm. Input: $u_{0:s}, c_{0:s}, N$ Output: $u_{N:N+s}$

1. Compute $B = A^N$ by binary powering $O(s^3 \log N)$
2. Return $B \cdot (u_0, \ldots, u_{s-1})^T$ $O(s^2)$
Nth term by binary powering in $\mathbb{K}^{s \times s}$

Matrix form of the recurrence:

$$
\begin{pmatrix}
  u_{n+1} \\
  u_{n+2} \\
  \vdots \\
  u_{n+s}
\end{pmatrix}
= \begin{pmatrix}
  1 & & \\
  & \ddots & \\
  -c_0 & -c_1 & \cdots & -c_{s-1}
\end{pmatrix}
\begin{pmatrix}
  u_n \\
  u_{n+1} \\
  \vdots \\
  u_{n+s-1}
\end{pmatrix}
$$

**Algorithm.** Input: $u_0:s$, $c_0:s$, $N$  
Output: $u_{N:N+s}$

1. Compute $B = A^N$ by binary powering  
   $O(s^\theta \log N)$
2. Return $B \cdot (u_0, \ldots, u_{s-1})^T$  
   $O(s^2)$

Integers: $\|A^n\| \leq a |\lambda|^n$ (for simple eigenvalues) $\leq 2^{Kn}$  
$\lambda$ = eigenvalue of maximum modulus
Nth term by binary powering in $\mathbb{K}^{s \times s}$

Matrix form of the recurrence:

$$
\begin{pmatrix}
    u_{n+1} \\
    u_{n+2} \\
    \vdots \\
    u_{n+s}
\end{pmatrix} =
\begin{pmatrix}
    1 & & & \\
    & 1 & & \\
    & & \ddots & \\
    & & & 1 \\
    -c_0 & -c_1 & \cdots & -c_{s-1}
\end{pmatrix}
\begin{pmatrix}
    u_n \\
    u_{n+1} \\
    \vdots \\
    u_{n+s-1}
\end{pmatrix}
$$

**Algorithm.** *Input:* $u_{0:s}$, $c_{0:s}$, $N$    \hspace{1cm} *Output:* $u_{N:N+s}$

1. Compute $B = A^N$ by binary powering \hspace{1cm} $O(s^\theta \log N)$
2. Return $B \cdot (u_0, \ldots, u_{s-1})^T$ \hspace{1cm} $O(s^2)$

Integers: $\|A^n\| \leq a |\lambda|^n$ (for simple eigenvalues) $\leq 2^{Kn}$ \hspace{1cm} $\lambda =$ eigenvalue of maximum modulus

Bit ops: $s^\theta M_{\mathbb{Z}}(K) + \cdots + s^\theta M_{\mathbb{Z}}\left(\frac{N}{4} K\right) + s^\theta M_{\mathbb{Z}}\left(\frac{N}{2} K\right) = O(s^\theta M_{\mathbb{Z}}(KN))$
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$\underbrace{(S^s + c_{s-1} S^{s-1} + \cdots + c_1 S + c_0)} \cdot (u_n)_{n \in \mathbb{N}} = 0$$  \hspace{1cm} \text{where} \hspace{1cm} S: (u_n)_{n \in \mathbb{N}} \mapsto (u_{n+1})_{n \in \mathbb{N}}$$
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$(S^s + c_{s-1} S^{s-1} + \cdots + c_1 S + c_0) \cdot (u_n)_{n \in \mathbb{N}} = 0 \quad \text{where} \quad S: (u_n)_{n \in \mathbb{N}} \mapsto (u_{n+1})_{n \in \mathbb{N}}$$

When $P = Q \chi + R$ in $\mathbb{K}[t]$, \quad $P(S) \cdot (u_n)_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}$

[Fiduccia 1985]
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$\chi(S) = (S^s + c_{s-1}S^{s-1} + \cdots + c_1S + c_0) \cdot (u_n)_{n \in \mathbb{N}} = 0$$

where $S: (u_n)_{n \in \mathbb{N}} \mapsto (u_{n+1})_{n \in \mathbb{N}}$

When $P = Q\chi + R$ in $\mathbb{K}[t]$, $P(S) \cdot (u_n)_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}$

For $R = t^n \text{rem} \chi$, $(u_{N+n})_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}$

$$R = R_0 + \cdots + R_{s-1}t^{s-1},$$

[Fiduccia 1985]
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$\chi(S) \equiv (S^s + c_{s-1} S^{s-1} + \cdots + c_1 S + c_0) \cdot (u_n)_{n \in \mathbb{N}} = 0$$

where $S: (u_n)_{n \in \mathbb{N}} \mapsto (u_{n+1})_{n \in \mathbb{N}}$

When $P = Q \chi + R$ in $\mathbb{K}[t]$, $P(S) \cdot (u_n)_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}$

For $R = t^N \text{rem} \chi$, $(u_{N+n})_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}$

$R = R_0 + \cdots + R_{s-1} t^{s-1}$, $u_N = R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}$
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$
\chi(S)
= (S^s + c_{s-1} S^{s-1} + \cdots + c_1 S + c_0) \cdot (u_n)_{n \in \mathbb{N}} = 0 \quad \text{where} \quad S : (u_n)_{n \in \mathbb{N}} \mapsto (u_{n+1})_{n \in \mathbb{N}}
$$

When $P = Q \chi + R$ in $\mathbb{K}[t]$,

$$
P(S) \cdot (u_n)_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}
$$

For $R = t^N \operatorname{rem} \chi$,

$$
R = R_0 + \cdots + R_{s-1} t^{s-1}, \quad (u_{N+n})_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}
\quad u_N = R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}
$$

**Algorithm.** Input: $u_{0:s}$, $c_{0:s}$, $N$  \quad Output: $u_N$

1. Compute $R = t^N \operatorname{rem} \chi$ by binary powering modulo $\chi$
2. Return $R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}$

Cf. $u_n = \sum_{\chi(\alpha) = 0} p_\alpha(n) \alpha^n$
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$\chi(S) \left( S^s + c_{s-1} S^{s-1} + \cdots + c_1 S + c_0 \right) \cdot (u_n)_{n \in N} = 0 \quad \text{where} \quad S : (u_n)_{n \in N} \mapsto (u_{n+1})_{n \in N}$$

When $P = Q \chi + R$ in $\mathbb{K}[t]$, \quad $P(S) \cdot (u_n)_{n \in N} = R(S) \cdot (u_n)_{n \in N}$

For $R = t^N \text{rem} \chi$, \quad $(u_{N+n})_{n \in N} = R(S) \cdot (u_n)_{n \in N}$

$$R = R_0 + \cdots + R_{s-1} t^{s-1}, \quad u_N = R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}$$

**Algorithm.** Input: $u_0:s$, $c_0:s$, $N$ \quad **Output:** $u_N$

1. Compute $R = t^N \text{rem} \chi$ by binary powering modulo $\chi$ \quad $O(M(s) \log N))$
2. Return $R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}$

Cf. $u_n = \sum_{\chi(\alpha) = 0} p_\alpha(n) \alpha^n$
Nth term by binary powering in $\mathbb{K}[t]/\langle \chi \rangle$

