Computing power series at high precision*

by Romain Lebreton

ECO Team

Pole Algo-Calcul seminar
LIRMM

February 6th, 2015

*. This document has been written using the GNU $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ text editor (see www.texmacs.org).

Field of research

My field of research: *Computer Algebra*

- in between Computer Science and Mathematics

- sub-field of *Symbolic Computation*

- typical objects
  - numbers $$2, \quad \frac{355}{113}$$
  - polynomials $$x + x^2 + 2\,x^3, \quad x^5\,y^8 - x^8\,y^7 - x^7 y^6$$
  - modular computation $$5 \bmod 7, \quad x + x^2 + 2\,x^3 \bmod (x^2 - 1)$$
  - matrices $$\begin{pmatrix} 1 & 0 & 1 & 1 \\ x & 1 & 1+x & 0 \\ 1 & x^2+x^3 & x & 0 \\ x^2 & 0 & x^3+x^4 & 0 \end{pmatrix}$$
  - ...

Formal power series

Let $\mathbb{K}$ be an *effective* field, *i.e.* a set with algorithms for $+, -, *, /$ e.g. $\mathbb{Q}, \mathbb{Z}/p\mathbb{Z}$

Definition

A formal power series $f \in \mathbb{K}[[x]]$ is a sequence $(f_i)_{i\in\mathbb{N}}$ of $\mathbb{K}$, denoted by

$$f(x) = \sum_{i \geqslant 0} f_i x^i.$$

Remarks:

- Like a polynomial but with no finite degree constraint
- Addition, multiplication same as polynomials

  If $f = \sum f_n x^n = (\sum g_n x^n)(\sum h_n x^n)$ then $f_n = \sum_{i=0}^{n} g_i h_{n-i}$
- Purely formal: No notion of "analytic" convergence

Formal power series

Let $\mathbb{K}$ be an *effective* field, *i.e.* a set with algorithms for $+, -, *, /$ e.g. $\mathbb{Q}, \mathbb{Z}/p\,\mathbb{Z}$

Definition

A formal power series $f \in \mathbb{K}[[x]]$ *is a sequence* $(f_i)_{i \in \mathbb{N}}$ *of* $\mathbb{K}$*, denoted by*

$$f(x) = \sum_{i \geqslant 0} f_i\, x^i.$$

Computationally speaking:

- only truncated power series
- denote by $f(x) = g(x) + \mathcal{O}(x^N)$ the truncation at the term x^N

  we say "modulo x^N " or "at order N" or "at precision N"
- Compute a power series: compute its first N terms

Formal power series

Let $\mathbb{K}$ be an *effective* field, *i.e.* a set with algorithms for $+,-,*,/$ e.g. $\mathbb{Q},\mathbb{Z}/p\,\mathbb{Z}$

Definition

A formal power series $f\in\mathbb{K}[[x]]$ *is a sequence* $(f_i)_{i\in\mathbb{N}}$ *of* $\mathbb{K}$*, denoted by*

$$f(x)=\sum_{i\geqslant 0} f_i\,x^i.$$

Motivation:

- Approximation of functions

  Example: Taylor expansion of f at $x=0$

$$f(x)=f(0)+f'(0)\,x+\frac{f''(0)}{2}\,x^2+\cdots+\frac{f^{(i)}(0)}{i!}\,x^i+\mathcal{O}_{x\to 0}(x^{i+1})$$

Formal power series

Let $\mathbb{K}$ be an *effective* field, *i.e.* a set with algorithms for $+, -, *, /$ e.g. $\mathbb{Q}, \mathbb{Z}/p\,\mathbb{Z}$

Definition

A formal power series $f \in \mathbb{K}[[x]]$ is a sequence $(f_i)_{i\in\mathbb{N}}$ of $\mathbb{K}$, denoted by

$$f(x) = \sum_{i\geqslant 0} f_i\, x^i.$$

Motivation:

- Generating functions in Combinatorics

  Example: Catalan numbers $(C_n)_{n\in\mathbb{N}}$ (number of full binary trees)

$$\begin{aligned} G((C_n), x) &= \sum_{n\geqslant 0} C_n\, x^n = \frac{1-\sqrt{1-4\,x}}{2\,x} \\ &= 1 + x + 2\,x^2 + 5\,x^3 + 14\,x^4 + 42\,x^5 + 132\,x^6 + \mathcal{O}(x^7) \end{aligned}$$

Formal power series

Let $\mathbb{K}$ be an *effective* field, *i.e.* a set with algorithms for $+, -, *, /$ e.g. $\mathbb{Q}, \mathbb{Z}/p\,\mathbb{Z}$

Definition

A formal power series $f \in \mathbb{K}[[x]]$ is a sequence $(f_i)_{i\in\mathbb{N}}$ of $\mathbb{K}$, denoted by

$$f(x) = \sum_{i\geqslant 0} f_i\, x^i.$$

Motivation:

- In computer algebra

  Power series are ubiquitous when computing with polynomials

  Example:

  Euclidean division of $a, b \in \mathbb{K}[x]$, $a = b\,q + r$ with $\deg(r) < \deg(b)$

  The quotient q is computed using $a/b \in \mathbb{K}[[x]]$

Our objective

Our objective

Compute basic operations like $1\,/\,f$, $\sqrt{f}$, $\log(f)$, $\exp(f)$ $\in$ $\mathbb{K}[[x]]$ quickly in theory (quasi-linear time) and in practice (see below).