Operator form of the recurrence:

$$\chi(S) = \underbrace{(S^s + c_{s-1}S^{s-1} + \cdots + c_1S + c_0)}_{S^s}(u_n)_{n \in \mathbb{N}} = 0$$

where $S: (u_n)_{n \in \mathbb{N}} \mapsto (u_{n+1})_{n \in \mathbb{N}}$

When $P = Q\chi + R$ in $\mathbb{K}[t]$,

$$P(S) \cdot (u_n)_{n \in \mathbb{N}} = R(S) \cdot (u_n)_{n \in \mathbb{N}}$$

For $R = t^N \text{rem} \chi$,

$$R = R_0 + \cdots + R_{s-1} t^{s-1},$$

$$u_N = R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}$$

**Algorithm.** *Input:* $u_{0:s}, c_{0:s}, N$  
*Output:* $u_N$

1. Compute $R = t^N \text{rem} \chi$ by binary powering modulo $\chi$  
   $O(M(s) \log N))$

2. Return $R_0 u_0 + R_1 u_1 + \cdots + R_{s-1} u_{s-1}$  
   $O(s)$

Cf. $u_n = \sum_{\chi(\alpha)=0} p_\alpha(n) \alpha^n$  

Variant: $(u_N, \ldots, u_{N+s-1})$ in $O(M(s) \log(N) + s^2)$ ops
Fast matrix powering

Goal: Given $A \in \mathbb{K}^{s \times s}$ and $N \in \mathbb{N}$, compute $A^N$

Binary powering in $\mathbb{K}^{s \times s}$ costs $\mathcal{O}(s \log N)$ ops

For distinct roots, $\approx$ diagonalizing $A$ as $P^{-1} \Delta P$ and computing $P^{-1} \Delta^n P$ in $\bar{\mathbb{K}}$
Fast matrix powering

Goal: Given $A \in K^{s \times s}$ and $N \in \mathbb{N}$, compute $A^N$

Binary powering in $K^{s \times s}$ costs $O(s^\theta \log N)$ ops

For distinct roots, $\approx$ diagonalizing $A$ as $P^{-1} \Delta P$ and computing $P^{-1} \Delta^n P$ in $\bar{K}$
Fast matrix powering

Goal: Given $A \in K^{s \times s}$ and $N \in \mathbb{N}$, compute $A^N$

Binary powering in $K^{s \times s}$ costs $O(s^\theta \log N)$ ops

**Algorithm.** Input: $A \in K^{s \times s}$, $N \in \mathbb{N}$  
Output: $A^N$

1. Compute the characteristic polynomial $\chi_A \in K[t]$
2. Compute $R = t^N \mod \chi$ by binary powering modulo $\chi$
3. Return $R(A)$

For distinct roots, $\approx$ diagonalizing $A$ as $P^{-1} \Delta P$ and computing $P^{-1} \Delta^n P$ in $\overline{K}$
Fast matrix powering

Goal: Given $A \in \mathbb{K}^{s \times s}$ and $N \in \mathbb{N}$, compute $A^N$

Binary powering in $\mathbb{K}^{s \times s}$ costs $O(s^\theta \log N)$ ops

**Algorithm.** Input: $A \in \mathbb{K}^{s \times s}$, $N \in \mathbb{N}$ Output: $A^N$

1. Compute the characteristic polynomial $\chi_A \in \mathbb{K}[t]$ \hspace{10cm} $O(s^\theta \log s)$
2. Compute $R = t^N \text{rem} \chi$ by binary powering modulo $\chi$
3. Return $R(A)$

For distinct roots, $\approx$ diagonalizing $A$ as $P^{-1} \Delta P$ and computing $P^{-1} \Delta^n P$ in $\bar{\mathbb{K}}$
Fast matrix powering

Goal: Given $A \in K^{s \times s}$ and $N \in \mathbb{N}$, compute $A^N$

Binary powering in $K^{s \times s}$ costs $O(s^\theta \log N)$ ops

**Algorithm.** *Input:* $A \in K^{s \times s}$, $N \in \mathbb{N}$  
*Output:* $A^N$

1. Compute the characteristic polynomial $\chi_A \in K[t]$  
   $\quad$ $O(s^\theta \log s)$
2. Compute $R = t^N \text{rem} \chi$ by binary powering modulo $\chi$  
   $\quad$ $O(M(s) \log N)$
3. Return $R(A)$

For distinct roots, $\approx$ diagonalizing $A$ as $P^{-1} \Delta P$ and computing $P^{-1} \Delta^n P$ in $\overline{K}$
Fast matrix powering

Goal: Given $A \in \mathbb{K}^{s \times s}$ and $N \in \mathbb{N}$, compute $A^N$

Binary powering in $\mathbb{K}^{s \times s}$ costs $O(s^\theta \log N)$ ops

**Algorithm.** Input: $A \in \mathbb{K}^{s \times s}$, $N \in \mathbb{N}$  
Output: $A^N$

1. Compute the characteristic polynomial $\chi_A \in \mathbb{K}[t]$  
   $O(s^\theta \log s)$
2. Compute $R = t^N \text{rem} \chi$ by binary powering modulo $\chi$  
   $O(M(s) \log N)$
3. Return $R(A)$  
   $O(s^\theta+1)$

Total $O(s^\theta+1 + M(s) \log N)$  
($s^\theta+1 \sim s^{\theta+1/2}$ later)

For distinct roots, $\approx$ diagonalizing $A$ as $P^{-1} \Delta P$ and computing $P^{-1} \Delta^n P$ in $\bar{\mathbb{K}}$
A Simple and Fast Algorithm for Computing the $N$-th Term of a Linearly Recurrent Sequence

Alin Bostan$^*$ and Ryuhei Mori$^†$
$^*$Inria, Palaiseau, France and $^†$Tokyo Institute of Technology, Japan
alin.bostan@inria.fr, mori@c.titech.ac.jp

Abstract
We present a simple and fast algorithm for computing the $N$-th term of a given linearly recurrent sequence. Our new algorithm uses $O(M(d) \log N)$ arithmetic operations, where $d$ is the order of the recurrence, and $M(d)$ denotes the number of arithmetic operations for computing the product of two polynomials of degree $d$. The state-of-the-art algorithm, due to Fiduccia (1985), has the same arithmetic complexity up to a constant factor. Our algorithm is simpler, faster and obtained consisting in a recurrence relation and sufficiently many initial terms that uniquely determine its terms.

Efficiency is measured in terms of ring operations (algebraic model), or of bit operations (Turing machine model). The cost of an algorithm is respectively estimated in terms of arithmetic complexity or of binary complexity. Both measures have their own usefulness: the algebraic model is relevant when ring operations have essentially unit cost (typically, if $R$ is a finite ring such as the prime field $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$), while the bit com-
The Gräffe transform

**Definition.** The Gräffe transform of a polynomial $P \in \mathbb{K}[t]$ is the polynomial $G[P]$ s.t.

$$G[P](t^2) = P(t)P(-t)$$

In other words: $G[P]$ is “the” polynomial whose roots are the squares of those of $P$.
The Gräfe transform

**Definition.** The Gräfe transform of a polynomial \( P \in \mathbb{K}[t] \) is the polynomial \( G[P] \) s.t.

\[
G[P](t^2) = P(t)P(-t)
\]

In other words: \( G[P] \) is “the” polynomial whose roots are the squares of those of \( P \).
The Gräffe transform

**Definition.** The Gräffe transform of a polynomial $P \in \mathbb{K}[t]$ is the polynomial $G[P]$ s.t.

$$G[P](t^2) = P(t) P(-t)$$

In other words: $G[P]$ is “the” polynomial whose roots are the squares of those of $P$
The Gräffe transform

**Definition.** The Gräffe transform of a polynomial $P \in \mathbb{K}[t]$ is the polynomial $G[P]$ s.t.

$$G[P](t^2) = P(t) P(-t)$$

In other words: $G[P]$ is “the” polynomial whose roots are the squares of those of $P$
The Gräffe transform

Definition. The Gräffe transform of a polynomial $P \in \mathbb{K}[t]$ is the polynomial $G[P]$ s.t.

$$G[P](t^2) = P(t) P(-t)$$

In other words: $G[P]$ is “the” polynomial whose roots are the squares of those of $P$.
Definition. The Gräffe transform of a polynomial $P \in \mathbb{K}[t]$ is the polynomial $G[P]$ s.t.

$$G[P](t^2) = P(t) P(-t)$$

In other words: $G[P]$ is “the” polynomial whose roots are the squares of those of $P$. 
Nth term by iterated Gräffe transforms (idea)

\[ \chi(S) \cdot (u_n)_{n \in \mathbb{N}} = 0 \]
Nth term by iterated Gräffe transforms (idea)

\[
\begin{align*}
\chi(S) \cdot (u_n)_{n \in \mathbb{N}} &= 0 \\
\chi(-S) \chi(S) \cdot (u_n)_{n \in \mathbb{N}} &= 0 \\
G[\chi](S^2)
\end{align*}
\]

[Bostan–Mori 2021]
Nth term by iterated Gräffe transforms (idea)

\[ \chi(S) \cdot (u_n)_{n \in \mathbb{N}} = 0 \]
\[ \chi(-S) \chi(S) \cdot (u_n)_{n \in \mathbb{N}} = 0 \]
\[ G[\chi](S^2) \]

\[ G[\chi](S) \cdot (u_{2n})_{n \in \mathbb{N}} = 0 \]

[Bostan–Mori 2021]
Nth term by iterated Gräffe transforms (idea)

\[
\begin{align*}
\chi(S) \cdot (u_n)_{n \in \mathbb{N}} &= 0 \\
\chi(-S) \chi(S) \cdot (u_n)_{n \in \mathbb{N}} &= 0 \\
G[\chi](S^2) &= 0
\end{align*}
\]

Algorithm. *Input*: \( \chi, u_0:s, N \)  
*Output*: \( u_N \)

- If \( N < s \), return \( u_s \)
- Compute \( \chi' = G[\chi] \)
- Compute initial values \( (u'_0, \ldots, u'_{s-1}) \) for \( (u'_n) = \begin{cases} 
(u_{2n}) & \text{if } N \text{ is even} \\
(u_{2n+1}) & \text{if } N \text{ is odd}
\end{cases} \)
- Repeat with \( \chi \leftarrow \chi', u_0:s \leftarrow u'_0:s, N \leftarrow \lfloor N/2 \rfloor \) \( \leq \log_2 N \) times
Nth term by iterated Gräffe transforms (idea)

$$\chi(S) \cdot (u_n)_{n \in \mathbb{N}} = 0$$
$$\chi(-S) \cdot (u_n)_{n \in \mathbb{N}} = 0$$
$$G[\chi](S^2)$$

Algorithm. Input: $\chi, u_0:s, N$ \hspace{1cm} Output: $u_N$

- If $N < s$, return $u_s$
- Compute $\chi' = G[\chi]$ \hspace{1cm} $O(M(s))$
- Compute initial values $(u'_0, \ldots, u'_{s-1})$ for $(u'_n) = \begin{cases} (u_{2n}) & \text{if } N \text{ is even} \\ (u_{2n+1}) & \text{if } N \text{ is odd} \end{cases}$ \hspace{1cm} $O(?)$
- Repeat with $\chi \leftarrow \chi', u_0:s \leftarrow u'_0:s, N \leftarrow \lceil N/2 \rceil$ \hspace{1cm} $\leq \log_2 N$ times

To do: Efficiently compute $(u_s, \ldots, u_{2s-1})$ from $(u_0, \ldots, u_{s-1})$.

(not exactly what the fast algorithm does)
1.2 First N terms
C-finite sequences and rational series

**Proposition.** The sequence \((u_n)_{n \in \mathbb{N}}\) satisfies

\[
\forall n \in \mathbb{N}, \quad u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0
\]

if and only if its generating series is of the form

\[
\sum_{n=0}^{\infty} u_n x^n = p(x) \frac{1}{1 + c_{s-1} x + \cdots + c_0 x^s} = \frac{p(x)}{\text{rev}_s(x)}
\]

for some \(p \in \mathbb{K}[x]_{<s}\).

denominator ↔ recurrence, numerator ↔ initial values / residual
C-finite sequences and rational series

**Proposition.** The sequence \((u_n)_{n \in \mathbb{N}}\) satisfies

\[\forall n \in \mathbb{N}, \quad u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0\]

if and only if its generating series is of the form

\[\sum_{n=0}^{\infty} u_n x^n = \frac{p(x)}{1 + c_{s-1} x + \cdots + c_0 x^s} = \frac{p(x)}{\operatorname{rev}_s(x)} \quad \text{for some } p \in \mathbb{K}[x]_{<s}.\]

**Proof.** Extend \((u_n)\) to \(n \in \mathbb{Z}\) by setting \(u_n = 0\) for \(n < 0\). Then

\[
(1 + c_{s-1} x + \cdots + c_0 x^s) \sum_{n \in \mathbb{Z}} u_n x^n = \sum_{n \in \mathbb{Z}} (u_n + c_{s-1} u_{n-1} + \cdots + c_0 u_{n-s}) x^n
\]

\[= \sum_{n \in \mathbb{Z}} (u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n) x^{n+s}\]

is a polynomial of degree \(< s\) iff the recurrence holds for all \(n \in \mathbb{N}\). \(\square\)
**Proposition.** The sequence \((u_n)_{n \in \mathbb{N}}\) satisfies

\[
\forall n \in \mathbb{N}, \quad u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0
\]

if and only its generating series is of the form

\[
\sum_{n=0}^{\infty} u_n x^n = \frac{p(x)}{1 + c_{s-1} x + \cdots + c_0 x^s} = \frac{p(x)}{\text{rev}_s(x)} \quad \text{for some } p \in \mathbb{K}[x]_{<s}.
\]

**Corollary.** Given \(p, q \in \mathbb{K}[x]_{<d}\) with \(q(0) \neq 0\), one can compute the \(N\)th term of the series expansion of \(p/q\) in \(O(M(d) \log N)\) ops.
From $s$ to $2s$ terms

\[ u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0 \]

**Problem.** Given \((u_0, \ldots, u_{s-1})\), compute \((u_s, \ldots, u_{2s-1})\).

Using the previous proposition, write \( \sum_{n \geq 0} u_n x^n = \frac{p(x)}{q(x)} \) with \( q = \text{rev}_s(x) \) and \( \deg p < s \).

\[ \frac{p(x)}{q(x)} = u_0 + \cdots + u_{s-1} x^{s-1} + O(x^s) \]

\[ U_0(x) \]
From s to 2s terms

[Fiduccia 1985, Shoup 1991]

\[ u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0 \]

**Problem.** Given \((u_0, \ldots, u_{s-1})\), compute \((u_s, \ldots, u_{2s-1})\).

Using the previous proposition, write

\[
\sum_{n \geq 0} u_n x^n = \frac{p(x)}{q(x)} \quad \text{with} \quad q = \text{rev}_s(x) \quad \text{and} \quad \deg p < s.
\]

\[
\frac{p(x)}{q(x)} = u_0 + \cdots + u_{s-1} x^{s-1} + O(x^s) \quad \Rightarrow \quad p(x) = q(x) U_0(x) \, \text{rem} \, x^s
\]
From $s$ to $2s$ terms

[Fiduccia 1985, Shoup 1991]

$$u_{n+s} + c_{s-1} u_{n+s-1} + \cdots + c_0 u_n = 0$$

**Problem.** Given $(u_0, \ldots, u_{s-1})$, compute $(u_s, \ldots, u_{2s-1})$.

Using the previous proposition, write

$$\sum_{n \geq 0} u_n x^n = \frac{p(x)}{q(x)}$$

with $q = \operatorname{rev}_s(\chi)$ and $\deg p < s$.

$$\frac{p(x)}{q(x)} = u_0 + \cdots + u_{s-1} x^{s-1} + O(x^s) \quad \Rightarrow \quad p(x) = q(x) U_0(x) \bmod x^s$$

**Algorithm.** *Input:* $u_0:s$, $c_0:s$  
*Output:* $u_0:N$

1. Compute $p = q U_0 \bmod x^s$  \(O(M(s))\)
2. Compute the first $N$ terms of $p/q$ by a power series division  \(O(M(N))\)
From s to 2s terms

Problem. Given \((u_0, \ldots, u_{s-1})\), compute \((u_s, \ldots, u_{2s-1})\).