Theoretical complexity "reminder":

Power series multiplication at order n costs (arithmetic complexity)

$$\mathsf{M}(n) = \mathcal{O}(n \log n \log\log n) = \tilde{\mathcal{O}}(n)$$

Practical complexity:

In **one second** with today's computer, in $(\mathbb{Z}\,/\,1048583\,\mathbb{Z})[[x]]$ we can

- multiply two power series at order $\simeq 2 \cdot 10^6$
- compute $1\,/\,f, \log(f), \exp(f), \sqrt{f}$ at order $\simeq 5 \cdot 10^5$
- compute the Catalan generating function at order $\simeq 5 \cdot 10^5$

  (and at order $\simeq 4000$ over integers)

Outline of the talk

Two paradigms for power series computation:

1. Newton iteration

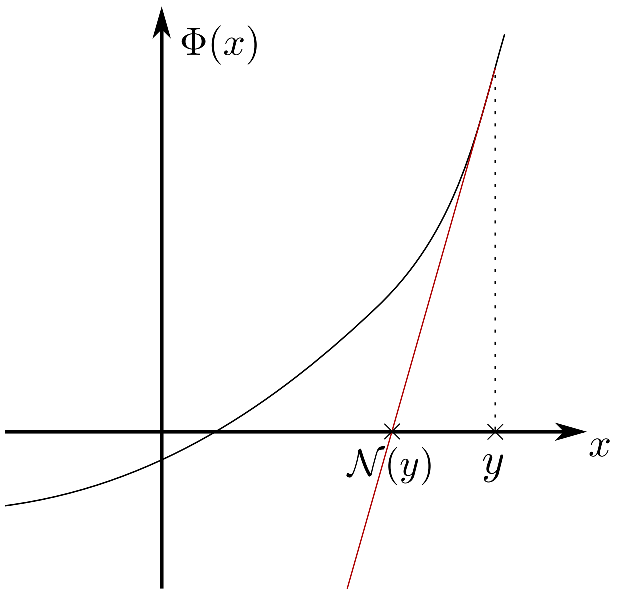
2. Relaxed algorithms

Newton operator - Numerical context

Historically comes from Isaac Newton, “La méthode des fluxions” in 1669

Goal of Newton iteration

Find approximate solutions of an equation $\Phi(x)=0$ where $\Phi\colon\mathbb{R}\to\mathbb{R}$.



Idea:

Approximate Φ around $y\in\mathbb{R}$ by a linear function

Choose $\mathcal{N}(y)$ to cancel the linear approx. of Φ

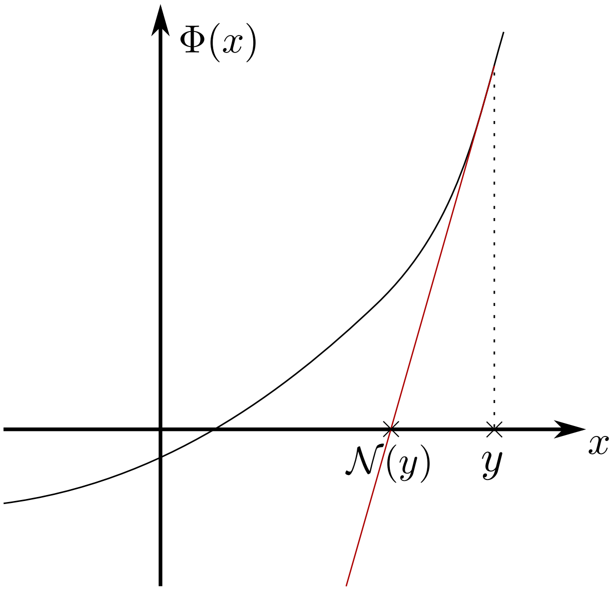
$$i.e. \quad \mathcal{N}(y)=y-\frac{\Phi(y)}{\Phi'(y)}$$

Intuition

If y is a “*good*” approximation of a solution of $\Phi(x)=0$ then

$\mathcal{N}(y)$ is an even better approximation.

Newton operator - Numerical context



Idea:

Approximate Φ around $x = y$ by a linear function

Choose $\mathcal{N}(y)$ to cancel the linear approx. of Φ

$$i.e. \quad \mathcal{N}(y) = y - \frac{\Phi(y)}{\Phi'(y)}$$

Newton iteration: Starting from $y_0 := y$, iterate $y_{k+1} := \mathcal{N}(y_k)$

Example: $\Phi(x) = x^2 - 2, \quad \mathcal{N}\colon y \mapsto y - \dfrac{y^2 - 2}{2\,y}$