Using the previous proposition, write 
\[
\sum_{n \geq 0} u_n x^n = \frac{p(x)}{q(x)} \quad \text{with} \quad q = \text{rev}_s(x) \quad \text{and} \quad \deg p < s.
\]

\[
\frac{p(x)}{q(x)} = u_0 + \cdots + u_{s-1} x^{s-1} + O(x^s) \quad \Rightarrow \quad p(x) = q(x) U_0(x) \text{ rem } x^s
\]

Algorithm. Input: \(u_0:s, c_0:s\) \quad Output: \(u_0:2s\)

1. Compute \(p = q U_0 \text{ rem } x^s\) \quad \(O(M(s))\)
2. Compute the first 2s terms of \(p/q\) by a power series division \quad \(O(M(2s))\)
Summary

\[ \{ L(\mathbf{u}) = 0, u_0 : s \} \xrightarrow{\text{div}} \frac{p(x)}{q(x)} \xrightarrow{\text{div}} \text{Padé} \quad \mathbf{u}(x) + O(x^{2s}) \]
First N terms

Idea: Iterate the extension $s \leadsto 2s$ terms $N/s$ times

$$(u_{ks+n})_{n \in \mathbb{N}} \text{ satisfies the same recurrence as } (u_n)$$

\[
\frac{p}{q} = u_0 + x^s u_1 + \cdots + x^{ks} u_k + x^{(k+1)s} \frac{p_{k+1}}{q}
\]

Algorithm. Input: $u_0:s$, $c_0:s$, $N$   Output: $u_0:N$

1. Let $U_0 = u_0 + \cdots + u_{s-1} x^{s-1}$

2. For $k = 0, \ldots, \lfloor N/s \rfloor$:
   a. Compute $P_k = q U_k \text{ rem } x^s$  
      [numerator of gen. series of $(u_{ks+n})$]
   b. Compute $U_{k+1} = u_{(k+1)s+1} + \cdots + u_{(k+2)s} x^s$ via $q^{-1} P_k + O(x^{2s}) = U_k + x^s U_{k+1}$

3. Return $(u_0, \ldots, u_{N-1})$

Total cost $O(NM(s)/s)$
First $N$ terms

Idea: Iterate the extension $s \rightarrow 2s$ terms $N/s$ times

$$(u_{ks+n})_{n \in \mathbb{N}}$$ satisfies the same recurrence as $(u_n)$

$$\frac{p}{q} = u_0 + x^s u_1 + \cdots + x^{ks} u_k + x^{(k+1)s} \frac{p_{k+1}}{q}$$

**Algorithm.** Input: $u_0:s, c_0:s, N$  Output: $u_0:N$

1. Let $U_0 = u_0 + \cdots + u_{s-1} x^{s-1}$
2. For $k = 0, \ldots, [N/s]$
   a. Compute $P_k = q U_k \text{rem } x^s$ [numerator of gen. series of $(u_{ks+n})$]
   b. Compute $U_{k+1} = u_{(k+1)s+1} + \cdots + u_{(k+2)s} x^s$ via $q^{-1} P_k + O(x^{2s}) = U_k + x^s U_{k+1}$
3. Return $(u_0, \ldots, u_{N-1})$

Total cost $O(N M(s)/s)$
Redundant: compute only once.

\[ \frac{P_k}{q} = U_k + x^s \frac{P_{k+1}}{q} \Rightarrow P_{k+1} = q U_k \text{ div } x^s, \quad U_{k+1} = q^{-1} P_{k+1} \text{ rem } x^s \]

- use short products, reuse DFTs
First N terms

Idea: Iterate the extension \( s \mapsto 2s \) terms \( N/s \) times

\[(u_{ks+n})_{n \in \mathbb{N}} \text{ satisfies the same recurrence as } (u_n)\]

\[
\frac{p}{q} = u_0 + x^s u_1 + \cdots + x^{ks} u_k + x^{(k+1)s} \frac{p_{k+1}}{q}
\]

Algorithm. Input: \( u_0; s, c_0; s, N \)  
Output: \( u_0; N \)

1. Let \( U_0 = u_0 + \cdots + u_{s-1} x^{s-1} \)
2. For \( k = 0, \ldots, \lfloor N/s \rfloor \):
   a. Compute \( P_k = q U_k \text{ rem } x^s \) \[\text{[numerator of gen. series of } (u_{ks+n})]\]
   b. Compute \( U_{k+1} = u_{(k+1)s+1} + \cdots + u_{(k+2)s} x^s \) via \( q^{-1} P_k + O(x^{2s}) = U_k + x^s U_{k+1} \)
3. Return \((u_0, \ldots, u_{N-1})\)

Total cost \( O(NM(s)/s) \)

Exercise. For \( F, G \in \mathbb{K}[x] \) with \( \deg F < n \) and \( \deg G < m \), give an algorithm that computes the division with remainder \( F = QG + R \) using \( O(nM(m)/m) \) operations in \( \mathbb{K} \).
2 P-recursive sequences
2.1 Introduction
Direct computation of $N!$

**Algorithm.** Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \ldots, N$. 
Direct computation of $N!$

**Algorithm.** Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \ldots, N$.

- Arithmetic complexity: $O(N)$ ops
- Optimal if computing $1!, \ldots, N!$
Direct computation of $N!$

**Algorithm.** Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \ldots, N$.

- Arithmetic complexity: $O(N)$ ops
  - Optimal if computing $1!, \ldots, N!$

- Bit complexity:
  \[
  \text{size}(n!) = 1 + \left\lfloor \log_2(n!) \right\rfloor = n \log_2 n + O(n)
  \]
Direct computation of $N!$

**Algorithm.** Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \ldots, N$.

- Arithmetic complexity: $O(N)$ ops
- Bit complexity:
  \[
  \text{size}(n!) = 1 + \lceil \log_2(n!) \rceil = n \log_2 n + O(n)
  \]

  Step $n$ is a multiplication of
  \[
  n \log_2 n + O(n) \quad \text{by} \quad \log_2 n + O(1) \quad \text{bits}
  \]
  costing $n M_Z(\log_2 n) + O(n)$ bit operations if done by blocks.

Optimal if computing $1!, \ldots, N!$
Direct computation of $N!$

**Algorithm.** Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \ldots, N$.

- Arithmetic complexity: $O(N)$ ops

  Optimal if computing $1!, \ldots, N!$

- Bit complexity:

  $$\text{size}(n!) = 1 + \lfloor \log_2(n!) \rfloor = n \log_2 n + O(n)$$

Step $n$ is a multiplication of

$$n \log_2 n + O(n) \text{ by } \log_2 n + O(1) \text{ bits}$$

costing $n M_Z(\log_2 n) + O(n)$ bit operations if done by blocks.

Total cost:

$$\sum_{n=1}^{N} \left( n M_Z(\log_2 n) + O(n) \right)$$
Direct computation of $N!$

**Algorithm.** Repeat $u_n = n \cdot u_{n-1}$ for $n = 1, 2, \ldots, N$.

- Arithmetic complexity: $O(N)$ ops Optimal if computing $1!, \ldots, N!$

- Bit complexity:

  $$\text{size}(n!) = 1 + \lfloor \log_2(n!) \rfloor = n \log_2 n + O(n)$$

Steps $n$ is a multiplication of

$$n \log_2 n + O(n) \quad \text{by} \quad \log_2 n + O(1) \quad \text{bits}$$

costing $n M_Z(\log_2 n) + O(n)$ bit operations if done by blocks.

Total cost:

$$\sum_{n=1}^{N} (n M_Z(\log_2 n) + O(n)) = \frac{N^2}{2} M_Z(\log_2 N) + O(N^2) \text{ bit ops}$$

Quasi-optimal for $N$ terms, unsatisfactory for a single term.
Nonsingular recurrences

\[ b_s(n)u_{n+s} + \cdots + b_1(n)u_{n+1} + b_0(n)u_n = 0 \]

**Definition.** We will say that the recurrence operator

\[ L = b_s(n)S^s + \cdots + b_1 S + b_0 \in \mathbb{K}[n] \langle S \rangle \]

is **nonsingular** if \( b_s(n) \neq 0 \) for all \( n \in \mathbb{N} \).

**Proposition.** If \( L \in \mathbb{K}[n] \langle S \rangle \) as above is nonsingular, then the solution space of

\[ L \cdot (u_n)_{n \in \mathbb{N}} = 0 \]

has dimension \( s \) and any solution \((u_n)_{n \in \mathbb{N}}\) is determined by \((u_0, \ldots, u_{s-1})\).

In other words: basis

- \( u^{(0)} = (1, 0, 0, \ldots, 0, *, *, *, \ldots) \)
- \( u^{(1)} = (0, 1, 0, \ldots, 0, *, *, *, \ldots) \)
- \( \vdots \)
- \( u^{(s-1)} = (0, 0, 0, \ldots, 1, *, *, *, \ldots) \)
First N terms, Nth term

\[ b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0 \]

**Problems.** Given a nonsingular recurrence as above, initial values \( u_0: s \), and \( N \in \mathbb{N} \):

a) Compute \((u_0, \ldots, u_{N-1})\)

b) Compute \( u_N \)

Complexity models: operations in \( K \) (“ops”)

- binary operations for \( K = \mathbb{Z} \)
First $N$ terms, $N$th term

$$b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0$$

**Problems.** Given a nonsingular recurrence as above, initial values $u_0:s$, and $N \in \mathbb{N}$:

a) Compute $(u_0, \ldots, u_{N-1})$

b) Compute $u_N$

Complexity models: operations in $\mathbb{K}$ ("ops")

- binary operations for $\mathbb{K} = \mathbb{Z}$

Bit sizes:

- for a single $u_n$,
- for $u_0:N$ (reached)
First $N$ terms, $N$th term

$$b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0$$

**Problems.** Given a nonsingular recurrence as above, initial values $u_{0:s}$, and $N \in \mathbb{N}$:

a) Compute $(u_0, \ldots, u_{N-1})$

b) Compute $u_N$

Complexity models: operations in $\mathbb{K}$ ("ops")

- Binary operations for $\mathbb{K} = \mathbb{Z}$

Bit sizes: $O(n \log n)$ for a single $u_n$, for $u_{0:N}$ (reached)
First N terms, Nth term

\[ b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0 \]

**Problems.** Given a nonsingular recurrence as above, initial values \( u_0:s \), and \( N \in \mathbb{N} \):

a) Compute \( (u_0, \ldots, u_{N-1}) \)

b) Compute \( u_N \)

Complexity models: operations in \( \mathbb{K} \) (“ops”)

binary operations for \( \mathbb{K} = \mathbb{Z} \)

Bit sizes: \( O(n \log n) \) for a single \( u_n \), \( O(N^2 \log N) \) for \( u_0:N \) (reached)
First N terms, Nth term

\[ b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0 \]

**Problems.** Given a nonsingular recurrence as above, initial values \( u_0: s \), and \( N \in \mathbb{N} 
\)

a) Compute \( (u_0, \ldots, u_{N-1}) \)

b) Compute \( u_N \)

Complexity models: operations in \( \mathbb{K} \) ("ops")

\[ \text{binary operations for } \mathbb{K} = \mathbb{Z} \]

Bit sizes: \( O(n \log n) \) for a single \( u_n \), \( O(N^2 \log N) \) for \( u_0: N \) (reached)

Direct algorithm: repeat

\[ u_n = -\frac{1}{b_s(n-s)}(b_{s-1}(n-s)u_{n-1} + \cdots + b_0(n-s)u_{n-s}) \]

Arithmetic cost:

Over the integers:
First $N$ terms, $N$th term

\[ b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0 \]

**Problems.** Given a **nonsingular** recurrence as above, initial values $u_0:s$, and $N \in \mathbb{N}$:

a) Compute $(u_0, \ldots, u_{N-1})$

b) Compute $u_N$

Complexity models: operations in $\mathbb{K}$ ("ops")

- binary operations for $\mathbb{K} = \mathbb{Z}$

Bit sizes: $O(n \log n)$ for a single $u_n$, $O(N^2 \log N)$ for $u_0:N$ (reached)

Direct algorithm: repeat $u_n = -\frac{1}{b_s(n-s)} (b_{s-1}(n-s) u_{n-1} + \cdots + b_0(n-s) u_{n-s})$

Arithmetic cost: $O(N)$ ops

Over the integers:
First $N$ terms, $N$th term

\[ b_s(n) u_{n+s} + \cdots + b_1(n) u_{n+1} + b_0(n) u_n = 0 \]

**Problems.** Given a nonsingular recurrence as above, initial values $u_0:s$, and $N \in \mathbb{N}$:

a) Compute $(u_0, \ldots, u_{N-1})$

b) Compute $u_N$

Complexity models: operations in $\mathbb{K}$ (“ops”)

- Binary operations for $\mathbb{K} = \mathbb{Z}$

Bit sizes: $O(n \log n)$ for a single $u_n$, $O(N^2 \log N)$ for $u_0:N$ (reached)

Direct algorithm: repeat

\[ u_n = -\frac{1}{b_s(n-s)} \left( b_{s-1}(n-s) u_{n-1} + \cdots + b_0(n-s) u_{n-s} \right) \]

- Arithmetic cost: $O(N)$ ops
- Over the integers: $O(N^2 M_{\mathbb{Z}}(\log N))$ binops

Optimal for fixed rec. for problem a $\longrightarrow$ Focus on problem b
2.2 Baby steps, giant steps
A baby steps-giant steps algorithm for $N!$

$$N! = 1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2 \ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2$$

$N^{1/2}$ blocks of size $N^{1/2}$

$\ell = N^{1/2}$

[Strassen 1976]
A baby steps-giant steps algorithm for $N!$

$$N! = 1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2\ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2$$