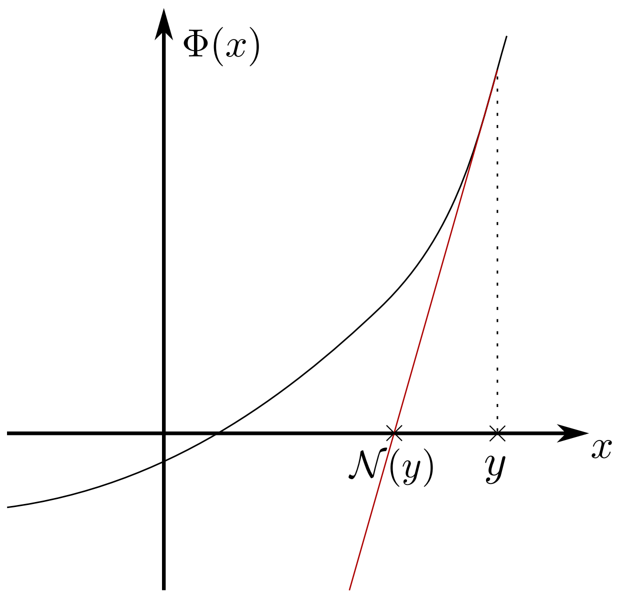
$y_0 = 1.50000000000000000000$

$y_1 = \mathcal{N}(y_0) = 1.41666666666666666663$

$y_2 = \mathcal{N}(y_1) = 1.41421568627450980392$

$y_3 = \mathcal{N}(y_2) = 1.41421356237468991059$

Newton operator - Numerical context



Idea:

Approximate Φ around $x = y$ by a linear function

Choose $\mathcal{N}(y)$ to cancel the linear approx. of Φ

$$i.e. \quad \mathcal{N}(y) = y - \frac{\Phi(y)}{\Phi'(y)}$$

Newton iteration: Starting from $y_0 := y$, iterate $y_{k+1} := \mathcal{N}(y_k)$

Theorem

If y_k converges then its number of correct decimal is approximately doubled.

Equivalently, if $y_k \xrightarrow[k\to\infty]{} r$ with r a regular solution of Φ (i.e. $\Phi'(r) \neq 0$) then

$$\frac{(y_{k+1} - r)}{(y_k - r)^2} \xrightarrow[k\to\infty]{} \Phi''(y)\,/\,(2\,\Phi'(y)) \qquad \textit{Quadratic convergence}$$

Newton operator - Numerical context

Side note: Knowing which starting values y_0 make $(y_k)_{k \in N}$ converge is a hard problem

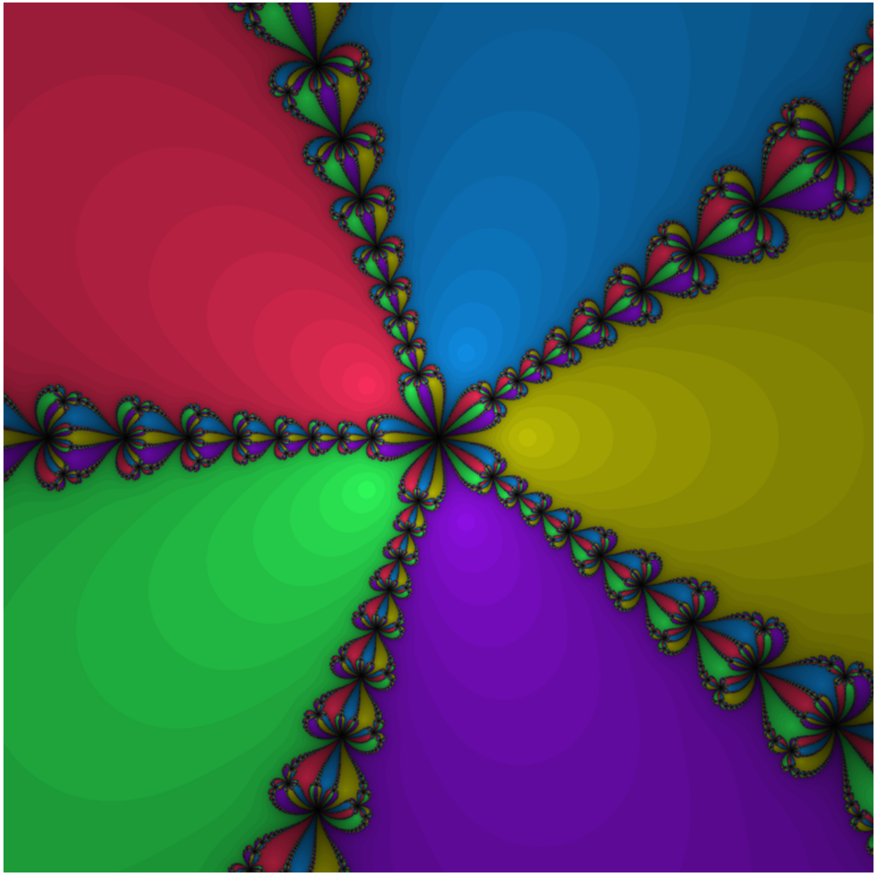


Figure. Basins of attraction for $x^5 - 1 = 0$ over $\mathbb{C}$ (source Wikipedia)

Symbolic Newton operator

This time, let $\Phi\colon \mathbb{K}[[x]] \to \mathbb{K}[[x]]$ be a polynomial function, *i.e.* $\Phi \in \mathbb{K}[[x]][y]$

Similarly to the numerical case, we define for $y \in \mathbb{F}[[x]]$ the newton operator

$$\mathcal{N}(y) = y - \frac{\Phi(y)}{\Phi'(y)}$$

However, the behavior is simpler in the symbolic world

Theorem - Symbolic Newton iteration

1. If y_0 satisfies $\Phi(y_0) = 0 + \mathcal{O}(x)$ then

   the sequence (y_k) will converge to a solution $s \in \mathbb{K}[[x]]$ of $\Phi(x) = 0$

2. Quadratic convergence is guaranteed:

$$s = y_k + \mathcal{O}\left(x^{2^k}\right)$$

Remark: Only works for regular root, *i.e.* $\Phi'(y_0) \neq 0 \bmod x$. Otherwise, $\Phi'(y)$ is not invertible.

Symbolic Newton operator

Reminder: $\Phi \in \mathbb{K}[[x]][y]$ and $\mathcal{N}(y) = y - \frac{\Phi(y)}{\Phi'(y)}$

Theorem - Symbolic Newton iteration

1. If y_0 satisfies $\Phi(y_0) = 0 + \mathcal{O}(x)$ then

   the sequence (y_k) will converge to a solution $s \in \mathbb{K}[[x]]$ of $\Phi(x) = 0$

2. Quadratic convergence is guaranteed:

$$s = y_k + \mathcal{O}\left(x^{2^k}\right)$$

Sketch of the proof:

Suppose we have an approximate solutions y, *i.e.* $\Phi(y) = 0 + \mathcal{O}(x^N) = x^N p(x)$.

Let us find a small perturbation $x^N \varepsilon$ of y that improves the solution.

Taylor expansion of Φ near y:

$$\begin{aligned} \Phi(y + x^N \varepsilon) &= \Phi(y) + \Phi'(y)\, x^N \varepsilon + \mathcal{O}((x^N \varepsilon)^2) \\ &= x^N \underbrace{(p(x) + \Phi'(y)\, \varepsilon)}_{\text{choose } \varepsilon \text{ to cancel this}} + \mathcal{O}(x^{2N}) \end{aligned}$$

□

Symbolic Newton operator – Examples

Example:

Compute the inverse of a power series $f \in \mathbb{K}[[x]]$

The series $1/f$ is a solution of $0 = \Phi(y) := 1/y - f$

Therefore we derive the Newton operator

$$\mathcal{N}(y) = y + (1 - y\,f)\,y$$

Newton iteration: Take $f = 1 + x + x^2$

$y_0 := 1$ (Starting point. $1 = \frac{1}{f} \bmod x$)

$y_1 = \mathcal{N}(y_0) = 1 - x - x^2 + O(x^{10})$

$y_2 = \mathcal{N}(y_1) = 1 - x + x^3 - 2\,x^4 - 3\,x^5 - x^6 + O(x^{10})$

$y_3 = \mathcal{N}(y_2) = 1 - x + x^3 - x^4 + x^6 - x^7 - x^8 - 6\,x^9 + O(x^{10})$

$y_4 = \mathcal{N}(y_3) = 1 - x + x^3 - x^4 + x^6 - x^7 + x^9 + O(x^{10})$

Symbolic Newton operator – Examples

Example:

Compute the Catalan generating function $G(x) = \sum C_n\, x^n = \frac{1-\sqrt{1-4\,x}}{2\,x}$

$G(x)$ is a solution of $0 = \Phi(y) := (2\,x\,y-1)^2 - (1-4\,x)$

The Newton operator becomes

$$\mathcal{N}(y) = y - \frac{(2\,x\,y-1)^2-(1-4\,x)}{4\,x(2\,x\,y-1)} = y + \frac{(1-y+x\,y^2)}{(1-2\,x\,y)}$$

Newton iteration:

$y_0 := C_0 = 1$

$y_1 = 1 + x + O(x^{10})$

$y_2 = 1 + x + 2\,x^2 + 5\,x^3 + 6\,x^4 + 2\,x^5 + O(x^{10})$

$y_3 = 1 + x + 2\,x^2 + 5\,x^3 + 14\,x^4 + 42\,x^5 + 132\,x^6 + 429\,x^7 + 1302\,x^8 + 3390\,x^9 + O(x^{10})$

$y_4 = 1 + x + 2\,x^2 + 5\,x^3 + 14\,x^4 + 42\,x^5 + 132\,x^6 + 429\,x^7 + 1430\,x^8 + 4862\,x^9 + O(x^{10})$

Remark: the inverse of $(1-2\,x\,y_k)$ is computed using previous Newton iteration

Cost of symbolic Newton operator

Complexity to compute an approximate solution of $\Phi(y) = 0$ at order $\mathcal{O}(x^N)$:

- The cost of the last iteration is dominant.
- The last iteration involves some multiplication and additions at order $\mathcal{O}(x^N)$ to evaluate $\Phi(y_k)$, $\Phi'(y_k)$ (and invert $\Phi'(y_k)$)

$\Rightarrow$ The cost is $\mathcal{O}(\mathsf{M}(n))$

Remark: The complexity of evaluation of Φ is a constant hidden in the $\mathcal{O}$

Intermediate checkpoint

So far,

- Newton iteration computes power series f solutions of implicit equations $\Phi(y) = 0$
- It costs asymptotically a constant number of multiplication.