$\ell = N^{1/2}$

$N^{1/2}$ blocks of size $N^{1/2}$

**Algorithm.** *Input: N  Output: N!*

1. Let $\ell = \lfloor N^{1/2} \rfloor$

2. Baby steps:
   
   a. Build a product tree for $F = (x + 1) (x + 2) \cdots (x + \ell)$

3. Giant steps:
   
   a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2\ell), \ldots, P_{\ell-1} = F((\ell - 1) \ell)$

   by multipoint evaluation

   b. Return $P_0 P_1 \cdots P_{\ell-1} \cdot (\ell^2 + 1) \cdots (N - 1) N$
A baby steps-giant steps algorithm for $N!$

$$N! = 1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2 \ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2 \quad \ell = N^{1/2}$$

$N^{1/2}$ blocks of size $N^{1/2}$

**Algorithm.** Input: $N$    Output: $N!$

1. Let $\ell = \lfloor N^{1/2} \rfloor$

2. Baby steps:
   a. Build a product tree for $F = (x + 1) (x + 2) \cdots (x + \ell)$    $O(M(\ell) \log \ell)$

3. Giant steps:
   a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2 \ell), \ldots, P_{\ell-1} = F((\ell - 1) \ell)$
      by multipoint evaluation
   b. Return $P_0 P_1 \cdots P_{\ell-1} \cdot (\ell^2 + 1) \cdots (N - 1) N$
A baby steps-giant steps algorithm for $N!$

$$N! = 1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2 \ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2$$

$\ell = N^{1/2}$

$N^{1/2}$ blocks of size $N^{1/2}$

**Algorithm. Input: $N$  Output: $N!$**

1. Let $\ell = \lfloor N^{1/2} \rfloor$

2. Baby steps:
   a. Build a product tree for $F = (x + 1) (x + 2) \cdots (x + \ell)$ $O(M(\ell) \log \ell)$

3. Giant steps:
   a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2 \ell), \ldots, P_{\ell - 1} = F((\ell - 1) \ell)$ by multipoint evaluation $O(M(\ell) \log \ell)$
   b. Return $P_0 P_1 \cdots P_{\ell - 1} \cdot (\ell^2 + 1) \cdots (N - 1) N$
A baby steps-giant steps algorithm for $N!$

$$N! = 1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2 \ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2$$

$\ell = N^{1/2}$

$N^{1/2}$ blocks of size $N^{1/2}$

**Algorithm.** *Input: N  Output: N!*

1. Let $\ell = \lfloor N^{1/2} \rfloor$

2. Baby steps:
   a. Build a product tree for $F = (x + 1) (x + 2) \cdots (x + \ell)$
      
      $O(M(\ell) \log \ell)$

3. Giant steps:
   a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2 \ell), \ldots, P_{\ell-1} = F((\ell - 1) \ell)$
      by multipoint evaluation
      
      $O(M(\ell) \log \ell)$

   b. Return $P_0 P_1 \cdots P_{\ell-1} \cdot (\ell^2 + 1) \cdots (N - 1) N$
      
      $O(\ell)$
A baby steps-giant steps algorithm for $N!$

$$N! = 1 \cdot 2 \cdots \ell \cdot (\ell + 1) (\ell + 2) \cdots (2 \ell) \cdots (\ell^2 - \ell + 1) (\ell^2 - \ell + 2) \cdots \ell^2$$

$\ell = N^{1/2}$

$N^{1/2}$ blocks of size $N^{1/2}$

**Algorithm.** *Input: N  Output: N!*

1. Let $\ell = \lfloor N^{1/2} \rfloor$

2. Baby steps:
   a. Build a product tree for $F = (x + 1) (x + 2) \cdots (x + \ell)$  $O(M(\ell) \log \ell)$

3. Giant steps:
   a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2 \ell), \ldots, P_{\ell-1} = F((\ell - 1) \ell)$
      by multipoint evaluation  $O(M(\ell) \log \ell)$
   b. Return $P_0 P_1 \cdots P_{\ell-1} \cdot (\ell^2 + 1) \cdots (N - 1) N$  $O(\ell)$

Total $O(M(N^{1/2}) \log N)$ ops

[Strassen 1976]
Deterministic integer factorization

Idea: if $N$ is composite, $\lceil \sqrt{N} \rceil! \wedge N$ is a nontrivial factor
Deterministic integer factorization

Idea: if $N$ is composite, $\lceil\sqrt{N}\rceil! \land N$ is a nontrivial factor

Algorithm. Input: $N$  Output: a nontrivial factor of $N$, or 1 if $N$ is prime

1. Let $\ell = \lceil N^{1/4} \rceil$

2. Baby steps:
   a. Build a product tree for $F = (x + 1)(x + 2) \cdots (x + \ell)$

3. Giant steps:
   a. Compute $P_0 = F(0), \ldots, P_{\ell-1} = F((\ell - 1) \ell)$ by mulpt ev.
   b. Compute $P_0 \land N, \ldots, P_{\ell-1} \land N$
Deterministic integer factorization

Idea: if \( N \) is composite, \( \lceil \sqrt{N} \rceil != N \) is a nontrivial factor

**Algorithm.** *Input:* \( N \)  \hspace{1cm} *Output:* a nontrivial factor of \( N \), or 1 if \( N \) is prime

1. Let \( \ell = \lceil N^{1/4} \rceil \)
2. Baby steps:
   a. Build a product tree for \( F = (x + 1) (x + 2) \cdots (x + \ell) \) \hspace{1cm} \( \mathcal{O}(M(\ell) \log(\ell) M_Z(h)) \)
3. Giant steps:
   a. Compute \( P_0 = F(0), \ldots, P_{\ell-1} = F((\ell - 1) \ell) \) by mulpt ev. \hspace{1cm} \( \mathcal{O}(M(\ell) \log(\ell) M_Z(h)) \)
   b. Compute \( P_0 \land N, \ldots, P_{\ell-1} \land N \) \hspace{1cm} \( \mathcal{O}(\ell M(h) \log(h)) \)

\( \text{size}(P_k) \leq h = \mathcal{O}(\log N) \)
Deterministic integer factorization

Idea: if $N$ is composite, $\lfloor \sqrt{N} \rfloor! \wedge N$ is a nontrivial factor

**Algorithm.** *Input:* $N$  
*Output:* a nontrivial factor of $N$, or 1 if $N$ is prime

1. Let $\ell = \lfloor N^{1/4} \rfloor$

2. Baby steps:
   a. Build a product tree for $F = (x + 1)(x + 2) \cdots (x + \ell)$  
      $O(M(\ell) \log(\ell) M_\mathbb{Z}(h))$

3. Giant steps:
   a. Compute $P_0 = F(0), \ldots, P_{\ell-1} = F((\ell - 1) \ell)$ by mulpt ev.  
      $O(M(\ell) \log(\ell) M_\mathbb{Z}(h))$
   b. Compute $P_0 \wedge N, \ldots, P_{\ell-1} \wedge N$  
      $O(\ell M(h) \log(h))$

Size $P_k \leq h = O(\log N)$  

Total $O(M(N^{1/4}) \log(N)^{O(1)})$
A TIME-SPACE TRADEOFF FOR
LEHMAN’S DETERMINISTIC INTEGER FACTORIZATION METHOD

MARKUS HITTMEIR

ABSTRACT. Fermat’s well-known factorization algorithm is based on finding a representation of natural numbers $N$ as the difference of squares. In 1895, Lawrence generalized this idea and applied it to multiples $kN$ of the original number. A systematic approach to choose suitable values for $k$ has been introduced by Lehman in 1974, which resulted in the first deterministic factorization algorithm considerably faster than trial division. In this paper, we construct a time-space tradeoff for Lawrence’s generalization and apply it together with Lehman’s result to obtain a deterministic integer factorization algorithm with runtime complexity $O\left(N^{2/9+o(1)}\right)$. This is the first exponential improvement since the establishment of the $O(N^{1/4+o(1)})$ bound in 1977.