Upcoming, Relaxed algorithms

- the second important paradigm to compute common power series
- It computes power series f solutions of "recursive" equations $\Phi(y) = y$

These two techniques are complementary
They yield the current best complexity to compute power series at high precision

Improved data structure for power series

The lazy representation – An improved data structure for $f \in \mathbb{K}[[x]]$

1. Storage of the current approximation modulo x^N of f
2. Attach a function `increasePrecision()` to f

Examples:

- Based on Newton iteration:

  Store the current approximation Y_k
  `increasePrecision()` perform $Y_{k+1} = \mathcal{N}(Y_k)$ $\rightsquigarrow$ doubles precision

- Naive multiplication of $f = g\,h \in \mathbb{K}[[x]]$:

  `increasePrecision()` computes one more term of $f = \sum f_n x^n$ using

$$f_n = \sum_{i=0}^{n} g_i\, h_{n-i}$$

Pros: Management of precision is more user-friendly

Controlling the reading of inputs

In a context of lazy representation, the following question is important:

Which coefficients of the input are required to compute the output at order $\mathcal{O}(x^N)$?

Why is it important ?

1. First of all, these coefficients of the inputs may require computation $\rightsquigarrow$ can be costly
2. Controlling the access of the inputs will be the cornerstone of the new technique to compute *recursive* power series

Controlling the reading of inputs

In a context of lazy representation, the following question is important:

Which coefficients of the input are required to compute the output at order $\mathcal{O}(x^N)$?

Very different dependency on the inputs

- *Newton iteration* for e.g. power series inversion

  Computing the coefficients of $1/f$ in $x^{2^k}, \ldots, x^{2^{k+1}-1}$ requires reading the same coefficients of f

  Indeed $y_{k+1} = \mathcal{N}(y_k) = [y_k + (1 - y_k f)\, y_k] \bmod x^{2^{k+1}}$

  More precisely: Read f_{2^k},..., read $f_{2^{k+1}-1}$ then output $(1/f)_{2^k}$,..., $(1/f)_{2^{k+1}-1}$

- *Fast multiplication* $f = g\, h \bmod x^n$ (FFT)

  Read all coefficients $g_0, \ldots, g_{n-1}$, $h_0, \ldots, h_{n-1}$ of inputs then output $f_0, \ldots, f_n$

- *Naive multiplication*

  Read g_0, h_0, output f_0 | Read g_0, g_1, h_0, h_1, output f_1 | Read $g_0, g_1, g_2, h_0, h_1, h_2$, output f_2

Relaxed algorithms

We are interested in algorithms that control the reading of their inputs

Definition

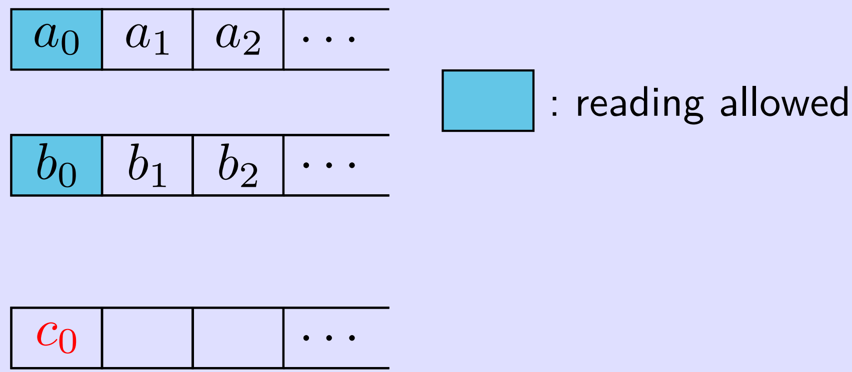
(*on-line* or *relaxed* algorithm) [Hennie '66]

$$a = \sum_{i \geqslant 0} a_i x^i$$

$$b = \sum_{i \geqslant 0} b_i x^i$$

$$\Big\downarrow f$$

$$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$$



Relaxed algorithms

We are interested in algorithms that control the reading of their inputs

Definition

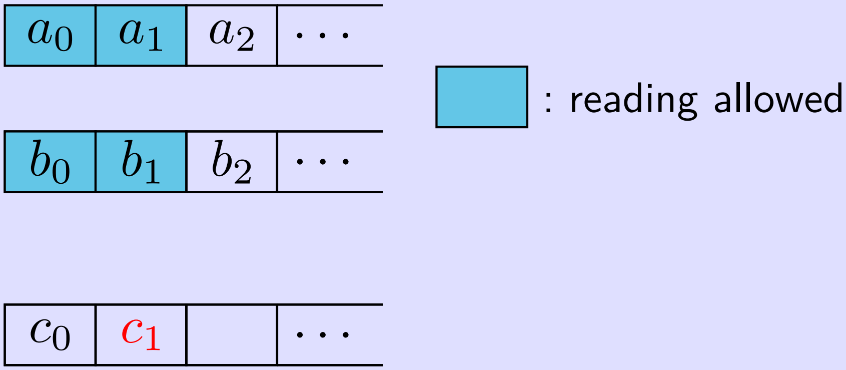
(*on-line* or *relaxed* algorithm) [HENNIE '66]

$$a = \sum_{i \geqslant 0} a_i x^i$$

$$b = \sum_{i \geqslant 0} b_i x^i$$

$$\Big\downarrow f$$

$$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$$



Relaxed algorithms

We are interested in algorithms that control the reading of their inputs

Definition

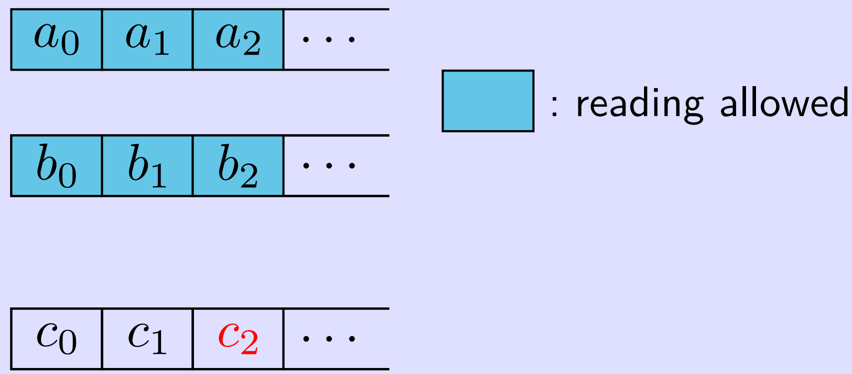
(*on-line* or *relaxed* algorithm) [Hennie '66]

$$a = \sum_{i \geqslant 0} a_i x^i$$

$$b = \sum_{i \geqslant 0} b_i x^i$$

$$\downarrow f$$

$$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$$



Relaxed algorithms

We are interested in algorithms that control the reading of their inputs

Definition

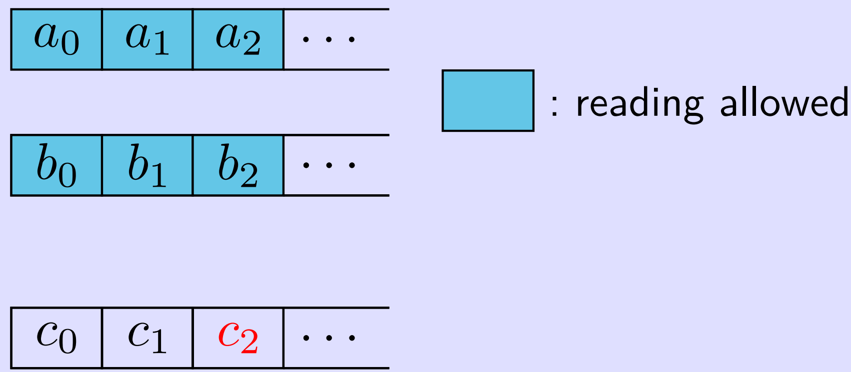
(*on-line* or *relaxed* algorithm) [Hennie '66]

$$a = \sum_{i \geqslant 0} a_i x^i$$

$$b = \sum_{i \geqslant 0} b_i x^i$$

$$\downarrow f$$

$$c = f(a, b) = \sum_{i \geqslant 0} c_i x^i$$



Off-line *or* zealous *algorithm:* *condition not met.*

Trivial examples of relaxed algorithms

Naive Addition: Compute $f = g + h$ using $f_n = g_n + h_n$

The addition algorithm is *online*:

$\rightarrow$ it outputs f_i reading only g_i and h_i.

Naive Multiplication: Compute $f = g\,h$ using $f_n = \sum_{i=0}^{n} g_i\,h_{n-i}$

1. This multiplication algorithm is *online*:

   $\rightarrow$ it outputs f_i reading $f_0, \ldots, f_i$ and $g_0, \ldots, g_i$.

2. Its complexity is quadratic !

Fast relaxed multiplications

Problem.

Fast multiplication algorithms (Karatsuba, FFT) are offline.

Challenge.

Find a *quasi-optimal on-line* multiplication algorithm.

Theorem. [Fischer, Stockmeyer '74], [Schröder '97], [van der Hoeven '97] [Berthomieu, van der Hoeven, Lecerf '11], [L., Schost '13]

From an off-line multiplication algorithm which costs $\mathsf{M}(N)$ at precision N,

we can derive an on-line multiplication algorithm of cost

$$\mathsf{R}(N) = \mathcal{O}(\mathsf{M}(N) \log N) = \tilde{\mathcal{O}}(N).$$

Fast relaxed multiplications

Problem.

Fast multiplication algorithms (Karatsuba, FFT) are offline.

Challenge.

Find a *quasi-optimal on-line* multiplication algorithm.

Theorem. [Fischer, Stockmeyer '74], [Schröder '97], [van der Hoeven '97] [Berthomieu, van der Hoeven, Lecerf '11], [L., Schost '13]

From an off-line multiplication algorithm which costs $\mathsf{M}(N)$ at precision N,

we can derive an on-line multiplication algorithm of cost

$$\mathsf{R}(N) = \mathcal{O}(\mathsf{M}(N) \log N) = \tilde{\mathcal{O}}(N).$$

Theorem. [van der Hoeven '07, '12]

$$\mathsf{R}(N) = \mathsf{M}(N) \log(N)^{o(1)}$$

Seems not to be used yet in practice.

Recursive power series

Definition

A power series $y \in \mathbb{Q}[[T]]$ is recursive if there exists Φ such that

- $y = \Phi(y)$
- $\Phi(y)_n$ *only depends on* $y_0, \ldots, y_{n-1}$

Example. Compute $g = \exp(f)$ defined as $\exp(f) := \sum_{i \geqslant 0} \frac{f^i}{i!}$ when $f(0) = 0$

Remark that $g' = f' g$.