1. Introduction

We consider the problem of computing the prime factorization of natural numbers $N$. There is a large
AN EXPONENT ONE-FIFTH ALGORITHM FOR DETERMINISTIC INTEGER FACTORISATION

DAVID HARVEY

Abstract. Hittmeir recently presented a deterministic algorithm that provably computes the prime factorisation of a positive integer $N$ in $N^{2/9+o(1)}$ bit operations. Prior to this breakthrough, the best known complexity bound for this problem was $N^{1/4+o(1)}$, a result going back to the 1970s. In this paper we push Hittmeir’s techniques further, obtaining a rigorous, deterministic factoring algorithm with complexity $N^{1/5+o(1)}$. 

1. Introduction

Let $F(N)$ denote the time required to compute the prime factorisation of an integer $N \geq 2$. By “time” we mean “number of bit operations”, or more precisely, the number of steps performed by a deterministic Turing machine with a fixed, finite number of linear tapes [Pap94]. All integers are assumed to be encoded in the usual binary representation.

In this paper we prove the following result:

Theorem 1.1. There is an integer factorisation algorithm achieving $F(N) = O(N^{1/5 \log 16/5})$.

One may show that the space complexity of this algorithm is $O(N^{1/5 \log 11/5})$, but we will not give the details of this analysis.

We emphasise that the new algorithm is deterministic, the complexity bound is rigorously established, and the algorithm runs on a classical computer. If one is willing to relax these conditions, better complexity bounds are known:

– The class group relations method of Lenstra and Pomerance [LP92] provably runs in expected time $\exp((1+o(1))((\log N)^{1/2}(\log \log N)^{1/2}))$. This is the best known rigorous bound for a probabilistic factoring algorithm.

– The general number field sieve is conjectured to run in time $\exp((64/9)^{1/3}+o(1)((\log N)^{1/3}(\log \log N)^{2/3}))$. See [CP05, §6.2] for a discussion of the heuristics supporting this conjecture. This algorithm is responsible for all recent record-breaking factorisations, such as the factorisation of the 250-digit challenge number RSA-250 [BGG+20].

– Shor’s factoring algorithm runs in polynomial time, specifically $(\log N)^2+o(1)$, on a quantum computer [Sho94].
Generalization to P-recursive sequences

[Chudnovsky & Chudnovsky 1987]

Write the recurrence in matrix form, pull out the denominator:

\[
\begin{pmatrix}
  u_{n+1} \\
  \vdots \\
  u_{n+s-1} \\
  u_{n+s}
\end{pmatrix}
= \frac{1}{b_s(n)} \begin{pmatrix}
  b_s(n) & \cdots & b_s(n) \\
  -b_0(n) & -b_1(n) & \cdots & -b_{s-1}(n)
\end{pmatrix} \begin{pmatrix}
  u_n \\
  \vdots \\
  u_{n+s-2} \\
  u_{n+s-1}
\end{pmatrix}
\]

Then

\[
U_N = \frac{1}{b_s(N-1) \cdots b_s(1) b_s(0)} B(N-1) \cdots B(1) B(0) U_0
\]

\[B(n) = \text{matrix of polynomials of degree } < d\]
Algorithm. Input: \( B \in \mathbb{K}[n]^{s \times s} \) of \( \deg < d \), \( N \in \mathbb{N} \) Output: \( B(N - 1) \cdots B(1) B(0) \)

1. Write \( N = \ell m \) with \( \ell = \) and \( m = \) (assumed exact for simplicity)

2. Baby steps:
   a. Compute \( B(X + 1), \ldots, B(X + \ell - 1) \)
   b. Build a product tree for \( F(X) = B(X + \ell - 1) \cdots B(X + 1) B(X) \) viewed in \( \mathbb{K}^{s \times s}[X] \)

3. Giant steps:
   a. Compute \( F(0), F(\ell), \ldots, F((m - 1) \ell) \) simultaneously
   b. Deduce and return the product \( F((m - 1) \ell) \cdots F(\ell) F(0) \)
Fast polynomial matrix “factorial”

Algorithm. *Input:*

\[ B \in \mathbb{K}[n]^{s \times s} \text{ of deg } d < d, \quad N \in \mathbb{N} \]

*Output:*

\[ B(N - 1) \cdots B(1) B(0) \]

1. Write \( N = \ell \cdot m \) with \( \ell = (N / d)^{1/2} \) and \( m = (N \cdot d)^{1/2} \) (assumed exact for simplicity)

2. Baby steps:
   a. Compute \( B(X + 1), \ldots, B(X + \ell - 1) \)
   b. Build a product tree for \( F(X) = B(X + \ell - 1) \cdots B(X + 1) B(X) \) viewed in \( \mathbb{K}^{s \times s}[X] \)

3. Giant steps:
   a. Compute \( F(0), F(\ell), \ldots, F((m - 1) \cdot \ell) \) simultaneously
   b. Deduce and return the product \( F((m - 1) \cdot \ell) \cdots F(\ell) F(0) \)

 naïve step 2 takes \( O(\ell^2 s) \)

Total \( O(M(m) \log m s) \)
**Algorithm.** *Input:* $B \in \mathbb{K}[n]^{s \times s}$ of $\deg < d$, $N \in \mathbb{N}$  

*Output:* $B(N-1) \cdots B(1) B(0)$

1. Write $N = \ell m$ with $\ell = (N / d)^{1/2}$ and $m = (Nd)^{1/2}$ (assumed exact for simplicity)

2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Build a product tree for $F(X) = B(X+\ell-1) \cdots B(X+1) B(X)$ viewed in $\mathbb{K}^{s \times s}[X]$ 

3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\ell)$ simultaneously
   b. Deduce and return the product $F((m-1)\ell) \cdots F(\ell) F(0)$

$\deg P(X) < \ell d$
Fast polynomial matrix “factorial”

**Algorithm.** *Input:* $B \in \mathbb{K}[n]^{s \times s}$ of $\deg < d$, $N \in \mathbb{N}$   
*Output:* $B(N - 1) \cdots B(1) B(0)$

1. Write $N = \ell \ m$ with $\ell = (N / d)^{1/2}$ and $m = (N d)^{1/2}$   
   (assumed exact for simplicity)

2. Baby steps:
   a. Compute $B(X + 1), \ldots, B(X + \ell - 1)$
   b. Build a product tree for $F(X) = B(X + \ell - 1) \cdots B(X + 1) B(X)$ viewed in $\mathbb{K}^{s \times s}[X]$  
      $O(M(\ell \ d) \log(\ell) \ s^\theta)$

3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m - 1) \ell)$ simultaneously  
      $O(M(m) \log(m) \ s^\theta)$
   b. Deduce and return the product $F((m - 1) \ell) \cdots F(\ell) F(0)$

$\deg P(X) < \ell \ d$
Fast polynomial matrix “factorial”

**Algorithm.** *Input:* $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$  
*Output:* $B(N - 1) \cdots B(1) B(0)$

1. Write $N = \ell m$ with $\ell = (N / d)^{1/2}$ and $m = (N d)^{1/2}$ (assumed exact for simplicity)

2. Baby steps:
   
   a. Compute $B(X + 1), \ldots, B(X + \ell - 1)$
   
   b. Build a product tree for $F(X) = B(X + \ell - 1) \cdots B(X + 1) B(X)$ viewed in $\mathbb{K}^{s \times s}[X]$  
      
      $O(M(\ell d) \log(\ell) s^\theta)$

3. Giant steps:
   
   a. Compute $F(0), F(\ell), \ldots, F((m - 1) \ell)$ simultaneously  
      
      $O(M(m) \log(m) s^\theta)$
   
   b. Deduce and return the product $F((m - 1) \ell) \cdots F(\ell) F(0)$  
      
      $O(m s^\theta)$

$\deg P(X) < \ell d$
**Algorithm.** *Input:* $B \in \mathbb{K}[n]^{s \times s}$ of $\deg < d$, $N \in \mathbb{N}$  
*Output:* $B(N - 1) \cdots B(1) B(0)$

1. Write $N = \ell m$ with $\ell = (N/d)^{1/2}$ and $m = (N d)^{1/2}$ (assumed exact for simplicity)

2. Baby steps:
   a. Compute $B(X + 1), \ldots, B(X + \ell - 1)$ \hspace{1cm} $O(\ell M(d) \log(d) s)$
   b. Build a product tree for $F(X) = B(X + \ell - 1) \cdots B(X + 1) B(X)$ viewed in $\mathbb{K}^{s \times s}[X]$ \hspace{1cm} $O(M(\ell d) \log(\ell) s^\theta)$

3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m - 1) \ell)$ simultaneously \hspace{1cm} $O(M(m) \log(m) s^\theta)$
   b. Deduce and return the product $F((m - 1) \ell) \cdots F(\ell) F(0)$ \hspace{1cm} $O(m s^\theta)$

Naïvely step 2a takes $O(\ell d^2 s)$ ops

$\deg P(X) < \ell d$
**Fast polynomial matrix “factorial”**

**Algorithm.** *Input:* $B \in \mathbb{K}[n]^{s \times s}$ of $\deg < d$, $N \in \mathbb{N}$  
*Output:* $B(N-1) \cdots B(1) B(0)$

1. Write $N = \ell m$ with $\ell = (N/d)^{1/2}$ and $m = (N d)^{1/2}$  
   (assumed exact for simplicity)

2. Baby steps:
   - a. Compute $B(X + 1), \dots, B(X + \ell - 1)$  
     $O(\ell M(d) \log(d) s)$
   - b. Build a product tree for $F(X) = B(X + \ell - 1) \cdots B(X + 1) B(X)$  
     viewed in $\mathbb{K}^{s \times s}[X]$  
     $O(M(\ell d) \log(\ell) s^\theta)$

3. Giant steps:
   - a. Compute $F(0), F(\ell), \ldots, F((m - 1) \ell)$ simultaneously  
     $O(M(m) \log(m) s^\theta)$
   - b. Deduce and return the product $F((m - 1) \ell) \cdots F(\ell) F(0)$  
     $O(m s^\theta)$


 naïvely step 2a takes $O(\ell d^2 s)$ ops  
Total $O\left(M(m) \log(m) s^\theta\right)$

$\deg P(X) < \ell d$
Nth term of a P-recursive sequence

**Algorithm.** Notation as before.

1. Compute $B(N - 1) \cdots B(1) B(0)$ by the previous algorithm $O(M(m) \log(m) s^\theta)$
2. Compute $b_s(N - 1) \cdots b_s(1) b_s(0)$ by the previous algorithm $O(M(m) \log(m))$
3. Divide, return $O(s^2)$

**Theorem.** Let $(u^{(0)}, \ldots, u^{(s-1)})$ be the basis of solutions s.t. $u_i^{(j)} = \delta_{i,j}$ of a nonsingular recurrence of order $s$ and degree $<d$. One can compute the matrix $(u^{(j)}_{N+i})_{i,j} \in \mathbb{K}^{s \times s}$ in $O(M(\sqrt{N} d) \log(N d) s^\theta)$ ops.

**Corollary.** One can compute the $N$th term of a P-recursive sequence given by a nonsingular recurrence in $O(M(\sqrt{N}) \log N)$ ops.
Application to sums of D-finite series *(idea)*

Let $\Sigma_n = \sum_{k=0}^{n-1} u_k \xi^k$ for some fixed $\xi \in \mathbb{R}$.

If $\left( u_n \right)_{n \in \mathbb{N}}$ satisfies a rec. with poly. coeffs, then $(\Sigma_n)$ too. *(why?)*
Application to sums of D-finite series (idea)

Let $\Sigma_n = \sum_{k=0}^{n-1} u_k \xi^k$ for some fixed $\xi \in \mathbb{R}$.

If $(u_n)_{n \in \mathbb{N}}$ satisfies a rec. with poly. coeffs, then $(\Sigma_n)$ too. (why?)

Better:

$$
\begin{pmatrix}
  u_{n+1} \xi^{n+1} \\
  \vdots \\
  u_{n+s} \xi^{n+1} \\
  \Sigma_{n+1}
\end{pmatrix}
= \frac{1}{b_s(n)}
\begin{pmatrix}
  (B(n) \xi) & \cdots & 0 \\
  b_s(n) & 0 & \cdots & 0 \\
  \vdots & \ddots & \ddots & \vdots \\
  b_s(n) & \cdots & b_s(n) & 0 \\
  \Sigma_n
\end{pmatrix}
\begin{pmatrix}
  u_n \xi^n \\
  \vdots \\
  u_{n+s-1} \xi^n \\
  \Sigma_n
\end{pmatrix}
$$
Application to sums of D-finite series (idea)

Let $\Sigma_n = \sum_{k=0}^{n-1} u_k \xi^k$ for some fixed $\xi \in \mathbb{R}$.

If $(u_n)_{n\in\mathbb{N}}$ satisfies a rec. with poly. coeffs, then $(\Sigma_n)$ too. (why?)

Better:

$$\begin{pmatrix} u_{n+1} \xi^{n+1} \\ \vdots \\ u_{n+s} \xi^{n+1} \\ \Sigma_{n+1} \end{pmatrix} = \frac{1}{b_s(n)} \begin{pmatrix} B(n) \xi \\ \vdots \\ b_s(n) \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ b_s(n) \end{pmatrix} \begin{pmatrix} u_n \xi^n \\ \vdots \\ u_{n+s-1} \xi^n \\ \Sigma_n \end{pmatrix}$$

Working with $p$-bit approximations and ignoring rounding errors:

$\Sigma_N$ to $p$-bit precision in $O(M(\sqrt{N}) \log(N) M_\mathbb{Z}(p))$ ops
Application to sums of D-finite series (idea)

Let $\Sigma_n = \sum_{k=0}^{n-1} u_k \xi^k$ for some fixed $\xi \in \mathbb{R}$.

If $(u_n)_{n \in \mathbb{N}}$ satisfies a rec. with poly. coeffs, then $(\Sigma_n)$ too. (why?)

Better:

$$
\begin{pmatrix}
  u_{n+1} \xi^{n+1} \\
  \vdots \\
  u_{n+s} \xi^{n+1} \\
  \Sigma_{n+1}
\end{pmatrix} = \frac{1}{b_s(n)} \begin{pmatrix}
  B(n) \xi \\
  \vdots \\
  b_s(n) 0 \cdots 0 b_s(n)
\end{pmatrix} \begin{pmatrix}
  u_n \xi^n \\
  \vdots \\
  u_{n+s-1} \xi^n \\
  \Sigma_n
\end{pmatrix}
$$

Working with $p$-bit approximations and ignoring rounding errors:

$$
\Sigma_N \text{ to } p\text{-bit precision in } O\left(M(\sqrt{N}) \log(N) M_{\mathbb{Z}}(p)\right) \text{ ops}
$$

Target accuracy $2^{-t}$ typically requires $N, p = O(t)$

$\leadsto$ evaluation of D-finite series to precision $t$ in $\tilde{O}(t^{3/2})$ ops