So g is recursive with $y_0 = 1$ and $y = \Phi(y) = \int f' y$.

Moreover

$$\begin{aligned} \Phi(y)_n &= \left(\int f' y\right)_n \\ &= 1/n \cdot (f' y)_{n-1} \\ &= 1/n \cdot (f'_0 y_{n-1} + \cdots + f'_{n-1} y_0) \end{aligned}$$

So $\Phi(y)_n$ only depends on $y_0, \ldots, y_{n-1}$.

Recursive power series

Definition

A power series $y \in \mathbb{Q}[[T]]$ *is recursive if there exists* Φ *such that*

- $y = \Phi(y)$
- $\Phi(y)_n$ *only depends on* $y_0, \ldots, y_{n-1}$

$\rightsquigarrow$ It is possible to compute y from Φ and y_0. But how fast?

Relaxed algorithms and recursive power series

Relaxed algorithms allow the computation of recursive power series

On an example.

Compute $g = \exp(f)$ for $f = x + \frac{1}{2}x^2 + \mathcal{O}(x^2)$

We know that g is recursive with $y_0 = 1$ and $y = \Phi(y) = \int f' y$.

$\rightsquigarrow$ Use the relation $\Phi(y)_n = 1/n \cdot (f' y)_{n-1}$ + online multiplication for $f' \cdot y$

Computations:

$y = 1 + \mathcal{O}(x)$

$\Phi(y)_1 = 1/1 \cdot (f' y)_0 = 1$ (read only y_0 using an online multiplication)

$$\begin{array}{lcl}
y = \sum_{i \geqslant 0} y_i x^i & & \begin{array}{|c|c|c|c} \hline y_0 & ? & ? & \cdots \\ \hline \end{array} \\
\quad \downarrow \Phi & & \\
\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i & & \begin{array}{|c|c|c|c} \hline \varphi_0 & \varphi_1 & & \cdots \\ \hline \end{array}
\end{array}$$

$\boxed{}$: reading allowed

Relaxed algorithms and recursive power series

Relaxed algorithms allow the computation of recursive power series

On an example.

Compute $g = \exp(f)$ for $f = x + \frac{1}{2}x^2 + \mathcal{O}(x^2)$

We know that g is recursive with $y_0 = 1$ and $y = \Phi(y) = \int f' y$.

$\rightsquigarrow$ Use the relation $\Phi(y)_n = 1/n \cdot (f' y)_{n-1}$ + online multiplication for $f' \cdot y$

Computations:

$y = 1 + x + \mathcal{O}(x^2)$

$\Phi(y)_1 = 1/1 \cdot (f' y)_0 = 1$ (read only y_0 using an online multiplication)

$$y = \sum_{i \geqslant 0} y_i x^i \qquad \begin{array}{|c|c|c|c}\hline y_0 & y_1 & ? & \cdots \\ \hline\end{array}$$

$$\Big\downarrow \Phi \qquad \qquad \uparrow$$

$$\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \qquad \begin{array}{|c|c|c|c}\hline \varphi_0 & \varphi_1 & & \cdots \\ \hline\end{array}$$

: reading allowed

Relaxed algorithms and recursive power series

Relaxed algorithms allow the computation of recursive power series

On an example.

Compute $g = \exp(f)$ for $f = x + \frac{1}{2}x^2 + \mathcal{O}(x^2)$

We know that g is recursive with $y_0 = 1$ and $y = \Phi(y) = \int f' y$.

$\rightsquigarrow$ Use the relation $\Phi(y)_n = 1/n \cdot (f' y)_{n-1}$ + online multiplication for $f' \cdot y$

Computations:

$y = 1 + x + \mathcal{O}(x^2)$

$\Phi(y)_2 = 1/2 \cdot (f' y)_1 = 1$ (read only y_0, y_1 using an online multiplication)

$$y = \sum_{i \geqslant 0} y_i x^i \qquad \boxed{y_0} \boxed{y_1} \boxed{?} \boxed{\cdots}$$

$\boxed{}$: reading allowed

$$\downarrow \Phi$$

$$\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \qquad \boxed{\varphi_0} \boxed{\varphi_1} \boxed{\varphi_2} \boxed{\cdots}$$

Relaxed algorithms and recursive power series

Relaxed algorithms allow the computation of recursive power series

On an example.

Compute $g = \exp(f)$ for $f = x + \frac{1}{2}x^2 + \mathcal{O}(x^2)$

We know that g is recursive with $y_0 = 1$ and $y = \Phi(y) = \int f' y$.

$\rightsquigarrow$ Use the relation $\Phi(y)_n = 1/n \cdot (f' y)_{n-1}$ + online multiplication for $f' \cdot y$

Computations:

$y = 1 + x + x^2 + \mathcal{O}(x^3)$

$\Phi(y)_2 = 1/2 \cdot (f' y)_1 = 1$ (read only y_0, y_1 using an online multiplication)

$$y = \sum_{i \geqslant 0} y_i x^i$$

y_0	y_1	y_2	$\cdots$

: reading allowed

$$\Big\downarrow \Phi$$

$$\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i$$

φ_0	φ_1	φ_2	$\cdots$

What about the general context?

Recursive equations are evaluated using shifted algorithms

Definition of *shifted algorithms*.

$$a = \sum_{i \geqslant 0} a_i x^i \qquad \begin{array}{|c|c|c|c}\hline a_0 & a_1 & a_2 & \cdots \\ \hline\end{array}$$

: reading allowed

$$\Big\downarrow f$$

$$c = f(a) = \sum_{i \geqslant 0} c_i x^i \qquad \begin{array}{|c|c|c|c}\hline c_0 & \quad & \quad & \cdots \\ \hline\end{array}$$

Remark. Shifted algorithms are built using online algorithms

Shifted Algorithms

What about the general context?

Recursive equations are evaluated using shifted algorithms

Definition of *shifted algorithms*.

$$\begin{array}{c} a = \sum_{i\geqslant 0} a_i x^i \\ \Big\downarrow f \\ c = f(a) = \sum_{i\geqslant 0} c_i x^i \end{array} \qquad \begin{array}{|c|c|c|c} \hline a_0 & a_1 & a_2 & \cdots \\ \hline \\ \\ \hline c_0 & c_1 & & \cdots \\ \hline \end{array}$$

$\boxed{}$: reading allowed

Remark. Shifted algorithms are built using online algorithms

Shifted Algorithms

What about the general context?

Recursive equations are evaluated using shifted algorithms

Definition of *shifted algorithms*.

$$a = \sum_{i \geqslant 0} a_i x^i \qquad \boxed{a_0}\boxed{a_1}\boxed{a_2}\boxed{\cdots}$$

: reading allowed

$$\Big\downarrow f$$

$$c = f(a) = \sum_{i \geqslant 0} c_i x^i \qquad \boxed{c_0}\boxed{c_1}\boxed{c_2}\boxed{\cdots}$$

Remark. Shifted algorithms are built using online algorithms

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{K}[[T]]$ be a recursive power series with $y = \Phi(y)$.

Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

This is usually $\mathcal{O}(\mathsf{R}(N))$.

Proof

$y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$y = \sum_{i \geqslant 0} y_i x^i$ $\quad$ $\boxed{y_0}\ \boxed{?}\ \boxed{?}\ \cdots$ $\quad$ $\boxed{}$: reading allowed

$$\downarrow \Phi$$

$\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i$ $\quad$ $\boxed{\varphi_0}\ \boxed{\varphi_1}\ \boxed{}\ \cdots$

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{K}[[T]]$ be a recursive power series with $y = \Phi(y)$.

Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

This is usually $\mathcal{O}(\mathsf{R}(N))$.

Proof

$y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$$\begin{array}{c} y = \sum_{i \geqslant 0} y_i x^i \\ \Big\downarrow \Phi \\ \Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \end{array}$$

y_0	y_1	?	$\cdots$
$\uparrow$			
φ_0	φ_1		$\cdots$

: reading allowed

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{K}[[T]]$ be a recursive power series with $y = \Phi(y)$.

Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

This is usually $\mathcal{O}(\mathsf{R}(N))$.

Proof

$y = \Phi(y) \Rightarrow \varphi_0 = y_0$

$$y = \sum_{i \geqslant 0} y_i x^i$$

y_0	y_1	?	$\cdots$

: reading allowed

$$\big\downarrow \Phi$$

$$\Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i$$

φ_0	φ_1	φ_2	$\cdots$

Relaxed recursive power series

Fundamental Theorem. [Watt '88], [van der Hoeven '02], [Berthomieu, L. ISSAC '12]

Let $y \in \mathbb{K}[[T]]$ be a recursive power series with $y = \Phi(y)$.

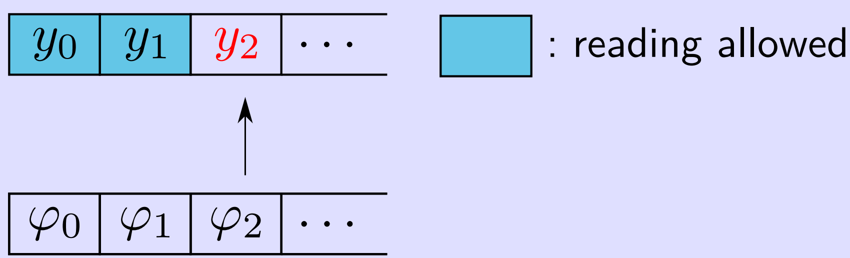
Given y_0 and Φ, we can compute y at precision N in the time necessary to evaluate $\Phi(y)$ by a shifted algorithm.

This is usually $\mathcal{O}(\mathrm{R}(N))$.

Proof

$y = \Phi(y) \Rightarrow \varphi_0 = y_0$ $\square$

$$\begin{array}{c} y = \sum_{i \geqslant 0} y_i x^i \\ \big\downarrow \Phi \\ \Phi(y) = \sum_{i \geqslant 0} \varphi_i x^i \end{array}$$



Conclusion

Two general paradigms:

Newton operator	Relaxed algorithms
Solve implicit equations $P(y) = 0$	Solve recursive equations $y = \Phi(y)$
Faster for higher precision	Less on-line multiplications

Implementations:

Relaxed power series (and p-adics) in MATHEMAGIX

Beginning of a `C++` package based on NTL

Also partially present in LINBOX